Comparativo entre Diferentes Interfaces de Comunicação para Programação Paralela*

Diego Luis Kreutz, Márcia Cristina Cera, Marcelo Pasin
Programa de Pós-Graduação em Engenharia de Produção - Tecnologia de Informação
Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria
{kreutz, cera, pasin}@inf.ufsm.br

Rodrigo da Rosa Righi
Programa de Pós-Graduação em Ciência da Computação
Universidade Federal do Rio Grande do Sul
rrrighi@inf.ufrgs.br

Resumo

As aplicações desenvolvidas para ambientes de programação paralela e distribuída possuem seu desempenho diretamente relacionado ao da interface de comunicação utilizada para viabilizar sua execução. Uma vez que há a necessidade de comunicação entre os nós processadores que compõem o sistema, esta tarefa acaba, muitas vezes, sendo mais onerosa que o próprio tempo de processamento da aplicação paralela. Considerando tal aspecto, neste estudo foram realizadas comparações entre diferentes interfaces de comunicação, a fim de analisar a influência do uso destas no desempenho de uma aplicação específica.

1 Introdução

Assim como ocorre a evolução das arquiteturas de computadores, ocorre também a evolução e complexidade dos problemas a serem resolvidos. Percebe-se isso através da crescente evolução de arquiteturas, como os aglomerados de computadores, e a demanda emergente de aplicações que necessitam de alto poder de processamento.

Aplicações para sistemas distribuídos têm, normalmente, entre suas características, a necessidade de um sistema de compartilhamento de informações bastante eficiente e, ainda, a necessidade de uma quantidade elevada de ciclos de processador. Para suprir os requisitos de aplicações deste gênero surgiram arquiteturas de alto desempenho como os aglomerados de computadores.

*Financiamento CAPES/CNPq: processos 181589/01-8 e 380049/03-1

Com o objetivo de explorar, da melhor forma possível, o potencial das arquiteturas para processamento de alto desempenho, foram desenvolvidos ambientes e bibliotecas de programação como MPI (*Message Passing Interface*) [9, 10], PVM (*Parallel Virtual Machine*) [14], libVIP (*Virtual Interface Protocol*) [4] e DECK (*Distributed Execution and Communication Kernel*) [12]. Estes ambientes fornecem recursos como diretivas de troca de mensagens e controle e alocação de processos. Isto facilita o trabalho dos desenvolvedores, uma vez que precisam preocupar-se, basicamente, com a resolução do problema ao implementarem uma aplicação paralela ou distribuída.

O presente artigo tem por objetivo avaliar o desempenho entre diferentes interfaces de comunicação para uma mesma aplicação. Foram adotadas as duas bibliotecas tradicionais, MPI e PVM, em suas versões para TCP/IP, e a biblioteca VIP, que é baseada na Arquitetura de Interface Virtual [13]. Além destas três bibliotecas, foi analisado o desempenho da aplicação implementada diretamente sobre a interface de *sockets*. Neste caso, foi necessário implementar todo o sistema de comunicação e lançamento de processos.

O artigo está dividido como segue. A segunda seção descreve a interface de *sockets*, as bibliotecas utilizadas para a avaliação de desempenho e, também, os modelos de execu-ção para as suas versões. A terceira seção apresenta a aplica-ção desenvolvida. A quarta seção apresenta os testes rea-lizados e a avaliação de desempenho, comparando o número de processdores utilizados, o tamanho das mensagens e o tempo de retorno da aplicação. Na quinta seção, situa-se a conclusão do artigo, que mostra as situações onde cada biblioteca obteve melhor desempenho e define modificações na pesquisa a cargo de trabalhos futuros. Para finali-

zar, são relacionadas as referências utilizadas para o desenvolvimento deste trabalho.

2. As Interfaces de Comunicação

Esta seção apresenta a descrição de quatro diferentes interfaces de comunicação utilizadas para a implementação da aplicação do cálculo do fractal de Mandelbrot. Foram construídas quatro versões desta aplicação, cada qual com uma interface de comunicação diferente. A primeira versão implementada utiliza a interface de *sockets* para viabilizar a comunicação. Esta interface de programação é considerada de baixo nível, pois realiza a interação direta com o sistema operacional. Também foi implementada a versão com a interface MPI, que é, provavelmente, a biblioteca de comunicação mais difundida na área de programação paralela. Outra interface de comunicação utilizada na análise efetuada foi a biblioteca PVM, que é igualmente bastante difundida no meio acadêmico. Para finalizar, têm-se uma versão utilizando a interface VIA (*Virtual Interface Architecture*), através da biblioteca VIP, a qual foi desenvolvida para oferecer troca de mensagens totalmente assíncronas.

As subseções que seguem descrevem as principais características das interfaces de programação utilizadas.

2.1 A Interface de *Sockets*

A interface entre os programas e os protocolos de comunicação em um sistema operacional acontece através de uma API (*Aplication Programming Interface*) [6]. A API de *sockets* é um padrão para comunicação entre computadores e está disponível para muitos sistemas operacionais, como Windows NT e 98, e em vários sistemas da família Unix, como Linux, Solaris, AIX, etc.

Sockets foram originalmente desenvolvidos como parte do sistema operacional BSD Unix e empregam muitos conceitos deste. Mais especificamente, *sockets* são integrados com o sistema de E/S (entrada e saída). Quando um aplicativo abre um arquivo ou dispositivo, a chamada `open()` retorna um descritor, que é um número inteiro que identifica o arquivo. A aplicação deve informar este descritor como um dos argumentos do procedimento para realizar uma função de transferência de dados (`read()` ou `write()`).

Para se programar com *sockets* é necessário especificar diversos parâmetros, como o protocolo de transporte, o endereço de uma máquina remota, se o aplicativo é um cliente ou um servidor, entre outros. Primeiramente, um programa cria um *socket*, que é um descritor de E/S, e após, invoca funções para trabalhar com ele.

Para acontecer uma conexão TCP/IP [6, 1] com *sockets*, uma máquina deve ficar esperando por uma conexão em uma deterninada porta, através da chamada à função `accept()`. Uma outra máquina, por sua vez, para estabelecer a comunicação executa o procedimento `connect()` na mesma porta.

A figura 1 (a) apresenta um esquema da aplicação desenvolvida neste estudo utilizando *sockets* como interface de comunicação. Analisando a figura pode-se visualizar que o processo mestre distribui e coordena o trabalho entre os escravos. No processo mestre, tem-se um vetor com todos os *sockets* onde os escravos estão conectados. Já, um processo escravo possui apenas um *socket*, que está ligado ao mestre e permite a interação entre eles.

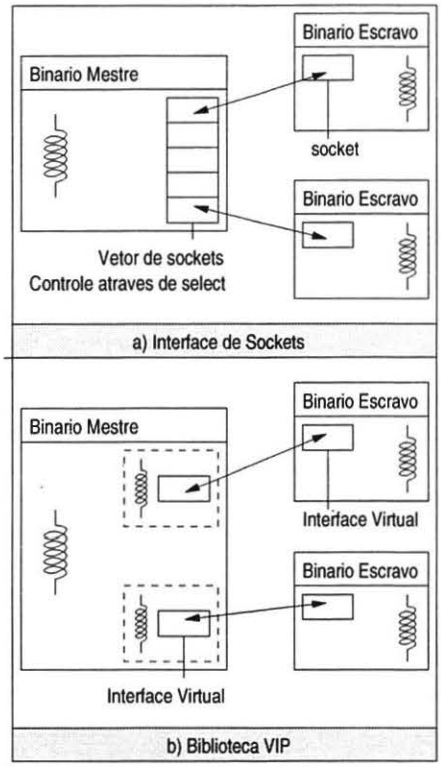

Figura 1. Arquiteturas: (a) Interface de Sockets e (b) Biblioteca VIP

2.2 A Biblioteca VIP

A biblioteca VIP [4], ou libVIP, é uma biblioteca de comunicação capaz de proporcionar comunicação assíncrona entre os processos que compõem um programa paralelo. Para tanto, a libVIP implementa um subconjunto de doze procedimentos descritos na especificação da Arquitetura de Interface Virtual (VIA) [3, 13]. A libVIP foi implementada com o sistema operacional Linux e utiliza a biblioteca *Pthreads*, para o controle de fluxos de execução, e a interface de *sockets* TCP, para realizar a comunicação entre computadores. Com a utilização do protocolo TCP, a biblioteca

VIP consegue comunicar dois computadores quaisquer que operem com o este protocolo e não há a necessidade de um módulo específico VIA no sistema operacional.

Na arquitetura VIA, uma interface virtual (VI) [3] é análoga a um *socket* para conexão tradicional TCP. Cada VI suporta transferência de dados bi-direcional e ponto-a-ponto [7]. Uma interface virtual é um ponto final de comunicação e é composta de um par de filas de trabalho, uma para envio e outra para requisição de dados. As requisições para troca de mensagens são realizadas através da adição de descritores nas fila de uma interface virtual. A libVIP foi implementada com duas modalidades de comunicação, a libVIP alterada e a normal. A libVIP alterada apresenta modificações no *socket* com o intuito de maximizar o desempenho na troca de mensagens. Já, a libVIP normal não possui alterações no descritor de comunicação. Mais detalhes sobre a biblioteca VIP podem ser encontrados em [5].

O *framework* de comunicação com a libVIP para a aplicação desenvolvida está ilustrado na figura 1 (b). No processo que representa o mestre, tem-se um fluxo de execução para cada processo escravo. Desta maneira, cada escravo pode receber e transmitir dados diretamente de e para o processo mestre.

2.3 A Biblioteca PVM

O PVM [14] é uma biblioteca que proporciona uma infraestrutura de *software* que emula uma máquina virtual para um sistema distribuído. Essa máquina virtual, pode ser composta de nós processadores com arquitetura heterogênea, ou seja, composta por máquinas de diferentes arquiteturas, trabalhando cooperativamente.

Para a troca de mensagens, PVM oferece a implementação das primitivas básicas de envio e recebimento de dados (`PVM_send` e `PVM_recv`). As mensagens podem ser trocadas com qualquer outro processo PVM sem restrições quanto a sua quantidade e seu tamanho.

O modelo de comunicação de PVM assume que as mensagens podem ter envio e recepção assíncrona e recepção síncrona. Logo, todas as rotinas de envio de dados serão não-bloqueantes, uma vez que os dados estarão armazenados em uma região de memória intermediária (*buffer*) e os processos não aguardam até que a troca de mensagem seja efetuada. Contudo, o recebimento pode ser tanto bloqueante quanto não bloqueante, possibilitando maior flexibilidade na escrita de uma aplicação.

Tem-se na figura 2 (a) a estrutura da versão da aplicação paralela que utiliza PVM como interface de comunicação. Nela pode-se ver o programa mestre que lançará os processos escravos e os coordenará a partir de um vetor de identificadores.

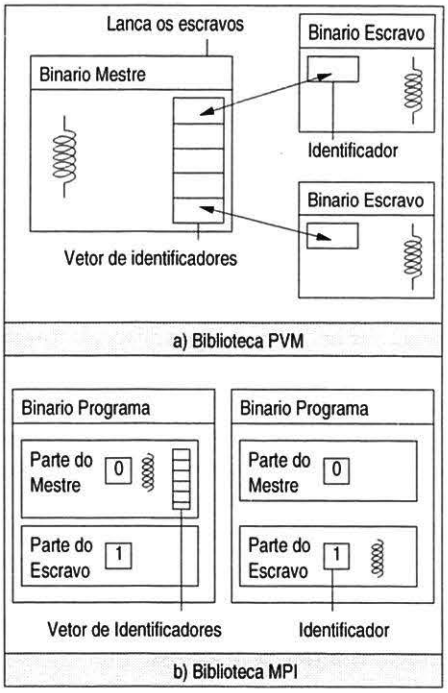

Figura 2. Arquiteturas: (a) Biblioteca PVM e (b) Biblioteca MPI

2.4 A Biblioteca MPI

A MPI [9, 10] é, provavelmente, a biblioteca de comunicação mais utilizada atualmente para o desenvolvimento de aplicações paralelas e distribuídas. É uma biblioteca de funções e macros que pode ser utilizada em aplicações escritas em linguagens de programação como C, FORTRAN e C++. Como seu próprio nome supõe, MPI foi projetada para ser utilizada em programas que exploram a existência de múltiplos processos e, para estabelecer a comunicação entre eles, utiliza-se a técnica de troca de mensagens.

MPI possibilita escrever programas que utilizam o modelo de um único programa para múltiplos dados (SPMD - *Single Program, Multiple Data* [2]). No início da execução de um programa MPI, é lançado um conjunto fixo de processos, criados na inicialização da biblioteca, cada qual destinado a um determinado elemento processador disponível para a execução do programa. Uma cópia do mesmo programa é disparada em todos os nós processadores, sendo executado um trecho específico do programa em cada um deles. Para verificar qual região de código deve ser executada, realiza-se uma seleção baseada no identificador do processo corrente. Portanto, tem-se um mesmo programa para ser executado na máquina mestre e nas demais. O trecho correspondente ao nó mestre será executado por um dos processadores e o restante do código executa nos demais

nós que compõem o sistema distribuído.

Os procedimentos mais utilizados para a troca de mensagem nesta biblioteca são a `MPI_Send` e `MPI_Recv`. Estas diretivas são utilizadas, respectivamente, para o envio e recepção de dados. No total, existem mais de uma centena de procedimentos disponíveis na interface dessa biblioteca. Ao se programar com MPI, as estruturas de dados enviadas em mensagens necessitam ser definidas e registradas.

A implementação da aplicação paralela com a biblioteca MPI pode ser vista na figura 2 (b). Ela apresenta o mesmo programa, com a implementação do mestre e escravo, executando nos nós processadores do sistema. A presença de um identificador determina qual parte do programa será exetutada. O mestre coordena as execuções dos escravos através de um vetor que contém identificadores dos seus escravos. É através destes identificadores que as mensagens chegam aos seus devidos destinos.

Tabela 1. Interfaces de comunicação analisadas e suas primitivas

Interface de Comunicação	Primitivas de comunicação
Sockets	`read()` `write()`

libVIP	`VipPostSend()`
	`VipPostRecv()`
	`VipSendWait()`
	`VipRecvWait()`

PVM	`PVM_send()` `PVM_recv()`
MPI	`MPI_send()` `MPI_recv()`

A tabela 1 apresenta cada uma das interfaces de comunicação utilizadas na implementação da aplicação paralela e suas respectivas primitivas básicas de comunicação.

3 A Aplicação Desenvolvida

O objetivo deste artigo foi realizar um estudo comparativo de desempenho, apontando as vantagens e as características de diferentes interfaces de comunicação utilizadas em aplicações paralelas. Para isso, foi projetado e implementado um programa paralelo capaz de possibilitar uma análise em termos de tempo computacional, associado à eficiência de comunicação e à escalabilidade. O ponto mais crítico no desenvolvimento da aplicação com várias versões de interfaces de comunicação foi alinhar a estrutura dos programas. Visando realizar uma análise de desempenho, cuidados especiais foram tomados para manter o código fonte e estruturas de dados iguais entre todas as versões implementadas.

A aplicação escolhida para tornar possível uma avaliação parametrizada, observando-se aspectos de diferentes bibliotecas de comunicação, tem por base um algoritmo que processa uma matriz de pontos e produz uma imagem do fractal de Mandelbrot. Esse algoritmo particiona uma matriz de 800 por 800 pontos em matrizes menores, e as distribui entre os nós de um aglomerado. O cálculo da cor de cada um dos pontos de todas as matrizes é independente. Logo, pode-se optar por diferentes dimensões para as submatrizes, sem com isso alterar o resultado final do fractal.

A interação entre o processo principal, chamado de mestre ou coordenador, e os demais processos, chamados de escravos, é descrita como segue. Os processos escravos recebem as coordenadas de uma matriz que representa sua tarefa atual. Em posse das coordenadas, calculam a cor para cada um dos pontos da matriz. Após o processamento dos valores de cores, o processo escravo envia uma matriz contendo os resultados ao processo mestre e aguarda uma nova tarefa. Esse processo é repetido até que todas as sub-matrizes, partes da matriz que formam toda a imagem, sejam processadas.

Na aplicação de cálculo do fractal podem ser alterados vários parâmetros. Pode-se variar, por exemplo, o tamanho das sub-matrizes. Tal parâmetro indica o número de comunicações entre os processos escravos e o mestre. Quanto menor é o tamanho da matriz enviada aos escravos, maior é a troca de mensagens destes com o processo mestre. Outro parâmetro alterável é o número de processos escravos. A modificação deste parâmetro pode acarretar no aumento, ou na redução, do tempo de processamento de um escravo e, também, na dificuldade de coordenação e comunicação entre os processos escravos e o mestre. A maleabilidade e diversidade desses parâmetros permite uma análise de vários aspectos das interfaces de comunicação utilizadas.

A métrica utilizada para avaliar as diferentes interfaces de programação foi o tempo de retorno da aplicação do fractal de Mandelbrot. Esta métrica permite a análise de características dos diferentes ambientes de comunicação, como a utilização da largura de banda e da latência, e sobre a sua escalabilidade e adaptabilidade.

3.1 Arquitetura e Estrutura da Aplicação

A arquitetura da aplicação utilizada é baseada na implementação simples mestre-escravo. Cada um dos processos escravos solicita trabalho e o recebe do processo mestre. O escravo retorna o resultado do processamento ao mestre e recebe uma nova tarefa, caso ainda existam tarefas a serem processadas. Se não houverem mais tarefas, o mestre envia uma mensagem dizendo que o escravo pode finalizar a sua execução.

As estruturas e tipos de dados utilizados são iguais e de mesmo tamanho em todas as quatro versões da aplicação.

Um cuidado especial foi aplicado nesse ponto, de forma a não haverem discrepâncias em termos de tamanho e tipos de estruturas de dados, o que poderia levar a uma análise falha entre os valores de desempenho obtidos.

A coleta do tempo de processamento também é realizada exatamente no mesmo ponto em todos os programas. Os laços de repetição e as funções utilizadas foram todos padronizados com o objetivo de evitar diferenças de processamento e possibilitar uma análise de desempenho mais precisa.

3.2 O Ambiente de Execução e Compilação Utilizados

O processo de teste e análise das aplicações foi realizado utilizando o aglomerado de computadores do LSC (Laboratório de Sistemas de Computação da UFSM). Foram empregados na execução dos testes oito nós do aglomerado, sendo que cada nó oferece dois processadores Pentium III de 1GHz, 768 MB de RAM, 20 GB de disco rígido, uma placa de rede 3Com Fast Ethernet (3C905) e uma placa de rede 3Com Gigabit Ethernet (3C996). A placa mãe dos nós é da marca ASUS (CUV4X-D) e possui barramento convencional de 133 MHz de frequência de *clock*. Atualmente, todos os nós rodam a distribuição Linux RedHat 8.0 com o núcleo do sistema operacional GNU/Linux em sua versão 2.4.18-14SMP.

No processo de teste e compilação foram utilizados o GCC 3.2 com padrão Pthreads, a versão LAM/MPI 6.5.6, a versão PVM 4.3.3 e a libVIP alterada 0.0.2, compiladas utilizando a versão do GCC recentemente citada, especificamente para a arquitetura dos elementos processadores do aglomerado.

A comunicação entre os processos das diferentes versões da aplicação desenvolvida foi realizada utilizando as placas Gigabit Ethernet. O comutador que interliga as placas é um 3Com SuperStack 3 4900 modelo 3C17700 de 16 portas.

Essa descrição detalhada do ambiente de execução e compilação teve como objetivo apresentar as características de *hardware* e *software* utilizados para efetuar as medições de desempenho. Além disso, esta descrição possibilita que sejam reproduzidos os testes e torna possível incluir comparações com outras bibliotecas ou ambientes de comunicação. As estruturas e os códigos fonte utilizados nesses testes poderão ser fornecidos a fim de evitar reimplementações desnecessárias.

3.3 Tempo de Execução dos Escravos

A tabela 2 apresenta o tempo médio de processamento dos escravos de cada uma das versões da aplicação. Essa tabela foi gerada a partir dos dados obtidos quando utilizados oito nós do aglomerado e tamanho de mensagens igual a 128 *bytes* (matrizes de ordem 10 por 10). Esses dados servem apenas para ilustrar a semelhança e proximidade entre as diversas implementações.

Tabela 2. Tempo total de execução de 16 escravos utilizando sub-matrizes de 10 por 10

libVIP	MPI	PVM	Sockets
3.635087	3.62288	3.86747	3.754826

Diferenças de tempos de processamento por parte dos escravos se deve, basicamente, a fatores como a troca de contexto e o escalonamento do sistema operacional. Foram lançados dois processos escravos em cada nó do aglomerado, totalizando 16 processos escravos, a fim de tirar proveito dos dois processadores disponíveis em cada nó.

O tempo de processamento de um escravo corresponde ao tempo necessário para realizar os cálculos efetivos da tarefa e preencher, com seus resultados, a estrutura de dados que irá ser enviada ao processo mestre. Esse tempo começa a contar a partir do momento que a mensagem foi recebida e armazenada, sendo computado até o momento em que todos os pontos da sub-matriz tiverem sido calculados e a estrutura de retorno estiver com todos os dados completados.

4 Comparativo entre as Interfaces de Comunicação

Em sistemas computacionais de alto desempenho, como o próprio nome já sugere, busca-se o maior desempenho e eficiência possíveis para a solução de problemas grandes, complexos ou que exijam uma quantidade bastante grande de recursos computacionais. Neste âmbito, procura-se também utilizar e desenvolver ambientes e bibliotecas de comunicação que possibilitem a utilização dos recursos diponíveis da melhor forma possível.

Com o advento e desenvolvimento da computação de alto desempenho começaram a surgir ambientes e bibliotecas de comunicação como MPI, PVM e libVIP. Aqui serão comparadas estas bibliotecas de comunicação e uma implementação da aplicação paralela utilizando somente a interface de *sockets* do sistema operacional.

O objetivo foi apresentar uma análise de desempenho e eficiência de diferentes implementações utilizando sistemas de comunicação variados. Atingiu-se esse objetivo através da implementação do programa paralelo descrito na seção 3.

Os gráficos das figuras 3, 4, 5 e 6 apresentam os resultados obtidos na execução da aplicação paralela variando-se o tamanho da mensagem e o número de elementos processadores. Nessa aplicação, o processo mestre simplesmente

sub-divide a matriz de pontos a ser calculada em matrizes menores. Logo, uma tarefa com lado igual a 10 significa uma matriz quadrada 10 por 10, o que resulta em uma mensagem de pelo menos 100 *bytes*. Além dos pontos da matriz, são enviadas as coordenadas dessa sub-matriz em relação a matriz inteira e alguns parâmetros de cálculo.

Para a efetivação e comparação de desempenho foi utilizada uma matriz de 800 por 800 pontos. Essa foi quebrada em tarefas menores a fim de gerar um fluxo maior ou menor de comunicação. Além disso, foi variado o número de elementos de processamento, a fim de verificar a variação de desempenho conforme variáveis de escalabilidade do sistema. Nesta aplicação, cada processo escravo é mapeado para um elemento processador.

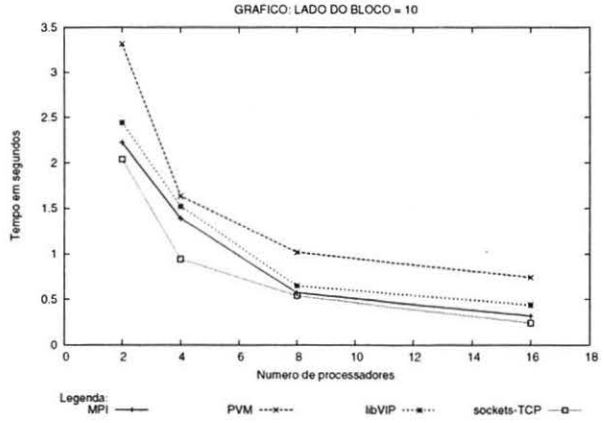

Figura 3. Tempo de execução utilizando mensagens de 128 *bytes*.

O gráfico da figura 3 exibe as variações de desempenho entre as diferentes versões do programa paralelo dividindo-se a matriz inicial em sub-matrizes de tamanho 10 por 10. Isto representa 6400 tarefas a serem processadas, gerando um montante de 12800 comunicações entre o processo mestre e os escravos. As mensagens entre mestre e escravos, neste caso, possuem 128 *bytes*, destes 100 *bytes* correspondem aos dados da sub-matriz e o restante compreende as coordenadas dessa sub-matriz em relação a matriz inteira e alguns parâmetros de cálculo.

Nesse ponto, pode-se avaliar a eficiência e desempenho dos sistemas de comunicação, quando da utilização de mensagens pequenas. Observando-se o gráfico resultante, conclui-se que com a variação em termos de escalabilidade, número de elementos computacionais, as quatro implementações apresentam um comportamento semelhante. A versão mais eficiente é a implementada em *sockets* e a menos eficiente é a que utiliza o PVM. Este resultado era esperado, uma vez que *sockets* é uma solução em mais baixo nível, oferecida pelo próprio sistema operacional. Já PVM, possui uma sobrecarga de início de transmissão de dados significativa, a qual reflete-se nos tempos de execução quando estão envolvidas mensagens de pequeno porte.

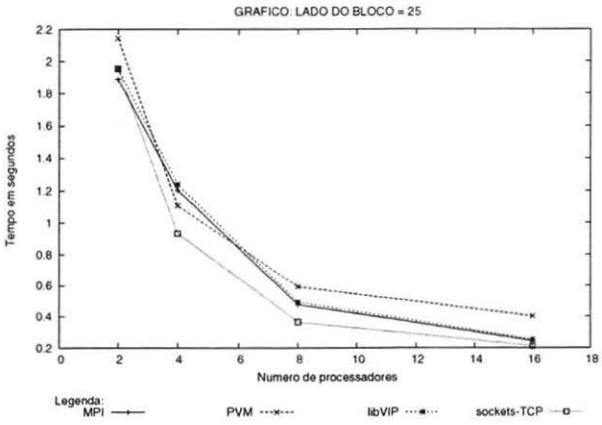

Figura 4. Tempo de execução utilizando mensagens de 653 *bytes*.

Os dados mostrados no gráfico da figura 4 apresentam o comportamento das aplicações com mensagens de 653 *bytes* (matrizes de ordem 25). O resultado é semelhante aò do gráfico anterior. A excessão é a implementação em PVM, que começa a apresentar um desempenho superior às implementações em MPI e libVIP, quando são utilizados quatro processadores. Ressalta-se que nos demais gráficos a versão PVM acaba sendo a melhor de todas as implementações, ao trabalhar com mensagens de tamanho superior a 653 *bytes* e quatro processadores.

Os gráficos das figuras 5 e 6 apresentam os dados referentes a execuções com mensagens de 2528 e 10028 *bytes*, respectivamente. Observando estes gráficos, destaca-se o comportamento da implementação em PVM. As demais implementações demonstram um comportamento semelhante em todos os gráficos, não ocorrendo diferenças de desempenho tão salientes quanto a versão em PVM.

O gráfico da figura 7 apresenta uma comparação de escalabilidade em relação ao tamanho das mensagens. Esses dados foram extraídos a partir dos dados coletados da execução com 16 elementos processadores. Verificou-se que esse comportamento é idêntico para as execução com 2, 4 e 8 unidades de processamento.

Pode-se perceber que o tempo computacional é ligeiramente elevado para o cálculo de blocos de lado igual a 10, ou seja, para mensagens de 128 *bytes*. A eficiência aumenta para as mensagens de 625 *bytes*, ou lado do bloco com 25 pontos. A aplicação apresenta uma eficiência menor com mensagens iguais ou maiores que 2528 *bytes* ou blocos com 50 pontos de lado. Esse comportamento se deve principalmente a sobrecarga (*overhead*), a latência e a utilização da largura de banda da rede. Tal observação pode ser verificada em [11].

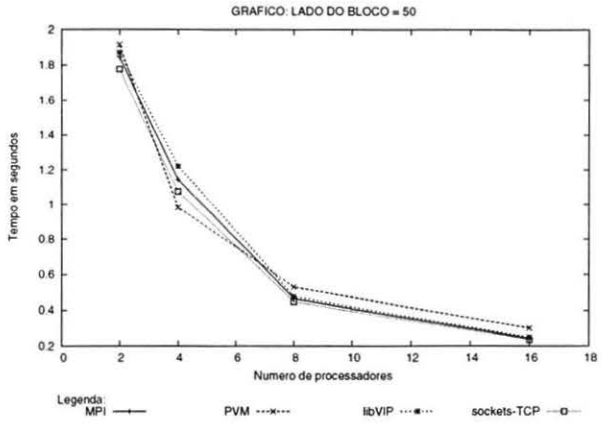

Figura 5. Tempo de execução utilizando mensagens de 2528 *bytes*.

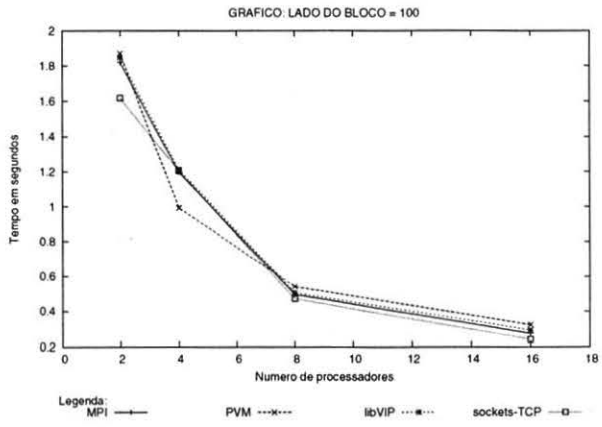

Figura 6. Tempo de execução utilizando mensagens de 10028 *bytes*.

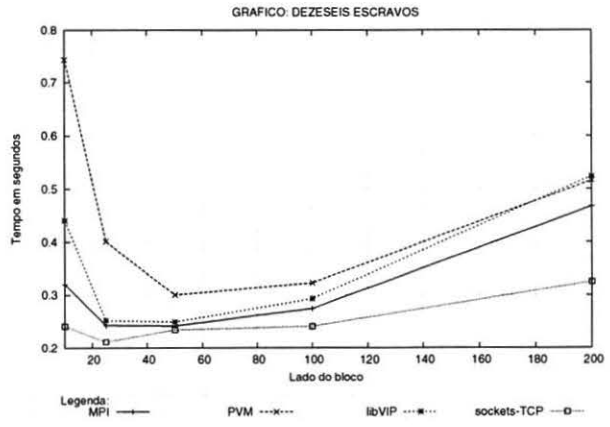

Figura 7. Tempo de execução de 16 elementos processadores para diferentes tamanhos de mensagens.

A partir dos resultados apresentados nos gráficos de desempenho verifica-se que a versão em PVM apresenta um comportamento diferenciado. Seu desempenho supera as demais somente quando são utilizadas mensagens de tamanho acima de 128 *bytes* e com número de elementos processadores entre 2 e 8.

Na maior parte dos casos de testes, a implementação em *sockets* apresentou o melhor desempenho. Em contrapartida, sua implementação foi muito mais complicada que a das demais. A simplicidade e rapidez no processo de desenvolvimento utilizando MPI e PVM, certamente são aspectos que tem um mérito elevado. Além disso, o desempenho da versão em MPI, no geral, é apenas levemente inferior a de *sockets*.

Não é viável a comparação entre os recursos e ferramentas disponibilizadas por ambientes e bibliotecas de comunicação largamente difundidos como MPI e PVM, com *sockets* ou bibliotecas simples como a libVIP. Estas duas últimas possuem apenas funções básicas para envio e recepção de mensagens. Com estas duas, a programação é mais demorada, complexa e menos legível.

Por outro lado, implementações em *sockets* e libVIP também têm seus méritos e vantagens. Diferentemente de MPI e PVM, o envio de estruturas de dados ou dados complexos é bastante simples. Basta fornecer o endereço inicial e o tamanho dos dados. Já em MPI e PVM é necessário utilizar diretivas de empacotamento e composição de mensagens, ou ainda, a definição de novos tipos de dados, no caso de MPI, o que não é uma tarefa muito prática. Neste aspecto, a utilização de *sockets* ou libVIP superam os concorrentes.

5 Conclusão

Sistemas computacionais de alto desempenho estão sendo utilizados e desenvolvidos cada vez mais. Existem conjuntos de aplicações que crescem em escala, complexidade e tamanho na mesma proporção, ou até mais depressa, que as próprias máquinas paralelas. Pesquisadores buscam resolver cada vez mais problemas e simulações mais complexos e precisos, exigindo recursos e poder computacional igualmente crescentes.

Visando fornecer alto desempenho computacional, surgem os ambientes e bibliotecas de comunicação para fornecer ferramentas e recursos que auxiliam no desenvolvimento de aplicações que façam um uso apropriado e eficiente dos recursos computacionais disponíveis. É com esses objetivos que surgiram ferramentais como o PVM, MPI, DECK e libVIP.

Cada um desses ambientes e bibliotecas possui características e peculiaridades próprias. Tais características as tornam mais úteis e eficientes para a solução de determinados tipos de problemas. É nesse ponto que se insere o objetivo deste trabalho, o qual visou apresentar informações

aos usuários e desenvolvedores de aplicações paralelas que podem ser úteis no processo de escolha e definição do ambiente de programação a ser utilizado para um determinado tipo de problema.

Através da análise de desempenho apresentada pode-se facilmente visualizar certas características e comportamentos dos ambientes e bibliotecas analisados. Os resultados permitem concluir que, de um modo geral, o MPI é a melhor opção para o desenvolvimento e implementação de aplicações paralelas. Isso devido a sua boa eficiência, escalabilidade e facilidade de uso. No entanto, em casos específicos, uma opção como PVM, libVIP ou mesmo *sockets* podem fazer uma diferença significativa. Como cada aplicação possui normalmente características e necessidades específicas, não é possível se definir um ambiente ou biblioteca de comunicação como sendo a melhor opção em todos os casos.

Este artigo compara MPI, PVM e libVIP com a interface nativamente oferecida pelo sistema operacional, *Sockets* TCP. É importante comparar ainda outras concorrentes destas bibliotecas, como DECK, libCE (Comunicação Eficiente) [2] e MPICH [8]. Trabalhos futuros incluem análises de desempenho com estas bibliotecas.

Os processos de empacotamento das bibliotecas avaliadas não foram exaustivamente testados. Testes mais completos poderiam serem feitos visando avaliar as vantagens de cada possibilidade de empacotamento por elas oferecidas. Em muitos casos, o empacotamento pode levar a uma solução mais simples.

Tecnicamente, nenhum ponto crítico foi claramente identificado. Pensava-se que os testes efetuados iriam mostrar diferenças mais evidentes, o que não foi verificado. Testes mais exaustivos podem ser feitos, com características levadas a valores mais extremos, no intuito de identificar possíveis pontos críticos das implementações.

Referências

[1] Andrew S. Tanenbaum. *Redes de Computadores*. Editora Campus, Rio de Janeiro , RJ, 3 th edition, 1997.

[2] Márcia Cristina Cera, Daniela Saccol Peranconi, and Marcelo Pasin. Biblioteca ce: Comunicação eficiente. In *Anais da Terceira Escola Regional de Alto Desempenho*, Anais da Escola Regional de Alto Desempenho, pages 173–176, January 2003.

[3] Compaq, Intel, and Microsoft. Virtual Interface Architecture Specification version 1.0, Dec. 1997. `www.cs.cornell.edu/barr/repository/cs614/san\_10.pdf`.

[4] Rodrigo da Rosa Righi. libVIP - Desenvolvimento em Nível de Usuário de uma Biblioteca de Comunicação que Implementa o Protocolo de Interface Virtual. Trabalho de Graduação nº 163. Curso de Ciência da Computação, UFSM. Santa Maria/RS, Fev, 2003.

[5] Rodrigo da Rosa Righi, Marcelo Pasin, and Philippe Olivier Alexandre Navaux. libVIP: Arquitetura de Interface Virtual sobre TCP/IP. In *Quarto Workshop em Sistemas Computacionais de Alto Desempenho*, São Paulo - SP, November 2003. aceito para publicação.

[6] Douglas E. Commer. *Redes de Computadores. Transmissão de dados, ligação inter-redes e Web*. Bookman, Porto Alegre, second edition, 2001.

[7] P. Druschel. Operating Systems Support for Highspeed Networking. P. Druschel. Operating systems support for highspeed networking. Technical Report TR 94-24, Department of Computer Science, University of Arizona, Oct. 1994.

[8] W. Gropp and E. Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):103–114, Summer 1997.

[9] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Massachusetts, USA, 1994.

[10] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, September 1996.

[11] Diego Luís Kreutz and Rodrigo da Rosa Righi. Comparação de Desempenho entre Diferentes Adaptadores de Rede Gigabit Ethernet. In *XVII Congresso Regional de Iniciação Científica em Tecnologia e Engenharia - CRICTE2002*, August 2002.

[12] F. A. D. de OLIVEIRA and et al. DECK-SCI: high-performance communication and multithreading for sci clusters. In *IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING*, Newport Beach, CA, 2001.

[13] Evan Speight, Hazim Abdel-Shafi, and John K. Bennett. Realizing the Performance Potential of the Virtual Interface Architecture. In *International Conference on Supercomputing*, pages 184–192, 1999.

[14] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.