# A Segunda Geração de Computadores de Alto Desempenho da COPPE/UFRJ *

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

COPPE Sistemas
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, 21945-970

{amorim,ricardo}@cos.ufrj.br

**Resumo**

Sistemas de memória compartilhada distribuída (DSMs) podem ser implementados completamente em *software*. Esses sistemas, conhecidos como *software* DSMs, exibem um baixo custo e, portanto, representam uma alternativa atraente para computação paralela. Entretanto, *software* DSMs puros atingem alto desempenho numa classe restrita de aplicações. O sistema paralelo $NCP_2$ introduz a segunda geração de computadores de alto desempenho da COPPE/UFRJ, a qual investiga a utilização de *hardware* simples e de baixo custo para otimizar *software DSMs*. Esse *hardware* consiste de controladores de protocolo programáveis que permitem a implementação de técnicas de tolerância à latência de comunicação e a *overheads* de processamento de coerência. Nossas simulações revelam que os nossos controladores de protocolo melhoram o desempenho de TreadMarks em 16 processadores em até 52%. Esses resultados sugerem que o $NCP_2$ exibirá uma ótima relação custo/desempenho numa grande gama de aplicações, proporcionando uma opção tecnológica muito boa para investimentos nacionais em computação de alto desempenho.

**Abstract**

Software-only distributed-shared memory systems (DSMs) can combine the ease of shared-memory programming with the low cost of distributed-memory architectures. However, they provide acceptable performance for only a limited class of applications, mainly due to high communication and coherence overheads. The $NCP_2$ parallel system introduces the second generation of high-performance computers from COPPE/UFRJ and is investigating the use of simple and low-cost hardware in support of software DSMs. This hardware consists of programmable protocol controllers that allow for tolerating the communication and coherence overheads in these systems. Detailed simulations show that our protocol controllers can improve the performance of TreadMarks on 16 processors by as much as 52%. These results suggest that the $NCP_2$ will exhibit an excelent cost/performance ratio, providing a technological opportunity for national investments in high performance computing.

*Esta pesquisa foi financiada pela FINEP/MCT, Projeto Computação Paralela, no. 5694039 e pelo CNPq.

# 1 Introdução

Nos últimos anos tem crescido significativamente o interesse por sistemas de memória compartilhada distribuída (DSM). Esses sistemas permitem integrar a escalabilidade de memórias distribuídas com a maior facilidade de programação de multiprocessadores, tais como DASH [12] e Alewife [1]. Em constraste com esses multiprocessadores, os sistemas denominados *software* DSMs (SW-DSMs) mantêm a coerência de dados em *software* e oferecem aos programadores a ilusão de uma memória compartilhada sobre um *hardware* que só permite troca de mensagens.

SW-DSMs provêem uma alternativa de baixo custo para a computação no modelo de memória compartilhada, visto que o sistema pode ser formado por estações de trabalho e sistemas operacionais padrão. No entanto, são poucas as classes de aplicações que alcançam um bom desempenho nestes sistemas. Isto se deve a uma alta taxa de comunicação e ao *overhead* gerado pela manutenção da coerência dos dados.

O NCP I [3], o primeiro sistema de computação paralela desenvolvido pela COPPE/UFRJ e operacional em 1990, pertencia à classe dos multicomputadores. Além de representar um marco de referência no esforço de desenvolvimento científico e tecnológico nacional, a experiência do NCP I permitiu romper barreiras tecnológicas na área de computação de alto desempenho e proporcionou lições que habilitaram o atual projeto $NCP_2$ [2, 4].

Tal como o NCP I, o $NCP_2$ é também um multicomputador, mas que adiciona suporte especializado de *hardware*, simples e de baixo custo, para suportar SW-DSMs eficientemente. Dessa forma, o sistema pode atingir alto desempenho numa variedade maior de aplicações do que um SW-DSM puro. O protótipo atualmente em desenvolvimento incorpora as tecnologias de microprocessadores PowerPC, o sistema operacional Unix, e uma rede de interconexão Myrinet. SW-DSMs como TreadMarks [9] estão sendo modificados para execução eficiente no $NCP_2$. Estima-se que o $NCP_2$ terá desempenho uma ordem de grandeza superior ao do NCP I.

A organização deste artigo é a seguinte. A seção 2 dá uma visão geral das principais características de SW-DSMs e seus principais *overheads*. Na seção 3, descrevemos o *hardware* e o *software* do $NCP_2$, ressaltando a solução inovadora que ele introduz por utilizar nossos controladores de protocolo programáveis. Na seção 4, comparamos o desempenho de TreadMarks modificado para um sistema semelhante ao $NCP_2$ com o de TreadMarks puro, através da simulações de aplicações reais. Nossos resultados preliminares mostram que os controladores de protocolo que propomos podem melhorar a performance de TreadMarks em 16 processadores em até 52%. Na seção 5, os trabalhos relacionados são identificados. Finalmente, na seção 6, apresentamos nossas conclusões.

# 2 *Overheads* em Software DSMs

A maior parte dos SW-DSMs realiza a manuntenção da coerência a nível de página através dos *bits* de proteção da memória virtual. Além disso, tendo como objetivo solucionar o problema de falso compartilhamento de páginas, *DSMs* modernos permitem a escrita simultânea na mesma página por vários processadores, garantindo a consistência de memória somente nos pontos de sincronização. Este adiamento da

manutenção da consistência é denominado consistência relaxada. TreadMarks é um sistema que mantém a consistência usando *Lazy Release Consistency* [8].

Em TreadMarks, as invalidações das páginas alteradas por outros processadores ocorrem nos pontos de aquisição de *locks*. Entretanto, as modificações (ou *diffs*) das páginas invalidadas só são requeridas na ocorrência de um *page fault*. Neste momento são enviados pedidos de modificações aos processadores que escreveram por último na página. As alterações que precisam ser coletadas pelo processador que adquiriu o *lock* são determinadas através da divisão da execução em intervalos, onde cada intervalo é associado a um vetor de *timestamps*. Este vetor descreve uma ordem parcial entre intervalos de processadores distintos. O processador que adquire um *lock* só pode continuar sua execução depois de coletar todas as atualizações referentes a intervalos cujo vetor de *timestamps* é menor que o seu. O último dono do *lock* é o responsável pela comparação do seu vetor de *timestamps* com o do novo *acquirer*, e pelo envio de *write notices* que indicam a alteração de uma página por um processador em um determinado intervalo. Quando ocorre um *page fault*, o processador percorre sua lista de *write notices* para determinar quais os diffs necessários a atualização da página. Uma descrição mais detalhada de TreadMarks pode ser encontrada em [8].

Os principais *overheads* em SW-DSMs estão relacionados à latência de comunicação e à manutenção de coerência. Latências de comunicação retardam a execução e portanto reduzem o desempenho do sistema. Ações de coerência (geração e aplicação de *diffs*, geração de *twins*, e manipulação de diretórios) também podem afetar o desempenho negativamente, já que elas não realizam trabalho útil e geralmente estão no caminho crítico das aplicações. O impacto dos *overheads* de comunicação e coerência é ampliado pelo fato de que processadores remotos são envolvidos em todas essas transações.

# 3 $NCP_2$: A Segunda Geração

O sistema $NCP_2$ segue os princípios gerais de projeto do NCP I. Ambos os projetos têm como objetivos principais a simplicidade do *hardware* e a maximização das suas contribuições científicas e tecnológicas. No entanto, a maior complexidade das tecnologias envolvidas em projetos arquiteturais recentes como o $NCP_2$ demandam uma maior dependência de sistemas de desenvolvimento CAD. Em termos de desempenho, estima-se que o $NCP_2$ será uma ordem de grandeza mais veloz do que o NCP I. Esperamos ainda que o modelo de programação mais simples do $NCP_2$ simplifique a implementação de aplicações paralelas complexas em comparação ao NCP I.

Nas próximas seções descrevemos as tecnologias envolvidas no projeto $NCP_2$, e as principais caracteísticas do *hardware* e do *software* do sistema.

## 3.1 Tecnologias Envolvidas no $NCP_2$

O projeto $NCP_2$ contempla o desenvolvimento, em três anos, de *hardware* de dois protótipos com 32 processadores. O primeiro protótipo do sistema utiliza o microprocessador PowerPC 604 (100 MHz, 160 SPECint e o 165 SPECfp) e o segundo utilizará o PowerPC 620. O $NCP_2$ incorpora ainda o sistema operacional Unix

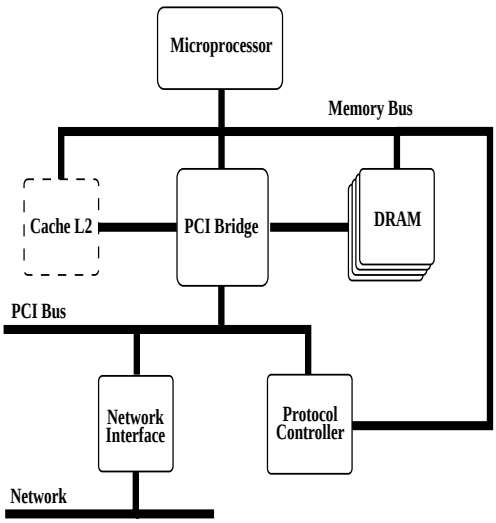


Figura 1: Arquitetura do Nó de Processamento.

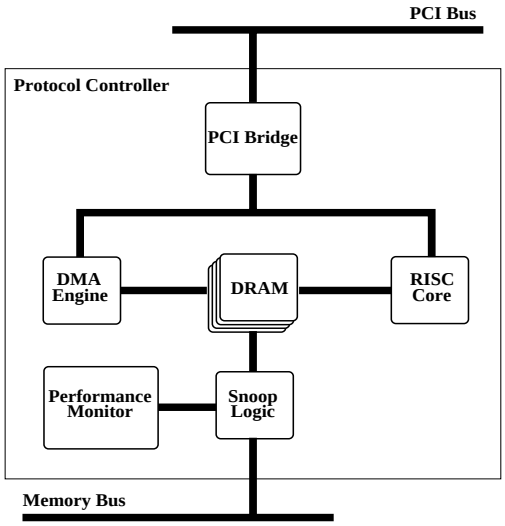


Figura 2: Zoom da Arquitetura do Controlador.

(AIX), enquanto que a rede de interconexão é a Myrinet (80 Mbytes/s de banda passante e 0.5-10us de latência). Além disso, o sistema usa o barramento PCI e a interface SCSI em cada unidade de processamento. O projeto licenciou também o SW-DSM TreadMarks que será adaptado ao $NCP_2$.

## 3.2 O *Hardware* do $NCP_2$

Como mostra a figura 1, o suporte de *hardware* que propomos é um controlador de protocolos associado a cada nó de uma rede de *workstations* ou de um multicomputador. No nosso sistema tanto o controlador de protocolos como a interface de rede estão conectadas ao barramento PCI. Como mostra a figura 2, nosso controlador de protocolos inclui um microprocessador (ou simplesmente um núcleo RISC inteiro), 6 Mbytes de DRAM e dois módulos de *hardware* específicos: a lógica para *snooping* do barramento de memória e um dispositivo DMA sofisticado. A comunicação entre o processador e o controlador, na maior parte dos casos, é realizada através da memória local do controlador e o *snooping* dos acessos de escrita. O processador principal dispara pedidos explícitos ao controlador local e continua a sua execução normalmente, a menos que o atendimento do pedido seja necessário imediatamente. Os pedidos locais e quaisquer pedidos provenientes de controladores remotos são enfileirados na memória local, enquanto aguardam o atendimento. Transações que necessitam de intervenção de nós remotos são divididas em três partes (pedido, acesso e resposta) que podem ser intercaladas com outras transacões no controlador de protocolo. Pedidos podem receber prioridades, de tal modo que pedidos com alta prioridade podem passar a frente de pedidos de baixa prioridade. Este esquema de prioridades é usado para evitar que operações de *prefetch* atrasem pedidos dos quais o processador depende para prosseguir a execução.

O controlador de protocolos pode se comunicar com o processador principal por interrupção quando for necessário. Interrupções ao processador principal devem ser evitadas, para não prejudicar o desempenho, mas podem ser utilizadas para evitar que operações complexas tenham que atravessar o barramento PCI para alcançar a

memória principal. De fato, a relação entre a velocidade do processador principal e a do controlador é uma questão importante; um processador principal muito rápido fornece outra justificativa para que nele sejam executadas as partes complexas do código de protocolo.

Tanto o processador como o controlador de protocolos realizam *snooping* no barramento de memória. O processador realiza *snooping* para invalidar dados escritos pelo controlador de protocolos diretamente na memória principal, no caso da aplicação de um *diff* remoto em uma página local, por exemplo. O controlador de protocolos realiza *snooping* para computar *diffs* dinamicamente. Isto é conseguido forçando a cache a escrever dados compartilhados no barramento e mantendo um registro (na memória do controlador) de todas as palavras que foram modificadas em uma página. Este registro é mantido sob a forma de um vetor de *bits*, onde cada *bit* representa uma palavra de dados. Sempre que o controlador de protocolos detecta que o processador principal escreveu uma palavra, simplesmente ativa o *bit* correspondente para registrar o evento. Posteriormente, quando é requisitado ao processador o *diff* de uma página, nosso dispositivo DMA checa quais *bits* estão ativados, lê as palavras correspondentes da memória e retorna, como resultado do *diff*, as palavras modificadas e o vetor de *bits*. O *diff* pode ser mantido na memória para uso posterior. A geração de um *diff* provoca a desativação de todos os *bits* do vetor. A aplicação do *diff* também envolve o DMA, que é utilizado para armazenar as palavras do *diff* na página de destino de acordo com o vetor de *bits*. Desta forma, nosso dispositivo DMA simplesmente realiza operações *scatter/gather* diretamente a partir dos vetores de *bits*.

O monitor de desempenho permitirá que a eficácia do controlador de protocolos seja avaliada em tempo de execução. Para essa finalidade, estatísticas de utilização de suas estruturas básicas incluindo o tamanho médio da FIFO, o total de escritas realizadas pelo processador principal e total de ciclos em que o processador principal esteve bloqueado, serão produzidas. Além disso, o CP implementará um relógio global para que novos métodos de avaliação de desempenho para sistemas de computação paralelos sejam desenvolvidos.

Em resumo, o *hardware* que propomos é extremamente simples e suas únicas partes customizadas são a lógica para *snooping* do barramento de memória e manipulação dos vetores de *bits*, e o nosso dispositivo de DMA. Maiores detalhes sobre o *hardware* do $NCP_2$ e como o seu dimensionamento foi realizado podem ser encontrados em [14].

## 3.3 O *Software* do $NCP_2$

Com o *hardware* descrito na seção anterior, o controlador de protocolos pode prover o seu processador associado com os mecanismos básicos utilizados por SW-DSMs. Mais especificamente, a funcionalidade oferecida pelo controlador que avaliamos nesse artigo é: a) pedido e envio remoto de página; b) pedido e envio remoto de *diff*; c) aplicação e geração dinâmica de *diff* localmente; d) envio e recepção de mensagem. As tarefas restantes do processamento relativo ao protocolo geralmente são mais complexas e portanto são executadas no processador principal. Para alcançar tal funcionalidade, definimos várias estruturas de dados e comandos pro controlador de protocolos, os quais estão descritos sucintamente nas subseções a seguir.

| Código | Operação |
|---|---|
| 1 | Pedido de página, processador p/ controlador |
| 2 | Pedido de página, controlador p/ controlador |
| 3 | Envio de página, controlador p/ controlador |
| 4 | Criação de diff, processador p/ controlador, interrompe processador |
| 5 | Criação e envio de diff, processador p/ controlador |
| 6 | Criação de diff, processador p/ controlador, não interrompe processador |
| 7 | Pedido de diff, processador p/ controlador |
| 8 | Pedido de diff, controlador p/ controlador |
| 9 | Envio de diff, processador p/ controlador |
| 10 | Envio de diff, controlador p/ controlador |
| 11 | Envio de mensagem, processador p/ controlador |
| 12 | Recepção de mensagem, controlador p/ controlador |

Tabela 1: Operações do Controlador de Protocolo utilizadas por TreadMarks

### Estruturas de Dados

As principais estruturas de dados utilizadas pelo controlador de protocolos compreendem: uma tabela de tradução de endereços, um contador de pedidos pendentes, uma tabela de *locks* de entradas de diretórios, uma fila de comandos, um buffer circular, e algumas variáveis de sincronização.

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

### Comandos

Os comandos utilizados na implementação do nosso TreadMarks modificado são apresentados na tabela 1. Em um trabalho mais detalhado [16], observamos que as aplicações sobre TreadMarks apresentam comportamentos comuns, como o enorme número de operações com diffs (códigos 7 a 10) e o pequeno número de transferências de páginas (códigos 1 a 3). A manipulação de diffs é muito custosa e, como acontece em grande quantidade em TreadMarks, justifica o nosso uso de um hardware dedicado a geração e aplicação de diffs. Além disso, o grande número de tranferências de mensagens pela rede (códigos 11 e 12) indica a necessidade de implementação de mecanismos eficientes para esse tipo de operação.

Como certos comandos do controlador podem necessitar da assistência do processador de computação, nosso controlador 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 da fila de comandos no controlador de protocolos ou logo depois de ser executado. Uma outra opção é a possibilidade de sinalizar, no comando, que o controlador de protocolos deve enviar um *acknowledgement* para o nó remoto, após esse comando terminar sua execução. Maiores detalhes sobre o *software* do $NCP_2$ podem ser encontrados em [16].

# 4 Avaliação do Sistema $NCP_2$

De forma a avaliar a performance do *hardware* e do *software* que propomos como base para o sistema $NCP_2$, simulamos a execução de TreadMarks modificado para tomar proveito dos nossos controladores de protocolo, e comparamos seus resultados com os de TreadMarks puro. Nessa secao apresentamos esses estudos, começando por uma descrição da metodologia que utilizamos.

## 4.1 Metodologia

O sistema paralelo que simulamos é bastante semelhante ao $NCP_2$, mas existem algumas diferenças, tais como o fato de simularmos um processador escalar. Independente disso, consideramos que nossas simulações permitem que façamos uma avaliação realística das características arquiteturais que propomos para o nosso computador paralelo.

Nosso simulador consiste de duas partes: *front end,* Mint [15], que simula a execução dos processadores, e *back end,* que simula o sistema de memória em detalhe (*write buffers* e caches com tamanhos finitos, comportamento de TLBs, 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). Em todas as referências a dados compartilhados, o *front end* chama o *back end* que simula as ações do protocolo simulado. A tabela 2 resume os parâmetros utilizados nas nossas simulações.

Apresentamos resultados para seis programas: TSP, Barnes, Radix, Water, Ocean e Em3d. TSP é da Universidade de Rice e faz parte do pacote do TreadMarks. As quatro aplicações seguintes são do conjunto Splash-2 [17]. Estas aplicações foram executadas com os tamanhos de entrada *default* para 32 processadores, como sugerido pelos pesquisadores de Stanford, com excessão de Barnes.

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.* 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 entre- e intra-moleculares em um conjunto de 512 moléculas de água. Utiliza-se um algoritmo $O(n^2)$ para a computação das interações. Ocean estuda movimentos em larga escala de oceanos baseado nas suas correntes. Simulamos uma grade oceânica de dimensão $258 \times 258$. Em3d [6] simula a propagação de ondas eletromagnéticas em objetos 3D. Simulamos 40064 objetos

| Constante do Sistema | Valor *Default* |
|---|---|
| Número de processadores | 16 |
| Tamanho da TLB | 128 entradas |
| Tempo de preencher a TLB | 100 ciclos |
| Qualquer interrupção | 400 ciclos |
| Tamanho da página | 4K bytes |
| Tamanho da cache | 128K bytes |
| Tamanho do *write buffer* | 4 entradas |
| Tamanho da linha de cache | 32 bytes |
| Tempo de *setup* da memória | 10 ciclos |
| Tempo de acesso à memória (depois do *setup*) | 3 ciclos/palavra |
| Largura do caminho de dados da rede | 8 bits (bidirecional) |
| *Overhead* de envio de mensagem | 200 ciclos |
| Latência de chaveamento | 4 ciclos |
| *Wire Latency* | 2 ciclos |
| Processamento de listas | 6 ciclos/elemento |
| Geração de *twin* | 5 ciclos/palavra + acessos à memória |
| Aplicação e criação de diff | 7 ciclos/palavra + acessos à memória |

Tabela 2: Valores *Default* dos Parâmetros do Sistema. 1 ciclo = 10 ns.

elétricos e magnéticos conectados aleatoriamente, com uma probabilidade de 10% de que objetos vizinhos residam em nós distintos. As interações entre objetos é simulada durante 6 iterações.

## 4.2 Resultados Experimentais

As figuras 3 e 4 apresentam os tempos de execução normalizados para cada uma das aplicações, tomando os tempos de TreadMarks puro como base e apresentando-os na barra à esquerda em cada grupo de barras. Cada barra compreende o tempo gasto pelo processador ocupado (*busy*), latência devido à busca e transferência de dados remotos (*pag*), *overheads* de sincronização (*synch*), *overheads* de comunicação entre processadores (*IPC*) e demais *overheads* (*others*), incluindo tempo de interrupção e latência devido a falhas na cache.

O impacto dos *overheads* no desempenho das aplicações sob TreadMarks puro é significativo e é mostrado nas figuras, desde a aplicação menos afetada (TSP) até a mais afetada (Ocean). Nessas duas aplicações, nossos controladores de protocolo conseguem reduzir o tempo de execução em 4 e 52% respectivamente. Water é a relativamente menos sensível, com redução de 10% enquanto que, nas demais aplicações, os tempos de execução são reduzidos em mais de 33%.

Uma análise do desempenho por categoria de overheads mostra que conseguimos reduzir significativamente o custo de transferência de dados remotos em Ocean, Em3d e Radix e menos nas demais. Em relação aos *overheads* de sincronização, a redução atinge fortemente duas aplicações (Barnes e Radix) e menos em Ocean e TSP. O tempo de interferência entre processadores (IPC) é drasticamente reduzido em todas as aplicações.

A capacidade da arquitetura que propomos de sobrepor computação útil com

overheads de ações de comunicação e coerência explica o ótimo desempenho alcançado na maioria das aplicações. Como mostramos detalhadamente em [4], dentre as técnicas de sobreposição avaliadas, a que obtem maior ganho de desempenho é a geração dinâmica de *diffs*, em todas as seis aplicações. A segunda característica mais importante é a capacidade do controlador de protocolos de executar tarefas simples sem interrromper o processador principal, contribuindo para o desempenho de quatro aplicações (Water, Radix, Barnes e Em3d). A técnica de busca antecipada também pode ser empregada com sucesso nas aplicações Em3d e Ocean.

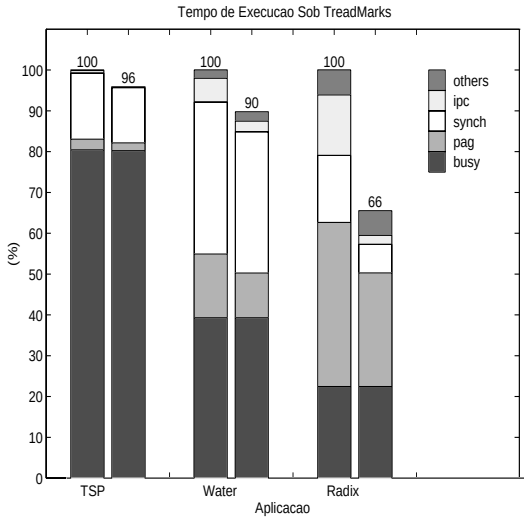


Figura 3: Tempos de execução: TM x $NCP_2$ + TM

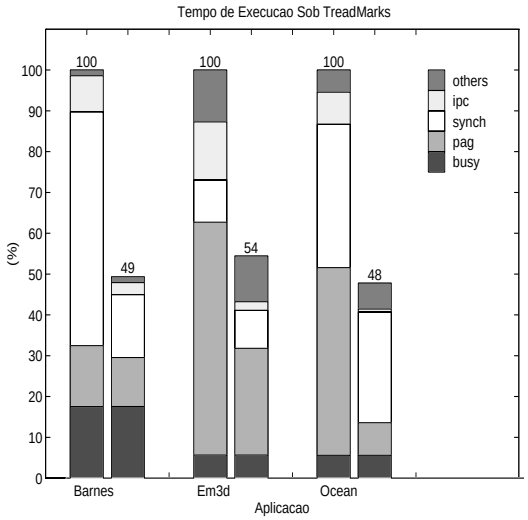


Figura 4: Tempos de execução: TM x $NCP_2$ + TM – Cont.

# 5 Trabalhos Relacionados

Diversos multiprocessadores com coerência de cache foram construídos recentemente [12, 1, 10]. 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 o seu uso mais difundido. 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 [11] e de Typhoon de Wisconsin [13]. Estes sistemas buscam prover um compartilhamento eficiente de dados ao nível de linhas de cache. A 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.

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 [5] como TreadMarks [9] foram projetados para execução em redes com estações de trabalho, sem nenhum *hardware* específico de suporte. Nosso trabalho mostra que uma melhora de desempenho signficativa pode ser obtida com uso de um *hardware* simples para esconder latência de comunicação e as perdas com a realização de coerência em sistemas deste tipo.

O trabalho de Iftode *et al.* [7] propõe AURC, um SW-DSM que usa um *hardware* específico para atualização automática de dados compartilhados. No artigo, AURC é comparado a TreadMarks puro, usando algumas das mesmas aplicações por nós utilizadas. Os autores mostram que o AURC supera TreadMarks em todas as aplicações. O desempenho de aplicações sobre o nosso TreadMarks modificado é igual ou melhor que o de AURC [4].

# 6 Conclusões

O sistema de computação paralela $NCP_2$ introduz a segunda geração de computadores de alto desempenho em desenvolvimento na COPPE/UFRJ. O $NCP_2$ irá implementar um sistema de memória compartilhada distribuída, permitindo que aplicações paralelas sejam mais facilmente programadas. Para alcançar alto desempenho numa classe maior de aplicações paralelas, o $NCP_2$ introduz controladores de protocolo implementados em *hardware* simples e de baixo custo. Nossas simulações de um sistema semelhante ao $NCP_2$ indicam que nossos controladores de protocolo permitem melhorar a performance de TreadMarks em até 52% para 16 processadores.

O primeiro protótipo do $NCP_2$ com 16 unidades de processamento deverá estar operacional ainda esse ano. Esperamos que o $NCP_2$ venha a proporcionar uma ótima oportunidade tecnológica para investimentos nacionais em computação de alto desempenho.

## Agradecimentos

Gostaríamos de agradecer à diretoria da COPPE/UFRJ, em especial ao ao prof. Luíz Pinguelli, atual diretor que, assim como no projeto NCP I, se empenhou em sua aprovação. Gostaríamos também de agradecer à Leonidas Kontothanassis pela ajuda na implementação da nossa infraestrutura de simulação e pelas inúmeras discussões sobre os tópicos desse trabalho.

# Referências

[1] 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.

[2] C. L. Amorim, R. Bianchini, G. Silva, R. Pinto, M. Hor-Meyll, M. De Maria, L. Whaterly, and J. Barros Jr. A segunda geração de computadores de alto

desempenho da coppe/ufrj. Technical Report ES-391/96, COPPE Sistemas, Universidade Federal do Rio de Janeiro, Junho 1996.

[3] C.L. Amorim, R. Citro, A. Ferreira, and E. Chaves Filho. O sistema de computação paralela ncp i. In *Anais do V Simpósio Brasileiro em Arquitetura de Computadores, SBAC-PAD*, pages 89–99, July 1993.

[4] 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 (ASPLOS 7)*, October 1996.

[5] 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.

[6] D. Culler *et al.* Parallel programming in split-c. In *Proceedings of Supercomputing '93*, pages 262–273, November 1993.

[7] 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.

[8] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, May 1992.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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 NCP2 da COPPE/UFRJ. Technical Report ES-394/96, COPPE Sistemas, Universidade Federal do Rio de Janeiro, Junho 1996.

[15] 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.

[16] 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 NCP2 da COPPE/UFRJ. Technical Report ES-395/96, COPPE Sistemas, Universidade Federal do Rio de Janeiro, Junho 1996.

[17] 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.