ANÁLISE DE SISTEMAS DE COMUNICAÇÃO PARA COMPUTAÇÃO PARALELA EM CLUSTERS

BRUNO OTTO THEODORO ROSA

Dissertação apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Mestre em Ciências "Física Aplicada-Opção Física Computacional"

Orientador: Prof. Dr. Jan Frans Willem Slaets

São Carlos – São Paulo
2002

Rosa, Bruno Otto Theodoro

"Análise de sistemas de comunicação para computação paralela em clusters"/
Bruno Otto Theodoro Rosa – São Carlos, 2002

Dissertação (Mestrado) – Instituto de Física de São Carlos da Universidade de São Paulo, 2002 – páginas: 52
Área: Física Aplicada-opção Física Computacional
Prof. Dr. Jan Frans Willem Slaets

1. Diminuindo latência, 2. Arquiteturas de nível de usuário, 3. Processamento em clusters

I. Título

IFSC UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Caixa Postal 369
13560-970 São Carlos, SP
Av. Trabalhador São-carlense, 400.
13566-590 São Carlos, SP

Fone/Fax 16 273 9777
www.if.sc.usp.br
wladerez@if.sc.usp.br

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE BRUNO OTTO THEODORO ROSA, APRESENTADA AO PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA, ÁREA DE CONCENTRAÇÃO FÍSICA APLICADA-OPÇÃO FÍSICA COMPUTACIONAL, DA UNIVERSIDADE DE SÃO PAULO, EM 26/02/2002.

COMISSÃO JULGADORA:

Prof. Dr. JAN FRANS WILLEM SLAETS (Orientador) – IFSC/USP

Prof. Dr. GUILHERME MATOS SIPAHI – IFSC/USP

Dr. ATTILIO CUCCHIERI – IFSC/USP

Sumário

Índice de figuras

Resumo

Apesar do aumento constante da largura de banda das tecnologias de rede de computadores as aplicações de processamento paralelo ainda necessitam de uma latência de comunicação mais baixa que a oferecida. Este aspecto não tem sido contemplado por estas tecnologias de rede pois está relacionado à maneira como o sistema operacional utiliza-se dos recursos do hardware com relação aos dados enviados pelas aplicações dos usuários. Neste trabalho apresentamos um estudo da técnica para diminuição desta latência e as características necessárias para implementação deste tipo de sistemas, incluindo mecanismos de transferência de dados, técnicas para tradução de endereços, proteção, transferência de controle, grau de confiabilidade e implementação de "Multicasting". Apresentamos também o estudo de um sistema já implementado, chamado M-VIA, comparando seu desempenho com o TCP/IP tradicional.

Introdução

O avanço tecnologico da microeletrônica impulsionou a construção de microcomputadores suficientemente velozes para atender as necessidades dos problemas científicos de simulação e estatística com custos acessíveis. Esta possibilidade fez com que cada vez mais cientistas começassem a utilizar recursos computacionais para resolução de seus problemas. Naturalmente modelos mais complexos foram propostos e os limites de processamento precisavam ser expandidos. Neste contexto surge a possibilidade de se multiplicar o poder de processamento através da aquisição de vários microcomputadores que sejam então ligados entre si e que funcionem em cooperação formando *clusters* para solucionar problemas computacionais.

Existe assim a possibilidade de construirmos em vários

casos sistemas com desempenho equivalente a supercomputadores da década passada com componentes fabricados em larga escala, comuns no mercado.

Entretanto, para se utilizar esta nova estrutura, além de dominar novos paradigmas de programação, o usuário deve analisar as variedades de sistemas paralelos baseados em *clusters* e tomar a sua decisão.

Neste trabalho abordamos a avaliação dos sistemas de comunicação para compor estes *clusters*, detectando as limitações do sistema de comunicação tradicional e analisando a possibilidade de substituição por outro sistema que seja mais eficiente para a transmissão de mensagens entre os diversos nós que compõe um *cluster.*

No capítulo 1 apresentamos os modelos de sistemas paralelos em função de seus sistemas de comunicação. Em seguida, no capítulo 2, abordamos o funcionamento destes sistemas de comunicação e o problema da latência. Analisamos então as possíveis soluções através de Sistemas de Comunicação de Nível de Usuário e dois destes sistemas são apresentados no capítulo 3. Realizamos no capítulo 4 testes com o sistema de comunicação tradicional nativo do sistema operacional (TCP/IP) em relação ao Sistema de Comunicação de Nível de Usuário e apresentamos os resultados experimentais. O capítulo 5 apresenta conclusões e observações e em seguida colocamos alguns anexos.

Capítulo 1

Clusters

1.1 A evolução tecnológica

A velocidade de um microprocessador depende de seu número de transistores [BASKETT 93]. Tanto os circuitos de memória quanto os circuitos de processamento tem dobrado sua capacidade de armazenamento a cada dois anos, aproximadamente.

O tamanho das vias de comunicação também aumentou. Em 1975, a família de processadores da Intel 8080 tinha largura de dados de 8 bits. Praticamente todos os microprocessadores tem largura de dados de 64 bits desde 1995, ou seja, o dobro a cada 5 anos. A velocidade das vias de comunicação (*backplanes*) também tem dobrado a cada 10 anos. As taxas de *clocks* dos microprocessadors também estão atingindo aquelas dos mais velozes supercomputadores disponíveis

Observa-se assim que os avanços se deram em muitas áreas na engenharia envolvida no desenvolvimento dos microprocessadores, não se restringindo apenas a uma de suas características, como a velocidade de *clock*, por exemplo. Tivemos a evolução para um conjunto melhor, que evoluiu até o patamar necessário para atender à demanda inicial de processamento científico. Hoje os microprocessadores incorporam tecnologias antes presentes somente nos *mainframes* e *supercomputadores*, como arquitetura superescalar, *pipelines*, instruções vetoriais e execução especulativa. [STALLINGS 00] [CULLER 99-2]

Também a dinâmica do mercado tem favorecido a comercialização de

sistemas de computação baseado em microprocessadores. Houve uma diminuição das aquisições de supercomputadores por parte dos órgãos governamentais, principalmente devido à redução das verbas para este fim e também uma conscientização quanto ao fato de que sistemas baseados em microprocessadores tem uma relação custo-benefício muito melhor daquele correspondente aos supercomputadores.

1.2 Sistemas baseados em *Clusters*

Tradicionalmente os sistemas baseados em *clusters* têm recebido vários nomes. Entre eles *farms*, sistemas de processamento concorrente baseado em estações de trabalho, *workstation array* e *hypercomputing*. Basicamente um sistema de *clusters* é um agrupamento de vários computadores, denominados nós, onde cada nó possui todos os componentes necessários para que funcione como uma máquina independente (CPU, memória, sistema de armazenamento). Estes recursos são otimizados de acordo com o propósito de funcionamento do *cluster*.

Desta forma, a construção de clusters é relativamente simples, uma vez que será composto de: *(1)* nós de processamento; *(2)* um sistema de comunicação destes nós; e *(3)* um sistema operacional que gerencie os componentes do cluster preferencialmente de forma transparente e otimizada.

Normalmente o sistema de comunicação são redes de dados comerciais, largamente disponíveis, como a *Ethernet*. O protocolo de comunicação é o TCP/IP, com documentação e aplicações largamente utilizadas. Neste trabalho analisaremos outros sistemas de comunicação para *clusters*.

Para comunicação entre diversos nós dois modelos de sistemas de comunicação são normalmente utilizados. Esses modelos diferenciam-se quanto a forma com que informações serão trocadas entre os nós, podendo ser de forma mais transparente ou explícita. Estes sistemas são o de passagem de mensagens

(*message passing)* e o sistema de memória distribuída (*distributed shared memory*). Os sistemas de passagem de mensagem transmitem informações explícitas sobre os dados que devem ser compartilhados, incluindo detalhes como nó de destino e processo comunicante, além do porto de comunicação. Os sistemas de memória distribuída em geral escondem esta complexidade do programador, e utilizam-se de um sistema de passagem de mensagens como o anterior para implementarem as suas funcionalidades. No caso do sistema baseado em mensagens haverá a necessidade de se reprogramar as aplicações, identificando os pontos de comunicação e tornando explícita a troca de dados, para que elas utilizem-se de sistemas de comunicação de linguagens como MPI [SNIR 96].

As vantagens do uso de *clusters* aparecem em diferentes características destes sistemas. Descrevemos a seguir aplicações para os *clusters.*

Existe um aumento crescente no meio acadêmico do uso de métodos numéricos, que exigem capacidades moderadas de processamento. Parte desta demanda pode ser suprida pelos *clusters* sem a necessidade da aquisição de computadores de grande porte. Pode-se adquirir diversos microcomputadores de última geração e interligá-los por uma rede de dados exclusiva e utilizar-se uma linguagem de programação paralela.

Outra aplicação teoricamente possível e também referenciada na literatura [KAPLAN 94] é a utilização dos ciclos ociosos de cada uma das estações de trabalho espalhados em um campus para processamento paralelo.

A característica de expansibilidade dá flexibilidade aos *clusters* pois pode-se a qualquer momento adquirir um novo nó, como um novo microcomputador, e facilmente acrescentá-lo ao cluster aumentando a capacidade de processamento.

Para o controle do funcionamento pode haver também um software de distribuição de trabalho que ofereça um equilíbrio de carga (*load balancing*) entre os

diversos nós. Tanto tarefas paralelas como processos individuais deverão ser contabilizados por este software para que a distribuição de cargas seja realista. Preferencialmente este sistema também deverá ser capaz de recuperar-se de falhas, redistribuindo processos para outros nós no caso de falha em algum deles. Este sistema é assim capaz de detectar um nó que falhou e redistribuir seu trabalho, impedindo que o processamento geral interrompa-se.

Destacam-se assim dois tipos de *clusters*. O primeiro é o cluster *não-dedicado*, que é formado por todos os equipamentos disponíveis dentro de uma corporação e que tem seus *ciclos perdidos* aproveitados. Criteriosamente estes ciclos perdidos podem ser aproveitados constantemente, ou dependerem de inatividade do nó correspondente. Dentro desta classe, podemos destacar o programa SETI (*Search for Extraterrestrial Intelligence*), [SETI 01] que distribui um *descanso de tela* (*screensaver*) que, quando entra em ação, comunica-se com o servidor do SETI, carrega um conjunto de dados e faz então cálculos correspondentes às necessidades dos pesquisadores desta agência, remetendo os resultados de volta ao servidor quando terminados. O SETI representa um *cluster* de tamanho global, onde cada microcomputador com *descanso de tela* é um nó e trabalha quando ninguém está utilizando a máquina.

O segundo critério é do uso fora do período de atividades do usuário da estação de trabalho. Durante o período da noite, por exemplo, nós que são normalmente utilizados como desktops durante o dia, são aproveitados para processamento. Este critério deve considerar que a tarefa atribuída a esse nó deverá acabar antes do início das atividades do próximo dia, para que não haja interrupção do processamento. Porém, a existência de unidades de processamento de boa capacidade a custo reduzido torna a validade deste critério questionável.

O segundo tipo é o *cluster* dedicado, e é deste tipo de cluster que trataremos aqui. Um conjunto de máquinas atende às necessidades de processamento sem no entanto serem utilizadas para outro fim. Normalmente este

tipo de *cluster* possui uma rede dedicada, que não concorre com o tráfego corporativo.

Neste trabalho verificamos que a montagem de um cluster para processamento paralelo utilizando-se uma linguagem de passagem de mensagens como MPI é viável. Além disso, verificamos que a adoção de sistemas de comunicação estudados neste trabalho, como o MVIA [MVIA FAQ], permite que trabalhemos com uma *granularidade* mais fina, uma vez que a comunicação torna-se mais eficiente.

1.3 Comparação entre sistemas SMP e *Clusters*

Em [PFISTER 95] temos uma comparação interessante entre sistemas SMP e *clusters*. Os sistemas SMP podem ser comparados ao cão mitológico *kerberos*, que possui um único corpo cheio de cabeças, enquanto que os *clusters* podem ser comparados a uma matilha de cães.

No nosso *kerberos*, cada cabeça representa um processador, que tem a capacidade de comer independentemente de todas as outras. O controle do *kerberos* é muito mais fácil, uma vez que basta uma coleira de bom tamanho para levá-lo para um passeio, ou levá-lo para se alimentar. Em contrapartida, cada cabeça tem que cooperar muito bem com todas as outras, com a finalidade de que consigam trabalhar juntas. A constituição de um *kerberos* também é toda especial. Assim como um computador SMP, nosso cão fictício necessita de um esôfago e de um estômago especiais, de forma que consiga transportar e digerir todo alimento engolido por cada uma das bocas mastigantes. No SMP, o barramento de comunicação entre processadores precisa ser especialmente construído, para que não constitua um gargalo para todo o sistema. Isto obviamente exige um projeto específico adequado à arquitetura da máquina SMP. No caso do cão se ferir, todas as outras cabeças ficam comprometidas. Em máquinas SMP é difícil implementar

uma degradação elegante. Acrescentar cabeças ao kerberos seria algo difícil até mesmo para o melhor cirurgião veterinário. Da mesma forma, máquinas SMP têm limite de número de processadores bem definido.

Na nossa matilha de cães, nosso *cluster*, cada cão representa um computador totalmente independente. Acrescentar ou retirar cães da matilha é uma tarefa simples, assim como colocar e retirar nós de um cluster é relativamente simples. Se um dos cães se fere, pode ser separado da matilha e levado ao veterinário. Nós defeituosos em *clusters* podem ser retirados e levados para manutenção sem maiores problemas. Todo cão da matilha é um cão comum, sem nenhuma característica bizarra de constituição. Nós de um cluster são microcomputadores comuns, que podem ser adquiridos no mercado. Em contrapartida, levar um matilha de cães para passear ou se alimentar pode ser uma tarefa muito difícil, assim como controlar a forma como todos os nós deverão comunicar-se e funcionar depende de um software de controle adequado.

Enquanto os computadores SMP já possuem ferramentas de *software* e um mercado específico, existe ainda a necessidade de desenvolvimento de sistemas que venham a controlar os clusters de forma transparente, escondendo todas as características técnicas e a complexidade do usuário.

1.4 A questão da comunicação

Os nós de um *cluster* precisam de um sistema de comunicação que seja adequado para sua finalidade. Este sistema de comunicação deverá fornecer uma banda de comunicação suficientemente larga, mas deverá principalmente oferecer uma **baixa latência** nas comunicações, que permita que a granularidade da paralelização dos problemas seja suficientemente fina para atender um conjunto de aplicações ainda maior. A latência influencia diretamente na comunicação pela rede, uma vez que ela está presente em cada um dos pacotes de informação transmitidos. Assim, qualquer diminuição da latência representa ganho significativo

para a comunicação entre os nós.

Em [MARTIN 99] temos uma análise deste comportamento através de simulações. Um estudo do impacto do desempenho das comunicações em aplicações paralelas colocadas em redes de alto desempenho demonstrou que a diminuição da latência em rede contribui para melhora no desempenho de aplicações. Como resultado teórico do simulador, algumas aplicações demonstram-se sensíveis à latência devido às sobretaxas de comunicação, diminuindo o desempenho na ordem de 60 vezes em sistemas de 32 processadores, quando a latência da rede é aumentada de 3 para 103 microssegundos.

No próximo capítulo abordamos o sistema de comunicação como um todo, analisando os aspectos de latência e soluções propostas para sua diminuição.

Capítulo 2

Sistemas de Comunicação

2.1- O Sistema de Comunicação e o Sistema Operacional

O sistema operacional escolhido para nosso trabalho foi o sistema operacional LINUX [LINUX 01], pela disponibilidade do código fonte dos controladores de dispositivo e pela documentação.

Os sistemas do tipo UNIX implementam o protocolo de rede TCP/IP utilizando um subconjunto de camadas do modelo conhecido como ISO/OSI (Fig 2.1) [COMER 95].

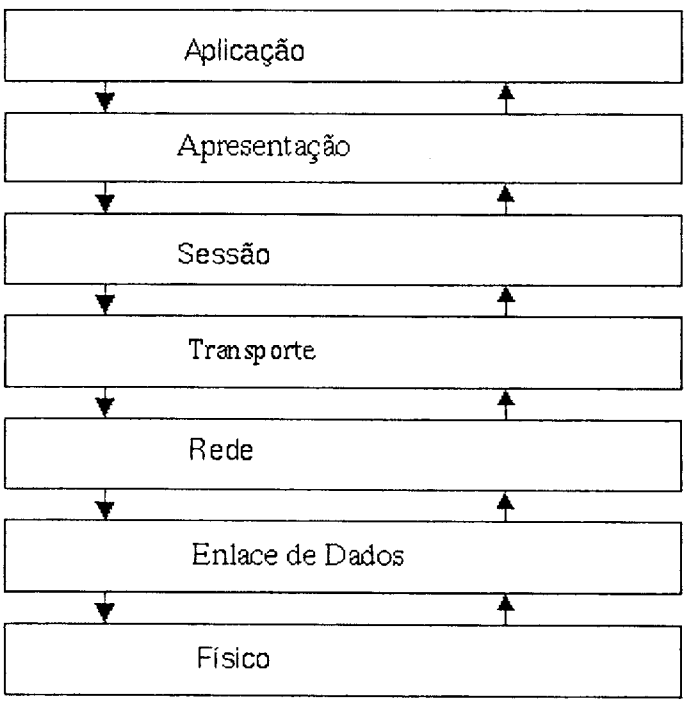

Fig 2.1 - Camadas TCP/IP modelo ISO/OSI

A implementação do protocolo de comunicação do modelo ISO/OSI no LINUX mantém as camadas por funcionalidade (Fig 2.2). Assim teremos uma

camada Física, que será responsável pelos sinais elétricos que irão trafegar pelo meio físico, representado pela placa de rede; a camada de Enlace de Dados, que irá cuidar do controle da placa de rede, sendo responsável pela comunicação ponto-a-ponto e pela montagem dos frames que serão enviados; a camada de Rede, responsável pela análise e diferenciação dos tipos dos dados que trafegam e também pelo encaminhamento de pacotes por caminhos diferenciados; e a camada de Aplicação, que são as chamadas dos usuários de serviços de rede.

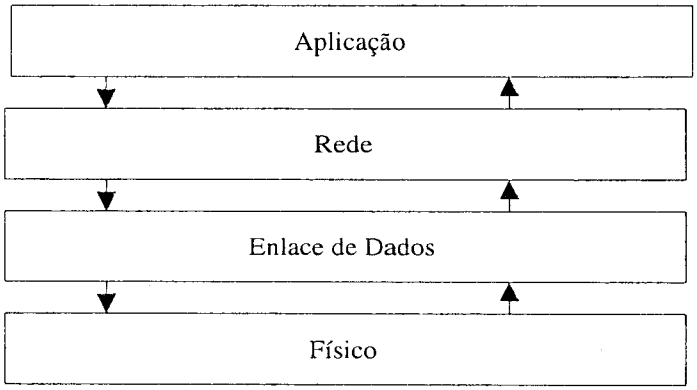

Fig 2.2 - Camadas TCP/IP nos sistemas UNIX

Se analisarmos o sistema de comunicação como *chamadas de função*, iremos verificar que a comunicação entre uma aplicação do usuário e a placa de rede segue o seguinte trajeto: Primeiramente, no caso da recepção de um pacote, a placa de rede receberá o pacote e após verificá-lo irá gerar uma interrupção que por sua vez ativará o sistema de gerência de interrupções do sistema operacional, no caso do LINUX chamado de *top half* [RUBINI 98]. Este sistema irá identificar a natureza da interrupção, colocá-lo na fila de atendimento e retornar ao processo normal do sistema. Quando for escalonado, o tratador de pacotes de rede irá identificar qual o pacote recebido e ativará a rotina de tratamento específica do protocolo correto.

É importante notar que normalmente os dispositivos existentes num computador são tratados pelo UNIX como arquivos dentro do sistema

operacional. Este é o caso dos discos rígidos, unidades de fita magnética e discos flexíveis, representados como dispositivos de bloco (*block devices*). Os dispositivos de bloco são controlados através de comandos enviados pelo *kernel* do sistema operacional. Esta característica passiva de funcionamento é que permite que eles sejam representados como arquivos, pois nunca existirá um pedido procedente de um dispositivo de bloco. Eles somente *receberão* comandos. No caso das placas de rede, todas as vezes que elas recebem um pacote elas devem acionar a rotina de tratamento, ativamente (Fig 2.3). Esta característica impede que elas sejam tratadas como um dispositivo de bloco.

```
extern int netcard_probe(struct device *dev);
static int netcard_probe1(struct device *dev, int ioaddr);
static int net_open(struct device *dev);
static int net_send_packet(struct sk_buff *skb, struct device *dev);
static void        net_interrupt(int irq, void *dev_id, struct pt_regs *regs);
static void        net_rx(struct device *dev);
static int net_close(struct device *dev);
static struct      net_device_stats *net_get_stats(struct device *dev);
static void        set_multicast_list(struct device *dev);

/* Example routines you must write ;->. */
#define tx_done(dev) 1
extern void        hardware_send_packet(short ioaddr, char *buf, int length);
extern void        chipset_init(struct device *dev, int startp);
```

Fig 2.3 - Funções existentes num controlador de uma placa de rede

2.2- Sistema de Comunicação na camada física

Para fins de estudo, consideramos aqui a camada física do protocolo como sendo aquelas funções realizadas dentro da placa de rede, seus buffers e circuitos integrados.

Apresentamos a seguir os blocos funcionais e os processos de recepção e envio de um pacote de uma placa de rede padrão *Ethernet* típica, que utilizamos no

nossa avaliação [NATIONAL 95].

Desserializador de recepção

Inicialmente os dados de recepção seriais são colocados no verificador/gerador de CRC (*Cyclic Redundancy Check - verificação de redundância cíclica)*. O desserializador de recepção também inclui um detector de sincronia que detecta o SFD (*Start of Frame Delimiter - Delimitador de início de frame*) para estabelecer onde estão localizados os limites do *byte* dentro da cadeia de bits seriais que compõe o preâmbulo. Sempre após oito *clock*s de recepção, os dados no tamanho de um *byte* são transferidos para a FIFO de 16 *byte*s e o Contador de *byte*s recebidos é incrementado. Os primeiros 6 *byte*s depois do SFD são checados comparando-se com ao Lógica de Reconhecimento de Endereço. Se a Lógica de Reconhecimento de Endereço não reconhece o pacote como um pacote destinado para aquele computador, a FIFO (*First In / First Out - estrutura de dados tipo pilha*) é apagada.

Verificador/gerador de CRC

Durante a *transmissão*, a lógica de CRC gera um campo local de CRC para a seqüência de bits transmitida. O CRC codifica todos os *byte*s depois do *byte* de *synch.* O CRC é transmitido para fora do gerador de CRC acompanhando o último *byte* transmitido. Durante a recepção a lógica de CRC gera um campo de CRC para o pacote sendo recebido. Este CRC local é serialmente comparado com o CRC colocado no fim do pacote enviado pelo computador de origem. Se ambos os campos de CRC forem iguais, um padrão específico será gerado e decodificado para indicar que nenhum erro nos dados foi encontrado. Erros de transmissão resultam num padrão diferente e assim são identificados, levando à rejeição do pacote.

Serializador de Transmissão

O Serializador de Transmissão lê dados paralelamente da FIFO e serializa-os para transmissão. O serializador é sincronizado pelo *clock* de

transmissão gerado pela Interface Serial de Rede (DP8391). Os dados seriais de entrada também são enviados ao verificador/gerador de CRC. No início de cada transmissão o gerador de Sincronização e de Preâmbulo acrescenta 62 *bytes* de 1,0 com função de preâmbulo e 1,1 de sincronização. Depois que o último *byte* de dados do pacote foi serializado o FCS (*Frame Check Sequance - Sequência de checagem de Frame)* de 32 bits é diretamente retirado do gerador de CRC. No caso de uma colisão o gerador de Preâmbulo e Sincronização é utilizado para gerar um padrão de JAM (colisão) formado de de bits em nível 1.

Lógica de Reconhecimento de Endereços

A lógica de reconhecimento de endereços compara o campo de Endereço Destino (primeiros 6 *bytes* do pacote recebido) com os registros de Endereço Físico armazenados no Array de Registro de Endereço. Se qualquer um dos 6 *bytes* não combinar com o endereço físico pré programado, a lógica de Controle de Protocolo rejeita o pacote. Todos os endereços de destino de *multicast* são filtrados utilizando uma técnica de *hashing*. Se o endereço de *multicast* indexar um bit que foi registrado no filtro de endereços de *multicast*, o pacote é aceito, caso contrário é rejeitado pela Lógica de Controle de Protocolo. Cada endereço de destino também é checado com todos os 1's que são reservados para o endereço de *broadcast*.

A FIFO e a lógica de controle da FIFO

A placa de rede possue uma FIFO de 16 *bytes*. Durante a transmissão a DMA escreve os dados diretamente na FIFO e o Serializador de Transmissão lê dados da FIFO para transmiti-los. Durante a recepção o Desserializador de Recepção escreve dados na FIFO e a DMA lê dados da FIFO. A Lógica de Controle da FIFO é usada para contar o número de *bytes* na FIFO para que depois de um nível pré ajustado, a DMA possa começar a utilizar uma acesso a via de dados (*BUS*) e ler/escrever dados da/para FIFO antes que uma sub carga/sobre carga (*underflow/overflow*) ocorra.

Devido ao fato da placa de rede precisar guardar o campo de endereços de cada pacote que chega para determinar se o pacote combina com os Registro de Endereço Físico ou se este mapeia um dos endereços nos registradores de endereços de Multicast, a primeira transferência local de DMA não ocorre até que 8 *bytes* tenham sido acumulados na FIFO.

Para garantir que não haja nenhuma sobreposição de dados na FIFO, a lógica da FIFO registra uma sobrecarga da FIFO quando o 13°. *byte* é escrito. Este procedimento efetivamente reduz a FIFO para 13 *bytes*. Além disso a lógica da FIFO opera diferentemente no *modo de byte* e no *modo de word*. No *modo de byte*, o início é indicado quando o *byte* n + 1 entrou na FIFO, assim, um início de 8 *bytes*, a placa de rede lança um Pedido de Via de Dados (*Bus Request - BREQ*) quando o 9°. *byte* entrou na FIFO. Para o *modo de word*, o BREQ não é ativado até que n + 2 *byte* tenham entrado na FIFO. Assim, com um início de 4 *word* (equivalente a um início de 8 *bytes*). O *BREQ* é ativado quando o 10° *byte* entrou na FIFO.

O PLA de Protocolo

O PLA (circuito composto de portas lógicas que pode ser programado) de protocolo é responsável pela implementação do protocolo 802.3, incluindo a recuperação de estados de colisão com atrasos dinâmicos. O PLA de protocolo também formata o pacote durante a transmissão e retira o preâmbulo e o sinal de sincronização durante a recepção.

Lógica de Controle de Buffer e DMA

A lógica de controle de Buffer e de DMA é utilizada para controlar dois canais de 16 bits DMA. Durante a recepção, a DMA local armazena os pacotes num buffer de recepção em forma de anel, localizado na memória de buffers. Durante a transmissão a DMA local usa o ponteiro programado e os registradores de tamanho para transferir um pacote da memória do buffer local para a FIFO. O segundo canal de DMA é utilizado como um DMA escravo para transferir os dados da memória do buffer local para o computador. A DMA local e a DMA remota são arbitradas internamente, com o canal de DMA local tendo a maior prioridade. Ambos os canais

de DMA utilizam um *clock* externo para gerar toda a temporização da via de dados. A arbitração interna é realizada com uma chamada de *bus* comum, o protocolo de comunicação da via de dados.

Encapsulamento de um pacote a ser transmitido/
Desencapsulamento de um pacote recebido

Um pacote típico do protocolo 802.3 consiste dos seguintes campos:

a) Preâmbulo
b) Delimitador de início de Frame (SDF)
c) Endereço de destino
d) Endereço de recepção
e) Tamanho
f) Dados
g) Seqüência de checagem de Frame (FCS)

Com o seguinte formato:

preâmbulo	SFD	destino	origem	tamanho	dados	FCS
62bits	2bits	6*bytes*	6*bytes*	2*bytes*	46-1500*bytes*	4*bytes*

Durante a operação de recepção, o preâmbulo e o Delimitador de Frame (SFD) - representados em tom mais claro - são retirados pela placa de rede, e as informações restantes são transferidas via DMA - representadas em tom mais escuro. Se tivermos uma operação de envio, a placa de rede será responsável pela geração do preâmbulo e do Delimitador de Frame (SFD) e também do FCS, que será acrescentado no final do frame a ser enviado.

Detalhamos agora as características de cada campo:

Preâmbulo e Delimitador de início de Frame

O SNI (DP8391) utiliza campo de preâmbulo de 1 e 0s alternados com a *codificação de Manchester* para conseguir sincronização a nível de bits com o pacote que está chegando. Quando é transmitido cada pacote possui um preâmbulo de 62 bits de 0s e1s alternados. Parte desse preâmbulo pode ser perdido quando o pacote trafega pela rede. O campo de preâmbulo é retirado pela placa de rede. O alinhamento do pacote é realizado com o reconhecimento do padrão do Delimitador de Início de Frame (SFD) que consiste de dos 1s consecutivos. A placa de rede não trata o SFD como um *byte*, ele detecta apenas o padrão composto de dois 1s. Isto permite que qualquer preambulo que preceda o SFD seja usado como sincronizador.

Endereço de Destino

O endereço de destino indica o destino de um pacote na rede e é utilizado para filtrar pacotes não desejados de entrarem no computador. Há três tipos de endereços suportados pela placa de rede: físico (*unicast*), *multicast* e *broadcast*. O endereço físico é o endereço único que corresponde a um único computador. Estes endereços são comparados com registros de endereços físicos armazenados internamente. Cada bit no endereço de destino deve combinar com o armazenado para que a placa de rede aceite o pacote. Endereços de *multicast* começam com um MSB de "1". O DP9380C filtra os endereços de *multicast* usando um algoritmo de *hashing* padrão que mapeia todos os endereços de *multicast* num valor de 6 bits. Se o endereço consiste somente de 1s então ele é um endereço de *broadcast*, indicando que aquele pacote é destinado a todos os nós. Se a placa de rede estiver no modo promíscuo todos os pacotes serão aceitos: não é necessário que o campo do endereço de destino combine com nenhum dos registros armazenados. Os modos físico, *multicast*, *broadcast* e promíscuo podem ser selecionados.

Endereço de origem

O endereço de origem é o endereço físico do computador que enviou o pacote. O endereço de origem não pode ser nem um endereço de *broadcast* nem um endereço de *multicast*. Este campo é simplesmente transferido para a memória do buffer.

Campo de tamanho de dados

Este campo de dois *bytes* indica o número de *bytes* que está dentro do campo de dados. A placa de rede não interpreta esse campo.

Campo de Dados

Este campo tem o tamanho variando entre 46 *bytes* e 1500 *bytes*. Se uma mensagem tiver um tamanho superior a 1500 *bytes* seu conteúdo deverá ser dividido em vários pacotes. Se a mensagem for menor que 46 *bytes* a área de dados deverá ser preenchida até que o campo de dados fique com o tamanho mínimo de 46 *bytes*. Se houver este preenchimento, o número real de *bytes* de dados estará indicado no campo de tamanho da área de dados. Não é a placa de rede que retira ou preenche o campo de *bytes* com o mínimo de *bytes* necessários. Nem é ela que checa pacotes com excesso de tamanho.

Campo de checagem de Frame

A seqüência de checagem de frame (FCS) é um campo de 32 bits de CRC calculado e acrescentado ao pacote durante a transmissão para que o computador destino consiga identificar erros durante o tráfego. Durante a recepção, pacotes sem erros resultam num padrão conhecido de CRC Pacotes com CRC errado serão rejeitados.

Funcionamento do *Ethernet*: a recepção de um pacote

Quando existe atividade na rede, os bits são transferidos para o Desserializador de recepção. O desserializador procura SFD com seu detector de

sincronia depois do preâmbulo e transfere os 6 *bytes* que vem em seguida para dentro da Lógica de Reconhecimento de Endereços, para certificar-se que aquele pacote pertence ao computador local. O pacote também é posicionado no Verificador/Gerador de CRC. Para certificação de que é um pacote válido. A cada 8 sinais de *clock*, um *byte* é transferido para a FIFO de 16 *bytes* e o Contador de *bytes* é incrementado. Se a Lógica de Reconhecimento de Endereços ou o Verificador/Gerador de CRC não reconhecer o pacote como um pacote válido, o pacote é descartado e a FIFO é apagada. A Lógica de controle de DMA é responsável pela transferência dos dados da FIFO para a memória do computador.

Funcionamento da *Ethernet*: a transmissão de um pacote

Quando existem dados na FIFO a serem transmitidos, o Serializador de Transmissão é ativado. Ele lê dados paralelamente da FIFO e os serializa para envio. O pacote também é enviado para o Verificador/Gerador de CRC, para que seja gerado um FCS que será colocado no fim do pacote. Antes do envio o gerador de Padrões de Preambulo e Sincronização gera o preâmbulo (62 bits) e a sincronização (2 bits em 1).

2.3- Sistema de Comunicação na camada de Enlace de Dados

Para compreender a maneira pela qual o dispositivo de rede se comunica com o *kernel*, vamos acompanhar o trajeto de um pacote desde o *buffer* da placa de rede até a função de tratamento deste pacote dentro do protocolo específico.

Suponhamos a chegada de um *frame*, agora armazenado como pacote dentro do buffer da placa de rede. Neste momento, o hardware da placa de rede sinaliza o *kernel* através de sua linha de interrupção que há um pacote esperando no buffer para ser tratado. É assim acionado no *kernel* a rotina chamada de TOP HALF, que é uma rotina de tratamento de interrupções bastante curta, pois deve ser executada o mais rápido possível. A razão desta urgência é que qualquer outra

interrupção que ocorrer durante o tratamento de uma interrupção não será atendida pelo sistema.

O TOP HALF analisa qual o tipo de interrupção que foi acionada e então chama o conjunto de funções localizado no BOTTOM HALF. Este é um tratador de interrupções que funciona com prioridades semelhantes àquelas dos outros processos, não influenciando assim diretamente no controle das outras interrupções. O BOTTOM HALF vai analisar o pedido de tratamento e detectar qual será a função de tratamento específica de cada protocolo que deverá ser chamada. Neste momento, entram as funções de tratamento de pacote.

A funções de tratamento de pacote

Antes de chegar até a aplicação do usuário, o pacote ainda passará pelas rotinas de manipulação do protocolo específico (Fig 2.4). Estas rotinas, executadas pelo kernel, utilizam uma área de memória reservada, chamada de *memória do Kernel.*

Suponhamos a transmissão de uma mensagem entre dois computadores utilizando a rede. No esquema tradicional, uma rotina *send()* é chamada pelo usuário para o envio da mensagem. Esta rotina irá ativar um *system call na área do kernel* e então a mensagem na área de memória do usuário será transferida para a área de memória do kernel, através de uma cópia de bloco de memória. Este procedimento consome tempo, relativamente grande quando abordamos a comunicação em rede. O kernel irá então tratar a mensagem, através das rotinas de protocolo, ativar o controlador de rede e este então irá enviar a mensagem para a estação destino.

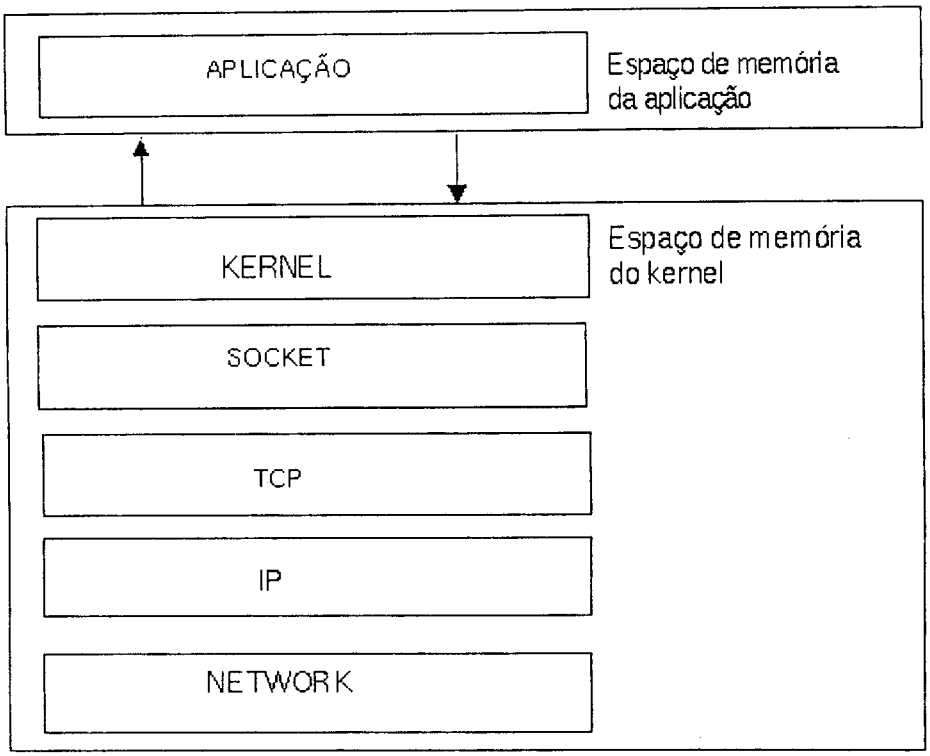

Fig 2.4 - Espaços de memória e as camadas de rede

Em [EICKEN 98] são destacados fatores que devem ser abordados na pesquisa por protocolos mais eficientes.

O primeiro deve ser a baixa **latência nas comunicações**. Esta latência ocorre principalmente devido ao tempo consumido durante a gerência dos *buffers* de memória, cópia de mensagens, cálculo de *checksums* e manipulação do controle de fluxo e das interrupções, além do controle da própria placa de rede.

No campo da computação paralela, muitos pesquisadores de redes de computadores utilizam estas técnicas para prover maior capacidade real de processamento. Porém programas paralelos em rede são difíceis de programar e seu uso eficiente exige que o custo de comunicação em termos de tempo gasto para transmissão diminua uma ordem de magnitude para atender esta necessidade [EICKEN 98].

Outras aplicações de computação distribuída, como servidores de

páginas HTTP, por exemplo, necessitam de **grande largura de banda para mensagens pequenas**, tipicamente com tamanhos próximos de 1 K*byte*. Neste caso o desafio é diminuir o *overhead* de processamento ao máximo, para contemplar o menor tamanho de mensagem possível. Esta abordagem irá também aumentar a velocidade de comunicação total, pois a utilização de *buffers* em protocolos como o TCP/IP é uma causa da latência das comunicações. O tamanho da janela de comunicação (*windows size*) é bastante afetada pela largura de banda e o tempo de comuncação das mensagens. Analogamente, é como se cada pacote fosse um trem, e o tempo de latência o tamanho da locomotiva que puxaria este trem, no trajeto entre duas estações. Se a locomotiva é grande, muitos trens pequenos são ineficientes, pois teremos longas locomotivas ocupando a linha com trens pequenos, provocando uma demora maior para atravessar o trecho entre as duas estações.

O desenvolvimento de aplicações de rede mais eficientes não é sempre possível porque não temos **protocolos de comunicação flexíveis**. É necessário que novos modelos de protocolo ofereçam duas funções básicas: integrar a aplicação com o controle de *buffers* e otimizar o fluxo de informações dentro do protocolo [EICKEN 98].

Algumas técnicas de projeto de protocolos avançados devem incluir funcionalidades como o *framing* dentro da aplicação, onde esta poderá controlar o esquema de armazenamento intermediário (*buffering*) de acordo com as características de seu processamento. Também deve existir a possibilidade de *processamento em camadas integradas*, que é a integração das funcionalidades de diversas camadas dentro de uma única camada funcional.

Além destas técnicas, pode ser oferecido um **compilador** para o protocolo de rede , que seja capaz de integrar o processamento específico da aplicação com as características necessárias de comunicação para tornar todo o sistema mais eficiente.

Outras técnicas sugeridas incluem a **geração do controle completo de todo o sistema de comunicação** por um compilador específico, incluindo as camadas mais baixas do protocolo, Ele se utiliza da verificação da tecnologia de rede disponível e é capaz de gerar protocolos otimizados para o sistema de comunicação e para a aplicação, levando em conta as características de processamento e de troca de informações. Para isso é necessário que todo o sistema seja implementado em único espaço de endereçamento de memória.

Uma das opções para aumento da velocidade de comunicação é a eliminação da cópia entre a memória do usuário e a memória do kernel. Este modelo é chamado de Arquiteturas de Comunicação de Nível de Usuário, que apresentamos a seguir:

2.4- Arquiteturas de Comunicação de Nível de Usuário

A origem das arquiteturas de nível de usuário baseia-se no modelo de passagem de mensagem já utilizado em multicomputadores. Para se comunicar neste modelo, é necessário especificar o endereço de memória de origem e o nó de processamento de destino, e então o sistema de comunicação realizará uma transferência diretamente para a área de memória do processador destino. Porém este processo necessita que as mensagens sejam colocadas em *buffers*, o que gera uma cópia de memória ou realiza um processo de estabelecimento de comunicação (*handshake*) demorado.

Para atender este problema desenvolveu-se o sistema de comunicação denominado *Active Messages* [CULLER 95]. Este sistema modificava o funcionamento da comunicação fazendo com que as mensagens que chegavam ao sistema de comunicação fossem disparadas imediatamente, sem ter que ser armazenadas para posteriormente serem enviadas. A ativação da transmissão dava-se justamente pela chegada da mensagem. Sobre o Active Messages desenvolveu-

se posteriormente o Fast Sockets [RODRIGUES 96]. O protocolo oferece a funcionalidade de STREAMS para o Active Messages, sendo utilizado em computadores como o *CM-5* [HILLIS 93].

Os protocolos tradicionais acrescentam sobrecarga (*overhead)* na transmissão de dados, tanto no envio (geralmente uma chamada de sistema e cópia de dados entre regiões de memória) e na recepção (uma interrupção, uma chamada de sistema e uma cópia de dados entre regiões de memória). Novos protocolos podem ser desenvolvidos, buscando-se diminuir estas sobrecargas do sistema. Basicamente, estas novas arquiteturas deverão eliminar a participação do sistema operacional o máximo possível, retirando as chamadas de sistema, interrupções e cópias entre regiões de memória. Uma dessas implementações são os sistemas de comunicação chamados de **Arquiteturas de Nível de Usuário** (*User Level Architectures*). São chamadas assim por que utilizam a memória do usuário com pouca ou nenhuma intervenção do *kernel* e não realizam operações de cópias de blocos de memória

O modelo típico de arquiteturas de comunicação a nível de usuário é mostrado na figura a seguir: (Fig 2.5):

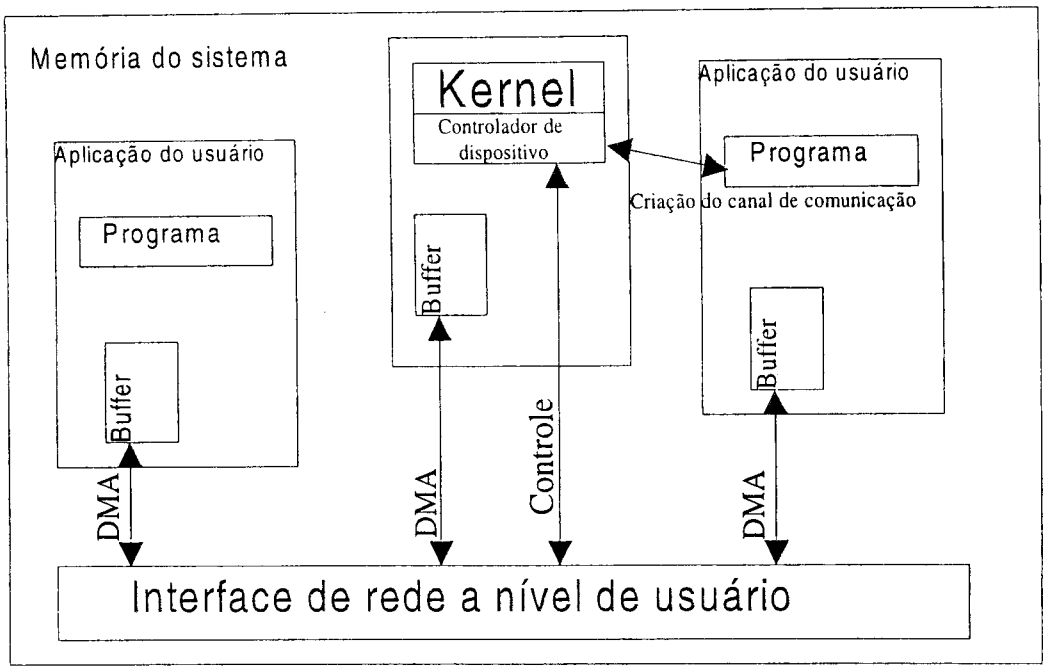

Fig 2.5 - Modelo esquemático da Arquitetura de Nível de usuário

Em [BHOEDJANG 98] temos a descrição de seis considerações importantes para implementação de protocolos de comunicação para arquiteturas de nível de usuário. A primeira delas é qual o mecanismo de *transferência de dados* que será utilizado pois a banda de comunicação e a latência serão influenciados por estes aspectos.

O uso de acesso direto à memória oferece mais velocidade, mas para que este mecanismo funcione é necessário prover entra as funções do sistema de comunicação um mecanismo de *tradução de endereços.*

Num ambiente de multiprogramação é necessário que o sistema de comunicação ofereça mecanismos de *proteção* que não dependam do sistema operacional

Os pacotes de rede são recepcionados e transferidos usando-se um mecanismo de *controle de transferências*, geralmente *polling* ou interrupções, cada um dos quais possuindo suas vantagens e defeitos.

As duas últimas características são *confiabilidade* e suporte para *multicast*, que podem ser garantidos dependendo das características das placas de rede e da infra-estrutura de comunicação existente.

Transferência de Dados

As transferências de dados otimizadas utilizam pelo menos 3 passos: O primeiro é da estação de trabalho para a interface de rede, a segunda é entre interfaces de rede de estações de trabalho diferentes e a terceira é da interface de rede destino para a memória da estação de trabalho destino, onde os dados são armazenados temporariamente na memória da própria interface de rede. Algumas tecnologias de interfaces de rede permitem a transferência de dados diretamente da estaçao de trabalho para a interface de rede e da interface de rede para a estação de trabalho, que utilizam o próprio mecanismo de entrada e saída (*outb/ inb*) presentes no kernel. Nestes caso não é utilizado o DMA, que depende da existência de uma área de memória na interface de rede. A decisão pela utilização de mecanismos de **I/O do kernel** ou **DMA** depende do *hardware* do sistema.

Por exemplo, no caso dos processadores denominados *Pentium Pro,* da *Intel Corporation*, o mecanismo de I/O do kernel pode ser mais veloz que o DMA, uma vez que o *hardware* oferece um mecanismo chamado de *write combining buffers*, que faz com que numa única utilização do *bus* de dados possam ser realizados vários comandos *write*.

O mecanismo de DMA possui uma outra desvantagem. Se durante a transferência de dados houver uma *paginação* do bloco de memória no sistema operacional, o processo será interrompido e toda a comunicação terá que ser refeita. Isso porque o DMA ocorre de forma assíncrona, sem controle por parte do sistema operacional.

Uma forma de se contornar este problema é permitir que o usuário

grampeie (*pin*) as páginas de memória que serão transmitidas por DMA e assim o sistema operacional não irá paginá-las. Porém a quantidade de páginas que podem ser *grampeadas* é limitada pela memória física disponível e também pelas políticas do sistema operacional. Para se *grampear* uma certa quantidade de memória, o processo de alocação deve ser realizado durante a inicialização do kernel.

No caso do I/O do kernel, mesmo que o sistema operacional realize uma *paginação*, a próxima referência a uma página de memória faltante irá gerar um *page fault*, fazendo com que o sistema operacional retorne a página para a memória novamente e os dados sejam transferidos corretamente.

No processo de especificação de como funcionarão as *transferências de dados,* um outro aspecto importante é o tamanho dos pacotes. A transmissão de pacotes grandes oferece um desempenho melhor, pois no caso de fragmentação em pedaços maiores diminuímos a latência que existe para a criação do *header* de cada pacote e sua respectiva transmissão. Porém devemos também levar em conta durante o projeto deste mecanismo características como o tamanho das páginas de memória do sistema, problemas de ocupação de espaço em memória e restrições de *hardware*.

No caso da **transmissão entre interfaces de rede** as especificações a serem consideradas dependem da tecnologia de *hardware* adotada (*ATM, Fast Ethernet, Mirynet*). Nas implementações apresentadas como exemplo no próximo capítulo, estas características são discutidas.

Tradução de Endereços

O uso de mecanismos de transferência do tipo DMA implica em garantir-se que as páginas de memória estarão *grampeadas* e que os endereços físicos de memória para transferência de dados sejam bem conhecidos.

Normalmente o sistema operacional não divulga os endereços de memória para os usuários para que chamadas de funções possam utilizar este dado como parâmetro. Mesmo que estes estivessem disponíveis, seria necessário verificar junto ao sistema se aqueles endereços realmente podem ser utilizados por um determinado usuário.

Uma das maneiras de se resolver este problema é evitar-se o uso de DMA e utilizar I/O do kernel. Porém este recurso somente pode ser utilizado pelo processo que está enviando, pois a recepção utilizará DMA com as tecnologias de rede existentes.

É possível também copiar os dados do usuário em áreas de memória especiais dentro do sistema de DMA. Estas áreas de memória só precisam ser *grampeadas* uma vez, quando o processo do usuário iniciar o acesso a interface, e não a cada uma das operações de recepção ou envio. Neste caso a tradução de endereços é fácil, pois o sistema operacional aloca áreas de memória contínuas e passa para a interface de rede o endereço físico desta área. O usuário fará o controle dos *buffers* de transmissão informando apenas um *deslocamento* (*offset*) em relaçao ao endereço original. Este mecanismo, relativamente de implementação simples, tem o problema de envolver uma **cópia de memória**, que consiste num gargalo de transmissão.

Uma solução diferente é a própria aplicação ou uma biblioteca alocar e desalocar dinamicamente as páginas que serão utilizadas pelos *buffers* de envio e recebimento. A placa de rede irá transferir dados diretamente para essas páginas por DMA. Neste caso, é necessário que a interface tenha o mapeamento dos endereços virtuais para os endereços físicos. Uma das maneiras de se fazer isso é oferecer um módulo que forneça o endereço físico a cada alocação realizada pelo usuário.

Proteção

Em sistemas de comunicação ao nível do usuário, o usuário escreve na memória da interface de rede para iniciar os descritores do processo de envio. Somente um usuário pode escrever de cada vez, pois senão corre-se o risco de um usuário estragar os descritores de outro usuário. Porém esta é uma solução de protocolo não desejável.

Uma das soluções é utilizar o sistema de mapeamento virtual de memória para oferecer uma área de trabalho específica para cada usuário. Toda vez que o usuário abre o dispositivo, uma área específica é mapeada no espaço de endereços do usuário.

Porém a área de memória dentro das interfaces é pequena, e isso limita o número de processos que podem ser executados simultaneamente. Para economizar esta memória, o sistema de comunicação pode mapear na memória de rede somente os processos comunicantes que estão ativos, mantendo os inativos na memória do *host*

Com o uso de DMA, os processos dos usuários obtém proteção *grampeando* suas áreas individualmente.

Controle de Transferência

Em protocolos de comunicação ao nível do usuário qualquer desperdício de tempo é motivo de preocupação. Assim, o uso de interrupções pode torna-se um mecanismo inadequado uma vez que o tempo de espera pode chegar a 10 microssegundos. Todas as arquiteturas de sistemas de comunicação possuem suporte para algum tipo de *polling*.

A utilização de *polling* visa oferecer ao *host* um mecanismo para verificar se mensagens chegaram pela rede. Esta função deve ser rapidamente

executada, uma vez que será executada com freqüência suficiente para não perder nenhum pacote que tenha chegado pela rede.

Em geral este mecanismo é implementado através de um sinalizador (*flag*) implementado na memória da interface de rede. Porém esta solução não é tão interessante uma vez que para checar a existência de um pacote haverá a necessidade de usar o duto de comunicação (*bus*), causando o aumento de operações de entrada e saída.

Uma solução para este caso seria colocar o sinalizador dentro da memória do host através de DMA.

Vale lembrar que o controle de *polling* deverá ser feito pelo usuário programador, utilizando as rotinas de manipulação oferecidas pelo sistema de comunicação.

Além disso, todo mecanismo de *polling* consome grande tempo de CPU, monopolizando-a, o que não é interessante para *clusters* de processamento científico, sendo assim mais adequado o mecanismo que utiliza DMA.

Confiabilidade

O sistema de comunicação em geral assume que a rede é suficientemente confiável e deixa o controle de fluxo a cargo do usuário-programador. Este critério é válido pois as redes de comunicação atuais oferecem baixas taxas de erros. Quando estes erros ocorrem são geralmente fatais.

Deve ser também considerada a possibilidade da interface de rede descartar pacotes devido a problemas de *overflow*. Um sistema possível para se recuperar de *overflow* é a existência de mensagens de controle do tipo ACK (reconhecimento). A cada pacote recebido um pacote de reconhecimento é enviado à estaçao de origem. Caso ocorra um *overflow* um pacote pacore tipo NACK é

enviado. Neste caso deve-se assumir que a interface de rede nunca descarta pacotes que sejam do tipo ACK ou NACK.

Outro mecanismo de controle de *overflow* possível é a prevenção de *overflows*. O protocolo do sistema de comunicação avisa a estação transmissora de origem caso o destinatário esteja ficando sem *buffers* de recepção.

Um outro sistema possível para prevenção de *overflows* é a utilização de um sistema de créditos de envio. Um sistema remetente só pode mandar seus pacotes para o destinatário caso ele possua um crédito para isso, indicando que o espaço existe na estação destino.

Multicast

A utilização de *multicast* em ambientes chaveados é uma área onde existe bastante pesquisa científica. O modo mais simples de se implementar *multicast* é enviar os dados **várias vezes para a placa de rede** e esta irá criar, entre o nó de origem e os nós de destino, várias comunicações ponto-a-ponto. Contudo esse processo envolve a transmissão dos dados primeiramente para a memória da interface de rede, após uma cópia realizada para a área de DMA.

Uma outra solução um pouco melhor é passar os dados **uma única vez** para a placa de rede juntamente com os endereços de destino, e essa irá então realizar as várias comunicações ponto-a-ponto. Porém essa solução ainda apresenta a interface de rede como um gargalo serial de transmissão de mensagens.

Capítulo 3

Estudo de Sistemas de Comunicação existentes

3.1- O U-Net

A proposta do U-Net [WELSH 97] é levar para o nível do usuário o processamento de protocolo. O usuário pode acessar o dispositivo de rede diretamente através de um mecanismo simples. Este processo oferece a possibilidade do programador integrar melhor as funções de comunicação às necessidades de sua aplicação, levando a um melhor desempenho.

Ele diferencia-se do *Active Messages* [RODRIGUES 96] pois enfoca os sistemas de rede em redes locais (LAN) e não como sistemas de comunicação de computadores baseados em transferência de mensagem. Desta forma ele busca aproveitar as características das redes locais, tanto quanto ao seu funcionamento como arquitetura. O sistema *Active Message* oferece um conjunto de funções no sistema disponíveis para os processos. O U-Net oferece uma estrutura de filas dentro da memória. Enquanto o *Active Messages* oferece uma camada de comunicação uniforme de utilização da rede, O U-Net especifíca as operações do *hardware* de comunicação permitindo que o usuário utilize os recursos de *hardware* de uma forma uniforme.

Os projetistas do U-Net tinham como objetivo diminuir a latência da recepção e envio de mensagens através da rede e também diminuir o custo da construção de clusters de processamento paralelo, utilizando para isso placas de comunicação baratas como *Fast Ethernet*. Eles tinham como objetivo também oferecer transporte eficiente e de baixa latência para transporte de pequenas mensagens, comuns a sistemas NFS (Network File System), e objetos distribuídos em sistema orientados a objetos.

Arquitetura de comunicação

O sistema oferece a cada usuário a interface de comunicação de rede como se ele fosse o único a se utilizar deste recurso. O usuário tem controle sobre o conteúdo das mensagens que são tansmitidas e também dos processos de envio e de recebimento. O sistema U-Net cuida da multiplexação do recurso de comunicação entre os diversos usuários.

No U-Net utilizamos 3 entidades básicas: Os portos (*endpoints*), as filas de mensagens (*message queues*) e os canais de comunicação (*communication channels*). Um porto é uma estrutura que contém as filas de mensagens (tanto de recepção como de envio) e uma área de *buffer* para as mensagens propriamente ditas. Um canal de comunicação é identificado por um par de portos e um identificador.

Uma mensagem utiliza o identificador do canal de comunicação e um rótulo de mensagem (*message tag*) para identificar a origem e destino únicos de uma mensagem.

Para se enviar uma mensagem o usuário deve compor os dados dentro da área de memória do porto e colocar um descritor na fila de mensagens de envio. Neste momento a interface de rede transmite a mensagem depois de colocar nela o rótulo apropriado.

No caso de recepção de mensagens, elas são tratadas de acordo com seu rótulo. Os dados são então colocados em uma ou mais áreas livres de memória, especificamente a área de memória que pertence àquele porto identificado através do rótulo. Um descritor apontando para a área de memória onde está a mensagem recebida é colocado então na fila de mensagens recebidas. O U-Net oferece como otimização a possibilidade de colocação do corpo da mensagem inteira dentro da fila de descritores de mensagens recebidas, uma vez que grande parte das mensagens

são de pequeno tamanho.

A implementação do U-Net para placas de rede do tipo *Fast Ethernet* não dispõe de um processador dedicado para cuidar da transmissão de dados entre os processos de usuários em diferentes nós de comunicação. Desta forma, a rotina que trata dos procedimentos de transferência de dados está implementada dentro do *Kernel*. Podemos observar a organização da comunicação do U-Net na figura a seguir:

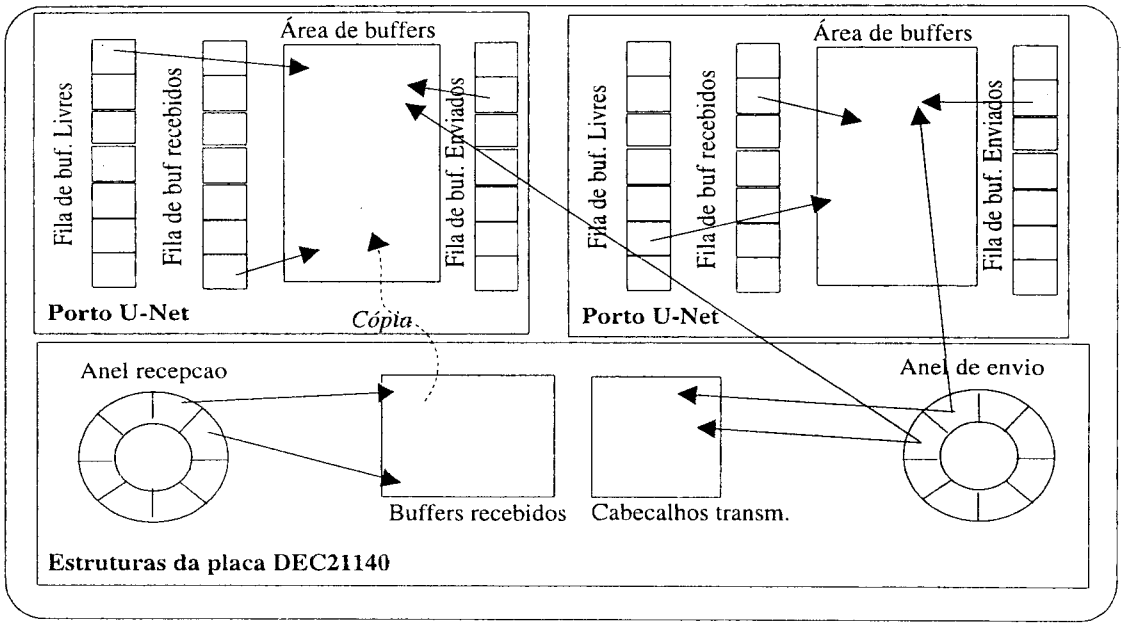

Fig 3.1 - Modelo esquemático – U-Net

Esta implementação pode ser considerada como uma sub rotina de proteção disponível para todos os processos. Ela irá garantir que não haverá perda de dados de usuários por falta de proteção para as áreas de memória. A diferença significativa com relação aos protocolos de comunicação tradicionais é que a área

de memória onde estão as mensagens a serem transmitidas está localizada dentro da área de memória do usuário e não na área de memória do *Kernel*. Isto evita uma cópia de dados entre estas duas categorias de memória, como ocorre nos protocolos tradicionais, diminuindo assim a latência.

Para existir comunicação entre dois processos é necessário primeiramente criar um canal de comunicação entre dois portos. Nesta implementação com *Fast Ethernet* os portos são identificados por uma combinação entre os 48 *bytes* correspondentes ao endereço de *MAC* da placa *Ethernet* combinados com 1 *byte* da identificação de porta do U-Net. Na arquitetura do U-Net/FE esta combinação é o rótulo da mensagem. A criação do canal de comunicação é feita passando-se como parâmetros os dois endereços de *MAC* e os identificadores de porta. Enquanto o *endereço* de *MAC* é utilizado para determinar qual é o computador destino da mensagem a *porta* identifica o porto ao qual ela deve ser entregue.

O sistema irá guardar este canal de comunicação que será referenciado pelo seu identificador. Todas as vezes que o sistema quiser realizar uma comunicação ponto-a-ponto ela colocará o identificador na fila de mensagens de envio.

Para enviar uma mensagem, o processo do usuário coloca o dado a ser transmitido dentro da área de memória do porto e coloca um descritor na fila de mensagens de envio.

O sistema então gera um *trap* no kernel, possível nas arquiteturas x86, que tem tempo de atendimento inferior a uma chamada de sistema (*system call*). Esta rotina irá percorrer a fila de mensagens de envio e para cada uma das entradas irá colocar o descritor no anel de envio do DC21140. Quando todos os descritores estiverem colocados no anel de envio, a rotina do kernel envia um comando de envio para a interface de rede, que realiza a trasmissão efetiva.

Durante a recepção, os pacotes são colocados no anel de recepção e a interface de rede gera uma interrupção que ativa a rotina de tratamento dos pacotes. Neste momento, a rotina do kernel verifica o pacote e, através do identificador de porta contido no cabeçalho (*header*), coloca a mensagem dentro da área de memória do porto adequado.

Considerações sobre os aspectos de implementação

No aspecto de **transferência de controle** o U-Net pode ser utilizado tanto orientado a eventos (com o uso de interrupções) como utilizando o método de varredura (*polling*). Ele implementa a varredura periódica da fila de mensagens recebidas, pode interromper o processamento até que chegue a próxima mensagem ou pode se utilizar de uma interrupção associada a fila de recebimento de mensagens, indicando quando seu estado se altera para fila não vazia. Como otimização, o U-Net permite que toda a fila seja esvaziada no atendimento de uma única chamada.

Considerando o aspecto de **tradução de endereço,** cada vez que a interface não encontra uma área de memória que está buscando para transferência, o sistema gera uma interrupção. O Kernel irá então procurar em sua tabela de páginas de memória por aquela página, *grampear* a página e passar o *endereço traduzido* para a interface de rede. Este mecanismo permite que a interface de rede receba o endereço físico que necessita para utilização de DMA, o que caracteriza o mecanismo de **transferência de dados** utilizado.

Quanto a **proteção** o sistema garante a integridade dos dados fazendo com o Kernel seja responsável pelo estabelecimento do canal de comunicações, garantindo assim que uma aplicação não irá invadir a área de trabalho da outra e

também não irá sobrepor dados dentro da área de memória interna da placa de rede.

A **confiabilidade** do sistema de comunicação está na consideração de que a rede é por si só bastante confiável para que a perda de pacotes seja mínima. Mesmo neste caso, cabe ao programador certificar-se da recepção da mensagem enviada.

O U-Net não oferece facilidades para **Multicast**. Assim, se o processo comunicante quiser se comunicar com mais de um destinatário, várias comunicações ponto-a-ponto (*unicast*) serão necessárias.

3.2- O MVIA

MVIA (*Virtual Interface Architecture)* pode ser considerado como uma evolução do sistema U-Net, possuindo suas orientações básicas de projeto. Oferece também as mesmas características de linguagens como VMMC, isto é, permite que o sistema seja utilizado como um replicador de conteúdos de memória, podendo assim trabalhar com modelos de programação de memória compartilhada.

O M-VIA é baseado na iniciativa do consórcio VIA, promovido por diversas empresas, entre elas a *Intel Corporation* e a *Microsoft* e *Compaq*. Em vez de um "padrão" o consórcio determina mais especificamente um conjunto de definições a serem seguidas para que seja implementado um sistema do tipo VIA. [MVIA FAQ]

A idéia de se criar um padrão para implementação de comunicação em *clusters* é um esforço para que existam cada vez menos soluções proprietárias.

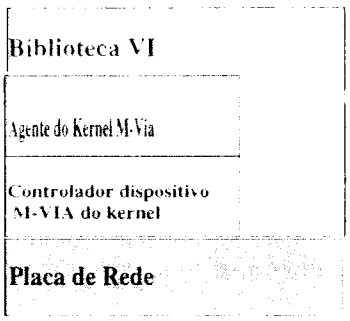

Fig 3.2 – M-VIA – Camadas de Controle

Como podemos observar, o sistema M-VIA implementa camadas de controle sobre a interface de rede. A primeira camada está dentro do próprio *driver* controlador (controlador de dispositivo), implementando neste nível as funções que serão utilizadas pelo programador. O sistema MVIA também coloca um agente dentro do kernel, que tem por função desviar as chamadas para a placa de rede

diretamente para suas funções de controle, não permitindo assim que o kernel assuma o controle e copie os dados a serem transmitidos da área de memória do usuário para a área de memória do kernel.

Neste sistema os processos do usuário agem diretamente sobre a interface de rede, através dos *daemons* implementados pelo M-VIA. Desta forma o sistema desvia das sobrecargas (overhead) causados pelo sistema TCP/IP e também evita que outros processos possam ler dados de outros usuários.

Podemos dizer que o VIA é composto de três partes: uma biblioteca de funções a nível de usuário, código do kernel que é capaz de estabelecer conexões protegidas e então devolve o controle para o usuário e a própria interface de rede.

A implementação original do VIA prevê a existência de um hardware específico, uma placa de rede que possua características originais e que permita, por exemplo, por si própria copiar os dados para dentro da área de memória do usuário diretamente, sem a intervenção do sistema operacional.

O M-VIA é uma implementação completa da proposta do VIA para sistemas baseados em LINUX. Ele é composto de um módulo para ser carregado para o kernel (*via_ka*) e uma biblioteca de funções para o usuário. Também o M-VIA necessita de um *driver* controlador de dispositivo de rede alterado (que também é um módulo do kernel). Ele também permite que o TCP-IP normal continue sendo usado na mesma interface.

Para executar um programa utilizando o sistema de comunicação M-VIA é necessário iniciar cada um dos processo manualmente, pois ainda não existe um processo do tipo *"viarun"* como no MPI. Depois que os processos são iniciados, eles se comunicam com um esquema *accept* e *connect*.

Para funcionar em máquinas SMP, normalmente utilizamos o *loopback*

do M-VIA, que é o módulo *via-lo*. Assim processos localizados dentro dos processadores de uma mesma máquina comunicam-se entre si com baixa latência.

Apesar da implementação do protocolo TCP/IP ser bastante rápida nas plataformas LINUX, deve-se observar que o uso do M-VIA para implementar as comunicações em um cluster tem pelo menos mais duas vantagens: primeiramente, a facilidade de se portar o código paralelo para outras plataformas diferentes do LINUX e depois a possibilidade de se ter o desempenho do M-VIA ainda melhor, quando surgirem as placas de rede específicas para utilização do VIA.

A implementação atual do M-VIA é totalmente livre de cópias de memórias no processo de envio, e tem uma cópia na recepção, o que é impossível de ser evitado sem o suporte de *hardware.*

Assim, as características do M-VIA se resumem em um projeto modular que permite facilidade quando for portado para outras plataformas; Otimização de funcionamento por *hardware* ou implementação total por software, dependendo da disponibilidade de *hardware*; Bom desempenho, através da utilização de *fast traps* do kernel e outras técnicas; Possibilidade de uso simultâneo com outros protocolos de rede tradicionais (*TCP/IP*) e compatibilidade total com as especificações do VIA.

O M-VIA também oferece o MVICH [MVICH 01] (*MPI for Virtual Interface Architecture*), uma implementação do MPI baseada em MPICH [MPICH 01]. Seu projeto inclui características como armazenamento de mensagens pequenas no receptor e funções específicas da implementação para o *hardware*, verificando por exemplo se os *buffers* dos usuários podem ser *grampeados* (*pinned*).

Para suportar protocolos com cópia-zero de memória, o MVICH oferece o registro dinâmico dos *buffers* dos usuários, verificando rapidamente se a memória foi registrada, para que qualquer acréscimo para processamento de memória seja desprezível.

Capítulo 4

Avaliação do Sistema de Comunicação a nível de usuário

4.1- Metodologia

Realizou-se a comparação dos tempos de comunicação utilizando-se o TCP-IP [COMER 95] tradicional e o pacote M-VIA [MVIA FAQ]. Para tanto, foi criado um programa utilizando-se a linguagem MPI que foi executado em cada um destes sistemas de comunicação.

O programa de testes consiste num modelo "ping-pong" simples. Dois processos "mestre" e "escravo" são criados em computadores (nós) diferentes. O processo "mestre" inicia a contagem do tempo e envia para o escravo pacotes de tamanho definido e o "escravo" devolve os mesmos pacotes para o mestre que verifica o tempo novamente e calcula a duração da comunicação entre os dois nós.

4.2 – Ambiente de testes

Para os testes foi utilizado o cluster pertencente ao grupo de pesquisa em programação paralela avançada do Instituto de Física de São Carlos, composto de 16 nós de processamento AMD K6 III 450Mhz com 256 Mbytes de RAM por nó. A estrutura de comunicação é composta de placas de rede 3COM de 100Mbits, suportadas pelo sistema de comunicação M-VIA. O sistema de comunicação ainda conta com uma chave de rede 3COM e o tráfego desta rede é isolado, isto é, somente os processos de comunicação dos programas de testes utilizavam-na durante a execução dos programas para verificação dos resultados.

Utilizou-se o pacote MPICH [MPICH 01] como biblioteca de rotinas MPI para implementação dos programas de teste. Esta versão do MPI foi escolhida pois

possui uma bibliotecas M-VIA compatível com a distribuição, chamada de MVICH [MVICH 01].

O processo de instalação apresenta poucas dificuldades para administradores de sistemas UNIX habituados a instalar pacotes de distribuição de programas. Nos ANEXOS colocamos instruções de instalação para os principais pacotes.

4.3- Resultados

O programa MPI executado sobre o sistema de comunicação M-VIA foi mais rápido em todos os casos, como ilustra a figura a seguir (Fig 4.1):

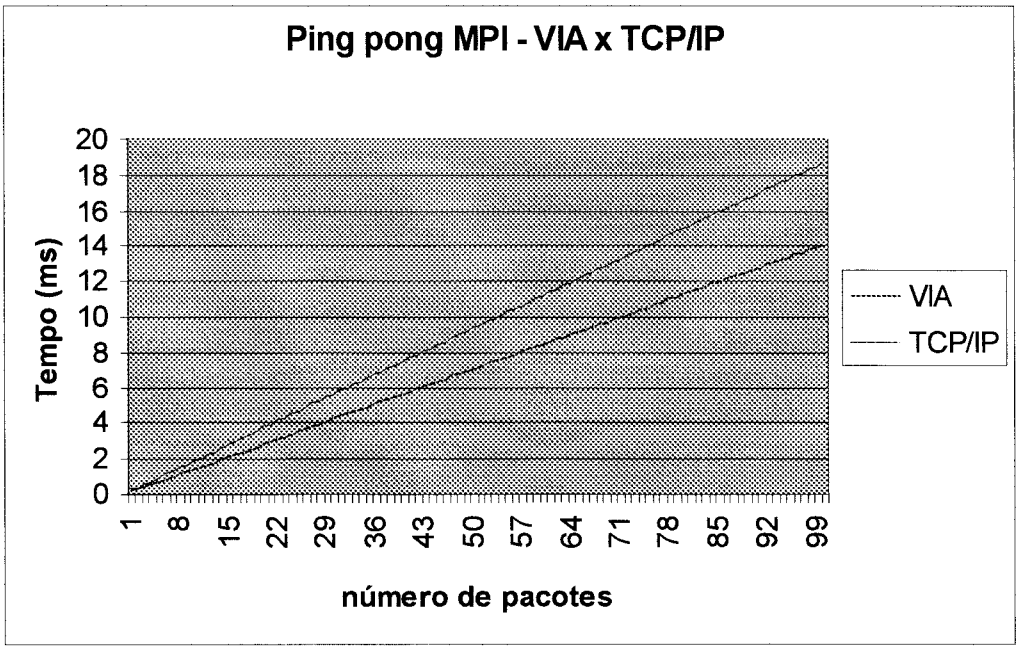

Fig 4.1: Comparação entre o tempo de comunicação de um programa MPI "ping-pong' executado sobre o TCP-IP e o M-VIA.

A figura 4-1 mostra a execução de um programa MPI que envia pacotes de 64 *bytes* para outro nó e aguarda o retorno destes pacotes para medir o tempo, de forma síncrona. Escolheu-se 64 *bytes* por ser um tamanho típico de

pacotes tipo *ping*. Foram feitas medidas de tempo para quantidades de pacotes variando de 1 até 100 pacotes. Para cada quantidade de pacotes o processo de envio e recebimento era repetido 5 vezes e uma média aritmética dos tempos era calculada, para uma medida mais realista do desempenho da comunicação.

O ganho obtido foi da ordem de 25% para este programa de testes.

Repetimos os testes para tamanhos de pacotes de 1 *byte*, para verificarmos o comportamento dos sistemas de comunicação com pacotes bem pequenos. O comportamento do sistema de comunicação pode ser visto na figura a seguir (Fig 4.2):

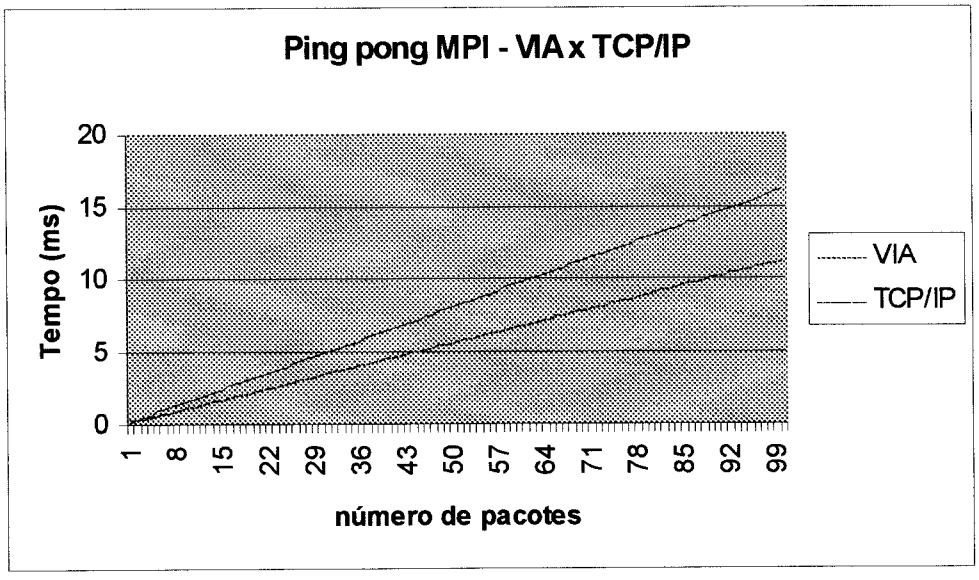

Fig 4.2: Comparação entre o tempo de comunicação de um programa MPI "ping-pong' executado sobre o TCP-IP e o M-VIA para pacotes de 1 byte.

Observa-se que neste caso a diferença de desempenho entre o TCP/IP e o M-VIA se torna ainda maior, demonstrando uma das características do M-VIA que é a otimização para trabalhar com pacotes pequenos.

Os resultados experimentais indicaram um ganho obtido da ordem de 30%.

Verificamos também o comportamento dos sistemas de comunicação com pacotes de 2500 bytes, que ultrapassa os 1500 *bytes* máximos de um *frame* e exige fragmentação da mensagem. Também neste caso o M-VIA teve melhor desempenho, porém o ganho foi da ordem de 14%, como o mostramos na figura a seguir (Fig 4.3):

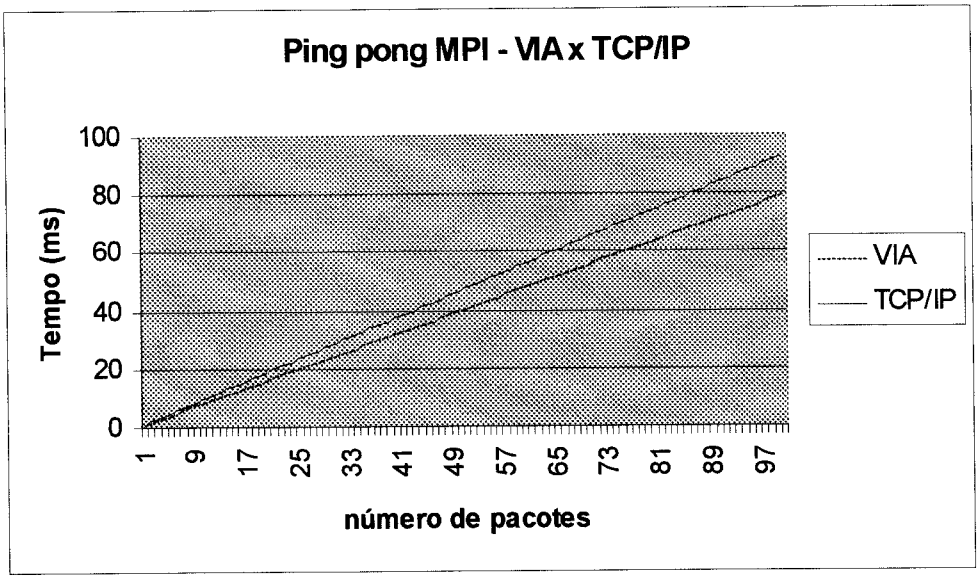

Fig 4.3: Comparação entre o tempo de comunicação de um programa MPI "ping-pong' executado sobre o TCP-IP e o M-VIA, para pacotes de 2500 *bytes*

Utilizamos também em nossos testes uma variação do programa *Mandelbrot*, disponível na distribuição do MPI-LAM [LAM 01], pois este programa representa um cálculo real executado em ambiente de processamento paralelo. A variação introduzida na nossa versão do programa foi unificar o código do programa *master.*c com o código do programa *slave.c.* Neste caso, definimos o *grid* de particionamento da imagem deste fractal como 30x30 pontos, sendo que a imagem final gerada possuía 512x512 pontos. Observamos (Fig. 4.4) que com poucos nós (2) temos um desempenho melhor do VIA, justamente porque há muitas comunicações uma vez que há poucos nós de processamento. Já quando há muitos nós (16) e o número de comunicações na rede diminui, o desempenho do TCP-IP e VIA se aproxima. Neste

caso não avaliamos o ganho, mas sim o comportamento do sistema de comunicação de acordo com o número de comunicações na rede.

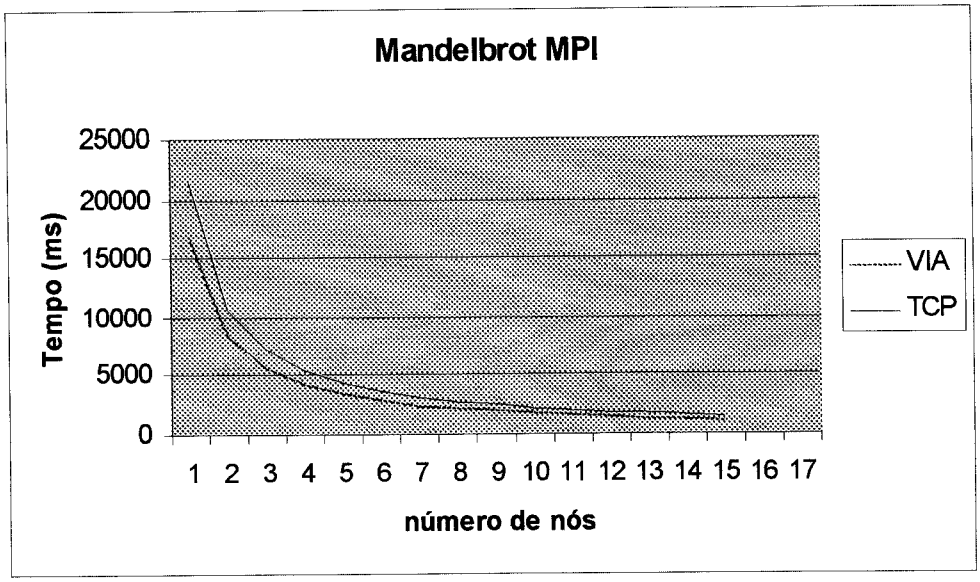

Fig 4.4: Programa *Mandelbrot* executado sobre VIA e TCP-IP *com grid de 30 pontos*

Capítulo 5

Conclusões

O gargalo de tempo para os sistemas de comunicação através de redes não está na implementação das camadas do protocolo, mas sim no sistema operacional, mais precisamente na maneira como o sistema operacional gerencia a memória do usuário e a memória do *kernel*. Por questões de proteção e organização do sistema operacional, os recursos compartilhados entre os usuários utilizam uma memória de acesso restrito, de uso do *kernel*. Há uma cópia de memória entre a área de memória do usuário e a área de memória do *kernel*, quando o usuário utiliza um destes recursos compartilhados. Desta forma, qualquer sistema de comunicação que queira se tornar mais eficiente deverá eliminar a cópia de memória que ocorre , como é o caso da comunicação por rede.

Sistemas de comunicação a nível de usuário que evitam esta cópia de memória já estão disponíveis para utilização em *clusters*. Estes sistemas apresentam eficiência maior que o tradicional protocolo *TCP/IP*, como pode-se verificar no Capítulo 4 deste trabalho. Assim, a montagem de clusters de processamento científico que utilize a linguagem de comunicação MPI pode aproveitar esta eficiência imediatamente. A opção por software de domínio público, como o LINUX, e plataforma de hardware comercial de custo moderado também favorecem a montagem destes sistemas no ambiente acadêmico. Observa-se porém o cuidado que deve-se ter para escolha da placa de rede, pois é necessário verificar se a distribuição do M-VIA possui os *drivers* controladores já implementados.

Os sistemas de comunicação de nível de usuário são interessantes para aplicações paralelas implementadas com a biblioteca MPI ou para grupos de pesquisa que optem por desenvolver a comunicação de suas aplicações com as primitivas de programação do M-VIA. Aplicações já desenvolvidas em MPI podem

ser portadas para MPI sobre M-VIA bastando recompilar os programas neste novo ambiente.

Porém, os sistemas de comunicação de nível de usuário ainda tem limitações quanto ao aspecto de segurança. A implementação dos *patches* para o *kernel* oferecem um nível de segurança mínimo para que as aplicações de um usuário não interfiram nas aplicações de outro usuário. Mas se considerarmos aspectos relativos a exploração de possíveis falhas de implementação para obtenção de acesso não autorizado, estes sistemas de comunicação podem ser melhorados.

Os sistemas de comunicação de nível de usuário também estão restritos a redes locais, pois não possuem nenhum mecanismo de controle para roteamento entre redes. Desta forma, o número de nós interligados pelo sistema estará restrito às limitações da tecnologia utilizada para interconexão.

Bibliografia

[BASKETT 93] Baskett, F. and Henessy. *Microprocessors: from desktops to supercomputers.* Science, Vol 261, 864-871.

[BHOEDJANG 98] Bhoedjang, R.A.F.; Rühl, Tim; Bal, Henri E.; "User-Level Network Interface Protocols" , in: IEEE November 1988

[COMER 95] Comer, D. E. *Internetworking With TCP/IP Vol. 1: Principles, Protocols, and Architecture Third Edition*, Prentice Hall, New Jersey, 1995

[CULLER 99] Culler, David E.; Mainwaring, Alan M. "Design Challenges of Virtual Networks: Fast, General-Purpose Communication" ACM, 1999.

[CULLER 99-2] Culler, D., et. al, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, ISBN 1558603433, 1999.

[EICKEN 98] Eicken, T, Vogels, W. *Evolution of the Virtual Interface Architecture, in*: IEE Computer November 1998, Vol 31, pp31-38.

[HILLIS 93] Hillis, W. D.; Tucker, L. W.*The CM-5 Connection Machine: a scalable Supercomputer.* Communications of ACM, November 1993, Vol.36, No11, Pags 31-40.

[LAM 01] http://www.lam-mpi.org, consultado em 15/02/2002.

[LINUX 01] Site encontrado em www.linux.org, Consultado em 14/01/2002.

[MARTIN 99] Martin, R., Culler, D.E. "*Effects of Communication Latency Overhead and Bandwidth Cluster*", ISCA97, 1997

[KAPLAN 94] Kaplan, J.A., Nelson, M.L. *A Comparison of Queueing, Cluster and Distributed Computing Systems,* June, 1994, disponível em : http://www.cmpharm.ucsf.edu/~srp/batch/tm109025.ps

[MPICH 01] http://www-unix.mcs.anl.gov/mpi/mpich/faq.html, consultado em 15/01/2002

[MVIA FAQ] http://www.nersc.gov/research/FTG/via/faq.html, consultado em 15/01/2002.

[MVICH 01] http://www.nersc.gov/research/FTG/mvich/, consultado em 15/01/20.02

[NATIONAL 95] *Writing Drivers for the DP8390 Family of Ethernet Controllers*, National Semiconductor Application Note 849, p1-310/1-325, 1995

[PFISTER 95] Pfister, G.F. "In Search of Clusters - The Coming Battle in Lowly Parallel Computing", Prentice Hall, New Jersey, 1995.

[RODRIGUES 96] Rodrigues, S. H. *Building a Better Byte Stream*, Master of Science Computer Thesis in Computer Science , University of California at Berkeley, 1996

[RUBINI 98] Rubini, A., *Linux Device Drivers*, O'Reilly & Associates, inc. 1998.

[SETI 01] http://setiathome.ssl.berkeley.edu/download.html, consultado em 15/01/2002

[SNIR 96] Snir, M.; Otto, S, Huss-Lederman; Walker, D; Dongara, J. *MPI: The Complete Reference.* MIT Press, Cambridge, Massachesetts, 1996.

[STALLINGS 00] Stallings, W. Computer *Organization and Architecture: Designing for Performance,* 5th Edition, Prentice-Hall, 2000.

[WELSH 97] Welsh, M; Basu, Anindya; Eicken, T., *ATM and Fast Ethernet Network Interfaces for User-level Communication*; Proceedings of the Third International Symposium on High Performance Computer Architecture, February, 1997.

Anexos

I- Manual de Instalação do M-VIA

Antes de se instalar o pacote do M-VIA, é necessário fazer algumas verificações:

- Você deve estar executando um kernel com versão superior a 2.2.X (X > 1).
- O controlador de da placa de rede que será usada com o M-VIA não deve estar incorporada ao código do kernel. Deve ser um módulo para ser inserido (*insmod*), pois o M-VIA faz a substituição deste módulo.
- Se você estiver utilizando o sistema em equipamento com um único processador, desligue a opção de SMP.

Observações: Siga todos os passos da instalação antes de tentar testar o sistema

* 1. Obtenha o arquivo mvia-X.XX.tar.gz do site
http://www.nersc.gov/research/FTG/via

* 2. Desempacote toda a distribuição do M-VIA

gunzip -c mvia-X.XX.tar.gz | tar xvf -
cd mvia-X.XX

* 3. Edite o arquivo Makefile.config

Este arquivo é auto-explicativo. Normalmente as únicas variáveis que você precisará ajustar são a versão do kernel e se o sistema será compilado para uma máquina SMP ou de processador único.

Obs.: Se a versão do kernel for, por exemplo, 2.2.14-5.0, coloque somente 2.2.14, ou o nome que estiver no diretório /lib/modules.

* 4. Faça a instalação dos executáveis e da biblioteca de nível de usuário:

make install

O padrão de instalação dos módulos os coloca em

/lib/modules/<número-da-versão-do-kernel>/net.

Os módulos para todos os dispositivos são instalados . Eles são:

- via_ka: o agente do kernel do M-VIA.
- via_lo: o dispositivo de loopback do M-VIA.
- via_ering: classe de dispositivos de anel ethernet do M-VIA, usados pelas implementações de Ethernet do M-VIA (3c59x, hamachi, yellowfin, eepro100 and

tulip)

- via_3c59x: dispositivo M-VIA 3c59x. Ele substitui o 3c59x do sistema.
- via_eepro100: dispositivo M-VIA eepro100
- via_hamachi: o dispositivo M-VIA PacketEngines GNIC-II
- via_tulip: dispositivo M-VIA tulip
- via_yellowfin: dispositivo M-VIA PacketEngines GNIC-I

A biblioteca VIPL é colocada no /usr/local/lib/libvipl.a e o arquivo de include VIPL em /usr/local/include/vipl.h.

* 5. Atualize as dependências de módulo do sistema

depmod -a <kernel-version-number>

Isto irá garantir que o carregador de módulos do kernel tem conhecimento das novas dependências criadas pela instalação do M-VIA entre os módulos.

* 6. Crie os arquivos no /dev para os dispositivos do M-VIA

O M-VIA automaticamente associa os dispositivos de rede com entradas no /dev. Basicamente são via_lo (via loopback), via_eth0, via_eth1, etc.:

make devices

* 7. Adicione a seguinte linha ao /etc/conf.modules para permitir o carregamento do módulo de loopback automaticamente;

alias char-major-60 via_lo

alias char-major-60 via_lo

Se você mudou a opção MVIA_CHAR_MAJOR no Makefile.config, troque o 60 na linha acima pelo número apropriado do dispositivo (*major* number).

* 8. Para cada interface de rede que você quer usar no M-VIA , coloque uma referência no etc/conf.modules no seguinte formato:

alias <dev> <módulo_via>

onde dev é o nome do dispositivo LINUX por exemplo *eth0*, e módulo_via é tanto via_eepro100 como via_hamachi, via_tulip ou via_yellowfin.

* 9. Remova as entradas /etc/conf.modules conflitantes.

Os dispositivos M-VIA são compatíveis com os módulos LINUX anteriores correspondentes, mantendo todas as suas funcionalidades, contudo, eles não

podem ser carregados simultaneamente. Remova ou tire o comentário de qualquer entrada no *conf.modules* que compartilham o mesmo sufixo de um módulo M-VIA por exemplo o M-VIA *via_tulip* and o *tulip* padrão do Linux não podem ser carregados simultaneamente.

* 10. Habilite os dispositivos de rede M-VIA nos seus arquivos de inicialização do sistema

Os métodos usados para configurar as interfaces de rede são específicos. Num sistema Red Hat crie as entradas ifcfg-devX apropriadas nos *scripts* /etc/sysconfig/network.

* 11. Copie o arquivo em *examples/etc/vip_hosts* para */etc/vip_hosts* e configure-o para sua rede.

Dependências dos módulos

Se você quiser carregar os módulos dos kernels manualmente, então ignore os passos de 7 a 9 . Se você quiser carregar os módulos usando insmod, você deve prestar atenção nas seguintes dependências

* Todos os módulos M-VIA necessitam do agente do kernel M-VIA, via_ka.
* Todos os drivers ethernet M-VIA necessitam o módulo da classe etherring, via_ering.

Assim, para usar apenas o dispositivo *loopback* M-VIA você executaria, a partir do diretório modules:

```
insmod via_ka.o
insmod via_lo.o
```

Para usar o driver eepro100 do M-VIA você você executaria, a partir do diretório modules:

```
insmod via_ka.o
insmod via_ering.o
insmod via_eepro100.o
```

Se você usa modprobe para carregar módulos, então estas dependências são manipuladas automaticamente.

II - Instruções específicas para instalação do MVICH

1) É necessário que o mvia esteja instalado e funcionando OK.
2) Ajusta-se a variável $MPICH_ROOT para o dir onde está a distribuiçao do MPICH
3) Deve-se ter o pacote do MPICH. Depois de abrí-lo, transfira o diretorio do MVICH para o diretório $MPICH_ROOT/mpid/via
4) Para funcionar no nosso cluster é necessário que a opção /dev/via_eth1 esteja dentro dos flags do **configure**
5) Siga as instruções de instalação do pacote.

III- Versões utilizadas

Pacote	*Versão*	*Origem*
MPICH	v1.2.1	http://www-unix.mcs.anl.gov/mpi/mpich
M-VIA	v1.0	http://www.nersc.gov/research/FTG/via
MVICH	v1.0	http://www.nersc.gov/research/FTG/mvich/
SLACKWARE LINUX	V8.0	ftp://ftp.slackware.com/pub/slackware/slackware-current/