Avaliação de Desempenho de Implementações Portáteis de PVM para Plataformas de Multiprocessamento Simétrico com Suporte a *Multithreading*

Cláudio M. P. Santos*
e-mail: claudio@nce.ufrj.br

Júlio S. Aude**
e-mail: salek@nce.ufrj.br

*Universidade Federal do Rio de Janeiro - COPPE e NCE
** Universidade Federal do Rio de Janeiro - IM e NCE

1. Introdução

O PM-PVM (*Portable Multithreaded* PVM), inicialmente chamado de M-PVM [Sant97], é uma implementação portátil de PVM para uso em ambientes de multiprocessamento simétrico, que mapeia *tasks* PVM em *threads* e implementa as funções de troca de mensagem do PVM usando os recursos de memória compartilhada entre *threads* disponíveis nessas arquiteturas.

A portabilidade do PM-PVM deriva do fato de que toda a funcionalidade do PVM é implementada a partir de uma conjunto mínimo de primitivas de programação paralela que permitem a criação, a extinção e a coleta de informações sobre *threads* bem como operações de sincronização por exclusão mútua e por ordem parcial. Esse conjunto mínimo de funções é o mesmo suportado pelo Mulplix [Azev93], um sistema operacional do tipo Unix projetado para executar de modo eficiente os aplicativos paralelos na plataforma Multiplus [Aude96] de memória compartilhada distribuída. Dessa forma, para efetivar o transporte do PM-PVM para diferentes plataformas, basta implementar esse conjunto reduzido de funções na plataforma de processamento paralelo desejada. Na realidade, este exercício de transporte já foi realizado e utilizado neste trabalho para *Solaris* LWPs, *Solaris threads* [Sun95] e P*threads* [IEEE94].

2. O Modelo PM-PVM

O ambiente de programação PM-PVM (*Portable Multithreaded* PVM) foi idealizado para ser um ambiente portátil de programação paralela PVM-like. A motivação para o seu desenvolvimento foi prover um ambiente de programação conhecido dos usuários e, portanto, de fácil assimilação, de forma a permitir que diversas aplicações paralelas fossem transportadas ou desenvolvidas eficientemente para diferentes plataformas baseadas nos conceitos de memória compartilhada e *multithreading*.

O ambiente PM-PVM foi construído através da implementação das funções do ambiente PVM com o uso de um conjunto reduzido de primitivas de programação paralela derivadas do sistema operacional Mulplix. As *tasks* PM-PVM, em vez de serem mapeadas em processos, são mapeadas em *threads*. Uma aplicação PM-PVM é constituída de um único processo, formado por *threads* (*tasks* PM-PVM) que executam em paralelo, compartilham os recursos do processo e se comunicam segundo o modelo de troca de mensagens, implementado através de memória compartilhada.

No PM-PVM, a figura do *pvmd* foi eliminada, pois a aplicação passa a estar delimitada a um único processo. Não existe comunicação entre *tasks* de processos distintos, somente dentro do mesmo processo. O processo de troca de mensagens passa a ser feito unicamente através da biblioteca PM-PVM.

Outra característica importante do PM-PVM é a existência de variáveis globais do processo, que oferecem um mecanismo adicional para troca de informações entre as *tasks*, além do processo de troca de mensagens. Mesmo um usuário PVM pode tomar proveito desta característica do PM-PVM, sem a necessidade de se aprofundar nos conhecimentos de programação por memória compartilhada.

3. Implementações do Modelo PM-PVM

A biblioteca PM-PVM foi implementada através da utilização de uma estrutura global contendo informações sobre as *tasks* em execução. Cada *task*, logo após a sua criação, é cadastrada nesta estrutura utilizando o seu *tid* para identificar a entrada que deve ocupar.

O PM-PVM possui, além do vetor de controle de *tasks*, uma outra importante estrutura chamada tabela de grupos. A tabela de grupos contém informações sobre os grupos criados dentro de uma aplicação PM-PVM. O acesso a cada uma de suas entradas é feito através do nome do grupo, utilizando-se a técnica de *hashing*.

O PM-PVM foi desenvolvido e testado em uma *Sparcstation 20* com 4 processadores. O conjunto reduzido de funções básicas do modelo de programação paralela do sistema Mulplix, utilizado na implementação do modelo PM-PVM, foi implementado em três ambientes: Solaris LWP's, Solaris *Threads* e P*threads*.

Em função da análise dos resultados dos programas de teste de desempenho, foram utilizadas três abordagens distintas na implementação do PM-PVM. As três implementações resultantes possuem a mesma interface, podendo ser utilizadas pelos programas de usuário sem nenhuma alteração de código.

A primeira abordagem, denominada PM-PVM1, implementa o processo de troca de mensagens na forma tradicionalmente empregada nos sistemas que possuem memória compartilhada. A mensagem é formada por uma lista de fragmentos e seu conteúdo é compartilhado entre as *tasks* para onde é enviada. Não há replicação de mensagens. A mensagem possui um campo adicional contendo o número de referências a ela. Este campo indica quando a mensagem pode ser realmente descartada, isto é, quando não há mais nenhuma referência pendente. Na implementação do PM-PVM1, cada *task* possui no seu vetor de controle um vetor de *buffers* e um ponteiro para a fila de mensagens recebidas. Cada *buffer*, quando ocupado, armazena um ponteiro para a mensagem associada a ele.

A segunda abordagem, denominada PM-PVM2, foi concebida de modo a otimizar o processo de troca de mensagens do PM-PVM1. No PM-PVM2, os fragmentos de dados são armazenados em um único vetor. Este vetor é alocado dinamicamente de acordo com o tamanho do primeiro fragmento a ser empacotado. O tamanho inicial é suficiente para armazenar o novo fragmento, sendo múltiplo de 256 *bytes*. Espera-se que, ao alocar um tamanho maior que o necessário, o número de alocações adicionais seja reduzido.

Foi mudado também o processo de envio/recepção de novas mensagens. De modo a eliminar o contador de referências, a mensagem é copiada durante o processo de envio, sendo armazenada na fila de mensagens recebidas da *task* destino.

A terceira abordagem, denominada PM-PVM3, foi derivada da combinação das duas abordagens anteriores, buscando aproveitar os seus aspectos mais positivos. O conteúdo da mensagem é compartilhado entre as *tasks* para onde é enviada, como no PM-PVM1, mas a mensagem é constituída de um único bloco. Ao invés de se utilizar uma lista de fragmentos como no PM-PVM1, os fragmentos de dados são armazenados em um vetor de *bytes* como no PM-PVM2. Dessa forma, no PM-PVM3, foram eliminados pontos de sincronização existentes

no PM-PVM1, pela não utilização da lista de fragmentos e foi evitado o *overhead* de replicação de mensagem existente na implementação das operações de envio e recepção de mensagens no PM-PVM2.

4. Análise Experimental e Conclusões

Resultados experimentais foram obtidos em uma *SparcStation 20* com 4 processadores e 128 Mbytes de memória.

Os testes foram realizados considerando duas implementações distintas das primitivas de programação paralela do Mulplix sobre as quais se baseiam as implementações do PM-PVM. A primeira implementação utilizada foi a feita para a plataforma *Solaris* LWPs e a segunda foi a desenvolvida para *Solaris threads* (*user level threads*).

A versão do PVM utilizada nos experimentos foi a 3.3.11 configurada para o ambiente *Solaris*/Sun. A comunicação entre as *tasks* PVM é feita, nesta versão, através de *streams* (pipe bidirecional). Cada *task* PVM é mapeada em um processo Unix.

Foram feitas medidas para comparar o desempenho do PM-PVM versus PVM em suas funções básicas. Os testes realizados foram divididos em categorias de modo a enfocar os aspectos mais importantes nos dois ambientes. São eles: criação de *tasks*, empacotamento, desempacotamento, envio e recepção de mensagens.

Foram realizados também testes de desempenho com quatro aplicativos: SOR, *Bubble Sort*, Eliminação Gaussiana com cálculo de determinante e Algoritmo Genético para *Placement*.

Os resultados dos experimentos de avaliação de desempenho das implementações das operações básicas de manipulação de *threads* e de sincronização do ambiente Mulplix mostraram que a versão com *Solaris threads* tem um desempenho superior à versão com LWPs na maioria dos casos. Além disso, a implementação com *Solaris threads* não requer proteção explícita na manipulação de áreas de memória dinâmica e realiza a troca de contexto de *threads* de forma mais rápida.

Os resultados obtidos nos experimentos de avaliação de desempenho das operações básicas do ambiente PVM indicam que, em geral, pelo menos umas das versões do PM-PVM tem um desempenho superior ao PVM. O PVM apresentou um fraco desempenho na criação de *tasks* e no processo de envio e recepção de mensagens. No empacotamento ou desempacotamento de múltiplos fragmentos, o seu desempenho foi comparável ao das versões do PM-PVM para LWPs, mas foi inferior ao das versões PM-PVM para *Solaris threads*.

O PM-PVM1 tem o pior desempenho no empacotamento de múltiplos fragmentos devido à alocação de memória dinâmica. O PM-PVM3 e o PM-PVM2, por sua vez, conseguem bons resultados nesse teste. Em todos os testes, o PM-PVM3 apresenta um desempenho igual ou superior ao PM-PVM1. O PM-PVM2, por sua vez, é pior no envio de mensagens, principalmente quando estas possuem grandes quantidades de dados, pois quanto maior a quantidade de informações, maior será o custo de cópia.

A análise dos resultados de desempenho obtidos com aplicativos é mais complexa do que a dos testes básicos, pois eles envolvem um conjunto de operações diferentes e a interação entre elas. Nas aplicações com algum tempo ocioso, devido, por exemplo, à sincronização, as versões do PM-PVM com *Solaris threads* tendem a produzir melhores resultados, pois a troca de contexto é mais rápida. Este fato é observado nos algoritmos SOR e eliminação gaussiana.

O PM-PVM2 parece produzir melhores resultados nas aplicações com mensagens pequenas. No algoritmo de eliminação gaussiana, pode-se ver que, quando ocorre um aumento do

número de *tasks*, o desempenho do PM-PVM2 piora, pois, no início do algoritmo, onde as mensagens tem um tamanho maior, também ocorre um aumento no número de mensagens enviadas. Por outro lado, no algoritmo genético, o desempenho do PM-PVM2 é muito bom em relação aos demais, apesar de parte das mensagens terem um tamanho razoável. Neste caso, o PM-PVM3 perde com o pouco compartilhamento das mensagens, pois cada mensagem enviada é diferente, sendo logo descartada. Isto acarreta um custo adicional devido à liberação da mensagem e a uma nova alocação para armazenar a nova mensagem. Além disso, o custo de manutenção do contador de referências não compensa o seu uso. Já no PM-PVM2, as áreas de memória utilizadas são reaproveitadas nas futuras mensagens. Caso o tamanho das mensagens aumentasse com o transcorrer do algoritmo, o PM-PVM2 possivelmente teria que realocar memória, o que acarretaria uma perda de desempenho.

O PM-PVM3 tende a apresentar melhores resultados que o PM-PVM2 nas aplicações onde o compartilhamento de mensagens é efetivo, principalmente quando operações de *broadcasting* são utilizadas. Ao contrário do que ocorre com PM-PVM2, o aumento do tamanho da mensagem não acarreta nenhum prejuízo ao seu envio, o que representa uma outra vantagem.

5. Trabalhos Futuros

A principio existem algumas pequenas modificações na implementação do PM-PVM3 que podem vir a melhorar seu desempenho. Exemplos são a redução do *overhead* na manutenção da fila de mensagens recebidas e a adoção de um mecanismo misto de envio de mensagens de forma a permitir a convivência de mensagens compartilhadas e mensagens copiadas em um mesmo modelo. O programador poderia, então, escolher, através de rotinas diferentes, o modo como a mensagem seria enviada.

6. Agradecimentos

Os autores gostariam de agradecer o apoio dado pela FINEP, FAPERJ, CNPq, CNPq/RHAE e CAPES/COFECUB ao desenvolvimento desse trabalho de pesquisa.

7. Bibliografia

[Aude96] "The Multiplus/Mulplix Parallel Processing Environment". Aude, J.S., et al., *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks* - I-SPAN 96, pp. 50-56, Beijing, China, Maio 1996

[Azev93] "Primitivas de Programação Paralela no MULTIPLUS", Azevedo, R.P., Azevedo, G.P., Silveira, J.T.C, Aude, J.S., *V SBAC-PAD*, Florianópolis, pp. 761-775, Setembro 1993

[Geis94] *PVM - A users guide and tutorial for Network Parallel Computing*, Geist A., et al., The MIT Press, Cambridge, Massachusetts, 1994.

[IEEE94] *Threads Extension for Portable Operating Systems*, Posix P1003.4a, IEEE, 1994

[Knop96] "Parallelization of Genetic Algorithms Applied to the Placement Problem in Workstation Clusters", Knopman, J., Aude, J.S., *VIII SBAC-PAD*, Recife, PE, Agosto, 1996.

[Sant97] "M-PVM: A Multithreaded PVM for Shared-Memory Architectures", Santos, C. M. P., Aude, J. S., *Proceedings of the 9th International Conference on Parallel and Distributed Computing and Systems*, Washington, DC, USA, Outubro, 1997.

[Sun95] *Multithreaded Programming Guide - Solaris 2.5*, SunSoft, Inc, 1995.