PVM × MPI: Um Estudo Comparativo em Redes de Workstations

Rímolo, G. S. (gugu@dcc.ufmg.br) Árabe, J. N. C. (arabe@dcc.ufmg.br)

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-010 Belo Horizonte, MG, Brasil
Telefone: (031) 499-5860 Fax: (031) 499-5858

Resumo

Redes de *workstations* têm sido consideradas como uma alternativa aos supercomputadores para a execução de aplicações computacionalmente intensivas. Hoje existem várias ferramentas que possibilitam ao usuário a utilização de uma rede de *workstations* para execução de aplicações paralelas, cada uma apresentando diferentes características em relação ao escalonamento de tarefas, balanceamento de carga, tolerância a falhas, topologia da rede, entre outras. Uma dessas ferramentas é o PVM (*Parallel Virtual Machine*), amplamente utilizado em todo o mundo. Outro sistema de utilização cada vez mais difundida é o padrão MPI (*Message Passing Interface Standard*). O objetivo principal deste trabalho é fazer uma análise comparativa entre o PVM e o MPI. Os resultados obtidos mostram que o MPI é mais eficiente para a execução de aplicações paralelas: suas primitivas de comunicação produzem melhores resultados e o seu gerenciamento do *buffer* de mensagens é mais seguro.

Abstract

Networks of workstations are becoming an alternative to high performance computing. Today there are many environments that allow the use of a network of workstations to execute parallel applications, each one having its particular features about task scheduling, load balancing, fault tolerance, network topology, and so on. One of these systems is PVM (Parallel Virtual Machine), widely used all over the world. Another system that has been largely used is MPI (Message Passing Interface Standard). The purpose of this work is to provide a comparative analysis of PVM and MPI. The achieved results show that MPI is more efficient: its communication primitives produce better results and the message buffer management is safer.

1. INTRODUÇÃO

Redes de *workstations* têm sido consideradas como uma alternativa aos supercomputadores para a execução de aplicações computacionalmente intensivas. A idéia básica é a utilização de várias *workstations* na execução simultânea de tarefas que compõem um programa paralelo. Hoje existem várias ferramentas que possibilitam ao usuário a utilização de uma rede de *workstations* para execução de aplicações paralelas, cada uma apresentando diferentes características em relação ao escalonamento de tarefas, balanceamento de carga, tolerância a falhas, topologia da rede, entre outras. Uma dessas ferramentas é o PVM (*Parallel Virtual Machine*) [9]. O PVM é um *software* que permite a execução de programas paralelos em um ambiente heterogêneo. Ele foi desenvolvido no Oak Ridge National Laboratory por um grupo acadêmico. Apesar de amplamente utilizado, apenas uma implementação é disponível, e não existe um padrão formalizado a partir do qual algum fabricante possa realizar a sua própria implementação.

Tal como o PVM, muitos outros sistemas são utilizadas na área de computação paralela. Numa tentativa de conciliar os principais atrativos desse conjunto de sistemas, um grupo de aproximadamente 60 pessoas e mais de 40 organizações iniciou um projeto de criação de uma interface padrão para trabalhar em um paradigma de troca de mensagens. Este projeto foi iniciado em abril de 1992 e seu primeiro resultado prático é o padrão MPI (*Message Passing Interface Standard*) [12]. Além do próprio PVM, as principais ferramentas que influenciaram o MPI são o Express [15], o p4 [3] e o Linda [5]. Outros trabalhos que também influenciaram o MPI foram: Chimp [6], Chameleon [10] e PICL [8], Zipcode [18] e Parmacs [4].

O objetivo principal deste trabalho é fazer uma análise comparativa entre o PVM e o MPI. Para isso, foram usados alguns *benchmarks* sintéticos, uma aplicação exemplo (Multiplicação de Matrizes) e um sistema real, complexo e já testado para programação paralela: o DOME (*Distributed Object Migration Environment*) [1] em desenvolvimento na Universidade de Carnegie Mellon (CMU).

O artigo começa apresentando as duas plataformas e suas principais diferenças. A seção 3 descreve os *benchmarks* e aplicações utilizadas nos experimentos. Os experimentos e resultados obtidos são discutidos na seção 4. Finalmente, a seção 5 conclui o artigo apresentando um balanço dos resultados alcançados e sugerindo pesquisas futuras.

2. PLATAFORMAS DE PROGRAMAÇÃO PARALELA

2.1 PVM - Parallel Virtual Machine

PVM [9] é um *software* que permite a execução de programas paralelos em um ambiente heterogêneo. Esse *software* atua interligando várias máquinas da rede, formando um ambiente preestabelecido pelo usuário. Esse ambiente passa a ser visto de uma forma transparente, como se fosse uma máquina. O PVM possui rotinas para inicializar tarefas e permitir a sua comunicação com outras tarefas via troca de mensagens. Do ponto de vista do usuário, pacotes de gerenciamento de um ambiente de computação distribuído diferem em alguns detalhes de implementação, mas os conceitos básicos são os mesmos em todos os casos. Ao iniciar-se a sua execução, um processo núcleo (*kernel*) de gerenciamento desse ambiente é iniciado para cada processador físico. Esses processos provêem um ambiente genérico onde as aplicações irão executar.

O sistema PVM é composto de duas partes. A primeira parte é um *daemon* que fica residente em todos os computadores que formam a máquina virtual. A segunda parte do sistema é uma biblioteca de funções da interface PVM. As aplicações PVM devem ser ligadas a essa biblioteca e podem ser executadas no *prompt* de qualquer nodo da máquina virtual PVM. Essa biblioteca contém funções que são chamadas pelo usuário para fazer troca de mensagens, disparar novas tarefas, modificar a máquina virtual, entre outras.

2.2 MPI – Message Passing Interface Standard

O trabalho sobre o MPI iniciou-se em 1992 no *Oak Ridge National Laboratory* e na *Rice University* para garantir que bibliotecas eficientes e *softwares* de aplicação possam ser desenvolvidos e portados para um amplo espectro de multicomputadores de alto desempenho. O MPI foi criado de forma a abranger as características mais importantes (úteis) das interfaces e bibliotecas já existentes sobre o paradigma de troca de mensagens. O MPI possui raízes no PVM, Express [15], p4 [3], Zipcode [18], Parmacs [4] e em sistemas desenvolvidos pelas empresas IBM, Intel, Meiko, Cray Research e Ncube.

O escopo do padrão MPI foi deliberadamente limitado ao modelo de programação de troca de mensagens. Não existe o conceito de espaço de endereçamento global. Questões de I/O paralela e criação dinâmica de processos foram deixados de lado intencionalmente. O tratamento de sinais da rede de comunicações e do sistema operacional também não é abrangido pelo MPI. Alguns dos principais objetivos do MPI são:

- o MPI pretende ser um padrão para aplicações e para bibliotecas de rotinas;
- o MPI visa multicomputadores e redes heterogêneas de computadores;
- a semântica da interface MPI foi escrita para ser independente de linguagem;
- o Foro MPI esforça-se para definir uma interface de troca de mensagens que seja possível de ser implementada eficientemente;
- confiabilidade e facilidade de uso são os objetivos principais do foro MPI.

Uma importante característica a ressaltar são algumas limitações impostas às primitivas de comunicação coletiva [7]. A sintaxe e semântica destas primitivas foram projetadas para serem consistentes com as primitivas de comunicação ponto-a-ponto. Entretanto, para manter um nível razoável de complexidade, foram feitas algumas restrições. Dentre elas, a simplificação maior é que as primitivas de comunicação coletiva só possuem versões bloqueantes.

PRINCIPAIS DIFERENÇAS DAS PLATAFORMAS

Comparações entre o MPI e o PVM são inevitáveis. O MPI apresenta algumas qualidades superiores, principalmente no que diz respeito à sua implementação, comunicação assíncrona, gerenciamento de *buffers* de mensagens, portabilidade, etc. [11]. Algumas diferenças significativas podem ser notadas entre as duas plataformas:

- o MPI não possui mecanismos para criação dinâmica de tarefas;
- o MPI possui um conjunto maior de primitivas de programação paralela;
- o MPI não possui funções para tratar a saída padrão das tarefas remotas;
- o PVM possui mecanismos sofisticados de gerência do ambiente de execução das tarefas paralelas, enquanto o MPI não possui nenhum recurso para isso;
- no MPI, o controle do *buffer* usado para troca de dados é feita pelo usuário, enquanto no PVM esse controle é feito pelo próprio ambiente;

3. BENCHMAKS E APLICAÇÕES

Alguns *benchmarks* foram criados para avaliar o desempenho das duas plataformas de programação paralela. Como a comunicação é um fator crucial para a programação paralela baseada em troca de mensagens, os principais testes de desempenho buscam avaliar o desempenho neste aspecto. Os *benchmarks* utilizados são basicamente a reprodução dos experimentos realizados em [14]. Além dos *benchmarks*, foram realizados experimentos com as aplicações Multiplicação de Matrizes e DOME.

3.1 Benchmark Ping

O objetivo deste *benchmark* é medir o *sturt-up coast* (custo de composição de *buffers*) ao enviar mensagens de um processo para o outro. Há um processo produtor (envia mensagens) e outro consumidor (recebe mensagens), como na figura 1. Na realização dos experimentos com *n* nodos, um nodo fica sendo o produtor, fazendo envio de mensagens (*ping*) para os *n-1* nodos restantes. A medida de tempo é feita pelo produtor, independente da velocidade de transferência das mensagens na rede (não bloqueante).

Figura 1: *Benchmark Ping*

3.2 Benchmark Ping-Pong

Neste *benchmark* será medida a latência *end-to-end*, enviando uma mensagem e a recebendo de volta. Ao contrário do *ping*, ambos os processos assumem a função de enviar e receber mensagens alternadamente, como pode ser visto na figura 2. A medida de tempo é feita pelo processo que envia a mensagem primeiro. A velocidade de transferência de mensagens na rede influi diretamente no tempo de resposta, já que o processo que envia uma mensagem também a recebe de volta. Para que isso não atrapalhe os experimentos é importante que os testes sejam feitos com a menor utilização possível da rede.

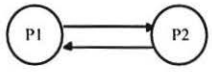

Figura 2: *Benchmark Ping-Pong*

3.3 Benchmark Broadcast

Neste *benchmark*, que avalia a comunicação coletiva de dados, é feita uma troca de mensagens de um processo para todos os demais processos do conjunto (figura 3). A medida de tempo é feita pelo processo que distribui os dados, como no *benchmark ping*.

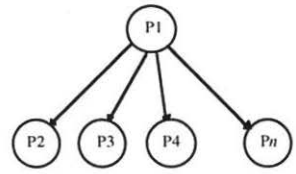

Figura 3: *Benchmark Broadcast*

3.4 Multiplicação de Matrizes

Existem na literatura diversos algoritmos para multiplicação de matrizes em paralelo. O algoritmo escolhido para os experimentos é apresentado em [16].

3.5 O Sistema DOME

O DOME [1] — *Distributed Object Migration Environment* — é um sistema que permite ao usuário escrever programas paralelos para uma rede heterogênea de workstations. O tipo de paralelismo suportado pelo DOME é o SPMD - *Single Program Multiple Data*. O DOME possui mecanismos internos e transparentes ao usuário de balanceamento de carga (não há migração de tarefas, apenas a migração de dados) e de tolerância a falhas. Ele também é o responsável pela criação de um ambiente de controle, que se responsabilizará pela criação de tarefas e distribuição inicial dos dados.

Todo o controle do ambiente paralelo no sistema DOME é feito sobre a plataforma PVM. A fim de fazer uma análise de desempenho comparativa entre as plataformas PVM e MPI, uma nova versão do DOME foi criada através da migração deste sistema para o padrão MPI. Detalhes sobre o processo de migração de uma plataforma para outra podem ser encontrados em [17]. Para avaliar alguns aspectos do sistema DOME nestas plataformas, dois *benchmarks* foram criados. O primeiro testa a comunicação coletiva entre as tarefas DOME; o segundo, a comunicação ponto-a-ponto. O objetivo destes *benchmarks* é recolher os dados de um objeto DOME (distribuído entre os processos) e fazer uma cópia local a todos os processos.

4. EXPERIMENTOS E RESULTADOS

Os experimentos foram realizados em um ambiente homogêneo formado por 6 workstations SUN SLC, executando SUN OS 4.1. Os testes foram realizados com as máquinas praticamente dedicadas aos testes, exceto pelos *daemons* do sistema. Com isso, a carga de trabalho da rede se manteve estavelmente baixa, o que resultou em séries de medições com tempos de resposta praticamente idênticos. Por isso, os desvios das medições apresentadas nos gráficos foram desconsideradas.

Os experimentos realizados com o MPI utilizaram a implementação LAM [2] por ser ela a melhor implementação MPI disponível publicamente [13, 14, 17]. Esses experimentos foram executados de duas maneiras: normal e com opção de otimização ("-c2c"). O LAM possui um mecanismo muito poderoso para controle e gerenciamento do *buffer* de mensagens. No entanto, isto acarreta um aumento significativo do tempo de resposta. A otimização minimiza o tempo de resposta em detrimento da monitoração do *buffer* de mensagens: as mensagens são passadas diretamente entre as tarefas da aplicação, sem a intervenção dos *daemons* do LAM.

4.1 Experimentos com o *Benchmark* Ping

Nos primeiros testes foram variados o número de workstations e o tamanho da mensagem ficou fixa em 1 Kbyte. Para que os tempos de resposta pudessem ser comparados, os *benchmarks* foram executados com 500 iterações. Nos testes seguintes, foi variado o tamanho da mensagem e fixado o número de nodos da máquina virtual.

As primeiras diferenças que se notam é entre os tempos de sistema e os tempos de CPU. Pode-se observar que no PVM o tempo de CPU é superior ao MPI (figura 4). Isso ocorre porque as mensagens primeiro devem ser inseridas no *buffer* PVM, e então enviadas ao processo destino. Esse tempo é então contabilizado pelo *daemon* PVM. Já no MPI, o envio dos dados é feito diretamente (da posição de memória em que se encontra), logo, não existe esse *overhead* de CPU. Em compensação, o tempo de sistema do PVM é menor, uma vez que a colocação e a retirada dos dados do *buffer* PVM é contabilizado no tempo de CPU, enquanto que no MPI são chamadas feitas pelos *daemons* LAM (figura 5).

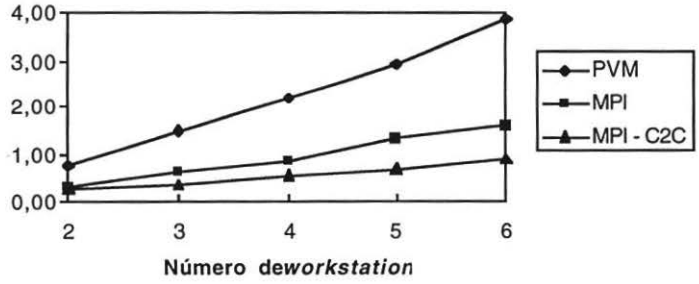

Figura 4: *Benchmark Ping* — tempo de CPU

O tempo real é mostrado na figura 6, onde se constata que o MPI é significativamente mais rápido que o PVM. No caso mais extremo, o tempo medido pelo MPI chegou a ser aproximadamente 4 vezes inferior ao tempo medido pelo PVM.

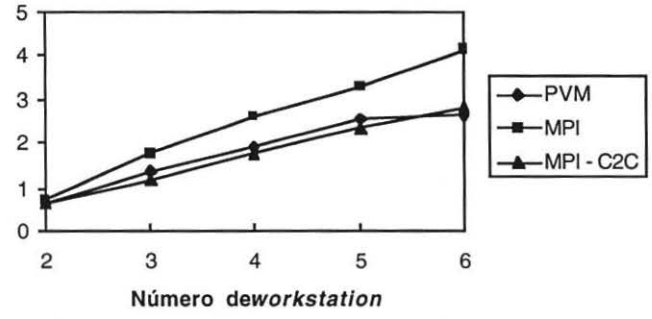

Figura 5: *Benchmark Ping* — tempo de sistema

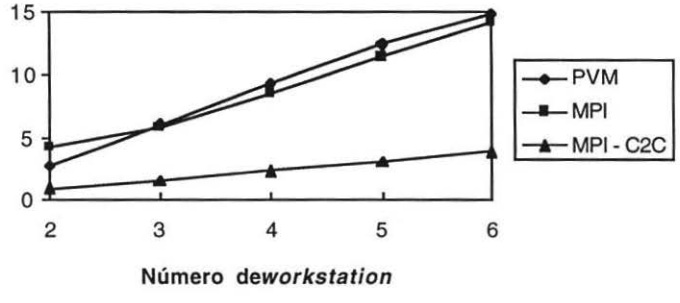

Figura 6: Benchmark Ping — *tempo real*

Na segunda bateria de testes com esse *benchmark*, o número de workstations foi fixada em 6 e o tamanho da mensagem variado de 256 bytes a 8 Kbytes. Também foram feitas 500 iterações para cada execução do *benchmark*. Esse teste mostrou que o PVM não garante a integridade do *buffer* de mensagens quando este não suporta armazenar

mais dados (figura 7). A máquina PVM neste caso é terminada e todas as tarefas são eliminadas.

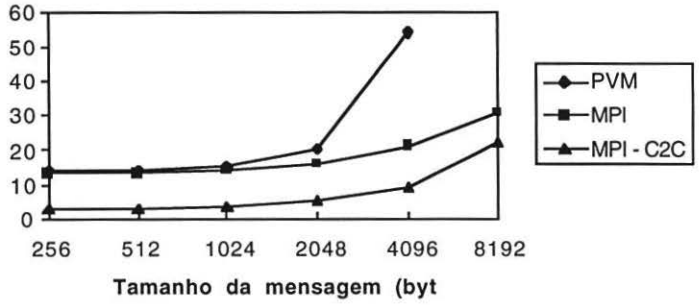

Figura 7: *Benchmark Ping* — tempo real

4.2 Experimentos com o *Benchmark* Ping-Pong

Neste benchmark o tamanho da mensagem foi fixado em 1 Kbyte e para cada execução foram feitas 500 iterações. As diferenças entre o PVM e o MPI foram reduzidas, uma vez que o processo consumidor esvazia o *buffer* de mensagens, evitando o seu congestionamento como verificado no benchmark *Ping*. No caso extremo, o tempo de execução obtido pelo PVM chega a ser aproximadamente 3 vezes maior que o tempo medido para o MPI. Pode-se observar também, a linearidade obtida neste gráfico (figura 8), uma vez que, a cada iteração, o *buffer* inicia-se vazio e termina neste mesmo estado.

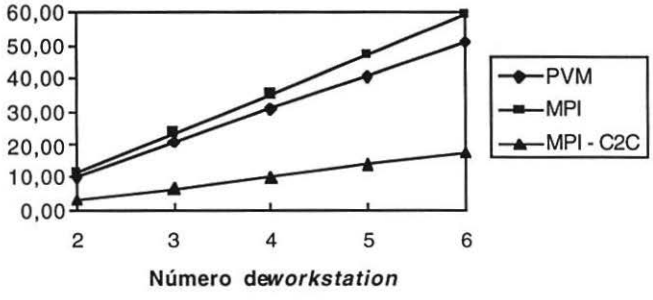

Figura 8: *Benchmark Ping-Pong*: tempo real

4.3 Experimentos com o *Benchmark* Broadcast

Os testes, que avaliam a comunicação coletiva, foram feitos nos mesmos moldes dos anteriores: variando o número de workstations, fixando o tamanho da mensagem em

1 Kbyte e realizando 500 iterações. Como se observa pelo gráfico da figura 9, à medida que o número de nodos da máquina virtual aumenta, o PVM aumenta substancialmente o tempo de resposta, enquanto que o MPI mostra-se mais regular. No ponto extremo, o tempo medido para o PVM foi cerca de 8 vezes maior que o tempo do MPI.

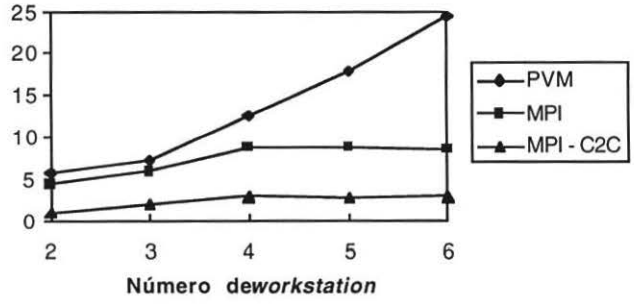

Figura 9: *Benchmark broadcast*: tempo real

4.4 Experimentos com o Algoritmo de Multiplicação de Matrizes

O algoritmo foi implementado em PVM e MPI. Os experimentos foram feitos em um ambiente homogêneo composto por 6 SUNs SLCs. Os primeiros experimentos mostram que a implementação MPI com a opção LAM "-c2c" tende a ser mais eficiente que o PVM. O gráfico da figura 10 foi gerado a partir de experimentos realizados variando o tamanho das matrizes (número de elementos *float*) e fixando-se o número de máquinas em 6. Já na figura 11, o tamanho das matrizes foi fixado em *100x100* e variou-se o número de máquinas. Nesta última, a implementação MPI chega a ser 30% mais eficiente que a implementação PVM.

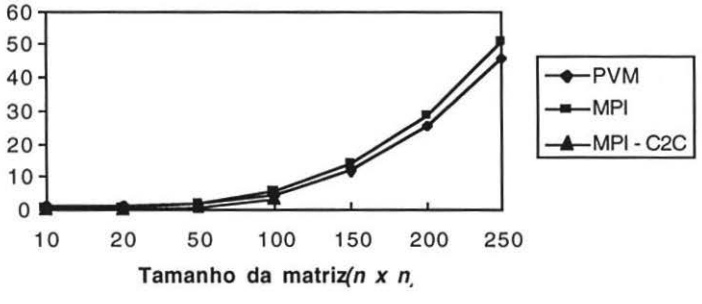

Figura 10: Multiplicação de matrizes (assíncrona): 6 *workstations*

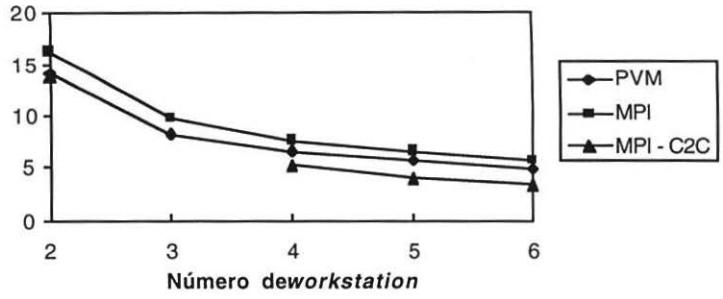

Figura 11: Multiplicação de matrizes (assíncrona): matrizes de tamanho *100x100*

Um resultado aparentemente surpreendente foi extraído dessas curvas (figura 10 e figura 11): em vários pontos a implementação MPI não chegou a nenhuma resposta quando usada com a opção "-c2c". Este resultado pode ser explicado pelo funcionamento dessa opção. A opção "-c2c" reduz a intervenção dos *daemons* LAM nas trocas de mensagens entre processos. Essa redução implica na redução da área de transferência (*buffer*) dos dados que são armazenados em cada processador. Quando a opção "-c2c" não é utilizada, os *daemons* possuem uma grande área para armazenamento de dados e total controle e monitoração das mensagens da máquina virtual.

Após a execução de diversos experimentos variando os parâmetros desse algoritmo, foi constatado que esse *buffer*, ao usar a opção "-c2c", possui uma capacidade em torno de 20 Kbytes. Ao preencher toda a capacidade desse *buffer*, as trocas de mensagens ficam bloqueadas até que o *buffer* comporte o armazenamento de novos dados.

O algoritmo de multiplicação implementado realiza trocas de dados assíncronas. Cada processo escravo, primeiro, envia os dados com que está trabalhando, e depois, recebe os dados do processo vizinho. Quando a quantidade de dados enviada preenche o *buffer* de mensagens, o processo fica bloqueado na operação *send*. Como todos os processos fazem isso, os processos ficam em *deadlock*. Por isso, a curva "MPI - C2C" não aparece na figura 12, onde o tamanho da matriz é *250x250*. Neste gráfico constata-se novamente que o desempenho do PVM é semelhante à do MPI sem otimização.

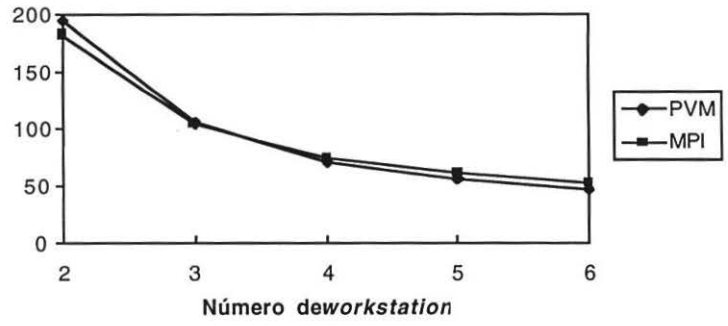

Figura 12: Multiplicação de matrizes (assíncrona): matrizes de tamanho 250x250

Para traçar todas as curvas por inteiro, foi feita uma pequena modificação no algoritmo de multiplicação: as trocas de mensagens passaram a ser feitas sincronamente. Um dos processos escravos, primeiro recebe os dados do seu vizinho, depois envia os seus. Dessa forma, o processo vizinho que mandou os dados não fica bloqueado (na operação *send*), podendo receber os dados do seu vizinho. O mesmo ocorre com o vizinho deste e, sucessivamente, todos os processos são desbloqueados.

Com esse novo algoritmo, a execução das tarefas estará garantida (os processos não entrarão em *deadlock)*, embora apareça um *overhead* adicional. Além do tempo de comunicação, surge um novo fator de custo: o tempo de sincronização.

Os experimentos com o novo algoritmo apresentam dois comportamentos. Com matrizes de tamanho até *150x150*, os resultados obtidos (figura 13 e figura 14) são semelhantes aos obtidos com o algoritmo original: o MPI obtém melhores resultados de desempenho, sendo em alguns casos, 30% mais eficiente que o PVM (figura 14).

Outro comportamento foi verificado quando o novo algoritmo foi executado com matrizes de tamanho maior (figura 13 e figura 15). Com o aumento dos tamanhos das matrizes, aumenta-se o número de mensagens e o tamanho dos dados trocados entre os processos. Isso acarreta em um aumento do tempo de comunicação. Quanto maior o tempo de comunicação, maior ainda é o tempo de sincronização, já que os processos são desbloqueados um a um. Como o tempo de sincronização é cada vez mais responsável pelo tempo total de execução do algoritmo, as implementações PVM e MPI tendem a chegar um mesmo tempo total de execução.

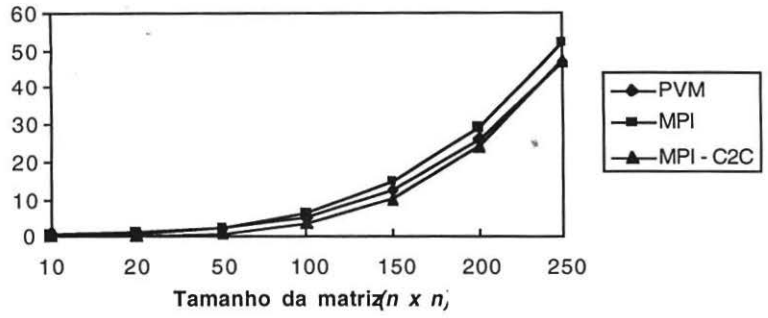

Figura 13: Multiplicação de matrizes (síncrona): 6 *workstations*

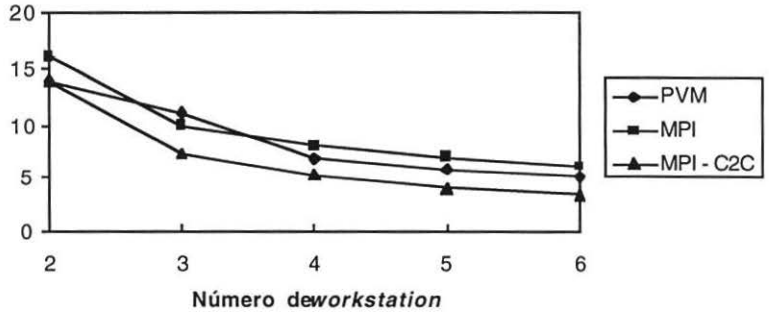

Figura 14: Multiplicação de matrizes (síncrona): matrizes de tamanho *100×100*

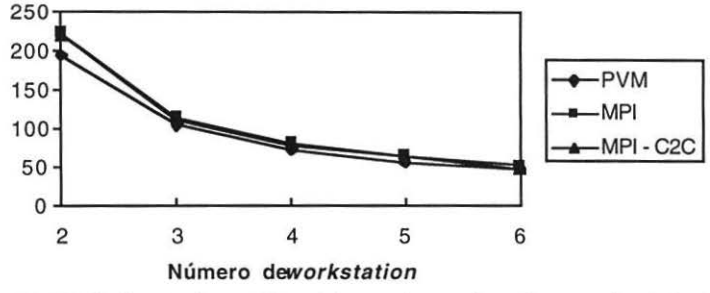

Figura 15: Multiplicação de matrizes (síncrona): matrizes de tamanho *250×250*

4.5 Experimentos com o DOME

Os experimentos foram gerados em um ambiente homogêneo formado por 6 *workstations* SUN SLC. O gráfico da figura 16 mostra que, para objetos (*dArrays*) pequenos, os tempos de resposta são parecidos para as duas implementações. As diferenças surgem para objetos maiores, onde o MPI se mostra mais eficiente. Para

10.000 elementos, o MPI foi em torno de 23% mais eficiente que o PVM. Já no gráfico da figura 17, o MPI se mostrou mais eficiente em todos os casos. O PVM, no ponto extremo, obteve tempos de resposta aproximadamente 55% mais lentos que os obtidos pelo MPI.

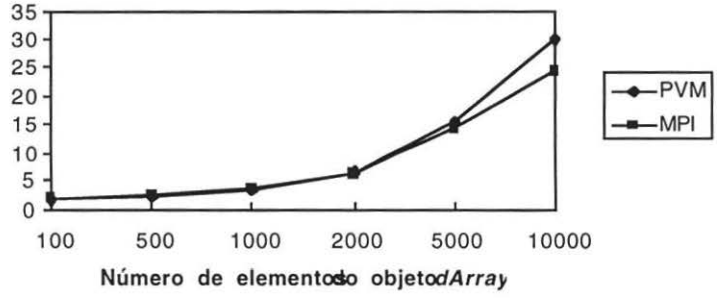

Figura 16: *Benchmark* DOME: comunicação coletiva

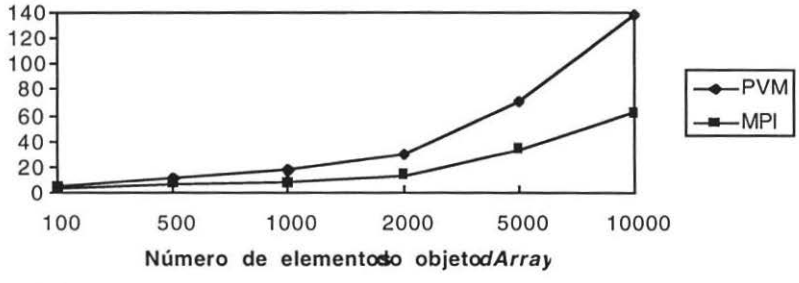

Figura 17: *Benchmark* DOME: comunicação ponto-a-ponto

5. CONCLUSÕES

Esse trabalho realizou uma comparação de desempenho entre a implementação do PVM e do MPI. O PVM é a plataforma de programação paralela mais utilizada em todo o mundo. O MPI é um padrão formal criado por vários pesquisadores e diversas empresas. O MPI foi criado na intenção de substituir as diversas plataformas de programação paralela por uma única plataforma, contendo as principais características.

A análise de desempenho das duas plataformas foi realizada através de vários experimentos: com *benchmarks* sintéticos, com a aplicação de multiplicação de matrizes e com um sistema real, o DOME. Com os *benchmarks* sintéticos, chegou-se a conclusão que as primitivas de comunicação MPI possuem um desempenho melhor que as

equivalentes PVM, e ainda, que o gerenciamento de *buffer* de mensagens do PVM não é seguro. Nos programas de aplicação, além dos gastos com comunicação, há os gastos com computação. Dessa forma, as diferenças são reduzidas, já que o custo de computação incide sobre as duas plataformas e reflete diretamente no tempo total gasto pela aplicação. Isso pode ser observados nos experimentos com a aplicação de Multiplicação de Matrizes. Mesmo assim, o MPI ainda se mostrou mais eficiente em alguns casos. Nos demais, as duas plataformas se mostraram equivalentes. Os experimentos com o DOME procuraram analisar a comunicação (ponto-a-ponto e coletiva) envolvida em algumas operações do sistema. Os resultados apresentados confirmam os resultados dos primeiros *benchmarks* sintéticos, onde o MPI produziu um melhor desempenho.

Como conclusão final pode-se dizer que os resultados vão depender muito da forma em que as aplicações sejam implementadas, porém, os custos das primitivas de comunicação MPI são inferiores às equivalentes PVM. E ainda, além de ser mais eficiente nesse aspecto, o MPI possui um gerenciamento de *buffer* seguro, o que não é garantido pelo PVM. O MPI, para aquilo que já foi definido, se mostra uma boa plataforma de programação paralela, superior em qualidade à plataforma PVM. Esta última, entretanto, possui outras características que ainda não são supridas pelo MPI. Com isso, um grande número de aplicações paralelas ainda precisam de uma plataforma como o PVM. O MPI está em constante debate através de foros. A sua tendência natural é a evolução. Assim, espera-se que as suas deficiências sejam solucionadas e, com o surgimento de novas versões, o MPI se torne uma plataforma cada vez mais abrangente. Dessa forma, um estudo comparativo deve ser feito periodicamente, mostrando o desempenho das implementações e das novas características adicionadas ao padrão atual.

Um ponto que pode marcar a evolução dessas plataformas é o modelo de paralelismo baseado em *threads*. *Threads* são essencialmente múltiplas seqüências de controles de um mesmo processo que compartilham porções de um mesmo espaço de endereçamento. Em termos de performance, o modelo baseado em *threads* aumenta significativamente o potencial para sobrepor computação e comunicação. Já é antecipado que as *threads* serão características padrões em grande parte dos sistemas nos próximos anos. O sistema PVM baseado em *threads* está em desenvolvimento e, embora o padrão MPI não especifique esse modelo, algumas implementações já trabalham com *threads*. Entretanto, essas implementações ainda não são seguras (*thread-safe*) e muitos aspectos são debatidos em listas de discussões na *Internet*.

Uma lacuna existente nesse trabalho é a realização de experimentos com um número maior de processadores. Eles poderiam ou não confirmar os resultados obtidos e ainda, mostrar novas características não apresentadas nos experimentos realizados nesse trabalho.

REFERÊNCIAS

[1] Árabe, J. N. C., Beguelin, A., Lowekamp, B., Seligman, E., Starkey, M. and Stephan, P. "DOME: Parallel Programming in a Computer Distributed Environment", *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, Hawaii, April 15-19, 1996.

[2] Burns, G., Daoud, R. and Vaigl, J. "LAM: An Open Cluster Environment for MPI", Ohio Supercomputer Center, Columbus, Ohio, 1994.
(ftp://lam@tbag.osc.edu/pub/lam/lam-papers.tar.Z

[3] Butler, R., and Lusk., E. "Monitors, Messages and Clusters: The P4 Parallel Programing System", *Parallel Computing*, 20:547-64, April, 1994.

[4] Calkin, R., Hempel, R., Hoppe, H., and Wypior, P. "Portable Programming with the PARMACS Message-Passing Library", *Parallel Computing, Special issue on message-passing interfaces*, 20:615-32, April, 1994.

[5] Carriero, N. and Gelernter, D., "Linda in Context", *Communications of the ACM*, Vol.32, N.4, April 1989, pp. 444--458.

[6] Clark, L., Trew, A., Heywood, N. White, M., "CHIMP Concepts", *Technical Report EPCC-KTP-CHIMP-CONC 1.2*, University *of* Edinburgh, June, 1991.

[7] Dongarra, J., Otto, S. W., Snir, M. and Walker, D. "A Message Passing Standard for MPP and Workstations", *Communications of the ACM*, Vol. 39, N° 7, June, 1996, pp.84-90.

[8] Geist, G. A., Heath, M. T., Peyton, B. W. and Worley, P. H. "A user's guide to PICL: a Portable Instrumented Communication Library", *Technical Report TM-11616*, Oak Ridge National Laboratory, October, 1990.

[9] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., "PVM: A User's Guide and Tutorial for Networked Parallel Computing". *MIT Press*, 1994. (ftp://www.netlib.org/pvm3/book/pvm-book.ps).

[10] Gropp, W. and Smith, B. "Chameleon Parallel Programming Tools User's Manual". *Technical Report ANL-93/23*, Argonne National Laboratory, March, 1993.

[11] *Top 10 Reasons to Prefer MPI Over PVM.* (http://www.osc. edu/Lan.mpi/mpi_ top10.html)

[12] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", *Technical Report CS--94--230*, Computer Science Department, University of Tennessee, Knoxville, TN, April, 1994.

[13] Nevin, N. "The Performance of LAM 6.0 and MPICH 1.0.12 on a Workstation Cluster", *Ohio Supercomputer Center OSC-TR-1996-4*, Columbus, Ohio, 1996.

[14] Nupairoj, N. and Ni, L. M. "Performance Evaluation of Some MPI Implementations on Workstation Clusters", Michigan State University, 1994.
ftp://epm.ornl.gov/~walker/mpi/papers/nupairoj-li.ps.Z

[15] Parasoft Corporation, Moronvia, CA. "Express User's Guide", version 3.2.5 edition, 1992. (Parasoft Corporation: parasoft@Parasoft.COM).

[16] Quinn, Michel J., "Parallel Computing - Theory and Practice", Mc Graw Hill Inc., USA, 1994, pp. 191-193.

[17] Rímolo, G. S. "Análise Comparativa de Duas Plataformas de Programação Paralela para Redes de Workstations", *Dissertação de Mestrado*, UFMG - Belo Horizonte, Brasil, Dezembro de 1996.

[18] Skjellum, A., and Leung, A. "Zipcode: a Portable Multicomputer Communication Library atop the Reactive Kernel", In D. W. Walker and Q. F. Stout, editors, *Proceedinds of the Fifth Distributed Memory Concurrent Computing Conference*, pages: 767-76. IEEE Press, 1990.