Comunicação Coletiva no Ambiente de Programação Paralela DECK

Rafael Ennes Silva[1]
Rafael Bohrer Avila[2]
Marcos Ennes Barreto[3]
Philippe Olivier Alexandre Navaux[4]

Resumo

Comunicação Coletiva é muito utilizada em aplicações paralelas, nas quais é necessário a manipulação de matrizes e vetores de dados por grupos de processos. Este artigo apresenta a implementação e a avaliação de desempenho do serviço de comunicação coletiva projetado para o ambiente de programação paralela DECK.

1 Introdução

A concepção de processamento paralelo é dividir tarefas grandes e complexas em pequenas tarefas que são distribuídas e executadas ao mesmo tempo por vários processadores em ordem para conseguir bom desempenho e minimizar o tempo de execução.

As operações de comunicação podem ser divididas em dois tipos dependendo de quantos processos estão participando. Se a comunicação envolve um único processo fonte e um único destino, ela é chamada de ponto-a-ponto; por outro lado, se a comunicação envolve mais de um processo fonte e/ou destino, esse tipo é chamado comunicação em grupo.

DECK(*Distributed Execution and Communication Kernel*) (BARRETO, 2000) (BARRETO; NAVAUX; RIVIÈRE, 1998) é um ambiente para aplicações paralelas e distribuídas que dispõe de mecanismos básicos para *multithreading* e comunicação ponto-a-ponto, e também serviços especiais como comunicação coletiva. O modelo de programção paralela utilizado no DECK é baseado na passagem de mensagens, o qual caracteriza-se pela presença de muitos processadores, cada um com a sua própria memória(memória distribuída)(BUYYA, 1999).

[1]resilva@inf.ufrgs.br Bolsista Projeto apoiado pela DELL-LabTeC e CNPQ
[2]avila@inf.ufrgs.br Doutorando UFRGS/INPG (Grenoble, França)
[3]barreto@inf.ufrgs.br Doutorando UFRGS - Professor UNILASALLE
[4]navaux@inf.ufrgs.br PhD INPG(Grenoble, França) - Professor UFRGS

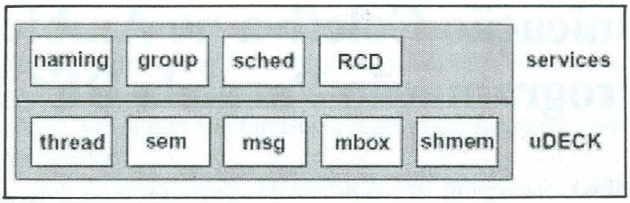

Figura 1: Estrutura do DECK

O objetivo deste trabalho é aperfeiçoar os algoritmos de comunicação coletiva já desenvolvidos para o DECK (CASSALI, 2000) , para conseguir melhorar o desempenho e disponibilizar outras primitivas para manipulação de dados numéricos. Essas primitivas são equivalentes às encontradas na biblioteca MPI(*Message Passing Interface*) (PACHECO, 1997) (FORUM, 1994)para que seja facilitada a utilização do serviço de comunicação coletiva do DECK por usuários de outros ambientes como MPI e PVM(*Parallel Virtual Machine*) (GEIST, 1994).

2 O Ambiente de Programação DECK

DECK - é um ambiente de programação com a finalidade de computação de alto desempenho. É composto por uma API que oferece um conjunto de abstrações e serviços especializados que são necessários para aplicações distribuídas. Uma aplicação DECK utiliza o modelo SPMD, no qual é feita uma cópia do processo em cada nodo utilizado do cluster.

O DECK é estruturado em duas camadas: uDECK e a camada de serviços conforme Figura 1. A uDECK interage diretamente com a camada de hardware, sistema operacional e protocolos de comunicação para prover recursos de comunicação, sincronização e *multithreading*. Portanto essa camada gerencia as seguintes abstrações: *threads, semaphores, messages, mailboxes* e *shared segments*. A camada de serviço dispõe um conjunto mais especializado de recursos, tais como comunicação em grupo, a qual é utilizada por muitas aplicações.

3 Comunicação Coletiva no DECK

As primitivas utilizadas para comunicação coletiva no DECK são: *Barrier*, *Broadcast*, *Gather*, *Scatter*, *Reduce*, *Allgather*, e *Allreduce*.

Algumas dessas primitivas, tais como *barrier, broadcast, allgather* e *allreduce*, utilizam o algoritmo de árvore binária, no qual o nodo raiz é responsável por enviar dados para os seus filhos, e então cada filho repassa a mensagem para seus filhos e assim por diante, até os dados chegarem nos nodos folhas (último nível da hierarquia de árvore).

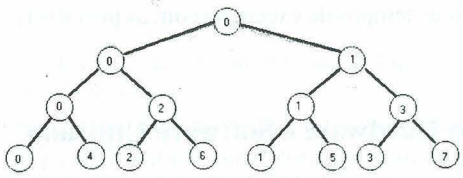

Figura 2: Árvore de configuração de processos MPI

Anteriormente, as primitivas do DECK recebiam *buffers* em seus parâmetros, os quais tinham que ser encapsulados em mensagens para depois serem enviados para os processos destinos. Nessa remodelagem das primitivas, elas passaram a receber diretamente mensagens como parâmetros, ou seja, o usuário encapsula os seus dados antes de chamar a função, o que minimiza o *overhead* gerado pelo número de encapsulamentos de *buffers*, que eram realizados internamente nas funções, conforme o número de processos do grupo.

Para as primitivas *scatter* e *gather* foi criado um mecanismo de submensagem para que fosse possível subdividir ou concatenar as mensagens conforme a funcionalidade de cada função respectivamente. Na primitiva *reduce* foram modificados os parâmetros (*buffer* para mensagem), e internamente as operações permaneceram as mesmas, porém com mensagens.

4 Características de Implementação do MPI

Na biblioteca do MPI existem diferenças com relação a implementação de algumas primitivas tais como: *broadcast, reduce, allgather* e *allreduce*. A árvore percorrida é um pouco diferente como mostra a Figura 2.

Durante o primeiro estágio o processo raiz envia os dados para o processo 1, no estágio seguinte o processo raiz envia para o processo 2 enquanto o processo 1 envia para o processo 3. E no últimmo estágio o processo raiz envia para o processo 4, enquanto o o 1 envia para o 5, 2 envia para o 6 e o 3 envia para o 7. Esse algoritmo é utilizado para a primitiva *broadcast*. Para as primitivas *reduce, allreduce* e *allgather* o MPI usa o algoritmo denominado *Butterfly*, no qual para uma operação, no caso do *allreduce* por exemplo, os resultados das operações ocorrem entre pares processos até que todos os membros do grupo obtenham o resultado final[PAC97].

5 Análise de Desempenho

Para avaliar o serviço de comunicação coletiva desenvolvido para o DECK, foram realizados testes com as primitivas citadas na seção 3, com exceção da *broadcast, allreduce* e

allgather, comparando seus tempos de execução com as primitivas similares da biblioteca do MPI.

5.1 Plataforma de Hardware e Software Utilizado

As simulações foram feitas no LabTeC DELL/UFRGS(Laboratório de Tecnologia em Clusters). As características do cluster são: um servidor com 2 processadores Xeon 1.8GHz e 1GB de memória RAM; 20 nodos com Pentium III Dual 1.13GHz e 1GB de memória RAM, e sistema operacional Debian Linux com kernel 2.4.

5.2 Simulações

Foram desenvolvidas pares equivalentes de aplicações, uma com primitivas MPI e a outra com primitivas DECK, para realizar a comparação de desempenho. Cada simulação foi realizada em torno de 100 vezes para garantir melhor precisão. O número de nodos foi incrementado de 2 a 20, e foram escolhidos 2 tamanhos de mensagens (100 e 10K bytes) para analisar o comportamento da aplicação.

Avaliação da Scatter e da Gather

A Figura 3 ilustra os resultados de uma sequência de *scatter-gather* com mensagem de 100 bytes considerando a variação do número de nodos. É possível observar que o comportamento de ambas bibliotecas é muito similar, com uma pequena vantagem para o DECK.

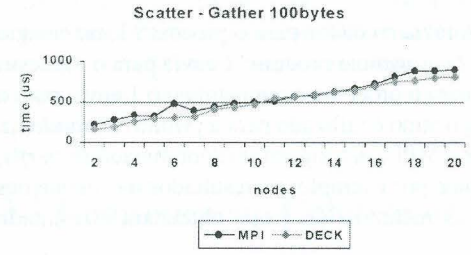

Figura 3: *scatter-gather* - 100bytes

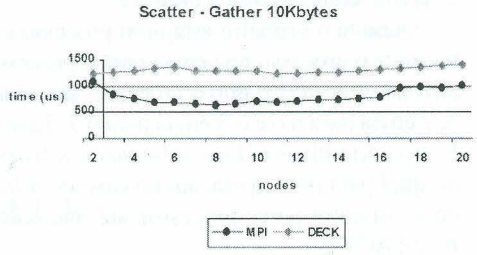

Figura 4: *scatter-gather* - 10K bytes

A vantagem obtida pelo DECK começa a cair para mensagens maiores que 10K bytes, como mostra a Figura 4. O algoritmo utilizado no MPI apresenta melhor desempenho do que o utilizado no DECK.

Avaliação da Reduce

Para mensagens de 100 bytes, o resultado obtido está na Figura 5. O DECK novamente conseguiu melhor desempenho que o MPI.

A Figura 6 mostra a resposta da operação *reduce* para mensagem de 10K bytes. O desempenho apresentado pelo DECK é bastante satisfatório comparado ao MPI. Este comportamento pode ser consequência das características de implementação, como no DECK a operação só é realizada uma vez pelo processo *root* após a coleta dos dados dos demais membros; enquanto que no MPI o algoritmo *Butterfly* é utilizado para realizar operações paralelas, ou seja, *reduces* parciais.

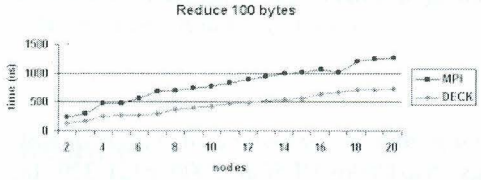

Figura 5: *Reduce* - 100bytes

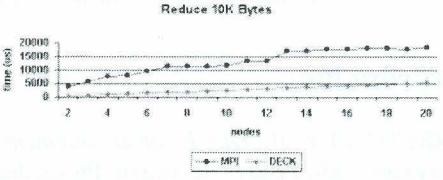

Figura 6: *Reduce* - 10K bytes

Avaliação da Barrier

A última medida feita foi a da primitiva *barrier*. Na Figura 7 estão os resultados obtidos. O comportamento do DECK é similar ao MPI para um número menor que 6 processos. Depois disso, uma pequena desvantagem para o DECK é observada como consequência da implementação.

Quando um processo chama uma primitiva *deck_collective_barrier()*, ele envia para o seu pai uma mensagem vazia. O pai realiza um *barrier* local para receber mensagens dos seus filhos; e após, passa adiante a mensagem vazia para o pai dele até a mensagem chegar ao processo *root*. Depois disso o *root* está livre para desbloquear os demais membros do grupo.

Para as outras primitivas, não foram apresentadas análises de desempenho devido a que ainda não foram suficientemente testadas.

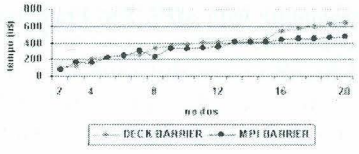

Figura 7: Tempos de execução para a *barrier*

6 Conclusões Finais

Comunicação em grupo é muito utilizada em uma grande quantidade de aplicações paralelas que precisam manipular dados numéricos. As bibliotecas do PVM e do MPI são os maiores exemplos de ambientes de programação que fornecem primitivas de comunicação coletiva e são utilizadas por muitos pesquisadores para o desenvolvimento de aplicações.

O objetivo desse trabalho foi avaliar o serviço de comunicação coletiva proposto para o ambiente DECK, através de uma comparação com as funções do MPI.

Para as operações com *reduce*, o DECK obteve melhores respostas do que o MPI. Para a *barrier* e operações de *scatter/gather*, esta simulação nos mostra que as primitivas do DECK precisam ser revisadas quando o tamanho de mensagem é maior que 10K bytes.

Referências

BARRETO, M. et al. *DECK: an environment for parallel programming on clusters of multiprocessors*. São Pedro - SP-Brazil. Proceedings...São Carlos:UFSCar, 2000, p321-329: In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 2000.

BARRETO, M. E. et al. Deck: a new model for a distributed executive kernel integrating communication and multiheading for support of distributed object oriented application with fault tolerance support. In: *Congreso Argentino de Ciencias de la Computacion*. Neuquém, Argentina: Neuquém: Universidad Nacional de Comahue: Facultad de Economia y Administración: Departamento de Informática y Estadística, 1998. v. 2, p. 623–637.

BUYYA, R. (Ed.). *High Performance Cluster Computing: Achitecture and Systems*. Upper Saddle River: Prentice Hall PTR, 1999.

CASSALI, R. et al. Group communication service for deck. In: . TEC de Monterrey. México: CONFERENCIA LATINOAMERICANA DE INFORMATICA, 2000.

FORUM, M. *The MPI message passing interface standard*. [S.l.], 1994.

GEIST, A. *PVM - Parallel Virtual Machine*. Cambridge: MIT Press, 1994.

PACHECO, P. *Parallel Programming with MPI*. São Francisco, Califórnia: Morgan kaufmann Publishers, Inc, 1997.