Otimização das Operações Coletivas para um Aglomerado de 8 PCs usando uma Rede Ethernet 10 Mbs baseada em Hub

Martha Torres[1], Sergio Takeo Kofuji[2]

[1] Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santader
A.A. 678, Bucaramanga, Colombia
{mxtd@uis.edu.co}
[2] Laboratório de Sistemas Integravéis, Universidade de São Paulo
CEP 05508-900, São Paulo, S.P., Brasil
{kofuji@lsi.usp.br}

Abstract—
This paper presents the optimization done on collective communication operations of message passing library MPICH version 1.1. The optimization efficiency was demonstrated evaluating the execution time of collective operations and measuring their impact on the total execution time of several application programs. We also compared our results with literature dates. We concluded implementing efficient collective communication operations on cluster based of Ethernet hub provides high performance gain.

Keywords— **Collective communication operations, MPI, performance evaluation**

I. INTRODUÇÃO

Na área de computação de alto desempenho, os aglomerados de estações de trabalho ou computadores pessoais interconectados por uma rede local surgem como uma alternativa aos computadores massivamente paralelos (*massively parallel computers* (MPC)). Embora os primeiros tenham sido utilizados por muitos anos nesta área, sua atual importância deve-se principalmente a três motivos. Primeiro, estações de trabalho de alto desempenho com microprocessadores que desafiam as arquiteturas tradicionais estão disponíveis a um custo relativamente baixo. Segundo, vários pacotes de *software* têm sido desenvolvidos no ambiente distribuído para ajudar ao desenvolvimento de aplicações paralelas. Finalmente, a disponibilidade de redes com ampla largura de banda como 100Mb/s Ethernet, ATM, Myrinet [LAU 97][HUA 96].

Ainda que a rede Ethernet baseada em hub não possua uma ampla largura de banda, aglomerados usando esta rede tem--se espalhado grandemente principalmente por razões econômicas. Portanto, realizar otimizações que melhorem o desempenho destas máquinas paralelas representa um aporte importante.

Operações de comunicação classificam-se em "ponto a ponto", as quais envolvem somente uma fonte e um destino, e operações de comunicação **coletiva** nas quais mais de dois processos participam. As operações coletivas podem ser usadas para movimento de dados (como *broadcast*, *scatter*, *gather*), operações de cálculo global (como redução, *scan*), e controle de processos (como sincronização de barreira). Operações de comunicação coletiva são particularmente importantes em algoritmos paralelos numéricos onde vetores longos de dados são particionados e distribuídos pelas memórias locais dos processos que estão executando o programa. Nestes algoritmos, as operações coletivas são usadas para distribuir, juntar, trocar dados, produzir operações globais nos dados e sincronizar o fluxo do programa [BAL 94]. Também estão presentes na manipulação de base de dados, algoritmos de grafos paralelos e algoritmos de busca.

De fato seu uso freqüente levou à padronização de sua sintaxe e semântica no MPI [SNI 96]. O MPI é um padrão de uma biblioteca de passagem de mensagens portável e eficiente. Esta biblioteca tem sido amplamente aceitada na comunidade de computação paralela desde a apresentação do padrão inicial em 1993 e sua formalização no foro MPI.

O objetivo deste trabalho é otimizar operações de comunicação coletiva e analisar sua influenza sobre o tempo de execução de programas de aplicação. A plataforma usada está constituída por uma rede Ethernet 10 Mbps baseada em hub com 8 Dual Pentium Pro 200 Mhz rodando Linux Red Hat 4.3 e utilizando a biblioteca de passagem de mensagens MPICH 1.1. Os 8 nós utilizam placa de rede Surecom, a qual realiza a entrada e saída de dados do nó de processamento pelo barramento ISA. Estes 8 nós estão interligados com o hub Surecom 517T. Neste artigo apresentam-se soluções otimizadas em software para várias operações de comunicação coletiva (*allgather*, *allreduce* e sincronização de barreira), realiza-se um detalhado análise do desempenho das operações individualmente e seu comportamento no programa de aplicação PETSc, e por último compara-se seu tempo de execução com outros aglomerados.

II. ALGORITMOS DE COMUNICAÇÃO COLETIVA IMPLEMENTADOS EM MPICH 1.1.

A implementação das operações de comunicação coletiva no MPICH versão 1.1 está baseada nas seguintes primitivas "ponto a ponto": MPI_Send, MPI_Recv, MPI_Isend, MPI_Wait, MPI_SendRecv. MPICH implementa três mecanismos de troca de dados (estes mecanismos trabalham acima da camada de protocolos TCP/IP e portanto contribuem no *overhead* de envio e recebimento de dados). MPI_Send é uma função de envio de dados de tipo "bloqueante". No MPICH, isto quer dizer que a função somente retorna quando o processador fonte esteja livre para utilizar de novo o *buffer* de envio. MPI_Recv é uma função de recebimento de dados de tipo "bloqueante". No MPICH, isto significa que a função retorna somente depois que o *buffer* de recebimento contenha a mensagem esperada. MPI_SendRecv é uma função que combina, em uma chamada, o envio de uma mensagem a um destino e o recebimento de outra mensagem desde uma fonte. A fonte e o destino podem ser os mesmos. As funções de envio e recebimento são de tipo "bloqueante". MPI_Isend é uma função de envio de dados "não-bloqueante". Esta função retorna imediatamente e a fonte não pode ter acesso ao *buffer* de envio até que seja executada uma função que indique que a operação foi completada. A função MPI_Wait é usada para completar envios "não-bloqueantes". Esta função retorna quando o envio foi completado e indica que o *buffer* pode ser usado.

O algoritmo usado para sincronização de barreira (MPI_Barrier) é o algoritmo distribuído *butterfly* [BRO 86] (chamado também de "*swap*" ou "*shuffle exchange*"). Estes algoritmos estão baseados nas primitivas de comunicação MPI_Send, MPI_Recv e MPI_Sendrecv, as quais transmitem e recebem mensagens de tamanho 0. O algoritmo utilizado para *Allgather* (MPI_Allgather) é um algoritmo em anel. Este anel e percorrido $N - 1$ vezes por cada nó. O algoritmo baseia-se na primitiva de comunicação MPI_Sendrecv. O algoritmo usado para *Allreduce* (MPI_Allreduce) é o algoritmo "torneio" [HEN 88] implementado por uma árvore *combining* na primeira fase e uma árvore binomial na última fase.

III. DESCRIÇÃO DAS OTIMIZAÇÕES EM SOFTWARE

As operações coletivas otimizadas são *Allreduce*, *Allgather* e sincronização de barreira. A operação coletiva *Allgather* é a que maior número de dados manipula. As otimizações realizadas baseiam-se em comparações entre algoritmos que teoricamente oferecem o melhor comportamento. Neste trabalho tem-se adotado algoritmos diferentes dependendo do tamanho de dados e do número de nós que participam. As otimizações feitas correspondem a operações coletivas com tamanho de mensagem pequeno (até 1kbyte) porque na plataforma usada são as que pior desempenho apresentam.

Para tamanho de dados muito pequenos adota-se o algoritmo *butterfly* para 4 e 8 nós participando. Os algoritmos mostrados na figura 1 são os algoritmos otimizados para 5, 6 e 7 nós, os quais apresentam o mesmo mecanismo que o algoritmo *butterfly*. De outra maneira, adota-se o algoritmo "torneio".

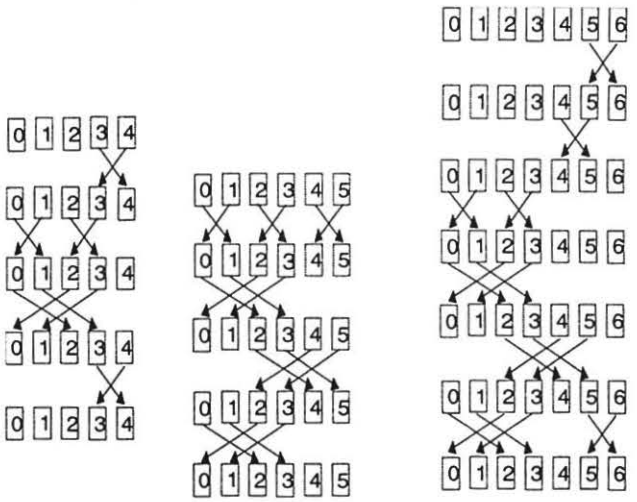

Fig.1 Algoritmos otimizados para 5, 6 e 7 nós

A operação *Allgather* usa o algoritmo *butterfly* e os algoritmos da figura 1 para dados de tamanho até de 7 bytes. Para tamanho de dados entre 8 e 512 bytes utiliza-se o algoritmo "torneio" e para dados maiores de 512 bytes o algoritmo original de MPICH 1.1.

A operação *Allreduce* utiliza o algoritmo *butterfly* e os algoritmos da figura 1 para dados de tamanho até de 40 bytes. Para tamanho de dados maiores, adota-se o algoritmo original de MPICH 1.1 que corresponde ao algoritmo "torneio".

A operação sincronização de barreira utiliza o algoritmo *butterfly* e os algoritmos da figura 1 já que o tamanho das mensagens que manipula esta operação é de tamanho 0.

No algoritmo *butterfly* e os algoritmos semelhantes ao *butterfly* (figura 1), a contenção é gerada na rede mas não nos nós já que a troca de dados faz-se em grupos de dois nós em cada etapa. Ao considerar dados de tamanho muito pequeno, resulta mais vantajoso explorar o paralelismo na biblioteca de passagem de mensagem e deixar que a placa de rede resolva o problema de contenção do hub, aproveitando melhor a largura de banda da rede. Para dados de tamanho maiores, resulta mais difícil otimizar a largura de banda da rede levando a uma perda de desempenho destes algoritmos. Nesta situação, é melhor adotar um algoritmo como "torneio" o qual em sua primeira fase diminui o número de nós que pedem acesso à rede ao mesmo tempo, diminuindo a contenção; isto permite maior aproveitamento da largura de banda da rede. Se o tamanho da mensagem da operação coletiva continua aumentando, maior será o tamanho da mensagem que o algoritmo

"torneio" deve transmitir na segunda fase de sua execução; portanto, seu desempenho degrada-se e por esta razão muda-se o algoritmo para a operação *Allgather* com tamanho de mensagem maior que 512 bytes. Nestes casos, adota-se o algoritmo original de MPICH cuja característica fundamental é que na rede sempre circulam mensagens do mesmo tamanho .

IV. DESCRIÇÃO DAS TÉCNICAS DE MEDIDA DE DESEMPENHO

Infelizmente não existe um padrão com o qual se possa medir o tempo de execução de uma operação coletiva, geralmente cada autor adota sua própria metodologia.

Para medir o tempo de execução destas operações usou-se o procedimento mostrado na figura 2, em cada um dos processos que participam da operação coletiva. A função medida() consiste em acumular em 50 vezes o tempo de execução da operação; este tempo acumulado foi medido k vezes (nossos experimentos sempre usam k = 30) das quais escolhe-se o tempo menor, para diminuir a influenza de outros processos rodando na máquina. Assim resulta *p* medidas (onde *p* é o número de participantes), das quais escolhe-se aquela que possui o tempo maior porque mostra qual foi o último processo em terminar a operação coletiva. Este método é o mais usado pêlos diferentes autores para este tipo de operação coletiva. As três primeiras comunicações não foram consideradas para não incluir na medida o tempo de estabelecimento dos canais de comunicação.

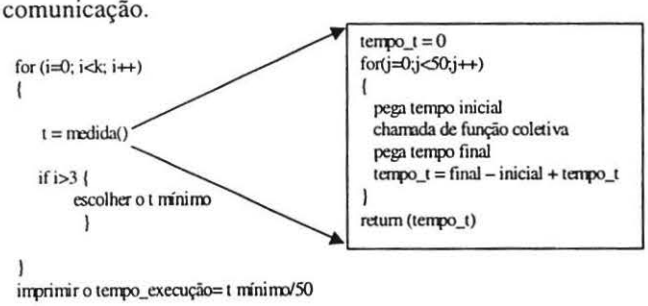

Fig.2 Procedimento para medir o tempo de execução das operações coletivas

Todos os programas de teste foram compilados com o compilador C GNU com a opção de otimização –O. A biblioteca MPICH foi configurada para trabalhar com o dispositivo ch_p4, usou-se a configuração *default* menos a opção –nodevdebug para excluir a sobrecarga de tempo por depuração. As medidas de tempo foram coletadas usando MPI_Wtime, a qual é implementada encima da chamada do sistema ***gettimeofday*** do UNIX.

V. DESCRIÇÃO DO PROGRAMA PETSC

Para medir a vantagem que representa a otimização das operações coletivas usa-se o programa de aplicação PETSc (*Portable, Extensible, Toolkit for Scientific Application*), o qual é um conjunto poderoso de ferramentas para a solução numérica de equações diferenciais parciais e problemas relacionados. Esta aplicação usa o padrão MPI para a comunicação de passagem de mensagem.

PETSc consiste de uma variedade de componentes (similares às classes em C++) onde cada componente manipula uma família particular de objetos (por exemplo, vetores) e realiza operações sobre estes objetos. Algumas destas componentes mexem com vetores, matrizes (esparsas e densas), arranjos distribuídos (úteis para paralelização de problemas baseados em grades regulares), precondicionadores, soluções não lineares, minimização não restrita, solução de equações lineais e não lineais dependentes do tempo, e dispositivos gráficos. Este programa foi desenvolvido pelo Argonne National Laboratory e a versão disponível atualmente é PETSC 2.0.21, a qual foi usada nesta análise de desempenho. Como PETSc está dividido em componentes, cada componente possui um conjunto de programas exemplo que testam seu funcionamento; estes programas de teste foram utilizados para medir seu tempo de execução.

A figura 3 mostra o comportamento de alguns programas PETSc. Cada programa exemplo está representado por duas gráficas, a gráfica da direita mostra quais operações coletivas foram usadas, sua quantidade e quantos nós participam (parâmetro *size*); a gráfica da esquerda proporciona informação sobre o tamanho das mensagens produzidas por estas operações coletivas.

O programa de teste ***ao-ex1*** faz parte da componente "ao" (estas iniciais referem-se Ordenamento de Aplicações); esta componente serve para resolver equações diferenciais parciais em paralelo, especificamente resolve o problema causado por ter varias formas de ordenar e numerar objetos tal como vértices e graus de liberdade, permitindo trabalhar de maneira limpa e eficiente com uma ou mais numerações no ambiente paralelo. Os programas de teste ***da-ex9*** e ***sda-ex1*** correspondem às componentes "da" e "sda" respetivamente; as iniciais referem-se Arranjos Distribuídos. Estes módulos são úteis na paralelização de problemas baseados em grades regulares. Um arranjo distribuído não é um matriz senão um arranjo de números de uma, duas ou três dimensões distribuído através dos nós de tal maneira que cada nó possui um arranjo retangular.

A característica principal dos programas analisados com PETSc é que a comunicação entre nós somente é realizada através de operações coletivas. Observando a figura 3 pode-se concluir que a operação coletiva mais usada nestes exemplos foi *Allreduce*. Além disso, observa-se que a maioria de tráfico produzido corresponde a mensagens curtos principalmente inteiros.

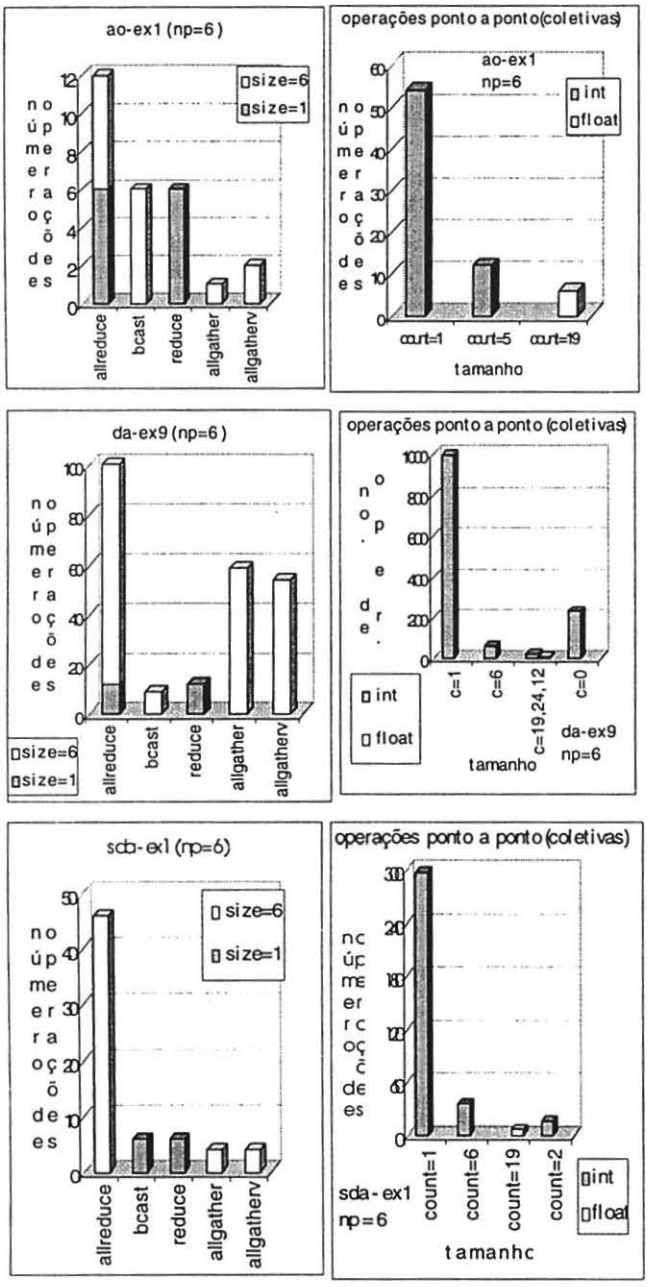

Fig.3 Características dos Programas de aplicação

VI. RESULTADOS E ANÁLISE

Para corroborar a efetividade das otimizações realizadas nas operações de comunicação coletiva primeiro comparam-se os tempos de execução individuais destas operações e depois avalia-se o desempenho de algumas aplicações usando estas otimizações.

Nas tabelas Ori refere-se ao algoritmo original e Oti ao algoritmo otimizado. Como se mostra na tabela I, os tempos de execução da operação coletiva barreira original (ori na tabela) são críticos quando o número de nós participando são 5, 6 e 7. Na tabela II mostra-se a comparação dos tempos de execução entre os algoritmos originais e os algoritmos otimizados da operação *Allgather*. Na tabela III mostra-se a comparação dos tempos de execução entre os algoritmos originais e os algoritmos otimizados correspondentes ao *Allreduce*.

TABELA I
Tempo de execução de Sincronização de Barreira

Número de Nós	MPI_Barrier (ms)	
	Ori	Oti
4	2.0	2.0
5	17.2	2.4
6	29.2	3.8
7	80.4	5.1
8	5.7	5.7

TABELA II
Tempo de execução de *Allgather*

Número de Nós	MPI_Allgather (ms)/bytes					
	4 bytes		512 bytes		1024 bytes	
	Ori	Oti	Ori	Oti	Ori	Oti
4	11.2	1.9	11.2	7.0	12.8	12.8
5	15.3	3.2	15.3	13.6	25.8	25.8
6	19.4	4.5	22.4	19.4	54	54
7	23.2	5.7	33.3	23.3	71.2	71.2
8	27.2	7.2	49.4	27.3	76	76

TABELA III
Tempo de execução de *Allreduce*

Número de Nós	MPI_Allreduce (ms)/bytes					
	4 bytes		32 bytes		512 bytes	
	Ori	Oti	Ori	Oti	Ori	Oti
4	150	1.9	2.5	2.1	5.7	5.7
5	90.5	3.2	4.6	3.4	6.2	6.2
6	90.1	4.5	9.4	5.0	7.6	7.6
7	80.7	5.7	17	6.3	9.8	9.8
8	156.9	8.7	86.7	7.6	11.3	11.3

A seguir tenta-se quantificar o efeito da otimização das operações coletivas sobre o tempo de execução total de um programa de aplicação determinado. A figura 4 apresenta a comparação de desempenho para programas exemplos de PETSc. Os tempos de execução mostram-se mediante uma relação entre o tempo de execução maior e o tempo de execução menor.

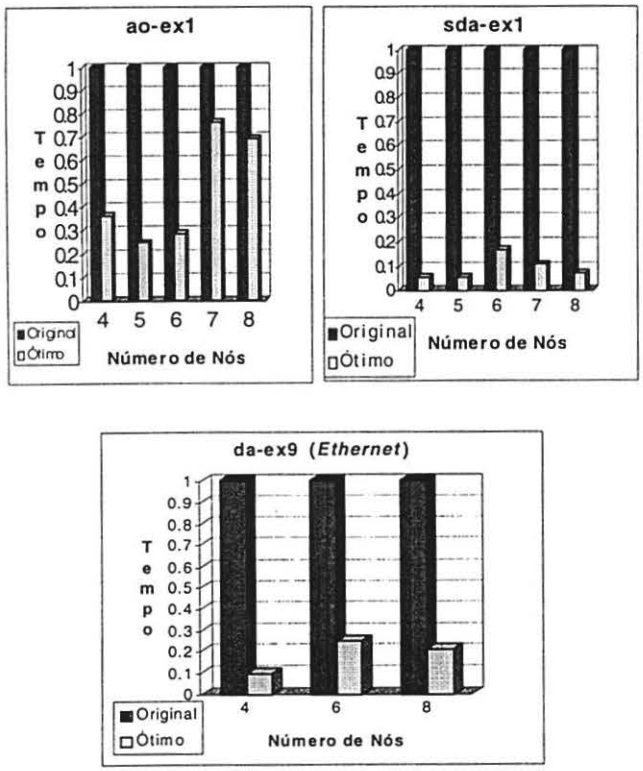

Fig.4 Comparação de PETSc

O programa ***ao-ex1*** apresenta um menor número de operações coletivas, a quantidade de operações *broadcast*, redução, *Allreduce* de tamanho 1 e *Allgatherv* é maior do que as operações *Allreduce* de tamanho 6 e *Allgather*, as quais são otimizadas. O peso das operações coletivas não otimizadas mostra-se mais quando se roda o programa com 7 e 8 nós, embora o ganho de desempenho continue sendo significativo nestas situações.

O programa ***sda-ex1*** apresenta uma proporção menor das operações *broadcast*, redução e *Allgatherv* com respeito às operações *Allreduce* e *Allgather* as quais são otimizadas. Portanto, o ganho de desempenho é sempre muito superior.

No programa ***da-ex9***, o número de operações *broadcast*, redução e *Allgatherv* é menor do que o número das operações *Allreduce* e *Allgather*. Portanto, o tempo de execução usando as operações coletivas otimizadas é sempre menor. Entretanto, sua proporção com a do programa anterior é menor e por isso este último apresenta um melhor desempenho.

VII. Comparação com Dados da Literatura

Antes de passar às comparações é necessário salientar que devido ao fato de ter diversas plataformas resulta difícil fazer comparações justas entre elas. Por isso, em muitos dos dados publicados também se proporciona o tempo que leva uma mensagem ser enviada desde um nó e ser recebida por outro; este tempo é conhecido em inglês como tempo "*ping-pong*", "*round trip*" ou "*end to end*". Este tempo leva em consideração a latência da rede de interconexão e a sobrecarga de tempo dos protocolos de envio de mensagem usados nos diferentes aglomerados. Na tabela IV mostram-se os tempos ministrados por alguns dos aglomerados selecionados para comparação.

Em todas as tabelas mostradas nestas comparações, o aglomerado usado corresponde ao Dual Pentium Pro/200 usando Red Hat linux 4.3.

TABELA IV
Comparação dos tempos *ping-pong*

Plataforma	Biblioteca passagem mensagens	ping pong
Sun Sparc 10, SUN OS 4.1.3 [WAN 96]	Mpich/p4	100 bytes: 3.1 ms 1 kbyte: 4.9 ms
WS com 40/50 MIPS [SUN 94]	PVM3	0 byte: 1.2 ms 1 kbyte: 3.2 ms
Dual Pentium Pro/200, Red Hat linux 4.3	MPICH/p4 1.1	32 bytes: 0.631 ms 1k byte: 2.4 ms

Uma vez que tenta-se estabelecer uma relação entre as diferentes medidas encontradas na literatura para poder fazer comparações entre elas e os aglomerados adotados neste trabalho tem-se escolhido vários níveis de comparação.

No primeiro nível de comparação, mostrado na tabela V, tem-se escolhido somente os aglomerados que usaram como biblioteca de passagem de mensagens a implementação de MPI chamada de MPICH [GRO 96] juntamente com a rede Ethernet 10 Mbps. Em [WAN 97] foi empregada a mesma metodologia de medida usada neste trabalho.

TABELA V
Comparação de desempenho da operação coletiva MPI_Allgather

Plataforma	Versão MPICH/p 4	MPI_Allgather (ms)/bytes			
		4	8	32	512
4 SUN-SPARC 10 [WAN 97]	1.0.11	6.9	7.2	7.3	9.6
4 DUAL Pentium Pro 200Mhz; linux 4.3	1.1	1.9	2.1	4.6	25.3

Na tabela VI mostra-se o segundo nível de comparação usando diferentes bibliotecas de passagem de mensagens. A tabela VI (a) compara a operação coletiva sincronização de barreira para 4 e 8 nós. O algoritmo de barreira usado em [TAN 96] é o *butterfly* e não se descreve a metodologia usada. Em [SUN 94] o algoritmo de barreira é o correspondente à biblioteca PVM de 1994 e não se tem detalhes sobre a metodologia de medida adotada. Embora não se possa estabelecer uma comparação direta devido ao desconhecimento dos métodos de medida, pode-se concluir

que para plataformas antigas [SUN 94] os tempos de execução são mais críticos. A tabela VI (b) compara a operação coletiva *Allgather* em [WAN 97], sendo que o algoritmo utilizado é o original da biblioteca LAM 6.0 que consiste em realizar primeiro uma operação *gather* e logo realizar uma operação *broadcast* para enviar a informação para todos os nós. A metodologia de medida usada é a mesma aqui.

TABELA VI

(a) Comparação de desempenho das operações Barreira

Plataforma	Biblioteca passagem mensagens	No. de nós	Barreira (ms)
Sun SPARCstation-20. [TAN 96]	PVM	4	2.8
		8	5.7
WS com 40/50 MIPS; [SUN 94]	PVM3	4	10.5
		8	28.1
DUAL Pentium Pro 200Mhz; Linux 4.3;	1.1	4	1.9
		8	5.7

(b) Comparação de desempenho das operações *Allgather*

Plataforma	Versão MPICH/p4	Allgather (ms)/bytes			
		4	8	32	128
4 DUAL Pentium Pro 100Mhz; Linux 4.3	1.1	1.9	2.1	4.6	7
4 SUN SPARCstation 10 [WAN 97]	LAM 6.0	3.7	3.7	4.2	5.0

VIII. CONCLUSÕES

Neste trabalho realizaram-se otimizações ao nível de software de três operações coletivas: *Allgather*, *Allreduce* e sincronização de barreira. Os resultados obtidos demostram que é fundamental implementar operações de comunicação coletiva eficientes para ganhar desempenho. A maioria dos artigos escritos sobre este tópico não realizam comparações com programas de aplicação, o qual é muito importante para corroborar a efetividade das otimizações.

Além de analisar soluções ótimas ao nível de software, devem-se analisar soluções que aproveitem a capacidade *multicast* da rede e soluções de hardware especial. Também, desenvolver soluções para aglomerados baseados em comutadores. Estes tópicos correspondem a trabalhos futuros.

AGRADECIMENTOS

Os autores agradecem a CNPq, FINEP e ao Laboratório de Sistemas Integráveis (LSI) da Universidade de São Paulo (USP) pelo suporte econômico e de infra-estrutura.

REFERÊNCIAS

[BAL 94] BALA, V. et al. CCL: a portable and tunable collective communication library for scalable parallel computers. Proceedings of 8th International Parallel Processing Symposium IEEE 1994, p.835-44.

[BRO 86] BROOKS, E. D. The butterfly barrier. *International Journal of Parallel Programming*, v.15, n.4, p.295-307, 1986.

[GRO 96] GROPP, W. et. al. A high-performance implementation of the MPI message passing interface standard. *Parallel Computing*. v. 22, p. 789-828, 1996.

[HUA 94] HUANG, Ch. McKINLEY, P.K. Design and implementations of global reduction operations across ATM networks. Proceedings of Third international symposium of high-performance distributed computing. San francisco, california, 1994, p.43-50.

[HUA 96] HUANG, Y, McKINLEY, P. K. Efficient Collective Operations with ATM network interface suport. Procedings of the 1996 International conference on parallel processing. Bloomingdale, Illinois, USA. August 1996, Vol. I, pp. 34-43.

[HEN 88] HENGSEN, D. et al. Two algorithms for barrier synchronization. *International Journal of Parallel Programming*, v.17, n.1, p.1-17, 1988.

[LAU 97] LAURIA, M., CHIEN, A. MPI-FM: high performance MPI on workstation clusters. *Journal of Parallel and Distributed Computing*, v.40, n.5, p. 431-52, Jan. 1997.

[SNI 96] SNIR, M. et al. *MPI: the complete reference*. London: The MIT Press, 1996

[SUN 94] SUNDERAM, V.S. et al. The PVM concurrent computing systems:Evolution, experiences, and trends. *Parallel Computing*. v.20, n.4, p. 531-545, april 1994.

[TAN 96] TANAKA, Y. A comparison of collective communication performance on parallel algorithms. 1996. Real World Computing Partnership parallel and distributed system performanc laboratory.

[WAN 96] WANG, X. BLUM E. K. Parallel execution of iterative computations on workstation clusters. *Journal of Parallel and Distributed Computing*, v.34, p.218-226, 1996.

[WAN 97] WANG, X. et al. The dance party problem and its application to collective communication in computer networks. *Parallel Computing*, 23, p.1141-1156, 1997.