Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Engenharia Elétrica
Área de Concentração: Sistemas de Informação

Crux:
Ambiente Multicomputador Configurável por Demanda

Thadeu Botteri Corso

Tese Submetida à Universidade Federal de Santa Catarina
para Obtenção do Grau de Doutor em Engenharia Elétrica,
Área de Concentração: Sistemas de Informação

Orientador: Prof. Joni da Silva Fraga

Florianópolis, março de 1999.

Crux: Ambiente Multicomputador Configurável por Demanda

Thadeu Botteri Corso

Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica Área de Concentração em Sistemas de Informação em que foi realizado o trabalho, e aprovada em sua forma final pelo Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Joni da Silva Fraga, Dr.
Orientador

Ildemar Cassana Decker, Dr.
Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

Banca Examindora:

Joni da Silva Fraga, Dr.
Presidente

Júlio Salek Aude, Dr.

Simão Sirineo Toscani, Dr.

Jean-Marie Farines, Dr.

Paulo José de Freitas Filho, Dr.

SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS UNIVERSITÁRIO - TRINDADE - CAIXA POSTAL 476
CEP. 88049 - FLORIANÓPOLIS - SANTA CATARINA
TEL. (0482) - 31.9000 - TELEX: 0482 260
CTC/EEL/CCPGEEL

ATA DA DEFESA DE TESE DE DOUTORADO

ATA DA DEFESA DE TESE PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS- (D.Sc.)

ESPECIALIDADE : ENGENHARIA ELÉTRICA
ÁREA DE CONCENTRAÇÃO : Sistemas de Informação
REALIZADA EM : 12 DE MARÇO DE 1999
CANDIDATO (A) ... THADEU BOTTERI CORSO
BANCA EXAMINADORA :
..... JONI DA SILVA FRAGA ..... (ORIENTADOR)
.....
..... JÚLIO SACEK AUDE (UFRJ)
..... SIMÃO SIRINEO TOSCANI (UFRGS)
..... JEAN-MARIE FARINES
..... PAULO JOSÉ DE FREITAS FILHO
.....
.....

TÍTULO DA TESE : CRUX : Ambiente Multicomputador Configurável por Demanda
.....
.....
.....
.....

LOCAL : Anfiteatro do CTC
HORÁRIO DE INÍCIO : 10h10

Em sessão pública, após exposição de cerca de 50 (cinquenta) .....minutos, o candidato (a) foi arguido (a) oralmente, pelos membros da banca ... tendo demonstrado suficiência de conhecimento e capacidade de sistematização no tema de sua tese, tendo sido .....aprovado (a) ..... por unanimidade, desde que satisfeitos os requisitos constante da folha 25 do livro de requisitos de tese de doutorado.
.....

Na forma regulamentar foi lavrada a presenta ata que é assinada pelos membros da banca, e pelo (a) candidato (a).

Florianópolis, 12 de março de 1999

Orientador .....
.....
.....
.....
.....
.....
.....
.....
.....

Candidato ...

SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS UNIVERSITÁRIO - TRINDADE - CAIXA POSTAL 476
CEP. 88049 - FLORIANÓPOLIS - SANTA CATARINA
TEL. (0482) - 31.9000 - TELEX: 0482 260
CTC/EEL/CCPGEEL

FOLHA DE MODIFICAÇÃO EM TESE

Modificações exigidas na Tese de Doutorado de THADEU BOTTERI CORSO
de número 57, em 12/3/99

As modificações foram as seguintes :
correções segundo recomendação da banca

O prazo para o cumprimento é de 90 (dias),
sendo responsável o Prof.: JONAS DA SILVA FRAGA
Presidente da Banca : [signature]
Candidato: [signature]

Atesto que as alterações exigidas foram cumpridas.

Florianópolis, 15 de MARÇO de 1999

Prof. Responsável : [signature]

Resumo

O objetivo deste trabalho é o de propor um ambiente para execução de programas paralelos organizados como redes de processos que se comunicam exclusivamente através de trocas de mensagens em um multicomputador com rede de interconexão dinamicamente configurável por demanda dos programas paralelos. Ambientes multicomputadores envolvem elementos de linguagem de programação paralela, de arquitetura de multicomputadores e de suporte de execução que foram objeto de estudos preliminares. Esses estudos iluminaram a concepção de um ambiente articulado em torno de um multicomputador reconfigurável e seu mecanismo de conexão dinâmica de canais físicos, conjuntamente responsáveis por uma drástica redução da complexidade dos problemas encontrados — principalmente os relativos ao trânsito de mensagens — se comparados aos problemas descritos nos ambientes multicomputadores citados na literatura. O ambiente proposto foi extensivamente submetido a ensaios comprobatórios de sua simplicidade de operação, de sua flexibilidade de adaptação a exigências distintas e de seu superior desempenho em resposta às aplicações previstas.

Abstract

The objective of this work is to propose an environment for the execution of parallel programs expressed as networks of exchanging message processes for a multicomputer with a demand-driven configurable interconnection network. Elements of parallel programming languages, multicomputer architecture and run-time execution support have been firstly studied. After these studies, a complete environment based on a configurable multicomputer and its physical channel's dynamic connection mechanism has been proposed that, in conjunction, brought an enormous simplicity to the communication mechanism if compared to those found in the literature. The proposed environment has been extensively subjected to probation tests concerning its simplicity of operation, its flexibility of adaptation to distinct requirements and its good performance in response to target applications.

Sumário

1 Introdução 1

1.1 Motivações 1

1.2 Objetivos 2

1.3 Adequação ao Curso 3

1.4 Organização do Texto 4

2 Ambientes Multicomputadores 6

2.1 Linguagens Paralelas 6
2.1.1 Classificação 6
2.1.2 Linguagens Apoiadas em Troca de Mensagens 9
2.1.2.1 Processos 10
2.1.2.2 Canais lógicos 10
2.1.2.3 Mensagens 12
2.1.2.4 Disciplinas de Comunicação 13
2.1.3 Exemplos 17

2.2 Computadores Paralelos 18
2.2.1 Classificação 18
2.1.2 Redes de Interconexão de Multicomputadores 23
2.1.1.1 Redes Estáticas 23
2.1.1.2 Redes Dinâmicas 27
2.1.3 Exemplos 31
2.1.3.1 Paragon 32
2.1.3.2 Supernode 34

2.3 Suporte de Execução 37
2.3.1 Mecanismos de Comunicação 38
2.3.1.1 Redes Estáticas 38
2.3.1.2 Redes Dinâmicas 40
2.3.2 Sistemas Operacionais 43
2.3.2.1 Estrutura 43
2.3.2.2 Fluxos de Execução 46

2.4 Conclusões 46

3 Ambiente Multicomputador Crux 48

3.1 Motivações 48
3.1.1 Linguagens e Arquiteturas 48
3.1.2 Adaptação das Redes Físicas às Redes Lógicas 50
3.1.3 Exploração do Paralelismo Físico 51
3.1.4 Mecanismos de Comunicação 52

3.1.5 Articulação entre Componentes 58
3.2 Princípios de Projeto 59
3.3 Objetivos 60
3.3.1 Delimitações Práticas Impostas à Proposta 60
3.3.2 Lista de Objetivos 62
3.4 Multicomputador Crux 63
3.4.1 Arquitetura 63
3.4.2 Conexão Dinâmica de Canais Físicos 64
3.4.3 Avaliação Qualitativa 65
3.5 Conclusões 68
4 Avaliação do Ambiente Crux 69
4.1 Simulação 69
4.2 Avaliação de Desempenho do Crux 73
4.2.1 Hipóteses Gerais 73
4.2.2 Avaliação Comparativa 75
4.2.2.1 Comunicações Generalizadas 75
4.2.2.2 Comunicações Seletivas 78
4.2.3 Avaliação Isolada 81
4.3 Considerações Gerais sobre a Avaliação 84
4.4 Conclusões 84
5 Aplicações para o Ambiente Crux 86
5.1 Sistema Operacional ACrux 86
5.1.1 Visão Geral 86
5.1.2 Camadas do Sistema ACrux 89
5.1.2.1 Camada do Sistema 90
5.1.2.2 Camada do Núcleo 91
5.1.3 Exemplo 92
5.1.4 Considerações sobre o Sistema Operacional 93
5.2 Interpretador para a Linguagem Superpascal 94
5.2.1 Linguagem Superpascal 94
5.2.2 Interpretador Paralelo 95
5.2.3 Considerações sobre o Interpretador Paralelo 98
5.3 Implementações 98
5.3.1 Sistema Operacional ACrux no Supernode 99
5.3.1.1 Componentes da Versão Preliminar do Sistema 99
5.3.1.2 Implementação das Interfaces 100
5.3.1.3 Operadores Regulares 103
5.3.1.4 Operadores Especiais 103
5.3.1.5 Sinais 105
5.3.1.6 Inicialização do Sistema 106
5.3.1.7 Desenvolvimento do Experimento 106
5.3.2 Projeto Nó // 106
5.3.2.1 Multicomputador Nó // 107
5.3.2.2 Simulador do Multicomputador Nó // 107
5.3.2.3 Sistema Operacional ACrux no simulador do Nó // 108
5.3.2.4 Interpretador Paralelo para a Linguagem Superpascal no Nó // 109
5.4 Conclusões 110

6 Conclusões **111**

6.1 Revisão dos Objetivos 111

6.2 Resumo do Trabalho Realizado 112

6.3 Contribuições 113

6.4 Perspectivas Futuras 114

A Modelos para Simulação **116**

A.1 Crux 116

A.2 Torus 118

A.3 Crossbar 120

A.4 Avaliação Comparativa Complementar 122

Bibliografia **125**

1 Introdução

Neste capítulo, são inicialmente identificadas as motivações e os objetivos gerais do trabalho exposto neste texto. É mostrada também a adequação do tema escolhido às linhas de pesquisa do Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina. Por fim, é apresentado o plano do texto desta tese.

1.1 Motivações

Muitos sistemas computacionais de uso geral compostos de múltiplos processadores, atualmente disponíveis, podem ser classificados em três grandes grupos em função do grau de integração dos seus componentes e da velocidade das interações possíveis entre eles [AUS91]: multiprocessadores, multicomputadores e redes locais de computadores.

Os *multiprocessadores* (ou máquinas paralelas com memória compartilhada) são computadores compostos de processadores que acessam memórias globais através de redes de interconexão dinâmicas. Os *multicomputadores* (ou máquinas paralelas com memória distribuída) são computadores compostos de nós largamente autônomos, cada um dos quais dispondo de um processador, de uma memória privativa e de canais de comunicação, ligados através de redes de interconexão estáticas ou dinâmicas formadas por múltiplos canais de comunicação. As *redes locais de computadores* são compostas de nós constituídos de computadores completos ligados através de redes de interconexão formadas de canais de comunicação compartilhados.

Na classificação aqui adotada, as redes de computadores não são consideradas máquinas paralelas. Os critérios que permitem fazer essa distinção de forma precisa são apresentados adiante em 2.2.1. Entretanto, outros critérios conduzem a outras classificações que incluem as redes de computadores como uma subclasse dos multicomputadores [BEL94].

Componentes desses três grupos podem formar um sistema integrado no qual os nós de uma rede local de computadores são representados por multiprocessadores e multicomputadores além de estações de trabalho monoprocessadoras convencionais. Estima-se que nós formados de multicomputadores podem difundir-se em sistemas integrados desse tipo por fornecerem grande potência de cálculo a custos moderados.

A computação paralela deve responder a critérios de maximização de desempenho de aplicações específicas e de disponibilização de serviços nos sistemas de computação. O princípio subjacente à programação paralela é o da composição de sistemas complexos através de processos que evoluam simultaneamente e cooperem entre si na consecução de um objetivo

comum [AKL89]. Os dois modelos gerais de computação paralela são o *compartilhamento de memória* e a *troca de mensagens* [BAI88].

Esses dois modelos não são mutuamente excludentes, podendo ser usados conjuntamente em um mesmo ambiente. Entretanto, no que se refere à exploração do paralelismo real, os multiprocessadores induzem à cooperação por compartilhamento de memória e os multicomputadores impõem a cooperação por troca de mensagens.

O modelo de troca de mensagens vem sendo considerado um bom estilo de programação paralela e tem dado origem a várias linguagens. Programas paralelos que se enquadram nesse modelo geram *redes de processos comunicantes* — redes formadas por processos que cooperam por troca de mensagens que fluem entre eles através de canais de comunicação.

Apesar da semelhança conceitual entre os componentes lógicos das redes de processos comunicantes e os componentes físicos dos multicomputadores, na prática, constata-se geralmente uma grande discrepância entre as topologias das redes lógicas e as das redes físicas. Essa discrepância, que tem sido geralmente aceita como inevitável, vem sendo superada por serviços de comunicação, através de métodos inerentemente complexos. (Além dos serviços de comunicação, a execução de um programa em um multicomputador depende ainda de outros serviços habituais de um sistema operacional, como a gerência de processos).

Os problemas iniciais que precisam de qualquer forma ser resolvidos para se ter acesso ao paralelismo nos multicomputadores referem-se às abstrações de processos e de trocas de mensagens. Sendo inevitáveis, eles constituem os *problemas básicos* visados por esta proposta. Estima-se que uma vez resolvidos de forma conveniente, todas as outras abstrações propostas pelos mais diversos modelos de programação paralela possam ser convenientemente suportadas.

O tema de fundo deste trabalho é o da exploração do paralelismo real provido pelos multicomputadores. Sua motivação fundamental parte da visão dessas máquinas como um meio natural para a expressão física das redes lógicas de processos comunicantes. Por isso, ela pretende seguir uma trilha alternativa às das soluções conhecidas neste domínio, rejeitando a discrepância constatada entre redes físicas e lógicas e buscando a aderência dos multicomputadores às redes de processos comunicantes, sem sacrificar a simplicidade, a flexibilidade e a eficiência do serviço de comunicação.

1.2 Objetivos

De uma forma geral, a execução de um programa em um computador depende de um *suporte de execução* interposto entre eles que sirva de interface para resolver as discrepâncias entre o modelo da linguagem de programação e a arquitetura do computador (figura 1.1). Ele costuma ser tão mais elaborado quanto maior for a diferença entre o modelo da linguagem e a arquitetura do computador.

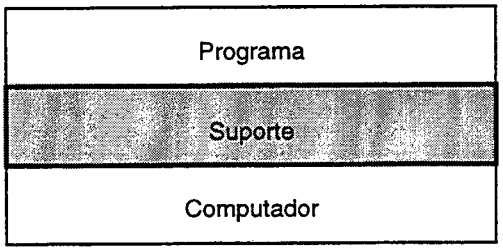


Figura 1.1: Suporte de execução.

A interface representada pelo suporte de execução é criada conjuntamente pelo sistema operacional e pelo ambiente de execução construído pelo compilador da linguagem de programação. Embora existam muitas questões abertas sobre a melhor maneira de combinar esses componentes [BAL89], o objetivo final é sempre o de fornecer ao programador a ilusão de que ele dispõe de uma máquina que segue a especificação da linguagem.

Considerando-se implementações em multicomputadores, é de se esperar que a interface de uma linguagem que segue o modelo de compartilhamento de memória seja mais complexa do que a de outra que segue o modelo de troca explícita de mensagens.

Este trabalho pretende envolver elementos das áreas de Arquitetura de Computadores, Sistemas Operacionais e Linguagens de Programação, *sustentando-se* em elementos consolidados e *introduzindo* novos, para compor um ambiente completo para a execução de programas paralelos. Ele visa essencialmente propor um sistema integrado composto por um modelo de multicomputador reconfigurável, com um mecanismo de comunicação para permitir a expressão de programas paralelos como redes de processos comunicantes. Ele pretende ainda propor a composição do próprio sistema operacional como uma rede de processos comunicantes. Esses objetivos são ampliados e precisados em 3.3.

1.3 Adequação ao Curso

Entre os interesses do Laboratório de Controle e Microinformática (LCMI) do Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC), estão os sistemas de computação distribuídos. Especificamente, temas relacionados com métodos e ferramentas de desenvolvimento, linguagens de programação e sistemas operacionais têm sido objeto de projetos de pesquisa, teses de doutorado, dissertações de mestrado e trabalhos de graduação.

O tema desta proposta de tese está em perfeita sintonia com os objetivos deste curso ao comprometer-se com a concepção de um ambiente integrando aspectos de arquiteturas de multicomputadores e serviços de comunicação de sistemas operacionais para suportar a execução de programas paralelos em ambientes fisicamente distribuídos.

1.4 Organização do Texto

O capítulo 2, que se reporta aos temas estudados para amparar este trabalho, é subdividido em três subcapítulos, cada um dos quais dedicado a uma das camadas de um ambiente para execução de programas paralelos identificadas na figura 1.1. Esse capítulo é preponderantemente descritivo, sendo a maior parte dos julgamentos de valores sobre os elementos nele apresentados postergados até os capítulos seguintes.

O subcapítulo 2.1 introduz as linguagens paralelas apoiadas em troca de mensagens. Uma classificação geral das linguagens de programação é proposta. Os conceitos fundamentais de programação paralela são descritos em termos de processos, canais lógicos de comunicação e mensagens. Como exemplos, cinco linguagens paralelas clássicas apoiadas em troca de mensagens são brevemente caracterizadas.

O subcapítulo 2.2 apresenta os multicomputadores. Uma classificação geral procura situar os multicomputadores dentro do espectro mais amplo dos computadores paralelos. São descritas as redes de interconexão dos multicomputadores em função da grande influência que esses componentes exercem sobre o desempenho global dessas máquinas. Para caracterizar o estado da arte nesse domínio, dois exemplos de multicomputadores modernos são descritos.

Os subcapítulos 2.1 e 2.2 são propositadamente autônomos, podendo ser lidos em qualquer ordem.

O subcapítulo 2.3 trata do suporte necessário para executar programas apoiados em troca de mensagens em multicomputadores. Aspectos essenciais do serviço de comunicação através das redes de interconexão dessas máquinas são descritos. Os sistemas operacionais distribuídos são apresentados, principalmente do ponto de vista de sua organização interna como um programa paralelo. Dois sistemas operacionais distribuídos são brevemente caracterizados, indicando uma possível organização de sistema operacional para multicomputadores.

O capítulo 3 contém os componentes básicos da proposta de tese. As motivações e os objetivos específicos do trabalho são precisados à luz do capítulo 2. Os componentes básicos da proposta, consistindo do multicomputador Crux e de seu mecanismo de comunicação com conexão dinâmica de canais físicos são introduzidos. Várias características são exploradas em uma avaliação qualitativa do ambiente proposto.

O capítulo 4 envolve a avaliação do desempenho dos componentes básicos do multicomputador proposto no capítulo 3. Usando técnicas de simulação discreta, esse ambiente multicomputador é comparado com outros citados na literatura. Além disso, ele também é avaliado isoladamente em condições potencialmente críticas para o seu desempenho.

O capítulo 5 contém os componentes da proposta de tese que, apoiando-se nos componentes básicos descritos no capítulo 3, completam o ambiente multicomputador para a execução de programas paralelos. Nele são descritas principalmente a organização do sistema operacional ACrux, a estrutura de um interpretador para uma linguagem paralela e a implementação de uma versão simplificada do sistema ACrux em um multicomputador real. Esses componentes evidenciam a flexibilidade e a simplicidade do ambiente básico.

O Capítulo 6 sintetiza as conclusões do trabalho através da verificação dos resultados obtidos, das contribuições da proposta e as perspectivas de projetos futuros.

O Apêndice apresenta os modelos para simulação referidos no capítulo 4, usados para a avaliação de desempenho do ambiente multicomputador Crux.

2 Ambientes Multicomputadores

Este capítulo está dividido em três subcapítulos, cada um dos quais dedicado a uma das camadas de um ambiente para execução de programas paralelos identificadas na figura 1.1. O subcapítulo 2.1 introduz as linguagens paralelas apoiadas em troca de mensagens dentro de uma classificação geral das linguagens paralelas e descreve os conceitos fundamentais de programação paralela em termos de processos, canais lógicos de comunicação e mensagens. O subcapítulo 2.2 apresenta os multicomputadores dentro de uma classificação geral dos computadores paralelos e caracteriza as redes de interconexão dos multicomputadores. O subcapítulo 2.3 trata do suporte necessário para executar programas apoiados em troca de mensagens em multicomputadores relativos ao serviço de comunicação e aos demais serviços dos sistemas operacionais distribuídos. Cada subcapítulo inclui exemplos associados aos temas nele tratados.

2.1 Linguagens Paralelas

Neste subcapítulo, são introduzidas as linguagens paralelas. Inicialmente, é apresentada uma classificação geral para as linguagens imperativas para situar as linguagens apoiadas em troca de mensagens no contexto das linguagens paralelas. Em seguida, são examinadas as propriedades das entidades lógicas fundamentais da programação paralela. Cinco exemplos clássicos de linguagens paralelas apoiadas em troca de mensagens são brevemente comparados.

2.1.1 Classificação

Vários paradigmas de programação têm sido propostos ao longo dos anos dando origem a uma grande variedade de linguagens. Entre elas, as linguagens imperativas tradicionais tais como Algol, Pascal e C são as que mais claramente exibem o modo de operação dos processadores convencionais. A implementação de outras linguagens tais como Lisp, Prolog e Smalltalk nesses processadores depende de ambientes de execução (interpretadores de máquinas virtuais) escritos em linguagens imperativas [BAL89]. Nesse sentido, pode-se dizer que essas linguagens são de nível superior àquelas.

Programas paralelos são compostos de atividades que evoluem simultaneamente e interagem entre si [AND91]. As entidades lógicas fundamentais do paralelismo são os *processos* para representar as atividades e os *mecanismos de comunicação* para permitir as

interações entre eles. Outras entidades de mais alto nível podem ser consideradas. Entretanto, elas devem ser desenvolvidas em processadores convencionais sobre essas entidades fundamentais.

A simultaneidade na evolução das atividades de um programa paralelo implica na atribuição de dispositivos materiais exclusivos para a execução de cada processo componente. Entretanto, os conceitos tratados neste capítulo referem-se ao paralelismo conceitual que pode tanto ser físico — quando cada processo dispõe de um processador real — quanto lógico — quando cada processo dispõe de um processador virtual — para sua execução. Referências à implementação são evitadas, exceto quando os conceitos emitidos se prestarem a confusões.

Os primeiros esforços para a programação paralela valeram-se de procedimentos de biblioteca chamados a partir de programas escritos em linguagens imperativas que permitiam acessar os recursos de sistemas operacionais para controlar o paralelismo [BAL89]. Evoluíram daí linguagens que incorporam construções sintáticas específicas para explorar explicitamente as entidades do paralelismo. (A expressão explícita do paralelismo concorre com a alternativa de extração do paralelismo implícito por compiladores que analisam as dependências lógicas entre os dados e geram código paralelo à partir de programas escritos em linguagens convencionais. Nesse caso, transfere-se do programador de aplicações para o programador do compilador as tarefas de lidar explicitamente com as entidades do paralelismo).

Diversos autores têm estabelecido taxonomias para as linguagens de programação [BAL89] [TRE88]. Diante da variedade dos modelos existentes e dos critérios adotados, resultam classificações diferentes. A classificação apresentada a seguir não pretende ser exaustiva: seu objetivo é antes o de situar as linguagens paralelas apoiadas em troca de mensagens no âmbito das linguagens de programação imperativas. Da mesma forma, a terminologia adotada não pretende ser universal: seu objetivo é apenas o de nomear de forma consistente os conceitos usados posteriormente no desenvolvimento deste trabalho.

A árvore da figura 2.1 representa uma possível classificação hierárquica geral para as linguagens de programação imperativas. Tendo em vista que a utilização da classificação apresentada é dirigida a aspectos da programação paralela, a árvore representada na figura foi podada nos ramos que divergem desse objetivo específico.

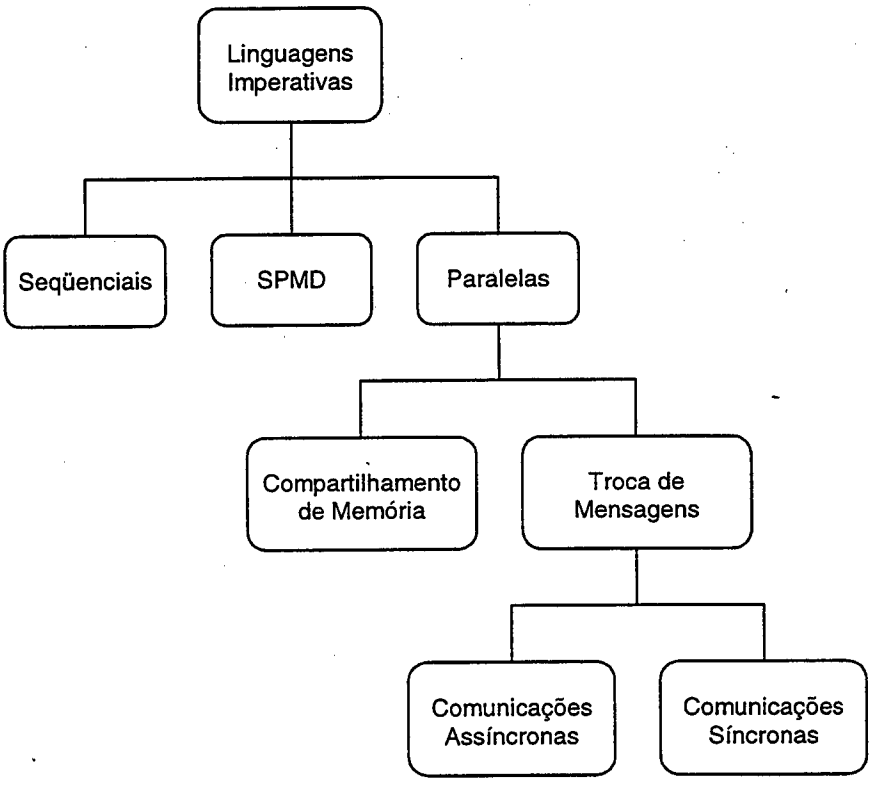


Figura 2.1: Uma classificação geral para as linguagens de programação imperativas.

Distinguindo o número de fluxos de instruções presentes simultaneamente no programa em único ou múltiplo, obtém-se duas categorias distintas de linguagens. Assim, no primeiro nível, à partir da raiz, aparecem as linguagens seqüenciais e as linguagens paralelas. Nas linguagens seqüenciais, apenas um fluxo de instruções determina a ordem de aplicação das operações sobre dados privativos. As linguagens paralelas permitem a especificação de vários fluxos de instrução simultâneos, cada qual determinando a ordem de aplicação das operações sobre dados privativos de cada fluxo ou compartilhado entre eles.

A classificação desse nível pode ser completada pelas linguagens SPMD (*Single Program Multiple Data*) que apresentam um tipo especial de paralelismo no qual um mesmo fluxo de instruções é aplicado a mais de um fluxo de dados. Normalmente elas são extensões de linguagens convencionais como Fortran, por exemplo, cuja aplicação principal envolve cálculos matriciais.

No segundo nível da classificação hierárquica apresentada na figura 2.1, as linguagens paralelas são diferenciadas de acordo com a capacidade de acesso à memória em linguagens apoiadas em compartilhamento de memória ou apoiadas em troca de mensagens. Os modelos de interação dessas linguagens são os seguintes [KAI89]:

- *Compartilhamento de Memória* - Os processos de um programa paralelo interagem através do uso de dados armazenados em um espaço de endereçamento lógico global. A disciplina necessária ao acesso a dados globais é provida por mecanismos de sincronização.

- *Troca de Mensagens* - Os processos de um programa paralelo interagem através da transferência de dados entre seus espaços de endereçamentos lógicos privativos. O transporte de dados entre memórias privativas é provido por mecanismos de comunicação.

Os modelos de interação aqui considerados referem-se à forma de expressão das entidades lógicas do paralelismo nos programas escritos nessas linguagens e não às formas de implementação física, embora as implementações mais imediatas façam corresponder os modelos lógicos e físicos. Entretanto, tanto os dados dos programas logicamente centralizados podem estar fisicamente distribuídos (ficando sujeitos ao transporte para que o acesso físico seja possível) quanto os dados dos programas logicamente distribuídos podem estar fisicamente centralizados (ficando sujeitos a uma disciplina para que o acesso lógico seja correto).

A classificação desse nível poderia ser acrescida de linguagens paralelas mistas (ausentes da figura 2.1) nas quais ambos os métodos de interação estão disponíveis.

Embora os mecanismos de comunicação das linguagens apoiadas em troca de mensagens variem segundo um grande número de propriedades ortogonais, a sincronização pode ser considerada a característica predominante. A figura 2.1 mostra, no terceiro nível da classificação hierárquica, as linguagens apoiadas em troca de mensagens distinguidas de acordo com o método de transporte das mensagens em comunicações síncronas ou assíncronas. Nas *comunicações síncronas*, o processo emissor é bloqueado até que o processo receptor esteja apto a receber a mensagem. Conceitualmente, nas *comunicações assíncronas* o processo emissor não é bloqueado pelo envio da mensagem.

2.1.2 Linguagens Apoiadas em Troca de Mensagens

Os processos de um programa apoiado em troca de mensagens se comunicam por intermédio de mensagens que transitam através de canais lógicos. Os mecanismos de comunicação dessas linguagens podem ser especificados pelas propriedades dos canais lógicos e das mensagens. Embora reconhecendo a importância de outros modelos de interação (como as mensagens ativas [WAL82]), neste trabalho os *processos*, os *canais lógicos* e as *mensagens* são consideradas como entidades lógicas independentes da programação apoiada em troca de mensagens.

As mensagens possuem sempre um processo como origem (dito *emissor*) e um (ou mais de um) processo como destino (dito *receptor*). Em geral, os processos envolvidos em uma comunicação (emissores ou receptores) são chamados de *correspondentes*. O estudo a seguir se restringe as comunicações que possuem um único receptor.

Processos e canais lógicos podem ser tratados como entidades autônomas com características específicas que podem ser examinadas de forma independente. Os programas paralelos apoiados em troca de mensagens formam redes de processos comunicantes. Uma rede de processos comunicantes toma a forma de um grafo bipartido onde o conjunto dos vértices que representam os processos está unido ao conjunto dos vértices que representam os canais lógicos através de arcos (orientados ou não) que definem a conetividade da rede (e o sentido do fluxo das mensagens). O grau de um processo pode ser definido a partir do número de canais lógicos a ele associados.

Em sua forma mais simples, um programa paralelo desse tipo pode conservar uma topologia estática durante toda sua execução. Em sua expressão mais geral, ele pode exibir variações topológicas resultantes da criação dinâmica de processos e/ou canais lógicos. Em um ambiente multicomputador de propósito geral, pode-se reconhecer a necessidade de uso de redes de processos comunicantes de ambos os tipos.

As características das linguagens apoiadas em troca de mensagens podem ser expressas em termos das propriedades de suas entidades fundamentais. Como essas linguagens empregam muitas vezes termos distintos para designar os mesmos conceitos, introduz-se a seguir uma terminologia uniforme com o objetivo de facilitar a comparação de linguagens, a exemplo da que aparece adiante (em 2.1.3).

2.1.2.1 Processos

Em uma linguagem imperativa seqüencial, um programa completo gera um único processo. Em uma linguagem paralela, procedimentos (comuns ou especiais) ou instruções (ou seqüências de instruções) são usadas para gerar os processos componentes de um mesmo programa.

Nas linguagens que permitem a geração de processos a partir de procedimentos, a ativação de cada processo pode ser *implícita* (quando a própria declaração do procedimento basta para provocar sua ativação) ou *explícita* (quando existe na linguagem uma instrução especial para ativá-lo).

Nas linguagens que permitem que instruções sejam executadas como processos, construções paralelas estruturadas do tipo *parbegin-parend* [DIJ68] delimitam as instruções executadas como processos. O processo que contém a construção paralela dispara suas várias instruções componentes como processos e aguarda bloqueado o término de todos eles para prosseguir sua própria execução.

Todas as linguagens paralelas necessitam de mecanismos para criação e remoção de processos. Em algumas linguagens, o número de processos pode ser determinado em tempo de compilação; em outras, isso não é possível. Diz-se que as primeiras permitem *criação estática* e que as segundas permitem *criação dinâmica* de processos.

Os processos dos programas paralelos podem possuir um ou mais *fluxos de execução* internos (*threads*).

Existem ainda dois aspectos cuja presença pode ser importante detectar nas linguagens paralelas: o não-determinismo e a recursividade. Geralmente, o não-determinismo é expresso nas linguagens apoiadas em troca de mensagens por construções derivadas dos comandos guardados [DIJ75]. O uso da recursividade permite a expressão elegante de algoritmos que envolvem criação dinâmica de um número indefinido de processos.

2.1.2.2 Canais lógicos

Em relação à sincronização imposta aos processos envolvidos em uma comunicação, os canais lógicos podem ser *síncronos* ou *assíncronos*. Essa propriedade pode ser expressa em termos da capacidade dos canais lógicos para armazenar mensagens. As mensagens que transitam por canais lógicos síncronos — aqueles que não podem armazenar mensagens —

fluem diretamente entre dois processos. As mensagens que transitam por canais lógicos assíncronos — aqueles que podem armazenar mensagens — ficam retidas temporariamente em um *depósito* de mensagens provido pelo canal lógico. Nesse caso, durante a emissão as mensagens fluem entre o processo emissor e o depósito do canal lógico e durante a recepção, entre o depósito do canal lógico e o processo receptor.

Os canais lógicos síncronos exigem atenção simultânea dos processos comunicantes durante a troca de mensagens. Assim, o primeiro processo apto a comunicar deve esperar que o correspondente faça o mesmo para que a mensagem seja transmitida.

Conceitualmente, na comunicação assíncrona o processo emissor não é bloqueado pelo envio da mensagem enquanto que o receptor só fica bloqueado se o depósito do canal lógico estiver vazio. Na prática, os depósitos dos canais lógicos assíncronos possuem capacidade de armazenamento limitada. Se o limite nunca é alcançado, a comunicação assíncrona se comporta estritamente conforme o conceito. É normal que canais lógicos assíncronos imponham uma disciplina de fila às mensagens que por eles transitam.

Conforme o fluxo das mensagens seja em um ou dois sentidos, os canais lógicos podem ser unidirecionais ou bidirecionais. Um *canal lógico unidirecional* permite o fluxo de mensagens em um único sentido e um *canal lógico bidirecional*, em cada um dos dois sentidos possíveis.

Em função do número de processos servidos, os canais lógicos podem ser classificados em dedicados ou compartilhados. Os *canais lógicos dedicados* estão conectados a exatamente dois processos e servem exclusivamente ao transporte de mensagens entre eles. Os *canais lógicos compartilhados* estão conectados a mais de dois processos e permitem o transporte de mensagens entre dois correspondentes diferentes em momentos distintos.

Em desenhos, os processos podem ser representados por retângulos, os canais lógicos síncronos por barras e os canais lógicos assíncronos por círculos. Os canais lógicos bidirecionais podem ser unidos aos processos que eles servem por arcos não-orientados e os unidirecionais por arcos orientados (indicando o sentido do fluxo). A figura 2.2 apresenta dois exemplos de uso dessas convenções.

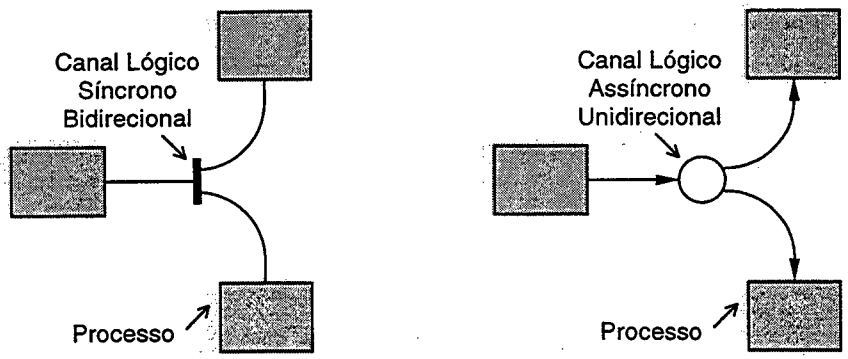


Figura 2.2: Exemplos de canais lógicos compartilhados.

Aos canais lógicos síncronos compartilhados deve se associar uma regra para definir o fluxo de mensagens quando mais de um processo emissor e/ou mais de um processo receptor estiverem aptos a comunicar. A associação aleatória de um emissor a um receptor entre os aptos a comunicar é um exemplo de regra possível.

Os canais lógicos compartilhados assíncronos são chamados de *caixas-postais* (*mailboxes*) quando servem vários emissores e vários receptores e *portas* (*ports*) quando servem vários emissores e apenas um receptor [AND91]. Quando existe apenas um emissor e um receptor, eles são chamados de *dutos* (*pipes*).

Os processos emitem e recebem mensagens designando os canais lógicos através dos quais as mensagens devem transitar. Se entre dois processos existir um único canal lógico síncrono bidirecional dedicado, os processos podem emitir e receber mensagens designando diretamente o processo conectado ao outro extremo do canal lógico. A designação de processo é dita *direta* e a de canal lógico *indireta*. Em geral, considera-se que a designação direta implica na definição implícita de canais lógicos anônimos síncronos bidirecionais dedicados únicos. Esses canais lógicos podem ser representados de forma simplificada em desenhos por arcos unindo diretamente os dois processos servidos por eles (figura 2.3).

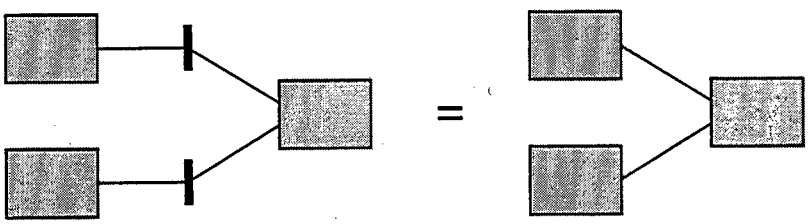


Figura 2.3: Canais lógicos síncronos bidirecionais dedicados únicos.

Como no caso dos processos, as linguagens apoiadas em troca de mensagens necessitam de mecanismos para criação e remoção de canais lógicos. Em algumas linguagens, o número de canais lógicos pode ser determinado em tempo de compilação; em outras, isso não é possível. Diz-se que as primeiras permitem *criação estática* e que as segundas permitem *criação dinâmica* de canais lógicos.

Em geral, as linguagens que permitem criação dinâmica de processos, permitem também a criação dinâmica de canais lógicos. Nas linguagens com criação estática de processos e canais lógicos, as redes de processos comunicantes podem ser desenhadas a partir da leitura do texto do programa paralelo.

2.1.2.3 Mensagens

As mensagens que transitam entre processos através de canais lógicos podem possuir várias propriedades. Assim, elas podem ter todas o mesmo tamanho ou podem ter tamanhos quaisquer. Elas podem também ter formato livre ou definido.

Os canais lógicos que permitem o trânsito de mensagens de diversos tamanhos favorecem a liberdade de programação. Entretanto, elas exigem maior esforço de gerência de espaço em seus depósitos no caso de canais lógicos assíncronos.

Os canais lógicos que obrigam a especificação de tipos de dados associados às mensagens, valem-se dessa característica para detectar erros de comunicação referentes à formatação de dados. Por outro lado, formatos livres oferecem maior flexibilidade de programação.

2.1.2.4 Disciplinas de Comunicação

Dois modelos de programação apoiados em troca de mensagens de grande importância prática são o modelo produtor-consumidor e o modelo cliente-servidor [AND91]. O modelo *produtor-consumidor* refere-se a interação que se desenvolve entre dois processos (o produtor e o consumidor) através de um fluxo unidirecional de mensagens do produtor para o consumidor. O modelo *cliente-servidor* refere-se a interação que desenvolve entre dois processos (o cliente e o servidor) através de um fluxo bidirecional composto de pares de mensagens requisição-resposta, a primeira das quais do cliente para o servidor e a segunda do servidor para o cliente.

Esses modelos de programação podem ser impostos pelas linguagens através de padrões de interação entre processos chamados de disciplinas de comunicação. Para provocar as interações específicas das disciplinas, *operadores de comunicação* são colocados à disposição dos processos correspondentes. A disciplina elementar do modelo produtor-consumidor consiste na *transmissão de mensagem* individual entre dois processos. Embora através dela seja possível expressar todas as redes de processos comunicantes, para aplicações que seguem o modelo cliente-servidor disciplinas de mais alto nível como o *encontro* e a *chamada de procedimento remoto* podem ser de uso mais confortável e de implementação mais eficiente [BAL89].

A transmissão de mensagem pode empregar com naturalidade tanto canais lógicos síncronos quanto canais lógicos assíncronos, mas as outras disciplinas citadas são de natureza essencialmente síncrona. Quando elas utilizam canais lógicos assíncronos, as restrições de tempo impostas por elas são obtidas através de outros expedientes.

As disciplinas de comunicação estão sujeitas a variações resultantes do uso de construções não-determinísticas. O emprego dessas construções variam sensivelmente entre as linguagens.

A seguir são descritas as características das três disciplinas de comunicação citadas. Elas são exemplificadas através de desenhos exibindo uma interação completa, onde cada linha associada a um processo representa sua execução ao longo do tempo. Linhas contínuas representam o processo realizando uma atividade autônoma e linhas tracejadas representam o processo em estado de espera. As comunicações são representadas por regiões retangulares envolvendo simultaneamente os dois agentes envolvidos (dois processos nas comunicações síncronas e um processo e um canal lógico nas comunicações assíncronas). Pontos de transição notáveis vêm assinalados com a indicação do evento causador.

Transmissão de mensagem

A transmissão de mensagem é uma disciplina de comunicação do modelo produtor-consumidor. Ela representa a interação elementar entre processos, permitindo a associação das propriedades dos canais lógicos em todas as combinações possíveis. Dessa forma, nessa disciplina os canais lógicos podem ser síncronos ou assíncronos, unidirecionais ou bidirecionais, dedicados ou compartilhados. No caso de canais lógicos síncronos bidirecionais dedicados únicos, a designação pode ser direta ou indireta.

As linguagens apoiadas em troca de mensagens que oferecem essa disciplina devem possuir dois tipos de operadores de comunicação: um para emissão e outro para a recepção de mensagens (designados genericamente no que segue pelas palavras-chave *send* e *receive*, respectivamente). A transmissão se completa quando ambos os processos correspondentes executam operadores complementares.

A figura 2.4 apresenta um exemplo de transmissão de mensagem entre dois processos através de um canal lógico síncrono.

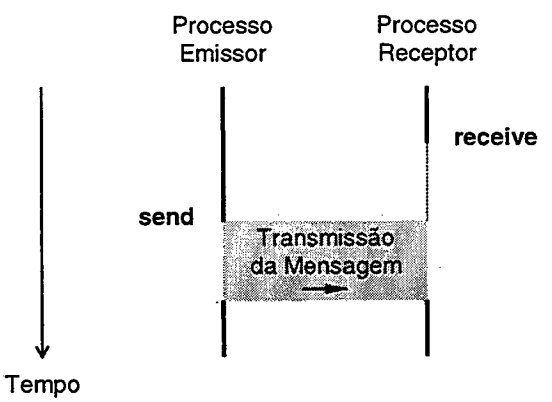


Figura 2.4: Exemplo de transmissão de mensagem através de um canal lógico síncrono.

Na transmissão síncrona exemplificada, os dois processos evoluem de forma autônoma até que um deles execute um operador de comunicação. A partir daí, esse processo fica a espera que seu correspondente execute o operador complementar para que a mensagem possa fluir entre eles. Após a transmissão da mensagem, os dois processos voltam a executar de forma autônoma. Essa disciplina é considerada elementar porque as outras podem ser implementadas a partir dela.

Encontro

O encontro é uma disciplina de comunicação do modelo cliente-servidor. Ela envolve a transmissão de duas mensagens de sentidos opostos entre os correspondentes através de um canal lógico síncrono bidirecional dedicado. As linguagens paralelas que oferecem essa disciplina devem possuir dois operadores de comunicação: um para convidar ao encontro e outro para aceitá-lo. Eles são designados genericamente no que segue pelas palavras-chave

call e *accept*, respectivamente. O encontro tem lugar entre dois processos, quando ambos se manifestam explicitamente através de operadores complementares.

O processo que aceita convites para encontros (processo chamado) define um ou mais *pontos de encontro* seguindo a sintaxe de um cabeçalho de procedimento convencional. O processo que convida ao encontro (processo chamador) designa diretamente o processo chamado e um ponto de encontro específico na execução do operador *call*. O processo que aceita convites também indica um ponto de encontro específico na execução do operador *accept*, mas aceita o encontro com qualquer processo.

A figura 2.5 exibe um exemplo de encontro entre dois processos.

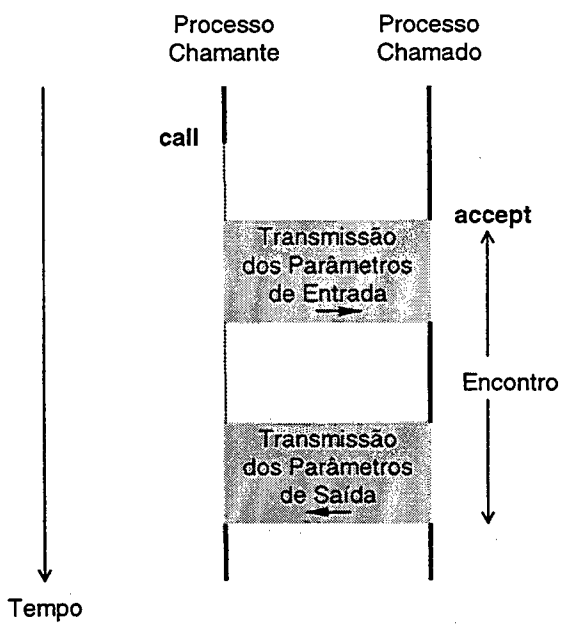


Figura 2.5: Exemplo de encontro.

Os dois processos evoluem de forma autônoma até que um deles execute um operador de comunicação. A partir daí, esse processo fica a espera que seu correspondente execute o operador de comunicação complementar para que o encontro possa ter lugar. Nesse momento, o processo que convida ao encontro envia uma mensagem contendo os parâmetros do ponto de encontro especificado e a partir de então espera pela sua execução por parte do processo que aceitou o encontro. Finda a execução deste, os parâmetros de saída do ponto de encontro são transmitidos ao processo que o convidou ao encontro. Daí em diante, os dois processos voltam a executar de forma autônoma. O mecanismo de troca de mensagens embutido nesta disciplina é escondido do programador.

Chamada de Procedimento Remoto

A chamada de procedimento é uma disciplina de comunicação do modelo cliente-servidor. Ela envolve a transmissão de duas mensagens de sentidos opostos entre os correspondentes através de um canal lógico síncrono bidirecional dedicado. As linguagens

paralelas que oferecem essa disciplina devem possuir um operador para a chamada do procedimento remoto (designado genericamente no que segue pela palavra-chave *call*).

Um processo que aceita chamadas (processo chamado) define um ou mais procedimentos que podem ser chamados por outros processos. O processo chamador designa diretamente o processo chamado e um procedimento específico na execução do operador *call*. A chamada de procedimento é aceita implicitamente, seja qual for o chamador.

A figura 2.6 mostra um exemplo de chamada de procedimento remoto.

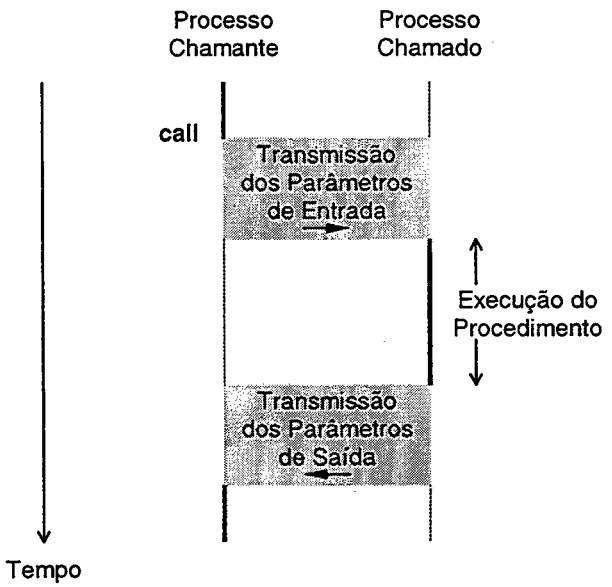


Figura 2.6: Exemplo de chamada de procedimento remoto.

O processo chamador segue de forma autônoma até executar o operador *call*, enquanto o processo que aceita chamadas permanece em estado de espera. Nesse momento, o processo chamante envia uma mensagem contendo os parâmetros de entrada do procedimento chamado e passa a aguardar a sua execução enquanto o processo que aceita chamadas cria um novo fluxo de execução (*thread*) a partir do procedimento chamado. Finda a execução deste, os parâmetros de saída do procedimento são transmitidos para o processo chamador. Daí para frente, o processo chamador volta a executar de forma autônoma e o processo chamado volta a esperar uma nova chamada. O mecanismo de troca de mensagens embutido nesta disciplina é escondido do programador.

Conforme essa descrição da chamada de procedimento remoto, o processo chamador tem consciência, através da própria sintaxe da chamada, que o procedimento chamado pertence a outro processo [BAL89]. Variantes dessa disciplina podem visar a assimilação de procedimentos remotos às chamadas de procedimentos regulares, procurando tornar as interações entre os processos envolvidos transparentes ao programa, como é mostrado em 2.3.2.1.

2.1.3 Exemplos

O estudo de linguagens apoiadas em troca de mensagens permite identificar os meios que devem ser colocados à disposição pelo serviço de comunicação para facilitar a implementação de linguagens para programação e a construção de aplicações paralelas em multicomputadores.

O número de linguagens paralelas propostas é expressivo — Bal apresenta uma lista com mais de uma centena delas [BAL89] — e tende a aumentar com o crescimento do interesse pelo paralelismo constatado em anos recentes. Em função dos objetivos deste trabalho, as linguagens que interessam estudar são as linguagens imperativas convencionais apoiadas em troca de mensagens através de canais lógicos síncronos. Trata-se de enfrentar o *problema básico* das trocas de mensagens síncronas, inevitáveis nos multicomputadores, por estarem presentes nas comunicações entre nós vizinhos das redes estáticas e em todas as comunicações das redes dinâmicas.

A seguir são sumariadas as principais características associadas aos processos, aos canais lógicos e às mensagens das linguagens CSP [HOA78], Superpascal [HAN94a], Joyce [HAN87], Ada [BUR85] e DP [HAN78], através da terminologia introduzida em 2.1.2. É importante estudar diversos mecanismos específicos de trocas de mensagens síncronas porque um dos objetivos deste trabalho é a flexibilidade para a implementação de diversas variantes de forma simples e eficiente.

Todas as cinco linguagens estudadas propõem mecanismos de interação entre processos que consistem de troca de mensagens síncronas. As interações de CSP, Superpascal e Joyce se resumem as transmissões explícitas de mensagens individuais. As de Ada e DP se fazem implicitamente aos pares: as mensagens correspondem às transmissões dos parâmetros de entrada e de saída do ponto de entrada ou do procedimento remoto, respectivamente.

Em CSP, os correspondentes se engajam em uma comunicação pela designação direta e recíproca. Em Ada e DP, o processo chamante deve designar diretamente seu correspondente enquanto que o processo chamado aceita chamadas anônimas. Em Superpascal e Joyce, as comunicações se fazem pela designação de canais lógicos mas apenas os canais lógicos de Joyce são compartilhados.

As disciplinas de comunicação de Ada (encontro) e DP (chamadas de procedimentos remotos) são de mais alto nível em relação aos das outras três linguagens no sentido em que se pode obter o mesmo efeito por composição. O caráter elementar das comunicações das outras linguagens confere a elas maior flexibilidade para a realização de redes de processos comunicantes.

CSP possui grandes restrições estáticas que se por um lado permitem a determinação da topologia da rede de processos comunicantes em tempo de compilação, por outro lado tornam difícil a programação de numerosas aplicações de importância prática. O caráter dinâmico das demais linguagens facilita o desenvolvimento de aplicações onde o número de processos e de canais lógicos variam durante a execução.

Com exceção de Superpascal e DP, as demais linguagens suportam comandos guardados permitindo o controle das interações entre processos. Em CSP e Ada esse controle se resume à recepção de mensagens. Em Ada, a recepção está implícita na instrução *accept*.

Em Joyce, ele compreende ainda emissões de mensagens. Em DP, os comando guardados não envolvem comunicações.

A tabela 2.1 resume as propriedades das entidades fundamentais do paralelismo dessas cinco linguagens.

Tabela 2.1: Características de algumas linguagens paralelas.

		CSP	**Superpascal**	**Joyce**	**Ada**	**DP**
Processo	Criação	Estática	. Dinâmica	Dinâmica	Dinâmica	Estática
	Fluxo de execução	Único	Único	Único	Único	Múltiplo
	Não-determinismo	Sim	Não	Sim	Sim	Sim
	Recursividade	Não	Sim	Sim	Sim	Sim
Canal lógico	Criação	Estática	Dinâmica	Dinâmica	Dinâmica	Estática
	Sincronismo	Síncrono	Síncrono	Síncrono	Síncrono	Síncrono
	Sentido	Unidirecional	Unidirecional	Bidirecional	Bidirecional	Bidirecional
	Acesso	Dedicado	Dedicado	Compartilhado	Dedicado	Dedicado
	Designação	Direta	Indireta	- .	Direta	Direta
Mensagem	Tamanho	Variável	Variável	Variável	Variável	Variável
	Tipo	Sim	Sim	Sim	Sim	Sim

2.2 Computadores Paralelos

Neste subcapítulo são introduzidos os multicomputadores. Inicialmente, é apresentada uma classificação geral para as arquiteturas de computadores para situar os multicomputadores no contexto mais amplo das máquinas paralelas. Em seguida, são examinadas as redes de interconexão dos multicomputadores em função da grande influência que elas exercem sobre o desempenho global dessas máquinas. Dois multicomputadores são descritos como exemplos.

2.2.1 Classificação

Neste texto, computadores paralelos designam máquinas que resultam de um projeto coerente de integração de componentes fisicamente próximos, tipicamente contidos em um único móvel. Essas máquinas podem eventualmente operar de forma autônoma, mas freqüentemente estão conectadas a uma máquina hospedeira ou a uma rede local de computadores.

Diversos autores têm procurado estabelecer taxonomias para os computadores [DUN90] [TRE88]. Diante da diversidade dos modelos propostos e dos critérios adotados, resultam classificações diferentes. A classificação apresentada a seguir não pretende ser completa: seu objetivo é antes o de situar os multicomputadores dentro do universo das máquinas paralelas.

A árvore da figura 2.7 representa uma possível classificação hierárquica geral para as arquiteturas de computadores.

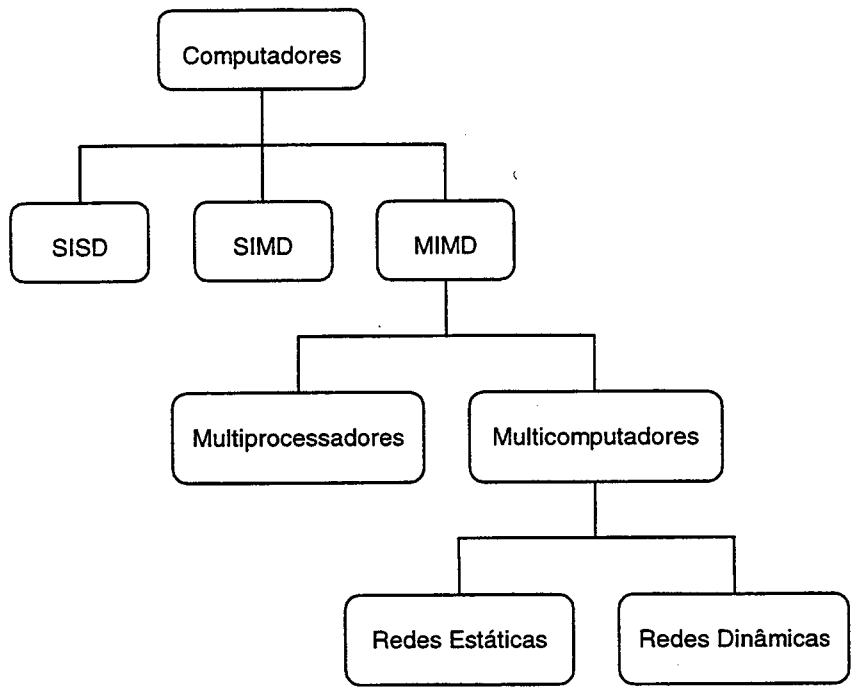


Figura 2.7: Uma classificação geral para as arquiteturas de computadores.

No primeiro nível, à partir da raiz, aparece a classificação de Flynn [FLY72] que se apoia sobre o número de fluxos de instruções e de dados ativos simultaneamente na arquitetura. Dessa forma, distinguindo o número desses fluxos em únicos ou múltiplos, obtém-se quatro categorias distintas de computadores:

- SISD (*Single Instruction, Single Data*) - A execução de um único programa sobre seus dados caracteriza os computadores seqüenciais convencionais. Este modelo é o mais largamente utilizado atualmente na construção de computadores comerciais.

- MISD (*Multiple Instruction, Single Data*) - Computadores dessa categoria (ausentes da figura 3.1), executando múltiplas instruções simultaneamente sobre os mesmos dados, representam uma conseqüência teórica de difícil aplicação prática do critério de classificação utilizado.
- SIMD (*Single Instruction, Multiple Data*) - A presença de uma única unidade de controle permite o cadenciamento da execução de um único programa operando simultaneamente sobre dados diferentes. Este modelo é adequado para aplicações envolvendo cálculos matriciais intensivos.
- MIMD (*Multiple Instruction, Multiple Data*) - Computadores dessa categoria se caracterizam pela execução de múltiplos programas independentes, cada um operando sobre seus próprios dados. Esse modelo de computador paralelo é o mais adequado para aplicações de uso geral e vem sendo largamente empregado na construção de máquinas experimentais e comerciais.

Computadores de arquitetura SISD e SIMD não são apresentados aqui com mais detalhes. Eles não despertam maior interesse para este trabalho, seja pela natureza seqüencial (arquitetura SISD), seja pelo emprego em domínios específicos (arquitetura SIMD). Os computadores paralelos de uso geral, se enquadram na categoria de computadores de arquitetura MIMD.

No segundo nível da classificação hierárquica apresentada na figura 2.7, os computadores de arquitetura MIMD são diferenciados de acordo com a capacidade de acesso à memória em multiprocessadores ou multicomputadores.

Os *multiprocessadores* são compostos por um conjunto de processadores homogêneos unidos a um conjunto de módulos de memória através de redes de interconexão dinâmicas (figura 2.8). As redes de interconexão devem prover a cada processador acesso a qualquer módulo de memória.

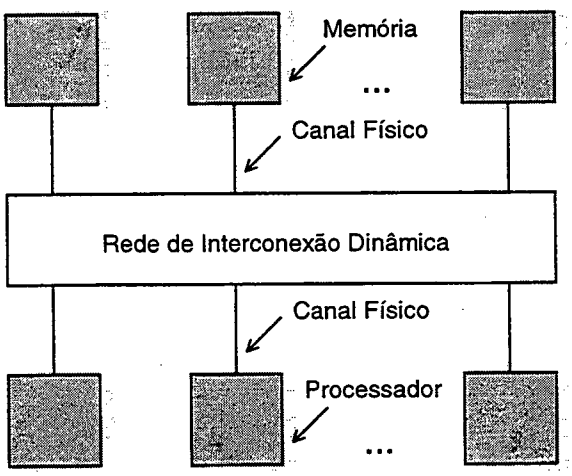


Figura 2.8: Arquitetura de um multiprocessador.

Os *multicomputadores* são compostos de *nós* ou *elementos processadores* largamente autônomos, cada um dos quais dispondo basicamente de um processador, de memória privativa e de suporte físico para comunicação com outros nós (figura 2.9). Em seus componentes essenciais, cada nó de um multicomputador segue, nos casos mais simples, a arquitetura SISD.

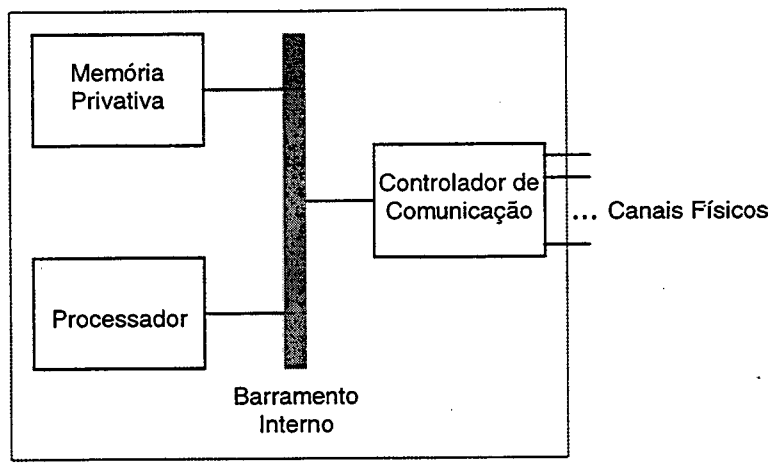


Figura 2.9: Composição básica de um nó de multicomputador.

A figura 2.7 mostra, no terceiro nível da classificação hierárquica, os multicomputadores distinguidos de acordo com a natureza das redes de interconexão de seus nós em redes estáticas ou redes dinâmicas.

Os nós dos multicomputadores podem ser unidos aos pares através de canais físicos bidirecionais diretos dando origem às redes de interconexão estáticas. A figura 2.10-a mostra um exemplo de multicomputador com rede estática. Alternativamente, os nós dos multicomputadores podem ser unidos através de uma rede de interconexão dinâmica como mostra a figura 2.10-b.

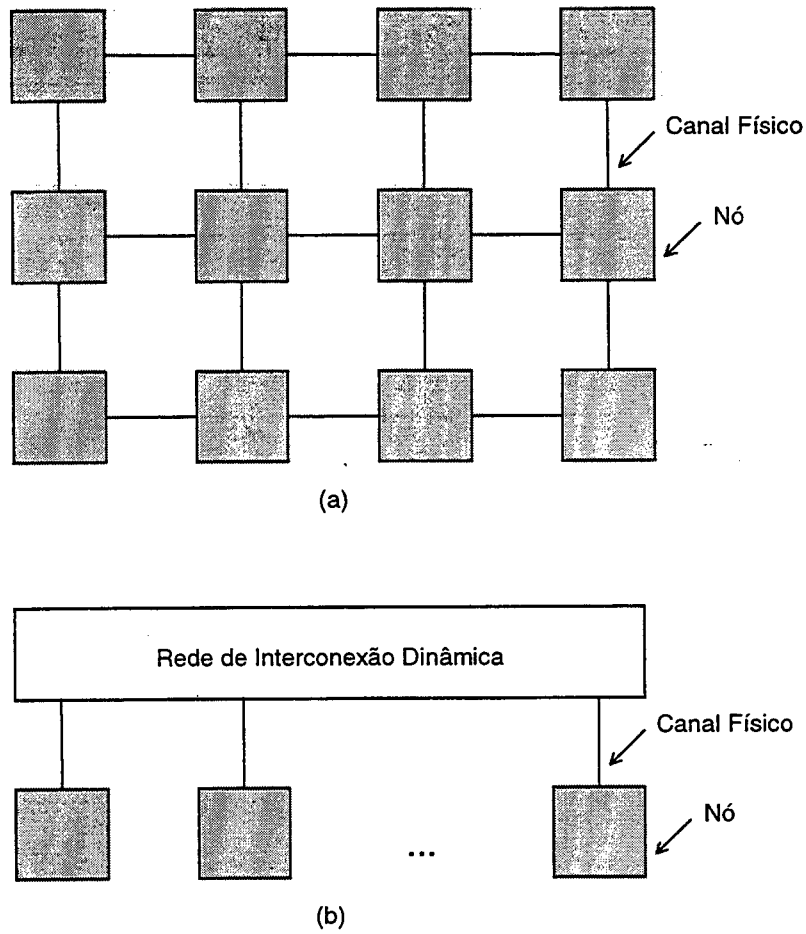


Figura 2.10: Arquitetura de um multicomputador.

A conjunção das seguintes características serve para identificar os multicomputadores e ajuda a distingui-los das outras máquinas paralelas e das redes de computadores [REE87]:

- *Grande número de nós homogêneos* – A disponibilidade comercial e o baixo custo dos microprocessadores, aliados à facilidade de montagem, permitem a construção de multicomputadores com dezenas, centenas e até milhares de nós construídos com os mesmos componentes.
- *Proximidade física dos nós* – Os multicomputadores resultam de projetos globais que permitem a integração de componentes geralmente confinados em um único móvel.
- *Comunicações apoiadas em troca de mensagens* – Os processos que executam em nós distintos só podem interagir através de troca de mensagens. Os nós evoluem de maneira assíncrona, salvo durante as comunicações.
- *Alta velocidade das comunicações* – As redes de interconexão dos multicomputadores ligam pares de nós através de canais físicos bidirecionais diretos que permitem comunicações confiáveis a altas velocidades.
- *Alto grau de paralelismo real* – O paralelismo real fornecido pelos multicomputadores pode ser ressaltado tanto pelo grande número de fluxos de instruções e de dados simultâneos quanto pelo grande número de comunicações diretas simultâneas.

- *Granulosidade média de processamento* – Os multicomputadores encorajam a exploração do paralelismo real pela composição de programas com muitos processos cooperantes de granulosidade média.

2.2.2 Redes de Interconexão de Multicomputadores

Os multicomputadores se distinguem entre si principalmente pela natureza de suas redes de interconexão. A eficiência dessas máquinas é predominantemente determinada pelas características dessas redes.

Uma característica presente tanto nas redes estáticas quanto nas dinâmicas é o uso de canais físicos diretos. De um modo geral, as redes estáticas não dispõem de canais físicos diretos conectando todos os pares de nós enquanto que as redes dinâmicas permitem em geral conectar todos os nós aos pares através de canais físicos diretos, ainda que em momentos diferentes.

As redes de interconexão dinâmicas empregadas nos multicomputadores possuem as mesmas características topológicas das empregadas nos multiprocessadores. Nos multiprocessadores, as conexões entre processadores e módulos de memória devem conter um grande número de fios, para transportar simultaneamente os *bits* de controle, endereço e dados necessários às velocidades normalmente requeridas nos acessos à memória. Nos multicomputadores, as conexões entre nós podem conter um número reduzido de fios que no caso mais simples se resume a um único fio para transportar serialmente os *bits* das mensagens trocadas entre nós, além dos fios de controle. Dessa forma, o custo dos circuitos de interconexão dos multicomputadores pode ser muito menor do que o dos multiprocessadores para o mesmo número de canais físicos. As características topológicas das redes dinâmicas são apresentadas adiante em 2.2.2.2 no contexto dos multicomputadores.

2.2.2.1 Redes Estáticas

Os multicomputadores com redes de interconexão estáticas podem ser representados esquematicamente por grafos não-orientados onde os vértices representam os nós e os arcos representam os canais físicos bidirecionais de comunicação (conforme visto na figura 2.10-a). A avaliação dessas redes, pode valer-se de alguns parâmetros característicos da topologia dos grafos não-orientados [FIN80].

Um conceito útil para a descrição de vários parâmetros é o da distância entre nós. A *distância entre dois nós* é o número de canais físicos no caminho mais curto entre eles. Nós ligados por um canal físico direto são ditos vizinhos. Dois valores fundamentais associados aos multicomputadores com rede estática são o número de nós (N) e o número de canais físicos (expresso em geral em função de N).

Dois parâmetros importantes para a caracterização das redes estáticas são o grau dos nós e o diâmetro do multicomputador, quando expressos em função do número de nós N. O *grau do nó* é o número de canais físicos a ele conectados. Quanto maiores os valores do grau, maiores as alternativas para comunicação direta com outros nós. Redes com valores constantes para o grau dos nós facilitam a escalabilidade (*escalability* [NUS91]) porque o número de nós pode crescer sem alterar a composição dos nós. O *diâmetro de um*

multicomputador é a distância existente entre seus nós mais distantes. Quanto menores os valores de diâmetro, menores os tempos máximos para o trânsito das mensagens. Em análises detalhadas, podem ser considerados mais parâmetros da geometria da rede, além de outros parâmetros dependentes do comportamento dos programas paralelos executados no multicomputador ou da forma de distribuição dos processos desses programas sobre a arquitetura [CHU80] [NOR93] [LO 97].

Muitas redes de interconexão estáticas têm sido propostas e analisadas. Algumas têm sido usadas apenas como modelos de referência enquanto outras têm sido adotadas na construção de máquinas experimentais e comerciais. A figura 2.11 coleciona oito tipos de redes estáticas escolhidas entre as mais citadas na literatura.

A *linha* é a mais simples das topologias de referência (figura 2.11-a), enquadrando-se no limite inferior de eficiência. Essas redes não são adequadas além de um número muito pequeno de nós, exceto para algoritmos expressos como redes lineares de processos (*pipelines*).

O *anel* é uma rede caracterizada por nós que possuem cada um 2 vizinhos (figura 2.11-b). Essa topologia é adequada somente para pequenas redes que executam algoritmos com pouca comunicação. O anel é uma rede econômica mas pouco confiável e pouco eficiente. Apesar disso, já foi usada na construção efetiva de computadores como o KSR-1 [BUR92], por exemplo.

A *árvore binária* completa é bem adaptada a algoritmos de pesquisa e de classificação (figura 2.11-c). A máquina DADO [STO86] serve de exemplo de emprego dessas redes.

A *estrela* é um caso especial de árvore caracterizada por um nó central ao qual são conectados todos os demais (figura 2.11-d). A rede em estrela pode ser interessante para um número moderado de nós porque seu controle à partir do nó central é simples [WIT81]. Em compensação ela não resiste à falha desse nó.

A *grelha* quadrada é uma rede caracterizada por nós interiores com 4 vizinhos (figura 2.11-e). A grelha é um tipo difundido de topologia, tendo sido empregada em diversas máquinas reais. A correspondência entre a topologia da grelha e os algoritmos orientados para cálculos matriciais justifica seu emprego nesse domínio. Como exemplo de multicomputador que emprega essa rede, pode ser citada a máquina Illiac IV [BAR68].

O *torus* pode ser visto como uma combinação da grelha e do anel (figura 2.11-f). Diversamente da grelha, o grau dos nós do torus é constante. Seu diâmetro é bastante inferior ao da grelha e do anel.

O número de nós de um *hipercubo* (figura 2.11-g) pode ser expresso como uma potência de 2. O grau de seus nós e o seu diâmetro são iguais a $\log_2 N$. (O número $\log_2 N$, tão representativo dessa topologia, é chamado de dimensão do hipercubo). Essa arquitetura é bastante difundida [PEA77]. Encontra-se na literatura sua avaliação para uma larga classe de algoritmos fundamentais, sobretudo no domínio do cálculo numérico. Como exemplos de multicomputadores que usam essa rede podem ser mencionados o iPSC [INT86] e o FPS T Series [GUS86]. O grau dos nós do hipercubo aumenta com a dimensão dessas redes, o que prejudica sua escalabilidade e limita tecnologicamente a construção de grandes configurações.

Entre as topologias estáticas de referência, o *grafo completo* (figura 2.11-h) determina o limite superior de eficiência. Como todos os nós são mutuamente vizinhos, as trocas de

mensagens são sempre feitas diretamente entre os nós envolvidos nas comunicações e conseqüentemente podem prescindir de mecanismos de roteamento de mensagens. Os grafos completos não podem ser realizados com a tecnologia atual, salvo para um reduzido número de nós.

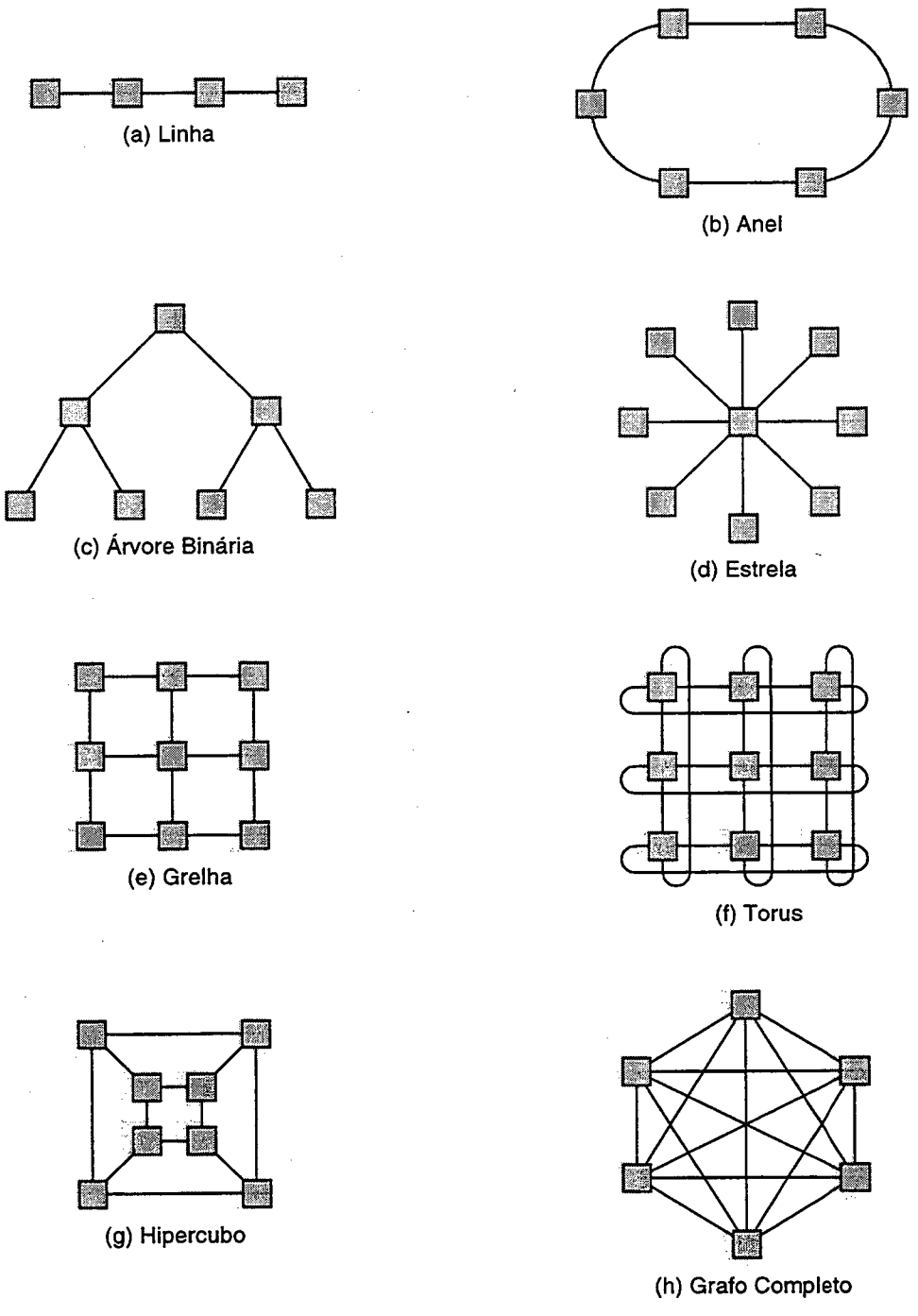


Figura 2.11: Redes de interconexão estáticas.

A tabela 2.2 resume algumas características das redes estáticas mostradas na figura 2.11, expressas em função do número de nós N.

Tabela 2.2: Características das redes estáticas (em função de N, o número de nós).

Tipo da Rede	Número de Canais Físicos	Grau dos Nós	Diâmetro da Rede
Linha	N - 1	1 ou 2	N - 1
Anel	N	2	$\lfloor N/2 \rfloor$
Árvore Binária	N - 1	1, 2 ou 3	$2 (\log_2 N - 1)$
Estrela	N - 1	1 ou N - 1	2
Grelha Quadrada	$2 (N - N^{1/2})$	2, 3 ou 4	$2 (N^{1/2} - 1)$
Torus	2 N	4	$2 \lfloor N^{1/2} - 1 \rfloor$
Hipercubo	$N \log_2 N / 2$	$\log_2 N$	$\log_2 N$
Grafo Completo	$N (N - 1) / 2$	N - 1	1

Além das redes de interconexão estáticas clássicas apresentadas na figura 2.11, novas topologias têm continuamente sido propostas em anos mais recentes. Entre as mais proeminentes não se pode deixar de citar as *fat-trees* [LEI85] usadas em multicomputadores como o CM-5 [TMC92].

As redes de interconexão estáticas propostas são invariavelmente representadas por topologias que levam em conta preponderantemente a regularidade das suas propriedades geométricas e seus aspectos construtivos. Sendo assim, os multicomputadores com redes estáticas podem apresentar diferentes características de adaptabilidade a aplicações específicas [CHA93]. Uma grelha, por exemplo, pode acomodar outras redes, como um anel ou uma árvore binária, tomando um subconjunto de seus nós e canais físicos, sendo que o inverso não se verifica [DUN90].

Cada rede estática considerada individualmente se adapta melhor a alguns algoritmos do que a outros. Por isso, suas qualidades podem ser melhor avaliadas em função das aplicações visadas [WAL97]. Quando não existe uma correspondência perfeita entre as topologias das redes de processos comunicantes e as das redes de interconexão estáticas de um multicomputador, surge a necessidade do roteamento das mensagens que devem ser transportadas entre dois nós não vizinhos, passando por nós intermediários.

O roteamento de mensagens é um problema intrínseco das redes estáticas que concorre para o decréscimo de desempenho dessas máquinas. Todas as máquinas construídas com redes estáticas necessitam de suporte para esse problema visando ampliar sua gama de aplicações. As soluções adotadas podem envolver suporte de *software* ou de *hardware*. Quando existe apenas suporte de *software*, os nós devem reservar parte do tempo do seu processador e da sua memória privativa para tratá-lo, conforme se apresenta mais adiante em 2.3.1.1. O suporte de *hardware* vem na forma de roteadores específicos acoplados aos nós que se encarregam do roteamento sem perturbar o funcionamento normal dos demais componentes dos nós

[DAL86]. Essa solução pode melhorar o desempenho geral da máquina, mas concorre para o incremento da complexidade dos nós, sem alterar a essência do problema.

2.2.2.2 Redes Dinâmicas

Nas redes dinâmicas, canais físicos de comunicação unem os nós através de um circuito de chaveamento eletrônico interposto entre eles que pode ser manipulado durante o funcionamento normal dos multicomputadores.

A figura 2.12 apresenta o diagrama genérico de uma rede de interconexão dinâmica quando se quer separar os elementos a ela conectados em dois grupos: as entradas e as saídas. Nesse caso, os elementos de um grupo somente podem ser conectados aos do outro. Entretanto, a designação de entrada ou saída serve aqui apenas para distinguir os elementos desses dois grupos e não para indicar o sentido do fluxo de dados, como ela dá a entender. Efetivamente, esses elementos podem ser canais físicos tanto unidirecionais quanto bidirecionais. Quando não há necessidade de distinguir esses elementos, a rede pode ser apresentada de forma simplificada conforme o diagrama visto na figura 2.10-b.

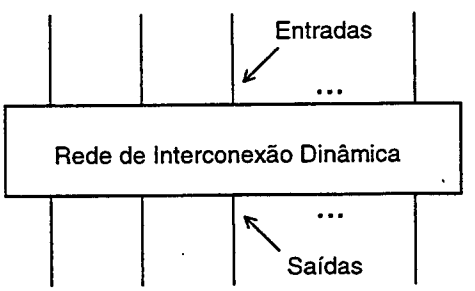


Figura 2.12: Redes de interconexão dinâmicas.

Para a avaliação das redes dinâmicas, podem ser estabelecidos parâmetros que permitam inferir sua eficiência. A *complexidade dos circuitos* de chaveamento pode ser medida pelo número de componentes eletrônicos necessários para sua composição. Esse parâmetro está relacionado com o custo da rede. O *tempo mínimo para uma conexão* é o tempo necessário para o estabelecimento de uma conexão entre dois canais físicos livres. A *conetividade da rede* dinâmica é medida pelo número máximo de conexões que podem estar ativas a cada momento. Esse parâmetro indica o número de comunicações que podem se desenrolar em paralelo.

As redes de interconexão dinâmicas podem ser ordenadas por ordem crescente de eficiência e de custo em barramentos, redes com múltiplos estágios e em *crossbars* [HWA93].

Barramento

O *barramento* permite a conexão de dois canais físicos quaisquer. O tempo mínimo para uma conexão é constante mas apenas uma conexão pode estar ativa a cada momento (figura 2.13).

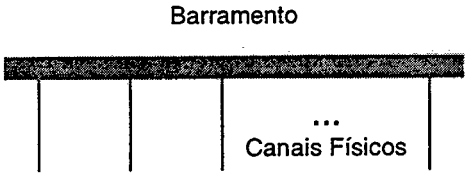


Figura 2.13: Barramento.

O barramento é conceitualmente simples, bem conhecido e de fácil construção. Seu uso é tão difundido que conta com diversas normas. Entretanto, um barramento único não pode ser razoavelmente eficiente além de um número reduzido de nós, exceto se as comunicações através dele não são muito freqüentes. Para aliviar essa ineficiência, têm-se proposto o uso de múltiplos barramentos tanto em multiprocessadores [MUD87] [WIL87] quanto em multicomputadores [SER93].

Redes com Múltiplos Estágios

Muitas *redes com múltiplos estágios* podem ser definidas. Essas redes podem ser classificadas como não-bloqueantes, rearranjáveis ou bloqueantes [FEN81]. Em uma rede não-bloqueante, toda conexão entre dois canais físicos livres pode ser satisfeita imediatamente sem interferir com outras conexões já existentes. Um exemplo importante de rede dessa classe é a rede de Clos [CLO53]. Em uma rede rearranjável, toda conexão entre dois canais físicos livres pode ser satisfeita, mesmo que eventualmente possa ser necessário rearranjar as conexões já existentes. Um exemplo de rede dessa classe e a rede de Benes [THU78]. Em uma rede bloqueante, uma conexão existente pode impossibilitar o estabelecimento de uma nova conexão entre dois canais físicos livres. Um exemplo de rede dessa classe é a rede Ômega [LAW75].

Uma rede genérica de múltiplos estágios emprega vários circuitos de chaveamento eletrônico em cada estágio. Entre dois estágios adjacentes são usados padrões estáticos de interconexão. O número de estágios determina o número de operações (e o tempo) de estabelecimento de uma conexão entre dois nós. O tempo mínimo para uma conexão é proporcional ao número de estágios.

Um circuito de chaveamento MxN possui M entradas e N saídas. Na prática é comum o emprego de circuitos de chaveamento com o número de entradas igual ao número de saídas, ambos expressos como uma potência de 2 ($M = N = 2^K$, para $K \geq 1$). O circuito de chaveamento 2x2 é chamado circuito binário. Quando uma entrada pode ser conectada a apenas uma saída a cada momento, o circuito é chamado de *crossbar*. O *crossbar* binário permite as conexões ditas direta e cruzada mostradas na figura 2.14.

Figura 2.14: *Crossbar* binário.

Muitos padrões estáticos de interconexão entre estágios podem ser definidos. A figura 2.15 mostra o exemplo do padrão denominado *perfect shuffle*. Nesse padrão, parte-se da identificação dos canais físicos de origem como números binários (de 3 dígitos, na figura) e o padrão estabelece a conexão de cada um deles com os canais físicos de destino cujas identificações são obtidas pela rotação de um *bit* para a esquerda.

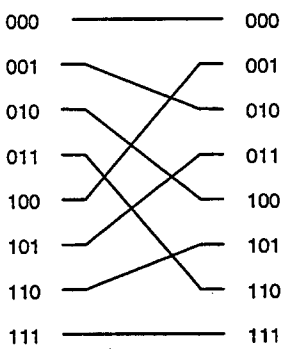


Figura 2.15: *Perfect shuffle*.

Como um exemplo, a rede Ômega 8x8 que aparece na figura 2.16 é construída com estágios compostos de *crossbars* binários e com padrão *perfect shuffle* de conexões entre estágios. Em geral, a rede Ômega tem $\log_2 N$ estágios para N entradas e N saídas.

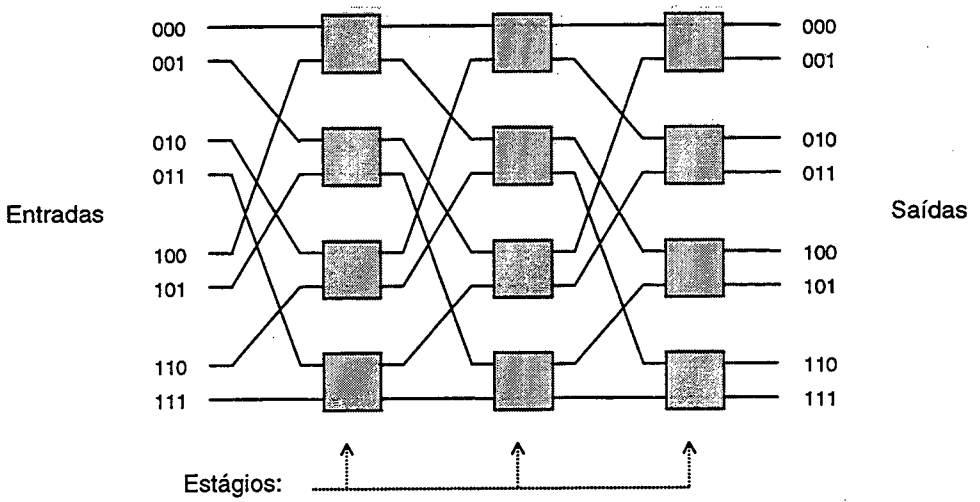


Figura 2.16: Rede Ômega 8x8.

Vários multicomputadores têm sido construídos com redes de múltiplos estágios, sendo importante citar aqui pelo menos o SP2 [AGE95].

Crossbar

O *crossbar* é uma rede dinâmica de estágio único. Ela é constituída de uma grelha no qual portas lógicas dispostas nos pontos de cruzamento permitem estabelecer a conexão de uma entrada a uma saída pela operação de uma única dessas portas (figura 2.17).

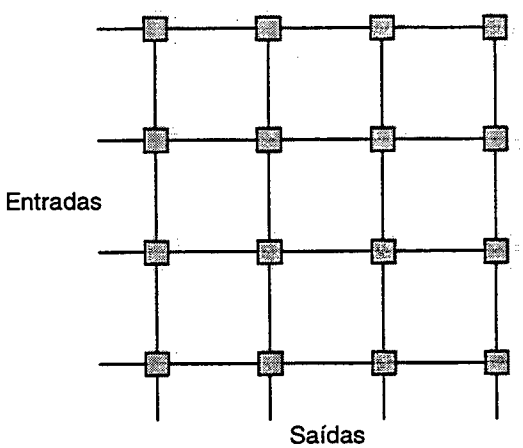


Figura 2.17: Um *crossbar* 4x4.

O tempo mínimo para uma conexão é constante e um *crossbar* NxN permite a conexão de até N pares entrada-saída a cada momento. Na figura 2.17, isso significa que apenas uma porta pode estar fechada em cada linha e cada coluna em um dado instante.

O *crossbar* é uma rede de interconexão muito poderosa mas seu custo aumenta fortemente com o tamanho [BRO83] e os meios tecnológicos atuais ainda limitam sua construção além de certos limites. Um *crossbar* 224x224 como o usado no multicomputador VPP500 da Fujitsu [FUJ92] é considerado muito grande. Entretanto, novas tecnologias óticas podem vir a viabilizar economicamente a construção de *crossbars* de maiores dimensões [SAW87].

A tabela 2.3 resume algumas características das redes de interconexão dinâmicas apresentadas acima.

Tabela 2.3: Características das redes dinâmicas.

Tipo de Rede (Dimensão)	**Complexidade dos circuitos**	**Tempo mínimo para uma conexão**	**Conetividade da rede**
Barramento (N)	$O(N)$	Constante	1 par a cada momento
Múltiplos estágios (NxN, com E estágios)	$O(N \cdot E)$	Proporcional a E	Entre 1 e N pares simultaneamente
Crossbar (NxN)	$O(N^2)$	Constante	N pares simultaneamente

2.2.3 Exemplos

Os primeiros multicomputadores com redes estáticas eram conectados a máquinas hospedeiras por um canal físico único que constituía em um gargalo tanto em relação à carga de programas paralelos quanto à entrada-saída dos dados dos algoritmos executados. A introdução de um nó especializado em entrada-saída, com periféricos acoplados transformou o gargalo de externo em interno. A realização de máquinas com grande poder de entrada-saída embutido é recente e pode ser exemplificada pelo multicomputador Paragon, descrito adiante em 2.2.3.1.

As redes de interconexão dinâmicas têm sido empregadas preferencialmente em multiprocessadores pela natureza das comunicações que se desenvolvem entre os componentes dessas máquinas, caracterizadas pela simplicidade da arbitragem das conexões que devem responder apenas à vontade dos processadores sendo invariavelmente aceitas pelas memórias e pela duração limitada das conexões que envolvem sempre uma operação de leitura ou escrita de uma palavra na memória.

O emprego das redes dinâmicas nos multicomputadores foi preterido durante muito tempo porque, sendo todos os nós dessas máquinas componentes ativos, as comunicações entre nós quaisquer dependem em princípio de acordo mútuo e a duração das conexões também deve ser definida pelos próprios nós. Sendo assim, a arbitragem das conexões é

necessariamente mais complexa nos multicomputadores do que nos multiprocessadores [HWA93]. Simplificações dessas normas gerais, exemplificadas pela aceitação implícita de quaisquer conexões implica em que o nó que as aceita deva estar preparado para tratar conexões indesejadas. O Supernode, descrito adiante em 2.2.3.2, é um exemplo de multicomputador que apresenta uma forma restrita de uso de redes dinâmicas, permitindo que uma mesma máquina seja configurada segundo várias topologias estáticas diferentes em momentos distintos.

Para caracterizar o estado da arte, as arquiteturas de dois multicomputadores modernos, um construído em torno de uma rede estática (o Paragon) e outro em torno de uma rede dinâmica (o Supernode), são apresentados a seguir com algum detalhe. Mais exemplos não acrescentariam muita informação uma vez que os multicomputadores existentes se distinguem mais pelo grau (valores de diâmetro, por exemplo) do que pela natureza (modo de operação, por exemplo) enquanto que o modelo que se pretende introduzir nesta proposta se distingue dos demais pela natureza (modo de operação, principalmente).

2.2.3.1 Paragon

O multicomputador Paragon foi desenvolvido pela Intel como uma máquina experimental [INT91]. Sua arquitetura consta essencialmente de um conjunto de nós unidos através de uma rede de interconexão estática com a topologia de uma grelha retangular. Cada nó inclui em sua composição um roteador de mensagens implementado em *hardware*.

O Paragon é um multicomputador concebido como uma máquina autônoma (que dispensa uma máquina hospedeira) mas que pode ser conectado a uma rede de computadores para que possa ser acessado remotamente. Existem exemplares dessa máquina construídos com até 560 nós.

Os nós do Paragon são distribuídos em quatro seções distintas (figura 2.18): uma com nós de cálculo, uma com nós de serviço e duas com nós de entrada-saída. A seção central, chamada de seção de cálculo, que pode ser considerada o multicomputador propriamente dito, contém nós construídos com microprocessadores i860. Ela fornece o grosso da potência de cálculo da máquina. Nos extremos existem duas seções complementares contendo nós de entrada-saída construídos com microprocessadores i386. O grande número desses nós com seus periféricos associados conferem ao Paragon grande capacidade de entrada-saída. Os nós de serviço formam a seção de diagnósticos do sistema.

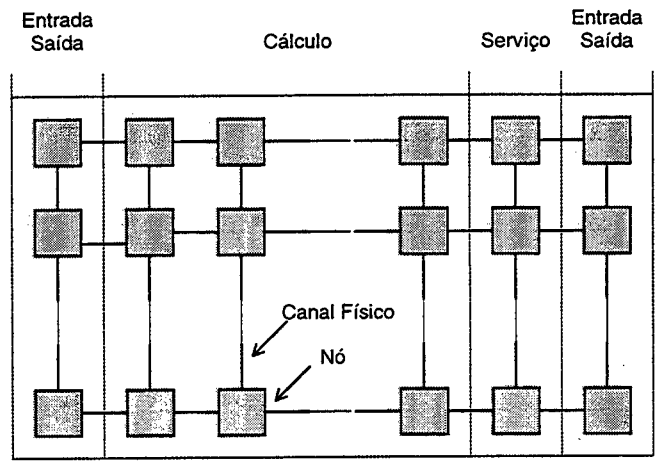


Figura 2.18: Arquitetura do multicomputador Paragon.

Composição dos nós

Cada um dos nós do Paragon é composto de um microprocessador de 32 *bits*, memória privativa e um roteador de mensagens. Os nós de cálculo são compostos de processadores i860 com unidade de ponto flutuante integrada no mesmo *chip*. Os demais nós possuem processadores i386. Um roteador de mensagens — orientado à topologia da grelha — implementado em *hardware* completa cada nó (figura 2.19).

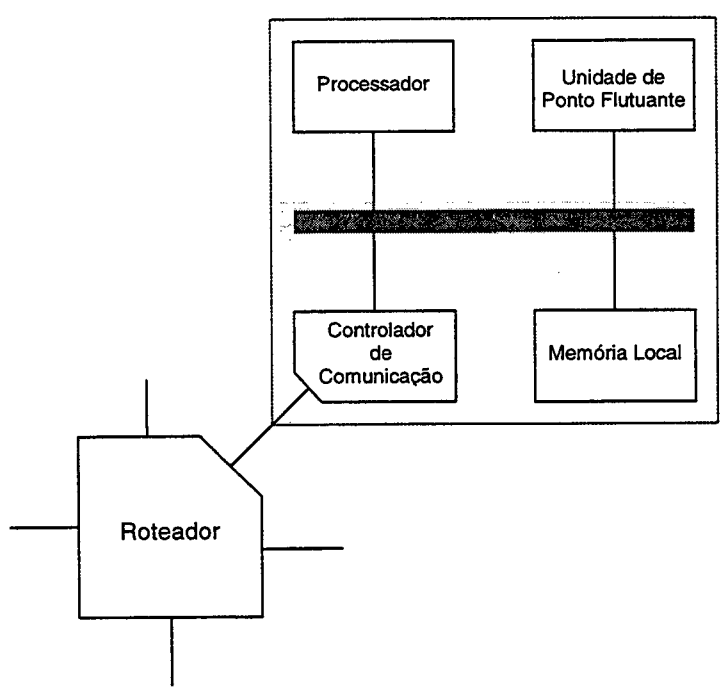


Figura 2.19: Composição de um nó de cálculo.

Roteador de Mensagens

O roteador de mensagens do Paragon é apresentado na figura 2.20. Ele possui 10 canais físicos de comunicação: 5 para recepção e 5 para a emissão de mensagens. Quatro desses pares de canais físicos são para conexões com nós vizinhos da grelha e o outro é para conexão com o nó local.

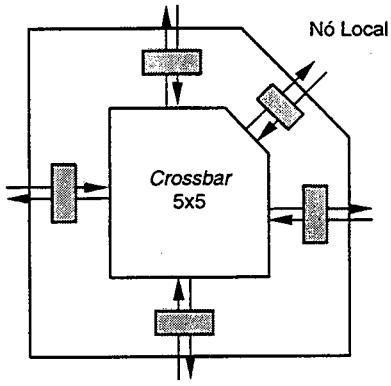


Figura 2.20: Estrutura de um roteador.

Um *crossbar* 5x5 é usado para estabelecer conexões de qualquer canal físico de recepção com qualquer canal físico de emissão. As funções do roteador implementado em *hardware* incluem o roteamento de pacotes em forma de *pipeline* e o tratamento dos *deadlocks* de *buffers* e canais físicos.

2.2.3.2 Supernode

O nome genérico Supernode representa uma família de multicomputadores produzidos pela Parsys [TRE91]. Configurações básicas dessas máquinas são compostas essencialmente por conjuntos de nós unidos através de uma rede de interconexão dinâmica de tipo *crossbar* e de um barramento de controle. Configurações compostas são obtidas conectando configurações básicas por intermédio de *crossbars* suplementares. As redes de interconexão das configurações compostas passam a ser redes de Clos (ver 2.2.2.2). As configurações básicas podem conter até 36 nós enquanto que as maiores podem atingir 1024 nós.

A presente descrição se limita às configurações básicas [WAI90]. O Supernode pode conter até 36 nós distribuídos em três grupos (figura 2.21):

- 1 nó de controle.
- 0 à 3 nós de serviço.
- 16 ou 32 nós de trabalho.

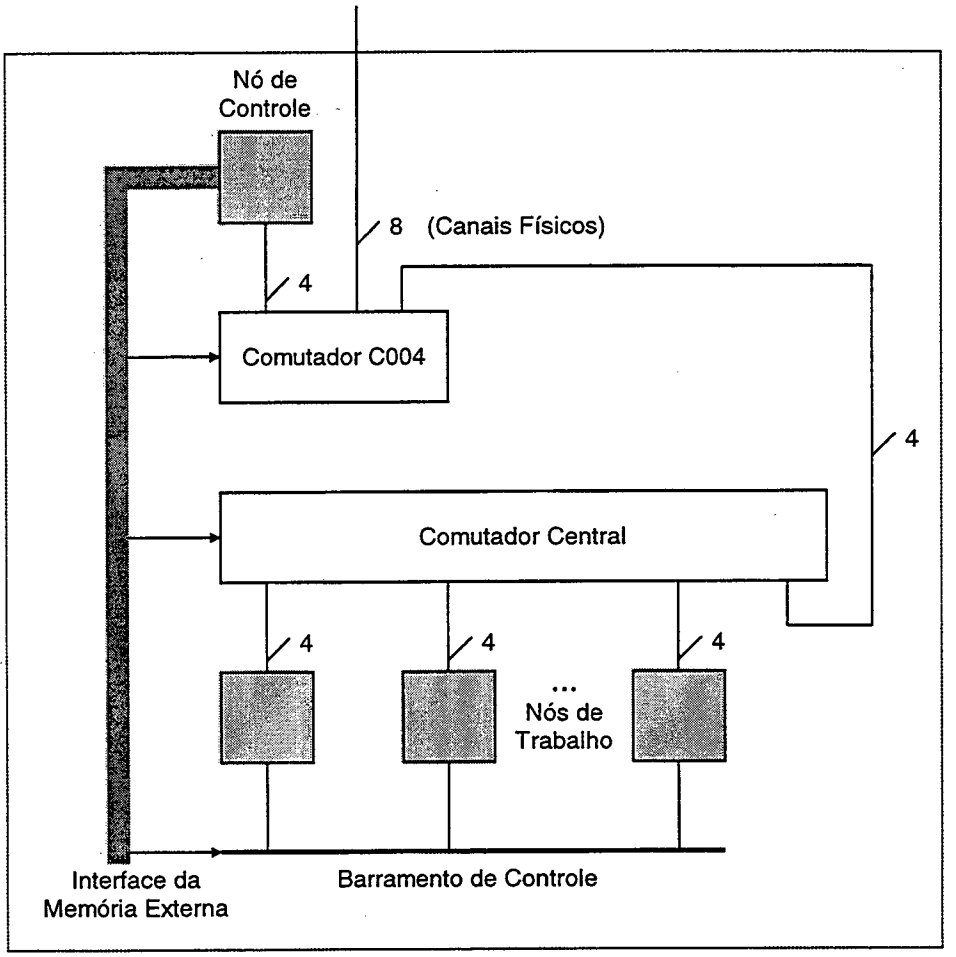


Figura 2.21: Arquitetura do Supernode.

O nó de controle é responsável pela gestão das três principais estruturas da máquina: o barramento de controle, o comutador C004 e o comutador central. Os nós de serviço (ausentes da figura 2.21) são de dois tipos: os servidores de memória (que possuem até 16 Mbytes de memória) e os servidores de disco (que possuem interfaces para periféricos como discos rígidos). Os nós de trabalho fornecem o grosso da potência de cálculo do Supernode. Todos os nós são compostos de um microprocessador (pertencente à família dos Transputers), de uma memória privativa e de canais físicos de comunicação.

Composição dos Nós

O Transputer é um microprocessador concebido especificamente como um componente para a construção de multicomputadores (figura 2.22). Para isso, os diferentes modelos de Transputers dispõem de canais físicos bidirecionais de comunicação serial que podem ser conectados a outros canais físicos do mesmo tipo sem circuitos adicionais. Eles integram no mesmo *chip* um processador de 32 bits, 4 canais físicos bidirecionais de comunicação (numerados de 0 a 3), 2 ou 4 *Kbytes* de memória interna e interfaces para uma memória externa adicional.

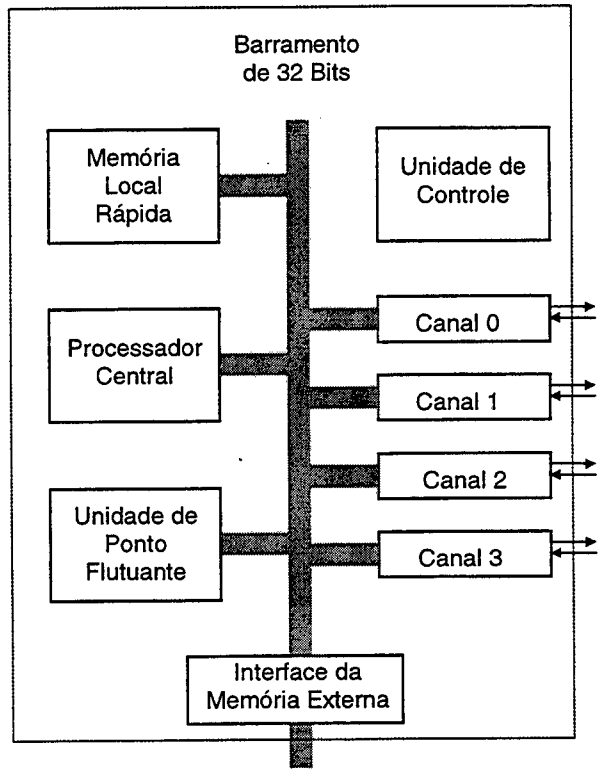


Figura 2.22: Transputer T800.

Os Transputers possuem várias características orientadas à linguagem de programação Occam. As principais características que influem na concepção de um sistema operacional são a existência de um núcleo de gerência de processos implementado em *hardware* e a ausência de mecanismos de proteção de memória e de tradução de endereços.

Esses microprocessadores possuem dois operadores de comunicação síncronos entre processos de Transputers distintos (ou de um mesmo Transputer) [SHE87]: in (channel, length, address) e out (channel, length, address). O operador in é usado para receber pelo canal físico channel uma mensagem de tamanho length no endereço address. O operador out é usado para enviar pelo canal físico channel uma mensagem de tamanho length à partir do endereço address. (Para as comunicações sobre um mesmo Transputer, channel representa um endereço de memória simulando um canal físico).

O Barramento de Controle

O nó de controle está ligado a todos os outros nós da máquina por um barramento de controle cuja interface de comando está conectada ao barramento de memória externa do nó de controle e pode ser operada através de instruções de leitura e escrita em endereços específicos. O barramento de controle oferece as seguintes facilidades:

- Manipulação de diversos sinais de controle de baixo nível (tais como o de *reset*) dos nós de trabalho. (Essa característica permite a gerência centralizada dos nós de trabalho).

- Sincronização de nós independente das oferecidas pelos canais físicos dos Transputers. (Essa característica é importante para a sincronização dos nós envolvidos na reconfiguração da máquina).
- Transmissão de mensagens curtas (1 *byte*) em baixa velocidade entre os nós de trabalho e o nó de controle. (Essa característica é importante para a transmissão códigos de erro, permitindo a centralização do diagnóstico de falhas).

O Comutador C004

O comutador C004 da INMOS é um *crossbar* 32x32. A interface através da qual ele recebe comandos de conexão e desconexão toma a forma de um canal físico de comunicação padrão dos Transputers. Entretanto, no projeto do Supernode, sua interface foi conectada ao barramento de memória externa do nó de controle.

No Supernode, esse comutador é usado para colocar os nós de trabalho em contato direto com o exterior por intermédio dos canais físicos padrões dos Transputers. Para isso, 4 canais físicos do comutador central são conectados ao Comutador C004 e 8 canais físicos desse comutador são levados ao painel frontal da máquina. Todos os canais físicos do nó de controle são ligados ao comutador C004. Isso permite dirigi-los para o comutador central ou para o exterior. Na inicialização do Supernode, o canal físico 0 do nó de controle é dirigido automaticamente para o exterior.

O Comutador Central

O comutador central é um *crossbar* 144x144 que recebe todos os canais físicos dos Transputers de todos os nós de trabalho e de serviço. Sua interface de comando está conectada ao barramento da memória externa do nó de controle.

O Supernode é uma máquina flexível, concebida para operar conforme diferentes redes de topologias estáticas, obtidas pela manipulação do comutador central durante o processo de configuração que antecede seu emprego efetivo. O Supernode, possuindo Transputers com 4 canais físicos cada um, permite a realização de toda rede de interconexão estática de grau 4 ou inferior.

O modo de operação do Supernode é caracterizado por ciclos compostos de uma fase de configuração seguida de uma fase de operação. Em sua composição básica, cada nó tem quatro canais físicos conectados a um *crossbar* que recebe de uma fonte externa, através de um canal físico especial, comados de conexão/desconexão de canais físicos. Além disso, todos os nós também estão conectados a um barramento de controle usado para sincronizar os nós entre fases de configuração. O Supernode opera entre fases de configuração como uma rede estática.

2.3 Suporte de Execução

Neste subcapítulo são tratados de forma independente dois aspectos do suporte de execução: os mecanismos de comunicação e os sistemas operacionais distribuídos. Os mecanismos de comunicação associados às redes de interconexão dos multicomputadores são

descritos em 2.3.1. O modelo básico dos sistemas operacionais distribuídos é descrito em 2.3.2.1 e dois exemplos típicos desse tipo de sistema são caracterizados sucintamente em 2.3.2.2, como exemplos. As seções 2.3.1 e 2.3.2 são autônomas, podendo ser lidas em qualquer ordem.

2.3.1 Mecanismos de Comunicação

Os mecanismos de comunicação são fundamentais nos sistemas de computação formados por múltiplos processadores. Mecanismos associados a dispositivos físicos de comunicação podem contar com suporte de *software* e/ou de *hardware*. A medida que vai se estabelecendo consenso sobre os serviços necessários sobre um dispositivo específico, os mecanismos tendem a ser incorporados ao *hardware*. O subcapítulo 2.2 introduziu as redes de interconexão dos multicomputadores através de uma perspectiva preponderantemente topológica. As características dos mecanismos de comunicação habituais oferecidos nessas redes tanto em *software* quanto em *hardware* são apresentados nesta seção.

Os processos dos programas paralelos executados em nós distintos de um multicomputador devem se comunicar através de mensagens de tamanho variável. Com o objetivo de aumentar o desempenho das comunicações, as unidades de transporte de dados mais empregadas diferem das mensagens. Os mecanismos de comunicação descritos a seguir fazem uso de pacotes e/ou de *flits* (*flow control digits*) como unidades de transporte. Pacotes resultam da decomposição de mensagens e *flits* resultam da decomposição de pacotes. Quando a unidade de transmissão de dados é o pacote, as mensagens são decompostas em um número moderado de pacotes. Quando a unidade de transmissão de dados é o *flit*, os pacotes são decompostos em um número expressivo de *flits*.

As mensagens estão associadas aos programas paralelos assim como os pacotes e os *flits* estão associados aos mecanismos de comunicação. Entretanto, enquanto as mensagens podem variar em tamanho e estrutura entre programas distintos e até mesmo em um mesmo programa, os pacotes e os *flits* possuem tamanhos e estruturas fixos determinados pelo mecanismo de comunicação implementado sobre a rede de interconexão.

2.3.1.1 Redes Estáticas

A comunicação entre dois nós quaisquer de um multicomputador com rede de interconexão estática é dita *ponto-a-ponto*. A comunicação ponto-a-ponto em uma rede estática exige o transporte dos dados ao longo de uma rota formada por uma seqüência de canais físicos diretos, passando por nós intermediários. Os mecanismos envolvidos nesse tipo de comunicação são chamados de *mecanismos de roteamento* [REE87].

Os mecanismos clássicos que permitem o desenrolar simultâneo de várias comunicações em uma rede estática se classificam em comutação de circuitos e comutação de dados. A *comutação de circuitos* consiste na atribuição de canais físicos ao longo de toda rota entre os nós de origem e de destino pelo período de tempo determinado por eles. A *comutação de dados* consiste na atribuição dinâmica de canais físicos, a medida que os dados seguem suas rotas entre os nós de origem e de destino.

Na comutação de circuitos, uma vez estabelecido o circuito, ele permanece intacto até que os nós de origem e de destino decidam desfazer a conexão. O *overhead* de roteamento deve-se somente à atribuição inicial dos canais físicos do circuito. A partir daí, uma certa capacidade de fluxo permanece disponível aos nós de origem e de destino. Quando os circuitos são implementados sem assistência de *hardware*, a unidade de transporte deve ser preferencialmente o pacote de tamanho fixo para simplificar a gerência de *buffers*. Quando os circuitos são implementados com suporte de *hardware* para estabelecer conexões diretas entre os canais físicos que passam pelos nós, a unidade de transporte pode ser a mensagem de tamanho variável porque a gerência de *buffers* nesse caso deixa de existir.

Na comutação de circuitos, se a capacidade de fluxo disponível não é usada integralmente pelos nós de origem e de destino, ela não pode ser aproveitada para outras comunicações. Este fato motiva a comutação de dados. Três tipos de comutação de dados são consideradas: a comutação de mensagens, a comutação de pacotes e o circuito virtual.

Na *comutação de mensagens*, a unidade de transporte transmitida através da rede é a mensagem de tamanho variável. O *overhead* do roteamento é maior do que na comutação de circuitos porque as decisões de roteamento devem ser tomadas individualmente para cada mensagem.

Na *comutação de pacotes* as mensagens são decompostas em pacotes de tamanho fixo que são transmitidos independentemente uns dos outros. Todos os pacotes carregam informações sobre roteamento. Como cada pacote pode seguir uma rota diferente, eles podem chegar ao destino fora de ordem. Assim as mensagens decompostas no nó de origem devem ser ordenadas e recompostas no nó de destino. O *overhead* do roteamento é maior do que na comutação de mensagens porque ele se refere a cada pacote que compõe as mensagens.

A motivação do mecanismo do *circuito virtual* provém da combinação das qualidades da comutação de circuitos e da comutação de pacotes. Assim, um circuito virtual é inicialmente estabelecido entre dois nós, por onde devem trafegar as mensagens transmitidas entre eles em um determinado sentido (o circuito virtual é unidirecional). O roteamento é similar ao da comutação de circuitos porque ele é aplicado somente quando o circuito é estabelecido. (Na realidade um pequeno *overhead* de roteamento é necessário, para tratar do compartilhamento dos canais físicos). Como os pacotes de uma mesma mensagem seguem uma rota fixa, eles chegam ao nó de destino na mesma ordem em que foram emitidos no nó de origem. A capacidade de fluxo é reduzida em relação à comutação de circuito porque os circuitos virtuais compartilham canais físicos.

Além desses mecanismos clássicos, outros mecanismos de roteamento têm sido propostos, como o *virtual-cut-through* [KER79] e o *wormhole* [DAL86]. A unidade de transporte de dados desses mecanismos é o *flit* de tamanho fixo. Todos os *flits* de um mesmo pacote seguem a mesma rota composta por uma seqüência de canais físicos do nó de origem ao nó de destino. Apenas o primeiro *flit* carrega informações sobre roteamento.

No mecanismo *virtual-cut-through*, tão logo um *flit* alcança um nó intermediário, ele pode ser transferido ao seguinte. Quando não existem canais físicos de saída livres para a transmissão do primeiro flit de um pacote, os *flits* seguintes podem ser armazenados nos nós intermediários que devem ter capacidade para acomodar até um pacote inteiro para cada canal físico de entrada.

No mecanismo wormhole, em princípio, *flits* de dois pacotes diferentes não podem compartilhar um mesmo canal físico. Assim que um *flit* alcança um nó intermediário, ele pode ser transferido ao seguinte. Quando não existem canais físicos de saída livres para a transmissão do primeiro *flit* de um pacote, como cada roteador só dispõe de um *buffer* para um *flit* para cada canal físico de entrada, os *flits* seguintes ficam bloqueados nos nós anteriores. Técnicas específicas devem ser empregadas para permitir a multiplexação de canais físicos a nível de *flits* e evitar *deadlocks* [DAL87] [DUA93].

2.3.1.2 Redes Dinâmicas

O processo de atribuição do controle de um canal de comunicação de uma rede dinâmica a um par de dispositivos requisitantes é chamado de *arbitragem*. A comunicação pelo barramento, por exemplo, se dá entre um dispositivo requisitante (dito *mestre*) e um dispositivo requisitado (dito *escravo*). Os dispositivos conectados às redes dinâmicas podem ser de dois tipos: ativos ou passivos. Os *dispositivos ativos* podem agir tanto como mestres quanto como escravos em momentos distintos. Os *dispositivos passivos* só podem agir como escravos. Em um multiprocessador, os processadores são dispositivos ativos e as memórias são dispositivos passivos. Em um multicomputador, os nós são dispositivos ativos.

Os serviços de comunicação oferecidos em sistemas de computação baseados em redes de interconexão dinâmicas contam com grande suporte de *hardware*.

Barramento

A placa do barramento contém uma série de encaixes (*slots*) onde podem ser inseridas as placas dos dispositivos. Cada placa conectada a um barramento pode ser identificada pelo endereço do encaixe. São necessários protocolos específicos (que podem ser tanto síncronos quanto assíncronos) para controlar essas operações. Nos *protocolos síncronos*, as interações respeitam ciclos fixos impostos por um relógio que cadencia as operações que se desenrolam sobre o barramento. Nos *protocolos assíncronos*, a temporização é baseada em confirmações (*handshaking*), dispensando ciclos fixos de relógio.

A arbitragem usa linhas separadas das usadas para as comunicações para permitir que essas atividades possam ocorrer simultaneamente. O processo de arbitragem pode ser centralizado ou distribuído [HWA93].

Na *arbitragem centralizada*, um árbitro central pode receber as requisições dos mestres através de uma linha compartilhada por todos eles ou de linhas independentes para cada um deles. No primeiro caso, uma linha especial é usada para propagar o sinal de atribuição do barramento do primeiro ao último mestre, conforme a ordem de suas identificações. Esse esquema determina uma prioridade fixa pela posição (pelo endereço do encaixe) aos mestres potenciais. No segundo caso, um árbitro central recebe as requisições dos mestres através de linhas independentes para cada um deles. Assim, estabelece-se uma rede estrela entre os dispositivos mestres e o árbitro central. Nesse caso, o escalonamento dos mestres não precisa se restringir a prioridades fixas.

Na *arbitragem distribuída*, cada mestre é equipado com um árbitro identificado através de um número único (usado para resolver conflitos). Todos os mestres enviam seus números de árbitro através de linhas de requisição compartilhadas e depois comparam o

número resultante com o seu próprio número. Os mestres cujos números de árbitro forem menores que o número resultante desistem do acesso ao barramento durante o próximo ciclo. Findo o ciclo atual, apenas o mestre vencedor obtém acesso ao barramento. Esse esquema é baseado em prioridades.

Crossbar

As conexões dos *crossbars* podem ser manipuladas por controladores externos ou internos.

Os *controladores externos* são dispositivos conectados aos *crossbars* através de interfaces específicas que emitem comandos de configuração (para fechar uma porta lógica, por exemplo). Esses controladores respondem em geral a requisições decididas globalmente por *software*. Os *controladores internos* estão embutidos nos circuitos dos pontos de cruzamento e respondem às requisições emitidas pelos dispositivos mestres. Nesse caso, o comportamento de cada ponto de cruzamento é semelhante ao de um barramento, sendo o controlador seu árbitro.

Os *crossbars* vêm sendo usados tanto em multiprocessadores [YOU94] quanto em multicomputadores [BEL94].

Nos multiprocessadores, eles servem como redes de acesso à memória. As conexões são estabelecidas por demanda dos processadores para cada acesso individual requisitado. Por questões de desempenho, cada canal deve comportar a transmissão simultânea de *bits* de controle, endereço e dados. Assim, a complexidade de cada ponto de cruzamento eqüivale a de um barramento usado com os mesmos propósitos. Os controladores são internos. Em máquinas comerciais, apenas pequenas redes de até 16x16 têm sido empregadas.

Nos multicomputadores, eles servem como redes de comunicação para troca de mensagens entre nós. Cada canal pode se resumir, no limite, a uma linha serial. A complexidade de cada ponto de cruzamento é pequena. Os controladores podem ser tanto internos quanto externos. Em máquinas comerciais, redes superiores a 200x200 já foram empregadas [HWA93].

Nos multicomputadores com dispositivos de controle internos, as conexões são estabelecidas por demanda do nó emissor pela duração de um único pacote de tamanho fixo [HEE93]. Em multicomputadores com dispositivos de controle externos, as seguintes alternativas são possíveis:

- A construção prévia à colocação do sistema em estado de execução efetiva de uma rede de interconexão fixa para reproduzir a situação de uma rede estática.
- A mudança dinâmica da topologia da rede física para a adaptá-la às redes lógicas dos programas em tempo de execução.

Em ambos os casos, o controlador deve receber de alguma fonte as informações sobre a configuração da rede. Um exemplo do emprego da primeira dessas alternativas é dado pelo Supernode (visto em 2.2.3.2) que executa um programa de configuração antes de cada fase de execução de um programa paralelo.

Redes com Múltiplos Estágios

Nas redes de múltiplos estágios, várias portas lógicas devem ser fechadas para estabelecer uma conexão entre um par de nós. Os serviços de comunicação oferecidos em sistemas de comunicação baseados em redes com múltiplos estágios contam com grande suporte de *hardware*. A lógica de arbitragem dessas redes está embutida na própria rede.

O estabelecimento de um caminho para conectar os correspondentes precede comunicação efetiva. Antes de disparar a comunicação, é necessário esperar a propagação desde a entrada até a saída do sinal portador da solicitação de conexão. A partir do estabelecimento da conexão, os recursos necessários são atribuídos permanentemente até o fim da comunicação e o único atraso interveniente é devido ao tempo de propagação dos dados.

Os mecanismos para a escolha do caminho dependem das características da rede. Na rede ômega, por exemplo, o endereço binário do correspondente fornece a chave para a escolha determinística do caminho único até ele.

No caso particular de um *crossbar*, entendido como uma rede dinâmica de um único estágio, uma conexão entre dois correspondentes quaisquer se faz pelo fechamento de uma única porta lógica. Os atrasos provocados pela etapa de conexão que precede toda comunicação podem ser importantes no caso de múltiplos estágios mas são reduzidos de maneira considerável no caso de um único estágio.

As redes de múltiplos estágios vêm sendo usadas em multiprocessadores e multicomputadores. As redes não-bloqueantes podem ser usadas facilmente. Entretanto, dificuldades operacionais decorrem do uso de redes rearranjáveis e bloqueantes [WU 92].

O mecanismo de roteamento *wormhole*, introduzido em 2.3.1.1 associado às redes estáticas também, podem ser aplicados às redes dinâmicas de múltiplos estágios, substituindo-se as portas lógicas por roteadores mais complexos, funcionalmente equivalentes aos empregados nas redes estáticas [NI 97]. Nas redes estáticas, os roteadores estão associados aos nós enquanto que nas redes dinâmicas eles formam uma estrutura independente dos nós.

Rede por Difusão

Para ser completo, inclui-se aqui os serviços de comunicação das redes de computadores convencionais embora esse meio de comunicação raramente seja utilizado em multicomputadores. Uma exceção é representada pela máquina iPSC da Intel.

Os protocolos de comunicação das redes por difusão são baseados em *software*. Os métodos de compartilhamento de uma rede local por difusão podem ser estáticos ou dinâmicos [TAN88]. No *compartilhamento estático*, cada emissor conhece o modo de utilizar a via sem provocar colisões. O *compartilhamento dinâmico* é classificado em compartilhamento por competição e compartilhamento por eleição.

No *compartilhamento por competição*, cada emissor é livre para escolher o momento de enviar uma mensagem sem se preocupar com os outros emissores. Nessas condições, ele deve estar preparado para reemitir quando as colisões acontecerem. Para evitar que colisões se sucedam, pode-se recorrer a soluções como fazer cada emissor esperar um tempo diferente antes de uma nova emissão.

No *compartilhamento por eleição*, cada emissor deve solicitar a utilização da via através de uma requisição especial e esperar sua eleição que pode ser feita por consulta ou por seleção. O compartilhamento por eleição pode ser realizado de maneira centralizada ou distribuída. Na forma centralizada, um árbitro único trata as requisições. No modo distribuído, os emissores escolhem eles mesmos o eleito

No *compartilhamento por consulta*, os estados de todos os emissores são examinados em uma ordem fixa sem relação com a ordem das solicitações. A consulta se interrompe com a detecção do primeiro emissor pronto. Dois modos comuns de consulta são:

- A lista dos emissores é percorrida sempre na mesma ordem desde o início.
- A lista de emissores é percorrida circularmente: ao final de um ciclo de consulta, marca-se o ponto de parada a partir do qual deverá começar o próximo ciclo.

No primeiro caso a ordem da lista estabelece a prioridade dos emissores. O segundo é mais justo e evita toda possibilidade de privação (*starvation*).

No *compartilhamento por seleção* as requisições são memorizadas. Quando a via está livre, um emissor é escolhido segundo um critério qualquer. As requisições são mensagens que podem ser transmitidas sobre uma via especializada ou sobre a própria via compartilhada (e nesse caso, segundo um modo de compartilhamento particular).

2.3.2 Sistemas Operacionais

2.3.2.1 Estrutura

Sistemas operacionais nativos para multicomputadores são raros. Na maioria das vezes essas máquinas estão acopladas a uma estação de trabalho hospedeira que fornece todo ambiente de desenvolvimento das aplicações e a maior parte dos serviços. O ambiente de execução se reduz nesses casos a um conjunto de rotinas de biblioteca incorporadas ao código dos programas paralelos.

A falta inicial de sistemas nativos, originou a busca de inspiração nos sistemas operacionais distribuídos. O ambiente físico objeto desses sistemas, embora só admitam interações eficientes de granulosidade grossa, possuem certos aspectos que podem ser adaptados para os ambientes dos multicomputadores. Os aspectos desses sistemas que mais interessam a este trabalho dizem respeito à organização global.

Os sistemas operacionais distribuídos são em geral organizados segundo o modelo cliente-servidor, nos quais os processos de usuário solicitam serviços dos processos do sistema através de chamadas de procedimentos remotos. Um núcleo garante apenas os mecanismos básicos para que os serviços do sistema possam ser prestados por processos ordinários. Os processos servidores podem ser organizados internamente com um ou mais fluxos de execução.

Chamadas de Procedimentos Remotos

O modelo de programação cliente-servidor e a disciplina de comunicação da chamada de procedimento remoto foram introduzidas em 2.1.2.4 no contexto das linguagens paralelas. Aqui esses conceitos são retomados no âmbito dos sistemas operacionais distribuídos.

As chamadas de sistema da interface de programação de um sistema operacional podem ser apresentadas segundo a sintaxe de uma linguagem de programação convencional. Elas são empregadas como chamadas de procedimentos regulares, escondendo o mecanismo de interação com o sistema. Nas implementações tradicionais o acesso ao sistema se faz por uma instrução de chamada ao supervisor; nas distribuídas, o modelo consagrado é o das chamadas de procedimentos remotos [BIR84].

Nos sistemas operacionais distribuídos organizados segundo o modelo cliente-servidor [GOS91], processos de usuário (clientes) solicitam serviços ao sistema através de mensagens enviadas a processos do sistema (servidores) e recebem o resultado do serviço em mensagens de resposta.

Um cliente envia ao servidor uma mensagem de requisição e espera uma mensagem de resposta para cada serviço solicitado. Um servidor implementado como um processo seqüencial repete indefinidamente o seguinte ciclo: espera uma mensagem de requisição de um cliente qualquer, identifica o serviço solicitado, realiza o serviço chamando o procedimento responsável pela sua execução e envia uma mensagem de resposta. A realização desse modelo depende de um núcleo que implemente processos e mecanismos de comunicação.

No contexto dos sistemas distribuídos, a disciplina de comunicação da chamada de procedimento remoto (vista em 2.1.2.4) resulta da associação do esquema de controle da chamada de procedimento convencional e do mecanismo de comunicação do modelo cliente-servidor com o objetivo de esconder dos clientes o mecanismo de troca de mensagens. Essa associação é mostrada na figura 2.23 pela interposição entre o cliente e o servidor de interfaces compostas de procedimentos locais (os *stubs*) incumbidos das trocas de mensagens embutidas no mecanismo [TAN92].

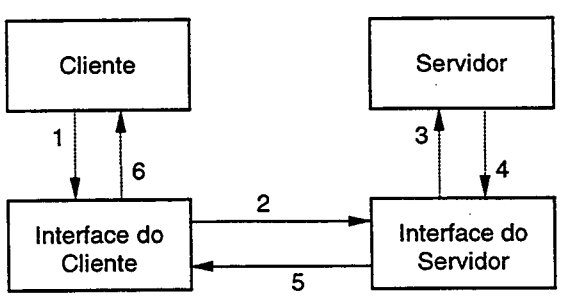


Figura 2.23: Chamada de procedimento remoto.

A realização das chamadas de procedimentos remotos levanta vários problemas. Alguns deles, como a construção dos *stubs*, a transmissão de parâmetros, a identificação dos correspondentes e a composição das mensagens — todos examinados a seguir — são de

caráter geral. Outros, como as diferenças entre formatos de dados em máquinas distintas e a semântica das chamadas na presença de comunicações não confiáveis, são de interesse mais específico dos sistemas de computação distribuídos e não são abordados neste texto.

Os *stubs* podem ser construídos manualmente ou automaticamente por um compilador especializado. O método manual é mais conveniente na produção de uma interface única estável com número limitado de procedimentos. A construção de um compilador se justifica quando interfaces podem ser definidas e alteradas freqüentemente por usuários na construção de aplicações distribuídas. Em qualquer caso, os *stubs* são reunidos em uma biblioteca acessível aos clientes.

A transmissão de parâmetros por valor se adapta bem ao mecanismo de chamada de procedimento remoto. Entretanto, a passagem de parâmetros por referência é de difícil tratamento em função da independência dos espaços de endereçamento dos processos, sobretudo entre máquinas diferentes. A substituição da passagem por referência para a passagem por armazenamento e restituição deve ser empregada com precaução porque essa solução pode apresentar anomalias em algumas situações [TAN92].

Para que a chamada do procedimento remoto seja transparente ao cliente, ele não deve preocupar-se com a identificação do servidor. Isso só pode ser alcançado no caso de servidores padrões do sistema que possuam identificação implícita conhecida. No caso de servidores definidos pelos usuários, um mecanismo adicional explícito deve ser executado para identificar o servidor.

Do ponto de vista do mecanismo de comunicação, uma mensagem pode ser tratada como uma seqüência não estruturada de *bytes*. Para permitir sua identificação durante o trânsito, um cabeçalho pode ser acrescido à mensagem na origem e retirado no destino. Esse cabeçalho possui em geral tamanho fixo e pode conter, entre outras, as seguintes informações de controle: a identificação do emissor, a identificação do destinatário e o tamanho do corpo da mensagem. Assim, a mensagem realmente transmitida é composta de duas partes (figura 2.24): o cabeçalho e o corpo (que constitui a verdadeira mensagem trocada pelos correspondentes).

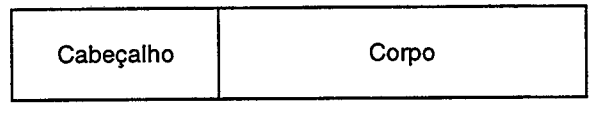


Figura 2.24: Composição de uma mensagem.

As mensagens possuem estruturas internas que devem ser conhecidas tanto pelos clientes quanto pelos servidores. Entretanto, eles podem trocar mensagens de estruturas diferentes. Em função disso, o corpo de uma mensagem pode ser composto de duas partes (figura 2.25): um descritor (contendo informações sobre o número e os tipos dos dados) e os próprios dados. Essa estrutura não precisa ser conhecida pelo serviço de comunicação.

Figura 2.25: Corpo de uma mensagem.

2.3.2.2 Fluxos de Execução

Nos sistemas operacionais distribuídos modernos, os processos são entendidos como as unidades de atribuição de recursos que admitem vários *fluxos de execução* (*threads*) internos [COU94]. Os fluxos de execução de um processo compartilham todos os seus recursos, notadamente, o espaço de endereçamento e os arquivos. Cada fluxo de execução possui, entretanto, essencialmente seu próprio ponteiro de instruções, sua própria pilha e seu próprio estado.

Fluxos de execução constituem um mecanismo para simplificar a expressão do paralelismo em servidores que dependem de chamadas de sistemas bloqueantes. Se a execução de algumas tarefas do servidor dependem de outros componentes do sistema, pode ser interessante aproveitar o tempo ocioso para atender outros clientes. Assim, quando recebe uma requisição, o servidor a examina e se não pode atendê-la imediatamente, ao invés de bloquear, registra o estado da requisição pendente em uma tabela interna e passa a atender a próxima. Mais tarde, quando a requisição puder ser satisfeita, o servidor acessa a tabela e completa o serviço.

Com o uso de fluxos de execução, a seguinte organização para o servidor é possível. Um fluxo de execução recebe uma requisição, ativa um fluxo de execução ocioso ou dispara um novo fluxo de execução para atendê-la e volta a esperar pela requisição seguinte. Cada fluxo de execução executa uma tarefa seqüencialmente e fica bloqueado se necessário. A programação do servidor é mais simples nesse caso.

De uma forma geral, as facilidades oferecidas pelos processos com vários fluxos de execução permitem a exploração do paralelismo por parte do sistema operacional ou das aplicações segundo três esquemas: um único processo com vários fluxos de execução compartilhando memória, vários processos trocando mensagens ou a combinação dos dois esquemas anteriores.

2.4 Conclusões

Pela natureza do trabalho descrito neste texto, que visa principalmente propor um modelo de multicomputador e seu mecanismo de comunicação como uma solução articulada respondente aos *problemas básicos* identificados em 1.1, o estudo desenvolvido para amparálo explorou os temas das arquiteturas de multicomputadores apresentado e os serviços de comunicação associados. O estudo de linguagens paralelas clássicas apoiadas em trocas de mensagens e o de organizações de sistemas distribuídos pretende buscar subsídios para validar

o modelo proposto, através da demonstração de seu emprego na implementação desses elementos.

3 Ambiente Multicomputador Crux

O objetivo principal deste capítulo é o de apresentar os componentes básicos da proposta de um ambiente multicomputador para a execução de redes de processos comunicantes, representados pela arquitetura do multicomputador Crux e seu mecanismo de conexão dinâmica de canais físicos. A apresentação desses componentes é precedida pelo detalhamento das motivações e dos objetivos orientadores deste trabalho.

3.1 Motivações

As motivações para a concepção articulada dos elementos de *software* e de *hardware* de um ambiente multicomputador resultaram de uma série de observações colhidas durante o desenvolvimento do projeto de um sistema operacional para um multicomputador convencional com rede de interconexão estática. Elas já foram genericamente introduzidas no capítulo 1 e são aqui reexaminadas e precisadas à luz do capítulo 2.

3.1.1 Linguagens e Arquiteturas

As linguagens de programação foram classificadas em 2.1.1 e as arquiteturas de computadores em 2.2.1. Da confrontação dessas classificações, estabelecida pela sobreposição das árvores das figuras 2.1 e 2.7, procurando acomodar as linguagens da forma mais natural sobre as arquiteturas, resulta a figura 3.1.

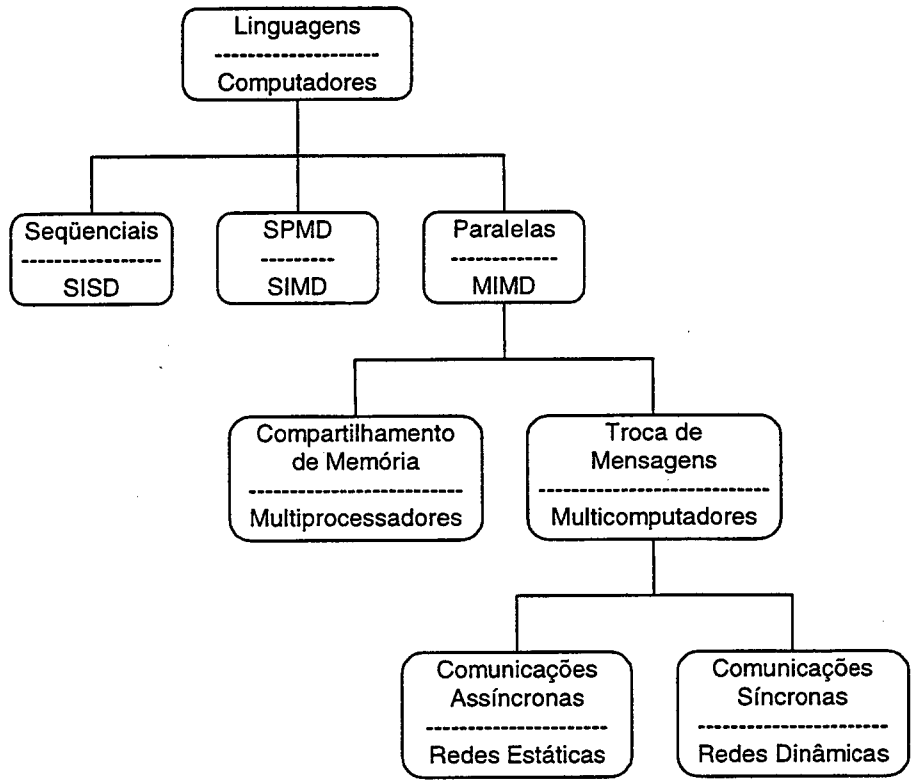


Figura 3.1: Linguagens e arquiteturas.

A figura 3.1 expõe a existência de uma correspondência estreita entre os modelos lógicos dos programas e os modelos físicos dos computadores. As correspondências do primeiro nível à partir da raiz são evidentes. As correspondências do segundo nível foram comentadas em 1.1. A correspondência estabelecida no terceiro nível entre as redes estáticas e as comunicações assíncronas resulta da impossibilidade de transferir mensagens diretamente entre os espaços de endereçamento dos processos comunicantes quando eles não executam em nós vizinhos. Por outro lado, a correspondência entre as redes dinâmicas e as comunicações síncronas resulta da possibilidade de transferência de mensagens diretamente entre os espaços de endereçamento dos processos comunicantes enquanto os nós onde eles executam estão diretamente conectados.

As linguagens de programação são concebidas segundo os mais diferentes objetivos que podem priorizar a execução eficiente em determinadas arquiteturas ou a facilidade de programação de determinadas aplicações, entre outros critérios das mais diversas naturezas.

Em muitos casos, a independência da arquitetura pode ser um critério importante. Entretanto, as arquiteturas impõem um modo de operação que deve ser respeitado pelos programas em execução efetiva sobre os computadores. Assim, por exemplo, qualquer que seja o modelo de programação de uma determinada linguagem, um programa paralelo que executa em um multicomputador necessita que dados transitem entre seus nós para que processos que executam em nós diferentes possam cooperar na execução de uma tarefa comum. Se a linguagem segue o modelo de troca de mensagens, o programador da aplicação se ocupa diretamente com o envio e a recepção das mensagens. Se ela segue o modelo de

compartilhamento de memória, o programador é liberado de tratar diretamente com mensagens que entretanto devem de qualquer forma fluir entre os nós.

A percepção da acomodação natural entre as trocas de mensagens síncronas e as arquiteturas de multicomputadores com redes de interconexão dinâmicas instigou a motivação de reunir esses elementos em um mesmo ambiente multicomputador.

3.1.2 Adaptação das Redes Físicas às Redes Lógicas

Pesquisadores envolvidos com projetos de linguagens de programação paralela têm expressado a conveniência de construir as redes físicas de multicomputadores para corresponder às redes lógica de processos comunicantes [HOA85] [HAN94a]. Os principais conceitos tratados pelos programadores de redes de processos comunicantes são processos, canais lógicos e mensagens de tamanho variável. Eles estão preocupados principalmente com a criação tanto de processos quanto de canais lógicos e com a transmissão de mensagens completas de tamanho variável entre processos. Nesse contexto, um mecanismo de roteamento, por exemplo, pode ser visto como um procedimento inconveniente, necessário para superar as discrepâncias entre uma rede de processos comunicantes e o multicomputador no qual ela deve executar.

Por outro lado, pesquisadores envolvidos com projetos de arquiteturas de multicomputadores, considerando impraticável a construção de máquinas que possam assumir todas as topologias concebíveis para os programas paralelos, deixam de considerar que redes de interconexão reconfiguráveis possam assumir topologias de muitas redes de processos comunicantes de grande importância prática.

É comum ler-se em textos sobre processamento paralelo referências admitindo que o desenvolvimento do *software* não tem conseguido acompanhar o extraordinário desenvolvimento do *hardware*, atestado pela proliferação de diferentes arquiteturas de multicomputadores tanto com redes estáticas quanto com redes dinâmicas [CAR89]. A opinião aqui emitida vai em sentido oposto, ao considerar que as arquiteturas propostas são a fonte dos problemas que frustram os anseios dos projetistas de *software*. Os projetistas dos serviços de comunicação são obrigados a tratar com problemas complexos que poderiam muitas vezes ser evitados se fosse dada maior atenção à integração entre os projetos de *hardware* e de *software*.

Em relação às redes de interconexão estáticas, os problemas de roteamento identificados em 2.3.1.1 constituem as preocupações centrais dos serviços de comunicação. Nesse sentido, procura-se definir topologias que possuam as melhores propriedades geométricas para diminuir ou facilitar o tráfego de mensagens. Além disso, a construção de roteadores implementados em *hardware* visam otimizar o tratamento de mensagens em nós intermediários, mesmo que essas soluções aumentem a complexidade dos nós.

Por outro lado, as redes de interconexão dinâmicas como o *crossbar* com controle interno se adaptam melhor aos multiprocessadores, onde as comunicações entre os componentes físicos envolvem sempre um elemento ativo (um processador) e um elemento passivo (um módulo de memória). Os inconvenientes de emprego desses dispositivos nos multicomputadores são de dupla ordem:

- Os nós dos multicomputadores são elementos ativos que, em princípio, só deveriam ser conectados por acordo mútuo. Num *crossbar* com controle interno, uma conexão é estabelecida pela solicitação do nó emissor e à revelia do nó receptor, introduzindo a necessidade de tratamento de conexões indesejadas.
- As conexões têm existência efêmera, em princípio pela duração da transmissão de uma único pacote de tamanho fixo. Num *crossbar* com controle interno, uma conexão é estabelecida como parte do procedimento de envio de um pacote. Assim, o nó receptor deve estar preparado para tratar uma seqüência de pacotes que pode representar um entrelaçamento de mensagens distintas, provenientes de diversos nós.

Uma exceção a este estado geral foi vislumbrada a partir do Supernode. Conforme visto em 2.2.3.2, um programa de configuração, antecedente à operação normal dessa máquina, deve ser executado para construir uma rede de interconexão estática qualquer, limitada ao grau de seus nós. O objetivo do projeto do Supernode foi o de permitir a construção de topologias regulares clássicas como as apresentadas em 2.2.2.1. Entretanto, uma experiência heterodoxa desenvolvida sobre essa máquina acenou com a possibilidade — não prevista inicialmente — de reconfiguração dinâmica, cuja eficácia dependia de alterações no projeto original da arquitetura do Supernode [COR89].

Do citado acima, resultou a motivação de reconfigurar a rede física em tempo de execução para corresponder exatamente à rede lógica de processos comunicantes à medida em que os processos e canais lógicos fossem criados e destruídos.

3.1.3 Exploração do Paralelismo Físico

O paralelismo físico disponível nos multicomputadores pode ser explorado de muitas formas:

- *Programas paralelos* - Um programa paralelo pode ser concebido como uma rede de processos comunicantes expressando o paralelismo da arquitetura explicitamente através de linguagens especiais ou de rotinas de biblioteca específicas.
- *Linguagens paralelas* - Numerosas linguagens que permitem a descrição explícita de programas paralelos através de redes de processos comunicantes têm sido freqüentemente propostas (à exemplo das linguagens citadas em 2.1.3). A implementação dessas linguagens pode ser facilitada quando o suporte de execução fornece operadores de comunicação adequados aos modelos propostos por elas.
- *Extração do paralelismo implícito* - Os compiladores podem reconhecer e extrair o paralelismo implícito de programas escritos em linguagens seqüenciais convencionais, gerando programas executáveis na forma de redes de processos comunicantes.

- *Compiladores* - O próprio compilador pode tomar a forma de uma seqüência linear de processos comunicantes, um para cada uma de suas fases (correspondentes, por exemplo, à análise léxica, à análise sintática, à análise semântica e à geração de código).
- *Núcleos de sistemas operacionais* - O núcleo de um sistema operacional pode ser replicado pelos nós dos multicomputadores.
- *Sistemas operacionais* - Um sistema operacional pode ser composto por servidores fisicamente distribuídos pelos nós dos multicomputadores.
- *Sistemas de arquivos* - Um sistema de arquivos pode ser concebido como uma coleção de servidores e implementado como uma rede de processos comunicantes com grande autonomia em relação ao resto do sistema operacional.

Dessa lista de possibilidades, resulta a motivação de privilegiar a execução de programas paralelos expressos como redes de processos comunicantes, resultantes da decomposição funcional de um problema em um conjunto de processos que respeitem os níveis de granulosidade computacional média adequados aos multicomputadores, incentivando a atribuição desses processos sempre a nós distintos. Por outro lado, exclui-se explicitamente das aplicações visadas por este trabalho, programas de granulosidade fina de qualquer natureza.

3.1.4 Mecanismos de Comunicação

Nesta seção, são inicialmente precisados os conceitos de comunicação direta e de mapeamento perfeito. A seguir, os diversos mecanismos de comunicação introduzidos em 2.3.1 são confrontados com a comunicação direta e o mapeamento perfeito em relação aos seguintes itens: unidade de transporte, escolha de rota, tempo em trânsito mínimo e gerência de *buffers*. O objetivo de conduzir esta comparação neste ponto é o de ressaltar a importância da comunicação direta e do mapeamento perfeito como uma das motivações fundamentais deste trabalho.

Comunicação Direta e Mapeamento Perfeito

Uma comunicação direta entre dois processos executando em nós diferentes de um multicomputador se refere à transmissão de uma mensagem através de um canal lógico síncrono dedicado atribuído a um único canal físico dedicado pela duração da transmissão completa da mensagem, pelo menos.

Mapeamento de processos se refere à atribuição de processos a nós e *mapeamento de canais lógicos* se refere à atribuição de canais lógicos a canais físicos. *Mapeamento de redes de processos comunicantes* designa a atribuição de todos os processos e canais lógicos de uma rede de processos comunicantes a nós e canais físicos de um multicomputador.

O mapeamento de um processo é dito *perfeito* quando a ele é atribuído com exclusividade um nó pela duração exata do processo.

Cada um dos mecanismos de comunicação vistos em 2.3.1 envolve um método específico de mapeamento de canais lógicos. Assim, na comutação de mensagens, por

exemplo, para cada mensagem transmitida, um canal lógico é atribuído sucessivamente à seqüência de canais físicos que compõe a rota da mensagem e a atribuição a cada canal físico da seqüência dura apenas o tempo necessário ao trânsito da mensagem através dele.

O mapeamento de um canal lógico dedicado é dito *perfeito* quando a ele é atribuído com exclusividade um canal físico dedicado pela duração exata do canal lógico.

O mapeamento de uma rede de processos comunicantes é dito perfeito quando o mapeamento de todos seus processos e canais lógicos forem perfeitos. Nessas condições, seguindo as convenções introduzidas no capítulo 2, os desenhos das redes de processos comunicantes que empregam canais lógicos síncronos bidirecionais dedicados únicos (conforme visto em 2.1.2) e os dos multicomputadores (conforme visto em 2.2.1) são imediatamente assimiláveis.

Unidade de Transporte

Uma vez que a mensagem de tamanho variável representa a unidade lógica através da qual os processos trocam informações, sua decomposição em pacotes e *flits* serve apenas aos propósitos dos mecanismos de comunicação. A manipulação de pacotes e *flits* é inconveniente porque envolve procedimentos, de complexidade variável conforme o mecanismo considerado, que sem exceção reduzem o desempenho final do transporte de mensagens entre processos, se comparado à comunicação direta através de mensagens completas.

A manipulação de pacotes envolve pelo menos os procedimentos de decomposição da mensagem no nó de origem e de sua recomposição no nó de destino. Todos os pacotes de uma mesma mensagem costumam ser enviados em uma seqüência ordenada. Nas redes dinâmicas, esses pacotes chegam em ordem. Na comutação de pacotes e nos mecanismos *virtual-cut-through* e *wormhole*, os pacotes de uma mesma mensagem podem não chegar ordenados se cada um deles puder seguir uma rota distinta. Além disso, pacotes de mensagens distintas destinadas ao mesmo nó podem chegar intercaladas tanto em redes estáticas quanto em redes dinâmicas porque os mecanismos de transporte de pacotes são invariavelmente acionados pelo nó emissor e à revelia do nó receptor.

Os pacotes provocam ainda um fenômeno de fragmentação que resulta da decomposição das mensagens de tamanho variável em pacotes de tamanho fixo, fazendo com que o último pacote transporte em média um número útil de dados equivalente à metade de seu tamanho. O efeito negativo desse fenômeno é tão mais importante quanto menor for a relação entre o tamanho médio das mensagens e o tamanho dos pacotes.

Na comunicação direta, a transmissão de mensagens de tamanho arbitrário como blocos monolíticos usa com exclusividade toda a capacidade de fluxo do canal físico e no mapeamento perfeito, a disponibilidade de todos os canais físicos é garantida e imediata.

Escolha de Rota

Todos os mecanismos de comunicação para as redes de interconexão estáticas incluem algoritmos específicos para a escolha de rotas a serem seguidas por mensagens, pacotes e *flits* entre o nó de origem e o nó de destino.

As políticas de roteamento utilizadas nas redes estáticas podem ser fixas ou adaptativas. As *políticas fixas* não levam em conta as variações de tráfego em tempo de

execução. As rotas que devem ser seguidas pelas mensagens trocadas entre cada par de nós é determinada previamente e informações sobre elas são disseminadas pelos nós quando o multicomputador é inicializado. As *políticas adaptativas* tomam suas decisões em função das características dinâmicas do tráfego. As políticas fixas são adequadas às situações de tráfego conhecido e regular. As políticas adaptativas podem ser mais convenientes quando existem grandes variações de tráfego. Além disso, políticas adaptativas podem assegurar caminhos alternativos em caso de falhas de nós ou canais físicos. Entretanto, a complexidade de implementação das políticas adaptativas é superior ao das políticas fixas.

Os canais físicos que conectam diretamente dois nós de um multicomputador são considerados como vias de comunicação diretas confiáveis. Isso não é suficiente para garantir comunicações ponto-a-ponto confiáveis em uma rede estática. De fato, a confiabilidade de uma rede estática pode ser comprometida pela falha de um único nó no caso de algoritmos de roteamento fixos, por exemplo.

Os mecanismos de comunicação para as redes de interconexão dinâmicas com múltiplos estágios também devem incluir algoritmos para a escolha de rota entre o nó de origem e o nó de destino. No *crossbar*, a existência de uma única conexão possível entre dois nós torna esse algoritmo elementar. Diferentemente dos mecanismos para as redes estáticas, uma vez estabelecidas as conexões da rota, a mensagem flui diretamente entre os nós de origem e de destino. O mesmo vale para a comutação de circuitos com suporte de *hardware* nas redes estática.

Na comunicação direta, a questão de escolha de rota nem mesmo se coloca e o mapeamento perfeito estende essa característica a toda rede de processos comunicantes.

Tempo em Trânsito Mínimo

O tempo em trânsito de uma mensagem é medido desde o início de sua transmissão no nó de origem até a sua completa recepção no nó de destino. O tempo em trânsito é muitas vezes a medida que melhor expressa o desempenho geral das redes físicas dos multicomputadores e dos mecanismos de comunicação. O tempo em trânsito mínimo é calculado sempre considerando a rede completamente desobstruída, quando não existe disputa pelo uso de canais físicos.

A figura 3.2 apresenta um exemplo de decomposição de uma mensagem em pacotes e *flits*.

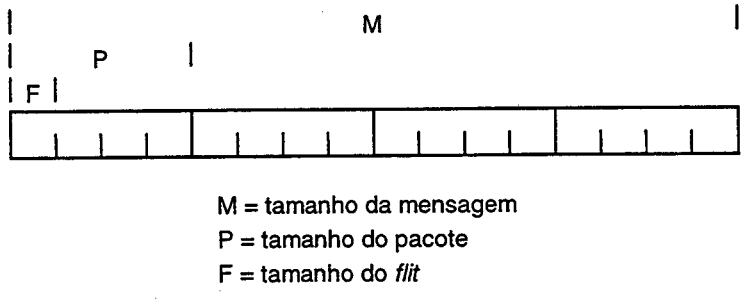


Figura 3.2: Mensagem, pacotes e *flits*.

Para os mecanismos de comunicação referidos em 2.3.1, os tempos em trânsito mínimos podem ser calculados conforme as fórmulas apresentadas na tabela 3.1. Nessas fórmulas, além dos símbolos M, P e F, introduzidos na figura 3.3, são usados ainda D para representar a distância do nó de origem ao nó de destino (medida pelo número de canais físicos atravessados), W para representar a capacidade de fluxo (medida em *Mbytes* por segundo, por exemplo) dos canais físicos e K para representar o tempo necessário para estabelecer uma conexão em uma rede de interconexão dinâmica. As fórmulas da tabela 3.1 para o tempo em trânsito mínimo de uma mensagem usando a comutação de pacotes e os mecanismos *virtual-cut-through* e *wormhole* valem quando todos os pacotes de uma mesma mensagem seguem a mesma rota.

Tabela 3.1: Tempo em trânsito mínimo.

Mecanismo de comunicação	**Tempo em trânsito mínimo**
Comutação de mensagens	$M D / W$
Comutação de pacotes	$(M + (P (D - 1))) / W$
Comutação de circuitos	$K (D - 1) + M / W$
Virtual-cut-through	$(M + (F (D - 1))) / W$
Wormhole	$(M + (F (D - 1))) / W$
Múltiplos estágios	$K D + M / W$
Crossbar	$K + M / W$
Comunicação direta	M / W

Nos mecanismos *virtual-cut-through* e *wormhole*, todos os *flits* de um mesmo pacote seguem obrigatoriamente a mesma rota. Assim, o tempo em trânsito mínimo de um pacote pode ser expresso por

$$(P + (F (D - 1))) / W.$$

Como em geral o número de *flits* em um pacote é consideravelmente maior que o número de pacotes de uma mensagem, o impacto dos mecanismos *virtual-cut-through* e *wormhole* em relação à transmissão de um pacote é muito maior do que o da comutação de pacotes em relação à transmissão de uma mensagem, como pode ser deduzido das fórmulas apresentadas. Entretanto, a comparação correta a ser estabelecida entre a comutação de pacotes e os mecanismos *virtual-cut-through* e *wormhole* deve considerar o tempo em trânsito mínimo de uma mensagem e não de um pacote [FAU97].

Um exemplo numérico pode mostrar claramente que a importância da redução conseguida pelo emprego do mecanismo *wormhole* em relação à comutação de pacotes não é extraordinária quando se faz a comparação correta. Considere-se para isso uma mensagem de

384 *bytes*, um pacote de 64 *bytes*, um *flit* de 1 *byte* e uma distância de 4 canais físicos. Usando a comutação de pacotes, o tempo em trânsito mínimo de uma mensagem é igual a 576W e usando o mecanismo *wormhole*, é igual a 387W, dando uma relação de apenas 576W / 384W = 1.49. Compare-se esse resultado com o seguinte, normalmente encontrado na bibliografia: usando a comutação de pacotes, o tempo em trânsito mínimo de um pacote é igual a 256W e usando o mecanismo *wormhole*, é igual a 67W, dando a relação muito mais expressiva que a anterior de 256W / 67W = 3.82.

Além disso, nas situações de interesse prático, quando existe algum congestionamento na rede de interconexão, o compartilhamento de canais físicos pode ter impactos diferentes sobre esses dois mecanismos de comunicação. Suponha-se a situação na qual dois pacotes de duas mensagens diferentes estão aptos a ser emitidos a partir de um mesmo nó através de um mesmo canal físico. Na comutação de circuitos e no *virtual-cut-through*, os dois pacotes, que já estariam armazenados no nó emissor, seriam serializados e um deles seria recebido pelo nó seguinte com um retardo de PW e o outro com um retardo de 2PW. No mecanismo *wormhole*, com o uso de canais virtuais, os dois pacotes cujos primeiros *flits* já estariam armazenados no nó emissor, seriam multiplexados *flit* a *flit* e ambos seriam recebidos pelo nó seguinte com um retardo de 2PW, aproximadamente.

As vantagens dos mecanismos *virtual-cut-through* e *wormhole* são maiores quando o tráfego é pequeno. A medida que o tráfego aumenta, o desempenho desses mecanismos diminui [KIM94] [ADV94]. O mapeamento das redes de processos comunicantes é uma questão importante mesmo para esses mecanismos porque um mapeamento inadequado é responsável por si só pelo aumento do tráfego [FRE98].

Se diferentes pacotes seguirem independentemente rotas distintas, o tempo em trânsito mínimo pode ser bastante reduzido na comutação da pacotes e nos mecanismos *virtual-cut-through* e no *wormhole* em relação aos tempos calculados pelas fórmulas da tabela 3.1 porque mais de um pacote da mesma mensagem estará sendo transportado simultaneamente por canais físicos distintos. Entretanto, se os mecanismos *virtual-cut-through* e *wormhole* não empregarem decomposição de mensagens em pacotes, o desempenho da comutação de pacotes pode ser superior. Assim, nas situações de interesse prático, quando existe algum congestionamento na rede de interconexão, o compartilhamento de canais físicos reduz consideravelmente o desempenho efetivo desses mecanismos.

Os tempos em trânsito mínimos na comutação de circuitos e nas redes de interconexão dinâmicas dependem da propagação prévia de um sinal de conexão entre o nó de origem e o nó de destino. O tempo mínimo para estabelecer uma rota é proporcional ao número de conexões da rota.

Na comunicação direta, o tempo em trânsito mínimo é o menor de todos os apresentados na tabela 3.1 e corresponde simplesmente ao tempo necessário ao seu transporte entre os nós de origem e de destino. No mapeamento perfeito, esse tempo não é apenas o mínimo, mas o efetivo para todos os canais físicos.

Gerência de Buffers

Com exceção da comutação de circuitos com suporte de *hardware* e da comunicação direta, todos os demais mecanismos de comunicação precisam tratar do problema de gerência de *buffers* nos nós intermediários da rota entre os nós de origem e de destino.

Na comutação de pacotes, cada nó deve reservar um espaço de memória para armazenar os pacotes em trânsito. A gerência de pacotes de tamanho fixo é mais simples do que a das mensagens de tamanho variável. Nesse aspecto, a comutação de pacotes é mais conveniente do que a comutação de mensagens.

No mecanismo *virtual-cut-through*, a gerência de *buffers* deve lidar com o armazenamento de até todos os *flits* de um pacote quando o canal físico estiver sendo usado por outra comunicação. No mecanismo *wormhole*, a gerência de *buffers* deve lidar com o armazenamento de um *flit* para cada canal virtual quando o canal físico de saída estiver sendo usado por outra comunicação.

Nas redes dinâmicas, a gerência de *buffers* deve lidar com os pacotes de mensagens distintas que podem chegar intercalados, embora os pacotes de cada mensagem cheguem em ordem.

Na comutação de circuitos com suporte de *hardware* e na comunicação direta, como as mensagens fluem diretamente entre os nós comunicantes, a gerência de *buffers* é completamente eliminada. No mapeamento perfeito, todas as comunicações prescindem da gerência de *buffers*.

Considerações sobre os Mecanismos de Comunicação

A comunicação direta pode ser vista como uma referência para os mecanismos de comunicação por representar simultaneamente o limite superior para o desempenho e para a simplicidade. Nesse sentido, todos os demais mecanismos de comunicação podem ser entendidos como alternativas à impossibilidade prática de se dispor da comunicação direta de forma irrestrita.

Como referência, a comunicação direta pode ser usada de duas formas: para verificar o quanto cada mecanismo de comunicação específico se distancia do ideal e para guiar a concepção de um novo mecanismo no sentido de atingir essa situação ideal.

O mapeamento perfeito estende os benefícios da comunicação direta à toda a rede de processos comunicantes e pode então ser considerado pelo seu efeito multiplicativo.

A realização do mapeamento perfeito garante que as mensagens sejam sempre transmitidas entre processos executando em nós vizinhos através de canais físicos dedicados. Esse fato conduz simultaneamente ao mais simples mecanismo de comunicação (porque não são necessários nem roteamento nem manipulação de pacotes) e ao mais alto desempenho (porque não existe disputa por canais físicos). Se a mesma rede de processos comunicantes (um anel, por exemplo) tiver um mapeamento perfeito em duas redes de interconexão distintas (um torus e um hipercubo, por exemplo), seus desempenhos em tempo de execução serão equivalentes. Quando o mapeamento perfeito não é possível, um mapeamento adequado para assegurar uma execução eficiente pode ser uma tarefa complexa [REE87].

O conceito de mapeamento perfeito não se aplica às redes dinâmicas. Elas estão livres do problema de mapeamento de processos, mas não dos problemas de conexão de canais físicos e da manipulação de pacotes. Conforme visto em 2.2.2.2, as redes de interconexão dinâmicas podem ser classificadas em não-bloqueantes, rearranjáveis e bloqueantes,. As redes rearranjáveis e bloqueantes têm seu desempenho potencial diminuído a medida que a disputa por canais aumenta com o aumento do tráfego [BRO83].

Os resultados reunidos nesta seção formam um sólido conjunto de argumentos motivadores do esforço de concepção de um ambiente multicomputador que permita privilegiar a comunicação direta e facilitar o mapeamento perfeito.

3.1.5 Articulação entre Componentes

Um ambiente completo para a execução de programas envolve elementos das áreas de Arquitetura de Computadores, Sistemas Operacionais e Linguagens de Programação. Pode-se considerar que essas três áreas fornecem os componentes básicos sobre os quais as aplicações finais devem se apoiar. É fato reconhecido que elas formam uma cadeia lógica na qual as evoluções em cada uma delas estimula e é estimulada pelas evoluções nas outras duas (figura 3.3).

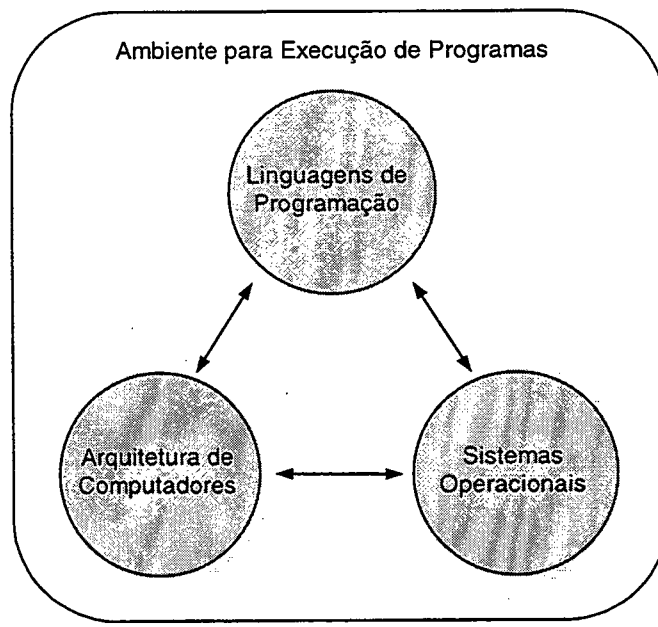


Figura 3.3: Ambiente para execução de programas.

Um projeto autônomo em uma dessas áreas deve se conformar ao emprego de componentes das outras duas, conforme o estado da arte delas. Quando esses componentes não se mostram adequados, podem surgir de um projeto desse tipo sugestões que eventualmente instigarão progressos nas outras áreas. Por outro lado, esse tipo de projeto pode eventualmente conduzir a um impasse: uma solução simples adotada isoladamente em uma das áreas pode gerar problemas complexos ou até insolúvel nas outras.

Ambientes completos integrando harmonicamente componentes dessas três áreas não costumam ser objeto de projetos de pesquisa em função de suas dimensões. Entretanto, como cada estágio de desenvolvimento científico e tecnológico cria suas próprias circunstâncias para o salto ao estágio seguinte, existem situações em que componentes dessas áreas precisam apenas ser corretamente associados para se compor um resultado importante. Existem outras

situações onde nem todos os componentes estão presentes, sendo necessário que se detecte os componentes faltantes e se consiga construi-los, para obter um resultado completo. Em ambos os casos, a dimensão do projeto integrado pode ser bastante reduzida e seus resultados podem se caracterizar por uma grande articulação entre seus componentes.

A necessidade de articular harmoniosamente os componentes de um ambiente multicomputador para a execução de programas paralelos expressos como redes de processos comunicantes conduz à motivação de manipular conceitos das três áreas que aparecem na figura 3.3.

3.2 Princípios de Projeto

São enunciados a seguir uma série de princípios de projeto, tanto de ordem geral quanto de ordem específica, que serviram para orientar o processo de tomada de decisões durante o desenrolar deste trabalho:

- *Simplicidade* - Projetos de pesquisa devem buscar ativamente as soluções mais simples como um princípio de ordem universal já que, entre as soluções efetivas, as mais simples tendem sempre a se impor. Infelizmente, soluções simples podem mascarar o esforço intelectual para a sua concepção. Dessa forma, o mérito de uma solução simples pode não ser reconhecido mesmo que sua importância e utilidade sejam inquestionáveis.
- *Flexibilidade* - Projetos experimentais devem ser concebidos e construídos de forma a estimular a introdução de modificações para responder às exigências de extensões que surgem naturalmente no curso da evolução deste tipo de projeto.
- *Inovação* - A inovação é um requisito básico de uma proposta científica. Entretanto, uma reação inicial comum a uma solução radicalmente nova, que provoca uma ruptura em relação a uma corrente de pesquisa consolidada, é de rejeição [BAR61]. Mesmo assim, propostas dessa natureza deveriam ser incentivadas pelos agentes científicos, apesar dos riscos inerentes, porque elas podem gerar novas correntes de pesquisa. Elas são arriscadas porque podem não se mostrar efetivas no futuro e serem esquecidas com o passar do tempo.
- *Redes de Processos Comunicantes* - Considera-se que as redes de processos comunicantes constituam o modelo de programa paralelo que mais naturalmente se adapta aos multicomputadores.
- *Exploração do Paralelismo Físico* - Admite-se a máxima distribuição dos processos pelos nós do multicomputador com o duplo objetivo de explorar o paralelismo físico e aumentar a disponibilidade dos processos individuais.
- *Atribuição Dinâmica* - Um multicomputador de uso geral deve permitir a execução simultânea de vários programas paralelos compartilhando os recursos físicos disponíveis. Considera-se que o ambiente deva permitir a introdução e a remoção dinâmica tanto de programas paralelos como de seus processos e canais lógicos componentes.

3.3 Objetivos

3.3.1 Delimitações Práticas Impostas à Proposta

O projeto e a construção efetiva de um ambiente da dimensão do que é aqui imaginado dependeria de recursos humanos e materiais que ultrapassariam largamente as ambições deste trabalho. Entretanto, a esperança de produzir já neste trabalho resultados que permitissem a avaliação da proposta em termos de sua flexibilidade, simplicidade e desempenho, conduziu à sua delimitação prática através dos elementos de referência destacados a seguir.

Arquitetura de Computadores

Propostas de novas arquiteturas de computadores paralelos vêm se constituindo em um tema proeminente de pesquisa. Duas tendências igualmente importantes se destacam nessa área: uma apoia-se em tecnologias sofisticadas de integração em larga escala de circuitos eletrônicos e outra na montagem de grande número componentes de tecnologia consolidada [HWA93].

A segunda tendência é preponderantemente representada pelos multicomputadores. Entretanto, como se observou no capítulo 2, as arquiteturas de multicomputadores propostas pecam muitas vezes por falta de generalidade no sentido em que cada uma delas é adequada para a execução eficiente de reduzida gama de algoritmos paralelos. De qualquer maneira, é necessário encontrar formas de explorar o paralelismo existente nessas máquinas em função do grande número de nós e canais físicos.

A situação atual corresponde, na esmagadora maioria dos casos, ao abandono da idéia do mapeamento perfeito, dando origem a arquiteturas com redes estáticas ou dinâmicas associadas a mecanismos de comunicação inerentemente complexos.

O ambiente proposto deve distanciar-se dessa tendência geral, buscando soluções alternativas que envolvam a reunião de componentes consolidados em estruturas físicas orientadas à simplicidade dos mecanismos de comunicação associados.

Sistemas Operacionais

Ao caracterizar um sistema operacional, convém considerar separadamente o conjunto dos serviços por ele prestados da estrutura usada para implementá-los.

Normalmente, os sistemas operacionais para multicomputadores têm adotado os serviços gerais de algum sistema consagrado, ao qual são agregadas extensões referentes à troca de mensagens entre processos [REE87]. Nos casos em que os multicomputadores estejam conectados a uma máquina hospedeira, é comum ser adotado o sistema dessa máquina. Como, na esmagadora maioria das vezes, esse sistema é o Unix, ele pode ser considerado um padrão de fato. Mais do que isso, algumas vezes a compatibilidade com o sistema Unix é um requisito de projeto.

Quando os multicomputadores estão conectados a máquinas hospedeiras, os serviços gerais do sistema ficam a cargo da máquina hospedeira e as extensões para troca de mensagens devem englobar ambos os ambientes. Alguns multicomputadores recentes

operaram como máquinas autônomas, nas quais os serviços do sistema são executados por alguns nós especializados. Nesse caso, a estrutura padrão para o sistema operacional segue o modelo cliente-servidor. Sua implementação envolve um núcleo idêntico (exceto pela identificação do nó no qual ele executa), replicado em todos os nós do multicomputador, encarregado de um pequeno grupo de serviços essenciais, relativos à gerência de processos e à troca de mensagens. Os demais serviços habituais de um sistema operacional são efetuados pelos seus servidores implementados como processos regulares. A implementação de um sistema desse tipo em um multicomputador pode ser altamente simplificada se comparada à implementação em um sistema computacional distribuído.

As redes de processos comunicantes envolvem mensagens completas trocadas diretamente entre processos. A transmissão de mensagens entre processos que executam em nós diferentes é essencial tanto ao sistema operacional quanto aos programas de usuários. Os multicomputadores convencionais possuem uma única rede de interconexão pela qual devem transitar todas as mensagens, sejam elas geradas pelo sistema operacional ou pelos programas de usuários. No caso mais freqüente das redes estáticas, o mapeamento inadequado das redes de processos comunicantes introduz a necessidade do roteamento de pacotes entre nós não vizinhos. A implementação de mecanismos de roteamento confiáveis é uma tarefa inerentemente complexa. Como resultado, o roteamento ocupa o centro das atenções do projetista do serviço de comunicação, como pode ser inferido pela vasta bibliografia sobre o tema.

Em relação aos serviços, o ambiente proposto deve assumir a interface de programação do sistema Unix. Essa interface deve ser acrescida apenas pelos operadores de um serviço específico de comunicação para suportar a execução de programas paralelos. Em relação à estrutura, o ambiente proposto deve adotar o modelo cliente-servidor, conforme exemplificado por sistemas distribuídos como o Mach e o Amoeba. As características fundamentais adotadas nesses sistemas referem-se apenas à concepção de um núcleo fornecendo as funções essenciais para permitir que a maior parte dos serviços dos sistema seja provida por servidores executando como processos regulares.

Linguagens de Programação

Se, por um lado, a exploração do paralelismo real disponível nos multicomputadores impõe a construção de programas paralelos sob a forma de redes de processos que se comunicam exclusivamente através de troca de mensagens, por outro lado, a programação explícita de redes desse tipo vêm sendo considerada um bom modelo de programação paralela e têm dado origem a várias linguagens de programação, conforme visto em 2.1. Além disso, esforços vêm sendo feitos ao longo dos anos para a formulação de modelos teóricos que permitam a expressão e a análise do comportamento dessas redes [HOA85].

Em sua expressão mais geral, um programa paralelo construído como uma rede de processos comunicantes pode exibir durante sua existência variações topológicas resultantes da introdução e/ou da remoção tanto de processos quanto de canais lógicos. As linguagens que permitem a expressão de programas desse tipo impõem regras específicas tanto para a criação dinâmica de processos quanto para a criação dinâmica de canais lógicos. A linguagem de referência é caracterizada a seguir através de propriedades simples, úteis para determinar os serviços que devem ser providos pelo ambiente objeto deste trabalho:

- *Processos* - Os processos são as unidades elementares de composição dos programas paralelos. Eventuais múltiplos fluxos de execução somente são reconhecidos internamente aos nós onde os processos executam. Um processo é criado pela vontade de outro.
- *Canais lógicos* - Os canais lógicos são síncronos dedicados. Um canal lógico é criado entre dois processos pela vontade de ambos, exceto durante a criação de processos, quando podem ser atribuídos canais lógicos ao novo processo pela vontade do processo criador.
- *Mensagens* - As mensagens trocadas pelos processos são blocos lógicos de tamanho variável cuja estrutura interna não é conhecida pelo ambiente.

O ambiente proposto deve simplificar a implementação de linguagens que ofereçam um subconjunto das propriedades enunciadas acima. Assim, por exemplo, uma linguagem que admita criação dinâmica de processos deve ser imediatamente atendida pelo ambiente. Os serviços não contemplados por essas propriedades deverão ser suportados por componentes específicos das próprias linguagens. Assim, por exemplo, nenhum suporte é previsto pelo ambiente para tratar dos tipos de dados transportados pelas mensagens.

Esse ambiente deve incentivar o mapeamento perfeito das redes de processos comunicantes em muitos casos de interesse prático. Em relação aos processos, apenas o mapeamento perfeito é admissível. Em relação aos canais lógicos, mesmo quando o mapeamento perfeito não for possível, apenas as comunicações diretas são admissíveis.

3.3.2 Lista de Objetivos

O objetivo fundamental deste trabalho consiste na concepção e avaliação de um ambiente multicomputador para execução de programas paralelos expressos como redes de processos comunicantes.

De forma mais precisa, podem ser listados os seguintes objetivos para este trabalho:

- Conceber um modelo de multicomputador reconfigurável e de um mecanismo de conexão dinâmica de canais físicos que, associados, permitam *em muitos casos de interesse prático* a construção da rede de interconexão em tempo de execução para corresponder perfeitamente à topologia das redes de processos comunicantes e *em todos os casos* a transmissão de mensagens completas de tamanho arbitrário como blocos monolíticos através de canais físicos dedicados entre quaisquer pares de nós
- Avaliar a flexibilidade e a simplicidade do ambiente integrado para a execução de redes de processos comunicantes.

- Avaliar o desempenho do conjunto formado pelo modelo multicomputador e seu mecanismo de conexão dinâmica de canais físicos proposto.
- Conceber a estrutura de um sistema operacional experimental com interface equivalente à do sistema Unix visto complementarmente como camadas de *software* hierarquicamente superpostas, definidas pela especificação das suas interfaces, e como uma rede de processos comunicantes segundo o modelo cliente-servidor, baseada na distribuição física de seus processos pelos nós do multicomputador e em um mecanismo específico de comunicação com conexão dinâmica de canais físicos.
- Conceber um suporte de execução para uma linguagem paralela apoiado nos serviços do sistema operacional acrescidos de um mecanismo específico de comunicação com conexão dinâmica de canais físicos.

3.4 Multicomputador Crux

3.4.1 Arquitetura

Para uma rede de processos comunicantes específica com topologia estática, o multicomputador ideal pode ser construído com uma topologia equivalente. Se a rede de processos comunicantes considerada possui topologia variável, o multicomputador ideal deveria suportar todas as suas variações topológicas em tempo de execução. Se várias redes de processos comunicantes independentes com topologias variáveis devem executar simultaneamente, o multicomputador ideal deveria acomodá-las perfeitamente a cada momento.

A única arquitetura estática que atende todas essas exigências é o grafo completo. Entretanto, essa topologia é inviável além de um número reduzido de nós [HWA93]. Além disso, a enorme quantidade de canais físicos providos por essa topologia somente seria exigida por raros algoritmos de interesse prático. As alternativas mais freqüentes têm sido a construção de multicomputadores com redes de interconexão estáticas ou dinâmicas mais simples associadas a mecanismos de comunicação necessariamente mais complexos.

A execução simultânea de programas paralelos construídos como redes de processos comunicantes de topologia variável exige o emprego de multicomputadores com redes de interconexão dinâmicas se as mensagens entre processos executados em nós diferentes devem fluir sempre através de canais físicos diretos.

Os processos de um programa paralelo são entidades ativas que só trocam mensagens por acordo mútuo. Assim, um canal físico direto entre dois nós só deveria ser estabelecido pela concordância dos processos neles executados. Esse método de construção dinâmica das redes físicas induz o emprego de uma rede dinâmica com controle externo centralizado. Essa solução é reforçada ainda pelo fato dessas redes serem mais simples para construir e operar.

O dispositivo central de controle deve estar permanentemente acessível a todos os nós do multicomputador para receber e tratar as requisições de conexão/desconexão de canais

físicos. Isso exige a utilização de outra rede de interconexão — independente da anterior — com essa conetividade específica.

Muitos problemas de interesse prático podem ser expressos como redes de processos comunicantes cujos nós possuem um número moderado de canais lógicos com alguma estabilidade temporal [HAN95]. O grau dos nós de um multicomputador determina o grau máximo dos processos das redes de processos comunicantes para os quais o mapeamento perfeito é viável.

A arquitetura do multicomputador Crux resulta da reunião dos seguintes componentes, conforme a figura 3.4 [COR98]:

- Um número expressivo de *nós trabalhadores* (cada um dos quais possuindo processador com memória privativa e vários canais físicos) para executar os processos das rede de processos comunicantes.
- Uma *rede de trabalho* (um *crossbar*) para transportar mensagens entre quaisquer pares de nós trabalhadores através de canais físicos diretos.
- Um *nó controlador* para executar um processo específico responsável pela configuração da rede de trabalho (e pela atribuição/liberação de nós de trabalho).
- Uma *rede de controle* (um barramento) para transportar mensagens entre os nós trabalhadores e o nó controlador.

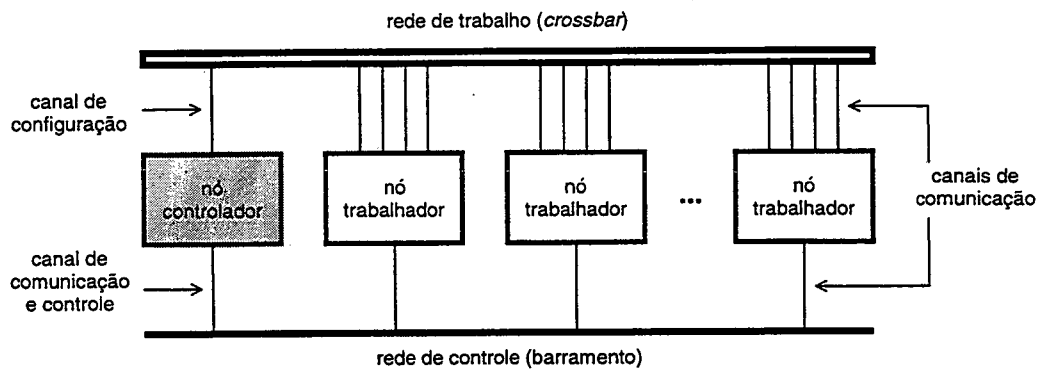


Figura 3.4: Arquitetura do multicomputador Crux.

Essa descrição geral da arquitetura do multicomputador Crux deixa em aberto vários aspectos de *hardware*, entre os quais o número de canais de comunicação associados a cada nó trabalhador. Adota-se como referência daqui para frente a existência de 1 canal conectado à rede de controle (barramento) e 4 canais conectados à rede de trabalho (*crossbar*), correspondendo exatamente a situação representada na figura 3.4. Outros aspectos precisariam ser especificados no âmbito de um projeto de pesquisa que visasse a construção efetiva de um protótipo desse multicomputador mas que não são imprescindíveis à avaliação do modelo geral.

3.4.2 Conexão Dinâmica de Canais Físicos

O multicomputador Crux possui alguma similaridade estrutural com o Supernode (visto em 2.2.3.2) que lhe serviu de inspiração inicial. Contudo, eles representam modelos de operação radicalmente diferentes, resultante das diferentes funcionalidades dos seus componentes.

No Crux, a rede de trabalho é usada para transportar mensagens de tamanho arbitrário através de canais físicos diretos entre dois nós trabalhadores e a rede de controle é usada exclusivamente para transportar mensagens de controle entre um nó trabalhador e o nó controlador. O modo de operação do multicomputador Crux pode ser caracterizado pelo seguinte mecanismo de comunicação genérico com conexão dinâmica de canais físicos por demanda do mapeamento de canais lógicos [COR97]. Antes da transmissão efetiva de uma mensagem entre dois nós trabalhadores, deve ser verificada a existência de um canal físico os unindo através da rede de trabalho. Cada nó trabalhador mantém localmente informações sobre o estado das conexões de seus canais físicos. Se os nós já estão conectados, a transmissão é iniciada imediatamente. Se não, a transmissão é precedida por uma *transação de conexão* entre os nós trabalhadores e o nó controlador através da rede de controle. Um nó trabalhador solicita uma conexão em uma mensagem enviada ao nó controlador pela rede de controle e espera a confirmação em uma mensagem recebida do nó controlador pela mesma rede de controle. O nó controlador é responsável pela gerência das conexões, de acordo com alguma política específica, mantendo as informações sobre o estado das conexões dos canais físicos de todos os nós trabalhadores.

A conexão dinâmica de canais físicos por demanda do mapeamento de canais lógicos é um mecanismo completamente geral: a atribuição de cada canal físico pode durar desde o tempo para a transmissão de uma única mensagem até a completa execução de uma rede de processos comunicantes, passando por todas as durações intermediárias concebíveis entre esses dois extremos.

Assim, é possível considerar diferentes políticas, conforme as características da rede de processos comunicantes. Para uma rede lógica estática em forma de estrela, por exemplo, na qual o grau do processo central excede largamente o grau do nó trabalhador onde ele executa, a transação de conexão deve ser ativada sempre que o canal lógico a ser usado não esteja atribuído correntemente a um canal físico. Por outro lado, para uma rede lógica estática em forma de árvore, por exemplo, na qual o grau dos processos é inferior ao grau dos nós trabalhadores, a transação de conexão é ativada apenas uma vez para cada canal lógico porque ele é atribuído a um canal físico por toda a execução da rede de processos comunicantes.

3.4.3 Avaliação Qualitativa

Em conjunto, a arquitetura do multicomputador Crux e seu mecanismo de conexão dinâmica de canais físicos definem um paradigma de comunicação em multicomputadores completamente novo, no qual

- mensagens completas de tamanho arbitrário são transmitidas sempre como blocos monolíticos através de canais físicos dedicados diretos entre quaisquer pares de nós, e,

- a rede física é construída em tempo de execução para corresponder perfeitamente à topologia de muitas redes lógicas de processos comunicantes de grande interesse prático.

Uma avaliação qualitativa, em relação a flexibilidade e a simplicidade, do mecanismo de conexão dinâmica de canais físicos pode ser estabelecida através de sua comparação com os mecanismos de mapeamento usados em outras redes de interconexão de multicomputadores.

Flexibilidade

Em relação à flexibilidade, pode-se comparar os procedimentos envolvidos no mapeamento das redes de processos comunicantes em diversos ambientes multicomputadores:

1. *Execução de uma rede estática de processos comunicantes única para a qual é possível estabelecer o mapeamento perfeito nas redes de interconexão estáticas.*

   Nas redes de interconexão estáticas, o mapeamento perfeito deve ser planejado antes da carga do programa paralelo à partir do conhecimento tanto da topologia da rede de processos comunicantes quanto da topologia da rede de interconexão do multicomputador. Nas redes de interconexão dinâmicas (com árbitro embutido), o mapeamento perfeito de processos é obtido em tempo de execução pela atribuição de nós arbitrariamente escolhidos mas o mapeamento perfeito de canais lógicos não é possível porque a atribuição aos canais físicos deve ser estabelecida para cada pacote transmitido. No Crux, o mapeamento perfeito é obtido em tempo de execução pela atribuição de processos a nós arbitrariamente escolhidos e o mapeamento perfeito de canais também é obtido em tempo de execução pela conexão dos canais físicos dos nós envolvidos.

   Em relação ao mapeamento de canais lógicos, o árbitro interno de uma rede dinâmica deve ser acionado para cada pacote enquanto que o mecanismo de conexão dinâmica de canais físicos do Crux é acionado apenas durante a criação dos canais lógicos. Além disso, nas redes dinâmicas, o compartilhamento de canais físicos exige sempre a manipulação de pacotes.

2. *Execução de uma única rede estática de processos comunicantes para a qual não é possível estabelecer o mapeamento perfeito nas redes de interconexão estáticas.*

   Nas redes de interconexão estáticas, o mapeamento mesmo que não seja perfeito ainda deve ser planejado, antes da carga do programa paralelo, quando se quiser tirar proveito de fenômenos de localidade através de uma adaptação conveniente da rede de processos comunicantes à rede de interconexão do multicomputador. Nas redes dinâmicas e no Crux, os procedimentos descritos no item 1 não são afetados por essa nova situação. Entretanto, no Crux o mesmo mecanismo de conexão dinâmica de canais físicos deve agora ser acionado sempre que atribuição de um canal lógico a um canal físico deva ser alterada.

   Apesar disso, o Crux admite o mapeamento perfeito sempre que o número de canais lógicos dos processos não supere o número de canais físicos dos nós. Por isso, a gama das redes lógicas para as quais é possível o mapeamento perfeito no Crux é superior ao de qualquer rede estática com grau dos nós equivalente.

3. *Execução simultânea de várias redes dinâmicas de processos comunicantes.*

   Nas redes de interconexão estáticas, o mapeamento mesmo imperfeito não pode ser planejado antes da carga dos programas paralelos pelo desconhecimento prévio das topologias das redes de processos comunicantes. Nessa situação o aproveitamento de fenômenos de localidade se torna uma tarefa bastante complexa a ser considerada em tempo de execução. Nas redes de interconexão dinâmicas e no Crux, os procedimentos descritos no item 1 continuam a não ser afetados por essa nova situação. Em relação à situação do item 2, ampliam-se consideravelmente as dificuldades de uso das redes de interconexão estáticas em relação às redes dinâmicas e ao Crux.

   Nessa situação completamente geral, apenas o Crux ainda admite o mapeamento perfeito. As condições em que isso é possível são limitadas apenas pelo número de nós (que deve ser superior ou igual ao de processos) e do grau dos nós (que deve ser superior ou igual ao grau dos processos)

A duração de um canal físico em uma rede de interconexão estática é igual ao tempo de existência da própria rede. A duração de um canal físico de uma rede dinâmica corresponde ao tempo de transporte de um pacote de tamanho fixo. Em nenhum desses casos a duração dos canais físicos coincide com a dos canais lógicos. A duração de um canal físico no Crux, operando com o mecanismo de conexão dinâmica de canais físicos, pode ser exatamente igual à duração do canal lógico quando o grau dos processos não supera o grau dos nós onde eles executam. Além disso, esse mecanismo é suficientemente geral para tratar de maneira uniforme a multiplexação de canais físicos quando o grau dos processos supera o grau dos nós onde eles executam. Em suma, a flexibilidade superior do Crux resulta da maior facilidade de adaptação de sua rede de trabalho que pode ser configurada em tempo de execução em resposta as exigências das redes de processos comunicantes, inclusive as que envolvem criação dinâmica de processos e de canais lógicos.

Simplicidade

Em relação à simplicidade, pode-se comparar os procedimentos envolvidos nas comunicações através das redes de interconexão de diversos ambientes multicomputadores. Nas redes estáticas, as comunicações envolvem roteamento de pacotes e/ou *flits*. Nas redes dinâmicas, as comunicações envolvem o acionamento do árbitro da rede a cada pacote transportado e a manipulação de pacotes. No Crux, os algoritmos elementares de conexão dinâmica de canais físicos, permite a comunicação direta através de mensagens completas. A simplicidade superior do Crux resulta da eliminação do roteamento de mensagens e da manipulação de pacotes que promovem em uma enorme redução da complexidade das comunicações, podendo ser consideradas justificativas suficientes para validar a proposta apresentada neste trabalho.

Inconvenientes

A proposta aqui apresentada se limita, pelo menos a curto prazo, a multicomputadores de porte médio, por causa do moderado número de canais dos *crossbars* disponíveis atualmente. Nesta proposta não se pensa além da centena de nós compatíveis com a tecnologia atual dos *crossbars* e não em centenas ou milhares de nós como vêm sendo cogitado para multicomputadores com redes estáticas. Entretanto, o aumento de demanda por *crossbars* de

maiores dimensões poderá no futuro reduzir consideravelmente seu custo comercial. Apesar disso, a flexibilidade do mecanismo de conexão dinâmica de canais físicos do Crux permite considerar o uso de vários *crossbars* independentes, a cada um dos quais seria conectado um canal físico distinto de cada nó. Se essa solução reduz a gama de topologias para as quais o mapeamento perfeito é possível em relação ao *crossbar* único, ela permite empregar *crossbars* de menores dimensões [NIC88].

O Crux possui dois componentes compartilhados críticos: o nó controlador e a rede de controle. Tanto suas principais qualidades como seus principais inconvenientes decorrem desses componentes. O nó controlador e a rede de controle podem se transformar em gargalos se o ambiente for usado por redes lógicas que envolvam comunicações com mensagens curtas que exijam freqüentes mudanças de atribuições de canais físicos. Além disso, em relação à tolerância a falhas, uma interrupção de funcionamento do nó controlador ou da rede de controle compromete todo o ambiente. No emprego de técnicas de tolerância a falhas, envolvendo a duplicação desses componentes, a adição de protocolos complementares para gerir as redundâncias não comprometeria o funcionamento normal da rede de trabalho no que concerne ao desempenho. A falha de um nó trabalhador, em compensação, pode ser isolada com facilidade, sem provocar prejuízos maiores a rede de trabalho.

3.5 Conclusões

A arquitetura do multicomputador Crux e seu mecanismo de conexão dinâmica de canais físicos por demanda do mapeamento de canais lógicos foram definidos para formar um par indissolúvel para servir de suporte básico ao ambiente multicomputador completo que deve envolver ainda um sistema operacional e pelo menos uma linguagem paralela.

O desempenho do ambiente Crux submetido a diversas hipóteses de tráfego de mensagens por suas redes de interconexão precisa ser avaliado, o que é feito no capítulo 4. A conveniência do ambiente Crux em relação à flexibilidade e à simplicidade foram detectadas nesse capítulo mas ainda devem ser submetidas a experimentos específicos de implementação como os descritos no capítulo 5.

4 Avaliação do Ambiente Crux

Este capítulo é dedicado à avaliação do desempenho do multicomputador Crux operando sob o mecanismo de conexão dinâmica de canais físicos. Ele inclui uma avaliação comparativa com outros dois multicomputadores para cargas de trabalho habituais e uma avaliação isolada para uma carga de trabalho crítica para o seu desempenho.

4.1 Simulação

A avaliação de desempenho de sistemas pode envolver métodos analíticos, simulação ou medição direta. Em relação a sistemas complexos, a simulação proporciona em geral maior credibilidade do que os métodos analíticos porque requer um número menor de simplificações e, conseqüentemente, retém um número maior de características do sistema real. Tanto é assim que muitos métodos analíticos foram validados por seus autores através da comparação dos resultados produzidos pelo novo método com os obtidos através de simulações.

No caso deste trabalho, a simulação se justifica duplamente como a melhor alternativa em vista da complexidade e da indisponibilidade dos sistemas avaliados.

Simulação pode ser definida como o processo de construir um modelo para representar um sistema e de realizar experiências com o modelo com o objetivo de fazer inferências sobre o comportamento do sistema. Assim definida, a simulação inclui a modelagem, a experimentação e a interpretação dos resultados. No âmago da definição de simulação estão os conceitos de sistema e de modelo. Um *sistema* é uma coleção de componentes que executam funções específicas e interagem entre si para a realização do objetivo global do sistema. Um *modelo* é uma representação simplificada de um sistema que preserva apenas as características do sistema consideradas significativas.

Os modelos que interessam ao presente trabalho são os modelos computacionais nos quais o sistema é representado por um programa de computador. Por envolver o desenvolvimento de um programa em geral complexo, a simulação pode se beneficiar das técnicas da Engenharia de *Software*.

Carga de Trabalho

Uma *carga de trabalho real* é considerada em um processo de simulação como a carga computacional utilizada efetivamente na operação normal do sistema. Em geral a carga real não é reproduzível e por isso ela é inconveniente como carga de teste. Uma *carga de trabalho*

sintética, com características similares às da carga real mas que pode ser reproduzida e aplicada repetidas vezes de maneira controlada é muitas vezes preferível em avaliação de desempenho.

Cargas de trabalho sintéticas podem ser amostras de distribuições de probabilidade produzidas pela aplicação de métodos estocásticos às variáveis que condicionam o comportamento do sistema. O método usado para gerar valores para essas variáveis aleatórias é chamado amostragem de Monte-Carlo. Nesse método, dados artificiais são produzidos usando um gerador de números aleatórios e a distribuição cumulativa de probabilidade da qual se quer obter uma amostra. Um gerador de números aleatórios conveniente deve produzir valores uniformemente distribuídos no intervalo de 0 a 1. A distribuição de probabilidade a ser amostrada pode ser baseada em dados empíricos observados no sistema real ou pode ser uma distribuição teórica.

Quando se usam dados históricos, simula-se apenas o passado. Cargas de trabalho que permitam estudar novas hipóteses operacionais do sistema, por exemplo, são produzidas mais facilmente a partir de distribuições teóricas. Existe uma experiência acumulada que permite selecionar a distribuição mais conveniente para processos estocásticos de vários tipos.

Análise dos resultados

Como as simulações geram dados que são amostras do comportamento do modelo, todas as técnicas referentes à inferência estatística a partir de amostras são aplicáveis. Esta atividade da simulação exige conhecimento específico em procedimentos estatísticos.

Uma característica do sistema que determina a forma de execução da simulação e os métodos de análise dos resultados diz respeito ao seu período de operação. Sistemas de duração limitada podem ser caracterizados por estados iniciais e finais determinados e/ou eventos que marcam o início e o fim de sua operação. Sistemas de duração ilimitada podem ser caraterizados por um estado inicial e/ou um evento que marca o início de sua operação e podem permanecer indefinidamente em atividade.

Os sistemas de duração ilimitada podem ser estacionários ou não-estacionários. Um sistema é estacionário se a distribuição de suas variáveis de resposta (e portanto as médias e as variâncias dessas variáveis) não se alteram com o passar do tempo.

Os sistemas que interessam a este trabalho podem ser considerados de duração ilimitada e estacionários. Métodos estatísticos específicos, como o método de lotes a partir de uma única execução da simulação, podem ser empregados na análise desses sistemas.

As análises dos resultados das simulações descritas neste capítulo basearam-se nos valores médios da variável de resposta considerada. As médias foram obtidas aplicando o método de lotes a partir de uma única execução da simulação. A execução correspondeu a um tempo simulado de longa duração objetivando a geração de um grande volume de dados. Na aplicação do método de lotes, os dados pertencentes ao estado transiente foram identificados e removidos. Os dados do estado estacionário foram então divididos em um conjunto de lotes cujos tamanhos foram testados até que os valores médios da variável de resposta fossem estatisticamente independentes uns dos outros. Finalmente, o intervalo de confiança da amostra das médias dos lotes foi obtido com 95% de confiabilidade.

Linguagem de Simulação

A linguagem usada nas avaliações de desempenho descritas neste capítulo foi a linguagem Siman [PEG95]. Uma das grandes vantagens de uma linguagem específica para simulação, a exemplo de Siman, é o suporte oferecido ao processo de interpretação dos resultados. Alguns conceitos básicos dessa linguagem são introduzidos a seguir e ilustrados através de um pequeno exemplo.

Siman é uma linguagem de simulação que permite a modelagem tanto de sistemas discretos quanto de sistemas contínuos. Os sistemas discretos que interessam a este trabalho são normalmente representados em Siman através de modelos *orientados a processos*, nos quais são estudadas as entidades que se movem através do sistema modelado. Um modelo consiste de uma descrição dos processos através dos quais as entidades se movem em seu progresso através do sistema.

Os conceitos fundamentais associados aos modelos orientados a processos são o processo e a entidade. Um *processo* denota uma seqüência de operações ativada por uma entidade que passa por ele. Uma *entidade* denota um *item* cujo movimento através do modelo produz alterações no estado do sistema. Um processo permanece inativo até que uma entidade passe por ele. As operações efetuadas pelos processos podem alterar tanto atributos das entidades definidas quanto o estado geral do modelo.

Um simulador em Siman compreende dois segmentos: um modelo e um experimento. O *modelo* é uma descrição funcional dos componentes do sistema e de suas interações. O *experimento* determina as condições sob as quais o modelo é estudado.

A modelagem envolve ainda o uso de variáveis. O termo variável denota itens que podem ser alterados e afetam globalmente todos os componentes do modelo. Em especial, esse termo não se refere às características das entidades.

Processos de interesse prático podem conter componentes aleatórios. Vários operandos podem ser especificados na forma de variáveis aleatórias. A linguagem Siman contém uma série de funções para obter amostras das distribuições de probabilidades mais comuns.

O seguinte exemplo permite caracterizar diversos aspectos da linguagem Siman. Um processo está conectado a dois canais lógicos assíncronos — um de entrada e outro de saída. O processo efetua uma transformação sobre cada mensagem recebida pelo canal lógico de entrada e emite a mensagem modificada pelo canal lógico de saída. As mensagens chegam ao canal de entrada a intervalos de tempo que variam segundo uma distribuição exponencial. O tempo despendido para realizar a transformação sobre as mensagens se distribui uniformemente entre dois limites determinados. O tempo despendido na recepção e na emissão das mensagens não são significantes. Entre muitas variáveis de resposta possíveis, apenas o número de mensagens transformadas e emitidas pelo processo em um determinado intervalo de tempo é considerado nesse exemplo.

Um programa escrito em Siman para esse exemplo aparece na figura 4.1. As entidades passam pelas instruções de um modelo escrito em Siman na seqüência textual. Existem instruções (não utilizadas nesse exemplo) que permitem desviar entidades dessa seqüência. Os rótulos com sufixo "$" são usados nas instruções de desvio. O significado das instruções do modelo e das declarações do experimento são referidas pelos números das linhas que

aparecem à direita da figura na descrição apresentada a seguir. Apenas o significado dos operandos mais significativos são mencionados.

```
#       Modelo
0$      CREATE,          1,0:EXPO (10.0);          1
2$      SEIZE,           1:Processo,1;             2
1$      DELAY:           UNIF (4.0, 12.0);         3
3$      RELEASE:         Processo,1;               4
4$      COUNT:           Mensagens,1;              5
5$      DISPOSE;                                   6

#       Experimento
        PROJECT,         Exemplo,,,Yes;            7
        RESOURCES:       Processo,Capacity(1,),;   8
        COUNTERS:        Mensagens,,Yes;           9
        REPLICATE,       1,0.0,60000,Yes,Yes,0.0;  10
```

Figura 4.1: Exemplo de programa escrito em Siman.

A instrução CREATE é usada tipicamente para introduzir entidades no modelo segundo um padrão específico. A instrução da linha 1 representa a chegada de mensagens no canal lógico de entrada a intervalos de tempo que variam segundo uma distribuição exponencial com média igual a 10.0 milisegundos.

Um recurso denota um objeto que pode ser atribuído a uma entidade. A declaração RESOURCES permite definir os recursos usados no modelo. A declaração da linha 8 define Processo como sendo um recurso.

A instrução SEIZE atribui um recurso à entidade que passa por ela. Entidades esperam pelo recurso conforme a ordem de chegada a essa instrução. A instrução da linha 2 representa a recepção da mensagem mais antiga presente no canal lógico de entrada pelo processo.

Tipicamente, uma entidade utiliza o recurso a ela atribuído por um certo período de tempo. A instrução DELAY é usada para representar atividades de consomem tempo, segundo um padrão específico. A instrução da linha 3 representa o intervalo de tempo despendido pelo processo para transformar a mensagem recebida, conforme uma distribuição uniformemente distribuída entre 4.0 e 12.0 milisegundos.

A instrução RELEASE libera o recurso previamente atribuído à entidade que passa por ela. A instrução da linha 6 representa a emissão da mensagem transformada pelo processo através do canal de saída.

A declaração COUNTERS permite a definição de contadores de eventos que se produzem no modelo. A declaração da linha 9 define Mensagens como sendo um contador.

A instrução COUNT incrementa um contador a cada entidade que passa por ela. A instrução da linha 5 conta o número de mensagens emitidas pelo processo através do canal lógico de saída.

A instrução DISPOSE expurga do modelo a entidade que passa por ela. A instrução da linha 6 elimina do modelo as mensagens do canal lógico de saída.

A declaração PROJECT serve a fins de documentação apenas. A declaração da linha 7 dá o nome de Exemplo ao simulador.

A declaração REPLICATE permite definir, entre outras coisas, a duração de uma execução do simulador. A declaração da linha 10 define a duração do experimento como sendo de 1 minuto (60000 milisegundos).

A execução do programa da figura 4.1 produz como resultado apenas o número de mensagens transformadas e emitidas pelo processo durante 1 minuto, coletado no contador Mensagens.

Esses poucos elementos servem apenas como ilustração dos recursos de modelagem da linguagem Siman. Uma apresentação extensiva de todo o processo de simulação usando essa linguagem pode ser visto em [PEG95].

4.2 Avaliação de Desempenho do Crux

Este subcapítulo é dedicado à avaliação do desempenho do multicomputador Crux. O objetivo principal desse estudo está centrado sobre a avaliação comparativa da sua rede de interconexão com outras duas — uma estática e outra dinâmica. Além disso, a rede de interconexão do Crux foi ainda avaliada isoladamente para uma carga de trabalho específica, considerada crítica para o seu desempenho.

A avaliação comparativa do desempenho da rede de interconexão do multicomputador Crux, envolveu sua confrontação com a rede estática do torus e a rede dinâmica do *crossbar*. Entre outras alternativas razoáveis de redes estáticas, a escolha do torus deveu-se não apenas ao seu bom desempenho relativo a outras redes estáticas mas também à uniformidade de composição dos seus nós, que não varia com o tamanho da rede [DAL86]. Como exemplo de multicomputador construído em torno de uma rede estática pode ser citado o Paragon [INT91]. Entre as redes dinâmicas, a escolha do *crossbar* deveu-se exclusivamente ao seu desempenho superior em relação às demais redes dinâmicas [BRO83]. Como um exemplo de multicomputador construído em torno de um *crossbar* pode ser citado o VPP500 [FUJ92]. A comparação do Crux pôde limitar-se a apenas uma rede estática e uma rede dinâmica porque ambas as redes escolhidas estão na faixa superior de desempenho em suas categorias.

4.2.1 Hipóteses Gerais

Neste estudo, os nós das três redes de interconexão consideradas possuem um processador principal, memória privativa e um controlador de comunicação composto de um processador especializado e canais físicos de entrada e de saída independentes. Nos modelos construídos para a realização de todas as simulações, foram assumidos os seguintes mecanismos de comunicação, específicos para cada rede de interconexão:

- Crux - Mensagens completas são transferidas entre o nó trabalhador de origem e o nó trabalhador de destino como um bloco monolítico através de um canal físico direto da rede de trabalho. Mensagens de controle de tamanho reduzido (16 *bits*, nos modelos construídos) trafegam pela rede de controle sempre que um pedido de conexão deva preceder a comunicação efetiva entre dois nós trabalhadores. Esse mecanismo é baseado no mapeamento de canais lógicos por demanda.
- Torus - Mensagens são divididas em *flits* de tamanho reduzido (1 *byte*, nos modelos construídos). Todos os *flits* de uma mesma mensagem seguem a mesma rota entre o nó de origem e o nó de destino. Assim que o primeiro *flit* alcança um nó intermediário, ele pode ser transferido ao seguinte. Se não existir um canal físico de saída livre, o nó intermediário pode acomodar até uma mensagem inteira para cada canal físico de entrada. Esse é o mecanismo *virtual-cut-through*. Nas simulações realizadas, apenas as rotas com distância mínima foram utilizadas. A escolha desse mecanismo baseou-se nas fórmulas apresentadas na tabela 3.1, através das quais se verifica que tempo em trânsito mínimo do mecanismo *virtual-cut-through* é igual ao do mecanismo *wormhole* (que teria sido outra escolha conveniente) e inferior ao da comutação de pacotes.
- *Crossbar* - Mensagens são divididas em pacotes de tamanho moderado (64 *bytes*, nos modelos construídos). Um árbitro embutido na rede permite a transferência de pacotes, realizando a conexão e a desconexão de um canal físico direto entre o nó de origem e o nó de destino para cada pacote. Esse é o mecanismo natural para essa rede dinâmica.

Todas as simulações envolvem multicomputadores de porte médio, com número de nós (N) variando entre 25 e 49. Os modelos do Crux, do torus e do *crossbar* utilizados na maioria das simulações realizadas possuem 36 nós. Esses modelos, escritos na linguagem de simulação Siman, são apresentados no apêndice. Modelos com 25 e 49 nós, utilizados em algumas simulações, derivam dos modelos apresentados e foram omitidos neste texto.

O *comprimento* das mensagens (C) segue uma distribuição exponencial. O *tempo de produção* de mensagem (P) é proporcional ao seu comprimento e, portanto, também segue uma distribuição exponencial. As mensagens são transmitidas sempre por iniciativa exclusiva dos nós emissores. Os intervalos de tempo são medidos em uma unidade hipotética, referida simplesmente por *unidade de tempo*. A velocidade de transmissão dos canais físicos é constante e igual a 1 *byte* por unidade de tempo.

Cada nó executa um processo cujo comportamento é definido por uma sucessão continuada de ciclos compostos de duas etapas seqüenciais — a primeira para produção de uma mensagem e a segunda para a sua emissão.

O comprimento das mensagens e o tempo de produção de mensagens são os parâmetros explícitos da maioria das simulações realizadas. O comprimento das mensagens é o único parâmetro independente, uma vez que o tempo de produção de mensagens é sempre proporcional ao comprimento das mensagens. O volume de tráfego de mensagens nas redes de interconexão também é um parâmetro importante, freqüentemente referido nas próximas seções. Entretanto, como ele cresce com o aumento da taxa de produção de mensagens que é inversamente proporcional ao tempo de produção de mensagens, ele está implicitamente definido na maioria das simulações realizadas.

A variável de resposta usada para estimar o desempenho na maioria das simulações foi o *tempo em trânsito* das mensagens (T), medido entre o instante em que elas estão prontas para ser transmitidas pelo nó de origem e o instante em que elas são completamente recebidas pelo nó de destino.

Além dos símbolos introduzidos acima, são utilizados ainda neste capítulo os valores médios de C, P e T, respectivamente, representados por $C_{médio}$, $P_{médio}$ e $T_{médio}$.

4.2.2 Avaliação Comparativa

O objetivo desta seção é o de realizar a avaliação comparativa de desempenho das três redes de interconexão descritas em 4.2.1, quando submetidas a diversas cargas de trabalho de teste, correspondentes a padrões de comunicação específicos.

No torus existe uma relação permanente de distância entre seus nós, o que não se verifica nem no Crux nem no *crossbar*. Por isso, quando as cargas de trabalho são definidas por padrões de comunicação dependentes das relações de distância do torus, os nós de destino das mensagens são escolhidos no Crux e no *crossbar* pelos nós de origem segundo o mesmo critério adotado para o torus.

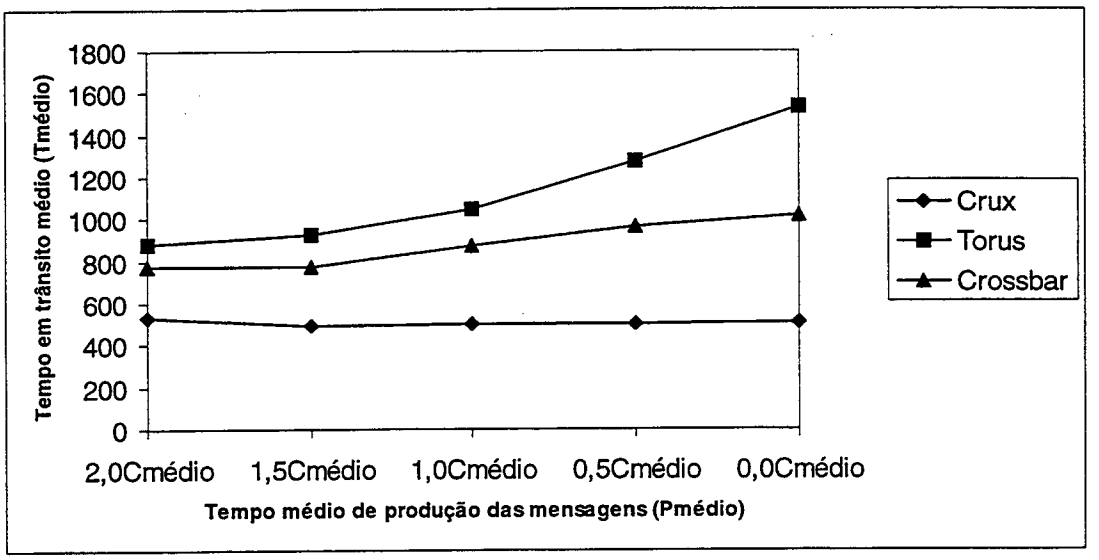
4.2.2.1 Comunicações Generalizadas

O objetivo das simulações descritas nesta seção foi o de avaliar o desempenho relativo das três redes de interconexão consideradas quando submetidas a um padrão de comunicação generalizado entre todos os nós, gerando um tráfego de mensagens uniformemente distribuído pelas redes. Apesar dessa carga de trabalho não corresponder a padrões de comunicação reais, ela é bastante difundida em avaliações de desempenho de redes de interconexão justamente por avaliá-las independentemente de comportamentos específicos das redes de processos comunicantes.

O padrão de comunicação generalizado é caracterizado, para efeitos das simulações realizadas, considerando que, para cada mensagem emitida, o nó de origem escolhe o nó de destino entre os demais nós segundo uma distribuição uniforme. Os parâmetros utilizados para variar essa carga foram o comprimento das mensagens e o tempo de produção de mensagens.

Em cada grupo de simulações que deu origem a uma das figuras 4.2 e 4.3, apresentadas adiante, variou-se apenas um desses parâmetros e mediu-se o tempo em trânsito mínimo, médio e máximo das mensagens, mas apenas os tempos em trânsito médios são apresentados nessas figuras.

A figura 4.2 apresenta o tempo em trânsito médio em função do tempo médio de produção de mensagens para simulações realizadas com modelos de 36 nós, comprimento médio das mensagens igual a 512 *bytes* e o tempos médios de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.



Número de nós N = 36
Comprimento médio das mensagens $C_{médio}$ = 512

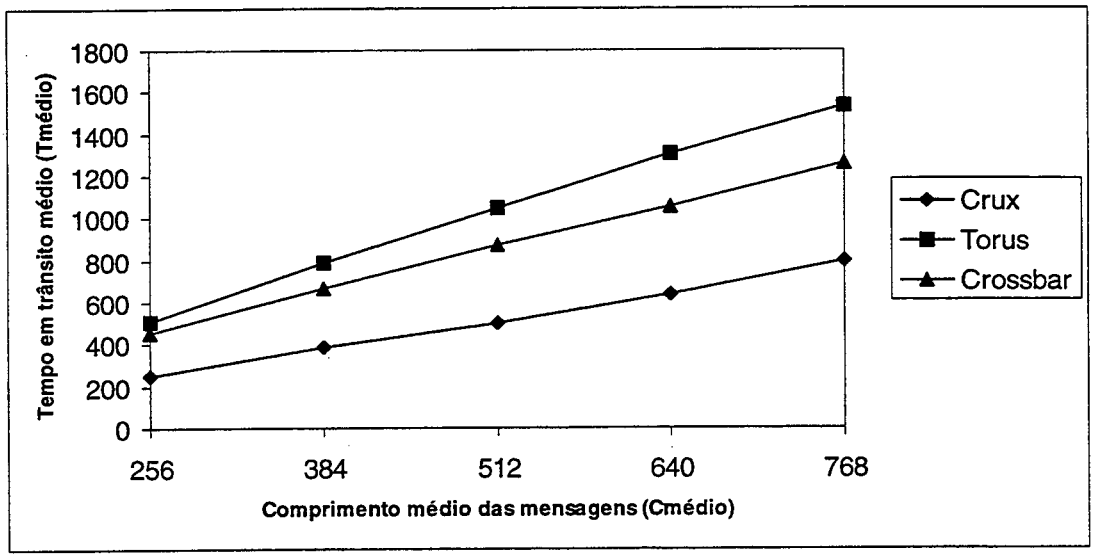
Figura 4.2: Comunicações generalizadas — $T_{médio}$ em função de $P_{médio}$.

Os resultados apresentados na figura 4.2 mostram, para as hipóteses estabelecidas para esse grupo de simulações, o desempenho claramente superior da rede de interconexão do Crux em relação às outras duas redes. Além disso, essa figura mostra que tanto o torus quanto o *crossbar* são sensíveis ao aumento do tráfego conseqüente à diminuição do tempo de produção de mensagens, sendo que o desempenho do torus é inferior e decresce mais rapidamente do que o do *crossbar*. Em contraposição, o Crux é praticamente insensível à variação de tráfego.

Esses resultados se justificam, para cada uma das redes de interconexão consideradas, da seguinte forma:

- No torus, quando duas ou mais mensagens disputam um mesmo canal físico, comum às rotas dessas mensagens, a transmissão delas é serializada.
- No *crossbar*, o envio simultâneo de mais de duas mensagem ao mesmo nó de destino provoca a multiplexação do único canal físico de entrada desse nó entre os pacotes dessas mensagens.
- No Crux, sempre existe um canal físico de entrada disponível no nó de destino se o número de mensagens simultâneas a ele destinadas não ultrapassa 4.

A figura 4.3 apresenta o tempo em trânsito médio em função do comprimento médio das mensagens para simulações realizadas com modelos de 36 nós, comprimentos médios das mensagens iguais a 256, 384, 512, 640 e 768 *bytes* e tempo de produção de mensagens igual a 1.0C unidades de tempo (P = C).



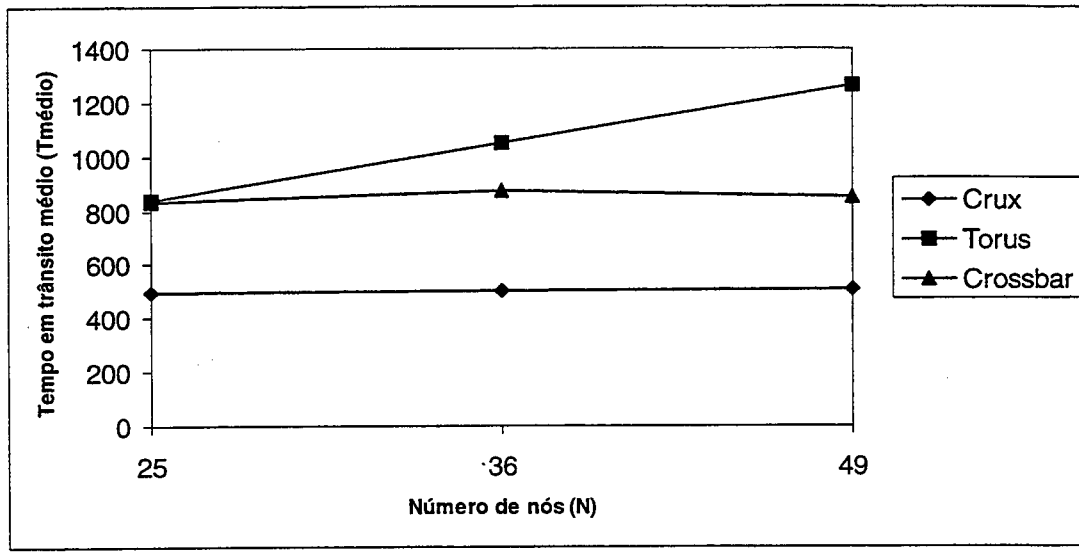
Número de nós N = 36
Tempo de produção de mensagens T = 1,0C

Figura 4.3: Comunicações generalizadas — $T_{médio}$ em função de $C_{médio}$.

Os resultados apresentados na figura 4.3 mostram que, para as hipóteses estabelecidas para esse grupo de simulações, o tempo em trânsito médio varia de forma aproximadamente linear em função do comprimento médio das mensagens nas três redes de interconexão, definindo três níveis nitidamente distintos de desempenho, decrescentes na seguinte ordem: Crux, *crossbar* e torus.

A figura 4.3 mostra ainda que no Crux o número que mede o tempo em trânsito médio é praticamente igual ao número que mede o comprimento das mensagens. Lembrando que a velocidade dos canais físicos é de 1 *byte* por unidade de tempo, esse resultado mostra a influência insignificante do procedimento de conexão dos canais físicos da rede de trabalho sobre o desempenho dessa rede. Como será visto adiante em 4.2.3, esta conclusão não se confirma quando as comunicações generalizadas envolvem mensagens muito curtas e freqüentes. Por outro lado, tanto no torus quanto no *crossbar*, o número que mede o tempo em trânsito médio é superior ao número que mede o comprimento das mensagens, o que resulta da disputa de seus canais físicos por mensagens distintas.

Para complementar as avaliações apresentadas nesta seção, é conveniente verificar também como o desempenho dessas redes de interconexão varia em função do tamanho da rede. Tanto o Crux quanto o *crossbar* podem crescer em incrementos de 1 nó. Entretanto, o torus cresce com o quadrado do número de nós em cada uma de suas duas dimensões. Por isso, os valores escolhidos para os tamanhos das redes consideradas respeitam o crescimento do torus. A figura 4.4 apresenta o tempo em trânsito médio das mensagens em função do número de nós para simulações realizadas com modelos de 25, 36 e 49 nós, comprimento médio das mensagens igual a 512 *bytes* e tempo de produção de mensagens T igual a 1,0C unidades de tempo.



Comprimento médio das mensagens $C_{médio} = 512$
Tempo de produção de mensagens T = 1,0C

Figura 4.4: Comunicações generalizadas — $T_{médio}$ em função de N.

Os resultados mostrados na figura 4.4 mostram que, para as hipóteses estabelecidas para esse grupo de simulações, os desempenhos do Crux e do *crossbar* independem do número de nós enquanto que o do torus decresce com o aumento do número de nós. Esse resultado reflete o fato de que tanto no Crux quanto no *crossbar* as comunicações são sempre diretas enquanto que no torus a distância média entre os nós cresce com o tamanho da rede. Essa figura confirma ainda o desempenho superior do Crux.

4.2.2.2 Comunicações Seletivas

O objetivo das simulações descritas nesta seção foi o de avaliar o desempenho relativo das três redes de interconexão consideradas, quando submetidas a um padrão de comunicação seletivo, definido pela distância — conforme a relação de distância do torus — entre os nós de origem e de destino das mensagens transmitidas.

Nas redes de interconexão estáticas, distâncias iguais a 1 entre os nós de origem e de destino de todas as mensagens transmitidas caracterizam o mapeamento perfeito. Quando o mapeamento perfeito é possível no torus, ele também é possível no Crux porque, conforme foi visto em 3.4, sua rede de trabalho pode assumir a configuração de qualquer rede de processos comunicantes cujos processos possuam grau igual ou inferior ao de seus nós. O conceito de mapeamento perfeito não se aplica ao *crossbar* conforme foi comentado em 3.1.4.

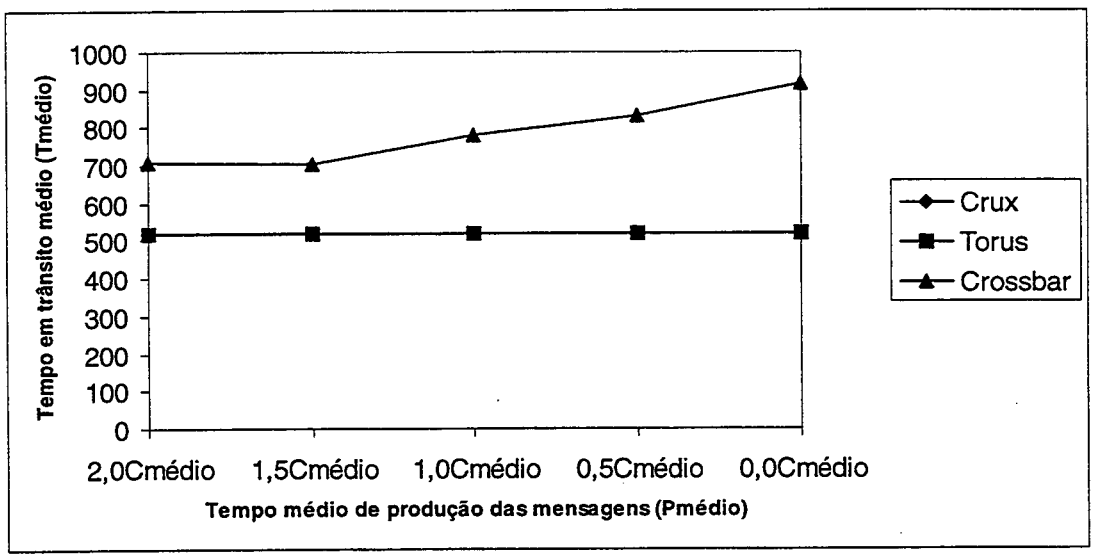
A medida que a distância cresce, o ajuste do mapeamento das redes de processos comunicantes diminui e o grau dos processos que as compõem aumenta. Distâncias moderadas, iguais a 2 ou 3, por exemplo, definem padrões de comunicação que podem ser considerados intermediários entre o mapeamento perfeito e as comunicações generalizadas.

O padrão de comunicação seletiva é caracterizado, para efeito das simulações realizadas, considerando que, para cada mensagem emitida, o nó de origem escolhe o nó de

destino entre os nós situados à distância D segundo uma distribuição uniforme. Na falta de relações de distância próprias, as simulações envolvendo o Crux e o *crossbar* usaram a relação de distância do torus na escolha dos nós de destino. O parâmetro considerado para variar essa carga foi apenas o tempo de produção de mensagens.

Em cada grupo de simulações que deu origem a uma das figuras 4.5, 4.6 e 4.7, apresentadas adiante, fixou-se a distância, variou-se o tempo de produção de mensagens e mediu-se os valores dos tempos em trânsito mínimo, médio e máximo das mensagens, mas apenas os valores dos tempos em trânsito médios são mostrados nessas figuras.

A figura 4.5 apresenta o tempo em trânsito médio em função do tempo médio de produção de mensagens para simulações realizadas com modelos de 36 nós, distância D igual a 1, comprimento médio das mensagens igual a 512 *bytes* e tempos médios de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.



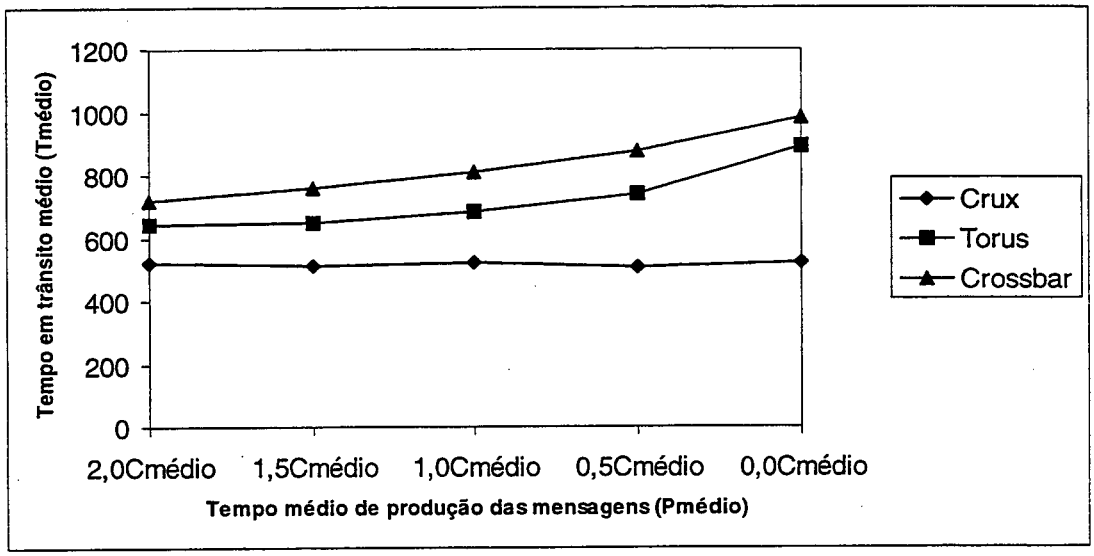
Número de nós N = 36
Comprimento médio das mensagens $C_{médio}$ = 512
Distância D = 1
(Observação: As curvas de desempenho do Crux e do torus estão sobrepostas)

Figura 4.5: Comunicações seletivas (D = 1) — $T_{médio}$ em função de $P_{médio}$.

Os resultados apresentados na figura 4.5 apenas comprovam, para as hipóteses estabelecidas para esse grupo de simulações, a constância do desempenho do Crux e do torus, uma vez que as mensagens são sempre transmitidas conforme a fórmula das comunicações diretas apresentada na tabela 3.1. Conforme foi visto em 3.1.4, esse resultado vale para qualquer rede de interconexão na qual se realiza o mapeamento perfeito. O desempenho decrescente do *crossbar* a medida que o tráfego aumenta resulta da disputa de seus canais físicos por mensagens distintas, conforme já foi constatado anteriormente.

A figura 4.6 apresenta o tempo em trânsito médio em função do tempo médio de produção de mensagens para simulações realizadas com modelos de 36 nós, distância igual a

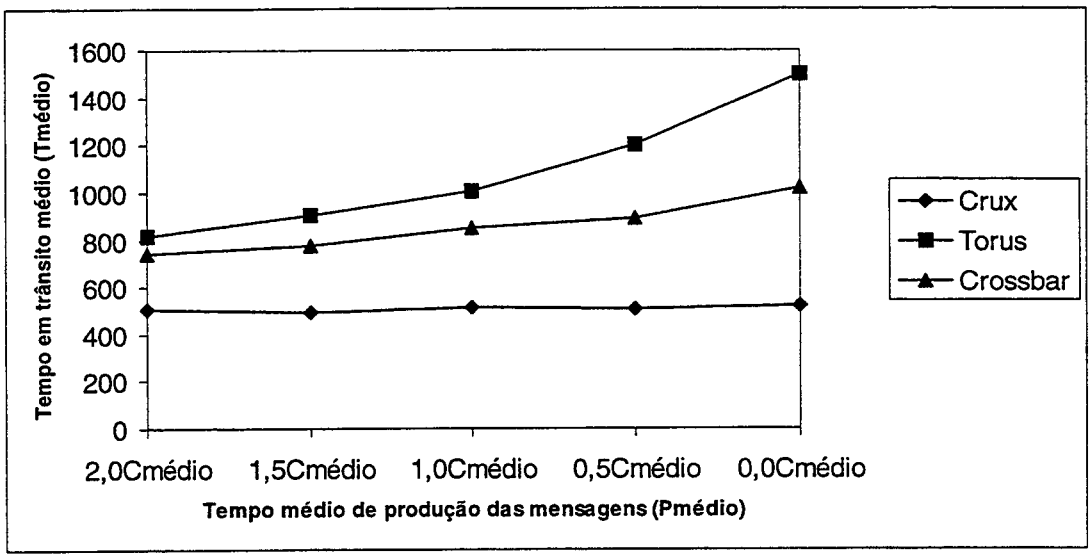
2, comprimento médio das mensagens igual a 512 *bytes* e tempo médio de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.



Número de nós N = 36
Comprimento médio das mensagens $C_{médio}$ = 512
Distância D = 2

Figura 4.6: Comunicações seletivas (D = 2) — $T_{médio}$ em função de $P_{médio}$.

A figura 4.7 apresenta o tempo em trânsito médio em função do tempo médio de produção de mensagens para simulações realizadas com modelos de 36 nós, distância igual a 3, comprimento médio das mensagens igual a 512 *bytes* e tempo médio de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.



Número de nós N = 36
Comprimento médio das mensagens $C_{médio}$ = 512
Distância D = 3

Figura 4.7: Comunicações seletivas (D = 3) — $T_{médio}$ em função de $P_{médio}$.

Os resultados apresentados nas figuras 4.6 e 4.7, combinados com os da figura 4.5, mostram que, para as hipóteses estabelecidas para esses grupos de simulações, o Crux é praticamente insensível à variação de tráfego conseqüente à variação do tempo de produção de mensagens. Em contraposição, essas mesmas figuras mostram também que tanto o torus quanto o *crossbar* são sensíveis ao aumento do tráfego, sendo que o desempenho do torus decresce mais rapidamente do que o do *crossbar*.

Como o conceito de distância não se aplica ao *crossbar*, suas curvas de desempenho apresentadas nas figuras 4.4, 4.5 e 4.6 são equivalentes. A influência da distância para o torus pode ser ressaltada pela inversão da posição relativa das curvas de desempenho do torus e do *crossbar* ao passar da distância 2 para a distância 3.

4.2.3 Avaliação Isolada

O objetivo das simulações descritas nesta seção foi o de avaliar especificamente a rede de interconexão do Crux em circunstâncias consideradas críticas para o seu desempenho.

Um dos inconvenientes presumidos do ambiente Crux, já citado em 3.4.3, diz respeito ao uso inadequado da rede de controle e do nó controlador. Uma disputa muito intensa por esses componentes pode transformá-los em gargalos do sistema. A carga de trabalho na qual isso mais claramente pode ocorrer envolve a troca generalizada e freqüente de mensagens curtas entre nós trabalhadores.

A figura 4.3, apresentada anteriormente em 4.2.2.1, mostra a variação de desempenho do Crux em função do comprimento de mensagens longas transmitidas conforme o padrão das

comunicações generalizadas. Nessa seção, essa avaliação é estendida, levando-se em conta mensagens cada vez mais curtas.

Considerando que os nós trabalhadores de origem escolham os nós trabalhadores de destino para os quais eles enviam mensagens segundo uma distribuição uniforme, a freqüência dos pedidos de conexão — gerando mensagens transmitidas pela rede de controle para serem manipuladas pelo nó controlador — cresce com a diminuição do comprimento das mensagens e com a diminuição do tempo de produção de mensagens, comprometendo o desempenho do Crux.

No grupo de simulações que deu origem à figura 4.8, apresentada a seguir, variou-se tanto o comprimento médio das mensagens quanto o tempo de produção de mensagens e mediu-se os valores dos tempos em trânsito mínimo, médio e máximo das mensagens, mas apenas os valores dos tempos em trânsito médios são mostrados na figura 4.8.

A figura 4.8 apresenta o tempo em trânsito médio em função do comprimento médio das mensagens para simulações realizadas com modelos de 36 nós, comprimentos médios das mensagens iguais a 256, 224, 192, 160, 128, 96, 64, 32, e 0 *bytes* e tempos médios de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.

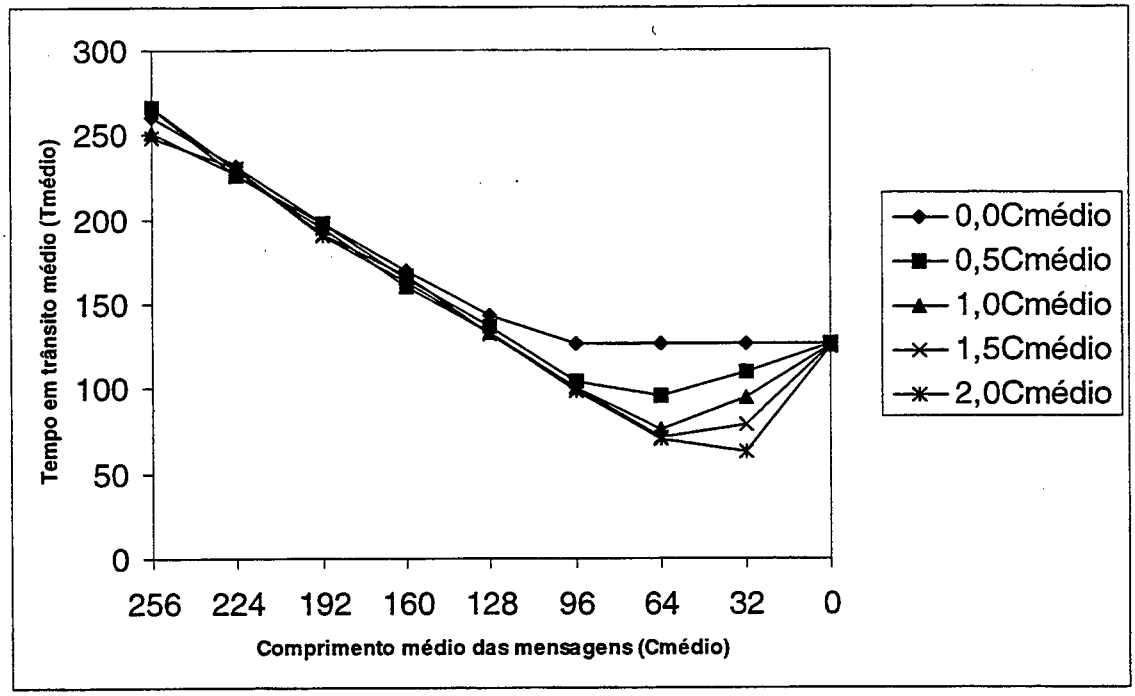


Figura 4.8: Crux — $T_{médio}$ em função de $C_{médio}$.

No grupo de simulações que deu origem à figura 4.9, apresentada a seguir, variou-se tanto o comprimento médio das mensagens quanto o tempo de produção de mensagens e

mediu-se os valores dos tempos de espera mínimo, médio e máximo para acesso à rede de controle, mas apenas os valores dos tempos de espera médios são mostrados na figura 4.9.

A figura 4.9 apresenta o tempo de espera médio para acesso à rede de controle em função do comprimento médio das mensagens para simulações realizadas com modelos de 36 nós, comprimentos médios das mensagens iguais a 256, 224, 192, 160, 128, 96, 64, 32, e 0 *bytes* e tempos médios de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.

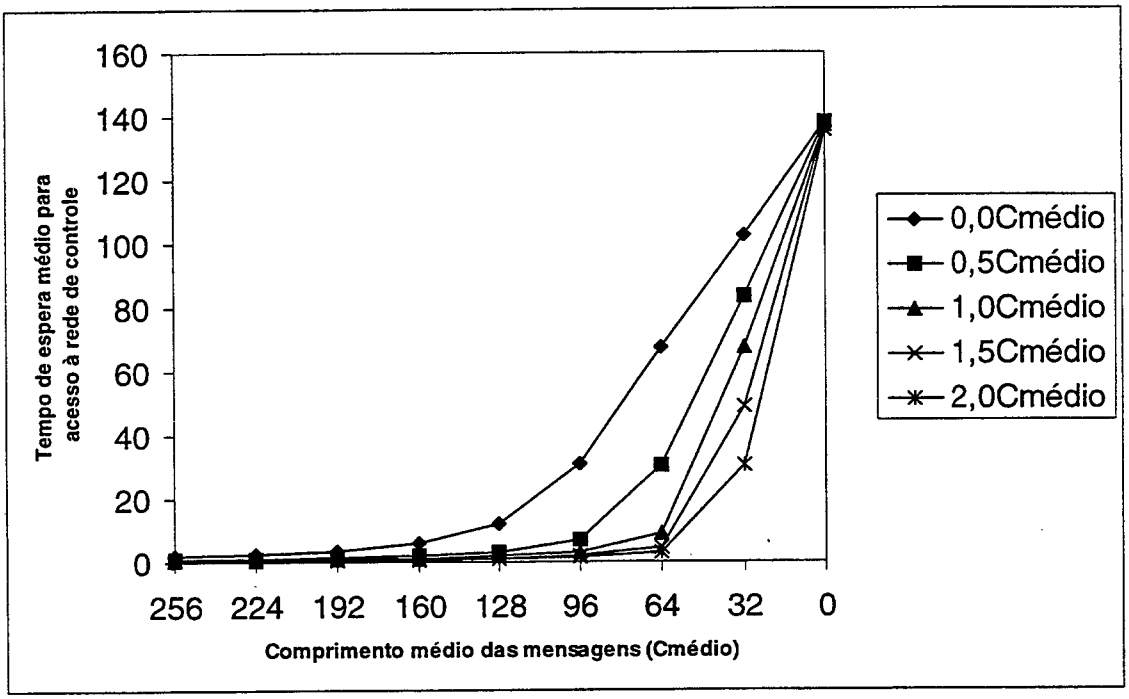


Figura 4.9: Crux — Tempo de espera para acesso à rede de controle em função de $C_{médio}$.

Os resultados apresentados na figura 4.8 mostram, para as hipóteses estabelecidas para esse grupo de simulações, comportamentos diferenciados conforme o comprimento médio das mensagens seja superior ou inferior a 128 *bytes*. Acima desse comprimento, os valores observados concordam com os da figura 4.3. Abaixo dele, verifica-se um aumento substancial do tempo em trânsito médio em relação ao comprimento médio das mensagens.

Os resultados apresentados na figura 4.9 ajudam a interpretar essa transição de comportamento. Para comprimentos médios das mensagens superiores a 128 *bytes*, o comportamento do Crux é muito pouco influenciado pelo tempo de espera pelo acesso à rede de controle e os valores observados concordam com os da figura 4.3. Para comprimentos médios das mensagens inferiores a 128 *bytes*, o fator determinante da composição do tempo em trânsito passa a ser cada vez mais o tempo de espera pelo acesso à rede de controle e cada vez menos pelo tempo de transmissão efetiva da mensagem pela rede de trabalho.

No limite, mensagens de comprimento nulo, e portanto de tempo de produção também nulo, representam um caso especial no qual cada processo emite um novo pedido de conexão imediatamente após ter seu pedido anterior atendido. Sendo assim, a cada instante quase todos os nós de trabalho estão esperando pelo acesso à rede de controle e o "tempo em trânsito" apresentado na figura representa praticamente o tempo de espera pelo acesso à essa rede.

A medida que o comprimento das mensagens cresce a partir de zero, a freqüência dos pedidos de conexão diminui porque cada processo passa a consumir mais tempo com a produção e a emissão de mensagens entre pedidos de conexão. Dessa forma, tanto o número de processos que ficam esperando pelo acesso à rede de controle a cada momento quanto o tempo de espera por essa rede diminuem. Entretanto, como o fator predominante para mensagens de comprimentos inferiores a 128 *bytes* é a disputa pelo acesso à rede de controle, em alguns casos, mensagens mais curtas produzem tempos em trânsito superiores aos de mensagens mais longas, como pode ser visto na figura 4.8.

4.3 Considerações Gerais sobre a Avaliação

As avaliações comparativas revelaram um desempenho global do Crux bastante superior ao do torus e do *crossbar*. Entretanto, alguns resultados merecem ainda alguns comentários adicionais.

A figura 4.5 mostra que o desempenho do Crux e do torus são equivalentes quando o mapeamento perfeito das redes de processos comunicantes é realizado em ambas as redes de interconexão, embora o esforço para obtê-lo seja bastante distinto. Enquanto no torus ele pode ser muito complexo, sobretudo para redes de processos comunicantes de topologia variável, no Crux ele está naturalmente embutido no mecanismo de mapeamento de canais lógicos por demanda. Além disso, o número de topologias de redes de processos comunicantes para as quais é possível o mapeamento perfeito no Crux é superior ao do torus. Efetivamente, basta que os processos possuam graus inferiores ou iguais aos graus dos nós de trabalho para que uma rede de processos comunicantes possa ser perfeitamente mapeada no Crux.

O mapeamento perfeito de processos é igualmente simples no Crux e no *crossbar*. Entretanto, o mapeamento perfeito de canais lógicos é impossível no *crossbar*. O desempenho inferior do *crossbar*, resulta em todos os casos da multiplexação dos canais físicos entre pacotes de mensagens distintas.

O controle centralizado do Crux somente se mostrou inconveniente em situações anômalas, distantes das aplicações visadas. Assim, apenas a conjunção de comunicações generalizadas e mensagens curtas produziram os resultados críticos mostrados na figura 4.8.

4.4 Conclusões

Os resultados das simulações apresentadas nas seções precedentes representam valores típicos do desempenho relativo das redes consideradas, selecionados de um conjunto muito maior de simulações omitidas neste texto.

Apesar de resultados tão positivos, a avaliação de desempenho é apenas um dos fatores da apreciação global de um ambiente multicomputador. A flexibilidade de operação e a simplicidade de utilização são também critérios fundamentais. A reunião desses critérios é ainda mais favorável ao Crux, porque ao desempenho superior verificado neste capítulo, pode se acrescentar ainda a flexibilidade e a simplicidade extraordinárias já enunciadas em 3.4.3 e materializadas pelas aplicações descritas no capítulo 5.

5 Aplicações para o Ambiente Crux

Para corroborar a efetividade do ambiente para execução de programas paralelos descrito neste texto, é importante acrescentar aos componentes básicos — o multicomputador Crux e seu mecanismo de conexão dinâmica de canais físicos — os serviços habituais de um sistema operacional e suporte específico para linguagens paralelas. O objetivo principal deste capítulo é o de descrever os projetos do sistema operacional ACrux e do interpretador paralelo para a linguagem Superpascal, desenvolvidos como dois exemplos demonstrativos da flexibilidade dos componentes básicos para se adaptar a requisitos operacionais distintos e da simplicidade de atendimento de cada requisito diferente. Neste capítulo são ainda citados vários resultados experimentais oriundos deste trabalho que, materializados em diversas implementações, servem como aval adicional à viabilidade do ambiente proposto.

5.1 Sistema Operacional ACrux

O sistema ACrux é um sistema operacional experimental com interface de programação equivalente à do sistema Unix que pode ser visto complementarmente como camadas de *software* hierarquicamente superpostas, definidas pela especificação das suas interfaces, e como uma rede cliente-servidor, na qual os processos distribuídos fisicamente pelos nós do multicomputador trocam mensagens através de um mecanismo específico de comunicação com conexão dinâmica de canais físicos.

5.1.1 Visão Geral

No ambiente do sistema operacional ACrux, podem ser identificados três tipos de processos: servidor do núcleo, servidor do sistema e processo de usuário. O *servidor do núcleo* é um processo único responsável pela execução dos serviços referentes à conexão/desconexão de canais físicos da rede de trabalho e à atribuição/liberação de nós trabalhadores. *Servidores do sistema* são processos responsáveis pela execução dos serviços equivalentes aos do sistema Unix. O servidor do núcleo e os servidores do sistema são processos perenes. *Processos de usuário* são processos que compõem os programas paralelos de usuário. Os processos de usuário estão sujeitos ao mecanismo de criação dinâmica de processos do sistema Unix.

Esses processos integram diversas redes de processos comunicantes, conforme a figura 5.1. A *rede do núcleo* é formada pelo servidor do núcleo e por todos os demais processos. A

rede do sistema é formada pelos servidores do sistema e pelos processos de usuário. As *redes de usuário* são formadas exclusivamente por processos de usuário. Essas redes não são independentes uma vez que um mesmo processo pode participar simultaneamente de mais de uma delas.

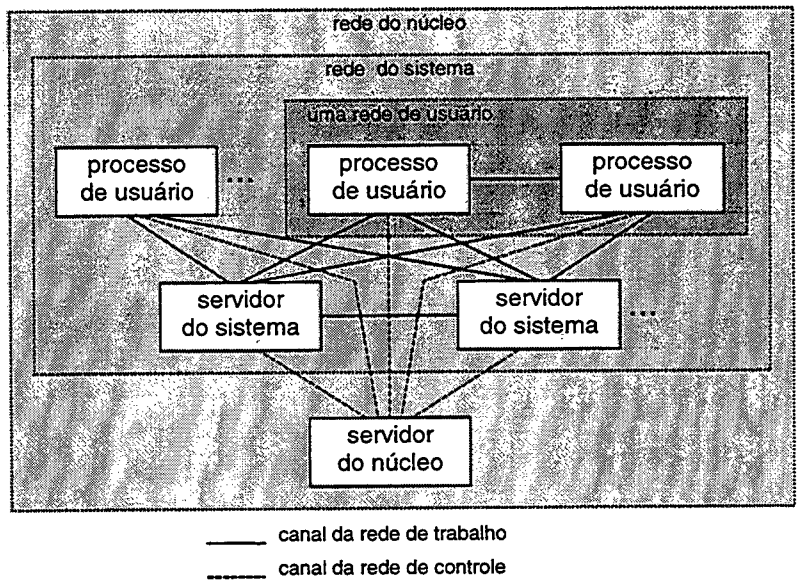


Figura 5.1: Redes de processos comunicantes no ambiente do sistema ACrux.

Tanto a rede do núcleo quanto a rede do sistema são construídas com canais lógicos síncronos bidirecionais dedicados. Os canais lógicos da rede do núcleo e da rede do sistema que não envolvem processos de usuário são perenes. Os canais lógicos da rede do núcleo e da rede do sistema que envolvem processos de usuário têm duração igual a dos processos de usuário. Os canais lógicos das redes de usuário têm duração determinada por essas próprias redes.

A atribuição do servidor do núcleo ao nó controlador é imposta pela arquitetura do multicomputador Crux porque tanto o dispositivo de configuração da rede de trabalho quanto o dispositivo de gerência da rede de controle estão fisicamente associados a esse nó, conforme a figura 3.4. Por outro lado, a atribuição dos servidores do sistema e dos processos de usuário a nós trabalhadores é arbitrária.

O sistema operacional ACrux compreende a rede do sistema e a rede do núcleo. Os serviços do sistema são acessíveis aos processos de usuário através da rede do sistema. Os serviços do núcleo são acessíveis aos processos de usuário e aos servidores do sistema através da rede do núcleo.

A rede do sistema é mapeada na rede de trabalho e a rede do núcleo é mapeada na rede de controle, conforme a figura 5.1. Assim, a estrutura do sistema operacional ACrux, caracterizada por duas redes de processos comunicantes fisicamente distribuídas pelos nós do multicomputador, cada uma das quais usando apenas uma das duas redes de interconexão do

multicomputador, se articula perfeitamente à arquitetura do multicomputador Crux e ao seu mecanismo de conexão dinâmica de canais físicos.

Do ponto de vista da composição dos processos, a rede do sistema e a rede do núcleo estão representadas em duas camadas superpostas, conforme a figura 5.2. A camada do sistema está presente nos processos de usuário e nos servidores do sistema. A camada do núcleo está presente em todos os processos.

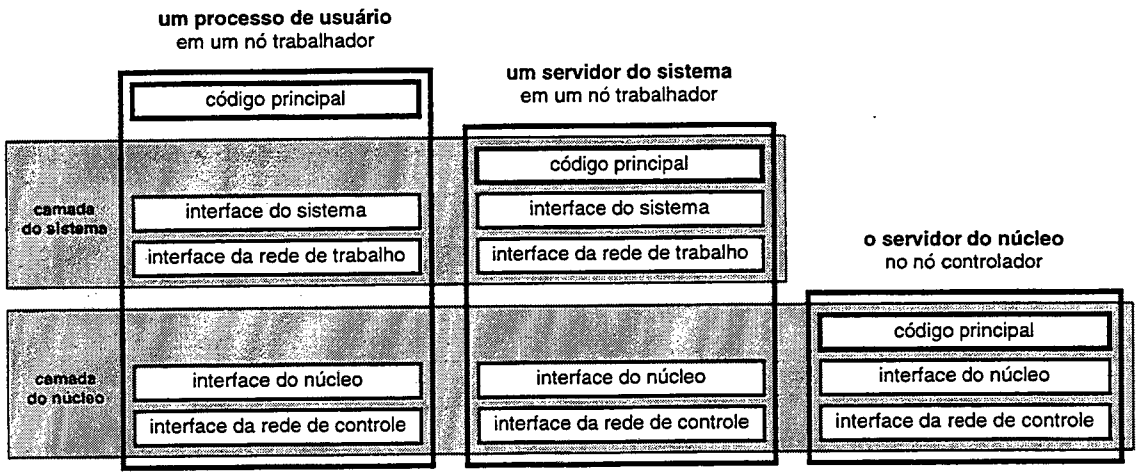


Figura 5.2: Composição dos processos do sistema ACrux.

As interações entre os processos de usuário e os servidores do sistema seguem na maioria das vezes o padrão da disciplina de comunicação da chamada de procedimento remoto. A camada do sistema contém os *stubs* do cliente e do servidor do sistema. Coletivamente, os *stubs* do cliente e do servidor dessa camada são referidos por *interface do sistema.*

Desvios do padrão da disciplina de comunicação da chamada de procedimento remoto descrita em 2.1.2.4 são inevitáveis na camada do sistema, por exemplo, para a realização dos operadores referentes à criação/remoção de processos. Procedimentos especiais devem então ser adotados para tratar esses casos.

As interações entre o servidor do núcleo e os demais processos seguem sempre o padrão das chamadas de procedimentos remotos. A camada do núcleo contém *stubs* para comunicação entre os clientes e o servidor do núcleo. Coletivamente, os *stubs* do cliente e do servidor dessa camada são referidos por *interface do núcleo.*

Para atender às necessidades de comunicação das duas camadas do sistema operacional ACrux, um mecanismo de comunicação específico com conexão dinâmica de canais físicos foi integrado à composição desse sistema. A rede do sistema é suportada por operadores de comunicação específicos da rede de trabalho e a rede do núcleo é suportada por operadores de comunicação específicos da rede de controle. Coletivamente, os operadores da rede de trabalho são referidos por *interface da rede de trabalho* e os operadores da rede de controle são referidos por *interface da rede de controle.*

5.1.2 Camadas do Sistema ACrux

As duas camadas identificadas na figura 5.2 são detalhadas na figura 5.3 pela inclusão dos operadores que compõem suas interfaces e do relacionamento lógico entre essas interfaces. A figura 5.3 não impõe entretanto características específicas de implementação desses componentes.

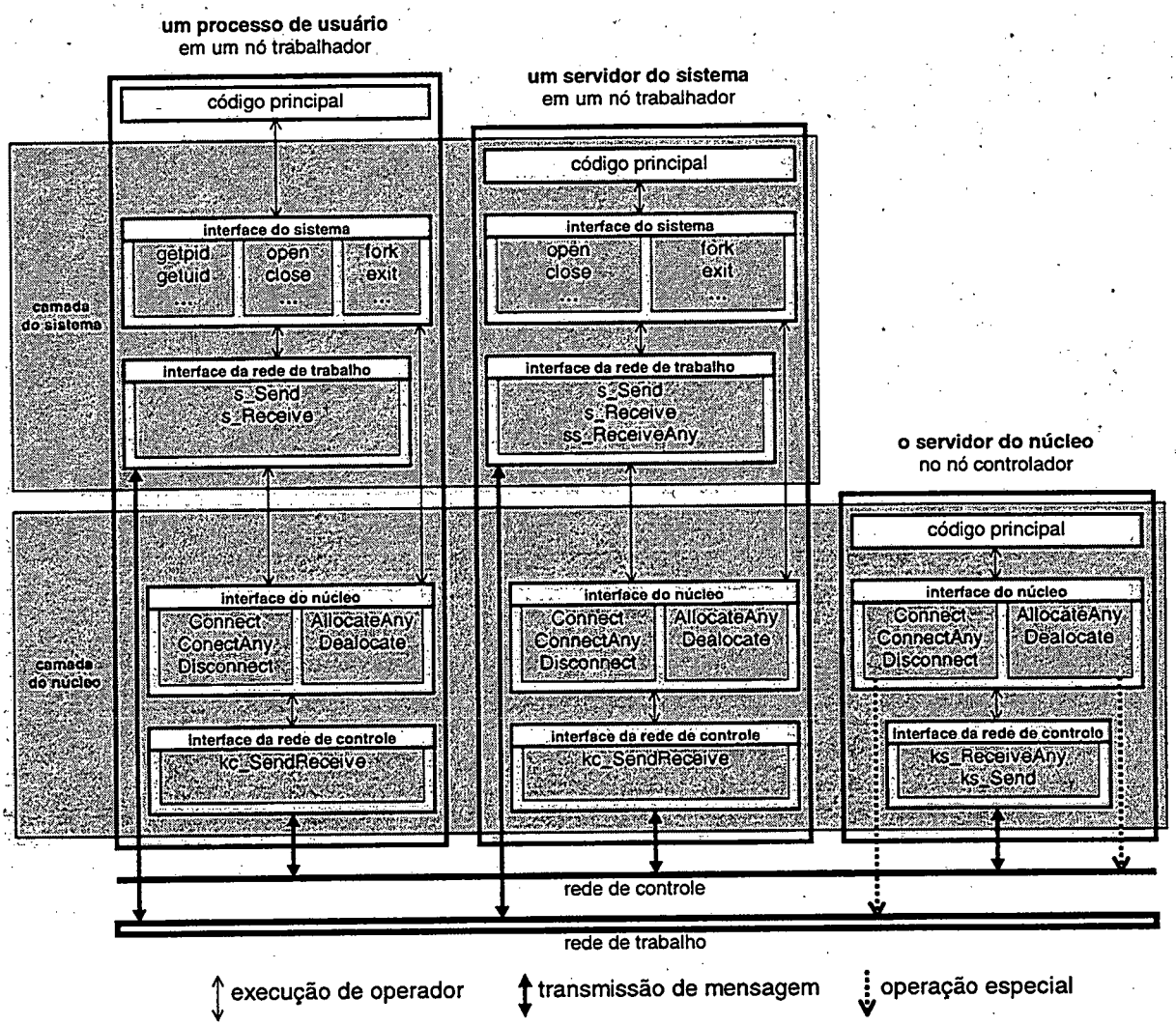


Figura 5.3: Camadas do sistema ACrux.

A camada do sistema e a camada do núcleo são apresentadas a seguir através da descrição dos serviços por elas prestado e dos operadores que compõem suas interfaces.

5.1.2.1 Camada do Sistema

Interface do Sistema

A interface do sistema fornece operadores equivalentes aos do sistema Unix, compatíveis com a norma Posix [IEE88]. Do ponto de vista da implementação, os operadores da interface do sistema podem ser divididos em três grupos: locais, regulares e especiais. Um *operador local* é totalmente executado no próprio nó onde o processo executa. Um exemplo de operador local é o getpid. Um *operador regular* segue o padrão das chamadas de procedimentos remotos nas interações entre os processos de usuário e os servidores do sistema. Um exemplo de operador regular é o open. Cada *operador especial* segue um padrão específico de interação entre o processo de usuário e os servidores dos sistema. Um exemplo de operador especial é o fork. (Exemplos de implementação de operadores especiais são apresentados adiante em 5.3.1.4.)

Interface da Rede de Trabalho

A interface da rede de trabalho visa exclusivamente a troca de mensagens entre os processos de usuário e os servidores do sistema através da rede do sistema.

Os operadores que compõem essa interface são os seguintes:

- s_Send (proc, msg, length)
- s_Receive (proc, msg, length)
- ss_ReceiveAny (proc, msg, length)

Um processo usa o operador s_Send para enviar pelo canal lógico que o conecta ao processo proc a mensagem msg de tamanho length. Um processo usa o operador s_Receive para receber pelo canal lógico que o conecta ao processo proc uma mensagem msg com tamanho máximo length. Um processo usa o operador ss_ReceiveAny para receber de um processo qualquer pelo canal lógico que o conecta a ele uma mensagem msg com tamanho máximo length. A identificação do processo proc é um parâmetro de retorno desse operador. Os operadores com prefixo s_ estão disponíveis tanto aos processos de usuário quanto aos servidores do sistema enquanto que o operador com prefixo ss_ está disponível apenas aos servidores do sistema.

Os operadores dessa interface designam processos através do parâmetro proc. A identificação de cada servidor do sistema é fixa e conhecida de todos os processos de usuário. A identificação do processo de usuário requisitante de um serviço é provida aos servidores do sistema através do parâmetro de retorno proc do operador ss_ReceiveAny.

A identificação do nó trabalhador onde um processo executa é um dos componentes da identificação do processo para simplificar a designação de nós usada nos operadores da camada do núcleo.

Cada servidor do sistema forma com os processos de usuário uma rede estrela. Assim, o mecanismo de comunicação deve prever uma transação de conexão (definida em 3.4.2) para cada transmissão de mensagem. Os operadores definidos são mais gerais do que os estritamente necessários ao padrão de comunicação cliente-servidor para permitir também a

implementação dos operadores especiais da interface do sistema que têm padrões de interação diferenciados.

5.1.2.2 Camada do Núcleo

Interface do Núcleo

Os serviços da interface do núcleo permitem a conexão/desconexão de canais físicos da rede de trabalho e a atribuição/liberação de nós trabalhadores.

Os operadores para efetuar conexões e desconexões através da rede de trabalho que compõem essa interface são os seguintes:

- Connect (node, link)
- ConnectAny (node, link)
- Disconnect (node)

Um nó trabalhador usa o operador Connect para estabelecer uma conexão pela rede de trabalho entre ele e o nó trabalhador node através do canal físico link. Um nó trabalhador usa o operador ConnectAny para estabelecer uma conexão pela da rede de trabalho entre ele e um nó trabalhador qualquer node através do canal físico link. A identificação do nó node é um parâmetro de retorno desse operador. A identificação do canal físico link é um parâmetro de retorno tanto do operador Connect quanto do operador ConnectAny. Um nó trabalhador usa o operador Disconnect para desfazer a conexão pela rede de trabalho que o conecta ao nó trabalhador node.

Uma conexão só é estabelecida entre dois nós trabalhadores por acordo recíproco mas pode ser rompida unilateralmente por decisão de qualquer um dos dois nós trabalhadores conectados. Apenas uma conexão física é admitida entre dois nós trabalhadores quaisquer a cada momento. Todos esses operadores provocam a realização de operações especiais de configuração da rede de trabalho por parte do servidor do núcleo.

Os operadores para atribuir e liberar nós trabalhadores que compõem essa interface são os seguintes:

- AllocateAny (node)
- Deallocate ()

Um processo usa o operador AllocateAny para obter um nó trabalhador qualquer cuja identificação node é um parâmetro de retorno. Um processo usa o operador Deallocate para liberar o nó onde ele está executando.

Esses operadores são usados pela camada do sistema na implementação dos operadores fork e exit. A chamada de sistema fork usa AllocateAny para obter um nó trabalhador livre para executar o novo processo. O operador exit usa Deallocate para liberar o nó trabalhador ocupado pelo processo que deixa de existir. Dessa forma, integra-se ao sistema ACrux o mapeamento perfeito de processos.

Interface da Rede de Controle

A interface da rede de controle visa exclusivamente a troca de mensagens entre o nó controlador executando o servidor do núcleo e os nós trabalhadores executando servidores do sistema ou processos de usuário através da rede do núcleo. As comunicações pela rede de controle possuem sempre o nó controlador como origem ou destino.

Os operadores que compõem essa interface são os seguintes:

- kc_SendReceive (request, reply)
- ks_ReceiveAny (node, request)
- ks_Send (node, reply)

Um nó trabalhador usa o operador kc_SendReceive para enviar ao nó controlador a mensagem de requisição request e receber dele a mensagem de resposta reply. O nó controlador usa o operador ks_ReceiveAny para receber de um nó trabalhador qualquer node uma mensagem request. A identificação do nó node é um parâmetro de retorno desse operador. O nó controlador usa o operador ks_Send para enviar ao nó trabalhador node a mensagem reply. O operador com prefixo kc_ está disponível apenas aos nós trabalhadores e os operadores com prefixo ks_ estão disponíveis apenas ao nó controlador.

O servidor do núcleo forma com os demais processos uma rede estrela. Os operadores definidos são os estritamente necessários ao padrão de comunicação cliente-servidor. A identificação do nó trabalhador requisitante de um serviço é provida ao servidor do núcleo através do parâmetro de retorno node do operador ks_ReceiveAny.

Todas a mensagens que transitam pela rede de controle possuem comprimento fixo muito pequeno, igual à largura do barramento (16 *bits*, por exemplo). Esses operadores devem ser simples extensões do mecanismo previsto para o *hardware* da rede de controle do multicomputador Crux.

5.1.3 Exemplo

Pode ser esclarecedor acompanhar através de um exemplo a execução de um operador regular da interface do sistema para compreender as interações que se desenvolvem na rede do sistema e na rede do núcleo do sistema operacional ACrux.

A execução de um operador regular da interface do sistema, como o open, por exemplo, segue o padrão das chamadas de procedimentos remotos descrito em 2.3.2.1. As mensagens são trocadas entre os componentes da interface do sistema pela rede do sistema conforme a seguinte seqüência:

- A interface do cliente do sistema envia à interface do servidor do sistema a mensagem de requisição correspondente ao operador regular open através do operador s_Send.
- Essa requisição é recebida pela interface do servidor do sistema através do operador ss_ReceiveAny.
- A interface do servidor do sistema envia à interface do cliente do sistema a mensagem de resposta através do operador s_Send.

- Essa resposta é recebida pela interface do cliente do sistema através do operador s_Receive.

Antes que uma mensagem possa ser transmitida entre um processo de usuário e um servidor do sistema pela rede do sistema, uma conexão entre os nós onde eles executam deve ser estabelecida na rede de trabalho. Os próprios operadores s_Send e ss_ReceiveAny podem executar o operador do núcleo Connect para solicitar a conexão necessária.

A execução de um operador da interface do núcleo como Connect, por exemplo, também segue o padrão das chamadas de procedimentos remotos. As mensagens nesse caso são trocadas entre um processo qualquer e o servidor do núcleo pela rede do núcleo conforme a seguinte seqüência:

- A interface do cliente do núcleo envia à interface do servidor do núcleo a mensagem de requisição correspondente ao operador Connect através do operador kc_SendReceive.
- Essa requisição é recebida pela interface do servidor do núcleo através do operador ks_ReceiveAny.
- A interface do servidor do núcleo envia à interface do cliente do núcleo a mensagem de resposta através do operador ks_Send.
- Essa resposta é recebida pela interface do cliente do núcleo através do operador kc_SendReceive.

As mensagens entre um processo qualquer e o servidor do núcleo pela rede do núcleo são sempre transmitidas através da rede de controle.

A partir das duas seqüências apresentadas acima seria simples (mas repetitivo) completar as seqüências faltantes para se descrever integralmente a execução de um operador regular. Em todo caso, a descrição dessas interações é complementada adiante em 5.3.1.3, enfatizando as interações entre os processos envolvidos, no contexto de um experimento de implementação específico.

As duas seqüências apresentadas acima respeitam exatamente o modelo cliente-servidor subjacente às chamadas de procedimentos remotos. A estrutura resultante é entretanto singular tanto pela sobreposição lógica das duas redes cliente-servidor quanto pelo emprego de dois suportes físicos de comunicação distintos, um para cada uma dessas redes.

5.1.4 Considerações sobre o Sistema Operacional

Os componentes mais volumosos do sistema ACrux são os servidores do sistema, sobretudo pela extensão e complexidade do sistema de arquivos do Unix. Em contraste, os demais componentes são todos extremamente compactos em função do reduzido número de serviços e da enorme simplicidade dos operadores de suas interfaces.

De fato, enquanto a interface do sistema possui pelo menos 50 operadores principais de complexidade variável, a interface da camada do núcleo possui apenas 5 operadores muito simples. Os operadores da interface do núcleo lidam com a atribuição de canais físicos a canais lógicos e de nós a processos. Tanto a escolha de canais físicos quanto a de nós são arbitrárias. Algoritmos associados às escolhas arbitrárias são elementares e as estruturas de

dados a eles associadas são muito simples. Além disso, as interfaces de comunicação do sistema e do núcleo possuem apenas 3 operadores cada uma, todos também muito simples.

A interface de comunicação do sistema atende apenas às necessidades de comunicação da rede do sistema. Várias possibilidades podem ser consideradas para atender as necessidades de comunicação das redes de usuário, além dos mecanismos de comunicação providos pela interface de programação do sistema Unix (a exemplo dos *pipes* e dos *sockets* [BAC86]) e daqueles que são normalmente construídos sobre essa interface (a exemplo do PVM [SUN93] e do MPI [WAL94]). As alternativas consideradas a seguir consistem na extensão da interface do sistema com operadores de comunicação.

A alternativa mais imediata seria estender a interface do sistema com operadores equivalentes aos da interface de comunicação do sistema ACrux. Entretanto, esses operadores não são adequados para um grande número de redes de usuários de interesse prático, nas quais o grau dos processos é pequeno e os canais lógicos têm longa duração.

Outra alternativa seria a definição de uma nova interface de comunicação genérica para atender convenientemente tanto as necessidades da rede do sistema quanto a de um grande número de redes de usuário.

Ainda outra alternativa poderia permitir a coabitação da interface de comunicação do sistema para uso exclusivo do sistema ACrux com outras interfaces de comunicação especificamente concebidas para as necessidades de cada linguagem considerada. Essa foi a alternativa adotada na implementação do interpretador para a linguagem Superpascal descrita adiante em 5.2.

Estima-se que essas alternativas representem as soluções mais eficientes para o ambiente ACrux porque elas podem ser implementadas diretamente sobre o mecanismo de conexão dinâmica de canais físicos próprio do ambiente Crux.

5.2 Interpretador para a Linguagem Superpascal

5.2.1 Linguagem Superpascal

Superpascal [HAN94a] estende os conceitos de programação paralela de CSP e os associa aos elementos seqüenciais de Pascal para dar origem a uma linguagem voltada à expressão de programas científicos paralelos como redes de processos comunicantes.

Os elementos de Superpascal introduzidos a seguir dizem respeito apenas às entidades do paralelismo. Seus elementos seqüenciais, herdados de Pascal são omitidos nesta descrição.

Processos são criados dinamicamente em Superpascal através das instruções parallel e forall.

A instrução parallel s_1 | s_2 | ... | s_n end gera um processo para cada instrução s_i, i=1,n. Todos os processos criados executam em paralelo e a instrução parallel termina quando todos os processos s_i terminam. O processo que executa a instrução parallel fica suspenso durante a execução dessa instrução.

A instrução forall i := ei to ef do s gera um conjunto de processos, cada um dos quais executa um exemplar da instrução s. O número de processos criados é definida pelo intervalo das expressões inteiras ei e ef. Cada processo possui um exemplar local da variável i, à qual é atribuído um valor diferente (de ei a ef) chamado de índice do processo. A instrução forall termina quando todos os processos por ela criados terminam. O processo que executa a instrução forall fica suspenso durante a execução da mesma.

Processos se comunicam em Superpascal por transmissão de mensagens através de canais lógicos síncronos criados dinamicamente pelo operador open e referidos através de variáveis de tipo channel.

A chamada open (c) gera um canal lógico e retorna à variável c de tipo channel uma referência ao canal lógico criado. A partir daí, as referências a esse canal podem ser feitas através da variável c. Uma vez criado, um canal lógico existe por toda a duração do programa paralelo.

A troca de mensagens entre processos se apoia sobre os operadores send e receive. A chamada send (c, e) provoca a emissão da mensagem resultante da avaliação da expressão e através do canal lógico referido pela variável c. A chamada receive (c, v) provoca a recepção de uma mensagem pelo canal lógico referido pela variável c cujo valor é atribuído à variável v.

O sentido de um canal lógico usado para a comunicação unidirecional entre dois processos só é estabelecido quando eles formam um par send-receive pela primeira vez. Dois processos formam um par send-receive quando um deles usa o operador send e o outro o operador receive designando o mesmo canal lógico. No momento em que ambos processos se encontram na execução dessas operações, uma mensagem é transmitida diretamente do espaço de endereçamento do processo emissor para o do processo receptor.

Apesar da linguagem Superpascal ter sido projetada especificamente para expressar redes de processos comunicantes a serem executadas em multicomputadores, o *software* distribuído atualmente pelo seu autor [HAN94b] consta de dois programas seqüenciais desenvolvidos para estações de trabalho Unix: um compilador e um interpretador. O *compilador* gera código intermediário para uma máquina virtual e o *interpretador* executa as instruções dessa máquina virtual.

Tanto o compilador quanto o interpretador foram escritos em Pascal seqüencial. O interpretador executa como um único processo Unix, simulando em seu interior o paralelismo provido pela linguagem. Sendo assim, o programa paralelo que ele interpreta não está sujeito a condições de corrida (*race condition*) [SIL89].

5.2.2 Interpretador Paralelo

A implementação eficiente de uma linguagem paralela que, como Superpascal, permite a criação dinâmica tanto de processos quanto de canais lógicos em ambientes multicomputadores convencionais é uma tarefa complexa [AND89]. Tanto é assim, que o autor de Superpascal se submete às seguintes tarefas para compor, depurar e publicar algoritmos nessa linguagem [HAN95]:

1. O programa é escrito inicialmente em Superpascal e testado em uma estação de trabalho usando o compilador e o interpretador já citados.
2. Quando o programa é considerado correto, ele é traduzido manualmente para Occam2 e testado novamente em um multicomputador.
3. Se na etapa 2, o programa sofreu correções, elas são transferidas manualmente ao programa em Superpascal para ser executado novamente na estação de trabalho.

A implementação do interpretador paralelo para a linguagem Superpascal no multicomputador Crux visa a execução de programas escritos nessa linguagem de tal forma que cada processo Superpascal seja executado como um processo Unix atribuído a um nó trabalhador distinto e que os processos desses programas troquem mensagens entre si através da rede de trabalho, usando um mecanismo de comunicação específico com conexão dinâmica de canais físicos. Além disso, o compilador da linguagem Superpascal — e portanto o código por ele gerado — não devem ser alterados.

A disposição física do interpretador paralelo no multicomputador Crux é semelhante à do núcleo do sistema operacional ACrux. Ele consta de um interpretador local a cada nó trabalhador e de um interpretador central atribuído ao nó controlador. Esses componentes compõem processos que formam uma rede estrela em torno do interpretador central. As comunicações que se desenvolvem nessa rede seguem o padrão cliente-servidor. O suporte físico dessas comunicações é a rede de controle.

O interpretador local é o próprio interpretador seqüencial original em relação ao grande número de instruções executadas localmente. Apenas o reduzido número de instruções correspondentes à criação dinâmica de processos ou de canais lógicos e à troca de mensagens — que envolvem interações com o interpretador central — foi modificado. Em relação à criação dinâmica de processos, o interpretador local deve interagir com e usar os recursos do sistema operacional ACrux; em relação à criação de canais lógicos e à transmissão de mensagens, ele deve interagir com o interpretador central.

Ao invés de precisar os operadores e suas funções como foi feito em 5.1 para o sistema operacional, a descrição do interpretador paralelo é conduzida pelos eventos significativos da execução de um programa simples com o objetivo principal de mostrar como a rede de trabalho do multicomputador Crux é configurada em tempo de execução para corresponder à rede de processos comunicantes expressa pelo programa Superpascal.

A descrição pressupõe a existência do sistema ACrux e que o mecanismo de comunicação específico da linguagem Superpascal seja integrado ao núcleo desse sistema operacional. Em nenhum momento perseguiu-se neste trabalho o objetivo de definir um mecanismo de comunicação genérico que suportasse tanto as necessidades do sistema operacional quanto as da linguagem Superpascal.

Apesar de simples, o programa da figura 5.4 é completo para os propósitos desta descrição porque envolve todos os recursos da linguagem Superpascal que dizem respeito às entidades do paralelismo. Ele consta de um processo Pai que cria, através de uma instrução parallel, dois processos filhos: o Produtor e o Consumidor. Antes disso, ele cria um canal lógico e atribui à variável canal uma referência ao canal criado. Produtor e Consumidor se comunicam através desse canal durante um certo número de iterações. Quando ambos terminam, o Pai retoma sua própria execução e termina em seguida. A rede de processos comunicantes gerada por esse programa aparece na figura 5.5.

```
program PC;

type vetor : array 1 ..1000 of integer;
type fluxo : *(vetor);
var canal : fluxo;

procedure Pai;
begin
  var canal : fluxo;
  open (canal);
  parallel
    Produtor (canal) |
    Consumidor (canal)
  end
end;
```

```
procedure Produtor (saida : fluxo);
var mensagem : vetor;
begin
  for i := 1 to 1000 do
    begin
      ... # produza mensagem
      send (saida, mensagem)
    end
end;

procedure Consumidor (entrada : fluxo);
var mensagem : vetor;
begin
  for i := 1 to 1000 do
    begin
      receive (entrada, mensagem);
      ... # consuma mensagem
    end
end;

end PC.
```

Figura 5.4: Programa em Superpascal.

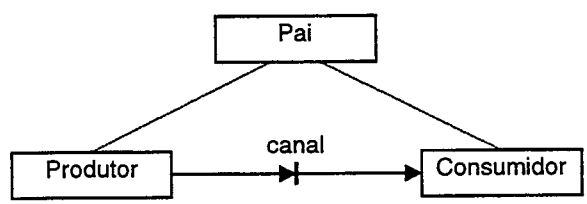


Figura 5.5: Rede de processos comunicantes gerada pelo programa em Superpascal.

Quando o Pai executa a chamada open (canal), o interpretador local requisita do interpretador central a criação de um canal lógico. A principal estrutura de dados manipulada pelo interpretador central é a *tabela de canais lógicos*. O interpretador central obtém uma

entrada livre nessa tabela, a inicializa e devolve seu índice ao interpretador local. A partir daí, as referências ao canal lógico são feitas através desse índice. A criação de um canal lógico é representada pela obtenção e inicialização de uma entrada da tabela de canais lógicos.

Quando o Pai executa a instrução parallel, ele usa duas vezes a chamada de sistema fork: uma para criar o Produtor e outra para criar o Consumidor. O Pai suspende sua própria execução e passa a aguardar o término de ambos os filhos através de chamadas de sistema wait.

O sentido do fluxo de mensagens pelo canal lógico criado é estabelecido em tempo de execução pelos filhos. Como as comunicações são síncronas, o primeiro dos correspondentes espera pelo segundo. Nessa descrição, supõe-se que o Produtor execute send pela primeira vez antes do Consumidor executar receive pela primeira vez. Essa chamada a send gera uma interação entre o interpretador local e o interpretador central. O interpretador central coloca na entrada da tabela de canais lógicos correspondente ao canal designado a anotação de que o processo Produtor está apto a enviar sua primeira mensagem através dele. A execução do Produtor é suspensa.

A primeira execução de receive pelo Consumidor gera uma interação entre o interpretador local e o interpretador central. Verifica-se nesse momento a formação de um par send-receive. O interpretador central escolhe então um canal físico de saída do nó que executa o Produtor e um canal físico de entrada do nó que executa o Consumidor e estabelece uma conexão entre esses nós utilizando seus operadores de configuração da rede de trabalho. Em seguida, devolve a identificação dos canais físicos locais aos processos envolvidos e a transmissão da primeira mensagem pode se completar diretamente entre eles pela rede de trabalho. A partir daí, as comunicações entre esses processos através desse canal se dão pela rede de trabalho sem a intervenção do interpretador central e portanto sem necessidade de acesso nem à rede de controle nem ao nó controlador.

Cada um dos processos Produtor e Consumidor termina executando a chamada de sistema exit. Quando ambos terminam, o processo Pai retoma sua execução.

5.2.3 Considerações sobre o Interpretador Paralelo

A implementação do interpretador paralelo no multicomputador Crux, suportado pelo sistema operacional ACrux, é surpreendentemente simples. Em outros multicomputadores, a criação dinâmica de processos e canais seria um fator adicional às dificuldades normais de implementação de qualquer linguagem paralela em multicomputadores [AND89].

A flexibilidade da conexão dinâmica de canais físicos pôde ser demonstrada pela reconfiguração da rede de interconexão do multicomputador em tempo de execução para corresponder à rede de processos comunicantes a medida que ela vai sendo criada em tempo de execução. Assim, a cada instante na vida de uma rede de processos comunicantes de topologia variável, o mapeamento perfeito de toda rede é sempre garantido para redes cujos graus dos processos seja inferior ou igual ao grau dos nós trabalhadores.

5.3 Implementações

A viabilidade do ambiente proposto pôde ser avaliada em relação ao desempenho através da construção dos simuladores apresentados no capítulo 4 e em relação à simplicidade e à flexibilidade através das aplicações introduzidas em 5.1 e 5.2. Entretanto, um trabalho que pretende se materializar em uma realização física deve ainda confirmar suas qualidades em trabalhos de implementação, sobretudo para verificar se todos os aspectos práticos de projeto foram efetivamente considerados.

Este subcapítulo descreve os experimentos de implementação realizados em ambientes que, na falta do multicomputador Crux, conservam suas qualidades no que se refere às exigências práticas essenciais.

O primeiro desses experimentos, descrito com algum detalhe, consta de uma versão preliminar do sistema operacional ACrux desenvolvida no multicomputador Supernode. Os demais experimentos, descritos brevemente, constam da construção de um simulador do multicomputador Nó // (uma versão reduzida do multicomputador Crux) de outra versão do sistema ACrux e de uma versão do interpretador paralelo da linguagem Superpascal desenvolvidas nesse simulador.

5.3.1 Sistema Operacional ACrux no Supernode

5.3.1.1 Componentes da Versão Preliminar do Sistema

Uma versão preliminar do sistema ACrux foi desenvolvida para multicomputador Supernode descrito em 2.2.3.2 [COR89]. O ambiente de *hardware* usado nesse experimento consta dos seguintes componentes:

- Um multicomputador Supernode com 16 nós de trabalho.
- Uma estação de trabalho Unix.
- Uma placa de interface entre o Supernode e a estação de trabalho.

A figura 5.6 mostra a distribuição dos componentes de *software* sobre os componentes desse ambiente de *hardware*, como se descreve a seguir. Para evitar referências diretas aos componentes do Supernode na descrição subsequente, eles são referidos pela designação dos componentes do Crux que lhes são correspondentes. Assim, nó de controle, nó de trabalho, comutador central e barramento de controle são referidos por nó controlador, nó trabalhador, rede de trabalho e rede de controle, respectivamente.

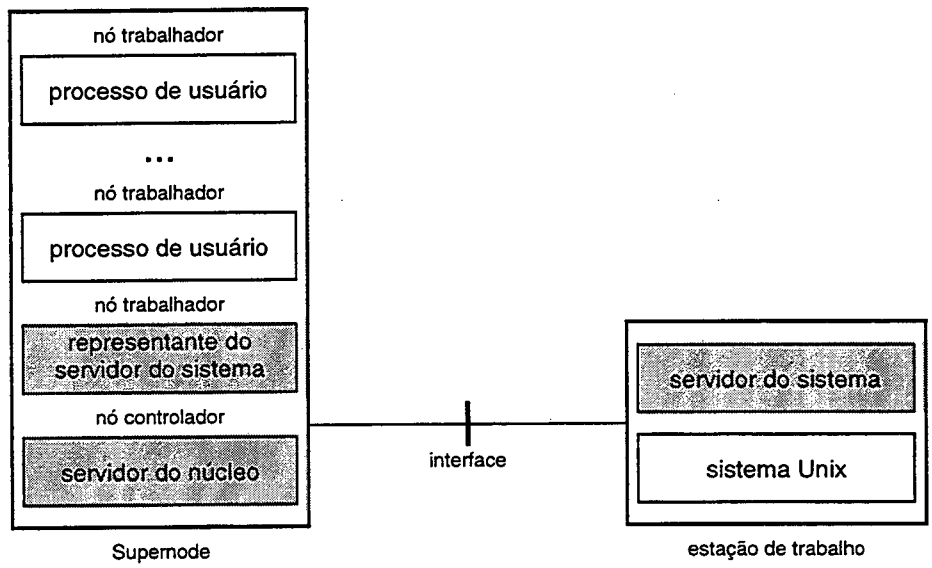


Figura 5.6: Versão preliminar do sistema ACrux no Supernode.

O objetivo principal desse experimento foi o de verificar a habilidade da rede de controle e da rede de trabalho do multicomputador Crux para suportar o sistema operacional conforme a estrutura da figura 5.1. Em função das restrições impostas pelo ambiente de *hardware* algumas alterações e simplificações foram introduzidas no modelo geral do sistema ACrux.

A decisão inicial de adotar um único servidor do sistema permitiu evitar a decomposição funcional dos serviços do sistema cujo custo de projeto e implementação só se justificaria em uma versão dirigida a um protótipo real do multicomputador Crux.

O exemplar do Supernode usado neste experimento não possui periféricos. A alteração que decorre imediatamente dessa situação refere-se à migração dos serviços do sistema de arquivos do Supernode para a estação de trabalho. Dessa forma, o servidor do sistema foi atribuído a um processo executando na estação de trabalho e apenas um seu representante — responsável pela intermediação das comunicações entre o servidor do sistema e os demais processos — foi atribuído a um nó trabalhador arbitrariamente escolhido do Supernode.

Todos os operadores identificados em 5.1.2 foram implementados como funções de biblioteca. Essas funções são ligadas ao código do programa antes de sua carga. Assim, cada processo possui apenas os códigos dos operadores usados por ele e cada nó do Supernode contém como única entidade a ser gerida o processo que nele executa. O servidor do núcleo é executado no nó controlador e os demais processos são executados nos nós trabalhadores.

5.3.1.2 Implementação das Interfaces

Os aspectos mais relevantes da implementação dos componentes da versão preliminar do sistema ACrux no multicomputador Supernode são apresentados a seguir.

Interface da Rede de Controle

Os operadores kc_SendReceive, ks_ReceiveAny e ks_Send da interface da rede de controle são implementados por procedimentos simples, apoiados diretamente nos operadores e nos registradores da interface de *hardware* da rede de controle. (Como foi visto em 2.2.3.2, essa interface é mapeada na memória e manipulada através de operações de leitura e escrita em endereços de memória específicos.)

Interface do Núcleo

Os operadores Connect, ConnectAny, Disconnect, AllocateAny e Deallocate da interface do núcleo são implementados como chamadas de procedimentos remotos na rede de controle.

A construção manual das funções de biblioteca correspondentes a esses operadores foi muito simples em função de seu reduzido número. Para minimizar as deficiências de transmissão de dados através um dispositivo não projetado com esse objetivo, todas as mensagens transmitidas pela rede de controle são compactadas e codificadas em um único *byte*.

O servidor do núcleo é o processo encarregado da execução efetiva dos operadores da interface do núcleo. As políticas de gerência dos canais físicos e dos nós trabalhadores foram comentadas em 5.1.2.2. A seguir são tratados os mecanismos especificamente associados à versão preliminar.

As comunicações sempre diretas exigem o estabelecimento das conexões entre canais físicos dos nós trabalhadores através da rede de trabalho. Uma lista de requisições de conexões pendentes associadas ao único servidor do sistema é suficiente para gerir os canais físicos. Uma requisição é inserida nessa lista quando um processo de usuário executa o operador Connect. A primeira requisição dessa lista é atendida quando o servidor do sistema executa o operador ConnectAny. O rompimento por demanda de uma conexão é realizada em resposta ao operador Disconnect executado por qualquer um dos processos envolvidos na conexão.

A atribuição de um único processo a cada nó trabalhador, associada às comunicações sempre diretas entre nós trabalhadores, permite que a escolha de um nó para cada novo processo criado seja arbitrária. Uma lista de nós disponíveis é suficiente para gerir os nós trabalhadores. O operador AllocateAny provoca a retirada de um nó arbitrário dessa lista e o operador Deallocate provoca a inserção do nó trabalhador envolvido nessa lista.

Uma vez que o processo de usuário que libera um nó é a única entidade nele presente, o operador Deallocate promove também a reinicialização (o *reset*) do nó liberado. A reinicialização é efetuada através do procedimento auxiliar do servidor do núcleo

InitNode (node)

onde node identifica o nó trabalhador envolvido. Esse procedimento se apoia diretamente sobre os operadores e registradores da interface de *hardware* da rede de controle.

Interface da Rede de Trabalho

A duração das conexões associadas à disciplina de comunicação cliente-servidor deve se restringir no máximo à seqüência das duas mensagens que caracterizam essa disciplina. Como o servidor do sistema deve estar apto a postergar as respostas, a duração das conexões para a transmissão de uma única mensagem é a mais adequada. Assim, o servidor do sistema fica livre para atender novas requisições deixando pendentes as que não podem ser atendidas imediatamente.

Os operadores out e in implementados diretamente no *hardware* do Transputer (ver 2.2.3.2) exigem dos processos correspondentes o conhecimento prévio do tamanho da mensagem transmitida. Entretanto, o tamanho das mensagens que transitam no sistema ACrux é variável e deve ser estabelecido dinamicamente pelos processos envolvidos. Por isso, é transmitido um par de mensagens: A mensagem preliminar length de tamanho fixo L contém o tamanho da mensagem efetiva msg, conforme a figura 5.7.

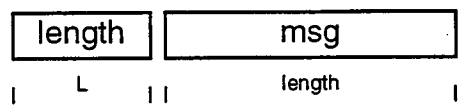


Figura 5.7: Par de mensagens transmitido pela rede de trabalho.

Os operadores s_Send, s_Receive e ss_ReceiveAny da interface da rede de trabalho são implementados simplesmente como segue:

```
s_Send (proc, msg, length) =
        Connect (node, link);
        out (link, L, length);
        out (link, length, msg);
        Disconnect (node)

s_Receive (proc, msg, length) =
        Connect (node, link);
        in (link, L, length);
        in (link, length, msg);
        Disconnect (node)

ss_ReceiveAny (proc, msg, length) =
        ConnectAny (node, link);
        in (link, L, length);
        in (link, length, msg);
        Disconnect (node)
```

Nesses operadores, além dos símbolos L, length e msg já introduzidos, proc representa a identificação de um processo, node a identificação do nó trabalhador onde o processo proc executa e link a identificação de um canal físico local do nó node. (A

identificação do nó é um dos componentes da identificação do processo, como foi comentado em 5.1.2.1.)

Interface do Sistema

A construção manual de várias (em torno de cinqüenta) funções de biblioteca, embora bastante repetitiva, não justifica um procedimento automático que exigiria a existência de um compilador específico.

O exemplo parcial de execução de um operador regular visto em 5.1.3 enfatizou o emprego dos operadores das interfaces da rede de trabalho e da rede de controle. A descrição dos operadores regulares apresentada a seguir em 5.3.1.3, enfatizando as interações entre os processos envolvidos, serve para complementar aquele exemplo. A descrição dos operadores especiais é apresentada adiante em 5.3.1.4.

5.3.1.3 Operadores Regulares

O comportamento de um processo de usuário durante a execução de um operador regular é representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado nessa conexão.

b O processo de usuário envia ao servidor do sistema mensagem contendo a identificação do operador regular considerado com seus parâmetros e recebe a resposta pertinente.

c O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

As interações a, b e c seguem o padrão cliente-servidor estrito. Entretanto, enquanto as interações a e c se desenrolam na rede de controle, a interação b se desenrola na rede de trabalho.

5.3.1.4 Operadores Especiais

Os operadores read, write, fork, exit e exec são especiais. As interações nas quais eles se envolvem são descritas a seguir.

Entradas e Saídas

O comportamento de um processo de usuário durante a execução do operador read é representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado nessa conexão.

b O processo de usuário envia ao servidor do sistema mensagem contendo a identificação do operador irregular read com seus parâmetros e recebe os dados lidos como uma mensagem especial diretamente no endereço fornecido como parâmetro e o resultado da operação em outra mensagem.

c O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

O comportamento de um processo de usuário durante a execução do operador write é representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado nessa conexão.

b O processo de usuário envia ao servidor do sistema mensagem contendo a identificação do operador irregular write com seus parâmetros, envia os dados lidos como uma mensagem especial diretamente a partir do endereço fornecido como parâmetro e recebe o resultado da operação em outra mensagem.

c O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

Criação Dinâmica de Processos

O comportamento de um processo de usuário durante a execução do operador fork pode ser representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor do núcleo o pedido de atribuição de um nó trabalhador livre qualquer para conter o novo processo e recebe a identificação do novo processo como resposta.

b O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó que vem de ser atribuído ao novo processo e recebe como resposta a identificação do canal físico local usado nessa conexão.

c O processo de usuário envia ao nó atribuído ao novo processo uma cópia de todo seu espaço de endereçamento.

d O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó atribuído ao novo processo e recebe uma confirmação como resposta.

e O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado nessa conexão.

f O processo de usuário envia ao servidor do sistema mensagem contendo a identificação do operador irregular fork e recebe o resultado da operação em outra mensagem.

g O processo de usuário envia ao servidor do núcleo o pedido de rompimento de conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

O comportamento de um processo de usuário durante a execução do operador **exit** é representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor de núcleo o pedido de liberação de seu nó e recebe uma confirmação como resposta.

b O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado na conexão.

b O processo de usuário envia ao servidor do sistema mensagem contendo a identificação do operador irregular **exit** com seu parâmetro e recebe a resposta pertinente.

c O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

Execução de Programas

O comportamento de um processo de usuário durante a execução do operador **exec** é representado pela seguinte seqüência de interações:

a O processo de usuário envia ao servidor do núcleo o pedido de conexão do seu nó com o nó onde o servidor do sistema executa e recebe como resposta a identificação do canal físico local usado nessa conexão.

b O processo de usuário envia ao servidor do sistema a mensagem contendo a identificação do operador irregular **exec** com seus parâmetros e recebe como uma mensagem especial o código do novo programa que se sobrepõe ao precedente.

c O processo de usuário envia ao servidor do núcleo o pedido de rompimento da conexão com o nó onde o servidor do sistema executa e recebe uma confirmação como resposta.

5.3.1.5 Sinais

Os sinais transitam na versão preliminar do sistema operacional ACrux com as mensagens trocada entre os processos de usuários e o servidor do sistema. Para isso, é reservado um espaço no cabeçalho das mensagens.

Independentemente de sua origem, um sinal é memorizado pelo servidor do sistema no registro descritor do processo sinalizado. Se uma chamada de sistema está na origem do sinal, ele é acrescido à mensagem enviada pelo processo de usuário ao servidor do sistema. A cada mensagem enviada pelo servidor do sistema a um processo de usuário são acrescentados os sinais que lhe são eventualmente destinados. As chamadas de sistema especialmente envolvidas com os sinais são **kill**, **wait**, **signal** e **exit**.

Os operadores kill e wait são regulares. O operador kill acrescenta o sinal ao cabeçalho da mensagem enviada pelo processo de usuário ao servidor do sistema. O servidor do sistema posterga a mensagem de resposta relativa ao operador wait enquanto não houver um sinal destinado ao processo de usuário envolvido. O operador signal é local. O operador exit (descrito em 5.3.1.4) é ainda responsável pelo acréscimo do sinal destinado ao processo pai no cabeçalho da mensagem por ele enviada o servidor do sistema.

5.3.1.6 Inicialização do Sistema

O procedimento de inicialização da versão preliminar do sistema consiste essencialmente da instalação do servidor do núcleo, do servidor do sistema e do processo inicial. O processo inicial é o único processo de usuário cuja criação não resulta da execução do operador fork. Esse processo é substituído pelo programa de login que, após a identificação de um usuário, é por sua vez substituído por um interpretador de comandos (shell). A partir de então, o sistema se torna operacional.

5.3.1.7 Desenvolvimento do Experimento

O desenvolvimento do sistema se desenrolou de maneira incremental. O objetivo foi o de produzir a cada etapa uma base segura e ferramentas para a construção da etapa seguinte. Os elementos principais das etapas distintivas da evolução do sistema foram as seguintes:

a Carga de um programa executável (escrito em linguagem de montagem) e operadores de comunicação entre a estação de trabalho e o nó controlador.

b Ambiente de execução de um programa executável (escrito em linguagem C) sobre um nó trabalhador único e uma versão especial de printf [KER88].

c Servidor de arquivos elementar sobre a estação de trabalho para responder às chamadas open, close, read e write.

d Mecanismos gerais para a carga de programas nos diversos nós do Supernode.

e A versão preliminar tal como descrita.

Mesmo que o resultado líquido de algumas dessas etapas pareçam simples, o esforço para realizá-los não foi pequeno. De fato, é difícil exagerar, por exemplo, a importância de dispor da ferramenta construída na etapa b sabendo que não se dispunha anteriormente a isso de absolutamente nenhuma facilidade de depuração.

Com exceção do procedimento de inicialização do Transputer e de algumas seqüências de instruções inseridas em linha em alguns procedimentos muito dependentes da máquina (como a chamada de sistema fork) escritas em linguagem de montagem, todo o sistema foi escrito na linguagem C.

O componente mais volumoso do sistema é a parte do servidor do sistema escrito sobre a estação de trabalho. A sua realização foi facilitada pelo conforto das ferramentas oferecidas pelo ambiente de programação Unix. O maior esforço de desenvolvimento se concentrou nos procedimentos de baixo nível, sobretudo aqueles que dependiam de interações entre os nós do Supernode.

A versão preliminar do sistema ACrux no Supernode constitui um sistema completo ao nível das interfaces identificadas em 5.3.1.2.

5.3.2 Projeto Nó //

O objetivo do Projeto Nó // (lê-se *nó paralelo*) foi o de produzir um ambiente multicomputador para a execução de programas paralelos que pudesse ser desenvolvido com escassos recursos financeiros por um grupo pequeno de pesquisadores e alunos de pós-graduação, preservando as propriedades essenciais do ambiente Crux.

Esse projeto previu a construção de um protótipo do multicomputador Nó // e versões simplificadas do sistema operacional ACrux (visto em 5.1) e do interpretador paralelo para a linguagem Superpascal (visto em 5.2).

Com o objetivo de permitir que os componentes de *software* e de *hardware* pudessem ser produzidos concomitantemente, construiu-se um simulador para o multicomputador Nó //.

Nesta seção, após a introdução da arquitetura do multicomputador Nó //, são descritos os componentes de *software* desse projeto.

5.3.2.1 Multicomputador Nó //

O multicomputador Nó // pode ser considerado uma versão simplificada do multicomputador Crux. Sua arquitetura difere da arquitetura do Crux (descrita em 3.4.1) apenas porque cada um de seus nós trabalhadores está conectado à rede de trabalho por um único canal físico.

Restrições financeiras motivaram a escolha de componentes de baixo custo disponíveis no mercado para a construção desse protótipo. Afinal, o objetivo principal de sua construção seria o de validar os componentes do *software* citados anteriormente neste capítulo através de um experimento de implementação real. O ganho de desempenho advindo apenas da velocidade dos componentes de *hardware* poderia ser obtido futuramente em um projeto futuro de maior envergadura.

Assim, os nós seriam construídos a partir de placas baseadas em microprocessadores Pentium da Intel e a rede de trabalho seria composta por um *crossbar* C004 da Inmos. Dessa forma, o projeto de *hardware* envolveria principalmente a rede de controle e as interfaces para a rede de trabalho [ZEF95] [ZEF96].

5.3.2.2 Simulador do Multicomputador Nó //

O simulador do multicomputador Nó // foi desenvolvido em um computador baseado em um microprocessador Pentium.

Como as instruções da máquina simulada e do microprocessador onde o simulador foi desenvolvido são as mesmas, a simulação se concentrou nos nós controlador e trabalhadores e nas redes de controle e de trabalho.

O simulador foi estruturado como um sistema operacional (referido por *núcleo do simulador*) orientado aos componentes da arquitetura do multicomputador Nó //, conforme a figura 5.8.

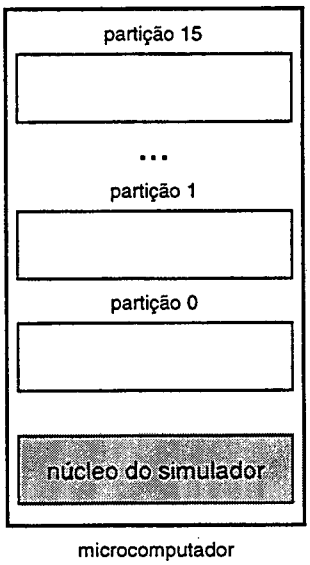


Figura 5.8: Simulador do multicomputador Nó //.

A memória do simulador é dividida em um bloco para o núcleo do simulador e outro para uma coleção de partições de tamanho fixo, cada uma das quais usada para simular a memória local de um nó do multicomputador.

O núcleo do simulador é responsável essencialmente pela gerência das partições e pelo escalonamento das partições. O objetivo do escalonamento é o de distribuir fatias de tempo entre as partições para simular os processadores dos nós do multicomputador.

O núcleo do simulador é ainda responsável pela simulação das redes de controle e de trabalho. O trânsito de mensagens em ambas as redes é simulado através da cópia de dados entre as partições que simulam as memórias dos nós comunicantes.

Para acelerar sua construção, esse simulador foi desenvolvido a partir do sistema operacional Xinu [COM88], adaptando-se os componentes originais desse sistema às especificações dos componentes do multicomputador simulado [CAM95]. [MON95]

5.3.2.3 Sistema Operacional ACrux no simulador do Nó //

O experimento de construção de um sistema operacional para o simulador do multicomputador Nó // reproduziu de certa forma o experimento realizado no Supernode. O objetivo desse experimento foi o de implementar uma versão preliminar do sistema ACrux análoga à descrita em 5.3.1 que pudesse ser posteriormente transportada sem dificuldade para o protótipo do multicomputador Nó // [CAM95] [MON95]. O ambiente de *hardware* nesse

caso foi composto por dois computadores baseados em microprocessadores Pentium conectados através de uma interface serial, conforme a figura 5.9.

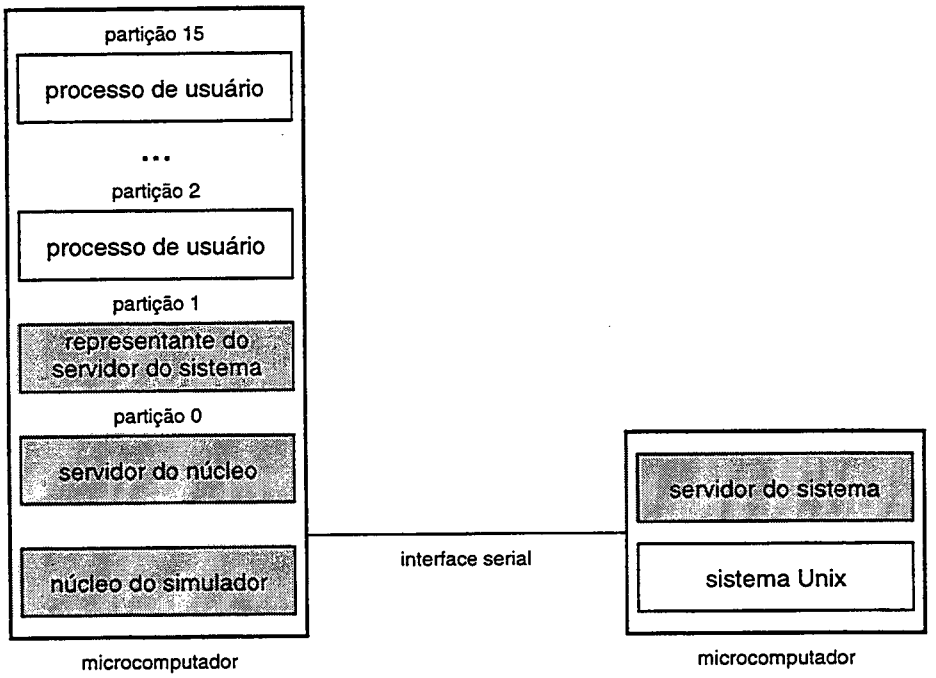


Figura 5.9: Versão preliminar do sistema ACrux no simulador.

A estrutura do sistema operacional do experimento descrito em 5.3.1 foi repetida nesse simulador. Os operadores das diversas interfaces foram entretanto simplificados porque puderam prescindir dos parâmetros relativos à identificação do canal físico local, que no caso do Nó // é único para cada nó trabalhador.

A distribuição dos componentes do sistema operacional pelos componentes do *hardware* também seguiu a mesma diretriz do experimento descrito em 5.3.1. Os componentes das diversas interfaces do sistema operacional que executa no simulador do Nó // foram elaborados prevendo sua transferência posterior para um protótipo real do multicomputador Nó //.

5.3.2.4 Interpretador Paralelo para a Linguagem Superpascal no Nó //

O ambiente composto pelo simulador e pelo sistema operacional serviram de base para a implementação de um interpretador paralelo para a linguagem Superpascal, baseado no modelo geral apresentado em 5.2.

Nesse experimento, em função as restrições do multicomputador Nó //, a rede física correspondente à rede lógica do programa paralelo não pode ser estabelecida. Assim, a cada mensagem transmitida pela rede de trabalho procede-se à conexão entre os nós comunicantes pela duração da transmissão da mensagem.

Como os mecanismos de comunicação adequados à linguagem Superpascal diferem dos usados pelo sistema operacional ACrux, definiu-se um conjunto de operadores de comunicação específico para a linguagem (e independente dos operadores usados pelo sistema operacional) [MER96]. Entretanto, os próprios mecanismos de criação dinâmica de processos do sistema operacional ACrux foram usados pelo interpretador paralelo.

5.4 Conclusões

Considerados isoladamente, os projetos do sistema operacional ACrux e do interpretador paralelo para a linguagem Superpascal evidenciaram a simplicidade prevista no capítulo 3. Além disso, a facilidade de implementação de mecanismos de comunicação distintos para cada um desses itens comprova flexibilidade também prevista no capítulo 3. Se à flexibilidade e à simplicidade verificadas neste capítulo se acresce o bom desempenho medido no capítulo 4, completa-se o quadro que permite aceitar o ambiente proposto como efetivo em relação aos objetivos perseguidos.

6 Conclusões

Neste capítulo, após a revisão dos objetivos, são sumariadas as tarefas efetuadas, destacadas as contribuições mais relevantes e avançadas as perspectivas para projetos futuros do trabalho apresentado neste texto.

6.1 Revisão dos Objetivos

O tema de fundo deste trabalho foi o da exploração do paralelismo real disponível nos multicomputadores, visando responder a critérios tanto de maximização de desempenho de aplicações específicas quanto de disponibilidade de serviços do sistema operacional.

Em função do grande número de nós e canais físicos dos multicomputadores, o paralelismo real deveria ser explorado de várias formas, a partir de um núcleo de sistema operacional com componentes presentes em todos os nós, passando por um sistema operacional composto por servidores fisicamente distribuídos pelos nós, até chegar a uma aplicação específica construída como uma rede de processos comunicantes.

Partindo da visão dos multicomputadores como ambientes naturais para a expressão física das redes lógicas de processos comunicantes, este trabalho procurou seguir uma trilha alternativa às das soluções mais difundidas que, abandonando a idéia da correspondência perfeita, deram origem a arquiteturas com redes estáticas ou dinâmicas associadas a mecanismos de comunicação inerentemente complexos. De fato, este trabalho buscou um alto grau de aderência mútua entre os multicomputadores e as redes de processos comunicantes, favorecendo a flexibilidade e a simplicidade sem sacrificar o desempenho do serviço de comunicação.

De uma maneira geral, o ambiente proposto deveria permitir a execução de redes de processos comunicantes resultantes da decomposição funcional de um problema em um conjunto de processos respeitando os níveis de granulosidade computacional média adequados aos multicomputadores. Além disso, estimou-se que a solução conveniente dos problemas referentes às abstrações de processos e de trocas de mensagens facilitaria a implementação de quaisquer abstrações propostas pelos mais diversos modelos de programação paralela.

De uma forma mais específica, o objetivo fundamental deste trabalho foi o de conceber e avaliar um ambiente multicomputador para execução de redes de processos comunicantes, sujeito aos seguintes critérios:

- O ambiente proposto deveria incentivar o mapeamento perfeito das redes de processos comunicantes em muitos casos de interesse prático. Em relação aos processos, apenas o mapeamento perfeito seria aceito. Em relação aos canais lógicos, apenas as comunicações diretas seriam aceitas, mesmo se o mapeamento perfeito não fosse possível em todos os casos.
- Os serviços do sistema operacional deveriam assumir a interface de programação do sistema Unix acrescida apenas pelos operadores de um mecanismo específico de comunicação. O sistema operacional deveria ser estruturado conforme o modelo cliente-servidor, onde um núcleo fornecesse as funções essenciais para permitir que a maior parte dos serviços do sistema fossem prestados por servidores executando como processos regulares.
- O ambiente proposto deveria simplificar a implementação de linguagens paralelas cujos programas gerassem redes de processos comunicantes trocando mensagens de tamanho variável através de canais lógicos síncronos dedicados.

6.2 Resumo do Trabalho Realizado

Em cumprimento aos objetivos introduzidos em 1.2, precisados em 3.3 e revistos em 6.1, as seguintes tarefas foram efetuadas:

- Concepção do multicomputador reconfigurável Crux e do seu mecanismo de comunicação com conexão dinâmica de canais físicos. Se o resultado apresentado em 3.4 é surpreendentemente simples e pôde ser expresso de forma tão compacta, o esforço envolvido na sua concepção não guarda correspondência com essas dimensões. Efetivamente, foi grande o número de propostas elaboradas e descartadas antes da obtenção do ambiente Crux que se mostrou tão efetivo.
- Avaliação comparativa do ambiente Crux com outros ambientes conhecidos, em relação à flexibilidade, à simplicidade e ao desempenho. A avaliação da flexibilidade e da simplicidade envolveu vários experimentos práticos de implementação de seus componentes. A avaliação do desempenho envolveu a construção de simuladores para os multicomputadores confrontados e a realização de inúmeros experimentos de simulação
- Concepção do sistema operacional ACrux. Esse sistema foi descrito tanto através da distribuição física de seus processos pelos nós do multicomputador quanto através de camadas de *software* hierarquicamente superpostas.
- Concepção de um ambiente para a execução de programas da linguagem paralela Superpascal, através da definição de um mecanismo específico de comunicação com conexão dinâmica de canais físicos e do emprego dos serviços do sistema ACrux principalmente para a criação dinâmica de processos.

- Implementação da primeira versão do sistema operacional ACrux em um multicomputador Supernode que, apesar do baixo desempenho (resultante do uso do barramento de controle dessa máquina para uma função de comunicação para a qual ela não tinha sido projetada), demonstrou a viabilidade da proposta, sobretudo em termos da facilidade de desenvolvimento de *software*.

Além dos elementos relacionados acima, outros resultados a ele associados, envolvendo pessoas engajadas em suas dissertações de mestrado, merecem ainda ser citados aqui:

- Construção de um protótipo simulado de uma versão simplificada do multicomputador Crux que, apesar de insuficiente para a avaliação efetiva de programas paralelos de interesse prático, ofereceu todos o suporte necessário para o desenvolvimento de *software*, enquanto se aguardam recursos materiais para a construção de um protótipo real.
- Implementação de uma versão preliminar do sistema ACrux com características semelhantes às da versão realizada no Supernode, usando o protótipo simulado para seu desenvolvimento mas visando seu transporte futuro para o protótipo real.
- Implementação de um interpretador paralelo para a linguagem Superpascal como uma extensão do interpretador seqüencial original com as seguintes características: cada processo Superpascal é representado por um processo Unix e eles trocam mensagens através de um mecanismo de comunicação específico com conexão dinâmica de canais físicos.

6.3 Contribuições

O resultado fundamental deste trabalho materializou-se na concepção e avaliação de um ambiente para execução de programas paralelos expressos como redes de processos comunicantes suportado pela integração do multicomputador Crux e do mecanismo de comunicação com conexão dinâmica de canais físicos.

Dessa integração resultou um novo paradigma para comunicação em multicomputadores que provê em todos os casos transferências monolíticas de mensagens completas de qualquer tamanho através de canais físicos dedicados entre quaisquer pares de nós e permite em muitos casos a construção em tempo de execução da rede física que corresponde perfeitamente à topologia da rede lógica de processos comunicantes.

Os mecanismos de comunicação citados na literatura envolvem o roteamento de mensagens e/ou o tratamento de pacotes. O mecanismo de comunicação com conexão dinâmica de canais físicos é o único que elimina simultaneamente a necessidade do roteamento de mensagens e do tratamento de pacotes.

Os sistemas operacionais convencionais para multicomputadores são fisicamente distribuídos conforme o modelo cliente-servidor mas suportados por um núcleo replicado em todos os seus nós. O sistema ACrux é o único apoiado em um núcleo também fisicamente distribuído conforme o modelo cliente-servidor. Também é único nesse sistema o emprego de suportes físicos de comunicação distintos em cada uma de suas duas camadas. Dessa forma, o

sistema operacional ACrux forma com o multicomputador Crux um conjunto completamente original cuja simplicidade de implementação foi constatada tanto no Supernode quanto em um o protótipo simulado e cuja eficiência foi comprovada usando técnicas de simulação discreta.

A expressão mais eloqüente da simplicidade reivindicada nesta proposta pode ser evidenciada pelo reduzido número e pela elementaridade dos operadores necessários ao núcleo do sistema operacional ACrux. O servidor do núcleo implementa uma interface extremamente reduzida, composta por apenas cinco operadores — três para gerência de canais físicos (referentes à conexão e à desconexão de canais físicos entre nós trabalhadores) e dois para gerência de nós (referentes à atribuição e a liberação de nós trabalhadores) — que foram suficientes para implementar um sistema Unix completo.

Os operadores das interfaces de comunicação também são implementados por algoritmos elementares. Os operadores da interface de trabalho são executados localmente quando a mensagem deve transitar entre nós trabalhadores já conectados; eles requerem o auxílio do componente central do núcleo quando é necessário requisitar a prévia conexão dos nós trabalhadores envolvidos na comunicação. Os operadores da interface da rede de controle são sempre executados localmente nos nós envolvidos na comunicação por algoritmos que são simples extensões da interface de *hardware* dessa rede.

Em resumo, os elementos acima citados que introduzem aspectos únicos a este trabalho foram responsáveis pelos resultados aferidos que revelaram as qualidades superiores do ambiente Crux entre os ambientes multicomputadores de porte médio em relação à flexibilidade, à simplicidade e ao desempenho.

Enfim, o fato do ambiente proposto se distanciar tanto dos demais referidos na literatura deve merecer pelo menos o crédito de ousadia pela tentativa de abrir uma nova trilha de pesquisa em oposição ao conforto de seguir uma trilha já parcialmente explorada.

6.4 Perspectivas Futuras

A simplicidade do paradigma de comunicação em multicomputadores introduzido neste trabalho só foi atingida porque foi ativa e laboriosamente perseguida. Apesar disso, a construção de um ambiente completo do porte do ambiente aqui proposto depende de recursos humanos e materiais que ultrapassam largamente as ambições deste trabalho. Por isso, deposita-se em projetos futuros a esperança de consolidação e ampliação dos resultados até aqui obtidos.

Em particular, a construção de um protótipo do multicomputador Crux usando processadores poderosos e canais físicos velozes para o qual pudesse ser transportado o *software* já desenvolvido se apresenta naturalmente como a etapa seguinte.

Mais adiante, outros projetos poderiam envolver tanto componentes de *hardware* quanto componentes de *software*. Em *hardware*, por exemplo, seria possível explorar a possibilidade de conectar os canais físicos dos nós trabalhadores através de *crossbars* distintos, o que permitiria ampliar o número de nós do multicomputador sem aumentar a dimensão dos *crossbars*. Em *software*, por exemplo, seria possível definir o projeto de um

sistema de arquivos resultante de sua decomposição funcional em componentes que pudessem ser fisicamente distribuídos pelos nós do multicomputador.

Enfim, a facilidade de experimentação no ambiente Crux permite que ele seja visto em certa medida como um verdadeiro laboratório de computação paralela porque ele pode ser usado em um extremo como uma rede estática (bastando para isso configurá-lo previamente como uma grelha ou um hipercubo, por exemplo) e em outro extremo como uma rede completamente dinâmica (a exemplo do Nó //), além de todas as possibilidades intermediárias propiciadas pela conexão dinâmica de canais físicos.

A Modelos para Simulação

Neste apêndice, são apresentados os modelos do Crux (em A.1), do torus (em A.2) e do *crossbar* (em A.3) expressos na linguagem de simulação Siman referidos no capítulo 4. Apenas os modelos com 36 nós usados nos experimentos envolvendo comunicações generalizadas são mostrados . Os demais modelos diferem dos mostrados apenas nos detalhes que caracterizam os diversos experimentos e foram omitidos neste texto. Além disso, para complementar a avaliação comparativa apresentada no capítulo 4, o Crux e o torus são confrontados (em A.4) com um multicomputador possuindo um *crossbar* ao qual cada nó está conectado através de 4 canais.

A.1 Crux

```
# Crux: Modelo com 36 nós; comunicações generalizadas.

1$      STATION,        NoS;
14$     SEIZE,          1:ProcS (NoOrig),1;
13$     DELAY:eTemProcMen;
15$     RELEASE:        ProcS (NoOrig),1;
6$      COUNT:          NumMensEnvs ,1:MARK(TemTrans);
16$     BRANCH,         1:With,eProbBar,8$,Yes:
                        With,eProbDir,22$,Yes;
8$      QUEUE,          FilaBar;
9$      SEIZE,          1:Bar,1;
7$      DELAY:eTemTransBar;
10$     RELEASE:        Bar,1;
11$     COUNT:          NumMensBar,1;
22$     ASSIGN:         NoDest=NoOrig;
24$     WHILE:          NoOrig == NoDest;
23$     ASSIGN:         NoDest=cNos * (eNoDest  - 1) + eNoDest;
25$     ENDWHILE;
26$     ASSIGN:         CanEntLoc=eCanEntLoc:
                        CanEntGlo=4 * (NoDest - 1) + CanEntLoc:
                        CCan=0;
28$     WHILE:          STATE (CanS (CanEntGlo)) == BUSY_RES .and. CCan < 4;
27$     ASSIGN:         CanEntLoc=MOD (CanEntLoc, 4) + 1:
                        CanEntGlo=4 * (Nodest - 1) + CanEntLoc:
                        CCan=CCan + 1;
29$     ENDWHILE;
2$      SEIZE,          1:CanS (CanEntGlo),1;
0$      DELAY:eTemTransMen;
3$      RELEASE:        CanS (CanEntGlo),1;
```

```
21$       COUNT:      NumMenRecs,1;
5$        TALLY:      Tempo em transito,INT(TemTrans),1;
12$       ROUTE:      0.0,NoS (NoOrig);

17$       BEGIN,      Yes,0M681U;

4$        CREATE,     1,0:eIntCria,ecMaxCria;
18$       COUNT:      xNumMenCria,1;
19$       ASSIGN:      vNoOrig=MOD (vNoOrig, ecNNos) + 1:
                      NoOrig=vNoOrig;
20$       ROUTE:      0.0,NoS (NoOrig);

#

PROJECT,              0C681U,TBC,,Yes;

DISCRETE,             200;

ATTRIBUTES:           NoOrig:
                      CanEntLoc:
                      NoDest,:
                      TemTrans:
                      CanEntGlo:
                      CCan;

VARIABLES:            cNMens,1:
                      vNoOrig,0:
                      cTam,8:
                      cNos,6;

QUEUES:               1,FilaBar,FIFO;

RESOURCES:            REPEAT (Can, 144),Capacity(1,),:
                      REPEAT (Proc, 36),Capacity(1,),:
                      Bar,Capacity(1,),;

STATIONS:             1,No1:2,No2:3,No3:4,No4:5,No5:6,No6:
                      7,No7:8,No8:9,No9:10,No10:11,No11:12,No12:
                      13,No13:14,No14:15,No15:16,No16:17,No17:18,No18:
                      19,No19:20,No20:21,No21:22,No22:23,No23:24,No24:
                      25,No25:26,Mo26:27,No27:28,No28:29,No29:30,No30:
                      31,No31:32,No32:33,No33:34,No34:35,No35:36,No36;

COUNTERS:             NumMenRecs,,Yes:
                      xNumMenCria,,Yes:
                      NumMensBar,,Yes:
                      NumMensEnvs,,Yes;

TALLIES:              Tempo em transito,"0CT681u.dat";

DSTATS:               NQ (FilaBar),,"0CF681U";

REPLICATE,            1,0.0,10000,Yes,Yes,0.0;

EXPRESSIONS:          eTemTransBar,2:
                      eNoDest,DISC (0.17, 1, 0.33, 2, 0.50, 3, 0.66, 4, 0.83, 5, 1.0, 6):
                      ecNNos,cNos ** 2:
                      eTemTransMen,eTamMen:
```

```
                    eTamMen,ANINT (EXPO ( 64 * cTam) + 0.5):
                    eIntCria,POIS (1):
                    eCanEntLoc,DISC (0.25, 1, 0.50, 2, 0.75, 3, 1.00, 4):
                    eProbDir,1 - eProbBar:
                    eProbBar,0.89:
                    ecMaxCria,ecNNos * cNMens:
                    eTemProcMen,0;

SETS:               CanS,Can1 .. Can144:
                    ProcS,Proc1 .. Proc36:
                    NoS,No1 .. No36;
```

A.2 Torus

```
# Torus: Modelo com 36 nós; comunicações generalizadas.

0$      STATION,    NoS;
8$      ASSIGN:     NoAtu=NoSeg;
12$     BRANCH,     1:If,NoAtu == NoOrig,14$,Yes:
                    If,NoAtu <> NoOrig,6$,Yes;
14$     ASSIGN:     Dist=eDistU:
                    SentGlo=eSentGlo:
                    TamMen=eTamMen:
                    Trecho=1:
                    NSentLoc (1)=0:
                    NSentLoc (2)=0;
16$     WHILE:      Trecho <= Dist;
19$     BRANCH,     1:If,NSentLoc (1) == eRaio,20$,Yes:
                    If,NSentLoc (2) == eRaio,21$,Yes:
                    Else,22$,Yes;
20$     ASSIGN:     SentLoc=2;
18$     ASSIGN:     NSentLoc (SentLoc)=NSentLoc (SentLoc) + 1:
                    CanLoc (Trecho)=tSent (SentGlo, SentLoc):
                    Trecho=Trecho+ 1;
17$     ENDWHILE;
10$     SEIZE,      1:Procs (NoOrig),1;
9$      DELAY:eTemProcMen;
11$     RELEASE:    Procs (NoOrig),1;
5$      COUNT:      NMenEnvs,1;
27$     ASSIGN:     Trecho=1:MARK(TemTrans);
6$      BRANCH,     2:If,Trecho == 2,36$,Yes:
                    If,Trecho > Dist,33$,Yes:
                    If,Trecho <= Dist,25$,Yes;
36$     DELAY:TamMen;
15$     ASSIGN:     NoSeg=NoOrig;
7$      ROUTE:      0.0,NoS (NoOrig);

33$     DISPOSE;

25$     ASSIGN:     CanGlo=4 * (NoAtu - 1) + CanLoc (Trecho):
                    NoSeg=tNoSeg (NoAtu, CanLoc (Trecho)):
                    Trecho=Trecho + 1;
23$     SEIZE,      1:CanS (CanGlo),1;
24$     DELAY:1;
28$     BRANCH,     2:Always,26$,Yes:
                    Always,29$,Yes;
```

```
26$     ROUTE:      0.0,NoS (NoSeg);

29$     DELAY:TamMen;
1$      RELEASE:    CanS (CanGlo),1;
37$     COUNT:      NMenCans,1;
30$     IF:         Trecho > Dist;
34$     TALLY:      Tempo em transito,INT (TemTrans),1;
35$     COUNT:      NMenRecs,1;
32$     ENDIF;
31$     DISPOSE;

21$     ASSIGN:     SentLoc=1:NEXT(18$);

22$     ASSIGN:     SentLoc=eSentLoc:NEXT(18$);

13$     BEGIN,      Yes,0T681UW;

2$      CREATE,     1,0:eIntCria,eMaxCria;
4$      ASSIGN:     vNoOrig=MOD (vNoOrig, ecNNos) + 1:
                    NoOrig=vNoOrig:
                    NoSeg=NoOrig:
                    vMen=vMen + 1:
                    Men=vMen;
3$      ROUTE:      0.0,NoS (NoOrig);

#

PROJECT,            0T681UW,TBC,,Yes;

DISCRETE,           1000;

ATTRIBUTES:         CanLoc(6),:
                    Men,:
                    NoOrig:
                    Trecho:
                    SentGlo:
                    Dist,:
                    TamMen:
                    TemTrans:
                    NSentLoc(2),:
                    CanGlo:
                    SentLoc:
                    NoAtu:
                    NoSeg;

VARIABLES:          tNoSeg(36,4),
                        2,5,6,3,10,11,12,7,8,17,18,19,
                        20,13,14,15,26,27,28,29,30,21,22,23,
                        24,1,4,9,16,25,36,31,32,33,34,35,
                        4,3,8,9,6,7,14,15,16,11,12,13,
                        22,23,24,25,18,19,20,21,32,33,34,35,
                        36,27,28,29,30,31,26,17,10,5,2,1,
                        26,1,4,27,2,3,8,9,28,5,6,7,
                        14,15,16,29,10,11,12,13,22,23,24,25,
                        30,17,18,19,20,21,32,33,34,35,36,31,
                        36,35,2,1,34,5,6,3,4,33,10,11,
                        12,7,8,9,32,17,18,19,20,13,14,15,
                        16,31,26,27,28,29,30,21,22,23,24,25:
```

```
                    vMen,0:
                    cNMens,1:
                    tSent(4,2),1,2,3,4,2,3,4,1:
                    vNoOrig,0:
                    cTam,8:
                    cNos,6:
                    cDiam,6;

RESOURCES:          REPEAT (Can, 144),Capacity(1,),:
                    REPEAT (Proc, 36),Capacity(1,),;

STATIONS:           1,No1:2,No2:3,No3:4,No4:5,No5:6,No6:
                    7,No7:8,No8:9,No9:10,No10:11,No11:12,No12:
                    13,No13:14,No14:15,No15:16,No16:17,No17:18,No18:
                    19,No19:20,No20:21,No21:22,No22:23,No23:24,No24:
                    25,No25:26,Mo26:27,No27:28,No28:29,No29:30,No30:
                    31,No31:32,No32:33,No33:34,No34:35,No35:36,No36;

COUNTERS:           NMenCans,,Yes:
                    NMenRecs,,Yes:
                    NMenEnvs,,Yes;

TALLIES:            Tempo em transito,"0TT681UW.dat";

REPLICATE,          1,0.0,10000,Yes,Yes,0.0;

EXPRESSIONS:        eSentGlo,DISC (0.25, 1, 0.50, 2, 0.75, 3, 1.00, 4):
                    ecNNos,cNos ** 2:
                    eTamMen,ANINT (EXPO (64 * cTam) + 0.5):
                    eIntCria,POIS (1):
                    eRaio,AINT (cDiam / 2):
                    eSentLoc,DISC (0.50, 1, 1.00, 2):
                    eMaxCria,ecNNos * cNMens:
                    eDistU,DISC (0.11, 1, 0.34, 2, 0.62, 3, 0.85, 4, 0.96, 5, 1.00, 6):
                    eTemProcMen,0;

SETS:               CanS,Can1 .. Can144:
                    ProcS,Proc1 .. Proc36:
                    NoS,No1 .. No36;
```

A.3 Crossbar

 # Crossbar: Modelo com 36 nós; comunicações generalizadas.

```
1$      STATION,        NoS;
9$      SEIZE,          1:ProcS (NoOrig),1;
8$      DELAY:eTemProcMen;
10$     RELEASE:        ProcS (NoOrig),1;
6$      COUNT:          NumMensEnvs ,1:MARK(TemTrans);
16$     ASSIGN:         NoDest=NoOrig;
18$     WHILE: NoOrig == NoDest;
17$     ASSIGN:         NoDest=cNos * (eNoDest  - 1) + eNoDest;
19$     ENDWHILE;
20$     ASSIGN:         CanEntLoc=eCanEntLoc:
                        CanEntGlo=4 * (NoDest - 1) + CanEntLoc:
                        CCan=0;
```

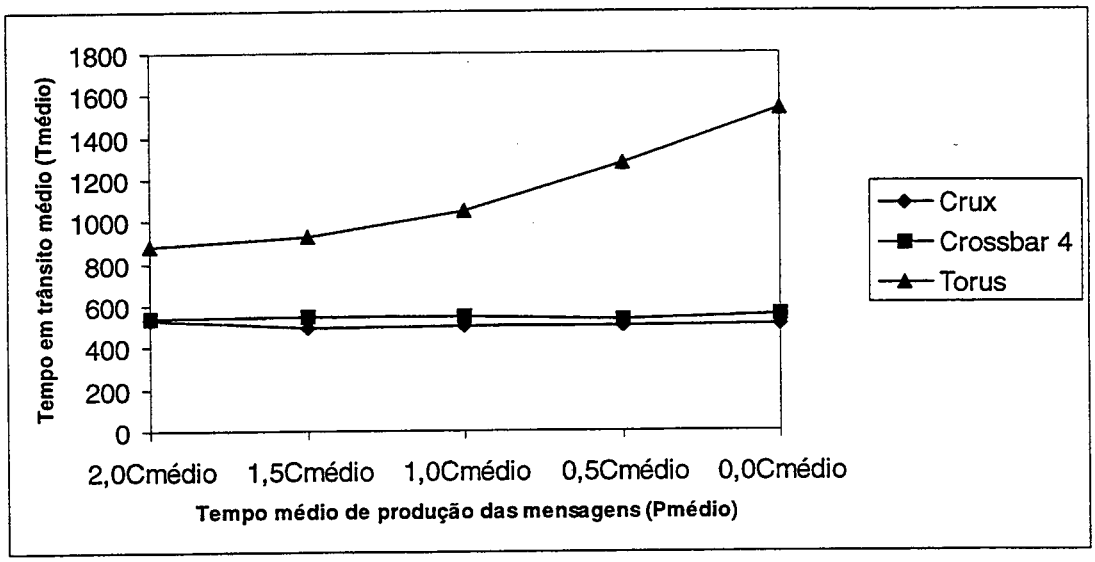
```
22$     WHILE:      STATE (CanS (CanEntGlo)) == BUSY_RES .and. CCan < 4;
21$     ASSIGN:     CanEntLoc=MOD (CanEntLoc, 4) + 1:
                    CanEntGlo=4 * (Nodest - 1) + CanEntLoc:
                    CCan=CCan + 1;
23$     ENDWHILE;
24$     ASSIGN:     NPacs=ANINT ((eTamMen / cTamPac) + 0.5):
                    CNPacs=0;
26$     WHILE:      CNPacs < NPacs;
2$      SEIZE,      1:CanS (CanEntGlo),1;
0$      DELAY:eTemTransPac;
3$      RELEASE:    CanS (CanEntGlo),1;
25$     ASSIGN:     CNPacs=CNPacs + 1;
27$     ENDWHILE;
15$     COUNT:      NumMenRecs,1;
5$      TALLY: Tempo em transito,INT(TemTrans),1;
7$      ROUTE:      0.0,NoS (NoOrig);

11$     BEGIN, Yes,0X681UP;

4$      CREATE,     1,0:eIntCria,eMaxCria;
12$     COUNT:      xNumMenCria,1;
13$     ASSIGN:     vNoOrig=MOD (vNoOrig, ecNNos) + 1:
                    NoOrig=vNoOrig;
14$     ROUTE:      0.0,NoS (NoOrig);

#

PROJECT,            0X681UP,TBC,,Yes;

DISCRETE,           200;

ATTRIBUTES:         NoOrig:
                    NoDest,:
                    CanEntLoc:
                    TemTrans:
                    NPacs:
                    CNPacs:
                    CanEntGlo:
                    CCan;

VARIABLES:          cNMens,1:
                    vNoOrig,0:
                    cTam,8:
                    cNos,6:
                    cTamPac,16;

RESOURCES:          REPEAT (Can, 144),Capacity(1,),:
                    REPEAT (Proc, 36),Capacity(1,),;

STATIONS:           1,No1:2,No2:3,No3:4,No4:5,No5:6,No6:
                    7,No7:8,No8:9,No9:10,No10:11,No11:12,No12:
                    13,No13:14,No14:15,No15:16,No16:17,No17:18,No18:
                    19,No19:20,No20:21,No21:22,No22:23,No23:24,No24:
                    25,No25:26,Mo26:27,No27:28,No28:29,No29:30,No30:
                    31,No31:32,No32:33,No33:34,No34:35,No35:36,No36;

COUNTERS:           NumMenRecs,,Yes:
                    xNumMenCria,,Yes:
```

```
                    NumMensBar,,Yes:
                    NumMensEnvs,,Yes;

TALLIES:            Tempo em transito,"0XT681u.dat";

REPLICATE,          1,0.0,10000,Yes,Yes,0.0;

EXPRESSIONS:        eNoDest,DISC (0.17, 1, 0.33, 2, 0.50, 3, 0.66, 4, 0.83, 5, 1.00, 6):
                    ecNNos,cNos ** 2:
                    eTamMen,ANINT (EXPO (64 * cTam) + 0.5):
                    eCanEntLoc,DISC (0.25, 1, 0.50, 2, 0.75, 3, 1.00, 4):
                    eIntCria,POIS (1):
                    eMaxCria,ecNNos * cNMens:
                    eTemTransPac,cTamPac:
                    eTemProcMen,0;

SETS:               CanS,Can1 .. Can144:
                    ProcS,Proc1 .. Proc36:
                    NoS,No1 .. No36;
```

A.4 Avaliação Comparativa Complementar

O objetivo das simulações descritas a seguir foi o de avaliar o desempenho do Crux em relação ao torus e a um multicomputador possuindo um *crossbar* ao qual cada nó está conectado através de 4 canais. Neste estudo, os mesmos modelos do Crux e do torus e as mesmas hipóteses gerais descritas em 4.2.1 quando submetidos a um padrão de comunicação generalizado descrito em 4.2.2.1.

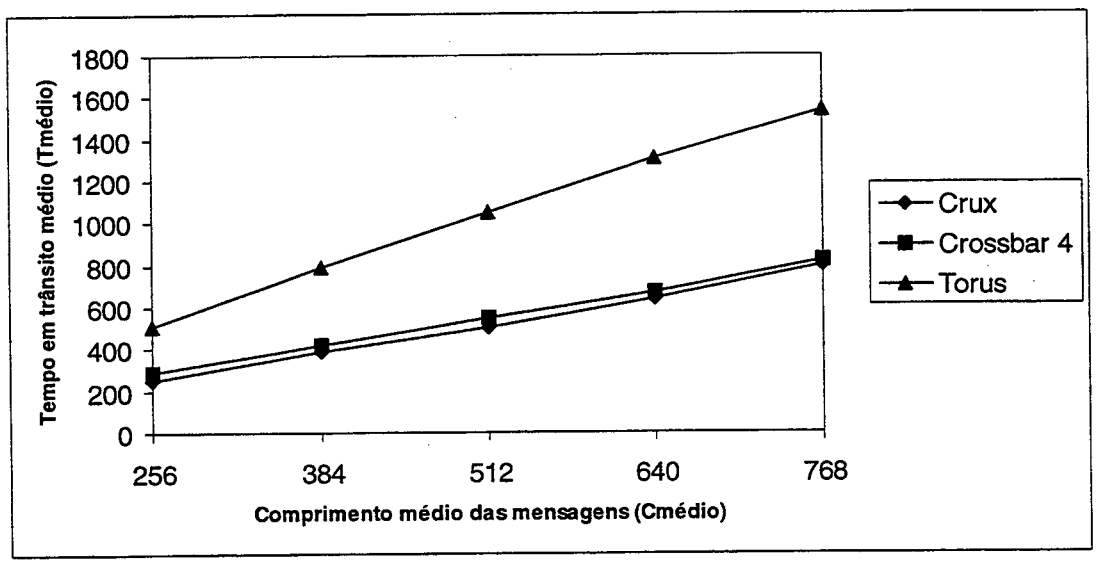
A figura A.1 apresenta o tempo em trânsito médio em função do tempo médio de produção de mensagens para simulações realizadas com modelos de 36 nós, comprimento médio das mensagens igual a 512 *bytes* e o tempos médios de produção de mensagens iguais a $2{,}0C_{médio}$, $1{,}5C_{médio}$, $1{,}0C_{médio}$, $0{,}5C_{médio}$, $0{,}0C_{médio}$ unidades de tempo.



Número de nós N = 36
Comprimento médio das mensagens $C_{médio}$ = 512

Figura A.1: Comunicações generalizadas — $T_{médio}$ em função de $P_{médio}$.

A figura A.2 apresenta o tempo em trânsito médio em função do comprimento médio das mensagens para simulações realizadas com modelos de 36 nós, comprimentos médios das mensagens iguais a 256, 384, 512, 640 e 768 *bytes* e tempo de produção de mensagens igual a 1.0C unidades de tempo (P = C).



Número de nós N = 36
Tempo de produção de mensagens T = 1,0C

Figura A.2: Comunicações generalizadas — $T_{médio}$ em função de $C_{médio}$.

As figuras A.1 e A.2 mostram que o Crux e o *Crossbar* 4 apresentam curvas de desempenho muito próximas, ambos tendo desempenhos superiores ao do torus.

A grande vantagem do Crux em relação ao *Crossbar* 4, entretanto, está na simplicidade do mecanismo de conexão dinâmica de canais físicos conforme foi estabelecido em 3.4.3. Além disso, o nó controlador do Crux é mais um grande fator positivo desse multicomputador no sentido de facilitar o controle centralizado dos recurso desse multicomputador.

Bibliografia

ADV94 Adve, V. S., Vernon, M. K., *Performance Analysis of Mesh Interconnection Networks with Deterministic Routing*, IEEE Transactions on Parallel and Distributed Systems, v. 5, n. 3, p. 226-246, March 1994.

AGE95 Agerwala, J. L., et al., *The SP2 System Architecture*, IBM Systems Journal, v. 34, n. 2, 1995.

AKL89 Akl, S. G., *The Design and Analysis of Parallel Algorithms*, Prentice Hall, 1989.

AND89 Andersen, B., *Hypercube Experiments with Joyce*, Sigplan Notices, v. 24, n. 8, p. 13-22, agosto de 1989.

AND91 Andrews, G. R., *Paradigms for Process Interaction in Distributed Programs*, ACM Computing Surveys, v. 23, n. 1, p. 49-90, março de 1991.

AUS91 Austin, P., et al., *The Design of an Operating System for a Scalable Parallel Computing Engine*, Software - Practice and Experience, v. 21, n. 10, p. 989-1013, outubro de 1991.

BAC86 Bach, M. J., *The Design of the Unix Operating System*, Prentice-Hall, 1986.

BAI88 Baillie, C. F., *Comparing Shared and Distributed Memory Computers*, Parallel Computing, North-Holland, v. 8, p. 101-110, 1988.

BAL89 Bal, H. E., Steiner, J. G., Tanenbaum, A. S., *Programming Languages for Distributed Computing Systems*, ACM Computing Surveys, v. 21, n. 3, p. 261-322, setembro de 1989.

BAR61 Barber, B., *Resistance by Scientists to Scientific Discovery*, Science, n.134, p. 596-602, 1961.

BAR68 Barnes, G., et al., *The Illiac IV Computer*, IEEE Transactions on Computers, v. C-17, n. 8, p. 746-757, agosto de 1968.

BEL94 Bell, G., *Scalable, Parallel Computers: Alternatives, Issues, and Challenges*, International Journal of Parallel Programming, v. 22, n. 1, p. 3-46, 1994.

BIR84 Birrel, A. D., Nelson, B. J., *Implementing Remote Procedure Calls*, ACM Transactions on Computing Systems, v. 2, n. 1, p. 39-59, fevereiro de 1984.

BRO83 Broomell, G., Heath, J. R., *Classification Categories and Historical Development of Circuit Switching Topologies*, ACM Computing Surveys, v. 15, n. 2, p. 95-133, junho de 1983.

BUR85 Burns, S. G., *Concurrent Programming in Ada*, Cambridge University Press, 1985.

BUR92 Burkhardt, H., *Technical Sumary of KSR-1*, Kendall Square Research Corporation, 1992.

CAM95 Campos, R. A., *Um Sistema Operacional Fundamentado do Modelo Cliente-Servidor e um Simulador Multiprogramado para Multicomputador*, Dissertação de Mestrado, CPGCC / UFSC, 1995.

CAR89 Carey, G. F., *Parallel Supercomputing: Methods, Algorithms and Applications*, John Willey & Sons, 1989.

CHA93 Chan, M. Y., Chin, F., *A Parallel Algorithm for an Efficient Mapping of Grids in Hypercubes*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 8, p. 933-946, agosto de 1984.

CHU80 Chu, W. W., et al., *Task Allocation in Distributed Data Processing*, IEEE Computer, v. 13, n. 11, p. 57-69, novembro de 1980.

COM88 Comer, D., Fossum, T. V., *Operating System Design: The Xinu Approach*, Prentice-Hall, 1988.

CLO53 Clos, C., *A Study of Nonblocking Switching*, Bell System Technical Journal, n. 32, p. 406-424, 1953.

COR89 Corso, T. B., et al., *Un Système Unix-Like pour une Station de Travvail à Base de Transputers*, Séminaire Franco-Brésilien sur les systèmes informatiques répartis, p. 142-148, setembro de 1989.

COR97 Corso, T. B., Fraga, J. da S., *Um Multicomputador com Rede de Interconexão Dinâmica*, Conferência Latino-Americana de Informática, Valparaíso, Chile, novembro de 1997.

COR98 Corso, T. B., Fraga, J. da S., *A Demand-Driven Configurable Multicomputer: Design and Evaluation*, Conference on Communication Networks and Distributed Systems Modeling and Simulation, San Diego, Estados Unidos, janeiro de 1998.

COU94 Coulouris, G., et al., *Distributed Systems - Concepts and Design*, Addison-Wesley, 1994.

DAL86 Daaly, W. J., Seitz, C. L., *The Torus Routing Chip*, Journal of Distributed Computing, v. 1, n. 3, p. 187-196, 1986.

DAL87 Dally, W. J., Seitz, C. L., *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*, IEEE Transactions on Computers, v. C-36, p. 547-553, maio de 1987.

DIJ68 Dijkstra, E. W., Co-operating Sequential Processes, Programming Languages, Academic Press, 1968.

DIJ75 Dijkstra, E. W., *Guarded Commands, Nondeterminism, and Formal Derivation of Programs*, Communications of the ACM, v. 18, n. 8, p. 453-457, agosto de 1975.

DUA93 Duato, J., *A New Theory of Deadlock-Free Adaptative Routing in Wormhole Networks*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 12, dezembro de 1993.

DUN90 Duncan, R., *A Survey of Parallel Computer Architectures*, IEEE Computer, fevereiro de 1990.

FAU97 Fausto, L. F., Corso, T. B, Freitas F., P. J. de, *Store-and-Forward Versus Wormhole Routing Mechanisns: A Comparative Performance Evaluation in Multicomputer Networks*, Summer Computer Simulation Conference 97 (SPECTS 97), Arlington, Estados Unidos, julho de 1997.

FAU98 Fausto, L. F., *Spi+: Um Interpretador Paralelo para a Linguagem Superpascal*, Dissertação de Mestrado, CPGCC / UFSC, 1997.

FEN81 Feng, T.-Y., *A Survey of Interconnexion Networks*, IEEE Computer, v. 12, n. 14, p. 12-27, dezembro de 1981.

FIN80 Finkel, R. A., Solomon, M. H., *Processor Interconnection Strategies,* IEEE Transactions on Computers, v. C-29, n. 5, p. 360-371, maio de 1980.

FLY72 Flynn, M. J., *Some Computer Organizations and Their Effectiveness*, IEEE Transactions on Computers, v. 21, n. 9, p. 367-376, 1972.

FRE98 Freitas Filho, P. J. de, Corso, T. B., Fraga, J. da S., Fausto, L. F., *The Impact of Parallel Program Mapping on the Performance of the Wormhole Routing Mechanism*, Conference on Communication Networks and Distributed Systems Modeling and Simulation, San Diego, Estados Unidos, janeiro de 1998.

FUJ92 Fujitsu, *VPP500 Vector Parallel Processor*, V International Symposium on Systems Research, Fujitsu Incorporated, 1992.

GOS91 Goscinski, A., *Distributed Operating Systems - The Logical Design*, Addison-Wesley, 1991.

GUS86 Gustafson, H. L., et al., *The Architecture of a Homogeneous Vector Supercomputer*, International Conference of Parallel Processing, p. 649-652, agosto de 1986.

HAN78 Hansen, P. B., *Distributed Processes: A Concurrent Language Concept*, Communications of the ACM, v. 21, n. 11, p. 934-941, novembro de 1978.

HAN87 Hansen, P. B., *Joyce - A Programming Language for Distributed Systems*, Software - Practice and Experience, v. 17, n. 1, p. 29-50, janeiro de 1987.

HAN89 Hansen, P. B., *A Multiprocessor Implementation of Joyce*, Software - Practice and Experience, v. 19, n. 6, p. 579-592, junho de 1989.

HAN94a Hansen, P. B., *The Programming Language Superpascal*, Software - Practice and Experience, v. 24, n. 5, p. 467-483, maio de 1994.

HAN94b Hansen, P. B., *Superpascal - A Publication Language for Parallel Scientific Computing*, Concurrency - Practice and Experience, v. 6, n. 5, p. 461-483, agosto de 1994.

HAN95 Hansen, P. B., *Studies in Computational Science - Parallel Programming Paradigms*, Prentice Hall, 1995.

HEE93 Hee, Y. Y., Chen, C. C.-Y., *A Comprehensive Performance Evaluation of Crossbar Networks*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 5, maio de 1993.

HOA78 Hoare, C. A. R., *Communicating Sequential Processes*, Communications of the ACM, v. 21, n. 8, p. 666-677, agosto de 1978.

HOA85 Hoare, C. A. R., *Communicating Sequential Processes*, Prentice Hall, 1985.

HWA93 Hwang, K., *Advanced Computer Architecture - Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

IBE93 Ibe, O. C., et al., *Performance Evaluation of Client-Server Systems*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 11, novembro de 1993.

IEE88 IEEE, *Posix - Portable Operating System Interface for Computing Environnements*, IEEE Computing Society, 1988.

INT86 Intel, *iPSC System Overview*, Intel Corporation, 1986.

INT91 Intel, *Paragon XP/S Product Overview*, Intel Corporation, 1991.

KAI89 Kain, R. Y., *Computer Architecture - Software and Hardware*, Prentice Hall, 1989.

KER79 Kermai, P., Kleinrock, L., *Virtual Cut-Through: A New Computer Communication Switching Technique*, Computer Networks, v. 3, n. 14, p. 267-286, outubro de 1979.

KER88 Kernoghan, B. W., Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1988.

KIM94 Kim, J., Chita, R. D., *Hypercube Communication Delay with Wormhole Routing*, IEEE Transactions on Computers, v. 43, n. 7, p. 806-814, July 1994.

LAW75 Lawrie, D. H., *Access and Alignement of Data in an Array Processor*, IEEE Transactions on Computers, v. C-24, n. 12, p. 1145-1155, dezembro de 1975.

LEI85 Leiserson, C. E., *Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing*, IEEE Transactions on Computers, v. C-34, n. 10, p. 892-901, outubro de 1985.

LO 97 Lo, V., et al., *Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 7, julho de 1993.

MER96 Merkle, C., *Ambiente para Execução de Programas Paralelos Escritos na Linguagem Superpascal em um Multicomputador com Rede de Interconexão Dinâmica*, Dissertação de Mestrado, CPGCC / UFSC, 1996.

MON95 Montez, C. B., *Um Sistema Operacional com Micronúcleo Distribuído e um Simulador Multiprogramado de Multicomputador*, Dissertação de Mestrado, CPGCC / UFSC, 1995.

MUD87 Mudge, T. N., Hayes, J. P., Winsor, D.C., *Multiple Bus Architectures*, IEEE Computer, v. 20, n. 6, p. 42-48, junho de 1987.

NI 97 Ni, L. M., et al., *Performance Evaluation of Switched-Based Wormhole Networks*, IEEE Transactions on Parallel and Distributed Systems, v. 8, n. 5, maio de 1997.

NIC88 Nicole, D. A., et al., *Switching Networks for Transputers Links*, VIII Occam Users Group, May 1988.

NOR93 Norman, M. G., *Models of Machines and Computations for Mapping in Multicomputers*, ACM Computing Surveys, v. 25, n. 3, p. 263-302, setembro de 1993.

NUS91 Nussbaum, D., Agarwal, A., *Scalability of Parallel Machines*, IEEE Transactions on Computers, v. 34, n. 3, p. 57-61, março de 1991.

PEA77 Pease, M. C., *The Indirect Binary n-Cube Microprocessor Array*, IEEE Transactions on Computers, v. C-26, n. 5, p. 458-473, maio de 1977.

PEG95 Pegden, C. D., et al., *Introduction to Simulation Using Siman*, McGraw-Hill, 1995.

REE87 Reed, D. A., Fujimoto, R. M., *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, 1987.

SAW87 Sawchuk, A. A., et al., *Optical Crossbar Networks*, IEEE Computer, v.20, n. 6, p. 50-60, junho de 1987.

SER93 Serrano, M. J., Parhami, B., *Optimal Architecture and Algorithms for Mesh-Connected Parallel Computers with Separable Row/Column Buses*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 10, p. 1073-1080, outubro de 1993.

SHE87 Shepherd, D., *The Transputer Instruction Set - A Compiler Writer's Guide*, Inmos, 1987.

SIL89 Silberschatz, A., Peterson, J. L., *Operating Systems Concepts*, Addison-Wesley, 1989.

STO86 Stolfo, S. J., Miranker, D. P., *The DADO Production System Machine*, Journal of Parallel and Distributed Computing, v. 3, n. 2, p. 269- 296, junho de 1986.

SUN93 Sunderan, V. S., et al., *The PVM Concurrent Computing System: Evolution, Experiences, and Trends*, Journal of Parallel Computing, 1993.

TAN88 Tanenbaum, A. S., *Computer Networks*, Prentice Hall, 1988.

TAN92 Tanenbaum, A. S., *Modern Operating Systems*, Prentice Hall, 1992.

THU78 Thurber, K. J., *Circuit Switching Technology: a State-of-the-Art Survey*, Distributed Processor Communication Architecture, IEEE Computer Society, p. 338-346, 1979.

TMC92 Thinking Machines Corporation, *Connection Machine CM-5*, Technical Summary, Cambridge - Massachusetts, novembro de 1992.

TRE88 Treleaven, Ph. C., *Parallel Architecture Overview*, Parallel Computing, n. 8, p. 59-70, 1988.

TRE91 Trew, A., Wilson, G., *Past, Present, Parallel: A Survey of Available Parallel Computing Systems*, Springer-Verlag, 1991.

WAI90 Waille, P., *Introduction à l'Architecture des Machines Supernode*, relatório técnico RT56, LGI, IMAG, fevereiro de 1990.

WAL94 Walker, D. W., *The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers*, Parallel Computing, 1994.

WAL82 Wall, D. H., *Messages as Active Agents*, Proceedings of the 9th Annual Symposium on Principles of Programming Languages, junho de 1982.

WAL97 Walter, B. L., Umahishore, R., *Toward a More Realistic Performance Evaluation of Interconnection Networks*, IEEE Transactions on Parallel and Distributed Systems, v. 8, n. 7, julho de 1997.

WIL87 Wilson, A. W., *Hierarchical Cache/Bus Architecture for Shared-Memory Multiprocessors*, XIV International Symposium on Computer Architecture, p. 244-252, 1987.

WIT81 Wittie, L. D., *Communication Structures for Large Networks of Microcomputers*, IEEE Transactions on Computers, v. C-30, n. 4, abril de 1981.

WU 92 Wu, C.-L., Lee, M., *Performance Analysis of Multistage Interconnection Network Configurations and Operations*, IEEE Transactions on Computers, v. 41, n. 1, p. 17-27, janeiro de 1992.

YOU87 Young, M., et al., *The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System*, XI Symposium on Operating Systems Principles, ACM, p. 63-76, novembro de 1987.

YOU94 Youn, H. Y., Chen, C. C.-Y., *A Comprehensive Performance Evaluation of Crossbar Networks*, IEEE Transactions on Parallel and Distributed Systems, v. 4, n. 5, p. 481-489, julho de 1994.

ZEF95 Zeferino, C. A., Lucke, H. A. H., Silva, V. A., *Um Multicomputador com Sistema Experimental de Comunicação*, VII Simpósio de Arquitetura de Computadores e Processamento de Alto Desempenho, Canela, julho de 1995.

ZEF96 Zeferino, C. A., *Projeto do Sistema de Comunicação de um Multicomputador*, Dissertação de Mestrado, CPGCC / UFSC, 1996.