Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Programa de Pós-Graduação em Ciências de
Computação e Matemática Computacional

**AVALIAÇÃO DE TECNOLOGIA
E REDE DE ALTO DESEMPENHO
PARA UTILIZAÇÃO NO SPP2**

Sílvia Tadao Fujisaki

Orientador: Prof. Dr. Onofre Trindade Jr.

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Área: Ciências de Computação e Matemática Computacional.

São Carlos
Janeiro de 1999

Aos meus pais
(em memória de meu pai Kyokazu)
e à minha família

Agradecimentos

Ao Prof. Onofre Trindade Jr. Pela amizade, orientação, incentivo, momentos alegres e pela oportunidade de aprender tantos assuntos novos, não somente os relacionados a este trabalho.

À minha família, pelo carinho, paciência e apoio em minhas empreitadas.

Aos amigos Gustavo, Augusto, Magnus, que tanto me ajudaram na concretização deste trabalho.

Ao Prof. Armando de Oliveira Fortuna, pela amizade e incentivo no desenvolvimento deste trabalho.

Aos amigos do LCAD, Intermedia (Taboca, Elder, Boni, Rudnei) e do ICMC pelos momentos de diversão e trabalho compartilhados.

Ao amigo Shiro (Antonio Marcus) e Milton pela amizade e grande incentivo no decorrer desse trabalho.

Aos amigos da república: Rogério, Mário Teixeira, Robson, Flávio, Shiro, Ronaldo, Teruo, Rogério Mouro, André e Mário, pela amizade, companheirismo, apoio e disposição para incentivar e colaborar.

A todas as pessoas que de alguma forma me incentivaram ou contribuíram com este trabalho.

A minha mãe, por sempre, sempre incentivar e apoiar.

A minha esposa, pela paciência, dedicação, apoio e incentivo que sempre tenho recebido.

Aos meus amigos da Unesp, em especial ao Prof. Neri e Messias, incansáveis no apoio e incentivo.

Aos meus Profs. da Unoeste, Álvaro e Ailton pela dedicação aos seus alunos.

Resumo

Este trabalho cobre as principais tecnologias de redes de comunicação de alto desempenho e suas aplicações. O enfoque principal é o estudo e a seleção de um padrão de rede de alta velocidade adequado para a implementação de sistemas computacionais paralelos de alto desempenho.

Os resultados do estudo feito foram aplicados no SPP2, uma arquitetura paralela baseada em computadores pessoais desenvolvida no LCAD-ICMC-USP. É proposta uma nova topologia de rede de alto desempenho para essa máquina baseada em componentes Myrinet. A nova rede de comunicações do SPP2 atende perfeitamente os requisitos iniciais do sistema. Os testes de desempenho realizados mostram um desempenho muito superior ao padrão de rede originalmente adotado no SPP2 e aos padrões de rede comumente utilizados em redes locais de estações de trabalho.

A arquitetura SPP2, com o novo sistema de comunicações, representa um grande passo na disseminação do processamento paralelo por apresentar características como baixo custo, alto desempenho, facilidade de construção e ampla disponibilidade de componentes para atualização do seu desempenho.

Abstract

This work presents some technologies and applications of high performance computer networks. The main goal is the study and selection of a computer network standard for the implementation of high performance parallel computer systems.

Results from the study phase were applied to the SPP2, a parallel architecture based on personal computers developed at the LCAD-ICMC-USP. A new topology of a high performance communication network is proposed for the SPP2, based on Myrinet components. The new communication network complies with the original requirements for the SPP2. Performance evaluation tests were conducted and show a much higher performance than the original SPP2 communication network and the network standards commonly used in workstation networks.

The SPP2 architecture and its new communication network are a big step towards the dissemination of parallel processing, presenting low price, high performance, easy implementation and large availability of components, allowing for easy updating of performance.

1. INTRODUÇÃO	1
1.1 Aspectos Gerais	1
1.2 Estrutura do Texto	2
2. Redes de Comunicação de Alta Velocidade	3
2.1 Considerações Iniciais	3
2.2 Redes de Computadores	3
2.2.1 Aplicações de Redes	5
2.3 Processamento Paralelo	6
2.4 Arquiteturas Paralelas MIMD com Memória Distribuída	7
2.5 O SPP2 - um Multicomputador de Alto Desempenho	10
2.6 ATOMIC: Rede Local de Alto Desempenho	12
2.6.1 Introdução	12
2.6.2 Arquitetura	12
2.6.3 O Hardware da Rede	13
2.6.4 O Software da Rede	15
2.6.5 Address Consultant	16
2.6.6 O Desempenho	17
2.7 Myrinet	17
2.7.1 Introdução	17
2.7.2 Principais Características	18
2.7.3 O Hardware da rede	20
2.7.4 Interface de Software e Desempenho	21
2.8 ATM	23
2.8.1 Introdução	23
2.8.2 Características da Rede ATM	23
2.8.3 A Camada Física	24
2.8.4 A Camada ATM	25
2.8.5 A Camada de Adaptação ATM	26
2.9 SCI – Scalable Coherent Interface	29
2.9.1 Introdução	29
2.9.2 Interface da Camada Física	30
2.9.3 Arquitetura CSR	31
2.9.4 Protocolos de Transferência	32
2.9.5 Tipos de Pacotes	32
2.9.6 Mecanismos de Coerência	33
2.9.7 O Protocolo de Coerência de Cache SCI	33
2.10 HIPPI - High Performance Parallel Interface	35
2.10.1 Introdução	35
2.10.2 HIPPI - Camada Física	36
2.10.3 O Sinal de Interface HIPPI	36
2.10.4 Camada de Enlace de Dados	37
2.10.5 Quadro de Dados	37
2.10.6 Outros Padrões HIPPI	38
2.11 Comparação dos Padrões de Rede Apresentados	39
2.12 Considerações finais	40
3. ESTUDO DETALHADO DA REDE MYRINET	42
3.1 Considerações Iniciais	42

3.2	O Hardware do Padrão Myrinet	42
3.2.1	A Interface do Host	42
3.2.2	Switches Myrinet	43
3.2.3	Cabos Myrinet	44
3.2.4	Acessórios Myrinet	46
3.3	O Software do Padrão Myrinet	47
3.3.1	O MCP – Myrinet Control Program	47
3.3.2	A API Myrinet	48
3.3.3	A BPI Myrinet	49
3.3.4	O Driver TCP/IP	49
3.3.5	O Fast Messages	50
3.3.6	Encapsulamento Pacotes Ethernet	51
3.3.7	O Formato do Pacote Myrinet	51
3.3.8	Myrinet, novo padrão ANSI	53
3.4	Sistemas existentes que utilizam a Myrinet	53
3.5	Considerações Finais	55
4.	A Rede de Comunicação de Alto Desempenho do SPP2	56
4.1	Considerações Iniciais	56
4.2	Detalhes da Organização e Implementação do SPP2	56
4.3	A Topologia da Rede de Alto Desempenho do SPP2	58
4.3.1	Interconexão dos Nós	58
4.3.2	Topologia mínima para o SPP2	60
4.3.3	Topologia com 32 nós para o SPP2	60
4.3.4	Topologia BBW Completa	61
4.3.5	Topologia 64 nós para o SPP2	62
4.3.6	Topologia BBW completa com 128 nós para SPP2	63
4.3.7	BBW Parcial com 256 nós para o SPP2	63
4.3.8	Relação custo/capacidade agregada da rede	65
4.4	Considerações Finais	66
5.	ANÁLISE DE DESEMPENHO DA REDE MYRINET	67
5.1	Considerações Iniciais	67
5.2	Descrição do Ambiente de testes	67
5.2.1	O Sistema Operacional	69
5.2.2	Padrões de Redes de Computadores Utilizados	69
5.2.3	Software de Comunicação	71
5.3	Resultados Obtidos	74
5.4	Análise dos Resultados	75
5.5	Considerações Finais	78
6.	CONCLUSÕES	80
6.1	Considerações Iniciais	80
6.2	Contribuições deste Trabalho	80
6.3	Sugestões para Trabalhos Futuros	81
	Referências Bibliográficas	82
	Apêndice A	86

Índice de Figuras

Figura 2.1	Topologias De Redes De Computadores mais difundidas	5
Figura 2.2	Implementação Física de Multiprocessadores	8
Figura 2.3	Implementação física Multicomputadores	9
Figura 2.4	Topologias de redes utilizadas em Multicomputadores	10
Figura 2.5	A arquitetura do Spp2	11
Figura 2.6	Chip do processador Mosaic	13
Figura 2.7	Placa de Interface de Host Sbus	14
Figura 2.8	A Configuração de uma rede local Atomic	14
Figura 2.9	O Encapsulamento de pacotes	15
Figura 2.10	Roteamento na rede Atomic	16
Figura 2.11	A rede Myrinet	18
Figura 2.12	Malha de rede Myrinet de duas dimensões	19
Figura 2.13	Switch Myrinet LAN / SAN	21
Figura 2.14	Sistema de computação heterogênea	22
Figura 2.15	Formato de uma Célula ATM	23
Figura 2.16	Modelo de camadas ATM	24
Figura 2.17	Frame definido para o nível STM-1	25
Figura 2.18	Formato dos cabeçalhos das células ATM	25
Figura 2.19	Operação esquemática da camada de adaptação ao ATM	27
Figura 2.20	Classes de serviços e protocolos na camada de adaptação ao ATM	28
Figura 2.21	Formato AAL5 do pacote apresentado à subcamada de segmentação e recomposição	28
Figura 2.22	Nó SCI sobre uma topologia em anel	30
Figura 2.23	Formato dos dados S20	30
Figura 2.24	Mapa de endereçamento do SCI	31
Figura 2.25	Formato típico de pacote envio SCI	32
Figura 2.26	Comutador <i>crossbar</i> HIPPI	36
Figura 2.27	Organização da informação do padrão HIPPI	38
Figura 2.28	As camadas do protocolo do Fibre Channel	39
Figura 3.1	Diagrama de uma Interface Myrinet	43
Figura 3.2	Cabo LAN Myrinet	45
Figura 3.3	Cabo SAN Myrinet	45
Figura 3.4	Acessórios Myrinet	46
Figura 3.5	Diagrama de cópia dos dados entre as diferentes interfaces de <i>software</i>	48
Figura 3.6	Formato de um pacote Myrinet	52
Figura 4.1	A arquitetura do SPP2	58
Figura 4.2	Conexão simples entre dois computadores com a rede Myrinet	59
Figura 4.3	Esquema da <i>switch</i> M2M-DUAL-SW8 e da M2M-OCT-SW8	59
Figura 4.4	Módulo do SPP2 com 8 nós	60
Figura 4.5	Topologia para 24 nós ou 32 nós	61
Figura 4.6	Topologia com 64 nós para o SPP2	62
Figura 4.7	Topologia BBW completa com 128 nós processadores para o SPP2	63
Figura 4.8	Topologia BBW parcial com 256 nós processadores para o SPP2	64
Figura 5.1	Bancada de testes com dois computadores	67
Figura 5.2	Conexão de dois computadores pelo padrão Ethernet de 10 Mbits/s	70
Figura 5.3	Conexão de dois computadores pelo padrão Fast Ethernet de 100 Mbits/s	70
Figura 5.4	Conexão de dois computadores pelo padrão Myrinet de 2.56 Gbits/s	71
Figura 5.5	Latência em Microsegundos	75
Figura 5.6	Latência em Microsegundos	76
Figura 5.7	Taxa de transferência em Mbytes/s	77

Índice de Tabelas

Tabela 2.1	Tipo de pacotes do padrão SCI	33
Tabela 2.2	Conjunto típico de estados coerentes de memória estável	34
Tabela 2.3	Conjunto típico de estados coerentes de cache estável	35
Tabela 2.4	Principais características dos padrões de redes descritos	39
Tabela 3.1	Relação dos Processadores e Sistemas Operacionais compatíveis com a rede Myrinet	44
Tabela 4.1	Custo de desempenho para BBW completa ou parcial	65
Tabela 5.1	Resultado da taxa de transferência em Mbytes/segundo	74
Tabela 5.2	Resultado da latência em microssegundos	75

1. INTRODUÇÃO

1.1 Aspectos Gerais

Apesar de a taxa de aumento do poder computacional verificada nos últimos anos ter se mantido constante, grande parte desse aumento é devido à utilização de técnicas de processamento paralelo internamente aos processadores. O uso dessas técnicas, aliado aos avanços na tecnologia de implementação de circuitos integrados, permite concluir que ainda não existe nenhum fator determinante para que esse aumento deixe de existir nos próximos anos. Por outro lado, observa-se que técnicas de processamento paralelo têm sido utilizadas em nível de sistema. Máquinas explorando arquiteturas paralelas MIMD com memória compartilhada têm se tornado cada vez mais comuns, utilizando de 2 a 32 processadores.

Redes de comunicação de alta velocidade têm se tornado disponíveis a baixo custo, permitindo que sejam utilizadas para a implementação de máquinas MIMD com memória distribuída, de uma maneira simples e direta. Essa classe de máquinas é representada pelo SPP2- uma arquitetura paralela MIMD com memória distribuída, de baixo custo, baseada em componentes de microcomputadores pessoais que sofrem avanços tecnológicos significativos em curto espaço de tempo, possibilitando, assim, atualizações contínuas no desempenho do SPP2 [TRI94].

O desempenho global de uma máquina MIMD de memória distribuída depende principalmente de dois fatores:

- desempenho dos processadores utilizados;
- desempenho da rede de comunicação utilizada.

Os nós de processamento do SPP2 são constituídos por placas com processadores Pentium da Intel, embora processadores de outros fabricantes possam ser utilizados podendo cada um deles conter um ou mais processadores, dependendo do tipo de placa utilizada.

Quanto à rede de comunicação, existem atualmente no mercado vários padrões de redes de alta velocidade com variados níveis de adequação para uso no SPP2.

O objetivo principal deste trabalho é o estudo desses padrões e sua adequação ao projeto do SPP2. Também faz parte do escopo deste trabalho a definição da topologia de rede do SPP2 e testes de desempenho preliminares do padrão de rede escolhido.

1.2 Estrutura do Texto

O presente texto é composto de 6 capítulos. O capítulo 2 fornece uma revisão da bibliografia sobre os diversos tópicos envolvidos em seu desenvolvimento, incluindo redes de computadores, processamento paralelo, arquiteturas MIMD com memória distribuída e o SPP2. A maior parte do capítulo é reservada para à descrição das características principais dos padrões de rede de alta velocidade mais atuais e no final, é apresentada uma tabela onde são resumidas essas características e a justificativa para adoção do padrão de rede Myrinet no SPP2. O capítulo 3 fornece informações mais detalhadas da rede de comunicação Myrinet, envolvendo o *hardware* e o *software* dessa rede, bem como a descrição de alguns sistemas que utilizam o padrão Myrinet. O capítulo 4 fornece detalhes da organização e implementação da arquitetura do SPP2, juntamente com uma proposta de topologia para uma rede com 256 nós de processamento. O capítulo 5 trata da descrição do ambiente e testes de desempenho efetuados, apresentando os resultados obtidos e a análise desses resultados. No capítulo 6 são apresentadas as conclusões finais e propostas de novos projetos relacionados com a continuidade do trabalho realizado. Finalizando, o apêndice A apresenta os códigos dos programas utilizados nos testes efetuados.

2. Redes de Comunicação de Alta Velocidade

2.1 Considerações Iniciais

Atualmente, muitos microcomputadores exibem desempenho maior do que os supercomputadores do início da década de 1980, máquinas que normalmente eram conectadas por redes HIPPI (*High Performance Parallel Interface*) de 800 Mbits/s [WHY98]. Os padrões de rede de alta velocidade também têm evoluído, não somente quanto ao desempenho, mas também quanto à queda em seus custos, tornando-os mais acessíveis. Isso permitiu sua utilização em muitas aplicações de computação distribuída, como é o caso da computação de alto desempenho utilizando-se máquinas MIMD com memória distribuída.

Neste capítulo é feita uma revisão bibliográfica dos temas principais deste trabalho. Na seção 2.2 são apresentados alguns conceitos básicos sobre redes de comunicação e suas aplicações. Conceitos sobre processamento paralelo são discutidos na seção 2.3, enquanto a seção 2.4 é reservada para uma discussão sobre as arquiteturas paralelas MIMD com memória distribuída, exemplificadas com o SPP2 na seção 2.5. As seções seguintes são reservadas para cada um dos padrões de rede de alta velocidade em estudo, mais especificamente o Atomic na seção 2.6, o Myrinet na seção 2.7, o ATM na seção 2.8, o SCI na seção 2.9 e o HIPPI na seção 2.10. Na seção 2.11 é apresentada uma tabela comparativa das principais características dos padrões de rede apresentados e a definição do padrão de rede que será utilizado no SPP2.

2.2 Redes de Computadores

Uma rede de computadores pode ser definida como uma coleção de computadores autônomos interligados por uma rede de comunicação de dados. Assim como na rede telefônica, cada computador está associado a um número ou endereço. Para que a troca de informações entre os vários componentes de uma rede de computadores se dê de forma ordenada, é estabelecido um conjunto de protocolos que definem as regras a serem usadas na comunicação.

Um dos objetivos de uma rede de computadores é efetuar o compartilhamento de recursos, fazendo com que programas, dados e equipamentos estejam disponíveis para qualquer usuário na rede, não importando a localização geográfica desses recursos ou dos usuários. Ainda mais importante que o compartilhamento de recursos é o compartilhamento de dados. Outro objetivo da rede é garantir uma alta confiabilidade do sistema oferecendo fontes alternativas de recursos computacionais, como replicação de arquivos em duas ou mais máquinas ou a presença de CPUs (Central Processing Unit - Unidade Processamento Central) múltiplas, de tal forma que, se um recurso falhar, outro assumirá o trabalho, ainda que com o ônus de uma redução no desempenho global do sistema.

O modelo de referência ISO/OSI (*International Standards Organization /Open Systems Interconnection*) [DAY83] especifica uma arquitetura padrão de redes de computadores em sete camadas que se comunicam através de serviços que uma camada disponibiliza para a camada superior [TAN89]. A camada inferior da arquitetura ISO/OSI é a **Camada Física**, que especifica o meio físico utilizado para a transmissão dos dados, seguida pela **Camada de Enlace de Dados**, que especifica o protocolo para a troca de dados ponto a ponto na rede. A **Camada de Rede** tem como principal objetivo o roteamento de mensagens na rede e a **Camada de Transporte** cuida da compatibilização entre as mensagens recebidas/enviadas entre as camadas de rede e as camadas de nível superior. A **Camada de Seção**, além de permitir também o transporte de dados, cuida do controle de tráfego, gerenciamento de *token* e sincronização na transferência de grandes arquivos. A **Camada de Apresentação** trata principalmente da sintaxe e semântica das informações transmitidas permitindo, por exemplo, que diferentes representações de cadeias de caracteres entre diferentes computadores possam ser trocadas. Finalmente, a **Camada de Aplicação** é detentora dos protocolos que cuidarão de incompatibilidades como, por exemplo, de diferentes terminais e diferentes meios de representação de dados. A utilização de uma estrutura de camadas é uma prática comum na definição de uma arquitetura de redes. O conjunto de protocolos TCP/IP (*Transmission Control Protocol-Internet Protocol*) já utilizava essa técnica antes da definição do modelo ISO/OSI [CER74]. A disseminação do uso e quase hegemonia das redes TCP/IP tomou o modelo ISO/OSI pouco utilizado.

Existem vários meios físicos para transmissão de dados, destacando-se o cabo coaxial, o par trançado, canais de rádio/satélite e o cabo de fibra ótica. Cada um desses meios apresenta diferentes características para a transmissão de sinais, sendo as mais importantes a largura de

banda (*bandwidth*), a latência, a imunidade a ruídos e a distância geográfica coberta. A largura de banda está relacionada com a velocidade de transmissão dos dados. A latência é o tempo para que o primeiro dado chegue ao destino. Os cabos de fibra ótica têm sido largamente utilizados, pois apresentam bom desempenho em cada um dos quatro aspectos mencionados acima.

É comum classificar as redes de computadores quanto ao tamanho da sua cobertura geográfica: LAN (*Local Area Network* – Rede Local), MAN (*Metropolitan Area Network* – Rede Metropolitana) e WAN (*Wide Area Network* – Rede Geograficamente Distribuída).

As topologias de rede mais comuns são ilustradas na figura 2.1.

Figura 2.1 - Topologias de redes de computadores mais difundidas

2.2.1 Aplicações de Redes

A substituição de um único *mainframe* por estações de trabalho em uma rede local, entretanto, não torna possíveis muitas aplicações novas, embora possa melhorar a confiabilidade e o desempenho do sistema. Ao contrário, a disponibilidade de uma WAN (pública) torna possível uma grande quantidade de aplicações novas, algumas das quais podem ter efeitos importantes na sociedade como um todo. Descrevem-se, a seguir, alguns exemplos práticos importantes do uso de redes de computadores.

1. Usuários podem se comunicar através de correio eletrônico com outras pessoas localizadas em qualquer lugar do planeta, bem como efetuar o acesso remoto a bancos de dados para fazer reservas de aviões, trens, ônibus, barcos, hotéis, restaurantes e teatros com confirmação instantânea. O banco em casa (*home banking*) e o jornal automatizado também

entram nessa categoria. Diferentes dos jornais atuais, os jornais eletrônicos podem ser facilmente personalizados, por exemplo, tratando sobre computadores, as reportagens principais sobre política e epidemias, etc;

2. A *Internet* tem permitido o acesso aos bancos de dados de vários locais (universitários e comerciais), contribuindo, assim, para a globalização mundial, derrubando fronteiras culturais e geográficas;

3. As redes de computadores continuam aumentando suas fronteiras de aplicação, como no caso de aplicações de Sistemas Multimídia Distribuídos, que integram textos, sons, imagens, gráficos e vídeo em uma única aplicação. Isso se tornou possível graças aos avanços tecnológicos, a redução do custo dos componentes de computadores e o surgimento das arquiteturas de redes de alta velocidade [NAH95];

4. *Video-on-demand* - Serviços de vídeo interativos foram projetados para suportar muitos canais simultâneos sobre um mesmo banco de dados da programação, permitindo ao usuário executar qualquer parte da programação a qualquer hora[LIE95];

5. Redes de alto desempenho possibilitam a utilização de técnicas de processamento paralelo para a execução de aplicações que exijam elevado desempenho computacional para que possam ser executadas em espaços de tempo que permitam a utilização efetiva dos resultados obtidos, como é o caso da previsão do tempo.

2.3 Processamento Paralelo

Segundo a definição de Hwang [HWA87], o processamento paralelo é uma forma eficiente de processamento da informação com ênfase na exploração de eventos concorrentes no processo computacional. Essa idéia não é nova. Em 1920, Vannevar Bush, do MIT, apresentou um computador analógico capaz de resolver equações diferenciais em paralelo; o próprio von Neumann, em seus artigos, por volta de 1940, sugere uma grade para resolver equações diferenciais em que os pontos são atualizados em paralelo. A razão principal para o surgimento do processamento paralelo foi a capacidade de aumentar o processamento de uma única

máquina. O marco inicial do surgimento de máquinas para processamento paralelo foi o ILLIAC IV construído em fins dos anos 60 na Universidade de Illinois, composto por 64 processadores.

O processamento paralelo existe a partir do momento em que dois ou mais processadores interagem entre si para resolver uma determinada tarefa. Executá-lo com eficiência, não é uma tarefa simples e, nem sempre, possível. Enquanto alguns problemas podem ser paralelizados ao extremo, isto é, podem ser divididos em quantas tarefas concorrentes se desejar, outros são inerentemente sequenciais e não admitem execução em paralelo com eficiência. Os algoritmos utilizados para processamento paralelo também podem diferir bastante dos algoritmos sequenciais “ótimos”. A paralelização de um programa também depende da arquitetura em que ele será executado. Ao contrário das arquiteturas sequenciais, que seguem um modelo computacional único e bem definido, as paralelas são variadas e, para cada uma delas, existe uma forma adequada à solução de um determinado problema computacional [GEI94].

O processamento paralelo envolve ainda as linguagens paralelas, os compiladores paralelos, os compiladores vetorizadores, os compiladores otimizadores e os sistemas operacionais para máquinas paralelas [NAV89]; além desses itens, está em desenvolvimento no LCAD-ICMC-USP, o PADE - Um Ambiente de Desenvolvimento para Aplicações Paralelas [TRI98].

2.4 Arquiteturas Paralelas MIMD com Memória Distribuída

Em 1966, Flynn propôs uma classificação para as arquiteturas de computadores em quatro categorias, dependendo da multiplicidade do fluxo de instruções e do fluxo de dados [FLY82], bastante aceita pela comunidade científica devido à sua simplicidade. As quatro categorias propostas por Flynn são:

- SISD (*Single Instruction, Single Data*) - fluxos de instruções e de dados únicos – os computadores sequenciais correspondem a essa categoria;
- SIMD (*Single Instruction, Multiple Data*) - fluxo de instrução único e múltiplos fluxos de dados - corresponde aos processadores matriciais;

- MISD (*Multiple Instruction, Single Data*) - múltiplos fluxos de instruções para um único fluxo de dados – alguns autores afirmam que não existem máquinas que atendam a essa classificação, enquanto outros incluem nessa categoria os processadores vetoriais;

- MIMD (*Multiple Instruction, Multiple Data*) - múltiplos fluxos de instruções e múltiplos fluxos de dados, em que cada unidade de processamento possui sua unidade de controle, executando instruções sobre um conjunto de dados próprio. São pertencentes a essa classificação os multiprocessadores e os multicomputadores.

Computadores MIMD são compostos por elementos de processamento assíncronos, caracterizados pelo controle de *hardware* descentralizado. Essas arquiteturas são adequadas para a exploração de paralelismo de granularidade média e alta ao nível de tarefas e subprogramas, podendo ser explorado pela técnica de “divisão e conquista” [ALM94, DUN90].

Nos multiprocessadores, ou arquiteturas MIMD com memória compartilhada, um conjunto de processadores compartilha uma memória única através de um barramento ou uma rede de interconexão qualquer, como ilustra a figura 2.2.

Figura 2.2 - Implementação física de Multiprocessadores

Nos multicomputadores ou arquiteturas MIMD com memória distribuída, um conjunto de nós de processamento, máquinas de von Neumann, é conectado por uma rede de comunicação de dados, como ilustra a figura 2.3. Cada nó é composto de CPU e memória. A comunicação e o sincronismo entre os nós são efetuados pela troca de mensagens entre eles [FOS95, NAV89].

Figura 2.3 - Implementação física Multicomputadores

Essas arquiteturas são mais adequadas para o processamento paralelo de propósito geral, pois oferecem baixo custo, facilidade de implementação e elevada escalabilidade [HIL91]. Várias topologias de interconexão de rede têm sido propostas para oferecer expansibilidade arquitetural e desempenho eficiente para programas paralelos. Os principais aspectos que definem as características de uma topologia são o diâmetro de comunicação e o número de canais necessários em relação ao número de processadores. O diâmetro de comunicação pode ser definido como a maior distância (número de saltos através de nós intermediários) entre dois nós quaisquer da rede. Por exemplo, uma malha totalmente interconectada, isto é, na qual cada nó tem canais de comunicação com todos os demais, apresenta diâmetro de comunicação 1 e $(n(n-1))/2$ canais, em que n é o número de nós. A figura 2.4 ilustra algumas dessas topologias:

1 - Topologia em anel: uma conexão ponto a ponto permite a comunicação entre dois computadores em uma rede. Conectando-se o último processador ao primeiro, obtém-se uma topologia em anel. Topologias em anel não são muito apropriadas quando a comunicação entre as tarefas em execução em cada nó é elevada;

2 - Topologia em malha: uma malha de duas dimensões, possuindo n^2 nós, em que cada nó é conectado a dois ou quatro nós vizinhos. Conexões nas margens da malha podem ser feitas para reduzir o diâmetro da comunicação de $2(n-1)$ para $2*[n/2]$, $[n/2]$ significando a parte inteira de $n/2$. Pode-se aumentar a capacidade de comunicação adicionando-se *links* diagonais ou usando-se barramentos para conectar nós através de linhas ou colunas;

3 - Arquitetura de topologia em árvore: esse tipo de topologia é bastante adequado para a implementação de algoritmos de pesquisa e classificação, algoritmos de processamento de

imagens e redução. Uma variedade de topologias em árvore tem sido sugerida e árvores binárias completas são as variantes mais analisadas;

Figura 2.4 - Topologias de redes utilizadas em Multicomputadores

4 - Arquitetura de topologia hipercubo: a topologia hipercubo utiliza $N = 2^n$ arranjos de processadores em um cubo n -dimensional. Nós individuais são identificados unicamente por valores numéricos de n -bits que estão numa faixa de 0 até $N-1$. O diâmetro de comunicação de tal topologia é de $n = \log_2 N$.

2.5 O SPP2 - um Multicomputador de Alto Desempenho

A grande motivação para o desenvolvimento do SPP2 é a disponibilidade de tecnologia de baixo custo e alto desempenho. O SPP2 consiste em uma arquitetura paralela MIMD com memória distribuída baseada em partes componentes de computadores pessoais. Essa arquitetura segue a tendência da maioria das arquiteturas MIMD com memória distribuída, disponíveis no mercado, ou seja, a utilização de microcomputadores interligados por uma rede de comunicação de alto desempenho. A figura 2.5 apresenta uma visão geral da arquitetura do SPP2.

Figura 2.5 - A arquitetura do SPP2

A versão inicial do SPP2 suporta até 256 nós processadores (PN00.00 a PN15.15) distribuídos em módulos de 16 processadores cada um. Uma máquina com um único módulo pode conter até 32 processadores que são interligados por duas redes de comunicação: uma rede Ethernet, utilizada para o controle e gerenciamento do sistema e uma rede de comunicação de alto desempenho. Os nós do SPP2 dispõem de processadores da linha Intel e executam o sistema operacional Linux [TRI94]. Os processadores Intel têm apresentado um desempenho compatível com os melhores processadores do mercado, conforme informações apresentadas em [CPU98].

O sistema operacional Linux [TRI94] tem se mostrado bastante confiável e impõe uma carga computacional leve aos processadores em relação a outros sistemas operacionais disponíveis, tais como o Windows NT e outras versões do UNIX. Além disso, um grande número de programas para computação paralela está disponível para o Linux, muitos dos quais de domínio público, como por exemplo, Fortran, HPF, C, C++, P2d, MPI++, HPVM, RCP, etc [PER96].

Inicialmente, conforme mostrado na figura 2.5, a rede de comunicação de alto desempenho do SPP2 seria totalmente desenvolvida no LCAD utilizando-se a tecnologia de comunicação da INMOS, mais especificamente a tecnologia que foi desenvolvida para os processadores Transputer da série T9000 [TRI94]. O projeto inicial previa uma velocidade de

comunicação ponto a ponto de 800 Mbits/s com latência inferior a 2 μ s. A dificuldade de obtenção dos componentes necessários para a implementação das interfaces de comunicação não permitiu que o projeto fosse implementado e outras alternativas estão sendo avaliadas, conforme descrição nos itens seguintes.

2.6 ATOMIC: Rede Local de Alto Desempenho

2.6.1 Introdução

A rede Atomic é um exemplo de rede local de altíssima velocidade, baixo custo e capacidade de transmitir dados de vídeo digital, multimídia e aplicações distribuídas com grandes taxas de transmissão de dados.

A rede Atomic emprega técnicas de computação paralela atingindo desempenho diversas ordens de magnitude maior que o apresentado por redes locais convencionais e até o princípio de 1994, era única nessa classe. Seu protótipo está operacional na USC/ISI (*University of Southern California /Information Sciences Institute*) desde outubro de 1991, estando sua tecnologia baseada no projeto Mosaic, um computador paralelo de granularidade fina que se comunica através de uma rede de troca de mensagens [FLO94].

2.6.2 Arquitetura

O Mosaic é constituído por uma malha de duas dimensões de 128 x 128 nós exigindo um grande desempenho da rede de passagem de mensagens com mensagens de tamanho variável. Cada uma delas contém um *byte* delta-X seguido por um *byte* delta-Y (indicando respectivamente a direção leste-oeste e norte-sul). Esses *bytes* podem assumir valores entre -127 a +127 passos de roteamento, subtraídos ao passar em cada nó até alcançar zero, ocorrendo todo o roteamento na direção X, antes daquele na direção Y, evitando-se *deadlocks* na rede. O *chip* Mosaic integra as placas de interface com o *host* e as chaves do roteador.

Cada *chip* do Mosaic contém um processador de propósito geral com memória ROM e RAM, um roteador em *hardware* com topologia de roteamento de duas dimensões e uma

interface de DMA (*Direct Memory Access* – Acesso Direto à Memória) entre o roteador e a RAM. Existem oito canais de comunicação *simplex* com uma taxa nominal de transferência de 500 Mbits/s cada um. Uma ligação *full-duplex* ponto a ponto é constituída por um par desses canais e oferece uma capacidade de transferência de dados bidirecional de 1 Gbit/s. O diagrama de blocos do *chip* Mosaic pode ser visto na figura 2.6.

Figura 2.6 - *Chip* do processador Mosaic

2.6.3 O Hardware da Rede

Uma interface de *host* Atomic é composta de um *chip* com 4 memórias de 128 K x 4 bits, o *chip* Mosaic, um circuito de *clock* e um *chip* lógico de interface com o barramento. Existem placas de interface Atomic para o barramento VME que usam memória de 30Mhz (16 bits), fornecendo, assim, até 480 Mbits/s, e para o barramento Sbus que utilizam memória de 20 Mhz para oferecer 320 Mbits/s.

Os canais Mosaic operam a 500 Mbits/s e são auto-temporizados, adaptando-se a canais com velocidades mais lentas. Cada placa contém várias interfaces de *host* Atomic completas (VME=4, Sbus=2) e quatro canais externos, dois de entrada e dois de saída. Cada *host* requer apenas uma interface de *host* Atomic, embora duas sejam preferíveis, já que uma pode ser dedicada para a entrada e a outra para a saída da rede, dobrando o potencial de tráfego e oferecendo processamento paralelo e administração do tráfego na rede. Para distâncias até 60,96 centímetros entre *chips* Mosaic, a interconexão de interfaces Atomic é simples. Acima disso, alguma lógica adicional é necessária para manter o desempenho do canal. A placa de interface de *host* é mostrada na figura 2.7.

Figura 2.7 - Placa de interface de *host Sbus*

Os *chips* Mosaic são interligados para constituir uma malha de duas dimensões. Em uma rede local Atomic, as malhas funcionam como comutadores *crossbar*, podendo uma placa típica conter 64 *chips* Mosaic organizados em uma matriz 8x8. A figura 2.8 ilustra uma configuração de rede local com malhas 4x4. Uma malha de $n \times n$ possui $4n$ pares de canais Mosaic *full-duplex*, podendo cada par de canais ser usado para conexão de um ou uma cadeia de *hosts* nas malhas ou para interligar outras malhas [FEL94].

Figura 2.8 - A configuração de uma rede local Atomic

2.6.4 O Software da Rede

O protótipo da rede Atomic foi implementado em estações de trabalho Sun-3 e Sun-4 sob o sistema operacional BSD UNIX (SunOS), utilizando o mecanismo padrão de *sockets* BSD, o que permite utilizar o protocolo IP (*Internet Protocol*) e todos aqueles de alto nível do conjunto TCP/IP, oferecendo todas as funcionalidades usuais, inclusive autoconfiguração, interrupção de ligação, resolução de endereço, controles de I/O, etc.

Na rede local Atomic, utiliza-se a família de endereços AF_ATOM, análoga à AF_INET, usada para abrir um *socket* BSD. Em cada transmissão, o programa do usuário deve fornecer o cabeçalho da camada de ligação no início do pacote que quando processado pelo núcleo, o pacote fornecido pelo usuário é encapsulado em um cabeçalho UDP (*User Datagram Protocol*) e em um cabeçalho IP, e o cabeçalho da camada de ligação da rede Atomic é copiado para a frente do pacote. Quando um pacote é recebido, a informação de endereçamento no cabeçalho IP e no cabeçalho UDP é usada pelo núcleo para direcionar o pacote para o processo de usuário apropriado [FEL94], a figura 2.9 ilustra essas operações.

Figura 2.9 - O Encapsulamento de pacotes.

Na figura 2.10 o *host* A precisa de uma rota para o *host* B. A transmissão X para Y Mosaic é insuficiente para conectar A com B. Entretanto, encadeando-se múltiplas rotas Mosaic podem-se criar caminhos de A para B. Um exemplo de caminho é $(+6,+1)(+4,+3)(+1,0)(0,0)$ e o caminho reverso é $(-1,-3)(-4,-1)(-6,0)(0,0)$.

Figura 2.10 - Roteamento na rede Atomic

Um pacote gerado por um processador A destinado a um processador B deve conter o endereço no cabeçalho conforme ilustrado na figura 2.9. Esse endereço formado por pares XY é roteado de par em par conforme ilustrado na figura 2.10 e a cada movimento XY, o pacote é armazenado e depois enviado para o próximo processador correspondente ao próximo par XY, sendo, então o par XY anterior eliminado. O processo continua até se alcançar o par XY correspondente a zero, isto é, o destino ou processador B.

2.6.5 Address Consultant

O CE - Consultor de Endereços (*AC - Address Consultant*) é responsável pelo provimento de rotas entre *hosts*. Quando o *host* A precisa enviar um pacote ao *host* B, ele primeiro envia uma requisição para o CE que retorna a rota entre A e B, semelhante a uma requisição ARP (*Address Resolution Protocol* – Protocolo de Resolução de Endereços) para um endereço Ethernet.

Os CEs realizam duas funções básicas: a primeira é mapear e remapear a rede mantendo um mapa atualizado da sua topologia; a segunda função é usar esse mapa para oferecer rotas entre *hosts*, quando requisitadas.

Na rede Atomic, qualquer *host* pode executar o processo CE, caso ele não possa ser encontrado, adicionando-se assim tolerância a falhas do CE. Em redes grandes, pode-se otimizar a consulta empregando-se múltiplos CEs.

Um outro benefício do CE é a habilidade de balancear a carga na rede. Por exemplo, se uma conexão for utilizada para a transmissão de dados que exigem grande largura de banda, o CE deve fornecer uma rota que evite aquela utilizada por essa conexão, tentando balancear o fluxo de dados nos diversos caminhos possíveis.

2.6.6 O Desempenho

Os processadores Sun, o barramento Sbus e o processamento de *sockets* UNIX são lentos para determinar o desempenho da rede Atomic, pois o desempenho de uma conexão TCP usando pacotes de 1500 *bytes* sobre a rede Atomic foi de 20 Mbits/s. O principal problema é que a placa de interface não suporta DMA e todas as transferências de dados são feitas usando I/O programado. Para a medida do desempenho real da rede, a equipe de pesquisadores da USC/ISI carregou o código de geração de tráfego e monitoramento na memória que é compartilhada pelos processadores Sun e Mosaic. Dessa maneira, as limitações de desempenho da arquitetura Sun são evitadas. Quando pacotes de 1500 *bytes* (tamanho típico de pacotes FTP) são transferidos através de um canal Mosaic, obtém-se uma taxa de transferência de 480 Mbits/s. Para pacotes de 54 *bytes*, tais como pacotes ATM, a taxa de transferência é de 205 Mbits/s. A taxa obtida para pacotes pequenos, de 4 *bytes*, é da ordem de 788 mil pacotes por segundo (25,2 Mbits/s). Quando dois fluxos competem pelo mesmo canal Mosaic, o desempenho obtido é de 405 Mbits/s para pacotes de 500 e 1500 *bytes* e 343 Mbits/s para pacotes de 54 *bytes* [FEL94].

2.7 Myrinet

2.7.1 Introdução

A rede local Myrinet é um novo tipo de rede local de alta velocidade e custo acessível baseado na tecnologia usada para comunicação de pacotes e *switching* (chaveamento) em MPPs (*Massively Parallel Processors* – Processadores Maciçamente Paralelos). Ela tem suas raízes

nos resultados de dois projetos de pesquisa de responsabilidade da *ARPA (Advanced Research Projects Agency)*: o *MOSAIC CALTECH (Multicomputador de granularidade fina)* [SEI93] e a Rede local *ATOMIC* [FEL94].

2.7.2 Principais Características

Tal como em uma rede conectada por *switches*, a capacidade de tráfego agregado de uma rede Myrinet aumenta com o número de *hosts*. Por exemplo, uma rede Myrinet conectando 8 *hosts* com uma *switch* de 8 portas pode transportar 8 pacotes de uma vez, conseguindo uma capacidade de tráfego agregado de 10.24 Gbits/s ($8 * 1.28 = 10.24$).

Figura 2.11 - A rede Myrinet

Na figura 2.12, os pacotes estão fluindo concorrentemente através da malha da rede MPP de duas dimensões. Tão logo o cabeçalho tenha sido decodificado, na entrada no canal, é realizado o roteamento *cut-through*, o qual faz com que o pacote avance até a saída do canal requerido, caso este não esteja em uso. Novos algoritmos do roteamento *cut-through* buscam maior eficiência [COH98]. Ao contrário, no roteamento tradicional, o *store-and-forward*, o pacote é armazenado e é feita uma verificação de *checksum* em cada nó intermediário antes do pacote ser enviado para o canal de saída requerido. No roteamento *cut-through*, se o canal estiver em uso no momento do avanço do pacote, este é bloqueado, como no caso do nó no centro superior da figura, até que o canal de saída esteja disponível. O controle de fluxo é realizado em cada *link*, através do reconhecimento de uma unidade de controle de fluxo, geralmente um *byte*. Desta forma, não é necessário existir um *buffer* para armazenamento de pacotes. O roteamento *x-então-y* em uma malha de duas dimensões elimina a dependência cíclica nos roteadores, evitando-se *deadlock* [BOD95].

Figura 2.12 - Malha de rede Myrinet de duas dimensões

2.7.3 O *Hardware* da rede

A rede Myrinet possui dois tipos de ligação para a camada física, SAN e LAN e brevemente terá um terceiro. A Myricon, empresa responsável pela rede Myrinet, está desenvolvendo um tradutor de fibra ótica, transparente ao usuário.

Os componentes Myrinet são implementados com a mesma tecnologia *full-custom* VLSI CMOS – empregada atualmente no desenvolvimento de unidades centrais de processamento de estações de trabalho e computadores pessoais. Quanto à segurança, a rede Myrinet está exposta a uma pequena taxa de erros, menor que um bit por dia em uma rede de grande porte, e é bastante robusta em relação a falhas por parte de *hosts*, *switches* e cabos. Outra medida em relação a falhas é feita através do cálculo e da verificação do CRC pelo *hardware* para cada pacote sobre cada *link*.

A LAN Myrinet é uma rede local de diâmetro físico relativamente pequeno, com computadores distribuídos em uma sala ou um prédio. Os *links* podem cobrir distâncias de até 25 metros, apresentando controle de fluxo e de erros.

A Myrinet é também uma rede em nível de sistema (SAN) e pode agrupar computadores num gabinete com o mesmo desempenho, a um custo mais baixo do que o de uma rede local Myrinet. O formato do pacote e o *Software* para uma Myrinet SAN são idênticos aos de uma Myrinet LAN. O *Chip* VLSI CMOS que implementa a comunicação de rede local Myrinet possui porta SAN. Nos componentes LAN, essas portas SAN são transformadas em portas LAN por outro *Chip* VLSI CMOS [BOD95].

A rede Myrinet utiliza rotas alternativas, se disponíveis, através de *switches*, para burlar falhas de comunicação/rota, causadas, por exemplo, por desconexão física ou queda de energia. A figura 2.13 ilustra um *switch* Myrinet de 8 portas que possui 4 portas LAN e 4 SAN [MYR96].

Figura 2.13 - Switch Myrinet LAN / SAN

Cada cabo SAN conectado na *switch* comporta 2 *links*, ao passo que os 4 cabos LAN comportam apenas um *link* cada um. O cabo extra, na parte superior direita da figura, oferece 12 V DC de alimentação para cada *switch*, as quais consomem somente 6 *watts*.

2.7.4 Interface de *Software* e Desempenho

Nos termos do modelo referência ISO para redes de computadores, a rede Myrinet implementa protocolos até o nível da camada de enlace de dados.

O MCP é um programa carregado e executado no *chip* LANai na placa de interface do *host*, cuja função é receber mensagens da *workstation* e enviá-las para a Myrinet ou receber mensagens da Myrinet e enviá-las para a *workstation*, podendo, além disso, mapear e monitorar a rede continuamente.

Os *clusters* Myrinet, diferentes de outros MPPs, permitem sistemas heterogêneos com diferentes tipos de computadores, programas e sistemas operacionais, como ilustra a figura 2.14.

Figura 2.14 - Sistema de computação heterogênea

Em testes efetuados por pesquisadores da Myricom, usando o protocolo TCP/IP (ou UDP/IP) padrão em equipamentos como Pentium Pro, Sun Ultra Sparc e Alpha DEC, os *hosts* Myrinet mostraram taxas de dados de aproximadamente 250 Mbits/s, limitadas pela pilha do protocolo IP no sistema operacional. *Hosts* mais rápidos podem aumentar essa taxa executando de forma mais rápida os códigos das pilhas de protocolos. O desempenho do protocolo TCP/IP tem atendido as aplicações de computação distribuída existentes.

O desenvolvimento de alguns *Softwares* especializados tem permitido obter tempos de latência tão baixos quanto 5 μ s para pequenas mensagens entre processos de usuário no sistema UNIX. A rede Myrinet apresenta um desempenho melhor do que a maioria dos MPPs de memória distribuída, tais como o Intel Paragon e o IBM SP-1/2, com taxas que podem exceder 1 Gbits/s [why98]. Implementações de várias interfaces padrão de *software* de MPPs de memória distribuída, tais como *PVM (Parallel Virtual Machine)* e *MPI (Message Passing Interface)* [MPI95], encontram-se disponíveis para uso em uma rede Myrinet.

Alto desempenho pode ser obtido utilizando-se *APIs (Application Programming Interface)* que evitam as chamadas ao sistema operacional. O MCP ativa a interface que passa a interagir diretamente com o processo do usuário, atingindo, assim, taxas de transferências acima de 550 Mbits/s.

Características como autoconfiguração e tolerância a falhas tornam a rede Myrinet com eficiência próxima dos MPPs. Com a adição de *links* de fibra ótica, a Myrinet pode oferecer também grande largura de banda, baixa latência, baixa taxa de erros e comunicação de diferentes

tipos de pacotes sobre redes com distâncias de vários quilômetros [BOD95]. No capítulo 3 são apresentados maiores detalhes do padrão Myrinet.

2.8 ATM

2.8.1 Introdução

As novas aplicações, especialmente as de multimídia e as de tempo real, exigem cada vez mais velocidade e alta capacidade de transmissão com garantia de resposta, características estas não atendidas pelas redes tradicionais [LIE95]. A rede ATM (*Asynchronous Transfer Mode*) é uma das poucas tecnologias com capacidade de atender a essas exigências, principalmente quando o usuário passa a demandar um volume cada vez maior de informações por unidade de tempo. Sua principal característica é a flexibilidade em adaptar-se às muitas plataformas e protocolos existentes [KAR97, BOU92].

2.8.2 Características da Rede ATM

A rede ATM efetua a comutação de células, como ilustra a figura 2.15, em que cada célula contém somente 53 *bytes* fixos, as quais são identificadas e comutadas por um *label* no cabeçalho em velocidades extremamente altas[VET95]. O processo de comutação de células é de mais fácil execução que as técnicas tradicionais de multiplexação, especialmente quando se utilizam cabos de fibra ótica.

Figura 2.15 - Formato de uma célula ATM

A velocidade de 155,52 Mbits/s possibilita a transmissão de imagens de vídeo em alta definição e proporciona compatibilidade com o sistema de transmissão SONET [TAN97]. A

velocidade de 622 Mbits/s foi escolhida para permitir a utilização de quatro canais de 155 Mbits/s.

O modelo ATM é representado através de 3 camadas, como ilustra a figura 2.16.

Figura 2.16 - Modelo de camadas ATM

2.8.3 A Camada Física

A especificação da UNI (*User Network Interface*), versão 3.1 do Forum ATM, estabelece quatro padrões diferentes. O SONET (*Synchronous Optical Network*) e o STS-3c (*Synchronous Transport Signal*), que operam a 155,52 Mbits/s. O DS3, operando a 44,736 Mbits/s, e o último, conhecido pela sigla *Taxi*, que opera a 100 Mbits/s [TAN97]. Os padrões diferem entre si pela velocidade, tipo do meio físico (fibra ótica ou cabo) e a forma de estruturação dos dados. O padrão SONET é uma iniciativa americana visando à interligação de centrais telefônicas com troncos, utilizando-se fibras óticas. O ITU-T europeu formulou o padrão SDH (*Synchronous Digital Hierarchy*), praticamente equivalente ao SONET, tendo ambos surgido como uma evolução do modo TDM (*Time Division Multiplexing*), que define o modo pelo qual múltiplos fluxos de informação compartilham um meio físico de transmissão comum, através do particionamento do tempo, em intervalos discretos conhecidos como janelas de tempo [CER97].

O padrão SONET que trata de ligações por sinais elétricos (STS-1) ou sinais óticos, o OC-1, opera a 51,84 Mbits/s. A partir dessa frequência básica estão definidas várias outras frequências de operação, múltiplas da básica. Assim, o nível OC-3 opera a 155,52 Mbits/s. A frequência básica de operação do SDH é de 155,52 Mbits, conhecida como STM-1, a partir da qual são definidas várias outras frequências múltiplas [KAR97].

A UNI do padrão ATM especifica para operação a 155,52 Mbits/s distâncias para cabo coaxial de até 200 metros e distâncias para cabo de fibra ótica de até 2 quilômetros.

Os padrões SDH e SONET definem a forma como os *bytes* são montados. Tal montagem é conhecida como *frame* e está ilustrada na figura 2.17.

Figura 2.17 - Frame definido para o nível STM-1

O *frame* é organizado em 9 linhas de 270 *bytes* cada, totalizando 2.430 *bytes*. Deste total, 81 são utilizados no *ponteiro* e na *section overhead*, que servem para delineamento, identificação, verificação de erros, sinalização, gerência e manutenção. A área de *payload*, com 2.349 *bytes*, contém as informações geradas pelos usuários.

2.8.4 A Camada ATM

A Camada ATM tem a função de direcionar as informações recebidas, ou seja, enviá-las às camadas superiores, caso tenham chegado ao seu destino final, ou, caso contrário, remetê-las para o próximo ponto da rede, segundo a conexão anteriormente estabelecida [PER95]. Essas informações, como visto anteriormente, estão na forma de células ATM que possuem dois formatos distintos apenas em seus cabeçalhos, conforme se mostra na figura 2.18. Esses formatos são aplicáveis, respectivamente, à comunicação entre dispositivos através da UNI ou da NNI (*Network-Network Interface*).

Figura 2.18 - Formato dos cabeçalhos das células ATM

O significado dos diversos campos segue abaixo:

- **CFG** (*Generic Flow Control* – Controle Genérico de Fluxo): composto de 4 bits na célula UNI. É utilizado para o controle de fluxo de células, evitando congestionamento. Este campo ainda não foi completamente definido;
- **VPI** (*Virtual Path Identifier* – Identificador de Caminhos Virtual): é composto de 8 bits no cabeçalho da célula UNI e 12 bits no cabeçalho da célula NNI. Sua função é servir como a parte mais significativa do código que identifica a conexão. Formado hierarquicamente pelo par VPI + VCI, com o VPI representando a parcela mais alta nessa hierarquia;
- **VCI** (*Virtual Channel Identifier* – Identificador de Canais Virtual): com 16 bits de comprimento, é a parte menos significativa do identificador de conexão;
- **PT** (*Payload Type* – Tipo de Carga): é um campo com 3 bits que indica se a informação contida na célula é proveniente do usuário ou de gerência de rede, acompanhados de informações sobre eventuais congestionamentos encontrados;
- **CLP** (*Cell Loss Priority* – Prioridade de Perda de Célula): composto de 1 bit, indica células de mais baixa prioridade sujeitas a serem descartadas antes das outras;
- **HEC** (*Header Error Control*): contém 8 bits para controle de erro, relativo apenas ao cabeçalho. Também auxilia no delineamento de células.

2.8.5 A Camada de Adaptação ATM

A Camada de Adaptação ATM, através de suas duas subcamadas, é responsável por gerar, dependendo da aplicação, através de uma mensagem de tamanho variável, pacotes com 48 *bytes* entregues à Camada ATM. Essa última agregará o cabeçalho de 5 *bytes*, compondo, assim, a célula ATM de 53 *bytes*. A figura 2.19 ilustra esquematicamente o que ocorre com uma mensagem oriunda de uma camada mais alta, ao passar pela Camada de Adaptação ATM, em direção à Camada ATM [CER97, TAN97].

Figura 2.19 - Operação esquemática da camada de adaptação ATM

Na tentativa de evitar uma proliferação de diferentes protocolos associados à Camada de Adaptação ATM, a ITU-T considerou três conjuntos de características:

- Relação temporal entre origem e destino: considera a imunidade da aplicação que está usando a rede ATM com relação ao valor absoluto do atraso das informações e à sua variação, confrontados os instantes de chegada e envio das informações;
- Taxa de informações úteis a transmitir: leva-se em consideração a taxa das informações geradas pela aplicação para transmissão pela rede ATM. Serviços de telefonia, rádio e televisão podem ser de taxa constante. Quando as informações são submetidas a algoritmos de compressão, a taxa é variável;
- Modo de conexão: orientado à conexão e não-orientado à conexão - aqui é considerada a natureza da aplicação nas camadas superiores à Camada de Adaptação ATM.

Da combinação dessas características, surgiram quatro classes, rotuladas de A a D pela ITU-T, que representam as classes de serviços esperadas da Camada de Adaptação ATM. Simultaneamente às classes de serviços, foram definidos quatro tipos de protocolos, conhecidos pelas siglas AAL1, AAL2, AAL3 e AAL4, que originalmente seriam mapeados um a um nas classes de serviços. Os protocolos AAL3 e AAL4 foram fundidos no AAL3/4. A figura 2.20 ilustra as relações entre essas diversas entidades.

	Classe A	Classe B	Classe C	Classe D
Relação Temporal Orig./Dest.	Necessária		Não-necessária	
Taxa de informações	Constante	Variável		
Modo de conexão	Orientado à conexão			Não-orientado
Protocolo	AAL1	AAL2	AAL3/4	AAL3/4

Figura 2.20 - Classes de Serviços e protocolos na camada de adaptação ATM.

Posteriormente, foi criado o quinto protocolo, conhecido por AAL5, que procura aliar maior simplicidade, melhor capacidade de detecção e correção de erros e menor *overhead*. Esse protocolo procura atender às classes C e D.

A figura 2.21 ilustra o formato do pacote de informações definido pelo AAL5, fracionado pela subcamada de segmentação e recomposição em blocos de 48 *bytes*, todos dedicados ao *payload* da célula ATM [CER97].

Figura 2.21 - Formato AAL5 do pacote apresentado à subcamada de segmentação e recomposição.

As funções básicas dos vários campos mostrados na figura 2.21 são:

- **Dados da Aplicação:** abriga os dados provenientes das camadas superiores. Pode ter até 65.535 *bytes* de comprimento;

- **Alinha:** destina-se a preencher o espaço necessário (0 a 47 *bytes* com qualquer conteúdo) para alinhar a segmentação do pacote com a fronteira de 48 *bytes* oferecida pela subcamada de segmentação e recomposição;
- **Controle:** função ainda não definida, reservada para uso futuro;
- **Compr:** indica o comprimento do campo Dados da Aplicação;
- **CRC:** Código de Controle de Erro.

2.9 SCI – Scalable Coherent Interface

2.9.1 Introdução

Em meados de 1988 foi proposta a interface de comunicação *SCI (Scalable Coherent Interface)*, visando a três aspectos principais:

- 1 - **Escalabilidade:** a rede deve exibir alto desempenho para sistemas com pequeno ou grande número de processadores;
- 2 - **Coerência de *cache*:** nos sistemas distribuídos com memória compartilhada, a coerência de *cache* deve ser mantida pela rede de forma transparente para o processador;
- 3 - **Interface:** as interfaces não devem ser vinculadas a uma tecnologia em particular, possibilitando a implementação de interfaces de acordo com cada necessidade.

O padrão SCI utiliza ligação ponto a ponto, permitindo várias topologias, mas a de anel é a mais indicada, pela sua simplicidade e menor custo. Para limitar o tráfego no anel, uma rede SCI é composta de sub-redes contendo de 5 a 10 nós, cada um deles composto de um processador com memória *cache*, uma memória local ao nó e o controlador da interface de *link* lógico, como ilustra a figura 2.22 [TVI93].

Figura 2.22 - Nó SCI sobre uma topologia em anel

O nó SCI pode ser dividido em duas partes: a camada física que especifica as características elétricas, térmicas e mecânicas, e a camada lógica que descreve o espaço de endereço, os registradores de controle e *status*, os protocolos de transferência de dados, o mecanismo de coerência e o protocolo de coerência de *cache*.

2.9.2 Interface da Camada Física

A interface da camada física pode ser de dois tipos. No primeiro, existe um barramento unidirecional de 18 bits: 16 bits carregam dados, um bit de *flag* carrega informações sobre a sincronização de pacote e o último bit é um sinal de *clock* com frequência de sincronização de 250 MHz. Os dados são transferidos com uma frequência de transmissão de 500 MHz, permitindo uma taxa de transferência de 1 *Gbyte/s*. O segundo tipo corresponde a uma interface serial que realiza transmissões através de um cabo coaxial de 50 ohms para transmissão a curtas distâncias ou fibra ótica para distâncias maiores que 20 metros, operando na frequência de 1250 MHz. O codificador serial SCI S20 converte os 18 bits em um fluxo serial de 20 bits. Um campo de *tag* de 4 bits contém informação sobre o bit de *flag* e se os dados foram complementados ou não. Para a detecção de erros de transmissão e perda de sincronização, é garantida uma transição entre o antepenúltimo e o penúltimo bit no campo de *tag*. O formato dos dados pode ser visto na figura 2.23.

Figura 2.23 - Formato dos dados S20

A Controladora de Interface de *link* Lógico é responsável por enviar e receber dados transmitidos dentro de pacotes de requisição ou resposta na rede de interconexão. Ela decodifica pacotes que entram, extrai dados, caso existam, envia dados ou informações sobre transações à memória conectada, ao *cache* ou ao processador, e executa as transações emitidas de dispositivos acima da controladora de interface de *link* lógico através do envio do pacote adequado para a rede. Pacotes não destinados para o nó atual têm que ser passados para o nó seguinte na rede.

O SCI implementa um espaço de endereço de 64 bits, composto de *IDs* (identificadores) de nós com 16 bits, com cada nó tendo um espaço de endereço local de 48 bits (*256 Tbytes*). Os endereços mais altos, entre 0xFFFO e 0xFFFF, são reservados para propósitos especiais, deixando espaço para 65520 possíveis nós SCI em um sistema. A figura 2.24 mostra o mapa de endereçamento SCI.

2.9.3 Arquitetura CSR

O sistema de memória SCI se aplica a uma arquitetura CSR (*Control and Status Register*) comum. É um padrão IEEE (*Institute of Electrical and Eletronics Engineers*) para sistemas de memória compartilhada, que define um espaço comum de endereço para cada nó contendo registradores de controle e *status*. O CSR também contém informações sobre seqüências de *reset* e iniciação assim como identificadores de nó e informação de *timeout* de transação.

Figura 2.24 - Mapa de endereçamento do padrão SCI

2.9.4 Protocolos de Transferência

A comunicação do padrão SCI é baseada em um protocolo de pacote, em que cada um deles corresponde a uma sequência de símbolos de 16 bits com comprimento variável. A figura 2.25 ilustra o formato típico de um pacote SCI. Os campos 1 e 4 (*ID*) de 16 bits, são os endereços origem e destino, respectivamente. O campo 2 é responsável pelo controle de fluxo e, por ser alterado durante a transmissão, não participa do cálculo de CRC. O campo 3 (comando) indica se é um pacote de envio ou eco e quais ações devem ser tomadas pelo receptor. O campo 5, ou *time of death* (tempo de morte), indica o tempo em que o pacote pode ser descartado com segurança. O campo 6, ou *ID* de transação, é utilizado para comparar pacotes de eco e de resposta que retornam com as requisições destacadas, permitindo que a requisição correta possa ser removida das filas. O campo 7, ou de endereço, mantém os endereços de 48 bits para serem acessados. Em caso de transações diretas de *cache* para *cache*, esse campo é repostado pela informação de *status* e ponteiros de *cache*. O campo 8, ou de cabeçalho estendido, contém 0 ou 16 *bytes* e é reservado para extensões futuras do padrão. O campo 9, ou campo de dados, é incluído em pacotes envolvendo transferências de dados. O tamanho do bloco de dados pode ser de 16, 64, ou 256 *bytes*. O campo 10, ou de CRC, é usado para verificar a integridade do pacote.

Figura 2.25 - Formato típico de pacote-envio SCI

2.9.5 Tipos de Pacotes

Existem três tipos de pacotes: de envio, eco e iniciação podendo estes ser requisições ou respostas e possuindo cada um deles um eco correspondente. Se um pacote de envio alcançou

seu destino, o nó retira-o e gera um de eco trocando o *ID* destino pelo *ID* origem do pacote de envio recebido. A tabela 2.1 ilustra os diferentes tipos de pacotes e suas respectivas funções.

Tabela 2.1 - Tipos de Pacotes do padrão SCI

Tipo de Pacote	Descrição
Pedido-envio	Requer sub-ação que instrui o receptor a executar certas ações
Pedido-eco	Requer sub-ação transportando um reconhecimento a um pacote de pedido-envio prévio
Resposta-envio	Resposta de sub-ação que responde a uma transação de requisição prévia
Resposta-eco	Resposta de sub-ação transportando o reconhecimento a um pacote de resposta-envio prévio
Iniciação	Pacote de controle e iniciação

2.9.6 Mecanismos de Coerência

O SCI suporta três mecanismo de coerência:

1 - a implementação mínima permite uma só cópia de uma lista da memória de cada vez, a qual sempre terá que ser escrita com permissão, implicando que nenhuma leitura compartilhada possa acontecer;

2 - o conjunto típico permite distribuir cópias nos *caches* de uma lista para leitura compartilhada. Transferências por DMA, precisam acessar a memória e, portanto, também as cópias armazenadas na memória *cache* das listas envolvidas;

3 - o conjunto pleno suporta compartilhamento de dados entre dois processadores, algo comum em programas paralelos, permitindo-se que os dados propaguem diretamente entre dois *caches* sem envolver a memória ou atualização da estrutura de ponteiro. Se nenhuma operação de escrita ocorrer, a lista não tem que ser escrita de volta à memória quando substituída [TVI93].

2.9.7 O Protocolo de Coerência de *Cache* SCI

Uma das formas de se manter a confiabilidade de dados compartilhados é a utilização de um sistema de diretórios distribuídos encadeados, usados devido à impossibilidade de se aplicar

esquemas de bisbilhotagem (*snooping*) em certas topologias de redes, principalmente quando a rede passa a ter grandes proporções. A aplicação do esquema de coerência deve seguir algumas regras:

1 - leitura compartilhada: vários *caches* podem compartilhar a mesma lista de memória para propósitos de *read-only* (somente leitura);

2 - escrita exclusiva: só um *cache* por vez (cabeça de lista) pode escrever em uma lista de memória *cached*;

3 - invalidação ao escrever: a escrita de uma lista de *cache* de compartilhamento múltiplo pelo processador deve expurgar todas as outras cópias da lista e a memória deve ser informada sobre a alteração;

4 - gerenciamento: a informação das cópias *cached* de uma lista de memória deve ser armazenada de modo acessível.

Cada memória *cache* deve ser associada a um processador que realiza o protocolo de coerência de *cache* ao emitir os comandos SCI adequados. Para manter o estado de coerência de uma lista específica, cada lista de memória e de *cache* possui um campo de estado associado. Em um sistema, a lista de memória pode assumir os três estados exibidos na tabela 2.2.

Tabela 2.2 - Conjunto típico de estados coerentes de memória estável.

Nome	Descrição
MS_HOME	Não existe lista compartilhada e a memória é válida ou cópia única
MS_FRESH	Cópias de listas compartilhadas são idênticas com a memória
MS_GONE	Cópias de listas compartilhadas podem ser diferentes da memória

Se uma lista de *cache* for inválida, estado indicado por CS_INVALID, ela pode ser repostada. Uma lista ativa é consistente com a memória se não tiver sido escrita por um processador ou se for cópia única e seu estado é CS_ONLY_FRESH. O estado será CS_ONLY_DIRTY se tiver ocorrido uma escrita. Já o caso CS_HEAD_FRESH é trivial para escrita, já que a cabeça e todos os elementos dentro da lista serão consistentes com a memória. Os elementos que residem no meio da lista estarão no estado CS_MID_VALID, e o elemento de

cauda estará no estado CS_TAIL_VALID. Um processador pode adicionar-se como cabeça da lista para escrever numa dada lista de *cache*, invalidando a memória e passando a lista *cached* ao estado CS_HEAD_DIRTY. Quando uma escrita é executada, o resto da lista será apagado. Os estados da lista de *cache* podem ser vistos na tabela 2.3.

Tabela 2.3 - Conjunto típico de estados coerentes de *cache* estável.

Nome	Descrição
CS_INVALID	A lista é inválida e pode ser usada para o <i>cache</i> de novas listas
CS_ONLY_FRESH	Apenas cópia <i>cached</i> , consistente com a memória
CS_ONLY_DIRTY	Apenas cópia <i>cached</i> , com permissão de escrita e inconsistente com a memória
CS_HEAD_FRESH	Cabeça de lista não manipulada, consistente com a memória
CS_HEAD_DIRTY	Cabeça de lista válida, com permissão de escrita e inconsistente com a memória
CS_MID_VALID	Elemento intermediário de lista válida, possivelmente inconsistente com a memória
CS_TAIL_VALID	Cauda de lista válida, possivelmente inconsistente com a memória

A coerência de *cache* é vital para os sistemas distribuídos com memória compartilhada e os controlos mencionados acima podem garantir essa confiabilidade, possibilitando, assim, a implementação de sistemas para processamento paralelo.

2.10 HIPPI - High Performance Parallel Interface

2.10.1 Introdução

A partir de 1987, pesquisadores do Los Alamos National Laboratory iniciaram a busca de um padrão para a interligação de supercomputadores. A especificação inicial exigia uma taxa de transferência de dados de 800 Mbits/s para visualizar cenas com quadros de 1.024 x 1.024 pontos, com 24 bits por ponto e 30 quadros/s, representando uma taxa de dados agregada de 750 Mbits/s. Posteriormente, foi proposto um padrão com uma taxa de dados de 1.600 Mbits/s chamado padrão HIPPI [TAN97].

2.10.2 HIPPI - Camada Física

Com relação ao modelo OSI, o padrão HIPPI cobre a camada física e parte da camada de enlace de dados transportando dados em paralelo por cabos de par trançado, no qual a versão do padrão HIPPI de 800 Mbits/s usa barramento de dados de 32 bits e a versão do padrão HIPPI de 1600 Mbits/s usa barramento de dados de 64 bits.

O padrão HIPPI foi originalmente projetado para ser um canal de dados em vez de uma rede local. Os canais de dados operam ponto-a-ponto, de um mestre (computador) para um escravo (periférico), com fios dedicados e sem comutação. Aqui não há contenção e o ambiente é inteiramente previsível. Mais tarde, a necessidade de comutar um periférico de um supercomputador com o de outro tornou-se evidente e foi acrescentado um comutador *crossbar* ao projeto do padrão HIPPI, como ilustra a figura 2.26 [TOL89,TAN97].

Figura 2.26 - Comutador *crossbar* HIPPI

2.10.3 O Sinal de Interface HIPPI

Para atingir alto desempenho usando os *chips* disponíveis no mercado, a interface básica tornou-se maior, com 32 bits de dados e 18 bits de controle, totalizando 50 bits, de forma que o cabo HIPPI passou a ter 50 cabos de par trançado. A cada 40 ns, uma palavra é transferida paralelamente através da interface. Para atingir 1.600 Mbits/s, são usados dois cabos (*full-duplex*) e duas palavras são transferidas por ciclo. Para obter comunicações bidirecionais são

necessários dois (ou quatro) cabos com tamanho máximo de 25 metros [TOL89, TAN96], nessas velocidades.

O padrão HIPPI oferece a detecção de erro. Para isso é utilizado um *byte* de paridade sobre o barramento de dados, sendo cada quadro imediatamente seguido por um LLRC (*length/longitudinal redundancy checkword*). A correção de erros não é realizada, deixando-se essa função para os protocolos das camadas mais altas. Em alguns casos isso não provoca grandes perdas como no caso do transporte de quadros de vídeo, no qual é melhor ignorar erros, pois o próximo os sobrescreverá.

2.10.4 Camada de Enlace de Dados

O padrão definido pelo comitê ANSI X3T9.3 trata das camadas de enlace de dados e da camada física, ficando a cargo dos usuários a definição das camadas de nível mais alto. O protocolo básico usado diz que, para se comunicar, primeiro um *host* pede que o comutador *crossbar* estabeleça uma conexão, para, em seguida, enviar uma única mensagem e liberar a conexão.

As mensagens são estruturadas com uma palavra de controle, um cabeçalho de até 1.016 *bytes*, e uma parte de dados de até $2^{32} - 2$ *bytes*. Com a finalidade de controle de fluxo, as mensagens são divididas em quadros de 256 palavras. Quando o receptor está apto a aceitar um quadro, ele sinaliza para o transmissor, que, em seguida, envia um quadro. Os receptores também podem solicitar diversos quadros de uma só vez. O controle de erro consiste em um bit de paridade horizontal por palavra e uma palavra de paridade vertical na extremidade de cada quadro. As somas de verificação tradicionais eram tidas como desnecessárias e muito lentas [TAN96].

2.10.5 Quadro de Dados

A conexão, quando estabelecida, é similar ao uso do telefone, os pacotes podem ser enviados da fonte para o destino. Cada pacote contém 0 ou mais quadros que por sua vez contém 256 palavras. Os quadros com menos palavras compostas de 32 ou 64 bits devem ocorrer no início ou final do pacote. O tempo de espera entre pacotes e quadros pode variar [TOL89]. A figura 2.27 ilustra a organização da informação.

Figura 2.27 - Organização da informação do padrão HIPPI.

2.10.6 Outros Padrões HIPPI

Como a fibra ótica que possibilitou a substituição de 50 ou 100 fios do tipo par trançado por uma única fibra dando origem ao *fiber channel*, sucessor do padrão HIPPI.

O *fiber channel* lida com conexões de rede ponto-a-ponto e canal de dados. É aplicado em conexão serial de alto desempenho para suportar protocolos de camadas mais altas, incluindo HIPPI, IPI (*Inteligente Peripheral Interface*), SCSI (*Small Computer System Interface*) e o canal multiplexador usado nos *mainframes* da IBM [BJJ89, TAN96].

O *fiber channel*, que transporta também pacotes de rede, incluindo IEEE 802, IP e ATM, aceita três classes de serviços. A primeira é uma comutação de circuitos autêntica, com entrega garantida dos pacotes na ordem em que foram transmitidos; a segunda é a comutação de pacotes, com entrega garantida; e a terceira é a comutação de pacotes, sem entrega garantida.

Como a figura 2.28 ilustra, sua estrutura é composta de cinco camadas que cobrem as camadas de enlace de dados e a camada física do modelo ISO/OSI. A camada FC-0 lida com o meio físico e aceita taxas de dados de 100, 200, 400 e 800 Mbits/s, enquanto a FC-1 manipula a codificação de bit, no qual são usados 10 bits para codificar 256 símbolos válidos. Juntas, essas duas camadas funcionam de forma equivalente à camada física do modelo OSI.

Figura 2.28 - As camadas do protocolo do *fiber channel*

A camada FC-2 define o *layout* do quadro e os formatos do cabeçalho. Os dados são transmitidos em quadros com capacidade de até 2.048 *bytes*. A camada FC-3 permite que no futuro, sejam fornecidos serviços padrão para a camada superior, se necessário. Por fim, a camada FC-4 fornece a interface com os vários tipos de computadores e periféricos aceitos[TAN96].

2.11 Comparação dos Padrões de Rede Apresentados

A tabela 2.4 resume as principais características dos padrões de rede apresentados:

Tabela 2.4 - Principais características dos padrões de redes descritos

	ATM	MYRINET	HIPPI	SCI
Meios Físicos	Cabos elétricos Fibra ótica	Cabos elétricos Fibra ótica	Cabos elétricos fibra ótica	Cabos elétricos fibra ótica
Distância Máxima para cabos elétricos	200 metros para cabo elétricos	Menor ou igual a 10 metros	Menor ou igual a 25 metros	Menor que 20 metros
Distância Máxima para fibras óticas	2 quilômetros para fibra ótica	Em projeto	300 metros-	-
Modo de Transmissão	Serial	Paralela, 18 bits/s, 9 bits em cada direção	Paralela de 50/100 bits ou serial	Paralela de 18 bits
Topologia	Ponto-a-ponto	Ponto-a-ponto	Ponto-a-ponto	Ponto-a-ponto
Velocidade	622 Mbits	1.28 + 1.28 Gbits/s	800 Mbits/s	500 Mbits/s
Latência		7.2 μ s	160 ns	4 μ s
Disponibilidade de Interfaces PCI	Sim	Sim	sim	Sim
Custo Aproximado por Nó	\$4.200	\$1.800	> \$ 4.000	-

As duas características mais importantes das redes de alto desempenho para utilização em máquinas paralelas são a velocidade de transmissão de dados e a latência de transmissão. A Myrinet destaca-se como a rede de maior velocidade entre todas as redes analisadas, embora

comecem a aparecer no mercado redes ATM operando a 2.5 Gbits/s. A latência apresentada é baixa, embora não se tenham conseguido dados sobre a latência de produtos comerciais nos outros casos.

A disponibilidade de interfaces para o barramento PCI [CLA97] é um fator importante no projeto SPP2, uma vez que esse é o único de alto desempenho, disponível nos computadores pessoais baseados na arquitetura Intel.

A distância máxima não é um fator importante no projeto SPP2, uma vez que as placas são instaladas em bastidores e ficam bastante próximas, embora, a interligação de vários bastidores exija que essa distância seja de alguns metros.

Desde que sejam satisfeitos os requisitos de velocidade e latência, não importa o tipo de meio físico utilizado. Condutores elétricos levam uma certa vantagem pela maior facilidade de manuseio, mas novas técnicas para a conectorização de fibras óticas tem suprimido essa vantagem.

O modo de transmissão não é importante se forem satisfeitos os requisitos de velocidade e latência. Os cabos para transmissão serial ou multi-serial são, no entanto, mais simples e mais baratos.

O número máximo de nós não é um fator importante, uma vez que para o SPP2 estão previstos apenas 256 nós.

2.12 Considerações finais

Neste capítulo foram apresentados os diversos tópicos relacionados com o presente trabalho. Foi dada ênfase à descrição das principais características e detalhes dos padrões atuais de redes de comunicação de alta velocidade.

Uma comparação de alguns parâmetros das redes estudadas mostra que a rede Myrinet se destaca, levando-se em consideração seu desempenho e sua adequação ao projeto do SPP2, conforme discutido na seção 2.11. Essa análise permitiu selecionar a Myrinet para a

implementação do SPP2, embora o padrão ATM possa ser considerado, no futuro, assim que o custo de sua utilização seja reduzido.

No capítulo 3 é apresentada uma visão mais detalhada do padrão Myrinet, em que são descritos alguns detalhes adicionais de funcionamento e o *software* disponível para a operação do sistema. Também são apresentados alguns exemplos de sistemas para processamento paralelo que utilizam esse padrão de rede.

3. ESTUDO DETALHADO DA REDE MYRINET

3.1 Considerações Iniciais

No capítulo 2 foram apresentados alguns conceitos sobre redes de computadores, processamento paralelo, arquiteturas paralelas MIMD com memória distribuída, exemplificadas pelo SPP2, seguidos por diversos padrões atuais de redes de comunicação de dados de alta velocidade.

A escolha da Myrinet para a implementação do SPP2 motiva a apresentação de uma visão mais detalhada do padrão neste capítulo, sendo descritos os componentes do *hardware* na seção 3.2. Os *drivers*, o Programa de Controle Myrinet, API e BPI Myrinet e o conjunto de protocolos TCP/IP são abordados na seção 3.3. e na 3.4 são apresentados alguns exemplos de sistemas que utilizam a rede Myrinet.

3.2 O Hardware do Padrão Myrinet

3.2.1 A Interface do *Host*

A interface Myrinet efetua a conexão de um *host* à rede através de uma porta, podendo cada um deles conter mais de uma interface, embora as portas operem de forma independente.

Existem interfaces Myrinet para os barramentos PCI ou SBUS, que permitem a construção de sistemas utilizando máquinas diferentes. Essas interfaces são constituídas por um processador de 32-bits e *clock* de 33MHz, com 512 KB ou 1 MB de memória tanto para interfaces do tipo LAN quanto para interfaces do tipo SAN, utilizada para armazenar os pacotes que estão sendo transmitidos entre o *host* e a rede e para a execução do programa MCP (descrito na seção 3.3.1).

O processador, a interface com o meio físico da rede e o mecanismo de DMA estão contidos no *chip* LANai, como ilustra a figura 3.1.

Figura 3.1 - Diagrama de uma Interface Myrinet

Para a utilização da Myrinet com processadores Pentium e barramento PCI podem ser utilizados os sistemas operacionais descritos na tabela 3.1, bem como outros processadores, barramentos e sistemas operacionais.

3.2.2 Switches Myrinet

As *switches* Myrinet, componentes de múltiplas portas para sistemas de redes LANs ou SANs, podendo ser mistas com 4 ou 8 portas LANs e SANs, roteiam pacotes que chegam ao canal de entrada para uma porta de saída especificada pelo *header* do pacote. Os endereços das portas variam conforme o número de portas de uma *switch*. Uma *switch* com N portas possui endereços de 0 até N – 1, sendo descartados pacotes com endereços fora dessa faixa. A latência de uma *switch* de 8 portas, no pior caso, é de 550 ns, enquanto uma *switch* de 16 portas exibe uma latência de 300 ns. Na seção 4.3 é feita uma descrição mais detalhada de algumas *switches* Myrinet.

Tabela 3.1 - Relação dos Processadores e Sistemas Operacionais compatíveis com a rede Myrinet.

Pentium Pentium Pro Pentium II Pentium MMX	Linux
	Linux/BIP
	Solaris 2.5 / 2.6
	OSF-1
	BSDi
	NT (Beta)
	Free BSD
Sun UltraSPARC	Solaris 2.5 / 2.6
	SunOS 4.1.3/4.1.4
	Linux
DEC Alpha	DEC UNIX
	Linux
	NetBSD
	NT
SGI	Irix 6.4

3.2.3 Cabos Myrinet

A conexão entre a interface do *host* e a *switch* é efetuada por cabos SAN/LAN. Para a conexão de sistemas LAN, existem dois tipos de cabeamento: para distâncias de até 10 metros, como ilustra a figura 3.2, que possui 0.34 polegadas de diametro, 18 pares trançados, sendo 9 em cada direção e para distâncias maiores que 10 metros e até no máximo 25 metros. Ambos são muito parecidos, embora a taxa de transmissão por canal do primeiro, que é de 1.28 Mbits/s, caia no segundo tipo para 0.64 Mbits/s por canal. Os cabeamentos do tipo LAN possuem apenas um *link*, enquanto os do tipo SAN possuem dois *links*.

Figura 3.2 - Cabo LAN Myrinet.

Para conexões SAN, utilizam-se cabos SAN-Myrinet dotados de dois *links*, sendo cada *um deles* composto por um par de canais *half-duplex*, que operam a 1.28 Gbits/s e permitem operações de transmissão e recepção simultâneas, atingindo-se uma taxa de transferência de 2,56 Gbits/s. Os dois *links* são rotulados A e B. O *link* B normalmente não é utilizado, a não ser quando a conexão é entre *switches* e seu principal objetivo é aumentar a taxa de transmissão entre as *switches*, característica importante para multicomputadores semelhantes ao SPP2. Outro ponto a ser destacado é que a taxa de transmissão agregada aumenta conforme aumenta o número de nós no sistema, um fator que deve ser considerado quando é importante uma escalabilidade elevada.

Os cabos SAN possuem tamanhos-padrão que variam de 30 centímetros até 3 metros, 2,5 centímetros de largura e são muito flexíveis, como ilustra a figura 3.3.

Figura 3.3 - Cabo SAN-Myrinet

3.2.4 Acessórios Myrinet

Através do acessório M2M-Y é possível a transformação de uma porta de *switch/switch* em uma porta *switch/host*, isto é, de *link A/B* em *link A/A*, como no caso do acessório M2M-Y.

O M2M-X é um acessório utilizado para a interligação de um *link A* com um *link B* em uma *switch*, como por exemplo em conectores SAN de uma *switch* M2M-DUAL-SW8. Consegue-se o mesmo efeito empregando-se um *splitter* M2M-Y, com um custo mais elevado.

O M2M-F é utilizado para permutar um *link A* com um *link B*, útil em certas topologias. Consegue-se o mesmo efeito com dois M2M-Y e dois cabos SAN-Myrinet, com um custo maior.

O M2M-L é utilizado para efetuar *loop back* quando conectado à porta de um componente Myrinet, retornando o canal de saída de cada *link* para o canal de entrada. A figura 3.4 ilustra os acessórios Myrinet descritos.

Figura 3.4. Acessórios Myrinet.

3.3 O Software do Padrão Myrinet

3.3.1 O MCP – *Myrinet Control Program*

O *software* que controla e oferece acesso à rede Myrinet é constituído pelo MCP, executado no *chip* LANai da placa de interface do *host* e pelo *device driver* e sistema operacional que residem no *host*.

O MCP (*Myrinet Control Program* – Programa de Controle Myrinet), um programa que controla e fornece acesso à rede Myrinet através da interface do *host*, é carregado pelo *device driver* quando o *host* é iniciado e começa a ser executado tão logo o *device driver* aciona um sinal de *reset*. Depois disso, o MCP interage concorrentemente com o *host* e com a rede.

Em relação ao *host*, a interface é controlada por uma série de filas de comandos de um único produtor, um único consumidor e de reconhecimento. Por exemplo, o MCP pode ser requisitado pelo *host* para realizar uma operação de acumulação em um conjunto de blocos de dados, gerar o *checksum* no pacote resultante e inseri-lo em uma dada localização no pacote. A seguir, o MCP envia o pacote resultante ao endereço de um *host* específico, manda um sinal indicando a conclusão dessas operações para uma fila de reconhecimento e, opcionalmente, produz uma interrupção. Para realizar essa operação, é necessário controlar o mecanismo de DMA, traduzir um endereço de rede em uma rota, anexar a rota e o tipo do pacote ao pacote de saída e controlá-lo. Para realizar a tarefa de recebimento de pacotes, o MCP verifica a validade do pacote de entrada, interpreta o cabeçalho, transfere pacotes de dados para os *buffers* espalhados pela memória do *host* e envia um sinal para uma lista de reconhecimento indicando a chegada do pacote, gerando, opcionalmente, uma interrupção.

Além de interagir com o *host*, o MCP também interage com a rede, mapeando-a e monitorando-a continuamente. Essa interação permite que a rede tenha configuração e correção automáticas, seleção de rotas e roteamento *multicast* do tipo *store-and-forward*. É feita a seleção de uma interface de *host* da rede Myrinet, a qual mapeará a rede e enviará pacotes de mapeamento para todas as interfaces, inclusive ela própria. A seleção da interface mapeadora

pode ser feita manualmente pelo administrador da rede ou automaticamente pela própria rede. Se a interface mapeadora estiver desligada ou se *links* defeituosos dividirem a rede em redes disjuntas, uma nova interface mapeadora é escolhida automaticamente. Depois de feito o mapa da rede, a interface mapeadora o distribui entre as outras interfaces, as quais calculam as rotas a partir de si mesmas para outras interfaces, sendo todas as rotas livres de *deadlocks*.

O MCP é bastante confiável, em termos de segurança da rede, já que não possui “porta dos fundos” que permita a outro *host* modificar a versão corrente do MCP, que fica em um segmento de memória da interface protegido contra escrita pelo próprio MCP, somente podendo ser carregado pelo *host*.

3.3.2 A API Myrinet

O *software* do *host* Myrinet oferece uma interface de programação entre a interface do *host* e o processo de usuário. Em uma interface de “cópia única” TCP/IP padrão, o UNIX copia o bloco do espaço do usuário para o espaço do *kernel*, computa o *checksum* e manda um comando para a interface enviar o pacote. O MCP copia os dados para a memória do LANai através do mecanismo DMA e transfere o pacote para a rede utilizando sua interface. Apesar dos dados terem sido movidos três vezes, a ação é vista como uma implementação de cópia única. A interface do *host* também foi projetada para permitir operações “sem cópia”. O diagrama de cópia da figura 3.5 ilustra os passos envolvidos na transferência da informação entre o processo do usuário e a rede.

Figura 3.5 - Diagrama de cópia dos dados entre as diferentes interfaces de *software*

A API é uma interface para programadores tanto de aplicações quanto de sistemas operacionais, atuando entre a interface do *host* e o processo de usuário. Ao ser inicializada, a API aloca um certo número de páginas de memória como área de troca de dados e alcança desempenho melhor evitando a pilha de protocolos IP do Sistema Operacional.

O *reset* na rede Myrinet pode ser efetuado através de uma chamada a FRES (Forward Reset), que libera todos canais e estruturas de dados reiniciando o MCP, ou pela função `myriApiReset()`, que libera um único canal.

Quanto ao endereço, a interface do *host* possui um único endereço de 48 bits gravado em memória EEPROM. Como já visto, quando o sistema é iniciado, este endereço é transferido para o MCP. Se o bit mais alto de um endereço Myrinet é ativado, então ele se comporta como um endereço *multicast*.

3.3.3 A BPI Myrinet

A biblioteca BPI Myrinet é uma interface de alto nível para programadores de aplicação, fornecendo os mesmos serviços que a API Myrinet, mas tornando transparente ao usuário as complexidades do gerenciamento de *buffer*. A BPI utiliza uma área de memória chamada "Copy Block" para seus *buffers*, que intermedeia mensagens entre a BPI e o MCP.

3.3.4 O Driver TCP/IP

Uma versão de TCP/IP que visa a um melhor aproveitamento do desempenho oferecido pelo hardware Myrinet, foi desenvolvida por pesquisadores da Sun e atingiu uma taxa de 70 Mbits/s em modelos Sun SPARC-2 com barramento SBUS a 20 MHz [BOD95].

Essa versão do conjunto de protocolos TCP/IP permite que os aplicativos para processamento paralelo disponíveis no SPP2 possam ser executados imediatamente. Embora o conjunto de protocolos TCP/IP se destaque pela sua robustez, o desempenho da rede utilizando esse conjunto de protocolos certamente não é compatível com a utilização do SPP2. O mais indicado é a utilização da interface proprietária API ou BPI Myrinet ou outras ferramentas

desenvolvidas para o ambiente da rede Myrinet, como por exemplo, o sistema *Fast Messages* da Universidade de Illinois, que apresenta desempenho muito superior ao TCP/IP.

3.3.5 O *Fast Messages*

O *Fast Messages* é uma biblioteca de comunicação de dados, desenvolvida por uma equipe de pesquisadores da University Illinois, cujo objetivo é aumentar o desempenho da rede de comunicação de dados Myrinet, tornando-a mais eficiente para as arquiteturas paralelas MIMD com memória distribuída.

O *Fast Messages* é uma camada de baixo nível que efetua a troca de mensagens pequenas, atingindo elevada taxa de transferência de dados com baixa latência [PAK97]. O desempenho do *Fast Messages* para pacotes maiores que 256 *bytes* começa a degradar uma vez que é utilizado DMA apenas para transferir dados da interface para a memória enquanto que a API Myrinet utiliza o mecanismo de DMA da memória para a interface.

Adaptações do *Fast Messages* foram efetuadas para estações de trabalho Suns com rede Myrinet e atualmente foram desenvolvidas versões para redes Myrinet e computadores Pentium com sistemas operacionais Windows NT e Linux.

Inicialmente a API do *Fast Messages* era composta de apenas 3 funções sendo 2 para o envio de mensagens - *FM_send()* e *FM_send4()* e uma para o recebimento - *FM_extract()*. Atualmente foram incluídas novas funções como iniciar e terminar a comunicação além de outras funções para o envio de mensagens, como por exemplo, *FM_begin_message* para abrir uma conexão e *FM_send_piece* para o envio de pequenas mensagens. A nova versão possibilita também endereçar mais que um processo em um mesmo nó de processamento, assim, aplicações que necessitem de um modelo de processos dinâmico, como o modelo cliente-servidor, podem agora ser implementadas.

Ao contrário do protocolo TCP/IP, que introduz alto *overhead* na comunicação, o *Fast Messages* oferece às aplicações velocidades próximas à velocidade da camada física da rede. O desempenho do *Fast Messages* sobre o barramento PCI foi muito superior ao barramento Sbus da Sun, atingindo uma taxa de transferência maior que 77 *Mbytes* por segundo no pico [PAK97].

3.3.6 Encapsulamento Pacotes Ethernet

O padrão Myrinet encapsula pacotes Ethernet, o que lhe permite transportar, dessa forma, ARP, IP, etc. Os endereços fonte e destino Ethernet devem ser acomodados nos endereços fonte e destino do pacote Myrinet, antes da transmissão. O *driver* do servidor Myrinet pode utilizar a maioria dos códigos desenvolvidos e testados para operação com redes Ethernet.

3.3.7 O Formato do Pacote Myrinet

No nível da camada de rede e superiores, a rede Myrinet pode transportar qualquer tipo de pacote. Na verdade, a Myrinet pode ser utilizada com vários protocolos simultaneamente sobre a mesma rede.

Um pacote Myrinet inclui:

- cabeçalho, que possui informações sobre a rota a ser seguida pelo pacote através das *switches*;
- um campo para identificar o protocolo do pacote;
- carga útil de comprimento arbitrário;
- um *byte* de CRC, que é calculado sobre toda a parte precedente do pacote, inclusive o cabeçalho.

A figura 3.6 mostra a estrutura de um pacote Myrinet. Ao chegar a uma *switch*, o primeiro *byte* do cabeçalho é lido, endereçando o pacote para a porta de saída; em seguida, esse *byte* é eliminado. Por causa dessa modificação do cabeçalho em cada *switch*, o *byte* de CRC é computado e verificado a cada *link*.

Figura 3.6 Formato de um pacote Myrinet

Quando um pacote entra em uma interface, o primeiro *byte* indica o tipo do pacote que pode ser, por exemplo, um pacote IP encapsulado, uma célula ATM encapsulada, um pacote transmitido por um protocolo qualquer ou um pacote de mapeamento da rede. O bit mais significativo de cada *byte* do cabeçalho diferencia os pacotes em relação ao seu destino, ou seja, se ele será direcionado para uma *switch* ou para um *host*. Esse tipo de informação é redundante para pacotes que trafegam em rotas conhecidas, mas permite que pacotes de mapeamento trafeguem por rotas desconhecidas. Além disso, esse bit permite que erros de roteamento sejam detectados. Se o destino do pacote for um *host* e ele for direcionado para uma *switch*, ele é descartado. Se o pacote for destinado a uma *switch*, mas direcionado a um *host*, ele é consumido mas tratado como um erro. Quando o pacote é recebido por um *host*, o bit é lido pelo programa de controle e é usado para selecionar a rotina que vai lidar com o tipo específico do pacote.

A carga útil pode ser de qualquer tamanho permitindo à rede Myrinet transportar o protocolo IP diretamente, sem camadas de adaptação, e todos os protocolos sobre o IP, tais como TCP, FTP, etc. De um modo geral, a Myrinet pode transportar qualquer protocolo utilizado com o padrão Ethernet. A interface de *software* IP para a rede Myrinet é vista pelas camadas superiores como Ethernet, mas com uma MTU (*Maximum Transmission Unit* - Unidade Máxima de Transmissão) maior.

3.3.8 Myrinet, novo padrão ANSI

Recentemente, John Rynearson, diretor técnico da VITA Standards Organization, anunciou a transformação da rede Myrinet em um padrão nacional americano, reconhecido como ANSI/VITA 26-1998. A especificação do protocolo Myrinet-on-VME foi oficialmente aprovada em 2 de novembro de 1998, devendo ser publicada brevemente [ANS98].

3.4 *Sistemas existentes que utilizam a Myrinet*

Até o início de 1997, a Myrinet era utilizada por mais de 150 instituições em diversos lugares do planeta. A disponibilidade de componentes para redes de alto desempenho e o empenho da Myricom e diversas instituições de pesquisa em aprimorar e desenvolver novos componentes e protocolos para aumentar o seu desempenho tem levado a rede Myrinet a ocupar lugar de destaque no meio das redes de comunicação de dados de alta velocidade [MYR98]. A seguir são apresentados alguns sistemas que utilizam esse padrão:

- A University of California at Berkeley é responsável pelo projeto NOW (*Network of Workstation*) que consiste em um conjunto de *workstations* (100 SUN Ultrasparc e 40 Sparcstation executando Solaris, 35 PCs Intel com Windows NT ou UNIX) interligados por uma rede comutada de alta velocidade, a Myrinet, envolvendo pesquisas em áreas como redes de comunicação de alta velocidade, sistemas operacionais distribuídos, sistemas de arquivos escaláveis, *web* sobre NOW, ambientes de programação e aplicações. Seu objetivo é alcançar dois pontos: melhor custo/desempenho para aplicações paralelas que as oferecidas pelas arquiteturas MPPs e melhor desempenho para aplicações sequenciais que as *workstations* individuais podem oferecer [BER98];

- A University of Southern California / Information Sciences Institute (ISI) é responsável pelo projeto Netstation que está utilizando componentes Myrinet para criar uma malha de E/S para uma estação de trabalho de alto desempenho. O ISI implementou um protocolo de transferência confiável sobre a API Myrinet, que utiliza PVM sobre o padrão Myrinet e também está trabalhando com um protocolo de alta velocidade para extrair mais velocidade da rede Myrinet [NET97];

- A ASCI (Advanced School for Computing and Imaging) é responsável pelo DAS (Distributed ASCI Supercomputer), que consiste em quatro *clusters* compostos por 200 nós de processamento, utilizando o sistema operacional BSD/OS (v.3.0). A interconexão de cada *cluster* é efetuada através do padrão Myrinet/Fast Ethernet com localização em diferentes universidades, formando um *cluster* distribuído que abrange uma área geográfica ampla [DAS98];

- A University of Illinois é responsável pelo projeto HPVM (*High Performance Virtual Machines*) sobre conjuntos de processadores Pentium executando o sistema operacional Linux ou Windows NT, conectados por uma rede de comunicação de dados de alta velocidade, a Myrinet. O projeto HPVM fornece ao programador as interfaces MPI, SHMEM, Global Arrays e Fast Messages para desenvolvimento de aplicações paralelas. Deverá ser incorporada, brevemente, uma versão do HPF (*High Performance Fortran*) [ILL97];

- O ICDCCR - Integrated Concurrent and Distributed Computation Research Laboratory, laboratório da Mississippi State University está construindo um novo MCP de propósito geral, assim como bibliotecas de *thread* para o *chip* LANai e o LANai debugger. Também escreveu a primeira implementação do PacketWay e MPI++ que está sendo migrado para a rede Myrinet, disponibilizando o código fonte do MPI++ baseado em objetos. Essas ferramentas que foram testadas sobre Suns/Solaris e LANai v.3.x e Windows NT, disponibiliza também um conjunto de tutoriais e demonstrações do MCP [INT98];

- A Hebrew University of Jerusalem é responsável pelo projeto MOSIX - Multicomputer Operating System for UNIX. Esse projeto está modificando o UNIX para que a rede de *workstations* (NOW) [BAR98] possa agir como um multicomputador de recursos compartilhados escalonável. Nesse projeto, o *kernel* do BSD/OS é melhorado com balanceamento de carga dinâmico e migração de processos preemptiva, compartilhando recursos de CPU de um PC NOW. O MOSIX está sendo implementado para diferentes versões do UNIX e para diferentes arquiteturas de computadores [HEB98] ;

- A University Duke é responsável pelo projeto Trapeze cujo objetivo é desenvolver novas técnicas para comunicação de alta velocidade e acesso rápido para dados armazenados em *clusters* de estações de trabalho com sistemas UNIX. A plataforma desse

trabalho é um *cluster* DEC Alpha e estações de trabalho baseadas na tecnologia Intel que são interconetados por uma rede Myrinet. Também faz parte desse projeto o desenvolvimento de *drivers* para sistemas DEC Alpha e um MCP completo, otimizado para obter comunicação sem-cópia via RPC [AND98].

Além desses exemplos, existem várias outras instituições que utilizam o padrão Myrinet em seus sistemas de processamento de alto desempenho, sendo possível acessar parte desses *sites* a partir da *homepage* da Myricom (<http://www.myri.com>).

3.5 Considerações Finais

Neste capítulo foram apresentados, de uma forma mais minuciosa, os detalhes que compõem o padrão de rede Myrinet.

Nesse sentido, foram descritos os acessórios Myrinet que compõem o seu *hardware*, como a interface do *host e acessórios*. O *software* do sistema também foi descrito com mais detalhes incluindo os *drivers*, o MCP, a API e a BPI. Completando esse quadro, também foi apresentada uma biblioteca desenvolvida por uma instituição usuária da rede Myrinet, o Fast Messages.

Também foram citados alguns exemplos de projetos de pesquisa que têm utilizado o padrão Myrinet. A diversidade, tamanho e importância desses projetos permitem avaliar o nível de utilização do padrão Myrinet pela comunidade científica interessada na pesquisa em computação paralela e redes de comunicação de alto desempenho.

Além da adoção da rede Myrinet por várias instituições, ela foi recentemente transformada em padrão ANSI, o que comprova a sua importância como opção para redes de comunicação de dados de alto desempenho.

No próximo capítulo são apresentados os detalhes da organização e implementação do SPP2, juntamente com uma proposta de topologia para uma rede com 256 nós processadores.

4. A Rede de Comunicação de Alto Desempenho do SPP2

4.1 Considerações Iniciais

No capítulo 3 foram vistos detalhes dos componentes de *hardware* e *software* do padrão Myrinet, além de exemplos de sistemas que o utilizam. Neste capítulo é apresentada uma nova versão da rede de comunicação de alta velocidade do SPP2 baseada em componentes Myrinet.

Conforme mencionado no capítulo 2, a rede de comunicação de alta velocidade projetada para o SPP2 não pode ser implementada, pela dificuldade de obtenção dos componentes eletrônicos necessários para isso. Na seção 4.2 são apresentados os detalhes adicionais da organização e implementação do SPP2, necessários para a definição da nova rede de comunicação. A proposta da nova topologia de rede e a motivação da sua escolha é feita na seção 4.3 juntamente com a seleção dos componentes de rede Myrinet disponíveis.

4.2 Detalhes da Organização e Implementação do SPP2

A arquitetura do SPP2 segue a tendência da maioria das arquiteturas MIMD com memória distribuída disponíveis no mercado, isto é, os nós são compostos por placas com até quatro microprocessadores e memória compartilhada, (atualmente cada placa contém somente um processador). Essas placas são interligadas por uma rede de comunicação de alto desempenho. A utilização de mais de um processador por nó faz da arquitetura do SPP2 uma arquitetura MIMD híbrida, utilizando ao mesmo tempo os paradigmas de programação de memória compartilhada e memória distribuída. É possível utilizar placas de processadores Pentium Intel, DEC Alpha ou PowerPC [TRI95]. Na verdade, o único requisito para o processador é que seja disponível para ele uma versão do sistema operacional Linux. A quantidade de memória depende da placa utilizada, mas pode atingir 1 *Gbyte* por processador, com os componentes disponíveis hoje no mercado.

A arquitetura do SPP2 é escalável podendo chegar até 256 nós, conectados ao *host* através de duas redes de comunicação: a primeira é uma Ethernet encarregada de efetuar controle e gerenciamento do sistema e a segunda, uma rede de alta velocidade para a comunicação de dados, a Myrinet.

Os nós processadores são organizados em módulos. Originalmente, cada módulo do SPP2 pode conter até 16 nós. Um único módulo pode conter até 32 processadores, caso a máquina paralela seja constituída por apenas um módulo.

O *host* tem a função de servidor de arquivos e servidor de boot remoto. Opcionalmente, cada nó pode ter seu disco local, o que torna mais ágil o armazenamento e acesso de dados individuais dos processos de cada nó. A área de memória *swap*, nesse último caso, pode ser local em contraposição ao caso de nós sem disco cuja área de swap nó reside no *host*.

Quando são usados nós com mais de um processador, o sistema operacional fica encarregado da distribuição de processos. É possível a utilização de programas paralelos escritos de forma a aproveitar a maior velocidade disponível para comunicação entre processos que estão sendo executados na mesma placa (utilizando a memória compartilhada da placa), ao passo que processos em placas diferentes devem efetuar a comunicação por passagem de mensagens através da rede de comunicação de alto desempenho.

O sistema operacional utilizado é compatível com o UNIX, o Linux. Sua utilização por vários sistemas semelhantes ao SPP2 provocou um aumento na disponibilidade de *software* para processamento paralelo para essa classe de sistemas. Uma indicação nesse sentido é a publicação do documento *Linux Parallel HOWTO* [LIN98] que relata os principais programas para processamento paralelo utilizando redes de estações de trabalho Linux. Esses programas podem ser utilizados no SPP2 embora nem todos possam fazer uso apropriado dos recursos da rede de comunicação de alto desempenho disponível.

A figura 4.1 ilustra a nova arquitetura do SPP2. Cada módulo contém 16 nós de processamento, facilitando-se a implementação física do sistema e mantendo as mesmas características da implementação inicial do SPP2. A rede de comunicação de alto desempenho originalmente prevista (com uma topologia *fat tree*) foi substituída pela rede Myrinet. Na

próxima seção é discutida e proposta uma nova topologia de rede em função dos componentes de rede Myrinet disponíveis.

Figura 4.1. A arquitetura do SPP2.

4.3 A Topologia da Rede de Alto Desempenho do SPP2

4.3.1 Interconexão dos Nós

Vários componentes Myrinet são utilizados para efetuar a conexão dos nós processadores do SPP2 e seu tipo depende do número de processadores e da distância entre os processadores. Para a interconexão de dois processadores necessita-se apenas de duas placas de Interface Myrinet mais o *link* (cabo). Essa conexão pode ser do tipo SAN ou LAN. Componentes LAN são utilizados para redes locais de diâmetro pequeno, enquanto componentes SAN são adequados para interligar processadores dentro de um gabinete, a exemplo do SPP2. Existem *switches* com portas SAN/LAN, que podem ser utilizados para a conversão entre os dois padrões. A figura 4.2 ilustra a conexão de dois computadores.

Figura 4.2. Conexão simples entre dois computadores com a rede Myrinet.

Para a conexão de nós adicionais devem ser utilizadas *switches*. As *switches* Myrinet possuem quatro, oito, doze ou dezesseis portas, tendo sido selecionados dois tipos selecionadas para a interligação dos nós de processamento: a M2M-DUAL-SW8, que possui 8 portas duplas, quatro de cada lado da *switch*, utilizada para efetuar conexão entre *switches* e a M2M-OCT-SW8, que possui 16 portas simples de um lado para a conexão de 16 processadores e 8 portas duplas de outro lado para conexão com outras *switches* [SEI98]. As figuras 4.3(a) e 4.3(b) ilustram os dois tipos de *switches* descritos acima. Para uma máquina com n nós, são necessárias n interfaces Myrinet SAN/PCI, n cabos SAN e uma quantidade de *switches* suficiente para a realização da topologia da rede de comunicação desejada.

(a)

(b)

Figura 4.3 - Esquema da *switch* M2M-DUAL-SW8 (a) e da *switch* M2M-OCT-SW8 (b)

A seguir são apresentadas as propostas de topologia para o SPP2 a partir de 8 nós, respeitando-se a escalabilidade desejada. As *switches* a serem empregadas dependem da quantidade de nós e da necessidade do valor agregado da taxa de transmissão.

4.3.2 Topologia mínima para o SPP2

Com uma *switch* M2M-DUAL-SW8, além dos cabeamentos e das interfaces dos processadores, podem-se conectar até 8 processadores de forma simples e direta, como ilustra a figura 4.4.

Figura 4.4 - Módulo do SPP2 com 8 nós

Para um número maior de nós é possível a adição de acessórios do tipo M2M-Y, cuja função é dividir um *link* duplo (A/B) em dois *links* A, como ilustra a figura 3.4. A aquisição de uma *switch* do tipo M2M-DUAL-SW8 é vantajosa somente quando não se pretende utilizar mais de 8 nós no SPP2.

4.3.3 Topologia com 32 nós para o SPP2

Para implementação de uma máquina com mais de 8 nós, pode-se utilizar a *switch* do tipo M2M-OCT-SW8, que permite a conexão de 1 a 24 nós de forma direta. Nesse módulo pode-se chegar a

ter até 32 nós processadores, utilizando-se o acessório M2M-Y conectados nas portas de *lin* duplos, isto é, nas portas para interconexão *switch/switch*. A figura 4.5 ilustra essa configuraçã

Figura 4.5 - Topologia para 24 nós ou 32 nós.

4.3.4 Topologia BBW Completa

Nos exemplos anteriores a topologia da rede é totalmente conectada. Quanto maior for o tamanho da rede, maior será a importância da topologia da rede [TAN97], pois ela determinará o grau de eficiência na transferência de dados entre os nós processadores. Quanto maior for a capacidade de transmissão agregada da rede, maior a possibilidade de melhorar o *speedup* das aplicações paralelas. Topologias que possibilitam a transmissão de dados com todos os nós enviando e recebendo pacotes ao mesmo tempo são topologia BBW completa (*Bisection Bandwidth*).

Em uma topologia BBW completa, a rede é dividida ao meio, de tal forma que cada parte da rede tem a mesma quantidade de nós. Cada parte possui uma ou mais *switches* que têm *switches* correspondentes na outra parte da rede. Dessa forma, a rede como um todo é composta

por pares de *switches* e a quantidade de *links* de cada parte da rede é igual ao número de *links* da outra parte. A BBW é definida, matematicamente, como o número mínimo de *links* que cruza as duas partes da rede. Em uma rede com 128 nós, 64 ficam de um lado e 64 nós ficam do outro lado, sendo necessários 64 *links* para a conexão de todos os nós, que é o valor de BBW completa. A grande importância de uma topologia BBW completa é que a capacidade agregada da rede cresce estatisticamente conforme cresce o valor de BBW, tornando-a atrativa para aplicações em multicomputadores [COH97].

4.3.5 Topologia 64 nós para o SPP2

Aplicando-se uma topologia BBW completa, pode-se obter no SPP2, 64 nós totalmente conectados com uma taxa de transmissão agregada na rede de 40.96 Gbits/s, como ilustra a figura 4.6. Para essa topologia é necessária a interconexão combinada de *switches* do tipo M2M-DUAL-SW8 juntamente com *switches* do tipo M2M-OCT-SW8, na qual a *switch* M2M-DUAL-SW8 (fabricada para efetuar conexões *switch/switch*) efetua a conexão entre as *switches* do tipo M2M-OCT-SW8. Nessa configuração, pode-se conseguir uma configuração com 48 nós, bastando, para isso, excluir um dos módulos, isto é, uma *switch* M2M-OCT-SW8.

Figura 4.6 - Topologia com 64 nós para o SPP2

4.3.6 Topologia BBW completa com 128 nós para SPP2

A figura 4.7 ilustra uma topologia com 128 nós. Para implementá-la são necessárias 8 *switches* M2M-OCT-SW8, 8 *switches* M2M-DUAL-SW8, 128 interfaces Myrinet SAN/PCI, 128 cabos SAN de 3 metros (*switch/host*) e 64 cabos SAN de 3 metros (*switch/switch*). Esta topologia atinge uma capacidade agregada da rede de 81.92 Gbits/s.

Figura 4.7 - Topologia BBW completa com 128 nós processadores para o SPP2

4.3.7 BBW Parcial com 256 nós para o SPP2

Para uma rede com 256 nós ainda não é possível implementar uma topologia BBW completa pois, para isso, deveria estar disponível uma *switch* do tipo M2M-OCT-SW8 com 16 portas, sendo 8 de cada lado da *switch* com conexões *switch/switch* (esta *switch* se pareceria com a M2M-DUAL-SW8 que possui 8 portas com 4 de cada lado). A topologia possível para uma máquina com 256 nós, como no caso do SPP2, é a implementação de uma topologia BBW

parcial para 128 nós, em dois níveis, com *links* efetuando a conexão entre esses níveis, como ilustra a figura 4.8. Para implementar essa topologia, são necessárias 16 *switches* M2M-OCT-SW8, 256 interfaces Myrinet SAN/PCI, 64 cabos SAN de 3 metros (*switch/switch*), 256 cabos SAN de 3 metros (*switch/host*). Esta topologia atinge uma capacidade agregada de $\approx 81,92$ Gbits/s.

Figura 4.8 - Topologia BBW parcial com 256 nós processadores para SPP2

4.3.8 Relação custo/capacidade agregada da rede

Quando se utiliza a topologia BBW, existe um compromisso entre o custo e o desempenho da rede. Uma topologia que emprega uma BBW não completa pode ser adquirida a um preço mais acessível, mas com menor capacidade agregada de transmissão. Se a topologia será utilizada em um multicomputador para aplicações que necessitam comunicação de alto desempenho, é aconselhável procurar uma topologia que possibilite a maior capacidade agregada possível. A tabela abaixo relaciona o custo e o desempenho das topologias proposta para o SPP2.

Tabela 4.1 - Custo e desempenho para BBW completa ou parcial.

N = 16, BBW = 8 (completa)					
Qt	Componentes	Unidade	Total	Subtotal	Gbits/s
16	Myrinet SAN/PCI interface	1,300	20,800		
16	Cabos SAN de 3 metros	140	2,240	23,040	
1	Switch M2M-OCT-SW8	6,000	6,000	6,000	
	Custo por nó = \$1,815		Total	29,040	10.24
N = 32, BBW = 8 (1/2 completa)					
32	Interface Myrinet SAN/PCI	1,300	41,600		
32	Cabos SAN de 3 metros	140	4,480	46,080	
1	Switch M2M-OCT-SW8	6,000	6,000		
8	M2M-Y	40	320	6,320	
	Custo por nó = \$1,638		Total	52,400	≈20.48
N = 32, BBW = 16 (completa)					
32	Interface Myrinet SAN/PCI	1,300	41,600		
32	Cabos SAN de 3 metros	140	4,480	46,080	
2	Switch M2M-OCT-SW8	6,000	12,000		
8	Cabos SAN de 3 metros (switch/switch)	140	1,120	13,120	
	Custo por nó = \$1,850		Total	59,200	20.48
N = 64, BBW = 32 (completa)					
64	Interface Myrinet SAN/PCI	1,300	83,200		
64	Cabos SAN de 3 metros	140	8,960	92,160	
4	Switch M2M-OCT-SW8	6,000	24,000		
4	Switch M2M-DUAL-SW8	2,000	8,000		
32	Cabos SAN de 3 metros (switch/switch)	140	4,480	36,480	
	Custo por nó = \$2,010		Total	128,640	40.96

N = 128, BBW = 64 (completa)					
Qt	Componentes	Unidade	Total	Subtotal	Gbits/s
128	Interface Myrinet SAN/PCI	1,300	166,400		
128	Cabos SAN de 3 metros	140	17,920	184,320	
8	Switch M2M-OCT-SW8	6,000	48,000		
8	Switch M2M-DUAL-SW8	2,000	16,000		
64	Cabos SAN de 3 metros (<i>switch/switch</i>)	140	8,960	72,960	
Custo por nó = \$2,010			Total	257,280	81.92
N = 256, BBW = 64 (1/2 completa)					
256	Interface Myrinet SAN/PCI	1,300	332,800		
256	Cabos SAN de 3 metros	140	35,840	368,640	
16	Switch M2M-OCT-SW8	6,000	96,000		
64	Cabos SAN de 3 metros (<i>switch/switch</i>)	140	8,960	104,960	
Custo por nó = \$1,850			Total	473,600	> 81.92

Infere-se da tabela acima que o custo por conexão de cada nó do SPP2, no caso de configurações completas (todos os nós instalados), varia aproximadamente 23% .

Configurações até 32 nós necessitam apenas de uma *switch* M2M-OCT-SW8, apresentando a melhor relação custo/benefício entre todas as configurações.

4.4 Considerações Finais

Neste capítulo foram apresentados os detalhes adicionais da organização e implementação do SPP2 e as modificações da arquitetura original para a utilização dos componentes de rede Myrinet. A topologia proposta permite que, a partir de um módulo, o usuário possa adicionar outros processadores ou módulos, à medida que suas necessidades de poder computacional cresçam com o tempo. Também foi feita uma análise do custo de implementação do sistema, de acordo com o número de processadores.

No próximo capítulo são apresentados alguns testes preliminares da rede Myrinet em comparação com outros padrões de rede comuns. Procura-se demonstrar o ganho de desempenho real que se pode conseguir com a utilização do SPP2 em relação a uma rede de estações de trabalho implementada com padrões de rede comuns no mercado.

5. ANÁLISE DE DESEMPENHO DA REDE MYRINET

5.1 Considerações Iniciais

No capítulo 4 foi apresentada a nova organização do SPP2, particularmente a nova topologia da rede de comunicação e os componentes necessários para a sua implementação.

Neste capítulo são apresentados alguns testes preliminares da rede Myrinet e a comparação dos resultados obtidos com os de outros padrões de rede comumente utilizados na implementação de redes locais. Neste sentido, na seção 5.2 é feita a descrição do ambiente de testes utilizado incluindo seus equipamentos e programas. Na seção 5.3 são descritos os resultados obtidos e na seção 5.4 é feita a análise desses resultados.

5.2 Descrição do Ambiente de testes

Para a realização das avaliações de desempenho dos sistemas de comunicação estudados, foi montada uma bancada de testes constituída de dois computadores interligados pelas redes de comunicações Ethernet, Fast Ethernet e Myrinet, como ilustra a figura 5.1.

Figura 5.1 Bancada de testes com dois computadores

O *host* (servidor de boot) ilustrado na figura 5.1 possui a seguinte configuração:

- Microcomputador Pentium II de 333 MHz;

- Sistema Operacional Linux ;
- 1 Disco Rígido de 6 GB;
- 256 Mbytes RAM;
- Placa de Interface Myrinet;
- Placa de Interface Fast Ethernet;
- Placa de Interface Ethernet.

O processador P0 ilustrado na figura 5.1 possui a seguinte configuração:

- Microcomputador Pentium II de 400 MHz;
- Sistema Operacional Linux ;
- 128 Mbytes RAM;
- Placa de Interface Myrinet;
- Placa de Interface Fast Ethernet;
- Placa de Interface Ethernet.

O processador (indicado por P0), conforme ilustrado na figura 5.1, é iniciado remotamente através da rede Ethernet, simulando um dos nós da arquitetura SPP2. Essa bancada de testes foi utilizada para substituir por processadores mais modernos o protótipo disponível do SPP2, implementado utilizando-se processadores Intel Pentium de 133 MHz.

5.2.1 O Sistema Operacional

O sistema operacional utilizado é o Linux, conforme descrição anterior da arquitetura SPP2. Apesar da possibilidade de realização desses testes utilizando-se outros sistemas operacionais, o Linux foi utilizado de forma exclusiva pelos seguintes motivos:

- É o sistema operacional utilizado no SPP2;
- É bastante confiável e exige uma baixa carga computacional para a sua execução, ao contrário do Windows NT e outras versões do sistema operacional UNIX;
- O crescimento da comunidade de processamento paralelo usuária do Linux acelerou o desenvolvimento de ferramentas paralelas para esse ambiente, aumentando a disponibilidade de *software* para processamento paralelo;
- Grandes companhias como a SUN e DEC, detentoras de UNIX proprietário, passaram a oferecer o Linux como um sistema operacional alternativo para as suas máquinas;

5.2.2 Padrões de Redes de Computadores Utilizados

Os seguintes padrões de rede de comunicação de dados foram utilizados para a realização das avaliações:

1. *Host* e Estação de Trabalho conectados pela rede Ethernet – nesta configuração, ilustrada pela figura 5.2, a taxa de transmissão nominal é de 10 Mbits/s, com os seguintes componentes:

- Hub Ethernet de 10 Mbits/s (D-LINK DE-812 TP+);
- Placa de Rede de 10 Mbits/s nos dois pontos da rede (Intel Pro 100 +);
- Cabos de conexão par trançado 10-BaseT.

Figura 5.2 Conexão de dois computadores pelo padrão Ethernet de 10 Mbits/s

2. *Host* e Estação de Trabalho conectados pela rede Fast Ethernet – nesta configuração, ilustrada pela figura 5.3, a taxa de transmissão nominal é de 100 Mbits/s, sendo seus componentes:

- Hub Fast Ethernet de 100 Mbits/s (D-LINK DFE-812 TX+);
- Placa de Rede de 100 Mbits/s nos dois pontos da rede (Intel Pro 100 +);
- Cabos de conexão par trançado categoria 5.

Figura 5.3 Conexão de dois computadores pelo padrão Fast Ethernet de 100 Mbits/s

3. *Host* e Estação de Trabalho conectados pela rede Myrinet – nessa configuração, ilustrada pela figura 5.4, a taxa de transmissão nominal é de 1.28 Gbits/s por canal, ou 2.56 Gbits/s, considerando-se um *link full-duplex*, isto é, de transmissão e recepção simultâneas. Os componentes de rede utilizados nessa configuração foram:

- *Switch* Myrinet 2.56 Gbits/s por *link*;

- Placas de interface de rede Myrinet de 2.56 Gbits/s nos dois pontos da rede;
- Cabos de conexão SAN Myrinet;

Figura 5.4 Conexão de dois computadores pelo padrão Myrinet de 2.56 Gbits/s

5.2.3 Software de Comunicação

Os parâmetros básicos utilizados para se medir o desempenho de sistemas de comunicação de dados são a latência e a taxa de transferência. A latência define o tempo que o primeiro elemento dos dados leva para se deslocar da origem até o destino, enquanto a taxa de transferência mede a quantidade de dados enviados entre dois processadores por unidade de tempo.

O teste de desempenho aplicado à rede de comunicação foi realizado visando a avaliar o padrão Myrinet contra os dados publicados pelo seu fabricante, comparando-os também com os dados obtidos de padrões de redes locais de computadores, comumente utilizados.

Além dessas configurações também foram efetuados testes com a biblioteca Fast Messages, adaptada para o padrão Myrinet pelos pesquisadores da universidade de Illinois, apresentada na seção 3.3.5.

O protocolo utilizado para o desenvolvimento dos programas é o UDP (*User Datagram Protocol*) em virtude de sua disponibilidade em todos os sistemas e configurações testados. O UDP também é o protocolo de transporte que impõe o menor custo computacional, minimizando a influência da carga computacional imposta pelo TCP/IP nos resultados.

O código desenvolvido para o cálculo de latência é composto por dois programas, o primeiro funciona como servidor e o segundo, como cliente. Ambos são executados nos dois computadores utilizados no teste. Para se obter a latência, pacotes são enviados através da rede de comunicação de dados entre os dois processadores, no estilo *ping-pong*, segundo o qual um pacote é transmitido e recebido por 100 vezes e o tempo total decorrido é dividido pelo número de pacotes enviados. Esse procedimento de cálculo da latência não corresponde à definição clássica apresentada no início desta seção. Na verdade, essa definição inclui o tempo de transporte dos dados do pacote. Os tempos obtidos por essa definição variam com o tamanho do pacote. Esse procedimento, entretanto, é o utilizado pela Universidade de Illinois nos seus programas de teste e ele foi adotado neste trabalho para facilitar a comparação dos resultados obtidos. Os tamanhos dos pacotes utilizados são: 10, 100, 1000, 10000 e 50000 *bytes*. Os programas utilizados podem ser descritos pelos algoritmos abaixo:

/* Algoritmo para cálculo de latência a ser executado como servidor */

```
inicializa enderecos
atribui numero de porta
cria socket datagrama
liga endereco ao socket
Se ligação ok
  Enquanto verdade
    gera (tamanho_mensagem, numero_envio, destino);
    Para i = 0; i < numero_envio; incrementa i
      buffer ← quantidade_de_caracteres(tamanho_mensagem);
      envia (buffer, tamanho_mensagem);
      libera (buffer);
```

/* Algoritmo para cálculo de latência a ser executado como cliente */

```
inicializa enderecos;
atribui numero de porta;
abre socket datagrama;
conecta;
t1 ← Tempo_inicial

Para (i = 0; i < numero_envio; incrementa i)
  buffer ← quantidade_de_caracteres(tamanho_mensagem);
  Enquanto verdade
    le (buffer, tamanho_mensagem);
encerra socket;
t2 ← Tempo_final;
Imprime resultados;
```

O próximo conjunto de programas foi codificado para efetuar o cálculo da taxa de transferência de dados entre os dois computadores, sendo os pacotes enviados em um único sentido entre os dois processadores. Obtém-se o resultado a partir da soma dos tamanhos dos pacotes enviados dividida pelo tempo total de envio. A taxa de transferência também é composta de dois programas, um para executar no servidor e o segundo para executar no cliente, permitindo, mediante seleção, a transmissão de pacotes de dados de tamanhos distintos entre os dois pontos da rede, conforme mostrado abaixo:

- 1 -> pacote com 10 bytes
- 2 -> pacote com 100 bytes
- 3 -> pacote com 1000 bytes
- 4 -> pacote com 10.000 bytes
- 5 -> pacote com 50.000 bytes

/* Algoritmo para o cálculo da taxa de transferência a ser executado no servidor */

```
inicializa enderecos
atribui numero de porta
cria socket datagrama
liga endereco ao socket
Se ligação ok
  Enquanto verdade
    gera (tamanho_mensagem, numero_envio, destino);
    count ← tamanho_mensagem * numero_envio;
    buffer ← quantidade_de_caracteres(count);
    envia (buffer, tamanho_mensagem);
    libera (buffer);
  encerra socket;
```

/* Algoritmo para o cálculo da taxa de transferência a ser executado no cliente*/

```
inicializa enderecos
buffer ← aloca_espaco(count);
atribui numero de porta
abre socket datagrama;
conecta;
t1 ← Tempo_inicial

Para (i = 0; i < numero_envio; incrementa i)
  buffer ← quantidade_de_caracteres(tamanho_mensagem);
  le (buffer,tamanho_mensagem);
encerra socket;
t2 ← Tempo_final;
Imprime resultados;
```

Para o teste de latência sobre a rede Myrinet, utilizou-se a biblioteca Fast Messages da universidade de Illinois, que a disponibilizou para testes e pesquisas [ILL98]. Os programas-fonte utilizados para os testes efetuados estão anexos no Apêndice A. A utilização dos programas da Universidade de Illinois permite comparar diretamente os resultados obtidos e os publicados pela instituição.

5.3 Resultados Obtidos

Para a obtenção da taxa de transferência e da latência, foram enviados pacotes com tamanhos de 10 bytes, 100 bytes, 1000 bytes, 10.000 bytes e 50.000 bytes respectivamente, utilizando UDP/IP sobre Ethernet, Fast Ethernet, Myrinet e Myrinet (*Fast Messages*).

Os resultados obtidos nesses testes constam na tabela 5.1 e 5.2 onde a latência está em microssegundos enquanto a taxa de transferência está em Mbytes/segundos.

Tabela 5.1 - Resultado da taxa de transferência em MBytes/segundo

Tamanho do Pacote (bytes)	Ethernet 10 Mb/s	Fast Ethernet 100 Mb/s	Myrinet (UDP)	Myrinet (Fast Messages)
10	0,12	0,35	0,28	3,02
100	0,41	3,76	2,21	4,02
1000	0,76	10,96	11,91	30,38
10.000	1,13	11,36	18,46	68,68
50.000	1,14	11,41	20,06	69,47

Tabela 5.2 - Resultado da latência em microssegundos

Tamanho do Pacote (bytes)	Ethernet 10 Mbits/s	Fast Ethernet 100 Mbits/s	Myrinet (UDP)	Myrinet (Fast Messages)
10	120	57	146	3,13
100	188	64	150	4,13
1000	919	150	189	11,91
10.000	8613	1080	587	152,10
50.000	43033	5203	2118	666,16

5.4 Análise dos Resultados

A figura 5.5 ilustra a latência para pacotes de 10, 100 e 1000 bytes respectivamente, enquanto a figura 5.6 ilustra a latência para pacotes de 10.000 e 50.000 bytes. Os resultados foram representados através de duas figuras para melhor visualização.

Figura 5.5 Latência em Microssegundos

Figura 5.6 Latência em Microssegundos

Analisando os gráficos das figuras 5.5 e 5.6 observa-se que:

- O padrão Ethernet, dos cinco testes de latência para pacotes com 10, 100, 1000, 10.000 e 50.000 *bytes*, teve a maior latência em quatro testes, sendo que a latência aumenta mais rapidamente com o aumento no tamanho do pacote do que nos outros padrões;
- O padrão Fast Ethernet teve menor latência que o padrão Myrinet utilizando UDP para pacotes de até 1000 *bytes*, no entanto, para testes com pacotes de 10.000 e 50.000 *bytes* o padrão Myrinet com UDP obteve menor latência;
- Em todos os testes de latência, o padrão Myrinet utilizando Fast Messages obteve a menor latência;
- A latência obtida pelo padrão Myrinet com UDP para pacotes de 10 *bytes* foi de 146 microssegundos contra 3.13 microssegundos com Fast Messages;

Os resultados obtidos no teste da taxa de transferência para diversos tamanhos de pacotes sobre os padrões Ethernet 10 Mbits/s, Fast Ethernet 100 Mbits/s, Myrinet (UDP) e Myrinet (Fast Messages) estão ilustrados na figura 5.7.

Figura 5.7 - Taxa de transferência em Mbytes/s

Analisando o gráfico da figura 5.7, observa-se que:

- O padrão Ethernet apresenta a menor taxa de transferência, dentre os cinco testes efetuados para pacotes com 10, 100, 1000, 10.000 e 50.000 *bytes*, Esse resultado é o esperado e não apresenta novidades;
- Para os testes com pacotes de 10 e 100 *bytes*, o padrão Fast Messages obteve melhores resultados que o padrão Myrinet com UDP, tornando-se um possível padrão para algumas aplicações, pois ainda apresenta um custo bem inferior;

- O padrão Myrinet com UDP apresenta melhores resultados que o Fast Messages para os testes com pacotes de 1000, 10.000 e 50.000 *bytes*, sendo que a taxa de transferência aumenta conforme aumenta o tamanho do pacote;
- O padrão Myrinet apresenta os melhores resultados quando utiliza a biblioteca Fast Messages, para todos os testes efetuados. Para pacotes com 50.000 *bytes*, a taxa de transferência atinge 555.76 Mbits/s.
- Este resultado está abaixo dos resultados publicados pela Myricom e contatos estão sendo feitos para uma avaliação mais precisa dos mesmos.
- A análise dos resultados obtidos nos testes revela uma clara superioridade do padrão Myrinet sobre os demais padrões de rede utilizados na comparação. Essa superioridade, entretanto, somente é bem caracterizada com a utilização de Fast Messages, revelando a importância dessa biblioteca de comunicação no cenário das redes de alto desempenho. Com a utilização de UDP, a Myrinet chega a ser mais lenta em alguns casos.
- Por outro lado, o desempenho máximo obtido com a Myrinet fica aquém do esperado. Essa discrepância está no momento sob investigação mais detalhada.
- bom desempenho do padrão Fast Ethernet sobre UDP para pacotes pequenos sugere que a utilização do padrão Gigabit Ethernet possa ser uma boa opção. Isto depende entretanto, da queda dos preços dos seus componentes e de testes de desempenho que comprovem a sua eficiência.

5.5 Considerações Finais

Neste capítulo foram apresentados alguns testes capazes de permitir a avaliação do desempenho da rede Myrinet em comparação com padrões comuns de redes locais de computadores. Apesar de os resultados obtidos apresentarem algumas discrepâncias em relação aos publicados pelo fabricante, o desempenho apresentado pela rede Myrinet é muito superior aos dos outros padrões utilizados na comparação, colocando os sistemas baseados na Myrinet em um patamar de

desempenho muito superior ao de sistemas para processamento paralelo baseados em redes de estações de trabalho convencionais.

No próximo capítulo são apresentadas as conclusões finais deste trabalho, abrangendo as suas principais contribuições e algumas sugestões para sua continuidade.

6. CONCLUSÕES

6.1 *Considerações Iniciais*

O principal objetivo deste trabalho foi a proposta de uma nova rede de comunicação de alta velocidade para o SPP2. Neste sentido, foram estudadas as principais tecnologias disponíveis atualmente de redes para essa classe de aplicações, como apresentado no capítulo 2. No capítulo 3 foi efetuada uma descrição mais detalhada do padrão Myrinet, envolvendo o *hardware*, o *software* e alguns exemplos de sistemas que utilizam esse padrão. No capítulo 4 foram apresentados de forma mais detalhada, a organização e a implementação do SPP2, bem como a proposta de uma nova topologia de rede utilizando os componentes Myrinet. No capítulo 5 foi feita uma descrição dos testes de desempenho efetuados e dos resultados obtidos, com subsequente discussão dos resultados.

Neste capítulo são apresentadas, na seção 6.2, as contribuições deste trabalho, enquanto que na seção 6.3 são feitas algumas sugestões para sua continuidade.

6.2 *Contribuições deste Trabalho*

Com a execução deste trabalho, foram atingidos vários objetivos, valendo ressaltar algumas contribuições adicionais:

- Estudo detalhado dos padrões de rede de alto desempenho, disponíveis no mercado, permitindo o entendimento das suas características e aplicações;
- Estudo completo do padrão de rede Myrinet, permitindo não somente sua escolha para a rede de comunicações do SPP2, mas também facilitando a realização de futuros trabalhos baseados nessa tecnologia;
- Proposta de uma topologia para o SPP2, que pode agrupar máquinas com até 256 nós de processamento;

- Avaliação da rede Myrinet, constatando-se um bom desempenho, com uma taxa de transmissão acima de 500 Mbits/s;
- Identificação de algumas discrepâncias entre os valores de desempenho publicados para a latência da rede Myrinet e os obtidos nos testes.

6.3 Sugestões para Trabalhos Futuros

As seguintes sugestões podem gerar trabalhos futuros dando continuidade ao projeto em questão, ou ainda motivando o desenvolvimento de outros projetos:

- Desenvolvimento de um aplicativo para a avaliação do desempenho de redes de computadores utilizando os protocolos de software mais comuns e os programas *driver* disponíveis para os principais sistemas operacionais;
- Avaliação mais detalhada da tecnologia ATM para aplicação no SPP2, principalmente dos novos componentes com velocidade acima de 2 Gbits/s;
- Avaliação da tecnologia Gigabit Ethernet quanto à sua aplicabilidade em multicomputadores;
- Avaliação do desempenho de outros protocolos de comunicação modernos sobre a rede Myrinet, como por exemplo Active Messages [EIC95, ACT98].

Referências Bibliográficas

- [ACT98] Active Messages. University Berkeley at California.
<http://now.cs.berkeley.edu/>
- [ALM94] ALMASI, G.S.; GOTTLIEB, A. **Highly parallel computing**. 2.ed. The Benjamin Cummings Publishing Company, Inc., 1994.
- [AND98] ANDERSON, Darrell; CHASE, Jeff; GADDE, Syam; GALLATIN, Andrew; YOCUM, Ken; and FEELEY, Mike. **Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet**. Proceedings of the 1998 USENIX Technical Conference, New Orleans, June 1998.
- [ANS98] ANSI/VITA 26-1998. <http://www.myri.com/news/98b02.html>, 1998.
- [BAR98] Barak A.; LA'ADAN, O., The MOSIX Multicomputer Operating System for High Performance Cluster Computing. **Journal of Future Generation Computer Systems**, Vol. 13, No. 4-5, p. 361-372, March 1998.
- [BER98] **Berkeley Network of Workstations (NOW)**. <http://now.cs.berkeley.edu/>, 1998.
- [BOD95] BODEN, Nanette J.M. **Myrinet – A Gigabit-per-Second Local-Area Network**. IEEE Micro, v.15, n. 1, p.29-36, february 1995.
- [BOU92] LI BOUDEC, J-Y., The Asynchronous Transfer Mode: a Tutorial. **Computer Network and ISDN System**, n.24, p.279-309, 1992.
- [CER74] CERF, V.; KAHAN, R. A Protocol for Packet Network interconnection. **IEEE trans. On Commum**, v. COM-22, p.637-648, may de 1974.
- [CLA97] CLARK, Elizabeth. The PCI Connection. **NETWORK MAGAZINE**, p.69 - 73, setptember 1997.
- [COH97] COHEN, Danny; SEITZ, Chuck. **M2M-OCT-SW8: Octal 8-Port Myrinet-SAN Switch**. Myricom Inc., <http://www.myri.com/myrinet/switches/m2m-oct-sw8.html>, 1997.
- [COH98] COHEN, Johanne; FRAIGNIAUD, Pierre. Optimized Broadcasting and Multicasting Protocol in Cut - Through Routed Nestwork. **Parallel and Distributed Systems**. V.9, n.8, p.788 - 803, august 1998.
- [CPU98] **CPU Info Center**. <http://infopad.eecs.berkeley.edu/CIC/>, Nov 1998.
- [DAS98] **DAS - The Distributed ASCI Supercomputer**.

<http://www.cs.vu.nl/~bal/das.html>, 1998.

- [DAY83] Day, J.D.; ZIMMERMANN, P.R. The OSI Reference Model. **Proc.of the IEEE**, v.71, p.1334-1340, dez.de 1983.
- [DUN90] DUNCAN, R. A. A Survey of Parallel Computer Architectures. **IEEE Comput. Mag.**, v.23, n.2, p.5-16, February 1990.
- [EIC95] Eicken, von Thorsten. **Active Messages: A Mechanism for Integrated Communication and Computation**. Proc.of the 19th International Symposium on Computer Architecture, p.256-266, May 1992.
- [FEL94] FELDERMAN, Robert. ATOMIC: A High-Speed Local Communication Architecture. **Journal of High Speed Networks** 1, p 1-28, 1994
- [FLO94] FLORISI, Danilo and YEMINI, Yechiam. **Protocols for Loosely Synchronous Networks**. IFIP/IEEE Protocols for High-Speed Networks, 1994.
- [FLY82] FLYNN, M.J. Some Computer Organization and Their Effectiveness. **IEEE Transactions on Computers**, New York, 21:948-60, sept. 1982.
- [FOS95] FOSTER, Ian. **Designing and Building Parallel Programs: concepts and tools for parallel software engineering**. Addison-Wesley Publishing Company, Inc, 1995.
- [GEI94] GEIST, G.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHECK, R., SUNDERAM, V. **PVM: Parallel Virtual Machine – A user's guide and tutorial for network parallel computing**. The MIT Press, 1994.
- [HEB98] Hebrew University of Jerusalem - Computer Science Institute. **MOSIX - Multicomputer Operating System for UNIX**. <http://www.cs.huji.ac.il/mosix>, 1998.
- [HIL91] HILLIS, W.D. **Massively Parallel Supercomputing: The Connection Machine**. University Communications: Stanford, 1991.
- [HWA87] HWANG, K.; BRIGGS, F.A. **Computer Architecture and Parallel Processing**. McGraw-Hill, 1984.
- [ILL98] Illinois Fast Messages (FM). <http://www-csag.cs.uiuc.edu/projects/comm/fm.html>, 1998.
- [INT98] **Integrated Concurrent and Distributed Computation - Mississippi State University**. <http://WWW.ERC.MsState.Edu/labs/icdcr/>, 1998.

- [KAR97] KARVÉ, Anita. ATM in the Fast Lane. **NETWORK MAGAZINE**. P.48-62, july 1997.
- [LIE95] LIEIRA, J.; MOREIRA, E.D.S. **Utilização de Som e Imagem em Sistemas de Gerenciamento de Redes de Computadores**. Submetido para o SEMISH, Canela, RS, 1995.
- [LIN98] Linux Parallel Howto.
<http://www.realnames.com/Resolver.dll?provider=1&realName=linux+parallel+howto>, 1998.
- [MPI95] **EPCC MPI course**. EPCC Training and Education Center, www.epcc.ed.ac.uk/epco-tec/package.html, 1995.
- [MYR96] MYRINET, A Brief, Technical Overview.
www.myri.com/myrinet/overview.html, 1996.
- [MYR98] Myrinet Customer Projects.
http://www.myri.com/myrinet/customer_projects/index.html, 1998.
- [NAH95] NAHRSTEDT, K.; HOSSAIN, A.; KANG, S.M. **Probe-based Algorithm for QoS Specification and Adaptation**. University of Illinois, 1995.
- [NAV89] NAVAUX, Philippe O.A. Introdução ao Processamento Paralelo. **Revista Brasileira de Computação**, v.5, n.2, p.31-43, out/dez 1989.
- [NET97] **Netstation Project**. University of Southern California. <http://www.isi.edu/>, 1997.
- [PAK97] PAKIN, S.; KARAMCHETI, V.; CHIEN, A. Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively - Parallel Processors. A ser publicado em **IEEE Concurrency**, 1997.
- [PER95] PERLOFF, Michael; REISS, Kurt. **Improvements to TCP Performance in High-Speed ATM Networks**. Communications of the ACM, v.38, n2, February 1995
- [PER96] PEREZ, P. A. S. **P2D – Um ambiente de auxílio à paralelização de aplicações Fortran**. São Carlos: Universidade de São Paulo, 1996. 156.p
- [SEI98] SEITZ, Chuck; COHEN, Danny. **M2M-DUAL-SW8: Dual 8-Port Myrinet-SAN Switch**. Myricom Inc., <http://www.myri.com/myrinet/switches/m2m-dual-sw8.html>, 1998.

- [TAN89] TANENBAUM, Andrew S. **Computer Networks**. Prentice-Hall, Inc, 1989.
- [TAN97] TANENBAUM, Andrew S. **Redes de Computadores**. Prentice-Hall, Inc, 1996.
- [TOL89] TOLMIE, Don. HIPPI: Simplicity Yields Success. **IEEE Network Magazine**, vol.7, p. 28-32, janeiro/fevereiro de 1993.
- [TRI94] TRINDADE, O.; MARQUES, E.; JEUKENS, I. A parallel architecture based on personal computers - requeriments and definitions - na overview. In: XV International Conference of the Chilean Computer Society, p.479-490, November 1995.
- [TRI95] TRINDADE, O.; MARQUES, E.; JEUKENS, I. A Parallel Architecture Based on Personal Computers - An Overview. University of São Paulo, USP, São Carlos, Brazil, 1994.
- [TRI98] TRINDADE, O. LaSD/SPP2 - Uma Arquitetura Paralela de Alto Desempenho Baseada em Computadores Pessoais. Universidade de São Paulo, USP, São Carlos, Brasil, 1998.
- [TVI93] TVING, Ivan. **MULTIPROCESSOR INTERCONNECTION USING SCI**. Advanced Computer Research Institute in France, 1994. 165 p. Master Thesis. <http://www.SCIzzL.com/HowToGetSCIidx.html>, august, 1994.
- [VET95] VETTER, Ronald J. **ATM Concepts, Architectures, and Protocols**. Communications of the ACM, v.38, n2, February 1995
- [WHY98] **Why Myrinet?** <http://www.myri.com/myrinet/pitch.html>, 1998.

Apêndice A

Programa Fonte para o Cálculo de Latência {servidor}

```
#include <fm.h>
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char    myname[256];
    int     i,s,t,flag=1,len,len1;
    struct sockaddr_in sa;
    struct sockaddr from,from1;
    struct hostent *hp;
    char *buffer;
    ULONG count,msgSize,numSends,r;

    memset(&sa, 0, sizeof(struct sockaddr_in));
    gethostname(myname, 256);
    hp= gethostbyname(myname);
    if (hp == NULL)
        return(-1);
    sa.sin_family= hp->h_addrtype;
    sa.sin_port= 2000;
    if ((s= socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        return(-1);
    if (bind(s,(struct sockaddr*)&sa,sizeof(struct sockaddr_in)) < 0) {
        close(s);
        printf("bind failed.\n");
        return(-1);
    }

    while(1)
    {
        recvfrom(s,&msgSize,sizeof(ULONG),0,&from1,&len1);
        recvfrom(s,&numSends,sizeof(ULONG),0,&from,&len);
        printf("msgSize: %d, numSends: %d\n",msgSize, numSends);
        buffer = (char*)malloc(msgSize);

        for(i=0;i<numSends;i++)
        {
            /* clear our address */
            /* who are we? */
            /* get our address info */
            /* we don't exist !? */
            /* this is our host address */
            /* this is our port number */
            /* create socket */
            /* bind address to socket */
        }
    }
}
```

```

        r = 0;
        while(r<msgSize)
            r = r + recvfrom(s,buffer,msgSize,0,&from,&len);
        sendto(s,buffer,msgSize,0,&from,1,len);
    }
    printf("Done.\n\n");
    free(buffer);
}
}

```


Programa Fonte para o Cálculo de Latência {cliente}

```
#include <fm.h>
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{ struct sockaddr_in sa;
  struct hostent *hp;
  int a, s, i, r;
  ULONG t1,t2,tElapsed;
  char *buffer;
  ULONG msgSize,numSends;

  if (argc!=4) {
    printf("usage: latc <server> <msgSize> <numSends>\n");
    exit(0);
  }

  msgSize = atoi(argv[2]);
  numSends = atoi(argv[3]);

  if ((hp= gethostbyname(argv[1])) == NULL) {          /* do we know the host's */
    errno= ECONNREFUSED;                               /* address? */
    return(-1);                                         /* no */
  }

  buffer = (char*)malloc(msgSize);

  memset(&sa,0,sizeof(sa));
  memcpy((char *)&sa.sin_addr, hp->h_addr, hp->h_length); /* set address */
  sa.sin_family= hp->h_addrtype;
  sa.sin_port= 2000;

  if ((s= socket(hp->h_addrtype, SOCK_DGRAM, 0)) < 0) /* get socket */
    return(-1);
  if (connect(s, (struct sockaddr *)&sa, sizeof sa) < 0) /* connect */
    close(s);
    return(-1);
  }

  write(s, &msgSize, sizeof(ULONG));
  write(s, &numSends, sizeof(ULONG));
```

```

t1 = microsecond_timer();

for(i=0;i<numSends;i++)
{
    write(s,buffer,msgSize);
    r = 0;
    while(r<msgSize)
        r = r + read(s,buffer,msgSize);
}

close(s);
t2 = microsecond_timer();

tElapsed = t2 - t1;
printf ("Start time   : %8lu us\n", t1);
printf ("Finish time  : %8lu us\n", t2);
printf ("Elapsed time  : %8lu us\n", tElapsed);
printf ("Message size  : %8lu bytes\n", msgSize);
printf ("No. messages  : %8lu\n", numSends);
printf ("Avg. latency   : %8.2f us/message (one-way)\t\t<==\n",
        ((double) tElapsed) / (2*numSends));
printf ("Avg. bandwidth: %8.2f MB/s\n",
        ((double) (2 * msgSize * numSends * 1000000)) /
        (tElapsed * 1048576));
}

```

Programa Fonte para o Cálculo da Taxa de Transferência {servidor}

```
#include <fm.h>
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char    myname[256];
    int     i,s,t,flag=1,len,len1;
    struct  sockaddr_in sa;
    struct  sockaddr from,from1;
    struct  hostent *hp;
    char *buffer;
    int count,msgSize,numSends,r,aux;
    fd_set rfd;

    memset(&sa, 0, sizeof(struct sockaddr_in));
    gethostname(myname, 256);
    hp= gethostbyname(myname);
    if (hp == NULL)
        exit(0);
    sa.sin_family= AF_INET;
    sa.sin_port= 2000;
    if ((s= socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        exit(0);
    if (bind(s,(struct sockaddr*)&sa,sizeof(struct sockaddr_in)) < 0) {
        close(s);
        printf("bind failed.\n");
        exit(0);
    }

    while(1)
    {
        recvfrom(s,&msgSize,sizeof(int),0,&from,&len);
        recvfrom(s,&numSends,sizeof(int),0,&from,&len);
        printf("msgSize: %d, numSends: %d\n",msgSize, numSends);

        FD_ZERO(&rfd);
        FD_SET(s, &rfd);

        count = msgSize*numSends;
    }
}
```

/* clear our address */

/* who are we */

/* get our address info */

/* we don't exist !? */

/* this is our host address */

/* this is our port number */

/* bind address to socket */

```

buffer = (char*)malloc(count);
printf("count = %d \n",count);
aux = i = r = 0;
while(r<count)
{
    select(s+1,&rfd, NULL, NULL, NULL);

    if((aux=recvfrom(s,buffer,count,0,&from,&len))== -1 )
        perror("erro:");
    r = r + aux;
    //printf("(%d) Bytes received=%d \n\n",++i,r);
}

printf("(%d) Bytes received=%d \n\n",i,r);

sendto(s,&flag,sizeof(int),0,&from,len);

free(buffer);
close(s);
}
}

```

Programa Fonte para o Cálculo da Taxa de Transferência {cliente}

```
#include <fm.h>
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{ struct sockaddr_in sa;
  struct hostent *hp;
  int o, s, i, flag;
  ULONG t1,t2,tElapsed;
  char *buffer;
  int msgSize,numSends,w;
  fd_set wfds;

  if (argc!=4) {
    printf("usage: bondc_udp <server> <msgSize> <numSends>\n");
    exit(0);
  }

  msgSize = atoi(argv[2]);
  numSends = atoi(argv[3]);

  if ((hp= gethostbyname(argv[1])) == NULL) {
    errno= ECONNREFUSED;
    exit(0);
  }

  buffer = (char*)malloc(msgSize);

  memset(&sa,0,sizeof(sa));
  memcpy((char *)&sa.sin_addr,hp->h_addr,hp->h_length);
  sa.sin_family= AF_INET;
  sa.sin_port= 2000;

  if ((s= socket(AF_INET,SOCK_DGRAM,0)) < 0)
    exit(0);

  if (connect(s,(struct sockaddr *)&sa,sizeof(sa)) < 0) {
    close(s);
    exit(0);
  }
```

```
/* do we know the host's */
/* address */
/* no */
```

```
/* set address*/
```

```
/* get socket*/
```

```
/* conect*/
```

```

write(s,&msgSize,sizeof(int));
write(s,&numSends,sizeof(int));

t1 = microsecond_timer();

FD_ZERO(&wfds);
FD_SET(s, &wfds);

for(i=0;i<numSends;i++)
{
    if (select(s+1,NULL,&wfds,NULL,NULL))
        write(s,buffer,msgSize);
}

read(s,&flag,sizeof(int));

close(s);
t2 = microsecond_timer();

tElapsed = t2 - t1;
printf ("Start time   :   %8lu us\n", t1);
printf ("Finish time   :   %8lu us\n", t2);
printf ("Elapsed time   :   %8lu us\n", tElapsed);
printf ("Message size   :   %8lu bytes\n", msgSize);
printf ("No. messages   :   %8lu\n", numSends);
printf ("Avg. latency   :   %8.2f us/message\n",
        ((double) tElapsed) / numSends);
printf ("Avg. bandwidth:   %8.2f MB/s\t\t<==\n",
        (msgSize * numSends * 1000000.0) / (tElapsed * 1048576.0));

}

```

Programa Fonte para o Cálculo da Latência com FM

```
/*****
 *
 * This file is part of the Illinois High Performance Virtual Machines Project
 *
 * The Illinois High Performance Virtual Machines (HPVM) software is *not* in
 * the public domain. However, it is freely available in binary form
 * without fee for education, research, and non-profit purposes. By
 * obtaining copies of this and other files that comprise the Illinois
 * High Performance Virtual Machines (HPVM) Project, you, the Licensee,
 * agree to abide by the following conditions and understandings with
 * respect to the copyrighted software:
 *
 * 1. The software is copyrighted in the name of the Board of Trustees
 *    of the University of Illinois (UI), and ownership of the software
 *    remains with the UI.
 *
 * 2. Permission to use, copy, and modify this software and its documentation
 *    for education, research, and non-profit purposes is hereby granted
 *    to Licensee, provided that the copyright notice, the original author's
 *    names and unit identification, and this permission notice appear on
 *    all such copies.
 *
 * 3. Licensee may not distribute this software or any derivative thereof
 *    in any form without explicit written permission from:
 *
 *        Professor Andrew A. Chien          achien@cs.uiuc.edu
 *        University of Illinois
 *        Department of Computer Science
 *        2215 Digital Computer Laboratory
 *        1304 West Springfield Avenue
 *        Urbana, Illinois 61801
 *        USA
 *
 * 4. Licensee may not use the name, logo, or any other symbol of the UI
 *    nor the names of any of its employees nor any adaptation thereof in
 *    advertising or publicity pertaining to the software without specific
 *    prior written approval of the UI.
 *
 * 5. THE UI MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
 *    SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS
 *    OR IMPLIED WARRANTY.
 *
 * 6. The UI shall not be liable for any damages suffered by Licensee from
 *    the use of this software.
 *
 * 7. The software was developed under agreements between the UI and the
 *    Federal Government which entitle the Government to certain rights.
 */
```

```

*****
*
* Developed by: The CSAG Research Group
*               University of Illinois at Urbana-Champaign
*               Department of Computer Science
*               1304 W. Springfield Avenue
*               Urbana, IL 61801
*
* Copyright (C) 1995,1996,1997
* The University of Illinois Board of Trustees.
* All Rights Reserved.
*
* Project Manager and Principal Investigator:
* Andrew A. Chien <achien@cs.uiuc.edu>
*
*****/

/*****
* File : latency.c
* Author : Scott Pakin and Matt Buchanan
* Date : 12 June 1997
* RCS : $Id: $
*
* Syntax : latency message-size num-sends progid-string
* Summary: FM 2.1 latency test receiver program measures the ping-pong latency
*          between two nodes. It transmits num-sends messages of
*          message-size bytes there and back. One-way latency is reported as
*          half the round-trip latency.
*****/

#include "fm.h"
#include <stdio.h>
#include <stdlib.h>

#define NUM_WARMUP_MSGS 5

/* FM handlers */
ULONG fmhReceiver;
ULONG fmhSenderDone;

/* Other variables */
ULONG numSends;
ULONG msgSize;
ULONGLONG *buffer;
BOOLEAN msgHasArrived;
FM_stream *replyStream;
ULONG t1, t2;
ULONG tElapsed;
BOOLEAN senderIsDone = FALSE;
char *progrname;
const ULONG recvrId = 0;

```



```

const ULONG senderId = 1;

/*****
 * Handler increments the global msgCount and optionally touches each
 * byte of the message
 */
#pragma FM_declare_handler
int receiver (FM_stream *s, unsigned senderId)
{
    #pragma FM_begin_decls
    #pragma FM_end_decls

    #ifdef TOUCHDATA
        if (msgSize)
            FM_receive (buffer, s, msgSize);
    #endif

    msgHasArrived = TRUE;

    return FM_CONTINUE;
} /* End receiver() */

/*****
 * Handler sets the global flag senderIsDone.
 */
#pragma FM_declare_handler
int senderDone (FM_stream *s, unsigned senderId)
{
    #pragma FM_begin_decls
    #pragma FM_end_decls

    senderIsDone = TRUE;

    return FM_CONTINUE;
} /* End senderDone() */

/*****
 * Send data to the remote process
 */
void senddata(void)
{
    ULONG i;

    /* Send a warmup message and wait for the receiver's reply */
    (void) FM_reallocate_credit();
    msgHasArrived = FALSE; /* Initialize way up here because sends can extract */
    if (!(replyStream = FM_begin_message (recvId, msgSize, fmhReceiver))) {
        fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
    }
}

```

```

    FM_finalize();
    exit (1);
}
FM_send_piece (replyStream, buffer, msgSize);
FM_end_message (replyStream);
while (!msgHasArrived)
    FM_extract (~0);
printf ("%s: Received warmup message reply; beginning test.\n", progname);

/* Let the test begin! */
t1 = microsecond_timer();
for (i=0; i<numSends; i++) {
    msgHasArrived = FALSE;
    if (!(replyStream = FM_begin_message (recvrd, msgSize, fmhReceiver))) {
        fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
        FM_finalize();
        exit (1);
    }
    if (msgSize)
        FM_send_piece (replyStream, buffer, msgSize);
    FM_end_message (replyStream);

    while (!msgHasArrived)
        FM_extract (~0);
}
t2 = microsecond_timer();

/* Say how well we did */
tElapsed = t2 - t1;
printf ("Start time   : %8lu us\n", t1);
printf ("Finish time   : %8lu us\n", t2);
printf ("Elapsed time   : %8lu us\n", tElapsed);
printf ("Message size   : %8lu bytes\n", msgSize);
printf ("No. messages   : %8lu\n", numSends);
printf ("Avg. latency   : %8.2f us/message (one-way)\t\t<==\n",
      ((double) tElapsed) / (2*numSends));
printf ("Avg. bandwidth: %8.2f MB/s\n",
      ((double) (2 * msgSize * numSends * 1000000)) /
      (tElapsed * 1048576));

#ifdef TOUCHDATA
/* Tell the receiver we're done sending */
if (!(replyStream = FM_begin_message (recvrd, 0, fmhSenderDone))) {
    fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
    FM_finalize();
    exit (1);
}
FM_end_message (replyStream);
#endif

/* Clean up */

```

```

    free (buffer);
}

/*****
* Receive data from the remote process
*/
void receivedata(void)
{
    ULONG i;

    /* Wait for a warmup message from sender and respond to it */
    msgHasArrived = FALSE;
    while (!msgHasArrived)
        FM_extract (~0);
    printf ("%s: Received warmup message; replying.\n", progname);
    (void) FM_reallocate_credit(); /* We know all the nodes are here */
    if (!(replyStream = FM_begin_message (senderId, msgSize, fmhReceiver))) {
        fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
        FM_finalize();
        exit (1);
    }
    FM_send_piece (replyStream, buffer, msgSize);
    FM_end_message (replyStream);

    /* Begin our part of the test -- wait for a message, then reply to it */
    for (i=0; i<numSends; i++) {
        msgHasArrived = FALSE;
        while (!msgHasArrived)
            FM_extract (~0);

        if (!(replyStream = FM_begin_message (senderId, msgSize, fmhReceiver))) {
            fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
            FM_finalize();
            exit (1);
        }
        if (msgSize)
            FM_send_piece (replyStream, buffer, msgSize);
        FM_end_message (replyStream);
    }

#ifdef TOUCHDATA
    /* Wait until we've really received all bytes in the message */

    /* Note that, in this case, it would be erroneous to terminate without
    /* waiting for acknowledgment from the sender. Because we're not touching
    /* the data, the sender may still be sending the last message and needs us
    /* to stay alive in order to consume its data.
    while (!senderIsDone)
        FM_extract (~0);
#endif

```

```

    /* Clean up */
    free (buffer);
}

#ifdef _WIN32
/*****
 * Try to shut FM down cleanly
 */
BOOL shutdownFM (DWORD event)
{
    FM_finalize();
    return FALSE;
}
#endif

/*****
 * main()
 */
void main (int argc, char *argv[])
{
    char *key;          /* Program identifier */

    /* Parse command line arguments */
    progname = argv[0];
    if (argc != 4) {
        fprintf (stderr, "Usage: %s message-size num-sends program-key\n", progname);
        exit (1);
    }
    msgSize = atol (argv[1]);
    numSends = atol (argv[2]);
    key = argv[3];

    /* Initialize Fast Messages */
    FM_set_parameter (FM_KEY_NAME, key);
    FM_initialize();
    fmhReceiver = FM_register_handler (FM_ANY_ID, receiver);
    fmhSenderDone = FM_register_handler (FM_ANY_ID, senderDone);

#ifdef _WIN32
    /* Optional, but polite to other processes */
    SetConsoleCtrlHandler((PHANDLER_ROUTINE)shutdownFM, TRUE);
#endif

    /* Initialize the bandwidth test */
    buffer = (ULONGLONG *) malloc (msgSize);
    if (!buffer) {
        fprintf (stderr, "%s: Memory allocation failed.\n", progname);
        FM_finalize();
    }
}

```

```

    exit (1);
}

/* Call either sending or receiving routine, based on our node number */
switch (FM_nodeid) {
    case 0:                /* Receiver */
        receivedata();
        break;
    case 1:                /* Sender */
        senddata();
        break;
    default:               /* Error */
        fprintf (stderr, "%s: The bandwidth test already contains two running processes. Aborting.\n",
progname);
        FM_finalize();
        exit (1);
}

/* Finish up */
FM_finalize();
exit (0);
}

```

Programa Fonte para o Cálculo da Taxa de Transferência com FM

```
/*****  
*  
* This file is part of the Illinois High Performance Virtual Machines Project  
*  
* The Illinois High Performance Virtual Machines (HPVM) software is *not* in  
* the public domain. However, it is freely available in binary form  
* without fee for education, research, and non-profit purposes. By  
* obtaining copies of this and other files that comprise the Illinois  
* High Performance Virtual Machines (HPVM) Project, you, the Licensee,  
* agree to abide by the following conditions and understandings with  
* respect to the copyrighted software:  
*  
* 1. The software is copyrighted in the name of the Board of Trustees  
* of the University of Illinois (UI), and ownership of the software  
* remains with the UI.  
*  
* 2. Permission to use, copy, and modify this software and its documentation  
* for education, research, and non-profit purposes is hereby granted  
* to Licensee, provided that the copyright notice, the original author's  
* names and unit identification, and this permission notice appear on  
* all such copies.  
*  
* 3. Licensee may not distribute this software or any derivative thereof  
* in any form without explicit written permission from:  
*  
* Professor Andrew A. Chien          achien@cs.uiuc.edu  
* University of Illinois  
* Department of Computer Science  
* 2215 Digital Computer Laboratory  
* 1304 West Springfield Avenue  
* Urbana, Illinois 61801  
* USA  
*  
* 4. Licensee may not use the name, logo, or any other symbol of the UI  
* nor the names of any of its employees nor any adaptation thereof in  
* advertising or publicity pertaining to the software without specific  
* prior written approval of the UI.  
*  
* 5. THE UI MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE  
* SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS  
* OR IMPLIED WARRANTY.  
*  
* 6. The UI shall not be liable for any damages suffered by Licensee from  
* the use of this software.  
*  
* 7. The software was developed under agreements between the UI and the
```

```

*   Federal Government which entitle the Government to certain rights.
*
*****
*
* Developed by: The CSAG Research Group
*               University of Illinois at Urbana-Champaign
*               Department of Computer Science
*               1304 W. Springfield Avenue
*               Urbana, IL    61801
*
* Copyright (C) 1995,1996,1997
* The University of Illinois Board of Trustees.
*   All Rights Reserved.
*
* Project Manager and Principal Investigator:
*   Andrew A. Chien <achien@cs.uiuc.edu>
*
*****/

/*****
* File   : bandwidth.c
* Author : Scott Pakin and Matt Buchanan
* Date   : 11 June 1997
* RCS    : $Id: bandwidth.c,v 1.1 1997/06/12 02:07:24 pakin Exp pakin $
*
* Syntax : bandwidth message-size num-sends
* Summary: FM 2.1 bandwidth test sender program measures bandwidth between
*          two processes. It sends num-sends messages of size message-size
*          bytes, in addition to NUM_WARMUP_MSGS warmup messages, from the
*          sender to the receiver. The receiver sends a message using the
*          recvrDone() handler which stops the timer. The sender then prints
*          out the results of the test.
*****/

#include "fm.h"
#include <stdio.h>
#include <stdlib.h>

#define NUM_WARMUP_MSGS 5

/* FM handlers */
ULONG fmhReceiver;
ULONG fmhRecvrAwake;
ULONG fmhRecvrDone;
ULONG fmhSenderDone;

/* Other variables */
ULONG msgCount;
BOOLEAN senderIsDone = FALSE;
ULONG numSends;
ULONG msgSize;

```

```

ULONGLONG *buffer;
BOOLEAN   recvrlsAwake = FALSE;
BOOLEAN   recvrlsDone = FALSE;
FM_stream *sendStream;
ULONG     t1, t2;
ULONG     tElapsed;
char      *programe;
const ULONG senderId = 0;
const ULONG recvrId = 1;

/*****
 * Handler sets the global flag recvrlsAwake
 */
#pragma FM_declare_handler
int recvrAwake (FM_stream *s, unsigned senderId)
{
#pragma FM_begin_decls
#pragma FM_end_decls

    recvrlsAwake = TRUE;

    return FM_CONTINUE;
} /* End recvrAwake() */

/*****
 * Handler sets the global flag recvrlsDone and stops the clock
 */
#pragma FM_declare_handler
int recvrDone (FM_stream *s, unsigned senderId)
{
#pragma FM_begin_decls
#pragma FM_end_decls

    t2 = microsecond_timer();
    recvrlsDone = TRUE;

    return FM_CONTINUE;
} /* End recvrDone() */

/*****
 * Handler increments the global msgCount and optionally touches each
 * byte of the message
 */
#pragma FM_declare_handler
int receiver (FM_stream *s, unsigned senderId)
{
#pragma FM_begin_decls
#pragma FM_end_decls

```



```

#ifdef TOUCHDATA
    if (msgSize)
        FM_receive (buffer, s, msgSize);
#endif

    msgCount++;
    return FM_CONTINUE;
} /* End receiver() */

/*****
 * Handler sets the global flag senderIsDone.
 */
#pragma FM_declare_handler
int senderDone (FM_stream *s, unsigned senderId)
{
    #pragma FM_begin_decls
    #pragma FM_end_decls

    senderIsDone = TRUE;

    return FM_CONTINUE;
} /* End recvrDone() */

/*****
 * Send data to the remote process
 */
void senddata(void)
{
    ULONG i;

    FM_reallocate_credit ();

    /* Wait until we know the receiver is scheduled */
    while (!recvrIsAwake)
        FM_extract (~0);

    /* Heat up the receiver's caches */
    for (i=0; i<NUM_WARMUP_MSGS; i++) {
        if (!(sendStream = FM_begin_message (recvrId, msgSize, fmhReceiver))) {
            fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
            FM_finalize();
            exit (1);
        }
        FM_send_piece (sendStream, buffer, msgSize);
        FM_end_message (sendStream);
    }

    /* Let the test begin! */

```

```

t1 = microsecond_timer();
for (i=0; i<numSends; i++) {
    if (!(sendStream = FM_begin_message (recvrId, msgSize, fmhReceiver))) {
        fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
        FM_finalize();
        exit (1);
    }
    if (msgSize)
        FM_send_piece (sendStream, buffer, msgSize);
    FM_end_message (sendStream);
}

/* Wait for acknowledgement of receipt from the receiver */
while (!recvrIsDone)
    FM_extract (~0);
free (buffer);

/* Say how well we did */
tElapsed = t2 - t1;
printf ("Start time   : %8lu us\n", t1);
printf ("Finish time  : %8lu us\n", t2);
printf ("Elapsed time  : %8lu us\n", tElapsed);
printf ("Message size  : %8lu bytes\n", msgSize);
printf ("No. messages  : %8lu\n", numSends);
printf ("Avg. latency   : %8.2f us/message\n",
        ((double) tElapsed) / numSends);
printf ("Avg. bandwidth: %8.2f MB/s\t\t<==\n",
        (msgSize * numSends * 1000000.0) / (tElapsed * 1048576.0));

#ifdef TOUCHDATA
/* If the receiver is not touching the data, we mustn't exit without */
/* flushing the entire previous message from this node. Because of FM's */
/* in-order delivery, sending a 0-byte message ensures this */
if (!(sendStream = FM_begin_message (recvrId, 0, fmhSenderDone))) {
    fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
    FM_finalize();
    exit (1);
}
/* No need to call FM_send_piece() for zero bytes */
FM_end_message (sendStream);
#endif
}

/*****
* Receive data from the remote process
*/
void receivedata(void)
{
    /* Tell the sender we're awake now */
    if (!(sendStream = FM_begin_message (senderId, 0, fmhRecvrAwoke))) {

```

```

    fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
    FM_finalize();
    exit (1);
}
/* No need to call FM_send_piece() for a 0-byte message */
FM_end_message (sendStream);

FM_reallocate_credit ();

/* Begin our part of the test -- receive warmup messages + real messages */
numSends += NUM_WARMUP_MSGS;
while (msgCount < numSends)
    FM_extract (~0);

/* We're through playing now */
if (!(sendStream = FM_begin_message (senderId, 0, fmhRecvrDone))) {
    fprintf (stderr, "%s: Couldn't get stream for sending\n", progname);
    FM_finalize();
    exit (1);
}
/* No need to call FM_send_piece() for a 0-byte message */
FM_end_message (sendStream);
free (buffer);
printf ("%s: Received %lu messages.\n", progname, numSends);

#ifdef TOUCHDATA
/* Wait until we've really received all bytes in the message */
while (!senderIsDone)
    FM_extract (~0);
#endif

#ifdef _WIN32
/*****
* Try to shut FM down cleanly
*/
BOOL shutdownFM (DWORD event)
{
    FM_finalize();
    return FALSE;
}
#endif

/*****
* main()
*/
void main (int argc, char *argv[])
{
    char *key;          /* Program identifier */

```

```

/* Parse command line arguments */
programe = argv[0];
if (argc != 4) {
    fprintf (stderr, "Usage: %s message-size num-sends program-key\n", programe);
    exit (1);
}
msgSize = atol (argv[1]);
numSends = atol (argv[2]);
key = argv[3];

/* Initialize Fast Messages */
FM_set_parameter (FM_KEY_NAME, key);
FM_initialize();
fmhRecvAwake = FM_register_handler (FM_ANY_ID, recvrAwake);
fmhRecvDone = FM_register_handler (FM_ANY_ID, recvrDone);
fmhReceiver = FM_register_handler (FM_ANY_ID, receiver);
fmhSenderDone = FM_register_handler (FM_ANY_ID, senderDone);

#ifdef _WIN32
/* Optional, but polite to other processes */
SetConsoleCtrlHandler((PHANDLER_ROUTINE)shutdownFM,TRUE);
#endif

/* Initialize the bandwidth test */
buffer = (ULONGLONG *) malloc (msgSize);
if (!buffer) {
    fprintf (stderr, "%s: Memory allocation failed.\n", programe);
    FM_finalize();
    exit (1);
}

/* Call either sending or receiving routine, based on our node number */
switch (FM_nodeid) {
    case 0: /* Sender */
        senddata();
        break;
    case 1: /* Receiver */
        receivedata();
        break;
    default: /* Error */
        fprintf (stderr, "%s: The bandwidth test already contains two running processes. Aborting.\n",
programe);
        FM_finalize();
        exit (1);
}
/* Finish up */
FM_finalize();
exit (0);
}

```

Glossário

AC (*address consultant*) - consultor de endereços utilizado pelo Mosaic, Atomic e é parte integrante do MCP na Myrinet.

ANSI (*American National Standard*) - órgão de padronização americano.

API - (*Application-Programming Interface*) – Interface para programação de aplicação.

ARP (*Address Resolution Protocol*) – Protocolo de resolução de endereços para redes TCP/IP.

ARPA (*Advanced Research Projects Agency*) – Agência de pesquisa do governo dos Estados Unidos.

ATM (*Asynchronous Transfer Mode*) - Modo de Transferência Assíncrono. Padrão da ITU destinado a altas taxas de transmissão em WANs com estrutura ISDN *broadband*, desenvolvido e baseado no princípio de multiplexação por divisão de tempo assíncrono. A tecnologia também é destinada para redes locais. Ela é baseada no transporte de pequenas unidades de dados de tamanho fixo (53 bytes) que recebem o nome de célula.

Backbone - em redes de comunicação, representa a coluna dorsal ou principal meio de transmissão.

Bandwidth (largura de banda) – define o tamanho da largura de banda de uma rede.

Bit - menor unidade de informação na computação.

Broadcasting - em telecomunicações, é a propagação de um fluxo de dados de uma determinada fonte para todos os destinos.

Buffers - espaço na memória onde se efetua armazenamento temporário.

Cache - espaço de memória de alta velocidade na hierarquia de memória principal de um processador.

Cached - estado da informação que tem uma cópia na memória cache de um sistema.

CFG - (*Generic Flow Control*) – Controle Genérico de Fluxo: previsto apenas na célula UNI, é utilizado para controle de fluxo de células ATM.

Checksum - mecanismo para checagem e validação das informações nos pacotes.

Chip - componente eletrônico composto por vários sub-componentes integrados em uma mesma pastilha semicondutora.

Clock - Gerador básico dos ciclos de operação em um processador.

Cluster - grupo de computadores acoplados.

CLP - (*Cell Loss Priority*)- indica células de mais baixa prioridade.

Comutação de Pacotes - técnica de multiplexação por divisão de tempo usada em comunicação de dados onde o fluxo de bits é dividido em pequenas unidades de dados.

Essas pequenas unidades são conhecidas como pacotes, frames ou células.

CPU (*Central Processing Unit*) - local onde ocorre o processamento dos dados no computador.

CRC (*Cyclic Redundance Checking*) – mecanismo que efetua a checagem de dados na transmissão.

Crossbar – rede de chaveamento matricial em duas dimensões onde é possível cada linha ser ligada a cada coluna não comprometidas com outras conexões.

CSR (*Control and Status Register*) - define um espaço comum de endereço para cada nó contendo registradores de controle e status em sistemas de memória compartilhada.

Deadlock - travamento que ocorre quando dois ou mais processos dependem um do outro para efetuar alguma ação.

DMA - (*Direct Memory Access*) – Acesso Direto a Memória.

Fabric - malha de interconexão entre as portas de uma *switch*. Pode incluir também uma estrutura de elementos de comutação e de armazenamento.

FDDI (*Fiber Distributed Data Interface*) - Interface de Dados Distribuídos por Fibra, padrão adotado pela ANSI para redes de fibra ótica que operam a 100 Mbits/s.

FM (*Fast Messages*) - biblioteca de passagem de mensagem desenvolvida na universidade americana de Illinois, efetuando comunicação mais rápida para redes de alta velocidade.

FTP (*File Transfer Protocol*) – protocolo utilizado para efetuar transferências de arquivos entre computadores.

Gbit/s - milhões de bits por segundo.

Hardware - o que se refere à parte física da máquina.

HEC (*Header Error Control*) - contém 8 bits para controle de erro relativo apenas ao cabeçalho na camada ATM.

HIPPI (*High Performance Parallel Interface*) - Interface Paralela de alto desempenho em redes para conexão de supercomputadores.

Home Banking - Banco em Casa – Serviço oferecido pelos bancos via Internet.

Host - anfitrião – máquina hospedeira.

HPVM (*High Performance Virtual Machine*) - ambiente para desenvolvimento de programação paralela e também de aplicações, disponibilizando bibliotecas de passagem de mensagem, linguagens de programação, etc.

HPF (*High Performance Fortran*) - linguagem de programação paralela.

ICMP - Internet Control Message Protocol – um dos protocolos TCP/IP.

E/S (Entrada/Saída) – dispositivo de Entrada/Saída.

IP (*Internet Protocol*) – protocolo de camada de rede utilizado na Internet.

ISDN (*Integrated Services Digital Network*) padrão de rede digital de comunicação de dados

ISO/OSI (*International Organization for Standardization/Open Systems interconnection*) padrão de sistemas que obedecem às condições que qualificam um sistema como sistema aberto.

ITU (*International Telecommunication Union*) – Órgão internacional de padronização na área de telecomunicações, mantida pela ONU (Organização das Nações Unidas).

LAN (*Local Area Network*) - Redes de comunicação que interligam computadores em um mesmo edifício ou em um conjunto de edifícios próximos.

Mainframes - termo que designa computadores de grande porte.

MAN (*Metropolitan Area Network*) - Redes que interligam computadores em uma cidade ou região.

Gbit/s - milhares de bits por segundo.

MCP (*Myrinet Control Program*) - programa que controla e fornece acesso à rede Myrinet através da interface do *host*.

MIMD (*Multiple Instruction Multiple Data*) - Múltiplas instruções e múltiplos fluxos de dados, termo utilizado na classificação de arquitetura de computadores por Flynn.

MISD (*Multiple Instruction, Single Data*) - múltiplos fluxos de instruções para um único fluxos de dados, alguns autores como Duncan, Tanenbaum e Navax afirmam que não existem computadores físicos pertencentes à essa classe.

MOSIX - (*Multicomputer Operating System do UNIX*) - projeto que melhora o *kernel* do UNIX com balanceamento de carga dinâmico e migração de processos preemptiva.

MPI (*Message passing Interface*) - padrão para bibliotecas de passagem de mensagem definido pelo Fórum (MPIF).

MPP (*Massively Parallel Processors*) - Processadores Maciçamente Paralelos: computadores cuja arquitetura é composta de grandes quantidades de processadores.

Multicomputador - é uma arquitetura de computador maciçamente paralelo consistindo de uma coleção de nós de computadores, cada qual com sua memória, conectados por uma rede de passagem de mensagens.

NOW (*Network of workstation*) - projeto da universidade de Berkeley abrangendo redes de comunicação de alta velocidade, sistemas operacionais distribuídos, sistemas de arquivos escaláveis, www e ambientes de programação.

Payload - representa a carga útil num quadro de informação.

PT - *Payload Type* - indica o tipo de informação contida em uma célula ATM.

Router/Roteador - Dispositivo para interconexão de redes que opera na camada de rede do modelo de referência ISO/OSI. Ele toma decisões sobre os caminhos a serem seguidos pelo tráfego na rede e encaminha o tráfego de uma rede a outra com base em parâmetros associados à camada de rede.

Routing/Roteamento - operação executada por dispositivos em uma rede de comunicação de dados visando encontrar uma rota que permita levar a informação pretendida da origem ao seu destino.

SAN (*System-Area Network*) - sistema de rede onde agrupa-se vários computadores em um mesmo gabinete.

Scanner - periférico utilizado para capturar imagens, armazenando-as em formato digital.

SCI (*Scalable Coherent Interface*) - Interface Coerente Escalável é um padrão para compartilhamento de dados entre vários processadores e caches mantendo-se sua coerência.

SDH (*Synchronous Digital Hierarchy*) - Padrão europeu para telefonia digital.

Setup - fase inicial de uma conexão ATM, na qual são estabelecidos os parâmetros de tráfego e a qualidade de serviço da conexão.

SISD (*Single Instruction, Single Data*) - fluxos de instruções e dados únicos, classe ao qual corresponde os computadores sequenciais.

SIMD (*Single Instruction, Multiple Data*) - fluxo de instrução único e múltiplos fluxos de

dados - classe ao qual correspondem os processadores matriciais.

Slot – em ATM é um quadro que possui 53 bytes (5 bytes do cabeçalho e 48 bytes de dados).

SMTP - (*Simple mail Transfer protocol*) – protocolo TCP/IP para transferência de correio eletrônico.

Switches - Dispositivos em uma rede de comunicações destinados à comutação de circuitos.

Switching - Chaveamento, ação que ocorre nos switches.

TCP - (*Transmission Control Protocol*) – protocolo TCP/IP.

TDM - (*Time Division Multiplexing*) – multiplexação por Divisão de Tempo.

UDP (*User Datagram Protocol*) – protocolo TCP/IP.

UNI - *User Network Interface*

VC (*Virtual Channel*) – Canal Virtual.

VCI (*Virtual Channel Identifier*) – corresponde à parte menos significativa do identificador de conexão, campo do cabeçalho da célula ATM.

VITA (*VMEbus International Trade Association*) - uma organização de padronização

VPI (*Virtual Path Identifier*)– é composto de 8 bits no cabeçalho da célula UNI e 12 bits no cabeçalho da célula NNI. Sua função é servir como a parte mais significativa do código que identifica a conexão.

WAN (*Wide Area Network*) - rede que interligam equipamentos a longas distâncias; podem ter cobertura nacional ou regional.