ALDRI LUIZ DOS SANTOS

AVALIAÇÃO DE DESEMPENHO DA COMUNICAÇÃO COM PVM EM AMBIENTE LINUX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Roberto A. Hexsel

CURITIBA
1999

Ministério da Educação
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE CIÊNCIAS EXATAS

PARECER

Nós, abaixo assinados, membros da Comissão Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Aldri Luiz dos Santos, avaliamos o trabalho intitulado **"Avaliação de Desempenho da Comunicação com PVM em Ambiente Linux"**, cuja defesa foi realizada no dia 20 de maio de 1999. Após a Avaliação, decidimos pela Aprovação do Candidato.

Curitiba, 20 de maio de 1999.

Prof. Dr. Roberto André Hexsel
Presidente

Prof. Dr. Douglas Paulo Bertrand Renaux

Prof. Dr. Elias Procópio Duarte Júnior

AGRADECIMENTOS

À amiga Célia Man pelo seu apoio e carinho nos momentos mais decisivos da minha vida.

Ao meu orientador prof. Roberto André Hexsel por ter confiado em mim e me ensinado a ser um verdadeiro pesquisador.

Ao prof. Alexandre Direne pelo seu apoio em todos os momentos e por ter me possibilitado escrever esta dissertação em LaTeX.

À prof. Olga Regina P. Bellon pelo seu grande trabalho como Coordenadora do Curso de Mestrado.

Ao prof. Elias Procópio Duarte Jr. por suas palavras de incentivos e de objetividade.

A todos os professores do Departamento de Informática que me apoiaram e confiaram em mim.

Ao amigo João Fábio que me ensinou os primeiros passos no Linux.

Ao amigo Claudio Matsuoka pelas nossas conversas no Laboratório do Mestrado e por ter me ensinado a instalar e configurar o Linux nas diversas máquinas que eu utilizei para desenvolver a minha dissertação.

Aos amigos Ewerton e Edilza pelas palavras de apoio e por me ajudarem nos momentos que foram necessários.

Ao amigo Andrey por ter me emprestado a sua dissertação como referência e ter sempre uma palavra de incentivo.

Ao amigo Denilson pela sua visão objetiva e por sempre me apoiar.

Aos amigos Ana, Paula, Elisa, Razer e Gilberto os quais eu considero como meus irmãos.

Aos amigos Tatiana, Nélis, Ruda, Diógenes, Alessandro, Albini, Luciane, Inali e Josiane pelo apoio e incentivo.

A todos que acreditaram em mim.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho avalia o desempenho do sistema de comunicação disponibilizado pelo "Parallel Virtual Machine" (PVM) sobre o Linux. Esta avaliação tem a finalidade de demonstrar a viabilidade de se implementar processamento paralelo distribuído através do PVM em ambientes de baixo custo. O processamento paralelo distribuído proporciona uma melhor utilização dos recursos computacionais desses ambientes, e por conseqüência um aumento do poder computacional. Assim, tarefas que exijam maior poder de computação, que demandam muito tempo de processamento ou são impossibilitadas de serem executadas por uma única máquina, poderiam ser executadas nos horários em que não houvessem usuários utilizando os computadores.

A avaliação do sistema de comunicação do PVM consiste na medição e discussão das vazões obtidas no sistema de memória, na comunicação via soquetes utilizando os protocolos de transporte TCP e UDP, e na comunicação via PVM nos ambientes estudados.

Uma descrição do mecanismo de comunicação inter-processos para aplicações distribuídas e da tecnologia Ethernet é apresentada. Uma revisão sobre a classificação das arquiteturas de computadores, com ênfase na conceituação de máquinas paralelas, é mostrada. Algumas propriedades básicas de programação paralela são descritas. Descreve-se também os componentes do PVM e a maneira como ocorre a comunicação entre as tarefas de uma aplicação PVM.

As medições efetuadas da vazão nos sistemas de memória, na comunicação via soquetes, e na comunicação via PVM são mostradas e comparadas. De acordo com os dados obtidos sobre a comunicação com PVM, o seu uso é viável em ambientes de baixo custo. Medições indicam que a comunicação com PVM é 28% menos eficiente do que a comunicação TCP e/ou UDP para as mensagens de 1 Kbyte, e 16% menos eficiente para as mensagens de 32 Kbytes (UDP). Outro resultado deste estudo é um conjunto de recomendações ao programador sobre como conseguir o máximo de desempenho na comunicação via PVM.

ABSTRACT

This work presents a performance evaluation of the communication system supported by "Parallel Virtual Machine" (PVM) over Linux. This study purports evaluation to demonstrate the viability of distributed parallel processing implementation through PVM in low cost computing environments. Distributed parallel processing may provide a better use of the computational features in these environments and, as a consequence, an increase of computational power. Thus, tasks that demand great computational power, long processing time or that are impossible to be executed in only one machine, could be scheduled to be executed when the computers are not being used.

The evaluation of PVM communication system consists of measurement of throughput in the memory system, communication via sockets using the TCP and UDP transport protocols, and in the communication via PVM in the studied environments.

A description of the inter-processes communication mechanism for distributed applications and Ethernet technology is presented. A revision of computer architecture classification, with emphasis in the conceptualization of parallel machines, is presented. Some basic properties of parallel programming are discussed. PVM components and the way communication is implemented in PVM are also described.

Measurements made on throughput in the memory systems, in the sockets communication, and in the PVM communication are shown and compared in detail. The data from the experiments show that communication via PVM is aproximately 28% less efficient than using TCP or UDP sockets with 1 Kbyte messages. For 32 Kbytes messages the performance loss with PVM is close to 16%. The results also indicate what set of parameters is most influential in achieving the best possible performance. A set of recommendations is presented so that a programmer might tune his/her application code to be able to achieve high performance in communication via PVM.

CAPÍTULO 1

INTRODUÇÃO

Este trabalho visa demonstrar a viabilidade de processamento paralelo distribuído em ambiente de baixo custo. As aplicações científicas e tecnológicas da computação cada vez mais necessitam de grande poder de processamento. Entretanto, o custo para se implantar um ambiente de trabalho capaz de suportar aplicações de alto desempenho é relativamente alto e demanda tempo e recursos humanos altamente especializados.

Atualmente, a maioria dos ambientes de computação de baixo custo é composto por redes locais Ethernet, onde são conectadas estações de trabalho e computadores do tipo PC. É possível aproveitar o poder computacional desses ambientes, nos horários em que não hajam usuários utilizando os computadores, para executar tarefas que exijam maior poder de computação, que demandam muito tempo de processamento ou são impossibilitadas de serem executadas por uma única máquina. O processamento paralelo distribuído tem o potencial de possibilitar uma melhor utilização dos recursos do ambiente, e consequentemente que se aproveite melhor o poder de computação.

O "Parallel Virtual Machine" (PVM) é uma sistema baseado em passagem de mensagens cujo objetivo é permitir o aproveitamento dos recursos computacionais de uma rede [GBD+94a]. O PVM foi desenvolvido numa parceria entre o Laboratório Nacional de Oak Ridge e as Universidades do Tennessee, Emory e Carnegie Mellon, e possibilita que computadores com arquiteturas diferentes e que fazem parte de uma rede, sejam considerados como processadores de uma grande máquina paralela. Numa máquina paralela, parte do tempo de processamento é gasto com computação, tempo utilizado no processamento dos dados, e parte é gasto com comunicação, tempo utilizado na transmissão e recepção das mensagens para comunicar os resultados parciais aos computadores cooperantes. Logo, a intercalação entre o tempo de computação e o de comunicação é fundamental para que

os processadores sejam utilizados da melhor forma possível. Para isso, é necessário a existência de um sistema de comunicação eficiente de forma a minimizar o tempo dispendido na comunicação.

Este trabalho avalia o desempenho do sistema de comunicação disponibilizado pelo PVM sobre o sistema operacional Linux. Esta avaliação leva em conta o desempenho das arquiteturas dos processadores da máquina paralela virtual, o desempenho dos protocolos "Transmission Control Protocol" (TCP) e "User Datagram Protocol" (UDP) da tecnologia TCP/IP - Internet, e o desempenho das formas de comunicação via PVM.

O texto está organizado em seis capítulos. O segundo capítulo descreve o mecanismo da comunicação inter-processos para aplicações distribuídas, a interface soquete, os protocolos TCP/IP e a tecnologia Ethernet. O terceiro capítulo classifica as arquiteturas de computadores, conceitua o que são máquinas paralelas e a diferença entre multicomputadores e multiprocessadores, e também discute como projetar algoritmos paralelos. No quarto capítulo são abordadas algumas bibliotecas de passagem de mensagens que permitem a criação de máquinas virtuais, com ênfase nos componentes do PVM, e como ocorre a comunicação entre os processos através do PVM. O quinto capítulo apresenta as técnicas de medição utilizadas para medir a vazão dos protocolos TCP e UDP, e da comunicação via PVM. Também mostra e discute as medições efetuadas nos sistemas de memória, na comunicação via soquetes com TCP e UDP, e na comunicação via PVM. O sexto capítulo apresenta as conclusões do trabalho.

1.1 Trabalhos Relacionados

Boggs et al., em [BMK88], apresentam um estudo sobre mitos e realidade envolvendo a capacidade efetiva e o trafégo de uma rede Ethernet. Eles mostram que alguns estudos teóricos sobre a Ethernet têm cometido equívocos quanto ao seu desempenho por não considerarem padrões de tráfego realísticos na rede. A partir de medições feitas no tráfego da rede, eles também mostram que a Ethernet é capaz de suportar grande tráfego e que a priori, não deve ser considerada como o gargalo na comunicação entre as aplicações; fato também detectado em nossas medições.

Doering et al., em [DDK90], apresentam um estudo comparativo das técnicas utilizadas para implementar os serviços de gerenciamento de conexões, confirmações dos dados recebidos, controle de fluxo e tratamento de erros desempenhados pelos protocolos de transporte APPN, Datakit, Delta-t, NETBLT, OSI/TP4, TCP, VMTP e XTP. Esse estudo surgiu da detecção de que o processamento dos protocolos de comunicação não estavam permitindo as aplicações aproveitarem a capacidade de transmissão nas redes. Os avanços em fibra ótica e em tecnologias VLSI proporcionaram um aumento da capacidade de transmissão das redes e uma diminuição das taxas de erros. Contudo, as aplicações não conseguiam um aumento significativo da velocidade de transmissão de dados. Como resultado do estudo, para cada um dos protocolos estudados é apresentada uma tabela que mostra de maneira resumida um diagnóstico de desempenho dessas técnicas em redes de alta-velocidade. Doering et al. concluem que há muito a ser trabalhar nos protocolos de comunicação de alto desempenho, e que esse estudo serve como referência para isso.

Um estudo detalhado feito por Kay e Pasquale [KP92] apresenta medições de vários componentes das pilhas de protocolos TCP/IP e UDP/IP numa estação de trabalho DEC 5000/200 rodando Ultrix 4.2a numa rede local. Eles mostram que o cômputo da soma de verificação dos dados *checksum* e a movimentação de dados são os componentes dominantes do tempo de comunicação para tráfego realista em rede local. Dessas medições, eles concluem que o uso de pacotes com Unidades Máximas de Transferências (UMTs) grandes é muito importante para se conseguir uma alta vazão na rede e que o desempenho dos protocolos TCP e UDP são semelhantes. Fórmulas deduzidas a partir das medições feitas, que servem para calcular o tempo dispendido nos componentes das pilhas de protocolos TCP/IP e UDP/IP são apresentadas.

O desempenho da comunicação inter-processos do SunOS e a implementação dos protocolos TCP/IP para atender aplicações distribuídas são discutidos por Papadopoulos [PP93] a partir de um estudo sobre o enfileiramento dos dados nas diversas camadas, os mecanismos de controle dos protocolos, o processamento dos pacotes, o gerenciamento de armazenadores e a interação com o sistema operacional.

Dalton et al., em [DWB+93], apresentam um estudo sobre a baixa vazão efetiva detectada em algumas implementações dos protocolos TCP/IP. De acordo com o estudo, o número excessivo de cópias aos quais os dados são submetidos, desde o momento em que são gerados até o instante em que são transmitidos, é o principal responsável pela baixa vazão desses protocolos. Este excesso de cópias limitaria a vazão dos protocolos à largura de banda do sistema de memória. Como uma solução para o problema, três abordagens que permitem a redução do número de cópias são apresentadas. Uma das abordagem, chamada de cópia única, foi implementada no protocolo TCP para HP-UX. Em seguida, eles efetuaram algumas medições em estações de trabalho HP Series 700 com adaptadores de rede Afterburner que apresentam uma vazão de 1 Gbps. As medições mostraram que a implementação do TCP com cópia única permitia as aplicações se comunicarem numa vazão acima de 200 Mb/s.

Uma abordagem sistemática sobre como projetar interfaces de rede para redes de alta velocidade é proposta por Steenkiste em [Ste94]. Seu estudo apresenta os protocolos de comunicação, as interfaces com as aplicações e arquitetura da interface da rede como os principais fatores do processo de comunicação entre as aplicações, e que em determinadas situações tais fatores podem levar a uma sobrecarga do sistema de comunicação. Otimizações a nível de hardware e software são propostas com o objetivo de reduzir uma possível sobrecarga na comunicação.

Rodrigues et al., em [RA96], discutem o fato das redes de alta velocidade apresentarem baixa latência e alta largura de banda, e a comunicação entre as aplicações não apresentarem desempenho semelhante devido às altas sobrecargas de processamento imposta pelos protocolos de comunicação. A criação de uma interface soquete otimizada baseada numa única cópia dos dados do usuário é proposta como uma alternativa de melhorar o desempenho da comunicação local. Enquanto que a comunicação remota continuaria a utilizar a interface soquete de Berkeley.

Um estudo sobre a sobrecarga de processamento apresentada pelas bibliotecas PVM e "Message Passing Interface" (MPI) em redes com estações de trabalho heterogêneas e homogêneas é discutido por Dillon et al. [DSG95]. Os resultados mostram que grande parte do tempo total do processamento é gasto nessas bibliotecas, e que esse tempo tende a aumentar significativamente numa rede de baixa velocidade, como a Ethernet, com máquinas heterogêneas. O estudo também mostra que o desempenho conseguido com a biblioteca MPI não é muito inferior ao desempenho obtido com a biblioteca PVM. De acordo com as medições apresentadas neste trabalho, realmente grande parte do tempo total de processamento é gasto na biblioteca PVM. Contudo, a largura de banda da rede Ethernet não influencia significativamente no tempo total do processamento.

Blum et al., em [BWT96], apresentam um estudo sobre uma implementação alternativa do PVM, chamada PSPVM, para ser executada em redes de alta velocidades ParaStation. O PSPVM pode ser usado para implementar aplicações paralelas eficientes de granularidade fina em aglomerados de estações de trabalho. O sistema ParaStation se baseia na utilização de hardware e software. O hardware é um adaptador de rede PCI que pode ser usado em diversos tipos de estações de trabalhos. Numa aplicação usando o PSPVM os processos se comunicam através de uma interface de soquete implementada em nível de usuário, e que substitui a interface de soquete Berkeley. O protocolo de rede ParaStation elimina completamente a interação do sistema operacional durante as trocas de mensagens; isto permite a comunicação via PSPVM apresentar uma vazão superior e uma latência inferior em relação ao PVM.

Souza et al., em [SSS+97], apresentam um estudo comparativo entre o desempenho dos protocolos TCP/IP implementados nos sistemas operacionais Linux e Windows95, com a finalidade de demonstrar o impacto desses protocolos nos ambientes de passagem de mensagens "Parallel Virtual Machine for Windows95" (PVM-W95) e "Parallel Virtual MAchine" (PVM). A avaliação dos resultados obtidos indicaram que o TCP/IP no Windows95 apresentou um desempenho pior do que o TCP/IP do Linux. O TCP do Windows95 apresentou uma perda média de desempenho da ordem de 2,32 em relação ao

TCP do Linux para as transmissões remotas utilizadas na comunicação entre as tarefas dos usuários no PVM-W95 e PVM. Nas transmissões locais a perda foi ainda maior. Já em comparação ao UDP do Windows95 e do Linux, a perda de desempenho do UDP do Windows95 é da ordem de 4,50 para transmissões remotas e de 14,08 para transmissões locais. A análise do desempenho do UDP é muito importante, visto que a transmissão remota do UDP é muito empregada na comunicação via PVM.

Os trabalhos acima relacionados apresentam estudos sobre técnicas de medições e de avaliação de desempenho de sistemas de memória, protocolos TCP/IP, adaptadores de interface de rede, interfaces de programação entre aplicações e protocolos de comunicação, e bibliotecas que possibilitam processamento paralelo distribuídos. Neste trabalho, esses estudos são usados como referência para medir e discutir cada um dos componentes envolvidos no sistema de comunicação disponibilizado pelo PVM sobre o sistema operacional Linux, desde a vazão da rede física e dos sistemas de memória das máquinas estudadas até o desempenho do sistema de comunicação do PVM.

CAPÍTULO 2

COMUNICAÇÃO INTER-PROCESSOS

A comunicação inter-processos para aplicações distribuídas envolve componentes diversos como protocolos de comunicação e tecnologias de rede que possuem mecanismos próprios de funcionamento, e cuja interação é fundamental para proporcionar um serviço de comunicação eficiente. Este capítulo descreve um modelo de comunicação inter-processos na Seção 2.1. A Seção 2.2 define a interface Soquete usada entre as aplicações e os protocolos de comunicação em rede. A Seção 2.3 explica a tecnologia TCP/IP usada na comunicação inter-processos com ênfase nos protocolos de transporte. Por último, a Seção 2.4 descreve o funcionamento da tecnologia de rede Ethernet.

2.1 Modelo de Comunicação Inter-Processos

O modelo de comunicação inter-processos (CIP) aqui descrito foi definido por Papadopoulos [PP93] em seu estudo sobre o desempenho dos protocolos TCP/IP e o mecanismo de comunicação inter-processos do sistema operacional SunOS para suportar aplicações distribuídas em redes de alta-velocidade. Seus experimentos se basearam em estudar, entre outras coisas, os mecanismos de controle dos protocolos TCP/IP, o processamento de pacotes individuais, as requisições de armazenadores e a interação da CIP com o sistema operacional.

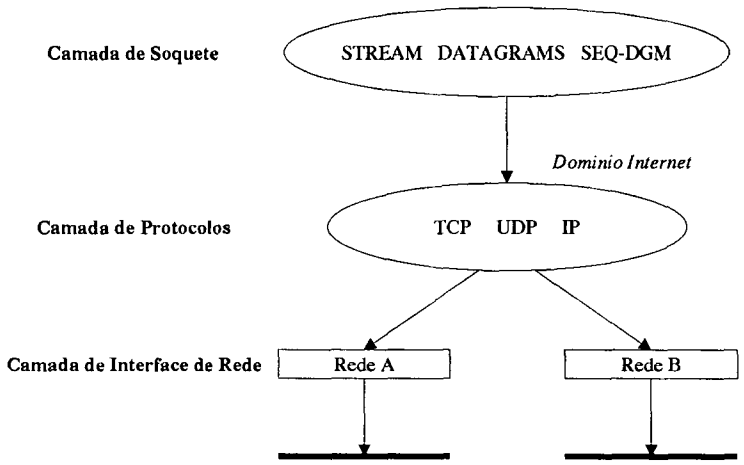

Figura 2.1: Modelo da comunicação inter-processos.

O modelo de comunicação inter-processos de Papadopoulos se baseia em 3 níveis de abstração, chamados de Camada de Soquete, Camada de Protocolos e Camada de Interface de Rede. A ***Camada de Soquete*** faz a interface entre as aplicações e os protocolos de comunicação, e tem como principais funções diferenciar as aplicações em execução numa mesma máquina, e fornecer armazenadores para o envio e recebimento de dados. A abstração suportada por esta camada é o *soquete*. Os soquetes podem suportar diferentes formas de serviços de comunicação. A Seção 2.2 descreve alguns destes serviços.

A ***Camada de Protocolo*** é formada por diversas famílias de protocolos. Cada família é composta pelos protocolos que pertencem a um mesmo *domínio*, que é o esquema de endereçamento que permite a identificação de uma máquina. A Figura 2.1 exibe apenas o domínio Internet com alguns dos protocolos que fazem parte do conjunto de protocolos TCP/IP. Há outros domínios que podem ser acessados através da interface soquete.

A última camada é a ***Camada de Interface de Rede***. Ela é composta pelos controladores dos dispositivos de rede e suas interfaces.

De acordo com este modelo, um processo que deseje se comunicar com outro processo utiliza a interface soquete para determinar o modo de comunicação e o protocolo que disponibilizará a comunicação. A Camada de Soquete então estabelece uma conexão com a máquina remota através de um dos protocolos da camada de protocolos. Os protocolos da Camada de Protocolos recebem os dados enviados pela Camada de Soquete e determinam a que interface da Camada de Rede devem ser repassados. A Camada de

Rede é responsável por gerenciar o envio e o recebimento dos dados através da rede física.

2.2 Soquete

O ***soquete*** é uma interface de programação entre uma aplicação e os protocolos de comunicação. Um soquete fornece serviços de comunicação dos tipos orientado a conexão e não-orientado a conexão. Os serviços orientados a conexão garantem que os dados enviados por uma aplicação chegam a aplicação destino sem erros, duplicação, e na ordem em que foram enviados. Por isso, estes são considerados serviços confiáveis. Os serviços não-orientados a conexão não garantem que os dados enviados por uma aplicação cheguem a aplicação destino; as mensagens podem ser perdidas, duplicadas ou chegar fora da ordem em que foram enviadas.

Nesses dois tipos de serviços, os dados são encapsulados em pacotes denominados *datagramas*. Datagramas são blocos de dados de tamanho definido, normalmente do tamanho da unidade máxima de transmissão da rede, e que possuem o endereço da máquina destino no seu cabeçalho. Esses pacotes são considerados independentes porque podem trafegar por diferentes caminhos até chegarem ao computador destino. A Figura 2.2 mostra a seqüência de passos a serem executados por uma aplicação para estabelecer uma comunicação através de soquete que executa um serviço orientado a conexão. As principais primitivas como *'read'* e *'write'* são discutidas em mais detalhes no decorrer do trabalho. [CS94, PP93] descrevem também outros tipos de interfaces.

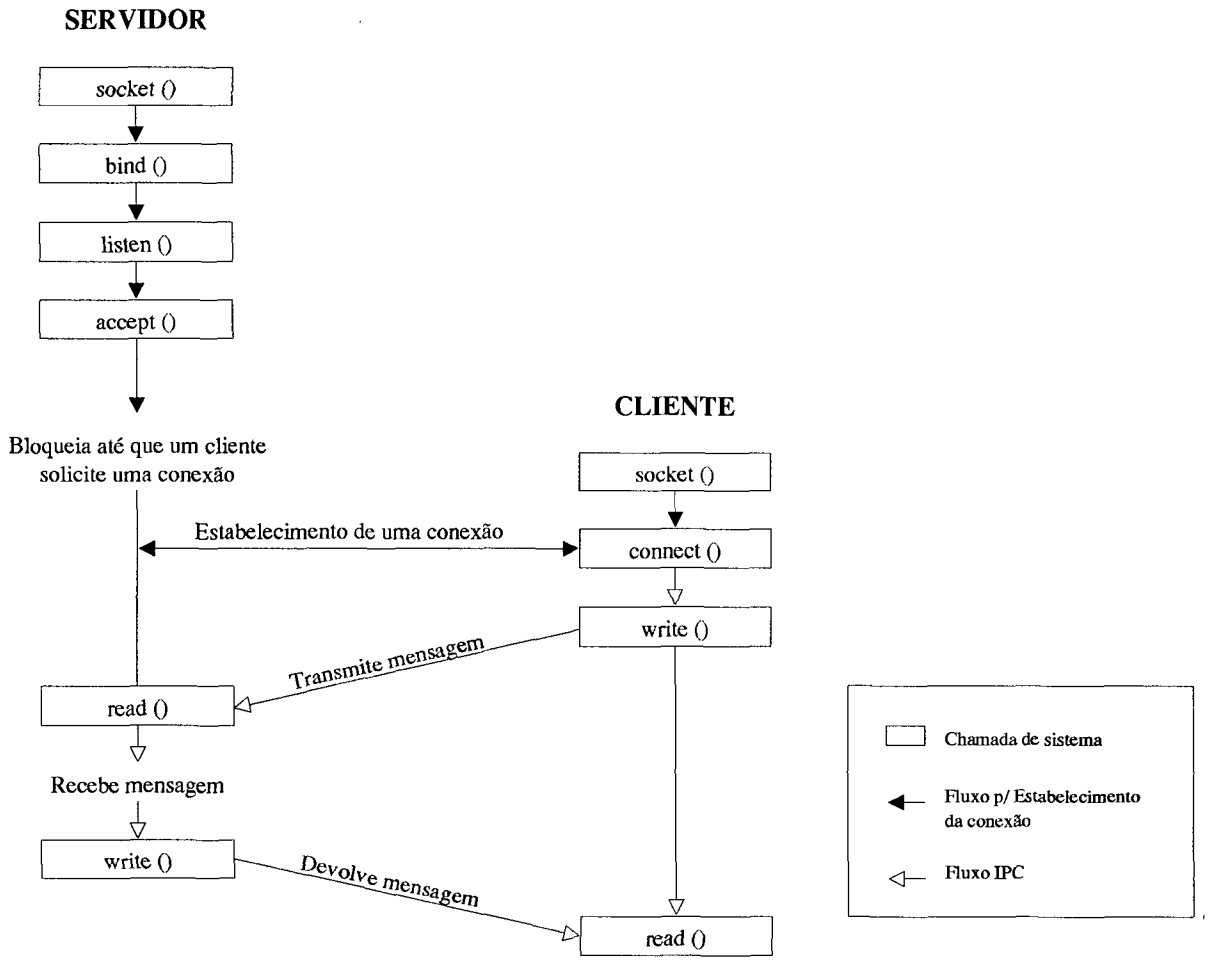

Figura 2.2: Soquete orientado a conexão.

Os soquetes funcionam como pontos terminais de uma comunicação e cada soquete representa uma dessas extremidades. Os soquetes atuam de forma equivalente aos descritores de arquivos. Quando uma aplicação deseja se comunicar com uma outra, ela solicita ao sistema operacional, através da chamada de sistema **socket()**, a criação de um soquete. O sistema operacional retorna um descritor de soquete. Assim como os descritores de arquivos, os descritores de soquetes são usados nas operações de leitura e escrita que permitem o envio e o recebimento dos dados, através de chamadas de sistemas como *'read'* e *'write'*. Diferente do descritor de arquivo que sempre é associado a um arquivo ou dispositivo, o descritor de soquete pode ser criado sem ser associado a um endereço de destino específico [CS94].

A chamada de sistema utilizada para criar um soquete, **socket()**, possui três argumentos. O primeiro argumento representa o domínio, e é definido através da escolha de uma

das famílias de protocolos a ser usada. Algumas das famílias de protocolos reconhecidas pelo Linux são: AF_UNIX para comunicação de processos UNIX, AF_INET protocolos da internet TCP/IP, entre outros. O segundo argumento representa o tipo de serviço que será disponibilizado pelo domínio como, por exemplo, uma comunicação confiável ou não confiável. O terceiro argumento especifica que protocolo irá executar o serviço definido pelo segundo argumento. Para a definição das demais primitivas mostradas na Figura 2.2 e também das primitivas usadas para a comunicação via soquete não-orientado a conexão veja [Ste90].

2.3 A Tecnologia TCP/IP

A família TCP/IP consiste de um conjunto de protocolos que possibilita a comunicação entre aplicações remotas. A arquitetura TCP/IP segue um modelo estratificado em quatro camadas. Da mesma forma como foi mencionado no modelo de comunicação interprocessos, cada camada suporta um conjunto bem-definido de funções. A Figura 2.3 mostra a associação entre a arquitetura e os protocolos que formam a família TCP/IP. As próximas subseções descrevem os protocolos "Transmission Control Protocol" (TCP) e "User Datagram Protocol" (UDP) da camada de transporte, e o protocolo "Internet Protocol" (IP) da camada de rede. Alguns dos demais protocolos da família TCP/IP como "Protocolo de Resolução de Endereços" (ARP), "Protocolo de Resolução de Endereço Reverso" (RARP) e "Protocolo Internet de Mensagens de Controle" (ICMP) são apenas definidos, visto que não importantes neste trabalho. Veja maiores detalhes sobre a arquitetura e os protocolos TCP/IP em [BRI97, CS94, Cen91].

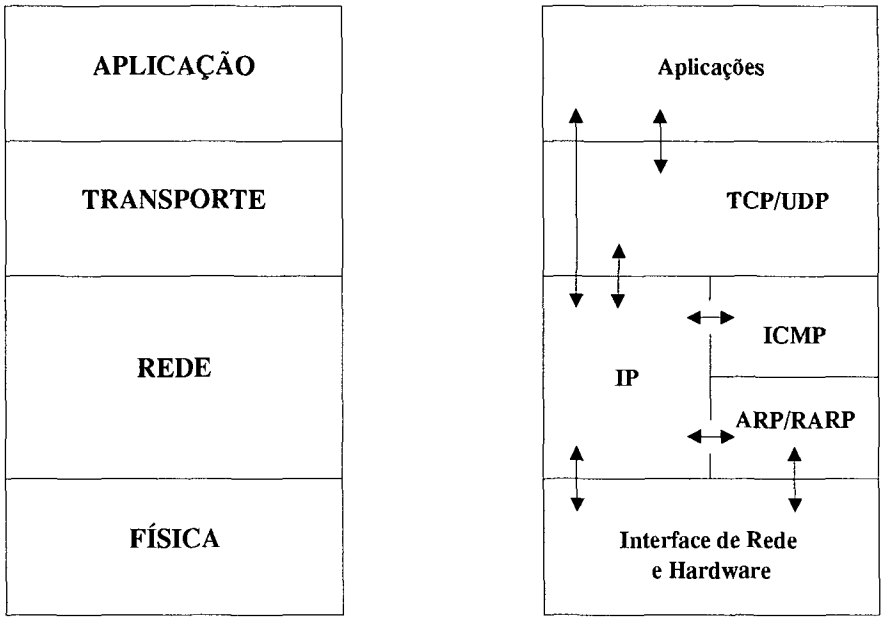

Figura 2.3: Associação entre a arquitetura e os protocolos TCP/IP.

2.3.1 A Camada de Transporte

A camada de transporte é composta pelos protocolos TCP e UDP que são usados na troca de dados entre as aplicações de acordo com o tipo de serviço solicitado. Estes protocolos liberam a camada de aplicação do encargo de gerenciar a infra-estrutura de comunicação usada pelos diversos tipos de aplicação.

O Protocolo TCP

O "Transmission Control Protocol" oferece um serviço confiável de transferência de dados às aplicações. Ele possui mecanismos que garantem a recuperação de dados perdidos, duplicados ou recebidos fora de ordem. O TCP é considerado um protocolo do tipo ponta-a-ponta pelo fato de permitir a comunicação entre aplicações que necessariamente não precisam ser executadas numa mesma máquina ou rede. O TCP é capaz de transferir dados simultaneamente em ambas as direções entre as aplicações. Para isso, ele agrupa os dados a serem transferidos no tamanho equivalente a sua unidade de transferência

de dados que é denominada de ***segmento***. No Linux, o tamanho padrão definido para um segmento TCP é 512 octetos. Contudo, o tamanho máximo de segmento TCP a ser usado numa transmissão é definido durante a fase de sincronização quando a conexão está sendo estabelecida. Primeiro, o TCP descobre a menor unidade máxima de transmissão (UMT) usada pelas redes que fazem parte da conexão. Caso, o tamanho máximo padrão de segmento TCP seja menor do que a menor UMT sem os cabeçalhos IP e TCP, então o tamanho máximo de segmento a ser usado na transmissão é a menor UMT sem os cabeçalhos IP e TCP. Isto permite um melhor desempenho do tráfego da rede porque evita que os segmentos transmitidos sejam fragmentados durante o caminho, e assim aumente o tráfego na rede.

Transferência de Dados Quando duas aplicações desejam comunicar-se, uma conexão deve ser estabelecida entre elas. Uma vez terminada a transferência de dados, a conexão é desfeita para liberar os recursos alocados, e assim permitir que esses possam ser alocados à outras aplicações. Mais detalhes podem ser encontrados em [BRI97].

Entre as funções desempenhadas pelo protocolo TCP se destacam: a multiplexação de fluxos, segmentação, controle de fluxo e o controle de erros. A seguir cada uma destas fuções é descrita.

Multiplexação: é o mecanismo que permite o compartilhamento da rede IP ou física pelos fluxos gerados por diversos aplicativos que executam simultaneamente em uma máquina.

Segmentação: é a maneira como o TCP trata individualmentte cada fluxo originado nas diversas aplicações. Cada fluxo de dados é dividido em segmentos de um determinado tamanho e estes são posteriormente entregues ao protocolo responsável pelo transporte de dados na rede (Veja Subseção 2.3.2). Informações de controle são associadas a cada segmento; estas informações permitirão ao TCP, na máquina destino, reconstituir o fluxo de dados original.

Controle de fluxo: através da técnica da janela deslizante de tamanho variável discutido abaixo, o TCP envia vários segmentos à máquina destino, e conforme o recebimento da confirmação dos segmentos enviados, ele vai se adaptando às diferenças de velocidade de

transmissão existentes na rede entre os sistemas envolvidos para manter a vazão próxima do máximo suportado pela rede.

Controle de erros: os segmentos são enumerados para que possam ser ordenados e entregues na seqüência correta à aplicação destino. É incorporado ao cabeçalho uma palavra de paridade, usada para verificar a correção dos dados que compõem o segmento recebido.

A Técnica da janela deslizante O método mais simples de transferência de dados confiável é aquele em um pacote (segmento) é enviado, e somente quando um pacote com a confirmação da sua aceitação pela aplicação destino retornar é que o próximo será enviado. No entanto, tal método não permite uma boa utilização da rede. É fácil notar que enquanto não houver a confirmação, fica-se impossibilitado de enviar outro pacote e conseqüentemente a rede deixa de ser utilizada no intervalo entre o envio da mensagem e o recebimento da sua confirmação. A técnica da janela deslizante é um mecanismo mais complexo de transmissão, aceitação e retransmissão de pacotes. Através do uso da janela deslizante é possível uma melhor utilização da rede, já que ela permite a transmissão de diversos pacotes antes que se espere por uma confirmação. A maneira mais fácil de compreender como a janela deslizante funciona é imaginar uma seqüência de pacotes numerados de 1 a 10, a ser transmitidos como na Figura 2.4.

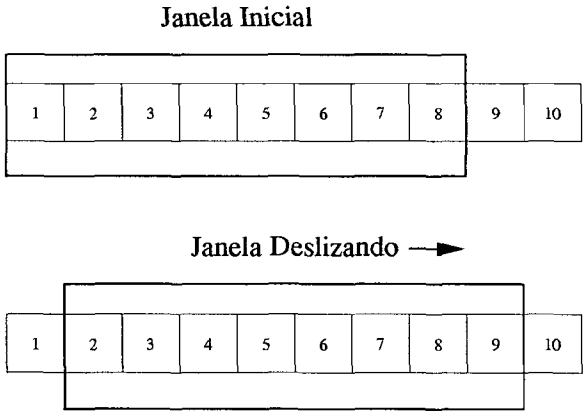

Figura 2.4: Funcionamento da Janela Deslizante.

Primeiro, o TCP transmite todos os pacotes que fazem parte da Janela Inicial. Em seguida, o TCP aguarda a confirmação de recebimento, pela máquina destino, do primeiro

pacote que faz parte da janela de transmissão, para então deslizar a janela. Neste caso, somente quando chegar a confirmação de recebimento do pacote (1) pela máquina destino, é que a janela de transmissão é deslizada e o próximo pacote a ser enviado (9) pode ser transmitido. Neste caso, o pacote (2) passa a ser o primeiro pacote da janela.

Todos os pacotes da janela de transmissão que foram transmitidos, mas cuja confirmação ainda tenha sido recebida são chamados de pacotes não-reconhecidos.

Para cada pacote transmitido pelo TCP, é associado um tempo de vida que corresponde ao tempo necessário para que chegue a confirmação de seu recebimento pela máquina destino. De tempo em tempo, o TCP percorre a janela de transmissão e retransmite todos os pacotes não-reconhecidos cujo tempo de vida tenha sido esgotado.

O fato do TCP transmitir apenas os pacotes que fazem parte da janela de transmissão, evita que sejam transmitidos mais pacotes do que o TCP da máquina destino possa tratar. Isto permite um controle do fluxo de dados, e ajuda a evitar que a rede fique congestionada com pacotes que serão descartados pelo TCP da máquina destino. Veja [CS94] para mais informações sobre a técnica da janela deslizante.

O Protocolo UDP

O "User Datagram Protocol" oferece um serviço de comunicação não-confiável. O protocolo UDP não possui mecanismos de reconhecimento que garantam o transporte de dados de uma aplicação a outra. Ao contrário do TCP, não há a garantia de recuperação dos dados perdidos, danificados ou que foram entregues fora de ordem. Também não há controle de fluxo. Por ser um protocolo simples, o UDP é muito útil principalmente para as aplicações que executam em redes locais onde a incidência de problemas com o transporte de dados é menor. As aplicações que usam o serviço de comunicação UDP são responsáveis pela confiabilidade da transferência dos dados.

2.3.2 A Camada de Rede

A camada de rede é responsável pelo serviço de envio de pacotes de dados de uma máquina para outra, independente de as máquinas estarem na mesma sala ou em continentes

diferentes. A cada máquina na Internet é associado um endereço único que possibilita a sua identificação. Esse endereço chamado de *endereço Internet* é composto de 4 octetos.

O endereço Internet permite aos protocolos TCP/IP ocultar as características físicas das redes que interligam dois pontos quaisquer em uma internet. O "Protocolo de Resolução de Endereços" (ARP) é responsável por obter o endereço físico correspondente a um endereço Internet e que é realmente o endereço utilizado na transmissão e na identificação dos pacotes na rede física. Além do ARP, a camada de rede é composta pelos "Protocolo Internet de Mensagens de Controle" (ICMP), "Protocolo de Resolução de Endereço Reverso" (RARP) e IP. O ICMP é responsável por enviar e tratar as mensagens de erro ou de controle geradas pelos nós intermediários numa internet. O RARP é responsável por obter o endereço Internet correspondente ao endereço físico de uma máquina.

O Protocolo IP

O "Internet Protocol" é responsável pelo transporte de dados na Internet através de redes físicas com tecnologias diversas. O IP é considerado um protocolo do tipo ponta-a-ponta, porque sua função é garantir que os datagramas (blocos de dados contendo informações das aplicações ou de controle) saiam da máquina origem e atinjam a sua máquina destino. Quando o datagrama chega à máquina destino, o protocolo IP, atráves de um dos campos do cabeçalho do datagrama, identifica a que protocolo deve repassar o bloco de dados (multiplexação). Normalmente, o bloco de dados é repassado para um dos protocolos da camada de transporte. Para obter informações mais detalhada do funcionamento do IP veja [BRI97, CS94].

2.4 A Ethernet

O funcionamento da Ethernet é descrito, dentre as diversas tecnologias de redes disponíveis, para exemplificar a camada de interface rede e porque faz parte deste estudo. A Ethernet é uma tecnologia para redes locais de computadores baseada na transmissão de pacotes chamados de *quadros*, num meio físico utilizando o mecanismo de controle de acesso ao

meio "acesso múltiplo por detecção de portadora com detecção de colisão" (CSMA/CD). Devido a facilidade de instalação e ao seu baixo custo por conexão, ela é uma das tecnologias de rede local mais utilizadas.

O IEEE é o orgão responsável pela publicação das normas que regulam a produção de equipamentos Ethernet [Hex99]. O padrão Ethernet define as camadas mais baixas dos protocolos de comunicação, normalmente chamadas de Nível Físico e Nível de Enlace. Este texto não faz uma descrição ou comparação dos componentes do Nível Físico. Veja [Hex99] para tais informações. No entanto, o Nível de Enlace será descrito mais adiante por possuir o mecanismo de controle de acesso ao meio físico, e também por ser responsável pelo formato e conteúdo dos pacotes que trafegam na rede.

2.4.1 Arquitetura

A configuração típica de uma rede Ethernet é composta de um meio físico, normalmente um barramento serial, ao qual os computadores são conectados. As conexões são feitas através de adaptadores de interface como este mostrado na Figura 2.5. Num adaptador de interface o componente chamado de Transceptor tem as funções de acoplar a interface de rede ao meio físico, transmitir os quadros gerados e receber os quadros que são injetados no meio físico por outros computadores. Os componentes "Controle de Acesso ao Meio" (CAM) e "Controle de Enlace Lógico" (CEL) são os dois subníveis do Nível de Enlace responsáveis, respectivamente, pelo controle de acesso ao meio físico, e pelo formato e conteúdo dos quadros. Eles são examinados abaixo.

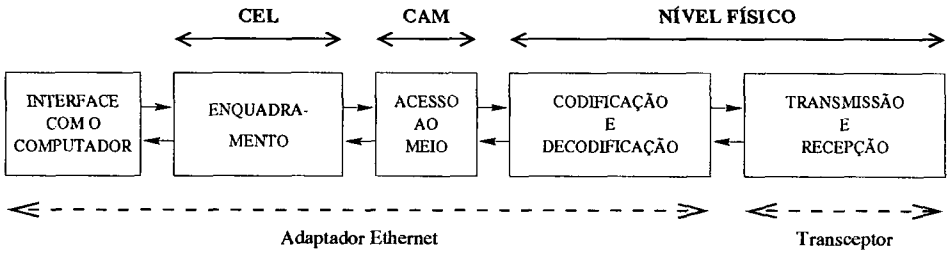

Figura 2.5: Arquitetura de um adaptador Ethernet.

2.4.2 Nível de Enlace

O subnível CEL é responsável por montar os quadros que são injetados na rede e por determinar se os quadros recebidos são endereçados endereçados para aquele computador. O subnível CAM controla o acesso ao meio físico através do protocolo distribuído CSMA/CD.

Controle de Enlace Lógico - CEL

O Controle de Enlace Lógico gera os quadros Ethernet e os repassa ao subnível de controle de acesso ao meio. Como mostra a Figura 2.6, um quadro Ethernet é formado por um preâmbulo, o endereço Ethernet do computador destino, o endereço Ethernet do computador origem, um campo Tipo, um campo Dados do Usuário e um bloco de detecção de erros baseado em "cyclic redudancy check" (CRC).

PREÂMBULO	DESTINO	ORIGEM	TIPO	DADOS	CRC
8 OCTETOS	6 OCTETOS	6 OCTETOS	2 OCTETOS	46-1500 OCTETOS	4 OCTETOS

Figura 2.6: Formato de um quadro Ethernet.

O preâmbulo é composto de uma seqüência de bits que ajuda na sincronização entre o transmissor e os receptores do quadro. O endereço Ethernet é composto de 6 octetos. O campo Tipo identifica o tipo de informação que está sendo transportada no quadro. Assim, quando o quadro chega ao seu destino é possível determinar a que protocolo de Nível de Rede a informação deve ser entregue. O tamanho do campo Dados do usuário pode variar de 46 a 1500 octetos. Logo, a unidade máxima de transferência da Ethernet (UMT) é 1500 octetos. Já o CRC é usado para detectar se houve algum erro durante a transmissão.

Controle de Acesso ao Meio - CSMA/CD

O controle de acesso ao meio físico é feito por todos os computadores da rede segundo o método CSMA/CD. O CSMA/CD é um protocolo distribuído que permite a cada computador ter os mesmos direitos de transmissão que os demais. Um computador que queira

começar uma transmissão deve primeiro "escutar" o meio físico. Caso seja detectada uma portadora, ou seja, existe um outro computador já transmitindo, o computador deve esperar e postergar sua transmissão até um momento quando o meio físico estiver em silêncio ou "sem portadora". É possível que mais de um computador detecte a ausência de portadora e comeÇe suas transmissões ao mesmo tempo, o que acaba gerando a "colisão" dos sinais de transmissão dos quadros. Como mostra a Figura 2.5, o Transceptor é responsável pela detecção de colisão e por abortar a transmissão sempre que ocorrer uma colisão. A medida que o tráfego na rede aumenta, o número de colisões também aumenta. Logo, para que o protocolo CSMA/CD funcione corretamente e resolva os problemas de conflitos de acesso ao meio de uma forma justa, todos os computadores devem seguir os seguintes passos: escuta, deferência, colisão e consenso, como visto a seguir.

Escuta: Cada computador deve primeiro detectar a ausência de portadora antes de transmitir.

Deferência: Caso seja detectada alguma transmissão, ou seja, portadora de sinais no meio físico, a interface deve esperar até que haja ausência de portadora.

Colisão: Ao detectar uma colisão, a interface deve abortar a transmissão do quadro e retransmití-lo num outro momento.

Consenso: Uma vez detectada a colisão durante a transmissão, a interface insere quatro octetos no meio físico para assegurar que os outros computadores detectem a colisão. Em seguida, a interface aborta a transmissão.

Tais passos formam a regra que determina o comportamento dos computadores na rede.

Transmissão e Recepção de dados pela interface Ethernet

Conforme mostra a Figura 2.5, o subnível CEL é responsável por gerar os quadros Ethernet. Em seguida, caso seja detectado pelo subnível CAM que o meio físico está em silêncio, ou seja, que não há portadora, o quadro Ethernet é então codificado e começa a ser transmitido. O Transceptor, a medida que vai injetando o quadro no meio físico, compara os bits que recebe com os que transmite. Caso haja colisão, ou seja, que os bits recebidos

diferem dos transmitidos, o Transceptor transmite mais alguns octetos para garantir que os outros computadores da rede detectem a colisão e então aborta a transmissão.

Já a recepção funciona da seguinte forma: o Transceptor, após detectar a portadora, sincroniza a sua recepção com a do transmissor a partir dos octetos do campo prêambulo. Uma vez recebido o quadro, caso o endereço do destinatário seja o mesmo do receptor, a interface copia os octetos do quadro para um armazenador ao mesmo tempo em que calcula o CRC. Se for verificado que não houve erro de transmissão, o conteúdo dos campos endereço Ethernet do computador de origem, Tipo e Dados do Usuário são passados para o CEL.

Esta Seção apresentou uma descrição dos conceitos e do funcionamento da Ethernet. Esta descrição é importante porque a maioria dos ambientes de computação de baixo custo é composto por redes locais baseadas nesta tecnologia. Assim, para implementar processamento paralelo distribuído nesses ambientes, é fundamental entender a tecnologia de rede usada, pois ela é o fator limitante da comunicação entre os processadores, e por conseqüência do sistema de comunicação das máquinas virtuais paralelas criadas nesses ambientes. O próximo capítulo discute a classificação das arquiteturas de computadores, e a importância do mecanismo de comunicação usado entre os processadores para um bom desempenho das máquinas virtuais paralelas.

CAPÍTULO 3

PROCESSAMENTO PARALELO

Uma maneira de se conseguir um maior poder computacional para executar determinadas aplicações, é através do uso de paralelismo. Numa máquina paralela, parte do tempo de processamento é gasto com computação, tempo usado no processamento dos dados, e a outra parte é gasto com a comunicação, tempo usado no envio e recebimento de mensagens entre os processos de uma aplicação. A intercalação entre o tempo gasto com computação e com comunicação é fundamental, para que todos os processadores sejam utilizados da melhor forma possível. Assim, conceitos como balanceamento de carga, granularidade, escalabilidade e sincronização que são discutidos neste capítulo, devem ser levados em consideração pelo programador na hora da implementação de uma aplicação paralela. Esses conceitos possibilitam ao programador desenvolver aplicações que utilizem a capacidade de processamento das máquinas paralelas de um forma mais eficiente, através da distribuição adequada dos processos e dados da aplicação entre os processadores e memória. Este capítulo descreve a classificação das arquiteturas de computadores proposta por Flynn [Fly72] na Seção 3.1. Em seguida, a Seção 3.2 descreve a subdivisão das máquinas parelelas em multiprocessadores e multicomputadores. A Seção 3.3 descreve como projetar algoritmos paralelos. Por último, a Seção 3.4 define algumas propriedades que devem ser levadas em consideração ao se implementar um programa em paralelo.

3.1 Classificação das Arquiteturas de Computadores

Em 1966, Flynn [Fly72] propôs o primeiro método que foi amplamente disseminado para classificar arquiteturas de computadores. O método se baseia nas possibilidades de combinação entre uma ou mais seqüências de instruções, onde cada seqüência corresponde

a um programa, atuando sobre uma ou mais seqüências de dados. A classificação divide os computadores em quatro classes: SISD (uma seqüência de instruções e uma seqüência de dados), SIMD (uma seqüência de instruções e múltiplas seqüências de dados), MISD (múltiplas seqüências de instruções e uma seqüência de dados), MIMD (múltiplas seqüências de instruções e múltiplas seqüências de dados).

As máquinas SISD, como mostra a Figura 3.1, possuem um único processador e cada instrução manipula um dado por vez. Posteriormente, esse tipo de máquina foi adaptado para possuir um certo paralelismo. De forma semelhante a uma linha de montagem, cada instrução pode ser executada em etapas; o que permite que haja mais de uma instrução sendo executada a cada instante. As máquinas com essa configuração são chamadas de máquinas segmentadas (pipeline).

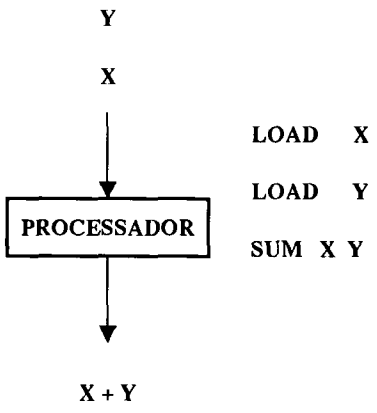

Figura 3.1: Máquina SISD.

As máquinas SIMD possuem diversos processadores, e cada instrução é replicada para cada um deles e executadas em paralelo. Essas máquinas são utilizadas para solucionar problemas em que se precisa executar uma mesma tarefa com diferentes dados.

Não existe máquina que possa ser classificada como MISD porque é ilógico se contruir uma máquina onde múltiplos programas venham a processar os mesmos dados num mesmo instante.

As máquinas MIMD possuem diversos processadores que executam suas instruções independentemente dos outros processadores. Elas têm como vantagem o fato de poderem executar diversas instruções simultaneamente e a maior dificuldade atualmente está no custo necessário para se obter uma boa distribuição das cargas de processamento.

Apesar das máquinas SIMD e MIMD serem máquinas paralelas, este texto trata especificamente das máquinas MIMD. Portanto, ao nos referirmos a máquinas paralelas, estaremos falando de máquinas MIMD.

3.2 Subdivisão das arquiteturas paralelas

De acordo com Patterson [PH96], as máquinas paralelas podem ser divididas do ponto de vista da organização da memória em máquinas com memória compartilhada centralizada e máquinas com memória fisicamente distribuída.

Como mostra a Figura 3.2, as máquinas com memória compartilhada centralizada são compostas de uma pequena quantidade de processadores e uma memória central que são interligados por um barramento. Essas máquinas se caracterizam pelo fato da memória central satisfazer as demandas de memória de cada um dos processadores.

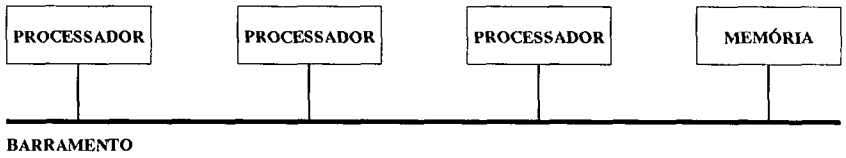

Figura 3.2: Máquina com memória compartilhada centralizada.

Já as máquinas paralelas compostas de um grande número de processadores ou de apenas alguns processadores muito rápidos possuem o sistema de memória composto de memórias fisicamente distribuídas. Isto ocorre porque a *largura de banda*, quantidade de dados passível de ser transmitida por unidade de tempo em um determinado meio de comunicação, disponível por um sistema de memória composto de uma única memória centralizada normalmente não satisfaz a demanda de processadores velozes.

A distribuição da memória entre os processadores tem duas vantagens. Primeira, se um processador faz muitos acessos à sua memória local, a memória distribuída reduz a demanda de largura de banda no barramento. Segunda, uma memória distribuída reduz a *latência*, tempo consumido entre a emissão de uma mensagem pelo processo transmissor e sua recepção pelo processo receptor, para os acessos a memória local, apesar de aumentar enormemente a latência para os acessos remotos. Estas vantagens tornam o sistema de memória distribuída muito atrativo para as máquinas paralelas compostas apenas de

alguns processadores e que requerem bastante largura de banda de memória.

A principal desvantagem numa máquina paralela com memória fisicamente distribuída é que a comunicação de dados entre os processadores torna-se mais complexa, o que acaba gerando um aumento na latência, visto que os processadores não mais compartilham uma única memória centralizada.

As máquinas com memória fisicamente distribuída podem ainda ser diferenciadas do ponto de vista de como ocorre a comunicação entre os processadores e de como a memória é logicamente tratada.

Conforme o método usado na comunicação de dados entre os processadores, as memórias fisicamente distribuídas podem ser acessadas como único espaço de endereçamento que é logicamente compartilhado ou como múltiplos espaços de endereçamento que são logicamente disjuntos.

As máquinas cuja memória fisicamente distribuída é acessada como um único espaço de endereçamento logicamente compartilhado são chamadas de *multiprocessadores* ou arquiteturas com *memória compartilhada distribuída* [PH96]. Nelas, todos os processadores podem fazer referência a qualquer posição do espaço de endereçamento. Logo, um mesmo endereço físico em dois processadores refere-se ao mesmo local da memória compartilhada por eles.

Já as máquinas com memória fisicamente distribuída, vista como múltiplos espaços de endereçamento logicamente disjuntos são chamadas de *multicomputadores.* Nestas máquinas, cada módulo processador-memória é basicamente um computador completo. Assim, um mesmo endereço físico em dois processadores diferentes refere-se a diferentes posições em duas memórias diferentes. Os multicomputadores podem ser implementados com computadores completamente separados conectados a uma rede local, como mostra a Figura 3.3.

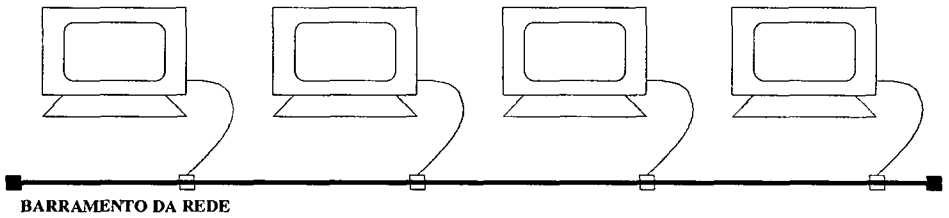

Figura 3.3: Máquina com memória distribuída.

Conforme mencionado acima, para cada maneira como é organizado o espaço de endereçamento de uma máquina com memória fisicamente distribuída, há um mecanismo de comunicação associado. Em multiprocessadores, a comunicação de dados ocorre de forma implícita atráves de operações de escrita e leitura pelo processadores. Já a comunicação nos multicomputadores ocorre de forma explícita através da passagem de mensagens entre os processadores. A passagem de mensagens é efetuada por pacotes como "Parallel Virtual Machine" (PVM) e "Message Passing Interface" (MPI).

3.3 Projeto de Algoritmos Paralelos

Segundo Quinn [Qui87], há no mínimo 3 maneiras de se projetar algoritmos paralelos para resolver um determinado problema: (1) detectando e explorando qualquer paralelismo existente num algoritmo seqüencial, (2) inventando um novo algoritmo paralelo ou (3) adaptando um algoritmo paralelo que soluciona um problema similar.

Os algoritmos projetados para máquinas paralelas podem ser divididos em 3 categorias: segmentado, particionado e relaxado.

Nos algoritmos *particionados*, cada processador assume deveres computacionais diferentes. O problema é dividido em sub-problemas ou "tarefas" e estes são entregues aos processadores. As soluções dos sub-problemas são então combinadas para formar a solução do problema.

Já um algoritmo *segmentado* é um conjunto ordenado de segmentos ou "tarefas" onde os resultados gerados por um segmento servem como dados de entrada para um outro segmento, e o resultado gerado pela último segmento é o resultado do algoritmo. Da mesma maneira como numa linha de montagem, todos os segmentos devem produzir resultados de forma sincronizada para que não haja um gargalo que venha a prejudicar o

desempenho do algoritmo.

Num algoritmo *relaxado* não há uma sicronização entre as tarefas, e as tarefas podem ter um comportamento semelhante a dos algoritmos particionados ou a dos algoritmos segmentados.

Como não se trata de uma divisão rígida, é possível a um algoritmo paralelo possuir as três características, como por exemplo, ser dividido em segmentos e um ou mais desses segmentos serem particionados ou novamente segmentados.

3.4 Fatores a considerar na implementação de um programa paralelo

A existência de procedimentos independentes que possam ser executados em paralelo e a relação de controle e dependência entre esses procedimentos são alguns dos aspectos que intuitivamente devem ser levados em consideração quando se deseja implementar um programa paralelo.

Algumas das propriedades que devem ser levadas em conta no desenvolvimento de uma aplicação paralela são o balanceamento de carga, a granularidade, a escalabilidade e a sincronização. Estas são discutidas abaixo.

Balanceamento de Carga: é a distribuição dos processos (tarefas) que compõem a solução de um problema entre os processadores de uma máquina paralela de acordo com a capacidade de processamento de cada processador, e de forma que a solução para o problema seja obtida o mais rápido possível. O balanceamento de carga pode ser feito por um processo ou pelo usuário. Os processos que não puderem ser executados por falta de processador livre ou desocupado ficam aguardando numa fila e a medida que os processadores terminem a execução dos processos anteriormente escalonados, estes processos são executados.

Granularidade: é a razão entre o tempo gasto com computação e o tempo gasto com comunicação. A granularidade descreve a relação entre o tamanho de cada processo e o tamanho total da aplicação. A granularidade pode ser grossa ou fina. Uma aplicação de granularidade grossa possui processos com muita computação e pouca comunicação, menor custo de processamento, maior desempenho, menor custo de sincronização e maior dificuldade no balanceamento de carga. Uma aplicação de granularidade fina possui muita

comunicação, menor desempenho, alto custo de sincronização e maior facilidade de balanceamento.

Escalabilidade: é a capacidade que um sistema paralelo possui de aumentar seu poder de processamento a medida que se aumenta o número de processadores. Entretanto, obstáculos a nível de software e de hardware impõem limitações na escalabilidade. A limitação a nível de software é imposta pela maneira como a aplicação é implementada. Há algoritmos que possuem melhor desempenho com um certo nível de paralelismo; e o aumento do número de processadores pode levar a uma diminuição do desempenho da aplicação. A limitação a nível de hardware se deve ao aumento de tráfego na rede que interliga os processadores, o que pode ocasionar a degradação da capacidade de comunicação da rede e conseqüentemente diminuir o desempenho da aplicação.

Sincronização: é a maneira como se estabelece a troca de mensagens entre os processos. A sincronização é fundamental nas situações onde há a necessidade de execução ordenada de instruções e dependências de dados.

Além dos aspectos mencionados acima, o modelo de comunicação, a tecnologia de rede, a largura da banda da rede, o sistema operacional e configuração da máquina paralela (número de processadores, tamanho da memória, uso de caches) são fatores que combinados determinam o desempenho da máquina. Veja [Cen96, CEN97, Sto93] para maiores informações.

CAPÍTULO 4

PVM

O uso de processamento paralelo distribuído em ambientes de redes locais permite o melhor aproveitamento dos recursos de hardware existentes. Tarefas que exigem maior poder computacional podem ser executadas através da implementação de aplicações compostas de diversos processos. Estes processos são alocados aos processadores, e trocam dados entre si através de passagem de mensagens com o objetivo de cooperarem na execução de cada tarefa. Entre as bibliotecas que permitem a criação de ambientes de processamento paralelo distribuído tais como PVM, P4 e MPI, o PVM se estabeleceu como um padrão *de facto* principalmente por ser de domínio público. No PVM esses ambientes são chamados de máquinas virtuais paralelas. O PVM destaca-se também por sua robutez, simplicidade e eficiência. A Seção 4.1 descreve resumidamente as bibliotecas P4 e MPI. A Seção 4.2 apresenta uma visão geral do PVM, e a Seção 4.3 descreve em maiores detalhes o seu funcionamento. A Seção 4.4 descreve a comunicação entre os componentes do PVM, e finalmente, a Seção 4.5 descreve as formas de empacotamento dos dados a serem transmitidos, e os tipos de roteamento usado no envio e recebimento das mensagens.

4.1 Os sistemas P4 e MPI

Segundo [GBD+94a], o P4 é uma biblioteca de macros e subrotinas desenvolvidas pelo Laboratorio Nacional de Argonne (EUA) para permitir a programação de uma variedade de máquinas paralelas. O sistema P4 suporta sistema de memória compartilhada (baseada em monitores) e o sistema de memória distribuída (usando passagem de mensagens). Para o modelo de memória compartilhada, o P4 fornece um conjunto de monitores, bem como

um conjunto de primitivas com as quais novos monitores podem ser construídos. Para o modelo de memória distribuída, o P4 fornece operações de *'send'* e *'receive'* e de criação de processos remotos.

O gerenciamento de processos no sistema P4 é baseado em um arquivo de configuração que especifica um conjunto de máquinas, o arquivo objeto a ser executado em cada máquina, o número de processos a ser iniciado em cada máquina (dirigida primeiramente para sistemas de multiprocessadores), e outras informações auxiliares.

A Passagem de Mensagem no sistema P4 é feita através do uso das primitivas *'send'* e *'recv'*, parametrizadas de forma similar a outros sistemas de passagem de mensagem. A parte de gerenciamento e alocação de armazenadores, no entanto, é responsabilidade do programador. Além de passagem de mensagens básica, o P4 também oferece uma variedade de operações globais, que incluem difusão (broadcast), máxima e mínima global e sincronização por barreira.

Segundo [SOHL96], a "Message Passing Interface" (MPI) é uma biblioteca de rotinas que utiliza passagem de mensagens na comunicação entre os processadores, sendo considerada um padrão por consenso, já que não há um padrão definido por organizações como ISO ou ANSI. A MPI é decorrente de esforço coletivo para tentar definir a sintaxe e a semântica de uma biblioteca de rotinas de passagem de mensagem que seja útil à comunidade de programação paralela e implementável eficientemente em grande variedade de máquina paralelas. A principal vantagem de estabelecer uma padrão de passagem de mensagem é a portabilidade dos programas baseados no padrão.

4.2 PVM

O "Parallel Virtual Machine" (PVM) é um sistema baseado em passagem de mensagens que possibilita que um conjunto de computadores com arquiteturas heterogêneas e/ou homogêneas seja usado como uma máquina virtual paralela, aproveitando assim os recursos computacionais de uma rede e potencialmente reduzindo o tempo total de execução de uma aplicação. O PVM baseia-se na idéia de que uma aplicação PVM consiste de uma coleção de tarefas, sendo cada tarefa responsável por uma parte do trabalho computacional da aplicação. O PVM é responsável pela alocação das tarefas aos processadores e

pelo gerenciamento da comunicação entre essas tarefas. Para isso, ele é composto de duas partes: o "processo servidor" *daemon* e as bibliotecas do sistema. O *daemon*, chamado de ***Pvmd***, é responsável por executar as aplicações PVM. Já as bibliotecas PVM são responsáveis pela implementação da comunicação entre os *daemons*, criação e sincronização entre as tarefas e tudo mais que for necessário à implementação de uma aplicação PVM.

O método mais comum de paralelização de uma aplicação é o paralelismo de dados. Neste método, todas as tarefas são iguais; entretanto, cada tarefa conhece e opera sobre uma parte dos dados do problema. Uma aplicação também pode ser paralelizada de forma que cada tarefa execute uma função diferente. O PVM suporta os dois métodos acima ou a combinação deles. Uma das principais características do PVM é a sua portabilidade, podendo uma aplicação ser executada em arquiteturas heterôgenas consistindo de PC's até computadores de grande porte ou supercomputadores. O PVM suporta as linguagens C, C++ e Fortran.

4.3 Como Funciona o PVM

O ***Pvmd*** é um processo servidor (daemon) que atende as solicitações feitas pelas tarefas pertencentes a máquina onde ele está sendo executado e aos Pvmds de outras máquinas. O Pvmd também possui todos os métodos e estruturas necessários a manipulação dos pacotes que compõem as mensagens. Todas as mensagens trocadas entre as tarefas são empacotadas antes de serem enviadas. O empacotamento ocorre pois a máquina virtual paralela usada na execução de uma aplicação PVM pode ser formada por computadores de arquiteturas heterogêneas. A Seção 4.5 descreve detalhadamente as formas de empacotamento nas quais as mensagens podem ser submetidas. O Pvmd é responsável por:

1. Controlar outros Pvmds escravos, e todo o roteamento de mensagens efetuado pelos Pvmds escravos;

2. Manter todas as filas de mensagens recebidas e/ou por enviar;

3. Criar e manter atualizadas a tabela com as máquinas componentes da máquinas virtual e a tabela de tarefas, permitindo assim que a configuração da máquina virtual

paralela esteja sempre correta;

4. Estabelecer e executar o protocolo de comunicação Pvmd-Pvmd e Pvmd-Tarefa, para que as requisições e respostas sejam transmitidas e/ou recebidas corretamente;

5. Criar novos Pvmds, permitindo assim modificar dinamicamente a máquina virtual.

O primeiro Pvmd (iniciado pelo usuário) é denominado de mestre, enquanto os outros (iniciados pelo mestre) são chamados de escravos. Durante uma operação normal, todos os Pvmd são iguais, mas somente o mestre pode iniciar novos escravos e adicioná-los à configuração.

As bibliotecas Libpvm possuem as primitivas necessárias para que cada tarefa possa se comunicar com o Pvmd e com outras as tarefas. Elas contêm funções para empacotar e desempacotar mensagens, e para enviar pedidos de serviço ao Pvmd. De uma maneira geral, as Libpvm contêm as funções para:

1. Implementar os codificadores e decodificadores de mensagens e dados utilizados em plataformas heterogêneas;

2. Criar os mecanismos de comunicação inter-processos (CIP) necessários à comunicação com o Pvmd e/ou à comunicação diretamente entre as tarefas;

3. Executar os protocolos de comunicação Pvmd-Tarefa e Tarefa-Tarefa.

4.4 Comunicação PVM

A Comunicação PVM é baseada nos protocolos TCP e UDP, e na interface soquete. A Figura 4.1 exemplifica uma aplicação PVM composta de quatro tarefas sendo executadas numa máquina virtual paralela com três processadores, P1 a P3. Como mostra a figura, há três formas de conexão no sistema PVM: (1) conexões entre Pvmds, (2) entre Pvmd e Tarefa, e (3) entre Tarefas. As próximas seções descrevem os mecanismos de comunicação disponíveis no PVM.

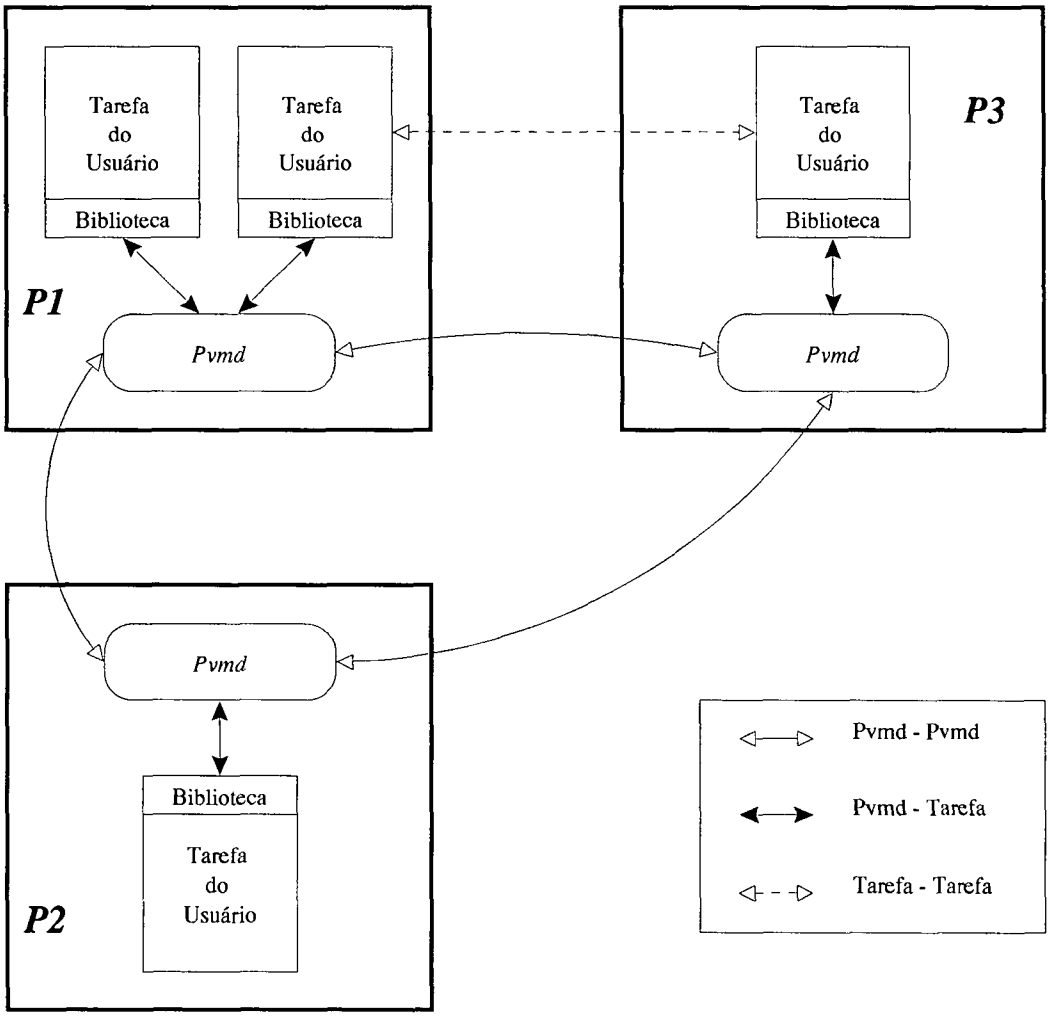

Figura 4.1: Comunicação PVM.

4.4.1 Comunicação Pvmd-Pvmd

Um Pvmd se comunica com outro Pvmd através de soquetes UDP. É importante lembrar que o protocolo UDP oferece um serviço não-confiável, ou seja, os pacotes transmitidos podem ser perdidos, duplicados ou entregues fora de ordem. Assim, para que a comunicação entre os Pvmds torne-se confiável, cada Pvmd possui um mecanismo de confirmação e retransmissão dos pacotes. O Pvmd não usa o protocolo TCP, que oferece um serviço confiável, para se comunicar com outros Pvmds. O uso do protocolo TCP implica em fatores como escalabilidade e sobrecarga (overhead) no sistema de comunicação. Numa máquina virtual com N nodos, por exemplo, cada Pvmd precisa estabelecer conexões com os outros $N - 1$ nodos. Como cada conexão TCP consome um descritor de arquivo, e dependendo do sistema operacional, o número de descritores de arquivos é limitado, a

escalabilidade é reduzida e torna o uso do TCP inapropriado para a comunicação entre os Pvmds. Já com o uso do UDP, cada Pvmd precisa apenas de um único soquete UDP para se comunicar com os outros Pvmds.

A comunicação entre os Pvmds via protocolo TCP seria ineficiente devido ao custo de processamento necessário para criar as conexões entre os Pvmds que formam a máquina virtual. Para N Pvmds são necessárias $N(N-1)/2$ conexões. Já o Pvmd com UDP não precisa estabelecer qualquer conexão.

4.4.2 Comunicação Pvmd-Tarefa e Tarefa-Tarefa

A comunicação entre as tarefas de uma aplicação PVM e o Pvmd executando numa mesma máquina é feita através de soquetes TCP. A comunicação via soquetes TCP garante a integridade dos dados transmitidos, e torna a comunicação Pvmd-Tarefa confiável. Da mesma forma, a comunicação direta entre as tarefas de uma aplicação PVM, sendo executadas numa mesma máquina, ou não, também é feita através de soquetes TCP, como mostra a Figura 4.1.

4.5 Formas de empacotamento e de transmissão de mensagens no PVM

O envio de uma mensagem através da comunicação PVM consiste de três etapas. Na primeira etapa, através da primitiva *init_send()*, o PVM cria um armazenador de envio e define o tipo de empacotamento que ele receberá. Na segunda etapa, a mensagem a ser enviada deve ser empacotada e colocada no armazenador de envio; e na última etapa, a mensagem empacotada é então transmitida.

O PVM permite três formas de empacotamento dos dados a transmitir: "External Data Representation" (XDR), "Data Raw" (RAW) e "In Place" (IN). A forma XDR é utilizada normalmente na troca de dados entre tarefas que estão sendo ou podem vir a ser executadas em máquinas de arquiteturas heterogêneas. Na forma XDR, os dados são codificados antes de serem colocados no armazenador de envio.

A forma RAW é utilizada em ambientes constituídos de máquinas homogêneas. Nesta forma de empacotamento, os dados são simplesmente copiados do armazenador da tarefa para o armazenador de envio, permitindo assim que as tarefas economizem o tempo que

é consumido com a codificação dos dados antes de colocá-los no armazenador de envio, como ocorre na forma de empacotamento XDR.

A forma IN também é utilizada em ambientes homogêneos. No entanto, ao invés de se copiar os dados do armazenador da tarefa para o armazenador de envio, apenas o tamanho dos dados e o ponteiro para o armazenador da tarefa são copiados no armazenador de envio. Assim, o número de vezes que os dados são copiados diminui, e as tarefas não desperdiçam tempo no empacotamento dos dados.

Além das três formas de empacotamento, o sistema PVM também possibilita às tarefas, através da primitiva *pvm_setopt()*, definir o tipo de roteamento usado para o envio e recebimento das mensagens: comunicação indireta via *daemons* do PVM utilizando o protocolo UDP ou comunicação direta entre as tarefas via protocolo TCP. Normalmente, uma tarefa origem que deseja enviar uma mensagem para uma tarefa destino roteia esta mensagem para o Pvmd da sua máquina local. O Pvmd local, então, repassa a mensagem ao Pvmd da máquina destino e este entrega a mensagem à tarefa destino, como mostra a Figura 4.1. No entanto, também é possível para as tarefas de uma aplicação PVM estabelecerem uma comunicação direta através de uma conexão TCP, e assim evitarem a sobrecarga que a comunicação via Pvmd acaba gerando. Informações mais detalhadas sobre as formas de empacotamento e de roteamento das mensagens são encontradas em [GBD+94a, GBD+94b, CEN97].

CAPÍTULO 5

Medições da Comunicação PVM

Este capítulo apresenta as medições efetuadas no sistema de memória, nos protocolos de comunicação TCP/IP, e no PVM em cada um dos ambientes de baixo custo estudados. Essas medições têm por objetivo avaliar o desempenho desses componentes e seus impactos no desempenho do sistema de comunicação disponibilizado pelo PVM sobre o sistema operacional Linux. Possibilitando assim, demonstrar a viabilidade de se implementar processamento paralelo distribuído através do PVM sobre o Linux em ambientes considerados de baixo custo.

O sistema de memória das arquiteturas estudadas é avaliado porque grande parte do processamento dos protocolos de comunicação TCP/IP e do PVM consiste na movimentação de dados de um armazenador para outro armazenador na memória. Os protocolos de comunicação TCP e UDP são avaliados porque o PVM utiliza o serviço de transferência de dados desses protocolos para disponibilizar o seu sistema de comunicação baseado em passagem de mensagens. Assim, comparando as medições efetuadas nos sistemas de memórias e nos protocolos de comunicação TCP/IP com as medições efetuadas no sistema de comunicação do PVM, é possível efetivamente avaliar o desempenho da comunicação PVM sobre o sistema operacional Linux em ambientes de baixo custo.

A Seção 5.1 define o método de medição usado para efetuar as medições dos protocolos de comunicação TCP/IP e do sistema de comunicação do PVM. A Seção 5.2 descreve a importância do sistema de memória; mostra as medições efetuadas e analisa o desempenho dessas arquiteturas. A Seção 5.3 descreve detalhadamente o mecanismo da comunicação via Soquetes usado para efetuar as medições do desempenho dos protocolos TCP e UDP. A Seção 5.4 mostra e analisa as medições efetuadas da comunicação entre máquinas via Soquetes. A Seção 5.5 mostra e analisa as medições efetuadas da comunicação local via

Soquetes. A Seção 5.6 descreve detalhadamente o mecanismo da comunicação com PVM usado para efetuar as medições do desempenho do PVM. A Seção 5.7 mostra e analisa as medições efetuadas da comunicação entre máquinas com PVM. A Seção 5.8 mostra e analisa as medições efetuadas da comunicação local com PVM. A Seção 5.9 apresenta sugestões aos programadores que pretendem implementar aplicações paralelas distribuídas com PVM para serem executadas em ambientes de baixo custo. Finalmente, a Seção 5.10 apresenta uma proposta para melhoria do desempenho do sistema de comunicação do PVM.

5.1 Técnica de Medição

Antes de descrever o método de medição usado durante as medições, alguns dos conceitos já vistos anteriormente, e que são usados no decorrer das medições são relembrados. Uma *mensagem* é um bloco de dados e corresponde a um certo número de bytes (octetos) transmitidos pelo programa do usuário. A *vazão efetiva* é definida como o número de bytes transferidos por unidade de tempo, e é a taxa de transmissão medida considerando-se somente o conteúdo das mensagens e ignorando as outras informações como cabeçalho, etc. A *latência* é o tempo consumido entre a emissão de uma mensagem pelo processo transmissor do usuário e o recebimento desta mensagem pelo processo receptor. O protocolo de transporte TCP transfere *segmentos*, o protocolo de rede IP transfere *datagramas* e a Ethernet tranfere *quadros*.

As medições foram efetuadas em horários de tráfego reduzido na rede, à noite e em fins de semana, para que os resultados das medições não apresentassem informações incorretas em virtude de tráfego temporário na rede.

As medidas de vazão efetiva foram efetuadas pelo padrão de medição de comunicação ponto-a-ponto, chamado de *método ping-pong*. As medidas envolvem duas máquinas A e B, e consistem da medida do tempo decorrido entre o envio de uma mensagem de A para B e a recepção da cópia desta mesma mensagem enviada de B para A. A latência, ou seja, o tempo de viajem de uma mensagem, é obtido dividindo-se por dois a medida de tempo de ida-e-volta, onde a *ida* é o envio de uma mensagem de A para B e a *volta* é o retorno desta mensagem. Para mais detalhes sobre modelos de comunicação ponto-a-ponto e

ponto-multiponto, e técnicas de medições veja [NN95, Nev96].

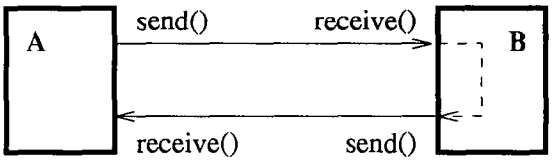

Figura 5.1: Método de Medição.

O método ping-pong é necessário porque as primitivas de envio *send()* são assíncronas, ou seja, não-bloqueantes: a função *send()* retorna assim que o conteúdo da mensagem é copiado para um armazenador do sistema operacional (SO), independentemente da transmissão da mensagem ter sido iniciada ou não. Por outro lado, as primitivas de recepção são bloqueantes: o processo que executa *receive()* fica bloqueado até que todos os bytes especificados no parâmetro de ativação do *receive()* sejam copiados pelo SO para o armazenador do processo do usuário. Assim, quando o processo na máquina A executar o par *send()+receive()*, o intervalo medido compreende o envio assíncrono da mensagem A e a recepção síncrona por A da cópia enviada por B. Se apenas a duração do *send()* for considerada, o tempo medido será menor que o tempo efetivamente decorrido na comunicação entre A e B.

O transporte de dados é efetuado pelos protocolos UDP, TCP usando IP sobre a Ethernet. A fim de se evitar flutuações nas medidas de vazão efetiva decorrentes do amortecimento causado pelas filas internas à implementação dos protocolos, as medidas são efetuadas com a transmissão de **N** mensagens de **M** bytes, totalizando 256 Kbytes para as medições no Soquete e 1 Mbytes para as medições no PVM. Estes volumes de dados correspodem a 4 e 16 janelas de transmissão do TCP, de 64 Kbytes por janela, e atenuam os efeitos associados à janela de transmissão, impedindo que estes mascarem as medidas. O valor de cada medida é a média de quatro vezes o tamanho de cada bloco de dados transmitido, totalizando 1 Mbytes para soquetes e 4 Mbytes para PVM.

As medidas foram efetuadas com mensagens de tamanhos variando, em potências de 2, de 16 bytes a 63000 bytes para as medições no Soquete com UDP e até 128 Kbytes para as medições no Soquete com TCP e no PVM. Além desses valores, o tempo de transmissão de mensagens com 1200, 1300, 1400, 1500 e 1600 bytes foi também medido. Estes tamanhos

permitem observar os efeitos da fragmentação de datagramas no transporte de mensagens com tamanho próximo da "Unidade Máxima de Transferência" (UMT) da Ethernet, que é de 1500 bytes.

Os sistemas avaliados são três pares de PCs interligados por Ethernet e a comunicação inter-processos nas máquinas locais. Um par consiste de PCs com pentium P75 (75 MHz, 32 Mbytes de RAM, 256 Kbytes de cache secundária) interligados por cabo coaxial, e que executam Linux 2.0.33. Estas máquinas são chamadas de 'P75' nos experimentos. A Figura 5.2 ilustra a rede com máquinas P75.

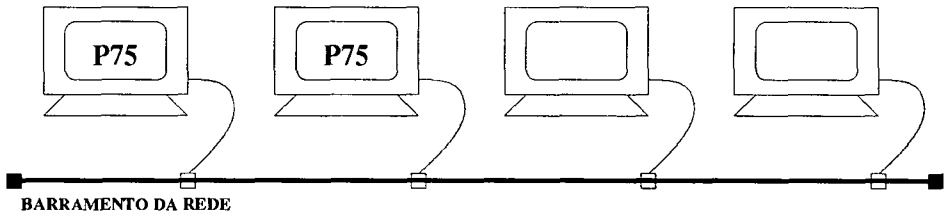

Figura 5.2: Rede com Máquinas P75.

O segundo par de máquinas também consiste de PCs com Pentium P90 (90 MHz, 64 Mbytes de RAM, 256 Kbytes de cache secundária) interligados por par trançado e um concentrador IBM-8224. Estas também executam Linux 2.0.33 e são chamadas de 'P90' nos experimentos. A Figura 5.3 ilustra a rede com máquinas P90.

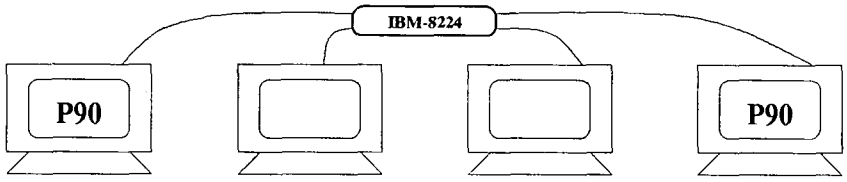

Figura 5.3: Rede com Máquinas P90.

O terceiro par consiste de um Pentium P166 (166 MHz, 64 Mbytes de RAM, 512 Kbytes de cache secundária) e de um Pentium-Pro 200 (200 MHz, 64 Mbytes de RAM, 512 Kbytes de cache secundária), interligadas por um par trançado e um concentrador IBM-8224. No terceiro par, as medidas são sempre efetuadas nas duas máquinas e chamadas de 'P166' e 'P200'. Esta máquinas executam Linux 2.0.0. Em todas máquinas está instalada a versão 3.3.11 do PVM. Todos os programas de teste foram compilados com pelo GCC-2.7.2, com a linha de comando "gcc -O2 -static". A função *gettimeofday()* foi usada para medir o

tempo da transmissão de **N** mensagens de **M** bytes.

Para fins de comparação dos três sistemas acima avaliados, medições também foram efetuadas em um sistema formado por duas máquinas IBM-RS6000 380 e 390 com sistema operacional AIX 4.1 interligadas por par trançado e um concentrador IBM-8224.

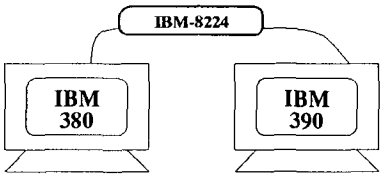

Figura 5.4: Rede com Máquinas IBM.

5.2 Sistema de Memória

Como discutido em [KP92, Dru96], uma grande parcela do processamento efetuado pelos protocolos TCP/IP consiste da movimentação de dados entre os armazenadores em memória e no cômputo do *checksum* do campo de informação dos segmentos TCP. Por exemplo, o envio de uma mensagem envolve (1) a preparação da mensagem pelo processo remetente, (2) a cópia da mensagem para um armazenador do SO, (3) o cálculo da paridade pelo TCP, (4) a cópia do armazenador para o adaptador de interface. Na recepção da mensagem, um processo similar ocorre na ordem inversa. Medidas efetuadas em DecStations 5000 indicam que aproximadamente 40% do tempo de processamento do TCP é consumido no cálculo do *checksum* e que outros 20% a 30% são consumidos no gerenciamento dos armazenadores para mensagens, os 'mbufs' [KP92].

O desempenho do sistema de memória, consistindo das caches primária e secundária, mecanismo de acesso direto à memória, barramentos de memória e de entrada/saída, limita a velocidade com que as cópias listadas acima e cômputo do *checksum* podem ser efetuados. Portanto, um primeiro passo na avaliação de um sistema de comunicação consiste na aferição das taxas máximas de transferência através do sistema de memória.

O desempenho dos sistemas de memória das classes de máquina estudadas, incluindo as versões do sistema operacional, pode ser comparado através da medida da taxa de transferência entre dois armazenadores em memória. A transferência consiste numa simples cópia entre dois vetores em memória, implementado na linguagem C com um comando

for() que move caracteres das posições definidas por um apontador para posições definidas por outro apontador. As curvas da Figura 5.5 mostram dois tipos de comportamento: execução com início "a frio" e com partida "a quente", definidos abaixo.

As curvas com sufixo '-frio', que representam a execução "a frio", mostram o desempenho das máquinas com o efeito das faltas nas caches que ocorrem na primeira execução do *for()*, quando a cache ainda não contém nenhum dos octetos a serem transferidos. As curvas com sufixo '-med', que representam a execução "a quente", mostram o desempenho das máquinas com a taxa de transferência medida sem o efeito ocasionado pelas faltas nas caches: a primeira execução do *for()*, a qual preenche a cache com os octetos a serem transferidos, é ignorada, e a medida é obtida pela média das quatros execuções subseqüentes do *for()*. A curva com a vazão efetiva máxima da Ethernet também é mostrada, para efeitos de comparação.

A primeira constatação é a de que, nas quatro máquinas, a vazão com a cache inicialmente vazia é menor do que com a cache inicialmente cheia. Esta diferença é uma boa indicação quanto ao custo das faltas nas caches primária (blocos de até 8 Kbytes) e secundária (blocos de 8Kbytes a 256-512 Kbytes). No P200, a diferença fica entre 2 a 42%. Para o P166 a diferença é de 5-28%, para o P90 de 6-27%, e de 10-25% para o P75. A diferença mostrada entre as vazões do sistema de memória com caches cheias e vazias aumenta com o tamanho dos blocos transferidos. Isso acontece aos blocos maiores que 8 Kbytes, porque estes excedem o tamanho das caches de dados dos processadores. A diferença entre caches cheias e vazias aumenta novamente quando blocos com tamanhos próximos aos da cache secundária são transferidos (não mostrado na figura).

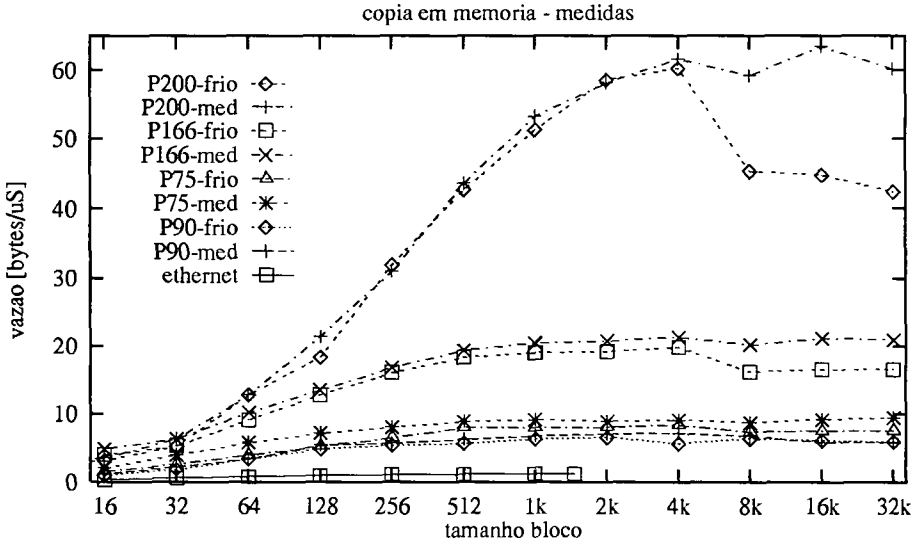

Figura 5.5: Desempenho do sistema de memória.

A segunda observação é que o sistema de memória com o processador P75 atinge uma vazão maior que o sistema com o processador P90. O P75 é 25-28% mais rápido que o P90, com a cache inicialmente vazia, e 34-36% mais rápido com a cache inicialmente cheia.

A vazão dos sistemas de memória avaliados é de aproximadamente 3 a 7 vezes a vazão máxima teórica da Ethernet (1,25 bytes por microsegundo) para os processadores mais lentos (P75 e P90), de 4 a 16 vezes no P166 e 5 a 48 vezes no P200.

No caso dos processadores mais lentos, o desempenho do sistema de memória é um fator limitante na comunicação via rede local Ethernet quando há de se considerar que, além do tráfego de rede, o sistema de memória atende não somente ao processador, como também aos demais periféricos (discos, mouse, teclado e vídeo, por exemplo).

A vazão do sistema de memória do P166 é mais do que suficiente para a comunicação via Ethernet a 10 Mbps. Caso esta máquina seja usada numa rede mais rápida, como Ethernet de 100 Mbps, o sistema de memória poderia se tornar um fator limitante no desempenho da comunicação. Já o sistema de memória do P200 não limitaria o desempenho da comunicação a 100 Mbps.

5.3 Comunicação com Soquetes

Em sistemas Unix, e no Linux em particular, a interface de soquetes permite uma forma despojada de comunicação entre os processos através do mecanismo de transporte implementado pelos protocolos TCP/IP: os quadros Ethernet transportam os datagramas IP, que por sua vez transportam os segmentos TCP ou UDP que contêm dados.

A curva da vazão máxima da Ethernet que aparece na Figura 5.5 e nas demais figuras que mostram o desempenho da comunicação com soquetes TCP/UDP foi obtida atráves do cálculo de utilização do canal de 10 Mbps. Como mostrado na Seção 2.4, um quadro Ethernet possui um cabeçalho com 8 octetos de preâmbulo, 12 octetos de endereçamento, 2 octetos para tamanho/tipo e 4 octetos de teste de correção "CRC", além do campo de informação com tamanho entre 46 e 1500 octetos. Entre o envio de dois quadros consecutivos deve haver um intervalo de tempo de 9,6 microsegundos, chamado de hiato e equivalente a 12 octetos. A partir desses valores, é possível estabelecer uma fórmula para cálcular a utilização do canal que é

$$U = \frac{i}{8 + 12 + 2 + \max(46, i) + 4 + 12}$$

onde i é o tamanho do campo de informação. Portanto, a utilização máxima possível do canal é de 0,975 com quadros que transportam 1500 octetos no campo de informação. Já quando os quadros transportam apenas 1 octeto de dados, sendo o campo de informação composto por 1 octeto de dados e 45 octetos em zero, utiliza-se apenas 0,012.

O limite de 1,25 bytes por microsegundo, equivalente a 10 Mbps, é mostrado nas figuras da comunicação com soquetes TCP/UDP pela linha horizontal tracejada.

As medidas de vazão na comunicação através de soquetes foram efetuadas após o estabelecimento da comunicação entre os dois processos. O tempo de transmissão é o intervalo decorrido entre a invocação do *send()* e a terminação do *receive()*. No caso da comunicação via soquetes com TCP foi usada a função ***int send(int s, const void *msg, int len, unsigned int flags)*** para transmitir uma mensagem para um outro

soquete, e a função ***int recv(int s, void *buf, int len, unsigned int flags)*** para receber mensagens de um soquete. No caso da comunicação via soquetes com UDP foi usada a função ***int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen)*** para transmitir uma mensagem a um outro soquete, e a função ***int recvfrom(int s, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen)*** para receber mensagens de um soquete.

Flags é o parâmetro responsável pela forma como as funções *send()* e *receive()* se comportam durante o envio e o recebimento dos dados. De acordo com o manual do Linux, este parâmetro pode ser do tipo MSG_OOB, MSG_PEEK ou MSG_WAITALL para a função *receive()* e do tipo MSG_OOB e MSG_DONTROUTE para a função *send()*. Estes valores significam

MSG_OOB processa dados Out-of-band.

MSG_PEEK copia os dados que se encontram no armazenador do *recv()* sem removê-los.

MSG_WAITALL mantem a função *recv()* bloqueada até todos os bytes especificados pelo parâmetro *len* sejam copiados.

MSG_DONTROUTE é usado por programas de roteamento e diagnósticos.

Apesar da função *recv()*, usada na comunicação via soquetes com TCP, ser bloqueante, não há a garantia de que permaneça bloqueada até que todos os octetos especificados pelo parâmetro *len* sejam copiados para o armazenador do processo do usuário. Como o valor MSG_WAITALL não é definido no Linux, a solução usada para garantir que a função *recv()* permaneça totalmente bloqueante e não influencie nas medições é mantê-la em loop até que cada mensagem seja completamente recebida. Já nas medições efetuadas nas máquinas IBM que executam o sistema operacional AIX, o valor MSG_WAITALL foi usado.

A comunicação com soquetes permite que partes de sua configuração, como o tamanho dos armazenadores de envio e recebimento, sejam manipuladas pelas aplicações. Isso é feito através da função ***int setsockopt(int s, int level, int optname, const void *optval, int optlen)***. Os armazenadores do soquete no Linux têm tamanho de 32 Kbytes

podendo estes serem configurados em 64 Kbytes. Nas medições efetuadas com soquetes TCP e UDP os tamanhos dos armazenadores de envio e recebimento foram alterados para 64 Kbytes. Com isso, tornou-se possível medir a vazão da comunicação soquetes com UDP para as mensagens de até 63000 bytes. Veja [Ste90] para mais informações sobre configuração de Soquete.

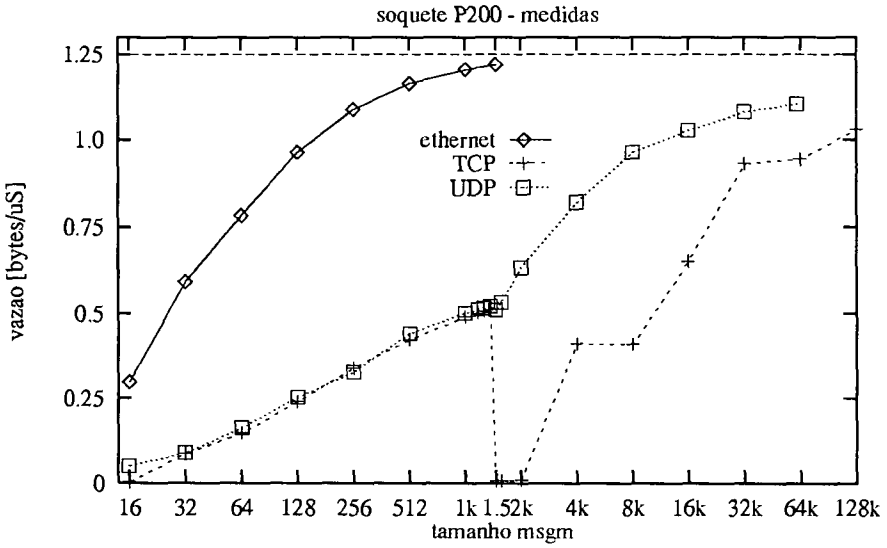

Figura 5.6: Desempenho da comunicação com soquetes TCP e UDP na máquina P200.

A Figura 5.6 mostra as curvas de vazão da comunicação soquetes com TCP e UDP na P200. Elas possuem um comportamento semelhante até 1400 bytes. A partir daí, as curvas apresentam comportamentos diferentes por motivos específicos a cada tipo de comunicação e que são discutidos mais detalhadamente adiante. Na comunicação soquete com TCP um dos motivos da mudança no comportamento da curva de vazão se deve a necessidade de mais de um quadro Ethernet no transporte de cada mensagem. Como dito anteriormente, um quadro Ethernet de tamanho máximo transporta um datagrama com tamanho total de 1500 octetos. Visto que o cabeçalho dos datagramas IP, sem os campos opcionais, ocupa 20 octetos, logo um datagrama é capaz de transportar um segmento de 1480 octetos. O segmento TCP, também supondo um cabeçalho sem os campos opcionais, ou seja, ocupando 20 octetos, transporta uma mensagem com 1460 octetos.

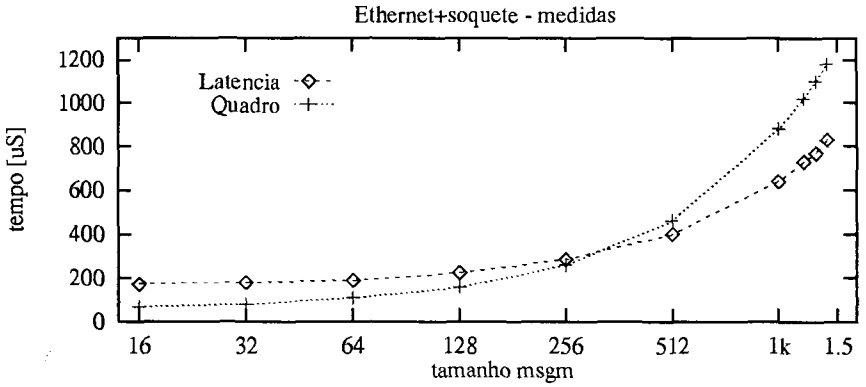

Figura 5.7: Custo da transmissão de um quadro Ethernet.

A Figura 5.7 mostra o lapso de tempo (latência) que decorre entre a invocação da primitiva *send()* e o início da transmissão do quadro pelo adaptador de interface de rede nas P75. Os valores da Figura 5.7 foram obtidos com o auxílio de um osciloscópio *Tektronix modelo Tds 210*, 60MHz de faixa de passagem e 1G amostras/segundo. Inicialmente, o processador escreve um octeto **0x00** na porta paralela do PC. Imediatamente antes que o *send()* seja executado, o processador escreve o octeto **0xFF** na porta paralela, disparando, assim, a medida de tempo pelo osciloscópio. O intervalo entre a alteração na porta paralela e o aparecimento de sinais no cabo coaxial é a latência na transmissão de um quadro através da Ethernet. Esta latência envolve o custo de processamento pelo sistema operacional, ou seja, interface de soquete e protocolos TCP/IP; e o custo da ativação do adaptador de interface. A Figura 5.7 também mostra a duração de cada quadro no meio físico, ou seja, o tempo de transmissão do quadro através do cabo coaxial. As medidas para mensagens maiores que a **UMT**, 1500 octetos, não são confiáveis porque cada mensagem é segmentada e a técnica de medida empregada não permite associar os sinais no meio físico às mensagens transmitidas pelos computadores. As medições foram efetuadas na P75 por ser teoricamente a máquina com o processador mais lento das usados nos experimentos, e conseqüentemente a que apresenta uma maior latência. O tempo total é a soma dos dois valores para cada tamanho de quadro. Para as mensagens pequenas, de até 256 octetos, a latência é dominante; já para mensagens grandes, o tempo de transmissão é dominante.

5.4 Comunicação Entre Máquinas com Soquetes

Esta Seção mostra e discute as medições efetuadas da comunicação via soquetes com TCP e UDP nas máquinas P75, P90, P166, P200 e IBM. A comunicação com soquetes entre máquinas tem a finalidade de mostrar o custo do processamento dos protocolos UDP e TCP nas arquiteturas estudadas e a influência que a tecnologia de rede Ethernet tem no desempenho da comunicação inter-processos.

A Figura 5.8 mostra as curvas de vazão da comunicação soquetes TCP para as quatro máquinas tipo PC estudadas. Nas medições, a função *recv()* foi mantida em loop até que a quantidade de dados especificada pelo parâmentro *len* equivalente ao tamanho da mensagem fosse copiada para o armazenador do usuário.

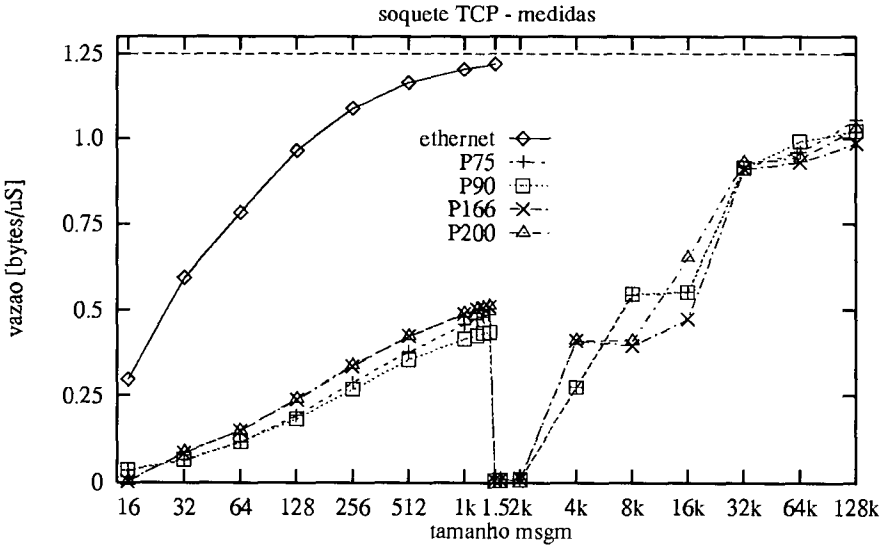

Figura 5.8: Desempenho da comunicação com soquetes TCP.

As curvas de vazão para as quatro máquinas são similares e apresentam basicamente três fases de desempenho. A primeira fase corresponde as mensagens que possuem tamanhos de até 1460 bytes, o equivalente a um segmento TCP. Nesta fase, cada mensagem é enviada em apenas um quadro e a vazão corresponde a uma fração da curva de vazão máxima da Ethernet. A diferença entre as vazões do soquete e a da Ethernet é causada pelo processamento dos protocolos, e pelo "Controle de Acesso ao Meio" (CAM) da Ethernet. Este processamento envolve tarefas como as cópias entre os armazenadores e o cálculo de paridade do segmento [KP92, Dru96].

A segunda fase compreende as mensagens que possuem tamanhos entre 1500 bytes a 16 Kbytes. Nesta fase, as medições sofrem a influência das heurísticas implementadas no protocolo TCP, e que tem por objetivo evitar a *Síndrome da janela desnecessária.* Esta síndrome as vezes ocasionava um sério problema de desempenho na comunicação entre processos remotos que operam em velocidades diferentes. O problema ocorria quando um aplicativo transmissor gerava dados numa velocidade superior ao que o aplicativo receptor pudesse tratar. Assim, a partir do momento que o armazenador do TCP receptor estava totalmente preenchido, este informava ao TCP transmissor que não havia área disponível no seu armazenador, e o TCP transmissor parava de transmitir segmentos. Contudo, quando o aplicativo receptor extraía, por exemplo, um octeto de dados do armazenador cheio do TCP receptor, um octeto de espaço tornava-se disponível. O TCP receptor, então, gerava uma confirmação informando ao TCP transmissor que havia espaço disponível no seu armazenador. À medida que o TCP transmissor sabia da existência de espaço disponível no armazenador do receptor, ele gerava segmentos que possuiam pequena quantidade de dados. Logo, havia um consumo de largura de banda de rede e uma sobrecarga computacional desnecessários em virtude da transmissão de segmentos pequenos.

Com a finalidade de evitar a síndrome da janela desnecessária, foram incluídas heurísticas no protocolo TCP. A heurística incluída na parte do protocolo TCP responsável pela transmissão dos segmentos impede o envio de pequena quantidade de dados em cada segmento. Para isso, o TCP retarda o envio de um segmento até que se tenha acumulado uma quantidade razoável de dados gerados pelo aplicativo transmissor. A heurística incluída na parte do protocolo TCP responsável pela recepção retarda o envio da confirmação dos segmentos recebidos. Em geral, o TCP receptor mantém um registro interno do tamanho da janela e retarda a confirmação dos segmentos recebidos até que o aplicativo receptor tenha extraído uma determinda quantidade de dados. Isto evita o envio de informações sobre o tamanho da janela quando esta não é suficientemente grande para ser informada.

As medições da comunicação com soquetes TCP para as mensagens que geram segmentos pequenos, e que utilizam mais de um quadro Ethernet sofrem os efeitos das heurísticas

descritas acima. A Figura 5.9 mostra os efeitos dessas heurísticas na medição da transmissão das mensagens de 1500 bytes nos ambientes avaliados.

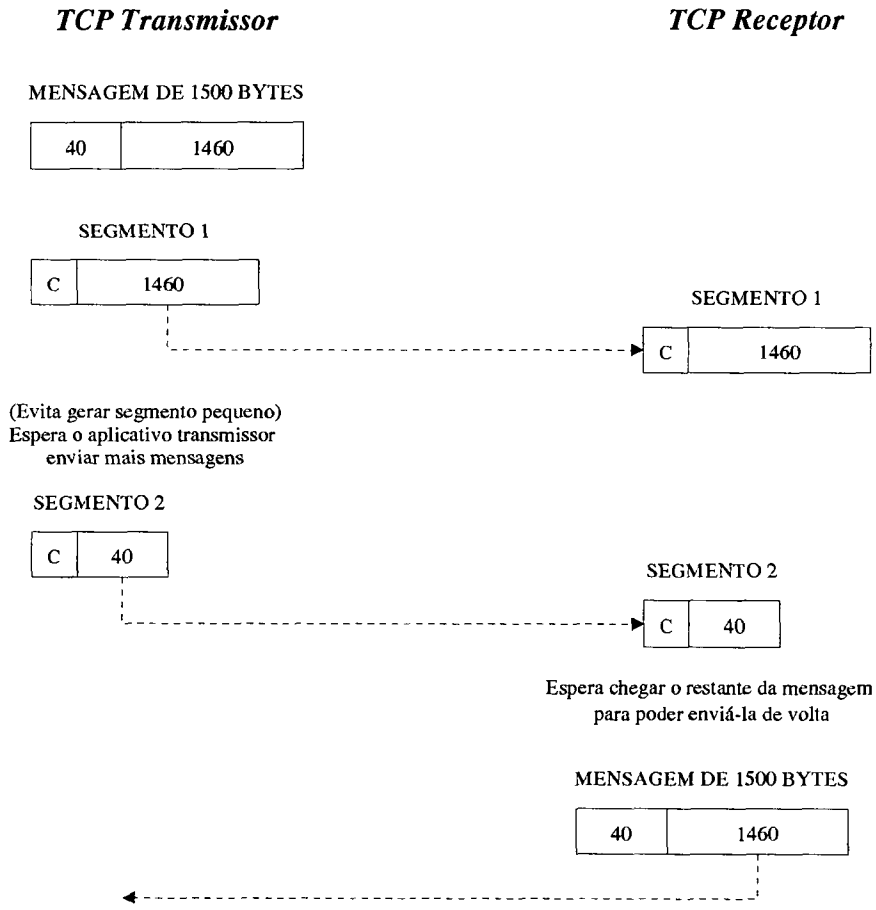

Figura 5.9: Efeitos das heurísticas no desempenho da comunicação com soquetes TCP.

Cada mensagem de 1500 bytes gera dois segmentos TCP de 1460 bytes e 40 bytes de dados respectivamente. O TCP transmissor envia primeiro o segmento (1). Visto que a quantidade de dados referente ao restante da mensagem enviada , 40 bytes, é pequena, o TCP transmissor retarda o envio dos 40 bytes restantes da mensagem, esperando que o aplicativo transmissor envie outras mensagens. Como a próxima mensagem somente será enviada quando a atual mensagem retornar do aplicativo receptor, o TCP transmissor é obrigado a enviar o segmento (2) quando o tempo de retardo de envio de segmentos expira. Isto compromete o tempo gasto no envio da mensagem, proporcionando um aumento na latência e por conseqüência a baixa vazão apresentada pelas mensagens que geram segmentos pequenos. Veja [CS94] para maiores informações sobre as heurísticas que evitam a *Síndrome da janela desnecessária.*

A terceira fase da Figura 5.8 mostra as curvas de vazão das quatro máquinas mais próxima da vazão máxima da Ethernet. Isto ocorre porque o tamanho das mensagens é maior, o que acaba gerando uma quantidade maior de segmentos. Como estes segmentos têm tamanho próximo da UMT, os quadros Ethernet são emitidos com o campo de informação completo, o que possibilita uma melhor utilização da largura de banda da rede. Além disso, a quantidade de segmentos gerado por cada mensagem torna eficiente o uso da janela de transmissão. Assim, esses fatores são responsáveis pela alta vazão medida.

As medidas de vazão obtidas via soquete UDP são mostradas na Figura 5.10. O protocolo UDP, ao contrário do TCP, não possui o mecanismo de janela de transmissão usado no controle de fluxo. As mensagens enviadas são segmentadas e passadas ao protocolo IP, que tem a função de gerar os datagramas. Assim como o TCP, o UDP se baseia na idéia de encapsulamento, ou seja, no UDP cada mensagem (cabeçalho UDP + segmento) deve ser transportada em um único datagrama. Cada datagrama também deve ser transportado por um único quadro de rede. Permitindo assim, uma transmissão mais eficiente na rede física.

O UDP oferece um serviço de comunicação não-confiável. Logo, os segmentos enviados não necessitam de confirmação. Assim, o número de quadros necessários na comunicação é menor, o que proporciona um aumento na vazão das mensagens.

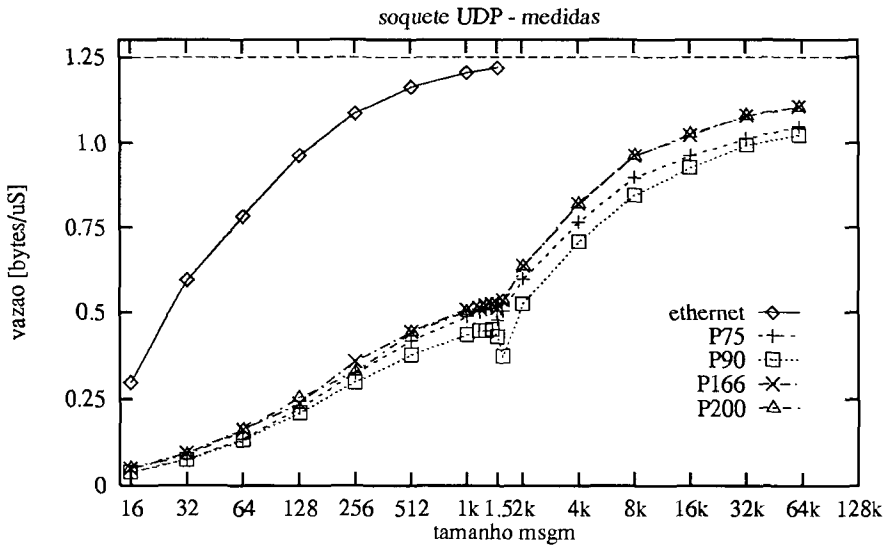

Figura 5.10: Desempenho da comunicação com soquetes UDP.

As curvas de vazão tanto da comunicação soquete com UDP quanto TCP nas máquinas

PCs mostram desempenhos semelhantes para as mensagens que podem ser transportadas por um único quadro. Como mostram as Figuras 5.8 e 5.10, nas mensagens de 1400 bytes, que é o maior tamanho de mensagem usado nas medições, transportado em um único quadro, a vazão é de aproximadamente 0.5 bytes/μS, o que corresponde a 40% da vazão máxima da Ethernet.

As curvas de vazão soquete com UDP apresentam comportamentos semelhantes para todos os tamanhos de mensagem. Com isso, é possível se notar a influência que cada arquitetura de PC tem na curva de vazão. Com destaque para a curva P75 que apresenta uma vazão melhor do que a P90, confirmando o havia sido detectado durante a medição do desempenho do sistema de memória.

A Tabela 5.1 mostra o número de quadros Ethernet transmitidos e recebidos para a P75. Nas conexões TCP e UDP, o número de quadros aumenta significativamente nas mensagens que possuem tamanhos de 1500 bytes. Isto ocorre porque cada mensagem precisa de 2 quadros para ser transportada. Um quadro com os 1460 bytes iniciais da mensagem e outro quadro com os 40 bytes restantes. Já o número de quadros na conexão TCP é maior porque além dos quadros usados no transporte das mensagens, há a necessidade de confirmação dos segmentos enviados.

A diferença entre o número de quadros transmitidos/recebidos nas conexões TCP e UDP vai diminuindo à medida que o tamanho das mensagens aumenta. Isto acontece porque a quantidade de segmentos gerados por cada mensagem torna eficaz o uso do mecanismo da janela de transmissão. Como o TCP usa o método de retardo de envio de segmentos para evitar a *Síndrome de janela desnecessária* (quando o TCP passa a enviar segmentos pequenos porque o espaço disponível do armazenador do TCP receptor é pequeno), uma única confirmação enviada pelo TCP receptor ao TCP transmissor confirma todos segmentos até então recebidos e não confirmados.

Tamanho	TCP		UDP	
da msg.	transm.	receb.	transm.	receb.
16	16389	16388	16388	16388
32	8198	8197	8195	8195
64	4100	4099	4099	4099
128	2053	2052	2051	2051
256	1032	1031	1027	1027
512	516	515	515	515
1024	260	259	260	260
1200	224	223	223	223
1300	206	205	209	209
1400	192	191	191	191
1500	716	540	355	355
1600	667	503	331	331
2048	260	259	259	259
4096	229	228	195	195
8192	230	243	195	195
16384	215	220	195	195
32768	211	209	187	187
63000	210	205	175	175

Tabela 5.1: Número de quadros Ethernet para conexões TCP e UDP, P75.

Os valores da Tabela 5.1 foram obtidos do arquivo */proc/net/dev* durante a execução dos programas de teste. As outras classes de máquinas usadas nos experimentos apresentam valores similares.

Comparando as medições efetuadas nas máquinas IBM e P200 com soquete UDP, mostradas na Figura 5.11, verifica-se uma vazão baixa e um comportamento semelhante entre as máquinas nas mensagens de tamanho de 16 bytes a 128 bytes. A vazão da Ethernet para as mensagens de 16 bytes é de 0.298 bytes/μS, e a vazão da IBM é 14% deste valor, enquanto a P200 é 17%. Em seguida, a vazão na IBM aumenta consideravelmente, chegando a ser de 71% acima da vazão da P200 para as mensagens de 1400 bytes. A vazão da P200 é apenas 41% da vazão permitida pela Ethernet, enquanto que a da IBM é de

aproximadamente 72% da vazão da Ethernet, ou seja, o tempo consumido pelo protocolo UDP do Linux é muito maior que o consumido pelo protocolo UDP do AIX. Isto ocorre por causa da diferença de desempenho dos processadores e dos sistemas de memória das máquinas. A partir das mensagens de 1500 bytes, a curva de vazão na P200 apresenta uma inclinação semelhante a apresentada pela IBM entre as mensagens de 128 bytes até 1400 bytes, enquanto que a curva de vazão na IBM apresenta apenas mais uma pequena subida e satura até as mensagens de 63000 bytes. Entre as mensagens de 16 Kbytes e 32 Kbytes a vazão na P200 supera a IBM chegando a ser de 88% da vazão máxima de Ethernet, enquanto a IBM apresenta 84% para as mensagens de 63000 bytes.

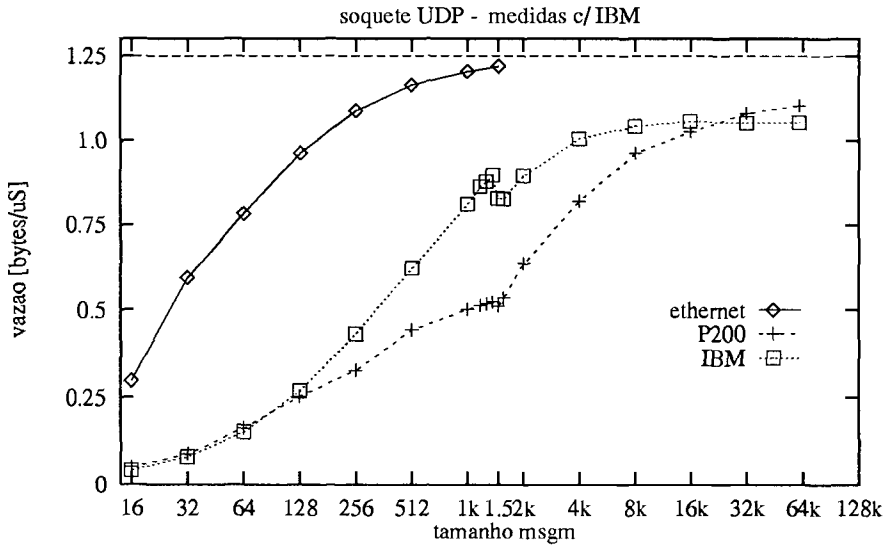

Figura 5.11: Comparação do comportamento dos soquetes UDP nas máquinas IBM.

5.5 Comunicação Local com Soquetes

A comunicação com soquetes em máquinas locais tem a finalidade de mostrar o custo do processamento dos protocolos UDP e TCP independentemente da tecnologia de rede, que neste caso é a Ethernet. Tanto o processo que envia as mensagens quanto o que recebe as mensagens executam numa mesma máquina. A comunicação com soquetes também permite medir a capacidade de processamento de cada máquina. Esta seção apresenta as medições efetuadas da comunicação com TCP e UDP nas máquinas P75 e IBM.

As medições efetuadas numa máquina P75, mostradas na Figura 5.12, apresentam uma vazão máxima de 8 bytes/μS, que é 6 vezes maior do que a vazão atingida na Ethernet. A

curva UDP apresenta sempre uma vazão maior do que a curva TCP, principalmente porque o TCP precisa da confirmação dos segmentos que são enviados tanto pela aplicação origem como pela aplicação destino. Como essas aplicações são executadas numa mesma máquina, ou seja, disputam o mesmo processador, a arquitetura da máquina e as características do sistema operacional influenciam bastante no comportamento da comunicação.

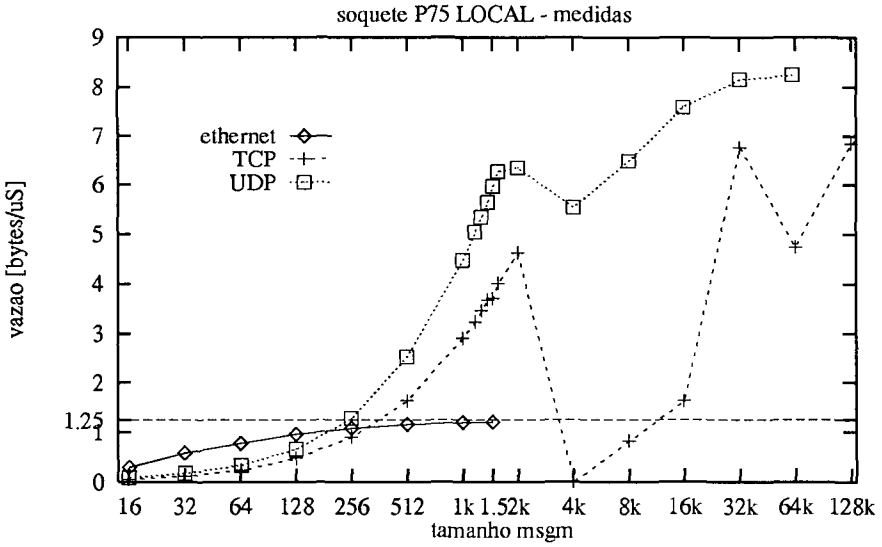

Figura 5.12: Desempenho da comunicação com soquetes UDP/TCP local P75.

As medidas TCP para as mensagens entre 1500 bytes a 16 Kbytes apresentam uma vazão baixa. Isto se deve ao mecanismo da janela de transmissão implementado pelo TCP e ao método de medição ping-pong usado nos experimentos.

O mecanismo de confirmação dos segmentos implementado pelo TCP espera durante um certo tempo pela chegada de diversos segmentos da mesma origem antes de confirmá-los. Isto evita que para cada segmento recebido pelo TCP seja enviado um segmento de confirmação. O TCP também usa um mecanismo chamado de "carona" (*piggyback*) que permite o aproveitamento do envio de um segmento com dados de uma máquina destino para uma máquina origem para confirmar os segmentos enviados pelo TCP da máquina origem e recebidos pelo TCP da máquina destino.

De acordo com o nosso método de medição, após o envio da primeira mensagem pela máquina origem, o envio de uma nova mensagem somente ocorre quando a mensagem anterior tiver retornado da máquina destino. Já na máquina destino, uma mensagem somente é enviada para a máquina origem após a máquina destino recebê-la inteiramente.

Assim, o número de segmentos que as mensagens de tamanho entre 1500 bytes a 16 Kbytes geram não é suficiente para usar o mecanismo de confirmação de maneira eficiente. O TCP fica esperando pela chegada de mais segmentos antes de enviar uma confirmação, e isto influencia no tempo gasto para receber a confirmação dos segmentos enviados e conseqüentemente no tempo gasto por cada mensagem enviada.

As medidas a partir de 32 Kbytes mostram uma curva TCP próxima da vazão atingida com UDP. Isto se deve ao aumento do número de segmentos gerados por cada mensagem, que possibilita o uso da janela de transmissão de maneira eficiente. Desta forma, múltiplos segmentos são enviados antes que uma confirmação chegue, e o número de confirmação de segmentos necessário para cada mensagem é menor.

As medições efetuadas numa máquina local IBM, veja a Figura 5.13, mostram uma vazão máxima de quase 18 bytes/μS para UDP. Uma observação interessante é que a vazão máxima do UDP na IBM é o dobro da vazão máxima UDP na P75. Já a vazão máxima do TCP é de aproximadamente 4.2 bytes/μS. A vazão do UDP é melhor nas IBM porque o hardware das máquinas IBM é melhor do que o hardware dos PCs.

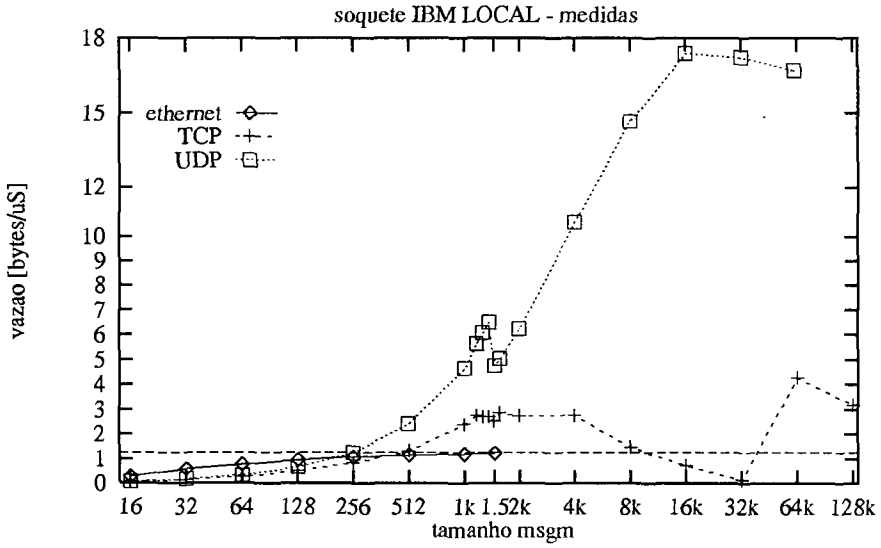

Figura 5.13: Desempenho da comunicação com soquetes UDP/TCP local IBM.

A Figura 5.14 mostra uma comparação entre as vazões obtidas no sistema de memória na máquina P75 e as vazões obtidas da comunicação local soquetes com UDP e TCP. As medições mostram que as curvas de vazão no sistema de memória são maiores do que as curvas de vazão da comunicação com soquetes. Isto significa que o sistema de memória

da P75 não é um gargalo no desempenho da comunicação local via soquetes.

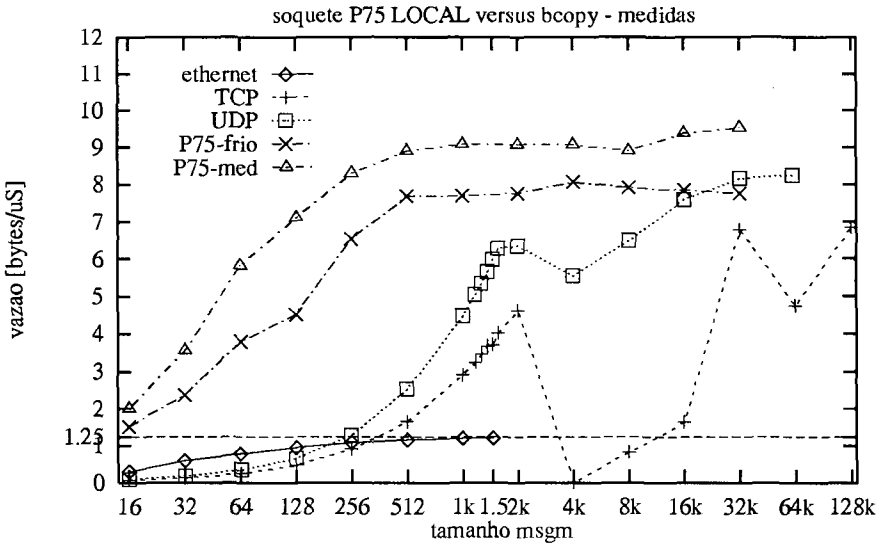

Figura 5.14: Desempenho da comunicação com UDP/TCP versus sistema de memória.

5.6 Comunicação com PVM

A comunicação suportada pelo PVM é construída sobre o mecanismo de transporte propiciado pelos protocolos TCP e UDP, através da interface de soquetes, como mostra a Figura 5.15. Assim como nos soquetes, a primitiva *pvm_send()* é assícrona e o controle retorna ao processo de usuário logo que os dados a ser transmitidos sejam copiados para um armazenador interno. Já a primitiva *pvm_receive()*, assim como nos soquetes, é síncrona e o processo receptor fica bloqueado até que todos os dados esperados sejam copiados para o armazenador do usuário.

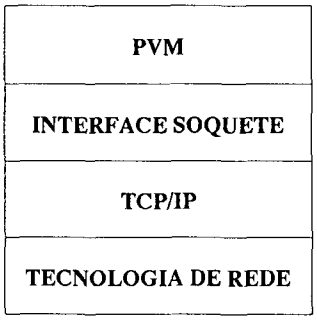

Figura 5.15: Arquitetura PVM.

Como descrito na Seção 4.5, o PVM permite três formas de empacotamento dos dados a serem transmitidos: *XDR*, *RAW* e *IN*. Normalmente, o PVM usa a forma *XDR* na

troca de dados. A forma *XDR* permite a troca de dados independente das arquiteturas que integram o sistema.

A forma *RAW* é usada na troca de dados em ambientes formados por máquinas homogêneas. Nesta forma de empacotamento, os dados não são codificados, e não há perda de tempo com a codificação.

A forma *IN* é também usada na troca de dados em ambientes formados por máquinas homogêneas. Contudo, os dados não são efetivamente copiados para o armazenador de envio durante o empacotamento. Apenas o tamanho dos dados e o ponteiro para o armazenador onde dados se encontram são copiados no armazenador de envio. Isso diminui o número de vezes que os dados são copiados, e por conseqüência o tempo de processamento consumido nas trocas de dados.

As medidas de vazão na comunicação com PVM são efetuadas considerando o tempo consumido pelas seguintes etapas:

1. Inicialização do armazenador de envio através da primitiva *init_send()* que também especifica a forma de empacotamento a ser usada.

2. O empacotamento da mensagem através da primitiva *pvm_pkbyte*.

3. O envio dos dados contido no armazenador de envio através da primitiva *pvm_send()*.

4. A cópia dos dados para o armazenador de recebimento atráves da primitiva *pvm_recv()*.

5. O desempacotamento da mensagem que está no armazenador de recebimento e sua cópia para o armazenador do processo do usuário através das primitivas *pvm_upkbyte*.

A Figura 5.16 mostra o método de medida *ping-pong* para a comunicação PVM. Assim como no soquete, o PVM possui uma primitiva chamada *pvm_setopt()* que permite a configuração de algumas opções do PVM. No caso das medições, a primitiva *pvm_setopt()* foi usada para determinar a política de roteamento das mensagens.

Uma importante consideração a ser abordada é a maneira como o PVM trata as mensagens. Normalmente, as mensagens enviadas são fragmentadas em blocos com tamanhos de 4 Kbytes. Dependendo da arquitetura utilizada, esse tamanho pode ser

alterado através da primitiva *pvm_setopt()*. No caso dos PCs, o PVM não permite a fragmentação das mensagens em tamanhos maiores que 4 Kbytes.

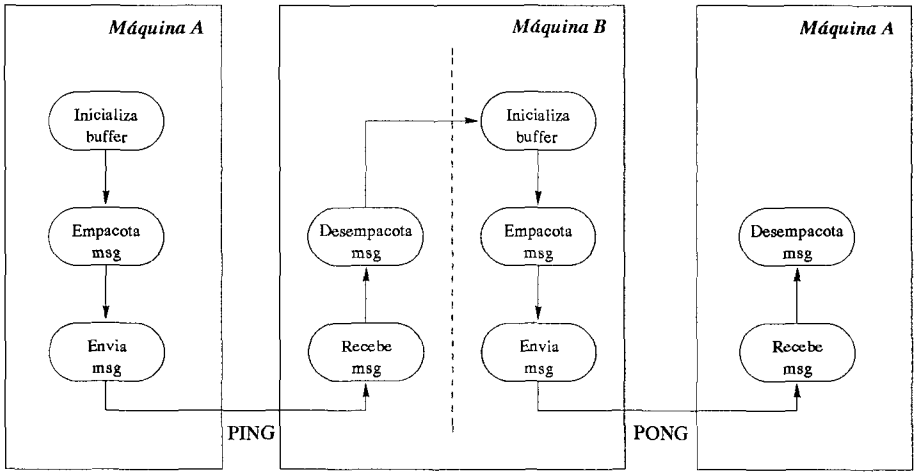

Figura 5.16: Comunicação PVM.

O PVM possibilita as tarefas definir o tipo de roteamento usado no envio e recebimento das mensagens. Normalmente, a transmissão é feita via o processo *daemon* do PVM. Neste caso, o transporte é efetuado pelo protocolo UDP, e o *daemon* do PVM é responsável por garantir a integridade do dados. Como discutido na Seção 4.4, essa forma de transferência acarreta aumento do tempo de processamento gasto na comunicação entre as tarefas, pois é necessário que cada tarefa estabeleça uma conexão com o processo *daemon* da sua máquina.

As tarefas também podem estabelecer uma comunicação direta com outra tarefa através do protocolo TCP, e assim reduzirem o *overhead* gerado pela comunicação via *daemon* do PVM.

No decorrer do texto, a comunicação com PVM usando o processo *daemon* do PVM, e portanto usando o protocolo UDP, é referênciada como comunicação PVM com UDP. Enquanto a comunicação direta entre as tarefas através do protocolo TCP é referênciada como comunicação PVM com TCP.

5.7 Comunicação Entre Máquinas com PVM

Esta Seção mostra e discute as medições da comunicação PVM com UDP e TCP efetuadas nas máquinas P75, P90, P166, P200 e IBM.

As formas de empacotamento *XDR*, *RAW* e *IN* são usadas nas medições. Contudo, as Figuras 5.17 e 5.18 exibem apenas as medições efetuadas com a forma de empacotamento *XDR*; visto que a comunicação PVM com *XDR* é a que mais tempo de processamento gasta, e conseqüentemente permite uma avaliação mais adequada da capacidade de processamento de cada máquina virtual paralela.

Uma comparação do desempenho da comunicação PVM com as três formas de empacotamento é efetuada nas máquinas P75 e IBM. Respectivamente, uma arquitetura de baixo custo e pior desempenho, e uma arquitetura de mais alto custo e alto desempenho.

A Figura 5.17 mostra as medições efetuadas da comunicação PVM utilizando o protocolo UDP e a forma de empacotamento *XDR*. A curva de vazão do Soquete com UDP da P200 foi incluída na figura com a finalidade de mostrar o custo adicional de processamento que representa o PVM na comunicação PVM com UDP.

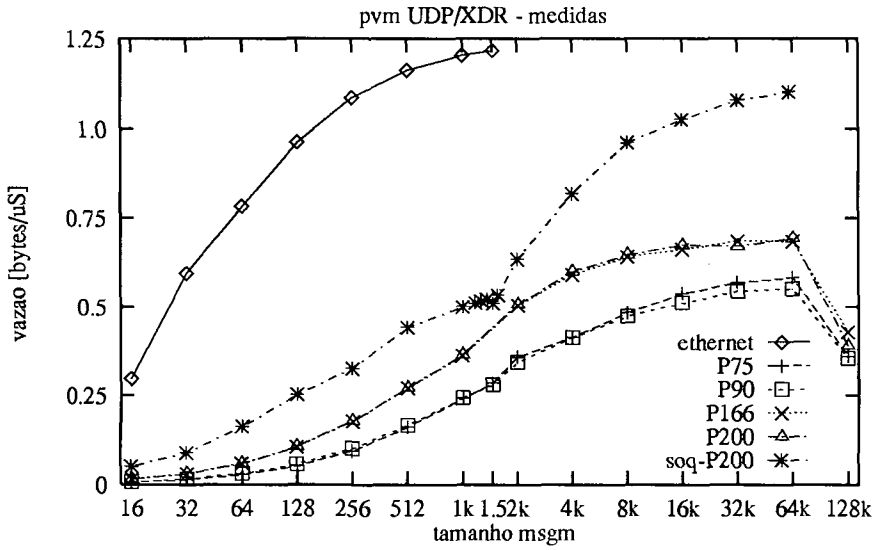

Figura 5.17: Desempenho da comunicação com PVM-UDP via Pvmd.

A primeira observação é que as curvas de vazão das quatro máquinas possuem um comportamento diferente do mostrado pela curva de Soquete. Enquanto a curva de vazão de Soquete mostra uma queda, à medida que o tamanho das mensagens transmitidas se aproxima da UMT, as curvas de vazão PVM apresentam um comportamento estável e progressivo. Para as mensagens de 1024 bytes, por exemplo, a vazão da comunicação PVM com UDP/XDR na P200 é de apenas 30% da vazão máxima da Ethernet e de 72% da vazão alcançada pelo soquete.

Uma segunda observação é que as máquinas P75 e P90 apresentam, para as mensagens de tamanho entre 1 Kbytes até 8 Kbytes, uma vazão de aproximadamente 70% da vazão alcançada pelas P166 e P200. Para as mensagens de tamanho de 16 Kbytes até 63000 bytes, a vazão é de aproximadamente 80% da vazão nas P166 e P200.

As medições efetuadas na P75 confirmam a tendência verificada durante as medições em nível de soquete e apresentam uma vazão maior do que a P90.

A última observação sobre a Figura 5.17 é que as mensagens de 128 Kbytes de tamanho mostram uma vazão muito baixa nos quatro tipo de máquinas. No caso da P75 esta vazão é de 0.3595 bytes/μS. Isto possivelmente decorre da limitação imposta pelo soquete aos seus armazenadores de envio e recebimento de mensagens. Como mencionado na Seção 5.3, os armazenadores do Soquete no Linux são normalmente de 32 Kbytes, e podem ser expandidos até 64 Kbytes através da função *setsockopt()*. Assim, um custo adicional deve ocorrer no gerenciamento dos armazenadores do soquete para que as mensagens de 128 Kbytes possam ser enviadas e transmitidas usando armazenadores de 64 Kbytes.

A Figura 5.18 mostra as medições efetuadas com transporte baseado em TCP e a forma de empacotamento *XDR*.

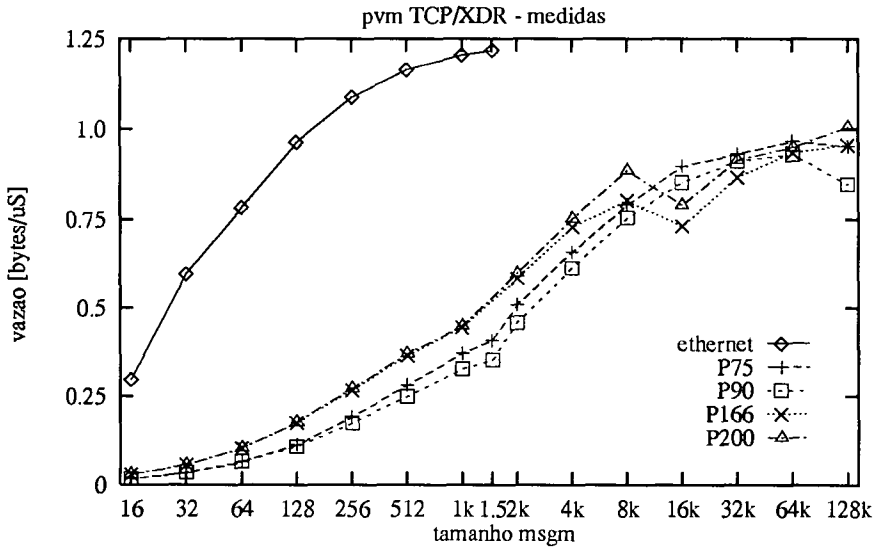

Figura 5.18: Desempenho da comunicação com PVM-TCP.

As medições efetuadas na P166 e P200 apresentam, para as mensagens de tamanho entre 512 bytes a 4 Kbytes, uma vazão entre 48% a 23% maior do que a vazão apresentada na P75 e P90. Em comparação ao soquete com UDP da P200, mostrada na Figura 5.10,

as medições da comunicação PVM com TCP apresentam vazões semelhantes para as mensagens de até 4 Kbytes de tamanho. Para as mensagens de tamanhos acima de 4 Kbytes a comunicação PVM com TCP da P200 apresenta uma vazão entre 77% a 92% da vazão alcançada atráves do soquete com UDP.

As curvas de vazão da comunicação PVM com TCP apresentam um desempenho melhor do que as curvas de vazão da comunicação PVM com UDP; porque na comunicação PVM com TCP as tarefas estabelecem uma comunicação direta, enquanto na comunicação PVM com UDP as tarefas se comunicam via *daemons* PVM.

Nas máquinas P200 a vazão PVM-TCP é 17% maior do que a vazão PVM-UDP para as mensagens de tamanho de 2 Kbytes. Já para as mensagens de tamanhos 32 Kbytes e 64 Kbytes a vazão PVM-TCP é 37% maior do que a vazão PVM-UDP.

As máquinas P90 apresentam o pior desempenho nas medições efetuadas. A vazão PVM-TCP é 33% maior do que a vazão PVM-UDP para as mensagens de tamanho de 2 Kbytes. Já para as mensagens de tamanhos 16 Kbytes a 64 Kbytes, a vazão PVM-TCP é aproximadamente 67% maior do que a vazão PVM-UDP.

Ao contrário do desempenho mostrado na comunicação PVM-UDP, as mensagens de tamanho de 128 Kbytes na comunicação PVM-TCP apresentam uma vazão próxima a vazão máxima da Ethernet. A vazão para a comunicação PVM-UDP é 0.3546 bytes/μS, enquanto na comunicação PVM-TCP é de 1.0051 bytes/μS, ou seja, a vazão da comunicação PVM-TCP é quase três vezes a vazão mostrada na comunicação PVM-UDP para as mensagens de 128 Kbytes. Essa diferença se explica pelo fato da comunicação com UDP ser através dos processos *daemons* como mostrado na Seção 4.4. Isto gera sobrecarga, pois cada mensagem enviada por uma tarefa é entregue ao *daemon* do PVM local, que a repassa ao *daemon* do PVM da máquina destino. O *daaemon* do PVM da máquina destino entrega a mensagem à tarefa destino.

A Tabela 5.2 mostra o número de quadros Ethernet transmitidos e recebidos durante a comunicação PVM-TCP e PVM-UDP. Assim como na comunicação com soquetes, os valores foram obtidos do arquivo */proc/net/dev* e representam a transmissão de um bloco de dados de 256 Kbytes. Para as mensagens de tamanho de 16 bytes até 1500 bytes, o número de quadros transmitidos e recebidos pela comunicação PVM-UDP é praticamente

o dobro da comunicação com PVM-TCP. Isto ocorre porque a comunicação PVM-UDP é feita através dos *Pvmds* do PVM. Os *Pvmds* são responsáveis por garantir a integridade dos dados trocados; para isso informações de controle sobre as mensagens recebidas são também trocadas. Assim, de acordo com os números apresentados na Tabela 5.2, as informações de controle são responsáveis por metade dos quadros efetivamente transmitidos e recebidos no envio e recebimento das mensagens até 1500 bytes.

Tamanho	TCP		UDP	
da msg.	transm.	receb.	transm.	receb.
16	16392	16390	32773	32773
32	8200	8197	16388	16388
64	4103	4101	8197	8197
128	2055	2053	4096	4096
256	1032	1030	2048	2048
512	519	517	1024	1024
1024	262	260	513	513
1500	358	356	528	528
2048	262	260	384	384
4096	273	271	384	384
8192	262	261	320	320
16384	234	231	288	288
32768	246	241	272	272
65536	242	240	264	264
131072	240	235	260	260

Tabela 5.2: Número de quadros Ethernet para conexões-PVM TCP e UDP, P75.

A seguir, as medições da comunicação PVM com as três formas de empacotamento efetuadas nas máquinas P75 e IBM são mostradas. O objetivo dessas medições é comparar o custo computacional e a influência que as formas de empacotamento têm na vazão da comunicação entre as tarefas numa arquitetura de baixo custo e desempenho como a P75, com uma máquina de alto desepenho como a IBM RS6000.

As medições da comunicação PVM com as três formas de empacotamento efetuadas na

P75 são mostradas na Figura 5.19, enquanto que as medições efetuadas na IBM RS6000 são mostradas na Figura 5.20.

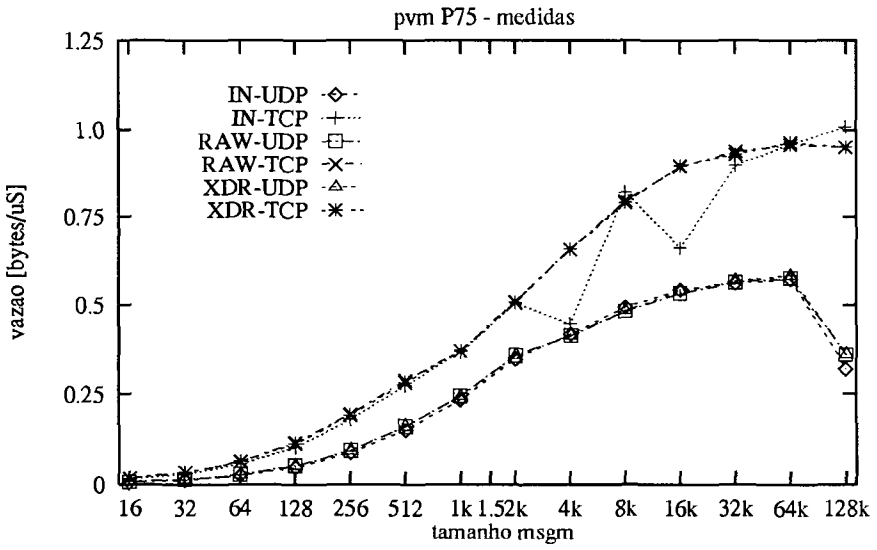

Figura 5.19: Desempenho da comunicação com PVM-TCP e PVM-UDP nas máquinas P75.

Ao contrário do que se poderia esperar, as curvas de vazão das medições efetuadas apresentam comportamentos semelhantes para as três formas de empacotamento. Ou seja, o tempo de processamento gasto em cada forma de empacotamento não influencia nas curvas de vazão obtidas na P75. No entanto, como já explicado anteriormente, o tempo de processamento gasto de acordo com o tipo de protocolo de transporte usado influencia de maneira significativa no desempenho da comunicação PVM.

Nas medições efetuadas com a forma de empacotamento *IN* foi observado que a curva de vazão da comunicação PVM com TCP não apresenta uma comportamento constante. As vazões medidas oscilam, fato não detectado na comunicação via Pvmd. A oscilação na curva de vazão IN-TCP da comunicação PVM pode ser decorrente de problemas de implementação do PVM ou do Linux. Sugere-se um estudo mais detalhado do código do PVM e do Linux para detectar as causas do problema.

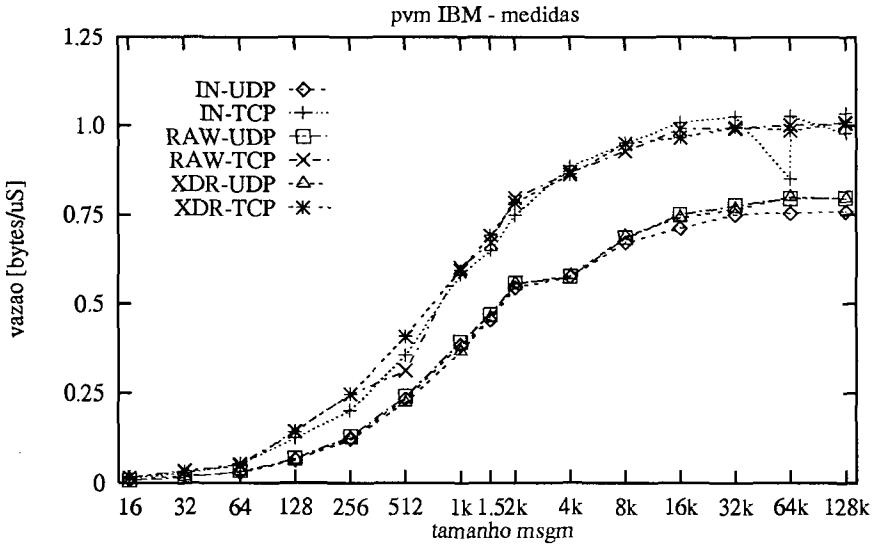

Figura 5.20: Desempenho da comunicação com PVM-TCP e PVM-UDP nas máquinas IBM.

Como já observado na P75, as curvas de vazão das medições efetuadas na IBM apresentam comportamentos semelhantes e apenas o protocolo usado no roteamento das mensagens é que influencia nas curvas de vazão obtidas.

A vazão máxima atingida na comunicação PVM com UDP nas mensagens de tamanho de 128 Kbytes é de 0.7953 bytes/μS. Para esse mesmo tamanho de mensagem, a vazão atingida com o transporte TCP é de 1.0047 bytes/μS. Já em comparação as medições efetuadas nos PCs, a vazão na comunicação PVM com TCP é praticamente a mesma nas mensagens de 128 Kbytes de tamanho.

5.8 Comunicação Local com PVM

Esta seção mostra e discute as medições efetuadas da comunicação PVM com TCP e UDP numa máquina P75 e numa máquina IBM. A comunicação com PVM em máquinas locais tem a finalidade de mostrar o custo do processamento do PVM sem a influência da rede.

A Figura 5.21 mostra as curvas de vazão da comunicação PVM-TCP e PVM-UDP numa máquina P75 para as três formas de empacotamento das mensagens. As medições efetuadas na máquina P75 mostram que, ao contrário das medições efetuadas entre máquinas, a forma de empacotamento influencia na vazão da comunicação com PVM. O tipo de protocolo de transporte usado pelo PVM também influencia na vazão.

As curvas de vazão com a forma de empacotamento *IN* são as que apresentam maior

vazão. Contudo, o desempenho da curva de vazão IN-TCP apresenta flutuações; desde um desempenho equivalente a curva de vazão IN-UDP, nas mensagens de 16 Kbytes, até uma vazão de aproximadamente 5.0 bytes/μS nas mensagens de 32 Kbytes.

Já as curvas de vazão com as formas de empacotamentos *RAW* e *XDR* apresentam desempenhos semelhantes e estáveis. A diferença entre essas curvas está apenas no protocolo usado no roteamento das mensagens. Nas mensagens entre 8 Kbytes a 128 Kbytes, por exemplo, a vazão alcançada nas curvas XDR-UDP e RAW-UDP é de apenas 65% da vazão obtida nas curvas XDR-TCP e RAW-TCP.

Uma comparação entre o custo do processamento da comunicação PVM em relação a comunicação via soquetes na P75, veja Figura 5.12, mostra uma vazão de 1.8719 bytes/μS na comunicação PVM-UDP e uma vazão de 8.2467 bytes/μS na comunicação via soquetes com UDP para as mensagens de 32 Kbytes. Esta diferença representa o custo adicional da comunicação PVM e corresponde a transferência da mensagem da tarefa *transmissora* para o *daemon* do PVM, e deste para a tarefa *receptora*.

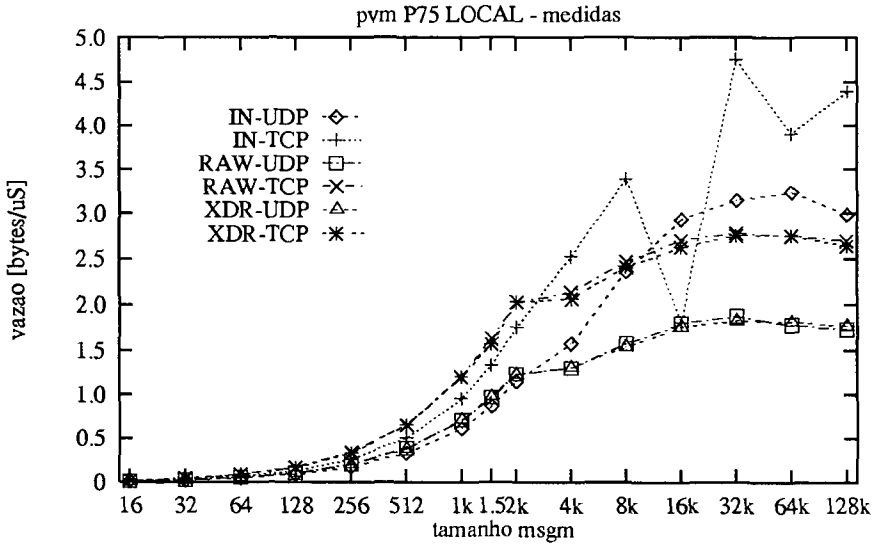

Figura 5.21: Desempenho da comunicação com PVM-TCP e PVM-UDP local P75.

As curvas de vazão da comunicação PVM-TCP e PVM-UDP local nas máquinas IBM, como mostra a Figura 5.22, apresentam comportamentos semelhantes as curvas de vazão da Figura 5.21. As medições efetuadas com a forma de empacotamento *IN* apresentam desempenhos acima das outras formas de empacotamento. No entanto, os valores não flutuam como acontece com a máquina P75. A vazão da comunicação PVM-TCP para

as três formas de empacotamento é aproximadamente o dobro da medida na P75. Já a vazão da comunicação PVM-UDP também é aproximadamente o dobro da medida na P75, exceto para a forma de empacotamento *IN*. Como era de se esperar, já que o hardware é melhor.

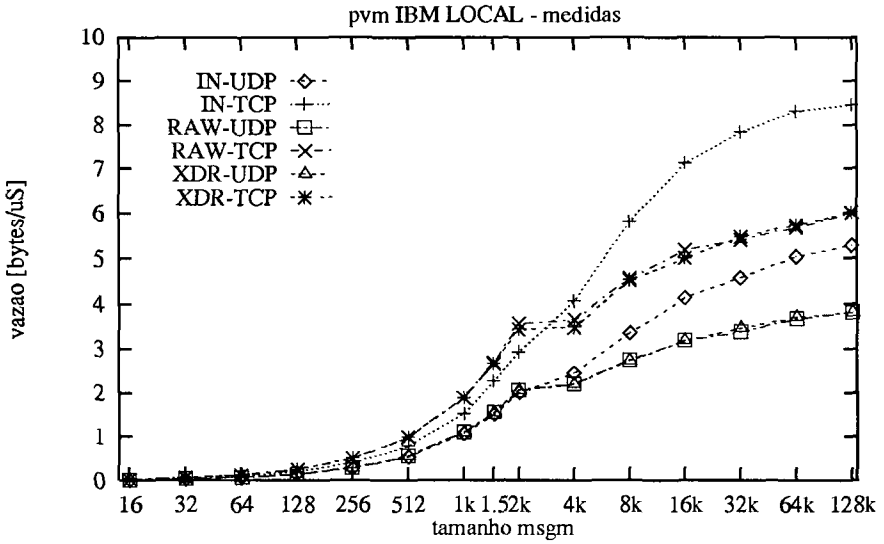

Figura 5.22: Desempenho da comunicação com PVM-TCP e PVM-UDP local nas máquinas IBM.

5.9 Sugestões aos Programadores

As medições do sistema de comunicação disponibilizado pelo PVM, seja entre máquinas ou numa máquina local, tem a finalidade de mostrar o desempenho do PVM em ambientes de baixo custo e de compará-lo com seu desempenho em arquiteturas de maior poder computacional, como as máquinas IBM RS6000 usadas nos experimentos. Essa comparação tem o objetivo de mostrar a possibilidade de implementar processamento paralelo distribuído utilizando o PVM em ambientes compostos por arquiteturas tipo PCs com o sistema operacional Linux. O processamento paralelo distribuído tem o potencial de permitir uma melhor utilização dos recursos dos ambientes, e consequentemente possibilitar a execução de problemas que requerem maior poder computacional e que demandam muito tempo de processamento.

Com base nas medições e discussões efetuadas nos sistemas de memória, na comunicação via soquetes com TCP e UDP, e na comunicação com PVM dos ambientes de

baixo custo estudados, são dadas a seguir algumas sugestões aos programadores de como implementar aplicações PVM que utilizem da melhor forma possível os recursos computacionais existentes em ambientes compostos por computadores tipo PC conectados a redes locais Ethernet.

Os programadores que pretendem desenvolver aplicações PVM para serem executadas em ambientes semelhantes aos estudados, devem se preocupar com o tipo de problema a ser paralelizado e se basear nas propriedades de programação paralela descritas na Seção 3.4. Técnicas de ajustes de balanceamento de carga, escalabilidade, granularidade e sincronização possibilitam ao programador implementar aplicações paralelas distribuídas sem necessariamente ser um especialista em máquinas paralelas.

Uma distribuição equilibrada da carga de trabalho de uma aplicação PVM entre as máquinas que compõem a máquina virtual paralela é importante para que as tarefas cooperem na solução do problema. Numa máquina virtual composta por arquiteturas iguais, ou seja, mesmo tipo de processador e sistemas de memória semelhantes, a distribuição das tarefas entre os processadores que formam a máquina virtual pode ser efetuada pelo próprio PVM. Contudo, quando as arquiteturas que formam a máquina virtual são diferentes, o programador tem a possibilidade de fazer uma melhor distribuição das cargas de trabalho através da implementação e alocação das tarefas de acordo com a capacidade de processamento de cada processador. Isto permite à máquina virtual ter um melhor aproveitamento dos recursos computacionais, e às aplicações PVM serem executadas mais rapidamente.

Caso haja muita comunicação entre as tarefas que compõem a aplicação PVM, a maneira como acontece a sincronização entre as tarefas é fundamental para que processadores gastem a maior parte do tempo de processamento com computação. Quanto menos tempo os processadores gastarem com comunicação, melhor é a sincronização, a cooperação entre as tarefas, e a execução da aplicação PVM. Assim, além de saber a capacidade de processamento do processador onde a tarefa é alocada, a escolha da forma de empacotamento e roteamento usado para a comunicação entre as tarefas possibilita que se gaste menos tempo com comunicação e mais tempo no processamento dos dados.

De acordo com as medições efetuadas na comunicação entre máquinas nas arquiteturas

tipo PCs, a forma de empacotamento usada não tem uma influencia significativa no desempenho da troca de mensagens. No entanto, o tipo de roteamento usado, comunicação direta entre as tarefas via protocolo TCP ou comunicação indireta via *daemons* do PVM utilizando o protocolo UDP, influencia no desempenho. Portanto, o programador deve se preocupar com o tipo de roteamento usado principalmente se há troca constante de mensagens entre as tarefas.

Já nas medições efetuadas na comunicação local nas arquiteturas tipo PCs, a forma de empacotamento *IN* influencia positivamente no desempenho em relação as demais formas. Assim, os programadores devem usá-las na troca de mensagens entre tarefas que são executadas em um mesmo processador. Quanto ao tipo de roteamento a ser usado, as trocas de mensagens entre as tarefas usando o roteamento direto apresentam um melhor desempenho.

Além da forma de empacotamento e roteamento, o tamanho das mensagens trocadas entre as tarefas deve ser levado em consideração. A comunicação com PVM sobre Linux apresenta oscilacões de desempenho conforme o tamanho da mensagem. Essas oscilações ocorrem na troca de mensagens entre as tarefas que usam o forma de empacotamento *IN* e roteamento direto das mensagens através do protocolo TCP.

Nos anexos estão as tabelas das medições efetuadas da comunicação via soquetes e da comunicação com PVM nas arquiteturas tipo PCs e IBM RS6000. As tabelas da comunicação via soquetes mostram uma comparação das medidas obtidas dos protocolos UDP e TCP nas arquiteturas tipos PCs. As tabelas da comunicação com PVM mostram uma comparação das medidas obtidas nas arquiteturas tipo PCs para cada forma de empacotamento e tipo de roteamento das mensagens. Também mostram uma comparação das medidas obtidas para cada forma de forma de empacotamento nas máquinas IBM RS6000.

5.10 Melhoria no Desempenho do PVM

Como um trabalho futuro visando uma melhora no desempenho do sistema de comunicação do PVM, um estudo mais aprofundado envolvendo os mecanismos de controle, política de envio, recebimento e fragmentação das mensagens efetuadas pelos processos

daemons do PVM é necessário. Este estudo terá por objetivo entender as estruturas de dados e algoritmos usados pelos *daemons* para se comunicarem com outros *daemons* e também com as tarefas de uma aplicação PVM. A partir desse estudo, algoritmos mais eficientes para gerenciar o envio, recebimento e fragmentação das mensagens pelos processos *daemons* poderão ser propostos. Mecanismos de controle mais eficientes que garantam a confiabilidades das mensagens também poderão ser propostos. Possibilitando assim, uma melhora no desempenho da comunicação do PVM do que a atualmente medida. Além disso, algumas limitações impostas pelo PVM poderão ser compreendidas. Por exemplo, a limitação imposta de 4 Kbytes como tamanho máximo para fragmentação das mensagens enviadas. Essa limitação influencia no desempenho do sistema de comunicação do PVM. Dada a redução nos custos, seria interessante medir o sistema de comunicação do PVM numa rede Ethernet de 100 Mbps e avaliar o impacto no seu desempenho. Atualmente, o Departamento de Informática não dispõe deste hardware.

CAPÍTULO 6

CONCLUSÃO

Este trabalho consiste na avaliação do desempenho do sistema de comunicação disponibilizado pelo PVM sobre o sistema operacional Linux em ambientes de computação de baixo custo. O PVM é um sistema baseado em passagem de mensagens que possibilita que um conjunto de computadores com arquiteturas heterogêneas e/ou homogêneas conectados a uma rede seja usado como uma máquina virtual paralela. Os ambientes de computação de baixo custo são normalmente compostos por redes locais, onde são conectados computadores do tipo PC e estações de trabalho.

O objetivo deste trabalho é demonstrar a viabilidade de se implementar processamento paralelo distribuído em ambientes de baixo custo através do PVM sobre o sistema operacional Linux. O processamento paralelo distribuído proporcionaria uma melhor utilização dos recursos computacionais destes ambientes, e por conseqüência um aumento do poder computacional. Assim, tarefas que exijam maior poder de computação poderiam ser executadas nos horários em que não houvessem usuários utilizando os computadores.

A avaliação do sistema de comunicação do PVM consistiu em medir e discutir as vazões obtidas no sistema de memória, na comunicação via soquetes utilizando os protocolos de transporte TCP e UDP, e na comunicação via PVM nos três ambientes de baixo custo estudados. Também é feita uma comparação entre as medições efetuadas na comunicação via soquetes e PVM com as medições aferidas em um ambiente composto por máquinas RS6000 de maior poder computacional com sistema operacional AIX, e que são máquinas muito mais caras.

As medições efetuadas nos sistemas de memória têm a finalidade de avaliar a capacidade de processamento das arquiteturas dos computadores. Numa máquina virtual paralela, é necessário conhecer a capacidade de processamento de cada arquitetura para determinar

o tempo de processamento gasto na comunicação entre os processadores.

As medições efetuadas na comunicação via soquetes têm a finalidade de avaliar o desempenho dos protocolos TCP e UDP. Estes protocolos são usados na comunicação via PVM. Logo, o desempenho desses protocolos influenciam nas vazões obtidas na comunicação via PVM.

As medições efetuadas na comunicação via PVM têm a finalidade de avaliar o desempenho da vazão do sistema de comunicação do PVM. As medições levam em consideração as formas de empacotamento e de roteamento no qual os dados a serem transmitidos podem ser submetidos. Essas formas influenciam no desempenho da comunicação via PVM, e são usadas conforme o tipo das arquiteturas envolvidas na comunicação.

Nossas medições mostram que a comunicação via PVM entre as tarefas executando em processadores distintos é capaz de alcançar uma vazão próxima da vazão máxima possível numa rede local Ethernet a 10 Mbps quando ninguém mais usa a rede. Para que isso ocorra, as mensagens trocadas entre as tarefas precisam ser grandes, isto é, acima de 16 Kbytes, e o tipo de roteamento usado deve ser o rotemento direto. Já a forma de empacotamento usada não influencia no desempenho da vazão alcançada. Isto é bom porque permite que computadores com arquiteturas heterogêneas, como estações de trabalho, façam parte de uma mesma máquina virtual com computadores do tipo PC.

Nossas medições confirmam que a comunicação local entre as tarefas de uma aplicação PVM alcançam vazões acima da vazão máxima da Ethernet, para mensagens maiores ou iguais a 2 Kbytes. A comunicação entre as tarefas quando possível, deve acontecer através de roteamento direto porque este alcança uma vazão muito superior ao roteamento via Pvmd. Além disso, a forma de empacotamento usada deve ser a "In Place", pois diminui o número de vezes que os dados são copiados entre armazenadores, e conseqüentemente o tempo gasto no empacotamento das mensagens.

De uma maneira geral, as medições mostram que o uso do PVM para viabilizar processamento paralelo distribuído nos ambientes estudados é viável e eficiente. Contudo, a criação de máquinas virtuais paralelas, nestes ambientes, não significa uma diminuição do tempo total de processamento para todos os tipos de problemas paralelizados. Em muitos casos, dependendo da implementação, o tempo de processamento aumenta con-

sideravelmente. O PVM é eficiente na implementação de programas paralelos no qual as tarefas trocam mensagens grandes; trocam poucas mensagens com mais de uma tarefa simultaneamente; e trocam mensagens pequenas raramente.

Como trabalho futuro, existe a possibilidade de melhoria no desempenho da comunicação através dos *daemons* do PVM pela otimização dos mecanismos de controle e da política de envio, recebimento e fragmentação das mensagens.

BIBLIOGRAFIA

[BMK88] David R. Boggs, Jeffrey C. Mogul, and Chritopher A. Kent. Measured capacity of an ethernet: Myths and reality. Technical Report 88/4, Digital Western Research Laboratory, 1988.

[BRI97] BRISA. *Arquiteturas de Redes de Computadores OSI e TCP/IP.* Makron Books, 2nd edition, 1997.

[BWT96] Joachim M. Blum, Thomas M. Warschko, and W. F. Tichy. Pspvm: Implementing pvm on a high-speed interconnect for workstations clusters. Technical report, University of Karlsruhe, 1996.

[Cen91] IBM International Technical Support Centers. TCP/IP - tutorial and technical overview. IBM - International Technical Support Centers, sep 1991.

[Cen96] Maui High Performance Computing Center. Introduction to parallel programing, 1996. http://www.mhpcc.edu/doc/ipp.html.

[CEN97] CENAPADNE. Introdução ao PVM on-line, 1997. http://www.cenapadne.br.

[CS94] Douglas E. Comer and David L Stevens. *Internetworking With TCP/IP - Design, Implementation, and Internals*, volume 2. Prentice Hall International Editions, 2nd edition, 1994.

[DDK90] Willibald Doeringer, Doug Dykeman, and Matthias Kaiserswerth. A survey of light-weight transport protocols for high-speed networks. *IEEE Trans. on Communications*, 38(11):2025–2039, nov 1990.

[Dru96] P. Druschel. Operating system support for high-speed computations. *Comm. of the ACM*, 39(9):41–51, sep 1996.

[DSG95] E. Dillon, C. G. D. Santos, and J. Guyard. Homogeneous and heterogeneous networks of workstations: Message passing overhead. Technical report, INRIA/CRIAN, jun 1995.

[DWB+93] Cris Dalton, Greg Watson, David Banks, Costas Calamvokis, Aled Edwards, and John Lumley. Afterburner. *IEEE Network*, pages 36–43, jul 1993.

[Fly72] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. on Computers*, C-21:948–960, sep 1972.

[GBD+94a] A. Geist, A. Beguelin, J. Dongarra, Jiang W., R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

[GBD+94b] A. Geist, A. Beguelin, J. Dongarra, Jiang W., R. Manchek, and V. Sunderam. Pvm3 user's guide and reference manual. Oak National Laboratory, sep 1994.

[Hex99] R. A. Hexsel. *Redes de Dados: Tecnologia e Programação.* a ser publicado por Livros Técnicos e Científicos Ed., 1999.

[KP92] Jonathan Kay and Joseph Pasquale. A performance analisys of TCP/IP and UDP/IP networking software for the decstation 5000. Technical Report 92093-0114, University of California, San Diego, 1992.

[Nev96] Nick Nevin. The performance of LAM 6.0 and MPICH 1.0.12 on a workstation cluster. Technical Report OSC-TR-1996-4, Ohio Supercomputer Center, mar 1996.

[NN95] Natawat Nupairoj and L. M. Ni. Benchmarking of multicast communication services. Technical Report MSU-CPS-ACS-103, Michigan State University, apr 1995.

[PH96] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, 2nd edition, 1996.

[PP93] Christos Papadopoulos and Gurudatta M. Parulkar. Experimental evaluation of SUNOS IPC and TCP/IP protocol implementation. *IEEE/ACM Trans. on Networking*, 1(2):199–216, apr 1993.

[Qui87] M. J. Quinn. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill Book Company, 1987.

[RA96] Steven H. Rodrigues and Thomas E. Anderson. High-performance local area communication with fast sockets. Technical Report 94720, University of California at Berkeley, 1996.

[SOHL96] Marc Snir, Steve Otto, and Steven Huss-Lederman. *MPI: The Complete Reference.* The MIT Press, 1996.

[SSS+97] P. S. Souza, M. J. Santana, R. H. C. Santana, L. J. Senger, and R. Picinato. O impacto do protocolo TCP/IP na computação paralela distribuída no ambiente Windows 95. In *XV Simpósio Brasileiro de Redes de Computadores*, pages 36–47, 1997.

[Ste90] W. R. Stevens. *Unix Networking Programming.* Prentice Hall Software. Prentice Hall International Inc., 1990.

[Ste94] Peter A. Steenkiste. A systematic approach to host interface design for high-speed networks. *IEEE Network*, pages 47–56, mar 1994.

[Sto93] Harold S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 3rd edition, 1993.

APÊNDICE A

Comunicação Soquetes com UDP

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	6467.0	0.0405	6485.7	0.0404	5093.2	0.0515	5101.9	0.0514
32	3413.4	0.0768	3435.2	0.0763	2738.9	0.0957	2929.7	0.0895
64	1911.6	0.1371	1971.3	0.1330	1592.0	0.1647	1608.7	0.1630
128	1173.9	0.2233	1241.8	0.2111	1105.3	0.2372	1035.6	0.2531
256	810.9	0.3233	879.3	0.2980	732.2	0.3580	802.5	0.3267
512	628.9	0.4168	695.8	0.3767	590.8	0.4437	593.3	0.4418
1024	538.3	0.4870	605.0	0.4333	517.2	0.5069	523.1	0.5012
1200	528.4	0.4996	592.0	0.4460	514.6	0.5130	516.1	0.5116
1300	519.2	0.5058	590.4	0.4448	505.4	0.5196	508.3	0.5166
1400	515.4	0.5106	585.5	0.4495	501.9	0.5244	504.0	0.5222
1500	552.8	0.4775	616.3	0.4284	515.8	0.5118	517.2	0.5104
1600	521.6	0.5031	706.5	0.3714	489.2	0.5364	491.8	0.5336
2048	438.9	0.5973	500.3	0.5240	412.1	0.6362	412.7	0.6352
4096	342.1	0.7662	370.2	0.7081	319.2	0.8211	319.9	0.8193
8192	292.0	0.8976	310.3	0.8449	272.3	0.9628	272.4	0.9623
16384	272.2	0.9631	282.5	0.9280	256.1	1.0234	255.4	1.0266
32768	258.8	1.0128	264.0	0.9929	242.5	1.0808	242.6	1.0804
63000	240.7	1.0471	246.6	1.0218	227.6	1.1073	228.3	1.1038

Tabela A.1: Comunicação Soquetes com UDP.

APÊNDICE B

Comunicação Soquetes com TCP

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	7432.1	0.0353	7314.9	0.0358	67153.4	0.0039	67170.6	0.0039
32	4088.1	0.0641	4032.2	0.0650	3089.0	0.0849	3013.8	0.0870
64	2275.7	0.1152	2269.2	0.1155	1775.3	0.1477	1767.8	0.1483
128	1363.8	0.1922	1437.7	0.1823	1100.8	0.2381	1091.9	0.2401
256	911.8	0.2875	978.3	0.2680	779.2	0.3364	775.1	0.3382
512	693.2	0.3781	737.6	0.3554	618.8	0.4237	619.7	0.4230
1024	573.0	0.4575	632.8	0.4143	537.8	0.4874	535.1	0.4899
1200	559.8	0.4716	621.4	0.4248	527.6	0.5004	527.4	0.5005
1300	548.4	0.4788	608.8	0.4313	518.7	0.5063	518.8	0.5061
1400	543.2	0.4846	606.5	0.4340	518.4	0.5077	515.1	0.5109
1500	43826.8	0.0060	43894.7	0.0060	34649.6	0.0076	35571.4	0.0074
1600	40836.1	0.0064	40848.3	0.0064	32308.2	0.0081	32245.2	0.0081
2048	31897.5	0.0082	31883.7	0.0082	25094.1	0.0104	25071.1	0.0105
4096	957.3	0.2738	955.3	0.2744	368.5	0.4105	637.8	0.4110
8192	483.6	0.5421	480.0	0.5462	661.1	0.3965	640.6	0.4092
16384	474.0	0.5531	475.1	0.5518	554.2	0.4730	401.7	0.6526
32768	286.1	0.9162	286.7	0.9144	287.6	0.9115	282.0	0.9296
65536	272.9	0.9605	264.7	0.9904	281.7	0.9305	278.0	0.9430
131072	248.8	1.0535	256.7	1.0212	265.8	0.9861	254.9	1.0285
262144	239.9	1.0927	240.3	1.0907	258.3	1.0149	243.5	1.0766

Tabela B.1: Comunicação Soquetes com TCP.

APÊNDICE C

Comunicação PVM com UDP/XDR

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	36064.1	0.0073	129266.7	0.0081	67215.9	0.0156	65477.3	0.0160
32	18202.4	0.0144	65525.6	0.0160	33887.0	0.0309	33580.8	0.0312
64	9311.8	0.0282	33753.5	0.0311	17842.6	0.0588	17731.0	0.0591
128	4947.1	0.0530	18129.6	0.0578	9871.1	0.1062	9821.9	0.1068
256	2756.0	0.0951	10243.8	0.1024	5877.5	0.1784	5846.9	0.1793
512	1617.0	0.1621	6251.8	0.1677	3864.0	0.2714	3847.8	0.2725
1024	1075.2	0.2438	4292.3	0.2443	2884.6	0.3635	2867.3	0.3657
2048	732.5	0.3579	3051.7	0.3436	2079.3	0.5043	2067.7	0.5071
4096	633.7	0.4136	2544.4	0.4121	1770.9	0.5921	1748.7	0.5996
8192	542.1	0.4835	2215.5	0.4733	1635.2	0.6412	1620.0	0.6473
16384	487.0	0.5383	2050.0	0.5115	1584.7	0.6617	1557.7	0.6732
32768	460.4	0.5693	1924.4	0.5449	1529.6	0.6855	1562.8	0.6709
65536	449.5	0.5833	1901.6	0.5514	1531.2	0.6848	1511.6	0.6937
131072	729.2	0.3595	2956.9	0.3546	2442.7	0.4293	2690.3	0.3898

Tabela C.1: Comunicação PVM com UDP/XDR.

APÊNDICE D

Comunicação PVM com UDP/RAW

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	35787.3	0.0073	128682.6	0.0081	65887.1	0.0159	64964.5	0.0161
32	18194.7	0.0144	65556.4	0.0160	33697.9	0.0311	33873.2	0.0310
64	9401.5	0.0279	13831.5	0.0758	17721.8	0.0592	17625.1	0.0595
128	4930.2	0.0532	18058.3	0.0581	9856.9	0.1064	9832.7	0.1066
256	2756.5	0.0951	10183.8	0.1030	5897.6	0.1778	5880.3	0.1783
512	1619.9	0.1618	6250.1	0.1678	3848.8	0.2724	3867.9	0.2711
1024	1069.5	0.2451	4274.3	0.2453	2879.1	0.3642	2841.3	0.3690
2048	731.1	0.3586	3040.2	0.3449	2060.7	0.5088	2036.7	0.5081
4096	631.4	0.4152	2552.7	0.4108	1767.8	0.5931	1743.9	0.6013
8192	543.1	0.4827	2212.4	0.4739	1637.0	0.6406	1618.0	0.6481
16384	491.2	0.5337	2054.6	0.5104	1587.3	0.6606	1557.2	0.6734
32768	463.9	0.5651	1933.4	0.5423	1534.1	0.6835	1522.1	0.6889
65536	454.8	0.5764	1903.1	0.5510	1523.8	0.6881	1511.0	0.6939
131072	729.3	0.3594	2991.8	0.3505	2432.5	0.4311	2256.1	0.4648

Tabela D.1: Comunicação PVM com UDP/RAW.

APÊNDICE E

Comunicação PVM com UDP/IN

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	39354.9	0.0067	142798.5	0.0073	70463.8	0.0149	71583.8	0.0146
32	20399.5	0.0129	73114.6	0.0143	35996.9	0.0291	36180.9	0.0290
64	10255.9	0.0256	37236.4	0.0282	19449.5	0.0539	19033.8	0.0551
128	5436.0	0.0482	19919.0	0.0526	10504.4	0.0998	10402.8	0.1008
256	2927.5	0.0895	11179.6	0.0938	6166.8	0.1700	6153.0	0.1704
512	1736.5	0.1510	6641.9	0.1579	4128.3	0.2540	3991.0	0.2627
1024	1115.8	0.2349	4467.8	0.2347	2956.7	0.3546	2932.7	0.3576
2048	751.3	0.3489	3127.6	0.3353	2092.3	0.5012	2105.7	0.4980
4096	626.3	0.4186	2524.2	0.4154	1756.2	0.5971	1741.5	0.6021
8192	530.5	0.4941	2190.0	0.4788	1611.8	0.6506	1603.3	0.6540
16384	482.6	0.5432	2015.4	0.5203	1549.9	0.6766	1533.1	0.6840
32768	466.4	0.5621	1950.9	0.5375	1539.7	0.6810	1508.9	0.6949
65536	456.7	0.5740	1891.0	0.5545	1520.2	0.6898	1507.0	0.6958
131072	816.9	0.3209	3112.1	0.3369	2899.4	0.3671	2906.1	0.3608

Tabela E.1: Comunicação PVM com UDP/IN.

APÊNDICE F

Comunicação PVM com TCP/XDR

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	14701.5	0.0178	63301.4	0.0166	35396.4	0.0296	34564.6	0.0303
32	7715.7	0.0340	29590.4	0.0354	18449.2	0.0568	18472.9	0.0568
64	4076.3	0.0643	15844.8	0.0662	10046.8	0.1044	10014.8	0.1047
128	2289.2	0.1145	9617.5	0.1090	5950.3	0.1762	5899.6	0.1777
256	1366.7	0.1918	5973.3	0.1755	3911.8	0.2681	3842.9	0.2729
512	930.0	0.2819	4190.9	0.2502	2871.0	0.3652	2827.6	0.3708
1024	705.5	0.3716	3199.5	0.3277	2354.7	0.4453	2333.7	0.4493
2048	515.7	0.5083	2289.3	0.4580	1796.9	0.5836	1754.7	0.5976
4096	397.9	0.6588	1713.3	0.6120	1437.1	0.7296	1391.5	0.7535
8192	331.2	0.7914	1387.7	0.7556	1304.8	0.8036	1183.7	0.8858
16384	292.2	0.8973	1229.4	0.8529	1433.7	0.7314	1326.3	0.7906
32768	281.5	0.9313	1149.0	0.9126	1208.9	0.8674	1145.9	0.9151
65536	271.2	0.9666	1126.6	0.9307	1120.0	0.9362	1103.3	0.9504
131072	275.4	0.9520	1236.5	0.8480	1096.9	0.9560	1043.3	1.0051

Tabela F.1: Comunicação PVM com TCP/XDR.

APÊNDICE G

Comunicação PVM com TCP/RAW

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	14758.8	0.0178	58343.3	0.0180	35112.5	0.0299	34725.6	0.0302
32	7607.1	0.0345	30421.8	0.0345	18263.1	0.0574	18156.4	0.0578
64	4050.5	0.0647	15995.4	0.0656	10106.6	0.1038	9986.0	0.1050
128	2234.5	0.1173	8906.8	0.1177	5936.2	0.1766	5875.0	0.1785
256	1343.3	0.1951	5561.2	0.1886	3875.0	0.2706	3826.9	0.2740
512	912.5	0.2873	3854.4	0.2720	2844.7	0.3686	2828.9	0.3707
1024	706.3	0.3711	3005.5	0.3489	2334.0	0.4493	2325.5	0.4509
2048	513.5	0.5105	2289.7	0.4580	1789.1	0.5861	1751.7	0.5986
4096	397.9	0.6588	1707.8	0.6140	1442.4	0.7269	1394.9	0.7517
8192	329.2	0.7962	1385.1	0.7570	1236.1	0.8483	1182.7	0.8866
16384	292.8	0.8954	1222.6	0.8577	1442.1	0.7271	1306.6	0.8025
32768	279.1	0.9394	1151.3	0.9108	1190.9	0.8805	1149.5	0.9122
65536	273.2	0.9595	1131.3	0.9269	1214.0	0.8638	1115.2	0.9403
131072	275.1	0.9528	1230.1	0.8524	1094.8	0.9578	1070.5	0.9795

Tabela G.1: Comunicação PVM com TCP/RAW.

APÊNDICE H

Comunicação PVM com TCP/IN

Número	P75		P90		P166		P200	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	177440.3	0.0150	68075.0	0.0154	47751.0	0.0220	42931.6	0.0244
32	8859.4	0.0296	34406.1	0.0305	21700.2	0.0483	25133.4	0.0417
64	4531.0	0.0579	17844.8	0.0588	11312.2	0.0927	11295.2	0.0928
128	2501.8	0.1048	9778.8	0.1072	8323.1	0.1260	7598.5	0.1380
256	1456.0	0.1800	5826.1	0.1800	4134.7	0.2536	4068.5	0.2577
512	957.1	0.2739	4012.1	0.2614	2972.7	0.3527	2940.5	0.3566
1024	711.7	0.3683	3071.1	0.3414	2390.2	0.4387	2370.3	0.4424
2048	518.4	0.5057	2295.2	0.4568	1770.4	0.5923	1757.8	0.5965
4096	586.5	0.4469	1633.0	0.6421	1380.0	0.7599	1392.4	0.7531
8192	319.0	0.8216	1325.5	0.7911	1397.4	0.7504	1657.6	0.6326
16384	395.7	0.6625	1155.7	0.9073	1335.5	0.7852	1283.1	0.8172
32768	291.8	0.8985	1120.1	0.9362	1165.8	0.8995	1125.0	0.9320
65536	272.6	0.9618	1158.8	0.9049	1084.3	0.9671	1009.8	1.0384
131072	259.7	1.0096	1093.7	0.9588	1071.5	0.9786	967.1	1.0842

Tabela H.1: Comunicação PVM com TCP/IN.

APÊNDICE I

Comunicação PVM com UDP nas máquinas IBM

Número	XDR		RAW		IN	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	125178.5	0.0084	130917.7	0.0080	120139.1	0.0087
32	65957.6	0.0159	71013.6	0.0148	68435.3	0.0166
64	32167.5	0.0326	31292.9	0.0335	34775.1	0.0302
128	16191.6	0.0648	15419.3	0.0680	16253.9	0.0645
256	8684.4	0.1207	8058.5	0.1301	8390.5	0.1250
512	4713.8	0.2224	4377.2	0.2396	4500.5	0.2330
1024	2869.5	0.3654	2671.5	0.3925	2732.9	0.3837
1500	2298.3	0.4595	2260.6	0.4671	2320.8	0.4550
2048	1886.4	0.5559	1882.0	0.5572	1929.5	0.5435
4096	1814.1	0.5780	1822.7	0.5753	1820.8	0.5759
8192	1518.2	0.6907	1529.3	0.6857	1560.5	0.6719
16384	1416.1	0.7405	1400.3	0.7488	1473.9	0.7114
32768	1369.3	0.7658	1352.4	0.7754	1396.4	0.7509
65536	1313.6	0.7982	1319.5	0.7947	1390.2	0.7542
131072	1319.2	0.7948	1318.5	0.7953	1382.1	0.7587

Tabela I.1: Comunicação PVM com UDP nas máquinas IBM.

APÊNDICE J

Comunicação PVM com TCP nas máquinas IBM

Número	XDR		RAW		IN	
de bytes	[μS]	[by/μS]	[μS]	[by/μS]	[μS]	[by/μS]
16	64328.3	0.0163	63607.4	0.0165	82833.3	0.0127
32	32102.5	0.0327	29612.8	0.0354	39145.5	0.0268
64	20993.4	0.0499	19344.7	0.0542	19362.0	0.0542
128	7380.8	0.1421	7309.3	0.1435	8399.7	0.1248
256	4252.9	0.2466	4243.8	0.2471	5201.8	0.2016
512	2578.0	0.4067	3365.7	0.3115	2960.7	0.3542
1024	1770.1	0.5924	1743.0	0.6016	1806.7	0.5804
1500	1527.1	0.6915	1574.2	0.6708	1627.1	0.6490
2048	1338.8	0.7832	1319.0	0.7950	1401.9	0.7480
4096	1217.0	0.8616	1208.5	0.8677	1187.6	0.8829
8192	1104.0	0.9498	1129.8	0.9281	1103.7	0.9501
16384	1086.3	0.9652	1059.6	0.9896	1042.0	1.0063
32768	1053.6	0.9952	1057.4	0.9916	1024.2	1.0238
65536	1063.6	0.9859	1049.0	0.9996	1023.5	1.0245
131072	1039.9	1.0083	1043.7	1.0047	1014.1	1.0340

Tabela J.1: Comunicação PVM com TCP nas máquinas IBM.