Monitoração e Análise de Desempenho do Sistema NCP_2*

A. Pereira E. Kraemer
DCC - UFMG
Belo Horizonte - MG
{adrianoc,kraemer,meira}@dcc.ufmg.br

W. Meira Jr. C. Amorim
COPPE Sistemas - UFRJ
Rio de Janeiro - RJ
amorim@cos.ufrj.br

Resumo

A paralelização de aplicações tem como grande atrativo a redução do tempo computacional necessário para a sua execução. O desenvolvimento de aplicações paralelas eficientes, entretanto, é freqüentemente uma tarefa árdua, principalmente em razão da complexidade das aplicações, arquiteturas e interações entre elas.

No caso da arquitetura NCP_2, a inovação representada pelo *hardware* e os protocolos de coerência de memória específicos tornam essa tarefa ainda mais árdua, não apenas para programadores como para os próprios projetistas.

Esse artigo apresenta uma proposta para a monitoração e análise de desempenho da arquitetura NCP_2. Essa proposta foi implementada sobre um emulador da arquitetura e a ferramenta resultante é apresentada e discutida em detalhes.

Abstract

The most appealing argument for parallelizing applications is reducing the computational time. Developing efficient parallel applications, however, is not a trivial task, as a result of the complexity of the applications, the architecture, and the interactions among them.

In the case of the NCP_2 architecture, the innovation associated with the hardware and the memory coherence protocols make this task even harder, not only for programmers, but also for system designers.

This paper presents a proposal for performance monitoring and analysis of the NCP_2 architecture. This proposal was implemented using an emulated architecture and the produced tool is presented and discussed in detail.

1 Introdução

A paralelização de aplicações tem como grande atrativo a redução do tempo computacional necessário para a sua execução. O desenvolvimento de aplicações paralelas eficientes, entretanto, é freqüentemente uma tarefa árdua, principalmente em razão da complexidade das aplicações, arquiteturas e interações entre elas. Entendimento de desempenho é uma tarefa essencial não apenas para melhorar o desempenho das aplicações, mas também avaliar e aperfeiçoar os ambientes de execução. Desta forma, *entendimento de desempenho*

*Este projeto foi financiado com recursos da FINEP e CNPq.

é a tarefa de identificar decisões de implementação (p.ex., estratégia de paralelização, escalonamento de iterações, distribuição de dados) ou características da arquitetura que explicam o desempenho de uma aplicação paralela.

A maior motivação para o desenvolvimento de ferramentas que auxiliem o entendimento de desempenho é o comportamento dinâmico exibido por sistemas paralelos. Esse dinamismo normalmente se manifesta através da ocorrência de fenômenos de desempenho tais como execução de computação extra resultante da paralelização, ou o bloqueio de processadores enquanto estes sincronizam ou esperam que requisições a outros processadores sejam satisfeitas.

CARNIVAL é uma ferramenta que permite ao programador entender como o dinamismo do sistema paralelo afeta o seu desempenho, através da incorporação de técnicas para entendimento de desempenho baseadas no paradigma de *análise de causa e efeito* (ACE) [6]. De acordo com esse paradigma, cada classe de fenômenos de desempenho é analisada por uma técnica que gera caracterizações e é particular à classe. Caracterizações sumarizam as operações efetuadas pelo programa (p.ex., segmentos de código executados, operações de coerência de memória) que causam a ocorrência de fenômenos, ou seja, caracterizações identificam interações dinâmicas entre operações executadas em um ou mais processadores e quantificam o seu efeito em termos de desempenho ao longo da execução. O uso dessas caracterizações e perfis de desempenho permite ao programador identificar com rapidez e precisão as causas de degradação de desempenho. Com base nessa informação, o programador altera a implementação da aplicação em termos da distribuição de dados, escalonamento de tarefas e iterações, entre outras decisões de implementação, com o objetivo de melhorar o desempenho da aplicação.

Neste artigo apresentamos uma proposta de monitoração e análise de desempenho para a arquitetura NCP_2, que apresenta um comportamento extremamente dinâmico não apenas pelas funcionalidades dos seus componentes, mas também pelo *software* que a utiliza, justificando a utilização de uma ferramenta como CARNIVAL. Este artigo está dividido em seis seções. Esta seção serve como introdução. Na próxima seção descrevemos brevemente a arquitetura NCP_2, seguido da descrição da implementação do *Treadmarks*, um sistema de memória compartilhada na Seção 3. A utilização do CARNIVAL para analisar o desempenho da arquitetura NCP_2 e' descrita na Seção 4. As duas últimas seções apresentam trabalhos correlatos e as conclusões.

2 A Arquitetura NCP_2

Nesta seção descrevemos os componentes da arquitetura NCP_2 e seu funcionamento. A arquitetura NCP_2 consiste de vários nós de processamento interconectados por uma rede de alta velocidade, que na implementação corrente é a *Myrinet*. Cada nó de processamento é composto de uma estação de trabalho de mercado e um controlador de protocolos, que é responsável pela execução de várias tarefas inerentes ao suporte de memória compartilhada distribuída (MCD): (1) processamento de tarefas básicas de comunicação e coerência sem utilização do processador principal, (2) busca antecipada de páginas e atualizações (*diffs*) das mesmas e (3) geração e aplicação dinâmica de atualizações de páginas com suporte de *hardware*. O funcionamento do controlador de protocolos é baseado em comandos, dessa forma, sempre que um processador (local ou remoto) necessita que o controlador de protocolos (CP) execute alguma tarefa ele a coloca na fila de comandos do CP, que as executa por ordem de chegada.

O controlador de protocolos é composto de um processador local, um dispositivo DMA, memória e uma lógica de *snoopy* que monitora escritas às páginas compartilhadas. Na

implementação corrente, cada nó do sistema NCP_2 é composto pelo controlador de protocolos descrito acima, de um processador principal PowerPC604 de 100 MHz com 32 Mbytes de memória principal (DRAM) e 256 Kbytes de cache L2, placa de rede comercial Myrinet de 1,28 Gbits/s e switch Myrinet com 8 portas.

A principal característica da arquitetura NCP_2[12] é o suporte à programação em memória compartilhada, onde todos os processos que compõem a aplicação acessam o mesmo espaço de endereçamento. O grande desafio que surge ao se utilizar esse modelo é manter a coerência de memória em todos os processadores, ou seja, fazer com que as alterações feitas por um processador sejam vistas pelos demais. Para resolver esse desafio são utilizados protocolos de coerência, que podem ser implementados em *hardware*, *software*, ou ambos. As implementações em *hardware* provêm alto desempenho, mas a um custo muito elevado. Por outro lado, implementações em *software* têm um custo muito menor, mas normalmente provêm desempenho aceitável apenas a uma gama limitada de aplicações. Abordagens híbridas, combinando arquiteturas tradicionais e redes de alta velocidade têm tido muito sucesso, combinando bom desempenho com baixo custo, como é o caso do NCP_2, que introduz suporte de *hardware* de baixo custo para implementação de protocolos de coerência de dados em *software*, denominados *software* MCDs.

Em termos de desempenho, o principal problema de sistemas *software* MCD é que os custos computacionais associados a comunicação e manutenção da coerência de dados podem ser muito significativos. Mais ainda, a natureza da computação de sistemas MCD é bastante complexa demandando uma extensa análise e entendimento do funcionamento dos protocolos, sistema operacional e da aplicação, o que não é uma tarefa trivial.

Por exemplo, uma tarefa normalmente árdua é associar custos de operações de coerência a segmentos de código ou estruturas de dados, devido à natureza dinâmica da comunicação e sincronização em sistemas MCD. O custo de uma operação é, freqüentemente, distribuído em termos de tempo (p.ex., uma operação de escrita executada por um processador causa uma invalidação em outro somente quando a próxima sincronização ocorrer) e espaço (p.ex., uma requisição por um processador deve ser satisfeita por outro, não importando o que o segundo estiver executando), tornando difícil entender as reais causas da degradação de desempenho observada durante a execução e essencial a presença de uma ferramenta que propicie facilidades para entendimento das reais causas de problemas de desempenho, justificando a utilização do CARNIVAL nesse projeto.

Na próxima seção descrevemos a implementação de um protocolo MCD, *Treadmarks*, na arquitetura NCP_2.

3 A Implementação do *TreadMarks* na arquitetura NCP_2

Um sistema passível de implementação na arquitetura NCP_2 é o *TreadMarks*[10], que se baseia no modelo de consistência de memória compartilhada *Lazy Release Consistency* [2]. Nesse modelo, a memória compartilhada pode ser modificada simultaneamente por mais de um processo, sendo as atualizações notificadas a outros processadores durante operações de sincronização.

O *TreadMarks* é baseado em invalidações de páginas, de forma que as atualizações de páginas de memória são divididas em 2 etapas: (1) as páginas que foram escritas por mais de um processo são invalidadas quando os processos sincronizam (barreira, *lock*) e (2) as alterações das páginas são requisitadas e aplicadas quando é feito o primeiro acesso a elas após a invalidação.

Utilizando o *TreadMarks* em sua versão original, a manutenção de coerência degrada o desempenho global do sistema MCD, devido à ocorrência de interrupções, muitas delas de alto custo, durante a computação útil do programa para tratar eventos externos, como requisições de páginas, atualizações, e tratamento de pedidos de outros processos. No NCP_2, a utilização do controlador de protocolos permite aliviar esta situação. Cada operação é segmentada em várias pequenas operações executadas por um ou mais componentes do sistema NCP_2, local ou remotamente. Nas subseções seguintes, analisamos três tipos de operações do *TreadMarks* em detalhe e sua implememtação no NCP_2 [13]: (1) requisição de página remota, (2) requisição de alterações, (3) sincronização em barreira.

3.1 Pedido de página para nó remoto

Nessa seção analisamos as operações envolvidas quando um processo acessa uma página que não está disponível em sua memória principal. Essa seqüência de operações pode ser visualizada na figura 1. O pedido se inicia quando ocorre um *page fault*, provocando a ativação do *handler* correspondente. Este envia para o CP (colocando na fila de comandos) um comando LOCAL_PAGE_REQUEST, contendo as informações da página. Esse comando é processado pelo CP local que envia uma mensagem REMOTE_PAGE_REQUEST para o CP do nó remoto que possui a página. O CP remoto recebe a mensagem e adiciona à sua fila o comando correspondente. Quando este comando é processado, a página requisitada é lida da memória principal e incorporada a uma mensagem PAGE_REPLY para o requisitante. Após o envio da resposta, o CP interrompe o processador principal notificando-o de qual processo contém a versão mais recente daquela página. A recepção do PAGE_REPLY faz com que o CP local gere um comando de mesmo tipo que é enfileirado e, ao ser executado, escreve a página recebida em memória e notifica o PP, que continua a sua execução.

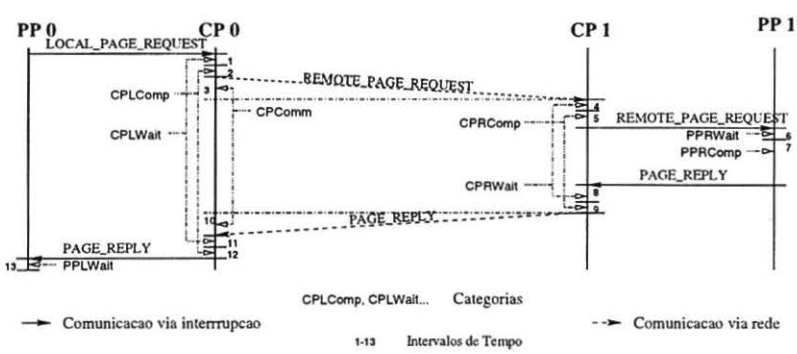

Figura 1: Requisição de Página Remota

3.2 Pedido de atualização remota

Analisamos nesta seção como processos atualizam suas cópias de página locais. Essa atualização ocorre quando um processo acessa uma página compartilhada que se encontra invalidada, ativando o *handler* que envia o comando LOCAL_DIFF_REQUEST para o CP local. Quando o CP executa o comando, uma mensagem REMOTE_DIFF_REQUEST é

enviada para os CPs dos nós remotos que possuem versões mais atualizadas da página. A recepção da mensagem causa a criação de um comando que é enfileirado pelo CP remoto. Ao executá-lo, as modificações requisitadas são agrupadas em uma mensagem DIFF_REPLY que é enviada para o CP requisitante. O PP remoto também é interrompido e informado do nó requisitante da atualização. Ao receber a mensagem DIFF_REPLY, o CP local enfileira um comando que, quando executado, grava as modificações indicadas e notifica a disponibilidade da página ao PP, que continua a sua execução. Esta seqüência de operações está ilustrada na figura 2.

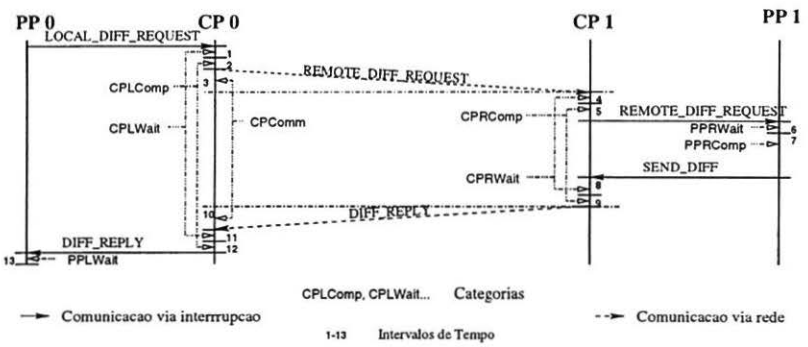

Figura 2: Requisição de Atualização *diff*

3.3 Sincronização por barreiras

A sincronização por barreiras envolve uma intensa troca de mensagens entre os nós, para que todos os processos sejam notificados das atualizações ocorridas. Ao chegar a uma barreira, o processo verifica se ele é o gerente. Caso afirmativo, ele deve receber a confirmação da chegada dos outros processos e agrupar as modificações feitas por eles. Caso contrário, o nó deve avisar ao gerente de sua chegada à barreira, o que é feito através do envio do comando SEND_MESSAGE para o seu CP. O CP enfileira o comando, e no momento de sua execução manda uma mensagem RECEIVE_MESSAGE para o CP do nó gerente. A partir dessa mensagem é gerado um comando no CP remoto (do gerente), que aciona o *handler* que trata da barreira. Quando todos os processos chegam à barreira, o PP enfileira um comando SEND_MESSAGE para cada nó, indicando o fim de barreira, e informando as páginas a serem invalidadas. Quando executado, esse comando causa o envio de uma mensagem RECEIVE_MESSAGE a cada CP. Cada CP recebe a mensagem, gera um comando que aplica as modificações descritas e interrompe o PP, que continua a execução normal do programa. Esse esquema é ilustrado na figura 3, onde estão representados dois processos, um sendo o gerente.

4 Utilizando CARNIVAL para Análise de Desempenho da Arquitetura NCP_2

CARNIVAL é uma ferramenta que permite ao programador entender como o dinamismo do sistema paralelo afeta o seu desempenho, através da utilização de técnicas para enten-

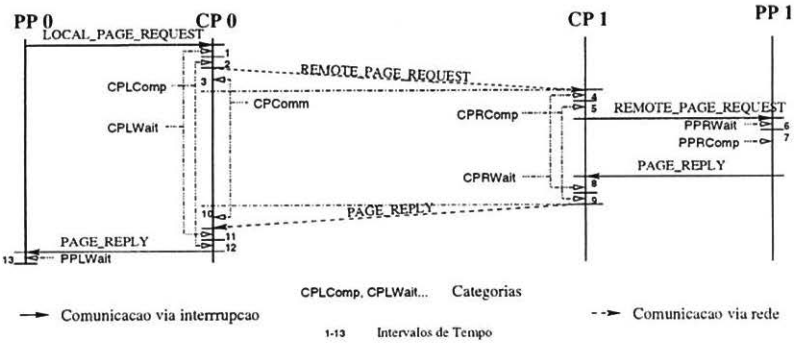

Figura 3: Sincronização de Nós Através de Barreira

dimento de desempenho baseadas no paradigma de análise de causa e efeito (ACE).

Nesta seção descrevemos a nossa estratégia de instrumentação, análise e visualização de desempenho da arquitetura NCP_2. Essa estratégia foi implementada no emulador do NCP_2 e as visualizações resultantes também são apresentadas. Por fim, discutimos a viabilidade de implementar essa estratégia na arquitetura NCP_2 propriamente dita.

4.1 Instrumentação

Do ponto de vista de instrumentação, informações de desempenho são coletadas em dois níveis: (1) perfis de desempenho e (2) caracterizações de tempo de espera.

Os perfis de desempenho são capturados através de instrumentação inserida nos códigos da aplicação, do *Treadmarks* e do controlador de protocolos que acumula o tempo de execução em quatro dimensões: (1) Nó de processamento, (2) Segmento de código associado, (3) Categoria de processamento, (4) Operação do protocolo MCD. Algumas dessas dimensões são abstraídas durante as medições. Por exemplo, operações não são consideradas quando se mede o custo no código da aplicação ao passo que escopo não tem significado para tempo de execução dentro das bibliotecas do *Treadmarks*. A diferenciação entre o processador principal e o controlador de protocolos se faz através de categorias. As medições do processador principal se enquadram em uma das quatro categorias seguintes: (1) `Comp`: Custo computacional da aplicação, (2) `Extra`: Custo computacional para manutenção da coerência (i.e., custo executando código do *Treadmarks*), (3) `Wait`: Tempo de espera para as requisições serem respondidas, (4) `Prste`: Custo computacional associado a requisições de nós remotos ou do CP local, que são efetuadas através de interrupções. Os custos no controlador de protocolos, por sua vez, são divididos em três categorias: (1) `CPComp`: O CP está executando comandos retirados da fila, (2) `CPWait`: O CP fez alguma requisição ao PP do mesmo nó e está esperando por uma resposta, (3) `CPIdle`: Tempo durante o qual não há comandos a serem processados.

A caracterização dos tempos de espera explora a forma de implementação das diversas operações do sistema MCD. Assim, as mensagens que transitam no sistema contém informações que são utilizadas para a geração das caracterizações. O objetivo dessas caracterizações é explicitar quais as atividades executadas a nível de sistema quando do atendimento de uma requisição. Um exemplo de tempo de espera passível de caracterização é o associado a falhas de página. Nesse caso, o processo que requisitou a página

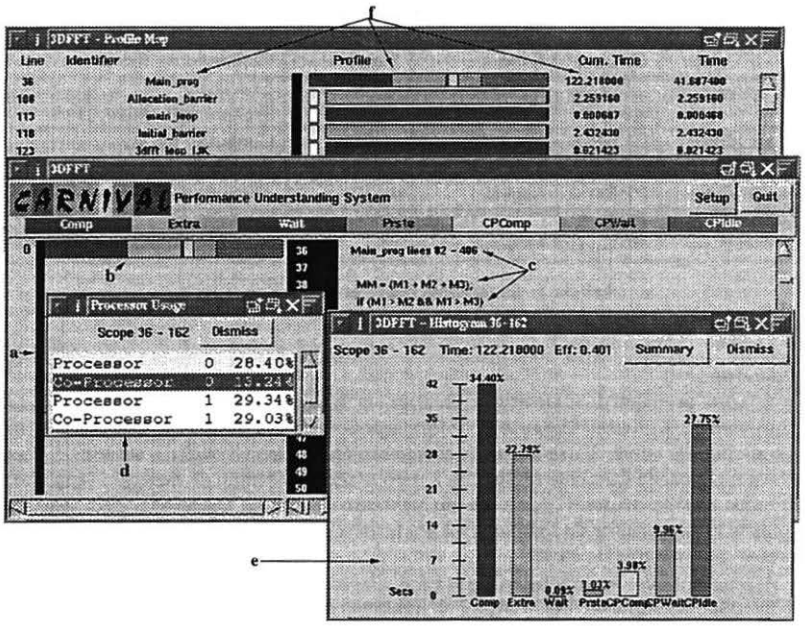

Figura 4: Perfis de Desempenho do CARNIVAL

envia a requisição ao CP local e espera pela resposta. Como visto na Seção 3, várias mensagens são trocadas, inclusive interrompendo o processador que contém a página requisitada. Com o intuito de medir o custo de cada um dos passos para obtenção de uma página, os vários tempos são medidos e acumulados nos próprios corpos das mensagens de forma que, quando a resposta chega ao requisitante, é possível determinar quais são as suboperações de maior custo. As caracterizações de tempo de espera da arquitetura NCP_2 explicam o custo de uma requisição classificando os vários custos em oito categorias. Essas categorias, sua localização e descrição podem ser vistas na Tabela 1.

4.2 Análise

Após coletados, os perfis de desempenho e as caracterizações são agrupados hierarquicamente, de forma a facilitar a análise por parte dos programadores e projetistas de protocolos MCD. As caracterizações também são associadas às respectivas instâncias de tempos de espera.

4.3 Visualização

As informações resultantes da análise podem ser visualizadas por uma interface Tcl/Tk [9] gerada automaticamente pelo CARNIVAL. A base desta interface é uma biblioteca de funções Tcl/Tk que implementam as várias janelas e manipulam dados numéricos. Nesta seção descrevemos a interface e como ela pode ser usada para entender o desempenho da

Categoria	Local	Tipo	Intervalos	Descrição
CPLWait	CPLocal	Espera	1 e 11	Comando na fila
CPLComp	CPLocal	Computação	2 e 12	Execução
CPComm	Rede	Comunicação	3 e 10	Transmissão entre CPs
CPRWait	CPRem	Espera	4 e 8	Comando na fila
CPRComp	CPRem	Computação	5 e 9	Execução
PPRWait	PPRem	Espera	6	Aviso de interrupção do PP
PPRComp	PPRem	Computação	7	Execução
PPLWait	PPLocal	Espera	13	Aviso de interrupção do PP

Tabela 1: Categorias dos Custos de Requisições

arquitetura.

A janela principal do CARNIVAL é dividida em duas partes. O código fonte da aplicação é apresentado à direita (Figura 4 **c**); informação sobre cada escopo de código é mostrada à esquerda. Os números das linhas de código são mostrados no centro e em tons de cinza, onde a intensidade da escala representa a percentagem de tempo de execução (cumulativa em todos os processadores) gasta em um segmento de código. Usuários podem identificar rapidamente segmentos de significativos em termos de custo computacional verificando as porções mais escuras da escala.

A porção esquerda da janela (Figura 4 **a** e **b**) identifica os escopos no programa. Um escopo pode ser um laço, uma função, ou um bloco de código ou mesmo o programa todo. Um escopo é caracterizado unicamente pela numeração das linhas que delimitam o seu início e o seu fim. Uma barra em tons de cinza é apresentada por toda a extensão de cada escopo; a intensidade da barra indica o tempo de execução cumulativo associado ao escopo. Diferentemente da escala de tons de cinza, essa escala inclui tempos associados a escopos aninhados, ou seja, o escopo mais externo (normalmente identificado por "0") possui a barra mais escura. Se o usuário "clicar" na barra vertical de um escopo, é produzido um perfil do escopo dividido por processador (Figura 4 **d**).

As cores na barra horizontal no início de cada escopo (Figura 4 **b**) representam a divisão do tempo associado ao escopo entre categorias. Se o usuário "clicar" nesta barra, é produzido um histograma detalhado das categorias e tempos associados a cada uma delas (Figura 4 **e**). De forma a facilitar a navegação na janela principal, há o *Profile Map* (Figura 4 **f**) que apresenta as barras horizontais e verticais de forma condensada em conjunto com informações sobre o escopo: tempo cumulativo, localização e categoria. Utilizando o *Profile Map*, o usuário pode facilmente identificar os escopos importantes da aplicação e acessá-los "clicando" na respectiva barra horizontal, quando a janela principal passa a mostrar o código associado ao escopo.

As caracterizações dos tempos de espera são apresentadas em outra janela, a *CPWait Map* (Figura 5) que apresenta, para cada fonte de tempo de espera as seguintes informações:

Localização: O escopo onde aconteceu a espera (Figura 5 **a**).
Categoria: A categoria de tempo de espera (`Wait` ou `CPWait`) (Figura 5 **b**).
Operação: A operação de protocolo associada ao tempo de espera (Figura 5 **c**).
Perfil: Barra colorida dividindo o tempo de espera entre categorias (Figura 5 **d**).
Tempo: Tempo cumulativo em todos os processadores (Figura 5 **e**).

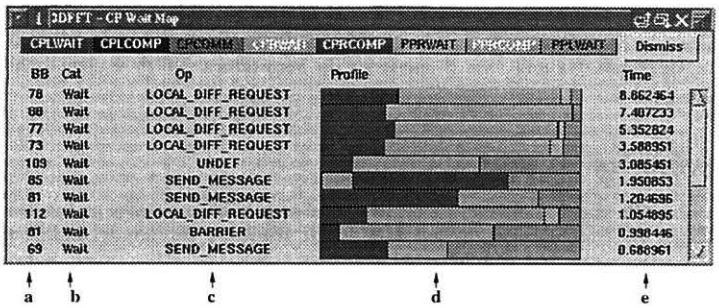

Figura 5: Caracterizações de Operações

4.4 Implementação

Tendo em vista o caráter cumulativo das medições e a existência de um relógio global acessível a todos os nós, a implementação da monitoração descrita é trivial. A tabela 1 mostra como os intervalos de tempo (números de 1 a 13 nas figuras 1, 2 e 3) são associados a cada categoria.

5 Trabalhos Correlatos

Abordagens tradicionais de análise de desempenho não são normalmente suficientes para o entendimento do comportamento dinâmico de aplicações, o que obriga o programador a derivar explicações baseado em sua experiência e conhecimento a respeito da aplicação e da arquitetura. Por exemplo, o uso de ferramentas que permitem visualizar a execução de uma aplicação a nível de eventos, tais como Nupshot [4], PPUTTs [5] e Paragraph [3], para entendimento de desempenho é laborioso e a geração de explicações errôneas é freqüente, uma vez que o número de eventos a serem analisados é enorme e a generalização das observações feitas é difícil. Outras ferramentas, tais como PP [1], NPI [11] e Paradyn [8], apresentam informações de desempenho a um nível mais abstrato, com o objetivo de diminuir o esforço requerido quando da análise. Embora essas abstrações sejam normalmente mais intuitivas ao programador, informações importantes a respeito do comportamento dinâmico da aplicação são descartadas.

6 Conclusões e Trabalhos Futuros

Neste artigo apresentamos uma proposta para monitoração e análise de desempenho da arquitetura NCP_2. Também descrevemos como a ferramenta CARNIVAL será utilizada por usuários e projetistas para analisar e entender o desempenho das aplicações e do *hardware* do NCP_2. Cabe ressaltar que essa proposta leva em consideração os recursos de monitoração do NCP_2 e sua implementação será trivial na arquitetura propriamente dita.

De forma a melhorar a ferramenta e enriquecer as informações por ela providas, estamos analisando outras aplicações e verificando a criação de classes de caracterizações, de forma a evitar os "efeitos de compensação" associados a dados puramente cumulativos. Essas classes seriam definidas e criadas à luz da teoria de Análise de Causa e Efeito [6, 7].

Finalmente, gostaríamos de agradecer a Raquel Pinto, Lauro Whately e Rodrigo Santos pela inestimável ajuda com o emulador da arquitetura NCP_2 e com detalhes da arquitetura em si.

Referências

[1] M. Crovella and T. LeBlanc. Performance debugging using parallel performance predicates. In *Proceedings of the 3rd ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 140–150, May 1993.

[2] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J.L. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, Maio 1990.

[3] M. Heath and J. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, pages 29 - 39, September 1991.

[4] V. Herrarte and R. Lusk. *Studying parallel program behavior with Upshot.* Argonne National Lab, 1991.

[5] T. LeBlanc, J. Mellor-Crummey, and R. Fowler. Analyzing parallel program executions using multiple views. *Journal of Parallel and Distributed Computing*, 9:203–217, June 1990.

[6] W. Meira Jr. *Understanding Parallel Program Performance Using Cause-Effect Analysis.* PhD thesis, Dept. of Computer Science – University of Rochester, Rochester, NY, July 1997. Available as TR 663 – DCS – University of Rochester.

[7] W. Meira Jr., T. LeBlanc, and V. Almeida. Using cause-effect analysis to understand the performance of distributed programs. In *Proceedings of SPDT98: SIGMETRICS Symposium on Parallel and Distributed Tools*, Welches, OR, August 1998. ACM.

[8] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, November 1995.

[9] John K. Ousterhout. *Tcl and Tk Toolkit.* Addison Wesley, 1994.

[10] Sandhya Dwarkadas et alli Pete Keleher, Alan L. Cox. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the USENIX Winter '94 technical Conference*, pages 17–21, Jan 1994.

[11] S. Sarukkai, J. Yam, and J. Gotwals. Normalized performance indices for message passing parallel programs. In *Proceedings of International Supercomputing Conference*, Manchester,UK, June 1994.

[12] 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/UFRJ, Junho 1996.

[13] 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/UFRJ, Junho 1996.