UNIVERSIDADE DE SÃO PAULO - USP
Instituto de Física e Química de São Carlos - IFQSC
Departamento de Física e Ciência dos Materiais

**SERVIDOR DE PROCESSAMENTO
PARALELO BASEADO EM BANCO
DE TRANSPUTERS**

Onofre Trindade Jr

**Tese Apresentada ao Instituto de Física e
Química de São Carlos para a Obtenção
do Título de Doutor em Ciências (Física
Aplicada).**

**Orientador:
Prof. Dr. Marcos José Santana**

São Carlos, SP - 1991

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE ONOFRE TRINDADE JUNIOR APRESENTADA AO
INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 18/12/1991

COMISSÃO JULGADORA:

Prof. Dr. Marcos José Santana

Prof. Dr. Alvaro Garcia Neto

Prof. Dr. Fernando Stella de Rodrigues Germano

Prof. Dr. Adriano Joaquim de Oliveira Cruz

Prof. Dr. Antonio Marcos de Aguirre Massola

*Aos meus pais, não apenas pelas palavras
encorajadoras, mas pelo apoio sincero, silencioso,
sentido na atmosfera especial que espero nunca se
desvaneça.*

O fato deste trabalho apresentar um único autor não é condição suficiente para torná-lo individual. Injusto seria não agradecer nesse espaço menor a enorme contribuição dada pelo prof. Marcos Santana, mentor do projeto e seu principal crítico; ao prof. Paulo Masiero pela inestimável contribuição dada no tratamento estruturado do problema; ao prof. Nivaldi e aos colegas Ana Luísa e Orlando, pelas intermináveis horas dispendidas na intrincada tarefa de programação de partes do software do sistema. Não poderia deixar de mencionar os técnicos Eduardo e Cabral, pelo seu paciente trabalho na montagem das placas.

AGRADECIMENTOS

É necessário ressaltar o apoio dado pela FAPESP, sem o qual não seria possível a realização deste trabalho.

A todos, mais aqueles que o espaço reduzido impediu que fossem nominalmente mencionados, sou grato pela amizade recebida.

RESUMO

Limitações de ordem física têm imposto limites máximos no aumento de desempenho de computadores baseados na arquitetura de von Neumann. Sistemas computacionais para processamento paralelo representam a alternativa mais promissora para superar esses limites.

O nível de paralelismo dos problemas pode resultar em uma subutilização das máquinas paralelas, particularmente em sistemas monousuários utilizados para a solução de problemas com baixo nível de paralelismo. Isso contribui para tornar os sistemas paralelos mais onerosos.

Uma nova arquitetura, baseada em um banco de transputers, é sugerida neste trabalho, introduzindo recursos para processamento paralelo para os usuários de um sistema distribuído baseado em uma rede local de computadores.

Neste trabalho são discutidos os pontos relevantes do desenvolvimento do sistema. No projeto do servidor de processamento paralelo foram levados em consideração diversos aspectos relacionados com utilização facilitada e custo reduzido. A flexibilidade de operação também foi considerada.

Os resultados obtidos dos testes efetuados mostraram que os principais objetivos foram atingidos. Duas linhas para a continuidade da pesquisa são propostas: Melhoramentos no sistema para incorporação de novas características e revisão da especificação inicial. Também são propostos diversos temas para novos trabalhos relacionados com o servidor.

ABSTRACT

Physical constraints have imposed performance bounds on von Neumann's architecture. Parallel computers are at present the best solution to overcome this limitation.

The cost/benefit relation in single-user parallel systems depends on the parallelism of the problem. Little parallelism implies poor cost/benefit relation. This contributes to make highly parallel systems more expensive.

A new architecture based on a Tranputer bank, named in this thesis "a parallel computing server", is suggested to provide parallel processing power to the user community of a LAN-based computing system.

The design of the parallel computing server took into account several aspects concerning the cost/benefit relation and ways to use the system more effectively. Flexibility was another key point.

This work discusses all major design and implementation aspects, aiming at recording the required knowledge for further work on the server proposed.

The results experimented in a prototype of the parallel computing server show that the main goals were reached, with parallel processing power being available at an effective cost/benefit relation.

Two main lines for further investigation are proposed: Improvements to the prototype of the parallel computing server in order to test additional characteristics and revision of the initial specification. Furthermore, several project themes concerning the system proposed are also suggested.

ÍNDICE

RESUMO	IV
ABSTRACT	V
ÍNDICE DAS FIGURAS	IX
GLOSSÁRIO	XII

1 INTRODUÇÃO

1.1	Motivação deste Trabalho	1
1.2	Descrição e Abordagem do Problema	3
1.3	Apresentação	4

2 CONTEXTO DE DESENVOLVIMENTO

2.1	Considerações Iniciais	6
2.2	Processamento Paralelo	7
2.3	Transputers e Linguagem Occam	19
2.4	Sistemas com Transputers	35
2.5	Sistemas Distribuídos	38
2.6	Considerações Finais	46

3 ARQUITETURA E ESPECIFICAÇÃO DO SISTEMA

3.1	Considerações Iniciais	48
3.2	Requisitos do Sistema	49
3.3	Ambiente de Utilização	53
3.4	Definição Geral do Sistema	55
3.5	Especificação do Hardware	59
3.6	Especificação do Software	71
3.7	Considerações Finais	95

4 PROJETO E IMPLEMENTAÇÃO DO SISTEMA

4.1	Considerações Iniciais	96
4.2	Projeto e Implementação do Hardware	97
4.3	Projeto e Implementação do Software	107
4.4	Instalação do Sistema de Desenvolvimento TDS	124
4.5	Considerações Finais	128

5 VALIDAÇÃO DO SISTEMA

5.1	Considerações Iniciais	130
5.2	Testes do Hardware	131
5.3	Testes do Software	135
5.4	Exemplo de Utilização do Sistema em Ambiente TDS	142
5.5	Comparação com Outros Sistemas Baseados em Transputers	144
5.6	Evolução do Sistema	146
5.7	Considerações Finais	152

6 CONCLUSÕES

6.1	Considerações Iniciais	153
6.2	Contribuições Deste Trabalho	154
6.3	Sugestões Para a Continuidade do Trabalho	155
6.4	Comentários Finais	161

REFERÊNCIAS BIBLIOGRÁFICAS	162
-----------------------------------	------------

APÊNDICES

A1	Análise da Rede de Chaveamento	169
A2	Descrição de Interface das Funções - PD	173
A3	Descrição de Interface dos Módulos - ET	177
A4	Descrição de Interface dos Módulos - PE	181
A5	Descrição de Interface dos Módulos - CM	185

ÍNDICE DAS FIGURAS

2.1	Esquema de Classificação de Computadores Segundo Flynn	9
2.1	Esquema de Classificação de Computadores Segundo Flynn - cont	10
2.2	Diagrama de Blocos Detalhado - T800	22
2.3	Protocolo de Comunicação - Canais INMOS	25
2.4	Processos e Comunicação Entre Processos na Linguagem OCCAM	25
2.5	Configuração Típica para Execução do TDS	30
2.6	Configuração de Carga e Execução Programas EXE	30
2.7	Configuração de Carga e Execução Programas PROGRAM	34
2.8	Configuração de Carga e Execução Programas PSP/SHP	34
2.9	"Supernode" P1085	36
2.10	Supercomputador Composto por "Supernodes"	36
2.11	Pacote de Dados Ethernet	43
2.12	Tratamento de Falhas em RPCs	45
3.1	Rede Local com Segmentos Interligados por Pontes	52
3.2	Visão Geral do Sistema	54
3.3	Servidor de Processamento Paralelo	54
3.4	Estrutura Hierárquica do Banco de Transputers	58
3.5	Diagrama de Blocos - Processador	58
3.6	Lote com 3 Processadores	60

3.7	Interface do Barramento de Controle	60
3.8	Diagrama de Blocos de um Módulo	63
3.9	Rede de Chaveamento "CrossBar"	63
3.10	Rede de Chaveamento do Módulo	65
3.11	Rede de Chaveamento do Banco	66
3.12	Anel Formado pelos Controladores dos Módulos e do Banco	69
3.13	Diagrama de Blocos - Controlador de Banco	70
3.14	Diagrama de Blocos - Controlador de Módulo	70
3.15	Símbolos Utilizados nos Diagramas Relativos ao Software	73
3.16	Mapa de Estados - PD	75
3.17	Organização dos Pares Cliente-Servidor	77
3.18	Mapa de Estados - ET (ME-ET)	79
3.19	Diagrama de Fluxo de Dados - ET (DFD-ET)	82
3.20	Diagrama de Estrutura de Tarefas ET (DET-ET)	83
3.21	Diagrama de Estrutura de Tarefas PE (DET-PE)	83
3.22	Mapa de Estados - PE (ME-PE)	85
3.23	Diagrama de Fluxo de Dados - PE (DFD-PE)	86
3.24	Mapa de Estados - CM (ME-CM)	88
3.25	Diagrama de Fluxo de Dados - CM (DFD-CM)	89
3.26	Diagrama de Estrutura de Tarefas CM (DET-CM)	90
3.27	Estrutura em Camadas do Software - ET	92
3.28	Estrutura em Camadas do Software - PE	93
3.29	Estrutura em Camadas do Software - CM	94
4.1	Diagrama de Blocos - Transputer T800	99
4.2	Diagrama de Blocos - C011/12 (modo 2)	99

4.3	Interface com Barramento de Controle	
	Processador	100
4.4	Placa de Processadores	101
4.5	Diagrama de Blocos - C004	103
4.6	Rede de Chaveamento do Módulo	104
4.7	Placa de Chaves	106
4.8	Placa do Controlador de Módulo	108
4.9	Placa da Interface PE/BT	109
4.10	Definição dos Pacotes de Dados	111
4.11	Diagramas de Estrutura Modular	
	ET (DEM-ET)	116
4.12	Diagramas de Estrutura Modular	
	PE (DEM-PE)	118
4.13	Diagramas de Estrutura Modular	
	CM (DEM-CM) cont	120
4.13	Diagramas de Estrutura Modular	
	CM (DEM-CM) cont ...	121
4.13	Diagramas de Estrutura Modular	
	CM (DEM-CM) cont ...	122
4.13	Diagramas de Estrutura Modular	
	CM (DEM-CM) cont ...	123
4.14	Diagrama de Estrutura Modular ISERVER	126
5.1	Configuração de Hardware Utilizada para	
	Testes de Software	137
5.2	Subdivisão do Sistema para Implementação	
	e Testes de Software	137
5.3	Esquema para Testes do Software do	
	Controlador de Módulo	140
5.4	Configuração para Execução de Programas	
	em Lotes Controlados pelo TDS	149

GLOSSÁRIO

ANALYSE	Sinal aplicado a um Transputer para possibilitar a leitura de dados internos sem reiniciação. Utilizado como parte do mecanismo utilizado na depuração de programas.
BT	Banco de Transputers. Constituído pelos processadores, controlador de banco, controladores de módulo, rede de chaveamento e componentes do software do sistema relacionados com a sua operação básica. Apresenta uma estrutura expansível, baseada em módulos.
CB	Controlador do Banco. Computador responsável pelo controle do banco de Transputers. Comunica-se com os controladores de módulo e com o processador de entrada.
CE	Canal ETL. Canais utilizados para conexão de estações de trabalho locais.
CM	Controlador de Módulo. Computador responsável pelo controle de um módulo do banco de Transputers. Comunica-se com outros controladores de módulo e/ou com o controlador do banco.
CRC	"Cyclic Redundancy Code". Código binário utilizado para detecção de correção de erros, em particular, em pacotes de dados ethernet.
CSP	"Communicating Sequential Processes". Linguagem desenvolvida por Hoare para especificação de sistemas concorrentes.
CU	Canal de Usuário. Canais utilizados na comunicação entre lotes de processadores e controladores de módulo/banco.

DARTS	"Design Approach for Real Time Systems". Metodologia para desenvolvimento de software empregada neste trabalho.
DEM	Diagrama de Estrutura Modular. Diagrama utilizado em projeto estruturado para representar os diversos módulos de um sistema e a comunicação entre eles.
DET	Diagrama de Estrutura de Tarefas. Diagrama representando as diversas tarefas, a comunicação e a sincronização entre elas, conforme a metodologia DARTS.
DFD	Diagrama de Fluxo de Dados. Diagrama utilizado em análise/projeto estruturados para representar ações sobre fluxos de dados em um sistema.
EL	Endereço de Lote. Número entre 1 e 15 (4 bits) indicando o endereço de um lote dentro de um Módulo.
EP	Endereço de Processador. Número com 5 bits indicando o endereço físico de um Processador dentro de um Lote.
ERROR	Sinal acionado por um Transputer para indicar a ocorrência de erros internos de processamento.
ETH	Meio físico e protocolo de Comunicação para redes locais de computadores. Implementa até a subcamada MAC do modelo de referência ISO-OSI.
ET	Estação de Trabalho.
ETL	Estação de Trabalho Local. Estação de Trabalho acoplada diretamente ao banco de Transputers.
EXE	Tipo de programa TDS executável sob o TDS.

IHM	"Information Hiding Module". Módulo de software composto por um tipo abstrato de dados utilizado na comunicação entre tarefas concorrentes, conforme metodologia DARTS.
ISERVER	Módulo de software servidor do sistema de desenvolvimento TDS.
ISO	"International Standard Organization". Organização internacional voltada para o estabelecimento de normas técnicas da qual o Brasil figura como país membro.
LLC	"Logical Link Control". Subcamada de mais alto nível da camada de enlace de dados no modelo de referência ISO-OSI.
LNK	Canal físico de comunicação INMOS.
LT	Lote de Transputers. Conjunto de processadores Transputers alocados para um usuário do servidor de processamento paralelo.
ME	Mapa de Estados. Ferramenta gráfica para a representação de estados e transição entre estados em um sistema.
MAC	"Medium Access Control". Subcamada de mais baixo nível da camada de enlace de dados no modelo de referência ISO-OSI.
MCM	"Message Communication Module". Módulo de comunicação entre tarefas concorrentes conforme metodologia DARTS.
MM	Módulo de Memória. Subsistema contendo memória de leitura/escrita em um computador.
MT	Módulo de Transputers. Um subconjunto de processadores do banco de Transputers, controlados por um controlador de módulo. Uma rede de chaveamento a nível de módulo possibilita a comunicação entre os processadores.

NCSA	National Center for Supercomputer Applications.
OCCAM	Linguagem de alto nível baseada na linguagem CSP. Dispõe de mecanismos intrínsecos para a sincronização de processos concorrentes.
OSI	"Open Systems Interconnection" - Reference Model. Modelo de referência ISO para conexão de sistemas abertos.
PAL	"Programmable Array Logic". Dispositivo lógico programável utilizado para a implementação de funções lógicas.
P	Processador. Constituído por um computador Transputer, memória e circuitos de interface com a rede de chaveamento e o controlador de módulo. Constitui a unidade básica do recurso de processamento compartilhado entre os usuários do servidor de processamento paralelo.
PD	"Packet Driver" - NCSA . Mecanismo para acesso e multiplexação de pacotes de dados implementado na universidade de Clarkson.
PE	Processador de Entrada. Computador responsável pela comunicação do servidor de processamento paralelo com a rede local e pelo atendimento de parte dos serviços disponíveis no SPP.
PKT	Protocolo de comunicação utilizado pelo "Packet Driver" - NCSA.
PT	Processador de Trabalho. Processador fazendo parte de um Lote que não se comunica diretamente com o controlador de módulo.
PR	Processador Raiz de Lote. Processador fazendo parte de um lote que se comunica diretamente com o controlador de módulo.
PROGRAM	Tipo de programa TDS constituído por módulos SC. Executável em uma rede de Transputers sob supervisão do TDS.

PSP	"Parameterless Standalone Program". Tipo de programa TDS executável em uma rede de Transputers.
RAM	"Random Access Memory". Memória de Leitura/Escrita.
RC	Rede de Chaveamento. Malha composta por chaves eletrônicas utilizada para o roteamento de dados entre processadores.
REDE LOCAL	Componentes de hardware e software relacionados com a conexão de estações de trabalho e servidores.
RESET	Sinal de reiniciação aplicado em Transputers para o estabelecimento das condições iniciais de funcionamento.
ROM	"Read Only Memory". Memória não volátil de somente leitura.
RPC	"Remote Procedure Call". Mecanismo para chamada de procedimentos remotos. Utilizado neste trabalho para implementar a comunicação entre os pares servidor - cliente.
RP	"Remote Procedure". Procedimento remoto associado a um mecanismo RPC.
SAI	Servidor de Arquivo e Impressão.
SC	Módulo de software TDS compilado separadamente.
SHP	"Standard Host Procedure". Tipo de programa TDS executável em uma rede de Transputers.
SISTEMA SPP	Todos os itens de hardware e software relacionados com a operação do servidor de processamento paralelo.

SISTEMA DISTRIBUÍDO	Sistema computacional composto por vários computadores autônomos, geograficamente separados, operando cooperativamente, de maneira transparente aos usuários do sistema distribuído.
SP	"Server Protocol" - INMOS. Protocolo utilizado para comunicação entre o programa iserver e redes de Transputers.
SPP (1)	Servidor de Processamento Paralelo. Constituído pelo banco de Transputers, pelo processador de entrada, processadores, e componentes do software do sistema responsáveis pela operação básica desses.
SPP (2)	Protocolo de comunicação utilizado pelo mecanismo de chamadas a procedimentos remotos utilizado no servidor de processamento paralelo.
TCP/IP	"Transmission Control Protocol/Internet Protocol". Conjunto de protocolos de comunicação utilizados em redes de computadores locais e de grande distância.
TDS	"Transputer Development System". Sistema para o desenvolvimento de software em Transputers. Inclui editor de textos baseado em dobras ("folds"), compilador OCCAM, editor de ligação, depurador, carregadores e diversos outros programas utilitários.
TRANSPUTER	Microcomputador fabricado pela INMOS. Possui características que o tornam indicado como elemento básico na implementação de máquinas paralelas.
UC	Unidade de Controle. Subsistema de um computador responsável pela busca, interpretação e execução das instruções.

- UF** **Unidade Funcional.** Subsistema de um computador responsável por uma função lógica definida.
- UP** **Unidade de Processamento.** Subsistema lógico de um computador responsável pela execução das instruções.

CAPÍTULO 1

INTRODUÇÃO

1.1 Motivação deste Trabalho

O processamento paralelo representa a principal alternativa para romper os limites no desempenho impostos pela arquitetura de von Neumann. Fatores físicos diversos têm determinado limites máximos para o aumento da velocidade de comutação dos circuitos eletrônicos utilizados na construção das máquinas. Entre esses, a própria velocidade limite de propagação da luz representa uma barreira. Por outro lado, o cérebro humano, uma máquina lenta considerada a sua tecnologia, apresenta desempenho surpreendente ao resolver problemas complexos tal como o reconhecimento de padrões gráficos. A que se deve isso? A algoritmos efficientíssimos ou ao esforço cooperativo de muitas unidades de processamento?

Entre os diversos modelos computacionais, arquiteturas e sistemas voltados para processamento paralelo, o Transputer, juntamente com a linguagem Occam, ganharam muitos adeptos na Europa [HE90b] [RO90]. Considerado um bloco construtivo básico, o Transputer permite a construção de máquinas MIMD com memória distribuída de uma maneira racional. O poder computacional e a facilidade de implementação de máquinas paralelas proporcionados pelo hardware, aliados a uma implementação eficiente da linguagem Occam, resultam em sistemas para processamento paralelo de baixo custo e utilização simplificada [TR90].

O emprego de processadores trabalhando cooperativamente apresenta, entretanto, algumas restrições. Entre essas, o grau de paralelismo de cada aplicação implica muitas vezes em uma utilização parcial dos recursos da máquina paralela. Esse fato contribui para

que sistemas paralelos monousuários, dotados de muitos processadores, apresentem uma relação custo/benefício desfavorável. Outro aspecto que impede uma maior difusão das máquinas paralelas é a multiplicidade de modelos computacionais existentes [HE91], não permitindo o transporte de software entre diferentes máquinas. A dificuldade de utilização também é responsável pela pequena difusão das arquiteturas paralelas.

Para resolver os problemas relacionados com o processamento paralelo, sob o ponto de vista do usuário, esforços devem ser feitos nas seguintes atividades de pesquisa:

- > Desenvolvimento de arquiteturas paralelas universais, isto é, modelos que possibilitem o desenvolvimento de software desvinculado da arquitetura física da máquina destino [VA89].
- > Desenvolvimento de metodologias, técnicas e ferramentas de engenharia de software para o desenvolvimento de programas paralelos.
- > Desenvolvimento de ferramentas para geração automática de código paralelo baseadas em paradigmas de programação.
- > Desenvolvimento de algoritmos paralelos eficientes para solução de problemas básicos, criando-se bibliotecas de funções.
- > Desenvolvimento de ambientes e interfaces homem/máquina que facilitem a utilização dos sistemas paralelos.
- > Desenvolvimento de arquiteturas custo/efetivas, incentivando sua disseminação.

Este trabalho constitui uma contribuição para as duas últimas atividades mencionadas, gerando recursos para o desenvolvimento das demais. Seu principal objetivo consiste no desenvolvimento de uma arquitetura paralela, baseada em Transputers, que possibilite a utilização simples e racional de recursos para processamento paralelo. Esses recursos devem ser compartilhados, otimizando a relação custo/benefício do sistema desenvolvido. O objetivo principal se desdobra em outros dois: tornar disponível um sistema computacional aberto, voltado para processamento paralelo, aos seus potenciais usuários e tornar disponíveis equipamentos e ferramentas computacionais para fomento de

pesquisa nas áreas de processamento paralelo e sistemas distribuídos. Como objetivo secundário, mas igualmente importante, pretende-se adquirir conhecimento tecnológico através do projeto de cada subsistema, selecionando metodologias estruturadas para o desenvolvimento integrado de hardware e software, que não somente auxiliem o processo de desenvolvimento, mas também gerem documentação consistente para permitir futuros trabalhos em equipe [PR87].

1.2 Descrição e Abordagem do Problema

Bancos de Processadores têm sido sugeridos como uma alternativa eficaz para o compartilhamento de recursos de processamento [BA89b]. Redes locais de computadores [CU88] representam um meio eficiente para proporcionar esse compartilhamento. Transputers, e mais recentemente IWarps [PE91], podem ser vistos como blocos construtivos básicos para a implementação de máquinas paralelas. O problema considerado neste trabalho pode ser resumido em como combinar esses três elementos básicos para dar origem a uma máquina paralela, que atenda aos objetivos mencionados no item anterior.

O projeto prevê o desenvolvimento de cada subsistema de hardware e software. A utilização de subsistemas comerciais para o desenvolvimento do projeto apresenta uma série de inconvenientes. Em primeiro lugar, não se teria domínio sobre cada etapa do projeto do sistema, dificultando sua contínua evolução. Segundo, o sistema deve ser aberto, permitindo pesquisa nas áreas de conhecimento envolvidas. Terceiro, pretende-se estudar a interface dos usuários com o sistema, particularmente quanto aos aspectos de transparência e facilidade de utilização. Para isso existe novamente a exigência de um sistema que possa evoluir dinamicamente. Por último, deve-se considerar a minimização do custo do sistema e a facilidade da sua duplicação, contribuindo para a disseminação do sistema e incentivando novas pesquisas.

O desenvolvimento de um sistema computacional complexo, envolvendo software e hardware, deve ser feito utilizando-se alguma metodologia em cada uma das etapas desse desenvolvimento. Neste trabalho são utilizadas as idéias apresentadas em [MI89]. No

desenvolvimento do software foi adotada a metodologia DARTS [GO84] e mapas de estado, convenientemente adaptados e integrados à natureza do desenvolvimento, no sentido de atender um dos objetivos estabelecidos.

1.3 Apresentação

Devido à extensão das etapas do desenvolvimento do sistema servidor de processamento paralelo, este trabalho não inclui todas as atividades previstas em cada uma delas. Ressalvando-se a característica dinâmica do projeto, procurou-se especificar todo o sistema, projetar parte dele e implementar um núcleo básico que permita um mínimo de operacionalidade. Este trabalho representa um marco no desenvolvimento do sistema, não o seu término.

O trabalho está dividido em seis capítulos e cinco apêndices. Foi incluído um glossário com as principais abreviaturas e conceitos utilizados no trabalho, particularmente aqueles que aparecem nas figuras.

Este capítulo apresenta a motivação do trabalho, uma descrição do problema e a abordagem adotada na sua solução. Essas idéias norteiam todo o restante do trabalho.

O capítulo 2 apresenta os tópicos relacionados com o desenvolvimento do trabalho à luz da bibliografia consultada. É feita uma revisão dos conceitos de processamento paralelo, Transputers, Linguagem Occam, sistema de desenvolvimento TDS, sistemas distribuídos e redes de computadores. Esses conceitos são importantes para o entendimento do restante do trabalho. Alguns sistemas baseados em Transputers são apresentados a título de exemplo.

O capítulo 3 apresenta as idéias centrais do trabalho, na forma da arquitetura e especificação do sistema, procurando definir os requisitos e o ambiente de utilização desejados, tomando-se por base a motivação apresentada neste capítulo. Na especificação de hardware e software procura-se prever a continuidade do desenvolvimento do sistema.

Parte da especificação apresentada no capítulo 3 é traduzida em projeto e implementação no capítulo 4. São apresentados o projeto e implementação do hardware e

do software do sistema. A estrutura do sistema de desenvolvimento TDS é apresentada, bem como as alterações que se fazem necessárias para a sua instalação.

O capítulo 5 ficou reservado à validação do sistema. Estão incluídos os testes realizados nos principais subsistemas de hardware e software componentes do servidor. Subsistemas de funcionamento crítico, como memória dinâmica, foram testados com cuidado especial. Um exemplo de utilização do sistema em ambiente TDS é apresentado. Foi incluída uma comparação das características do servidor de processamento paralelo com outros sistemas baseados em Transputers. No final do capítulo 5 são feitas diversas considerações a respeito da evolução do sistema.

No capítulo 6 são comentadas as contribuições introduzidas por este trabalho. Uma série de novos trabalhos é motivada com o término deste. Uma descrição sucinta desses trabalhos é apresentada no capítulo 6, na sua maior parte baseada nas considerações apresentadas no final do capítulo 5.

Algumas considerações sobre a seleção da arquitetura da rede de chaveamento do servidor são apresentadas no apêndice A1. Nos demais apêndices são mostradas as especificações de interface dos módulos do software do sistema. Essas especificações complementam os diagramas de estrutura modular apresentados ao longo do texto. A documentação adicional utilizada no desenvolvimento do projeto pode ser encontrada em [TR91a] [TR91b] [TR91c].

CAPÍTULO 2

CONTEXTO DE DESENVOLVIMENTO

2.1 Considerações Iniciais

O tema central e a motivação deste trabalho, bem como o método empregado no seu desenvolvimento, foram descritos em linhas gerais no capítulo anterior. Neste capítulo é feito um compêndio dos tópicos relevantes relacionados com o trabalho. Devido à diversidade desses tópicos, somente os pontos importantes para a situação e entendimento do problema são apresentados em detalhe.

O item 2.2 trata de processamento paralelo. É apresentada uma classificação para sistemas paralelos e discutidas algumas arquiteturas típicas.

A família de processadores INMOS - Transputer é introduzida no item 2.3, juntamente com a linguagem Occam e o sistema de desenvolvimento TDS.

No item 2.4 são apresentados exemplos de sistemas computacionais baseados em Transputers. Alguns desses sistemas são de interesse apenas acadêmico enquanto outros estão disponíveis comercialmente.

Sistemas distribuídos e redes de computadores são assuntos introduzidos no item 2.5, com ênfase especial em sistemas distribuídos baseados em redes locais de computadores. O modelo de referência ISO-OSI e o padrão Ethernet para redes locais são apresentados. É discutida a implementação do modelo cliente-servidor baseada no mecanismo de chamadas a procedimentos remotos (RPC).

2.2 Processamento Paralelo

A idéia do processamento paralelo, isto é, a realização de diversas atividades computacionais simultaneamente no tempo, não é recente. Vannevar Bush (MIT), apresentou em 1920 um computador analógico capaz de resolver equações diferenciais em paralelo. von Neumann, na década de 40, sugeriu sistemas para solução de equações diferenciais em paralelo. Apesar disso, o aparecimento do ILLIAC IV, na universidade de Illinois em 1967, é considerado o marco inicial do processamento paralelo [NA89]. As restrições de desempenho impostas pelo modelo computacional de von Neumann, agravadas por limitações de ordem tecnológica, têm motivado o desenvolvimento de novos modelos computacionais explorando o conceito de processamento paralelo.

Apesar de existir alguma divergência na bibliografia consultada, é adotado neste trabalho o termo processamento concorrente para designar qualquer forma de execução simultânea de duas atividades computacionais. Entre essas incluem-se, por exemplo, os processos executados simultaneamente em sistemas multi-tarefas por tempo compartilhado. Por processamento paralelo entende-se aquele onde as atividades são executadas simultaneamente no tempo, em diferentes unidades de processamento.

2.2.1 Classificação das Arquiteturas de Computadores Segundo Flynn

Diversos esquemas para a classificação dos computadores têm sido introduzidos, destacando-se Flynn, Treleaven, Feng e Handler [NA89] [AL89]. O esquema proposto por Flynn tem sido criticado por falhar na descrição de algumas máquinas e não fazer distinção entre outras. Apesar disso, esse esquema é largamente aceito na comunidade científica devido à sua simplicidade e clareza.

A classificação de computadores, segundo Flynn, baseia-se na multiplicidade dos fluxos de dados e instruções existentes, definindo-se quatro classes:

- > SISD ("Single Instruction Single Data")
Fluxo de instruções único - Fluxo de dados único

- > SIMD ("Single Instruction Multiple Data")
Fluxo de instruções único - Fluxo de dados múltiplo
- > MISD ("Multiple Instruction Single Data")
Fluxo de instruções múltiplo - Fluxo de dados único
- > MIMD ("Multiple Instruction Multiple Data")
Fluxo de instruções múltiplo - Fluxo de dados múltiplo

A figura 2.1 mostra os diagramas de blocos correspondentes à organização de cada uma das quatro classes mencionadas. Três componentes são representados nesses diagramas:

- > módulo de memória (MM) - armazena instruções e dados
- > unidade de controle (UC) - decodifica instruções
- > unidade de processamento (UP) - executa instruções

Cada unidade de controle gera um fluxo de instruções. O fluxo de dados entre unidades de processamento e módulos de memória é bidirecional.

A classe SISD representa a máquina seqüencial de von Neumann. A maioria dos computadores seqüenciais existentes se enquadra nessa categoria. Uma unidade de processamento pode ser composta por várias unidades funcionais, comandadas pela unidade de controle. Alguns autores, como Hwang [HW84], enquadram nessa classe os computadores com unidades funcionais operando em "pipeline". Outros acreditam que a classe MISD é mais adequada para descrevê-los ou simplesmente admitem a insuficiência do esquema proposto por Flynn.

Unidades operando em "pipeline" assemelham-se uma linha de produção onde cada tarefa é decomposta em subtarefas executadas por diferentes unidades funcionais. Quando em regime, uma tarefa é executada no mesmo tempo da subtarefa de maior duração. Unidades de processamento "pipeline" apresentam paralelismo temporal, isto é, diferentes subtarefas componentes de diferentes tarefas são executadas no mesmo instante de tempo.

FD Fluxo de Dados
FI Fluxo de Instruções
MM Modulo de Memoria
UC Unidade de Controle
UP Unidade de Processamento

2.1 Esquema de Classificação de Computadores Segundo Flynn

MISD

MIMD

2.1 Esquema de Classificação de Computadores Segundo Flynn - cont ...

Processadores vetoriais constituem exemplos de máquinas com uma ou mais unidades de processamento operando em "pipeline". Processamento vetorial é o principal conceito explorado no projeto dos atuais supercomputadores.

A classe MISD caracteriza-se por múltiplos fluxos de instruções operando sobre um único fluxo de dados. Alguns autores consideram que essa classe não apresenta interesse prático, não existindo máquinas que se enquadrem nessa categoria.

Máquinas SIMD envolvem um único fluxo de instruções que agem simultaneamente sobre múltiplos fluxos de dados, através de múltiplas unidades de processamento. Processadores matriciais são enquadrados nessa categoria. O ILLIAC IV é uma máquina SIMD com 64 unidades de processamento. Em [AL89] é feita uma distinção entre as máquinas SIMD. Máquinas com unidades de processamento operando em "pipeline" são classificadas como SIMD "pipeline". Processadores com múltiplas unidades de processamento, tal como o ILLIAC IV, são classificados como SIMD paralelo.

Máquinas MIMD possuem múltiplos fluxos de instruções operando sobre múltiplos fluxos de dados. Cada processador opera sobre seu próprio conjunto de dados. A tabela 2.1 mostra alguns exemplos de máquinas classificadas, segundo Flynn.

2.2.2 Classificação das Arquiteturas de Computadores Segundo Treleaven

Treleaven [AL89] introduz uma classificação adicional para as máquinas MIMD, conforme mostrado na tabela 2.2.

O mecanismo de dados é subdividido entre memória compartilhada, (von Neumann), e comunicação por passagem de mensagens. Quatro mecanismos de controle são relacionados em ordem decrescente de rigidez. Máquinas von Neumann apresentam fluxo de controle rígido em oposição às máquinas dirigidas por fluxos de dados.

Comunicação entre processos é a base dos atuais multi-computadores, tais como sistemas implementados com processadores Transputer.

Classe	Modelos de Computadores/Processadores
SISD uma UF	IBM701, IBM1620, IBM7090, VAX11/780
SISD mltiplas UF	IBM 360/91, IBM 370/198, CDC 6600, TI-ASC, CRAY-1, FUJITSU VP-200
SIMD palavra	ILLIAC IV, PEPE, BSP
SIMD bit	STARAN, DAP, MPP
MIMD mem dist	IBM 370/168 MP, TANDEM, IBM 3084, CM* Sistemas com TRANSPUTERS e IWARPs
MIMD mem comp	CRAY-2, BURROUGHS D-825, CRAY-X MP, C.MMP

Tabela 2.1 Algumas Máquinas Classificadas Segundo Flynn

Nas máquinas dirigidas por fluxo de dados uma instrução pode ser executada tão logo os operandos necessários se tornem disponíveis. Nas máquinas dirigidas por demanda uma instrução pode ser executada quando seu resultado é necessário como operando para outra instrução, pronta para execução. Uma comparação entre fluxo de dados e demanda considera a tentativa de melhorar o modelo computacional de von Neumann em dois aspectos distintos. Máquinas de fluxo de dados tentam otimizar a transferência de dados fazendo com que esses fluam diretamente de instrução para instrução. Máquinas de redução tentam facilitar o processo de programação. Uma série de vantagens é atribuída à utilização de linguagens funcionais nessa classe de máquinas [AL89].

2.2.3 Outras Arquiteturas Paralelas

Além das máquinas discutidas, existem outras com elevado desempenho que não se encaixam perfeitamente na classificação de Flynn. Entre essas podem ser citadas as matrizes sistólicas e redes neurais.

Mecanismo de Controle	Mecanismo de Dados	
	Mem Comp	Mem Dist
Dirigido por Controle	Tipo von Neumann	Comunicação entre Processos
Dirigido por Padrão	Lógica	Atores
Dirigido por Demanda	Redução de Grafos	Redução de Cadeias
Dirigido por Dados	Fluxo de Dados Estrutura I	Fluxo de Dados Tokens

Tabela 2.2 Classificação das Máquinas MIMD Conforme os Mecanismos de Controle e Dados

Matrizes sistólicas ("systolic arrays") foram propostas por Kung [KU82] para a solução de problemas específicos. Elas podem ser vistas como "pipelines" multidimensionais. (normalmente bidimensionais). Os dados fluem, através das unidades funcionais que compõem a matriz, sincronamente sob o comando de um "clock" único. O termo sistólico foi adotado por analogia do fluxo de dados com o fluxo de sangue impulsionado pelos movimentos sistólicos do coração. Uma das vantagens das matrizes sistólicas é a transmissão direta de dados entre unidades funcionais, sem necessidade de armazenamento intermediário em memória.

Processadores "Wavefront" combinam matrizes sistólicas com o paradigma envolvido no mecanismo dirigido por dados [DU90]. A sincronização provida pelo "clock" nas matrizes sistólicas é substituída por um mecanismo assíncrono de transmissão de dados entre os elementos do "pipeline" multidimensional.

Um esforço conjunto da Universidade Carnegie Mellon e da Intel produziu o microprocessador IWarp [PE91]. O IWarp, assim como o Transputer, é um microcomputador projetado para a implementação de sistemas multicomputadores (MIMD com memória distribuída). O IWarp dá suporte aos mecanismos de comunicação via passagem de mensagens e o mecanismo característico das matrizes sistólicas. Os

processadores podem ser interligados, formando redes toroidais, através dos oito barramentos disponíveis (4 de entrada e 4 de saída). A taxa de comunicação de dados em cada barramento é de 40 Mbytes/s, resultando uma taxa global de 320 Mbytes/s. O padrão IEEE 754 foi seguido na implementação das operações em ponto flutuante [IE85b]. O desempenho do microprocessador IWARP, considerando a execução paralela de operações nas 5 unidades funcionais disponíveis, atinge 100 Mops (Million Operations Per Second). Essas operações incluem transferências de dados de e para a memória e pelos barramentos de comunicação.

Modelos artificiais de Redes Neurais têm sido estudados por muitos anos como uma tentativa de atingir o desempenho apresentado pelo cérebro humano na solução de problemas tais como reconhecimento de imagens e fala. Esforços têm sido recentemente empreendidos motivados por novos algoritmos e topologias de redes, avanços na integração de circuitos analógicos em larga escala e por se acreditar que somente máquinas maciçamente paralelas podem resolver a contento os problemas mencionados [LI87].

Os elementos processadores utilizados em redes neurais são não lineares, tipicamente analógicos e lentos, se comparados com elementos digitais. Apesar disso, a exemplo do cérebro humano, o desempenho observado na solução de alguns problemas é surpreendente. Redes Neurais são compostas por muitos elementos processadores comunicando-se através de ligações ponderadas. Essas redes são caracterizadas pela sua topologia, pela função dos elementos processadores e por regras de aprendizado. As regras de aprendizado especificam um conjunto inicial de pesos para as ligações e ainda como esses pesos podem ser adaptados dinamicamente para melhorar o desempenho da rede.

2.2.4 Aspectos da Utilização de Máquinas Paralelas

O principal objetivo da utilização de máquinas paralelas reside na redução do tempo de processamento. Enquanto a maioria dos problemas pode se beneficiar disto, alguns podem ter sua utilidade aumentada e outros podem se tornar exequíveis [BE89] [CO91].

Diversos aspectos devem ser considerados para a utilização de uma máquina paralela na solução de um determinado problema. O primeiro desses aspectos envolve o

nível de paralelismo apresentado pelo problema. Amdhal considera o tempo de processamento dividido em duas componentes principais, uma relativa à execução da parte paralelizável do problema e outra relativa à execução da parte sequencial.. Mesmo que o tempo relativo à parte paralelizável possa ser reduzido a zero, utilizando-se infinitos processadores, o tempo global de execução não pode ser menor que o dispendido na parte sequencial [AL89]. O nível de paralelismo intrínseco da parte paralelizável do problema indica quantos processadores podem ser efetivamente utilizados na redução do tempo de processamento. Uma medida dessa eficiência é denominada "speedup" e relaciona o tempo de execução utilizando-se vários processadores com o tempo gasto utilizando-se um único processador. Na condição ideal o "speedup" cresce linearmente com o aumento do número de processadores utilizados. Isso geralmente não ocorre, o que limita o número máximo de processadores que podem ser efetivamente utilizados. Algumas aplicações não utilizam eficientemente mais que alguns processadores. Outras, algumas dezenas. Algumas entretanto, podem utilizar milhares de processadores.

O nível de paralelismo está relacionado com a granularidade das tarefas componentes do problema. O conceito de granularidade indica o tamanho das tarefas executadas em paralelo. Um mesmo problema pode ser dividido em tarefas com diferentes tamanhos. Diz-se que o paralelismo pode ser explorado em diferentes níveis de granularidade. Hwang, [HW87], sugere 5 níveis de granularidade para exploração de paralelismo:

- > Processos independentes e programas
- > Processos e partes de programa
- > Rotinas, sub-rotinas e co-rotinas
- > Iterações
- > Instruções

O primeiro nível está relacionado com multiprogramação, e não atinge o principal objetivo do processamento paralelo, ou seja, a redução do tempo de execução de um único programa. O paralelismo explorado dentro de uma instrução, através de múltiplas

unidades funcionais, é desconsiderado por ser extensivamente utilizado e não representar um avanço digno de nota em relação a máquina seqüencial de von Neumann.

Quanto menor é a granularidade, maior é o nível de paralelismo que pode ser explorado. A necessidade de comunicação entre as tarefas executadas em paralelo geralmente cresce com a diminuição da granularidade. É importante que o tempo dedicado em comunicação seja pequeno se comparado com o tempo de execução das tarefas, sob pena de uma baixa eficiência.

Em [FU91] discute-se granularidade sob o ponto de vista dos recursos de hardware e software atualmente disponíveis para processamento paralelo. Sugere-se, como mais promissora, a utilização de sistemas com algumas centenas de processadores explorando paralelismo com granularidade a nível de rotinas ou instruções.

Os aspectos até aqui discutidos levam em consideração o modelo computacional, a arquitetura e a linguagem utilizados na implementação do problema. O modelo computacional é mais geral que a arquitetura de um sistema. Um modelo computacional pode abrigar diversas arquiteturas [AL89] [KU89]. O modelo computacional descreve o que o computador pode fazer, isto é, que ações primitivas podem ser realizadas, quando elas podem ser realizadas e como os dados são armazenados e recuperados.

Enquanto arquiteturas seqüenciais seguem estritamente o modelo computacional de von Neumann, muitos modelos computacionais existem para as máquinas paralelas. Na tabela 2.2 são apresentados 8 deles, todos pertencentes a um modelo mais geral, ou seja, o que descreve as máquinas MIMD.

Além das características físicas da máquina, o tempo gasto na solução de um determinado problema está intimamente relacionado com os algoritmos utilizados. Muito esforço foi dedicado na elaboração de algoritmos otimizados para a solução de problemas comuns em computação, tais como ordenação e busca. Esses algoritmos, na sua maioria, são eficientes se executados no modelo computacional de von Neumann. Nada se pode afirmar em relação aos outros modelos computacionais, o que abre uma grande área de pesquisa, ou seja, a elaboração de algoritmos eficientes para problemas clássicos em cada um dos muitos modelos computacionais existentes.

Cada problema pode ser mais facilmente ou eficientemente resolvido em uma máquina pertencente a um determinado modelo computacional. O grande sucesso do modelo computacional de von Neumann reside na sua universalidade [VA89]. Surge daí a questão: é possível definir uma máquina paralela de propósito geral? Ou seja, é possível definir padrões para as linguagens de programação e garantir transporte dos programas fonte entre diferentes máquinas? Para conseguir isso é necessário isolar os aspectos relativos ao software, daqueles relativos ao hardware. O projeto ESPRIT GP-MIMD [HE91] é voltado para esse tema. Os primeiros resultados são promissores. Se for concluído com sucesso, as máquinas paralelas deixarão de ser consideradas somente em aplicações especiais.

As linguagens de programação existentes para a codificação de programas paralelos podem ser classificadas em duas categorias principais: linguagens com mecanismos intrínsecos para programação concorrente e linguagens seqüenciais estendidas. A seleção de uma determinada linguagem deve considerar a aplicação e a existência de implementações eficientes no hardware utilizado [BA89a] [ST82].

Três aspectos fazem a distinção entre uma linguagem seqüencial e uma linguagem voltada para processamento concorrente [BA89a]:

- > existência de múltiplos processadores
- > cooperação entre os processadores
- > potencial para tolerância a falhas parciais do sistema

O primeiro aspecto está relacionado com a capacidade de carga e execução de partes do código entre os diferentes processadores. O segundo aspecto trata da comunicação e sincronização entre as partes do código carregadas. Sistemas paralelos têm uma capacidade intrínseca para tolerância a falhas, com degradação controlada. Linguagens paralelas podem dispor de recursos que explorem convenientemente essa característica.

A tabela 2.3 relaciona algumas linguagens para programação concorrente segundo os aspectos mencionados acima [BA89a].

ASPECTO	LINGUAGEM
PARALELISMO Forma de Expressão Processos Objetos Comandos Expressões Cláusulas Mapeamento Estático Dinâmico Migração	Ada, C concor., Linda, NIL Emerald, Smalltalk concor. Occam ParAlfl, FX-87 Prolog concorrente, PARLOG Occam, StarMod Prolog concorrente, ParAlfl Emerald
COMUNICAÇÃO Passagem de Mensagem Ponto a Ponto Rendezvous RPC Uma para Muitas Compartilhamento de dados Estruturas Distribuídas Var. Lóg. Compartilhadas Não Determinismo Comandos Seleccionados Claús. de Horn Cond.	CSP, Occam, NIL Ada, C concorrente DP, CLU concorrente, LYNX BSP, StarMod Linda, Orca Prolog concorrente, PARLOG CSP, Occam, Ada, C conc. Prolog concorrente, PARLOG
FALHAS PARCIAIS Detecção Transações Atômicas Transparência	Ada, SR Argus, Aelus, Avalon NIL

Tabela 2.3 Características Principais de Algumas Linguagens para Programação Concorrente.

A programação paralela não é uma atividade simples. Além dos problemas normalmente encontrados na programação seqüencial, existe uma série de outros problemas decorrentes da diversidade de modelos computacionais e da pouca disponibilidade de algoritmos e técnicas específicas para programação paralela. Muitos esforços têm sido feitos nesse sentido. A ausência de máquinas com propósito geral, no sentido definido por Hey em [HE91], torna, entretanto, esses esforços quase sempre não cooperativos.

Diversos paradigmas e metodologias para programação têm sido introduzidos, tais como os apresentados em [HE87] [CA89] [HE89a] [KU89] [PR88] [BR90a]. Máquinas MIMD com memória compartilhada normalmente dispõem de compiladores e outras ferramentas para detecção de paralelismo [CH90] [BR90a], simplificando o trabalho de programação.

A depuração de programas paralelos é uma tarefa mais complexa que a depuração de programas seqüenciais. Os principais problemas encontrados são o aumento de complexidade, o não determinismo, a mudança de comportamento provocada pela inserção de comandos para depuração e a ausência de sincronismo entre os processadores. As técnicas de depuração de programas seqüenciais podem ser aplicadas com algum sucesso. McDowell [MC89] discute algumas técnicas para depuração de programas paralelos.

2.3 Transputers e Linguagem Occam

O Transputer surgiu de um esforço conjunto entre indústrias e universidades Européias para o desenvolvimento de uma arquitetura com desempenho elevado [AS86] [AS87] [HE90a] [WH90]. Esse esforço está concentrado no projeto ESPRIT ("European Strategic Program of Research and Development in Information Technology") [AM90] [AN90]. A primeira fase desse projeto, denominada ESPRIT I, iniciou-se em 1984 e terminou em 1989. Entre os diversos produtos dessa primeira fase do projeto merece destaque o Transputer T800. A segunda fase do projeto, ESPRIT II, iniciou-se em 1988 e está em andamento. Uma nova série de dispositivos, incluindo um novo Transputer (T9000), está em vias de ser lançada comercialmente. O T9000 multiplica por 10 o desempenho global apresentado pelo Transputer T800 [IN91].

O Transputer foi desenvolvido para dar suporte ao mecanismo de comunicação entre processos da linguagem CSP, e conseqüentemente da linguagem Occam. Existem compiladores de diversas linguagens que geram código para processadores Transputer, destacando-se Occam, C, Pascal, Fortran, Ada e Prolog [IN89c] [IN89d] [IN89e] [LO89a] [LO89b].

2.3.1 Transputers

Transputers podem ser considerados microcomputadores integrados em uma pastilha de silício. Eles apresentam todos os elementos básicos de um computador, ou seja, unidade de controle, unidade de processamento, memória e comunicação de E/S.

Os Transputers comercialmente disponíveis até 1991 estão relacionados na tabela 2.4, juntamente com suas principais características [IN89a]:

TIPO	Pal. bits	Clk nS	Mips pico	FPU (1)	Mfps pico	Ram KB	Can. n-Kb/s
T805 (2)	32	33	30	X	4.3	4	4-20
T801 (3)	32	33	30	X	4.3	4	4-20
T800 (4)	32	33	30	X	4.3	4	4-20
T425 (5)	32	33	30	-	-	4	4-20
T414 (5)	32	50	20	-	-	2	4-20
T222 (6)	16	50	20	-	-	4	4-20
T225 (6)	16	33	30	-	-	4	4-20

Observações:

- 1 - De acordo com a especificação IEEE 754
- 2 - Barramentos externos de dados e endereços multiplexados. Suporte para memória dinâmica.
- 3 - Barramentos externos de dados e endereços independentes. Nenhum suporte para memória dinâmica.
- 4 - Barramentos externos de dados e endereços multiplexados. Suporte para memória dinâmica.
- 5 - T425 substitui o T414 em novos projetos
- 6 - T225 substitui o T222 em novos projetos

Tabela 2.4 Principais Características dos Tranputers Disponíveis Comercialmente

2.3.2 O Transputer T800

Um diagrama de blocos detalhado do Transputer T800 é apresentado na figura 2.2 [IN90] [HO87]. Os seguintes subsistemas merecem destaque:

- > Unidade de Ponto Flutuante (FPU) - Atende à especificação IEEE 754/1985.
- > Unidade Central de Processamento (CPU) - Unidade de processamento de inteiros com 32 bits. Opera em paralelo com a FPU.
- > Canais (Links) - Lógica de controle dos canais.
- > Memória (4 Kbyte RAM) - RAM interna de alta velocidade (um ciclo do processador). Está mapeada no início do espaço de endereçamento do processador.
- > Interface de Memória Externa - Lógica programável para geração de sinais de interface com dispositivos de memória.

Deve ser observado que a comunicação entre dois Transputers é possível com a interligação da entrada e a saída de um canal do Transputer 1, respectivamente com a saída e a entrada de um canal do Transputer 2. A comunicação é bidirecional e os canais transmitem informação de dados e reconhecimento. O protocolo é orientado a bytes. O nível lógico em repouso dos canais é "0". Cada byte transmitido consiste de um bit de partida "1" seguido de outro bit "1", oito bits de dados, iniciando pelo bit menos significativo e um bit de parada "0", totalizando 11 bits por byte de dado. O transmissor aguarda um sinal de reconhecimento após a transmissão de cada byte. O sinal de reconhecimento significa que o receptor recebeu o byte e está pronto para receber outro byte. Esse sinal é constituído por um bit "1" seguido por um bit "0", sendo enviado tão logo o receptor aceite o byte transmitido. Como isso geralmente ocorre antes do término da transmissão do byte, o fluxo de dados unidirecional não sofre interrupções. Esse mecanismo é mapeado nas linguagens de programação disponíveis. O reconhecimento de um byte implica que ele foi atribuído a uma variável dentro de um processo em execução. Cada canal físico dá suporte a dois canais lógicos unidirecionais com sentidos opostos. A figura 2.3 ilustra o mecanismo de comunicação descrito.

2.2 Diagrama de Blocos Detalhado - T800

Interface de memória externa é configurável e permite a utilização de diferentes disposições sem necessidade de lógica externa complexa. A configuração da interface pode ser feita a partir de dados armazenados no espaço de endereçamento de memória do processador ou pela seleção de uma das configurações pré-definidas.

Este novo processador Transputer, (T9000) [IN91], apresenta uma série de soluções inovadoras e melhor desempenho em relação aos demais membros da família. As principais características do novo processador são:

- > Arquitetura super-escalar "pipeline" (32 bits)
- > Clock de "clock" processador 20 nS
- > 200 Mflops, 25 Mflops (pico)
- > Cache de instruções e dados com 16 Kbytes
- > 4 canais de comunicação bidirecionais 100 Mbits/s
- > Processador de comunicação dedicado. Disponibilidade de canais virtuais para comunicação entre processadores.
- > 16 canais de comunicação dedicados para iniciação, carga e controle. Esses canais podem ser ligados em cascata em sistemas com múltiplos T9000.

2.3.3 Linguagem Occam

A linguagem Occam foi desenvolvida tendo como base a linguagem CSP ("Communicating Sequential Processes") [HO78] e a linguagem EPL ("Experimental Programming Language") [IN88]. É uma linguagem de propósito geral com mecanismo específico para programação de sistemas concorrentes e distribuídos. A primeira versão dessa linguagem é denominada Occam 1. Uma segunda versão, sem algumas das limitações originais [PO89] está atualmente disponível [IN88] [BR89] [BU88] [JO88] [PO87] [WE89].

Algumas implementações da linguagem Occam para processadores não pertencentes à família Transputer são encontradas na literatura, tais como [IN85] e [FI86]. A arquitetura dos processadores Transputer foi projetada para dar suporte eficiente aos mecanismos de comunicação da linguagem Occam. Também foi previsto um conjunto de instruções direcionado para a implementação de compiladores. Esses fatores resultam em um desempenho comparável àquele obtido por programas escritos em linguagem montadora [BU88].

Um sistema programado na linguagem Occam pode ser descrito como uma coleção de processos sequenciais, executados concorrentemente, que se comunicam através de canais unidirecionais [HE89b]. A figura 2.4 ilustra essa situação. Como a comunicação não se processa através de variáveis, a linguagem Occam não é adequada para sistemas cujo mecanismo de comunicação baseia-se no compartilhamento de memória. Os processos podem ser executados em paralelo, em diferentes processadores, ou concorrentemente em um único processador. Os canais são implementados como dispositivos de E/S para comunicação entre processos executados em diferentes processadores Transputer. No caso de processos executados concorrentemente em tempo compartilhado dentro de um mesmo processador eles são mapeados como variáveis na memória interna.

Existem 5 processos básicos em Occam:

- > **v := e** Atribui à variável "v" o valor da expressão "e".
- > **c ! e** Transmite o valor da expressão "e" pelo canal "c".
- > **c ? v** Recebe um valor pelo canal "c" e atribui à variável "v".
- > **SKIP** Um processo que tem início, não faz nada e termina.
- > **STOP** Um processo que tem início, não faz nada e nunca termina.

2.3 Protocolo de Comunicação - Canais INMOS

2.4 Processos e Comunicação Entre Processos na Linguagem OCCAM

Os processos básicos podem ser combinados dando origem a outros processos mais complexos utilizando-se as seguintes estruturas:

- > SEQ - Sequência. Os processos componentes são executados sequencialmente:

```
SEQ
  entrada ? novo_dado
  resolver_problema (novo_dado, resposta)
  resultado ! resposta
```

- > PAR - Paralelo. Os processos componentes são executados concorrentemente. Exemplo:

```
PAR
  editar (teclado, tela)
  ler (teclado)
  SEQ
    tela ? ch
    screen ! ch
```

- > ALT - Alternativa. A execução de uma série de processos alternativos é condicionada por uma expressão booleana e/ou uma entrada. O primeiro processo apto é executado. Exemplo:

```
ALT
  porta & entrada ? dado
    saída ! dado
  porta & SKIP
    saída ! nenhum_dado
```

- > IF - Condicional. Combina vários processos, cada um condicionado por uma expressão lógica. O processo associado com a primeira expressão lógica avaliada verdadeira é executado. Se nenhuma expressão for verdadeira, a estrutura condicional comporta-se como o processo STOP:

```
IF
  x < y
    x := x + 1
  x >= y
    SKIP
```

- > CASE - Seleção. Combina opções condicionadas por um seletor. Caso o valor assumido pelo seletor exista, o processo associado é executado. Caso contrário a

seleção se comporta como o processo STOP. Por exemplo:

```
CASE nível_sonoro
  aumentar
    volume := volume + 1
  diminuir
    volume := volume - 1
ELSE
  SKIP
```

- > WHILE - Iteração. Um processo é executado iterativamente enquanto a expressão lógica associada for verdadeira. Exemplo:

```
WHILE buffer <> eof
  SEQ
    entrada ? buffer
    saída ! buffer
```

Essas estruturas constituem processos que podem ser componentes de outras estruturas, dando origem aos programas. Deve-se observar que não existem símbolos indicando o fim de cada estrutura. Os comandos abrangidos pelas estruturas são indicados pela indentação relativa das linhas. A indentação deve ser feita em duas posições adicionais para cada novo nível.

As estruturas SEQ, PAR, ALT e IF podem ser replicadas, criando-se "cópias" dos processos abrangidos pela estrutura. O significado exato dessas cópias depende da estrutura replicada [IN88]. A título de exemplo, uma sequência replicada origina uma estrutura de iteração com número de repetições conhecido (comando DO na linguagem Fortran). O seguinte trecho de código realiza a soma de dois vetores com n elementos, com índices variando entre 0 e n-1:

```
SEQ i = 0 FOR n
  soma[i] := vetA[i] + vetB[i]
```

Os tipos de dados primitivos disponíveis na linguagem Occam 2 são os seguintes:

- > BOOL - Valores lógicos.
- > BYTE - Valores entre 0x00 e 0xff

- > INT - Inteiros com sinal representados em complemento de dois. Comprimento de palavra dependente do processador.
- > INT16 - Inteiros com sinal, representados em complemento de dois, com 2 bytes.
- > INT32 - Inteiros com sinal, representados em complemento de dois, com 4 bytes.
- > INT64 - Inteiros com sinal, representados em complemento de dois, com 8 bytes.
- > REAL32 - Reais, conforme IEEE 754, com 4 bytes.
- > REAL64 - Reais, conforme IEEE 754, com 8 bytes.

Um programa Occam atua sobre variáveis, canais e temporizadores. Uma variável pode ter um dos tipos primitivos de dados apresentados. Podem ser declarados vetores multidimensionais de variáveis desses tipos. Canais podem estar associados a protocolos que definem tipos de dados estruturados. Temporizadores são definidos pelo tipo **TIMER** e são utilizados em operações relacionadas com a passagem do tempo. Os valores associados com temporizadores são derivados do "clock" do sistema.

Os principais grupos de operadores disponíveis na linguagem são:

- > Aritméticos
- > Módulo
- > Bit a bit
- > Lógicos
- > Relacionais

Um programa Occam pode ser estruturado em procedimentos e funções. Um procedimento em Occam define um nome para um conjunto de processos. Funções

constituem processos com um valor de um tipo primitivo de dados associado ao seu nome. Esse nome pode ser utilizado em expressões.

2.3.4 Sistema de Desenvolvimento TDS

O sistema de desenvolvimento INMOS - TDS torna disponível um ambiente que integra diversas ferramentas para o desenvolvimento de programas Occam. É possível a edição, geração de código executável, execução e depuração de programas [GL87] [IN90].

Estão disponíveis versões do sistema de desenvolvimento TDS para diversos tipos de hospedeiros, incluindo VAX, SUN e IBM/PC. Essas máquinas executam um módulo do TDS, o ISERVER. A parte principal do TDS, incluindo todas as ferramentas de desenvolvimento, é executada em um microcomputador baseado em Transputers. A comunicação entre o computador hospedeiro e o Transputer é feita através de um canal físico. O hospedeiro deve dispor de uma interface adequada para isso. A figura 2.5 ilustra a configuração descrita.

O ISERVER permite a execução de programas no Transputer e pode permanecer ativo por tempo indeterminado, atuando como um servidor de impressão, arquivos e console para esses programas. As seguintes opções estão disponíveis na execução do ISERVER:

- > SA - Aplica o sinal ANALYSE e obtém 8 KB da memória do Transputer. Esses dados podem ser posteriormente utilizados pelo depurador.
- > SB <nome_arquivo> - Carrega um programa. Equivale às opções SR, SS, SI e SC combinadas.
- > SC <nome_arquivo> - Transmite um arquivo pelo canal.
- > SE - Termina a execução do ISERVER se o sinal ERROR do Transputer se tornar ativo.
- > SI - Mostra mensagens indicando os progressos na carga de programas.

- 1 Microcomputador Transputer**
- 2 Interface**

2.5 Configuração Típica para Execução do TDS

2.6 Configuração de Carga e Execução Programas EXE

- > SL <endereço_canal> - Especifica o endereço do canal.
- > SP <n> - Permite especificar o número de bytes obtidos na opção SA.
- > SR - Aplica o sinal de RESET no Transputer.
- > SS - Permanece ativo servindo o canal.

Para permitir a instalação do sistema de desenvolvimento TDS em diversos tipos de hospedeiros, a INMOS fornece o seu código fonte bem como instruções detalhadas da sua operação [IN90].

O editor de textos constitui a principal interface do sistema TDS com o usuário. Além das funções de edição, existem comandos para acesso às ferramentas de desenvolvimento. O editor opera no modo tela plena ("full screen"). O conceito de dobras ("folds"), é introduzido, permitindo a edição de documentos hierarquicamente estruturados. Isso favorece a visualização dos documentos produzidos, especialmente programas fonte, e encoraja o desenvolvimento de programas a partir dos níveis hierarquicamente mais elevados ("top down"). O conceito envolvido pode ser entendido como a possibilidade de esconder partes do texto em dobras, da mesma forma como é possível fazer com textos impressos em folhas de papel. Existem comandos para entrar numa dobra, abrir uma dobra, criar e destruir dobras. Uma dobra fechada é representada por uma linha e está sujeita aos comandos de edição aplicáveis às demais linhas. O seguinte trecho de código serve como exemplo:

```
... Declarações
SEQ
  ... Iniciações
  WHILE buffer <> eof
    ... Copiar caracter
```

As linhas iniciadas por "..." representam dobras cujo conteúdo é explicitado pelos comentários subsequentes. Observe que cada dobra tem um nível de indentação associado. Abrindo-se a dobra "Copiar caracter" a tela de edição toma a aparência:

```
... Declarações
SEQ
  ... Iniciações
  WHILE buffer <> eof
    {{{ Copiar character
    SEQ
      entrada ? buffer
      saída ! buffer
    }}}
  }}}

```

Entrando-se na dobra mencionada a partir da tela inicial, obtém-se:

```
      {{{ Copiar character
      SEQ
        entrada ? buffer
        saída ! buffer
      }}}

```

É possível armazenar o conteúdo de uma dobra em um arquivo separado. Nesse caso a dobra é denominada arquivada ("filed fold"). O arquivo é atualizado automaticamente quando a dobra é fechada.

Para compilar um trecho de código é necessário colocá-lo dentro de uma dobra especial, definindo-se uma unidade de compilação. Os seguintes tipos de unidades de compilação são possíveis:

- > EXE - Programa executável em um Transputer dentro do sistema de desenvolvimento TDS.
- > UTIL - Programa executável como um utilitário do TDS. A interface para a criação desses utilitários não está especificada na versão IMS700E do TDS [IN90].
- > PROGRAM - Programa executável em um ou mais Transputers sob supervisão do TDS. Um PROGRAM é configurado e carregado pelo TDS. O Transputer raiz da rede deve executar um programa do tipo EXE para monitorar a execução do PROGRAM.
- > SC - Unidade de compilação em separado. Uma SC deve conter uma ou mais declarações de procedimentos ou funções.

- > **LIB** - Unidade de compilação de biblioteca. Permite o compartilhamento de código entre partes de um programa ou entre diferentes programas.

É possível o desenvolvimento de programas que operam independentemente do sistema de desenvolvimento TDS. Duas formas existem para isso:

- > **PSP** "Parameterless Standalone Program"
- > **SHP** "Standard Host Procedure"

Uma vez carregados em uma rede de Transputers e em execução, esses programas podem se comunicar com o **ISERVER** e utilizar os serviços disponíveis de console, impressão e arquivos. Para isso, um canal de comunicação com o **ISERVER** deve ser convenientemente alocado. As duas formas de programas originam código binário, sendo possível transportar esse código sem alterações para diferentes tipos de computador hospedeiro.

Programas **PSP** devem ser codificados como "**PROGRAMs**". O carregamento e execução desses programas pode ser feito utilizando-se o **ISERVER**.

Programas **SHP** devem ser codificados como **SCs**. Após a compilação, deve ser adicionado código específico para possibilitar o seu carregamento pelo módulo **ISERVER**. A possibilidade de carregamento por qualquer um dos canais físicos disponíveis no Transputer raiz da rede e posterior comunicação através desse mesmo canal, é considerada uma vantagem de programas **SHP** sobre programas **PSP**. O sistema de desenvolvimento **TDS** é codificado como um programa **SHP**.

As figuras 2.6, 2.7 e 2.8 ilustram, respectivamente, a configuração no carregamento e execução de programas **EXE**, **PROGRAM**, **PSP** e **SHP**.

O **TDS** dispõe de utilitários para a depuração de programas executados em redes de Transputers. É possível a análise do código fonte **Occam**, ou em linguagem de máquina, após uma interrupção do programa motivada por uma condição de erro. Podem ser exibidos os valores de variáveis e canais em qualquer Transputer pertencente à rede.

2.7 Configuração de Carga e Execução Programas PROGRAM

2.8 Configuração de Carga e Execução Programas PSP/SHP

O processo de depuração se inicia com a ocorrência do sinal de erro. Parte do conteúdo da memória do Transputer raiz da rede é armazenado em disco para posterior análise. O TDS e os utilitários para depuração são carregados no Transputer raiz.

É possível a utilização dos utilitários na depuração de programas codificados em qualquer uma das formas mencionadas neste item.

2.4 Sistemas com Transputers

Existem diversos sistemas comerciais empregando Transputers. Esses sistemas normalmente são constituídos por placas contendo processadores Transputers, interfaces com computadores hospedeiros e software para desenvolvimento. O sistema de desenvolvimento TDS e a placa INMOS B004 constitui um exemplo típico [IN86]. Outros sistemas apresentam recursos adicionais [IN89f]. A disponibilidade de configuração por software da topologia da rede de processadores e operação em ambiente de rede local constituem exemplos dos recursos disponíveis.

2.4.1 Projeto ESPRIT P1085

O projeto ESPRIT P1085 teve como principal objetivo o desenvolvimento de uma arquitetura de propósito geral com elevado desempenho, baseada no Transputer T800, mantendo o modelo de programação intrínseco da linguagem Occam e proporcionando capacidade de configuração por software da topologia da rede de processadores.

A arquitetura desenvolvida consiste basicamente em um sistema com múltiplos Transputers conectados por uma rede de chaveamento e um barramento de controle. A arquitetura é estruturada na forma de uma árvore, conforme pode ser observado nas figuras 2.9 e 2.10. Em um primeiro nível os processadores são agrupados constituindo um "supernode". Um sistema com desempenho de um supercomputador pode ser constituído pela associação de vários "supernodes". Essa organização possibilita a expansão modular do sistema e permite conectividade total entre os processadores. As chaves utilizadas foram desenvolvidas especificamente para o projeto.

2.9 "Supernode" P1085

2.10 Supercomputador Composto por "Supernodes"

O barramento de controle permite o controle da rede de processadores e um canal de comunicação entre eles. A reconfiguração dinâmica da rede é possível com a utilização desse barramento.

O sistema comunica-se com os usuários através de um computador hospedeiro. A operação multiusuária do sistema é prevista pela utilização de computadores hospedeiros multiusuários.

A arquitetura definida no projeto ESPRIT P1085 foi implementada comercialmente pela PARSYS, Reino Unido e TELMAT, França.

2.4.3 Sistema Computacional Meiko

O sistema computacional MEIKO pode ser considerado como um conjunto de módulos de software e hardware que podem ser utilizados na implementação de máquinas paralelas baseadas na família de processadores Transputer [CL90]. Nesse sentido, não existe uma arquitetura pré-definida.

Os módulos implementam funções gerais e podem ser conectados em bastidores padrão. A quantidade e o tipo dos módulos disponíveis, a interligação entre eles e o tipo de computador hospedeiro determinam as características do sistema.

Os seguintes tipos de módulos, entre outros, estão disponíveis [ME90]:

- > Processadores. Vários tipos de módulos, com 1 a 4 processadores e 256 Kbytes a 48 Mbytes de memória por processador.
- > Interface com hospedeiro.
- > Interface com hospedeiro tipo IBM/PC.
- > Hospedeiro independente com interfaces SCSI e Ethernet.
- > Interface para hospedeiro VME (Sun 3).

- > Armazenamento de massa (Mbytes RAM e interface SCSI)
- > Interfaces de vídeo 1024x768 ou 1280x1024/4Mbytes.
- > Interface para captação de imagens.
- > Interfaces seriais RS232.
- > Placa para protótipos. Utilizada para o desenvolvimento de protótipos de hardware.

2.5 Sistemas Distribuídos

O hardware e o software dos sistemas computacionais têm sofrido constante evolução. O hardware atravessou quatro gerações que foram caracterizadas pela tecnologia dos componentes utilizados na sua implementação. Inicialmente eletromecânicos, surgiram posteriormente as válvulas, transistores e finalmente os circuitos integrados. A evolução do hardware tem se caracterizado pelo aumento da velocidade de processamento e pela redução do espaço físico e consumo de energia. O software operacional evoluiu dos carregadores de código em linguagem de máquina, sistemas operacionais monousuários, sistemas operacionais multiusuários em tempo compartilhado até os sistemas operacionais distribuídos. A feliz combinação do hardware, mais acessível e de maior desempenho, com os sistemas operacionais distribuídos e as redes de computadores, introduz aos usuários recursos computacionais nunca antes proporcionados pelos sistemas até então existentes.

Existem na literatura diversas definições conflitantes para sistemas distribuídos. O único ponto em comum nessas definições é a existência de múltiplos processadores. Neste trabalho entende-se por sistemas distribuídos aqueles onde vários computadores autônomos, geograficamente isolados, podem operar cooperativamente trocando mensagens através de uma rede de comunicações [CO88] [MU89]. Em [BA89a] esses sistemas são denominados Sistemas Distribuídos Fracamente Acoplados. Essa definição exclui sistemas com memória compartilhada e aqueles onde a distribuição é geograficamente restrita às placas dentro de um bastidor.

2.5.1 Redes de Computadores

Redes de computadores podem ser consideradas sistemas distribuídos, segundo a definição apresentada em 2.5. Tanenbaum, em [TA88], faz uma distinção adicional, considerando o software básico do sistema sob o ponto de vista do usuário. Desta forma, uma rede de computadores é considerada um sistema distribuído se esse software ocultar do usuário a existência de múltiplos processadores, isto é, o usuário utiliza o sistema como se utilizasse um único processador.

Os objetivos básicos de uma rede de computadores são:

- > Compartilhamento de recursos.
- > Confiabilidade provida pela multiplicidade de recursos.
- > Economia proporcionada pela utilização de muitas estações de trabalho, ao invés de um computador centralizado, e pelo compartilhamento de recursos através de servidores especializados.
- > Modularidade, ou seja, o aumento gradual na capacidade do sistema. Vale observar que o acréscimo de uma estação de trabalho aumenta a capacidade computacional de uma rede de computadores, o que não ocorre com o acréscimo de um terminal a um sistema multiusuário por tempo compartilhado.
- > Meio de comunicação entre usuários geograficamente separados.

As redes de computadores podem ser classificadas, quanto à dimensão da área coberta, em redes locais de computadores (LAN - "Local Area Network") e redes de computadores de longa distância (WAN - "Wide Area Network"). LANs cobrem distâncias que tipicamente variam entre dezenas de metros até alguns quilômetros. WANs cobrem distâncias entre dezenas de quilômetros e alguns milhares de quilômetros. As LANs caracterizam-se pela maior taxa de transferência de dados pelo meio físico e confiabilidade elevada [ST84] [CL78] [HO86]. Entre LANs e WANs tem sido mencionada na literatura uma outra classe de redes, MAN ("Metropolitan Area Networks"). MANs se caracterizam por utilizar a tecnologia das LANs, cobrindo porém a extensão de toda uma cidade [TA88].

Com o objetivo de reduzir sua complexidade, o software operacional das redes de computadores é comumente estruturado em camadas sobrepostas. O propósito dessas camadas é o de oferecer serviços às camadas superiores, escondendo os detalhes de como esses serviços são implementados. Na comunicação entre diferentes máquinas, tudo se passa como se as camadas correspondentes trocassem informação diretamente. O conjunto das regras que regem essa troca de informações constitui um protocolo. Existe, conseqüentemente, um protocolo associado com cada camada.

O primeiro passo dado no sentido da padronização dos protocolos utilizados, para comunicação entre camadas de redes locais heterogêneas, é representado pelo modelo de referência OSI ("Open Systems Interconnection"), patrocinado pela ISO ("International Standards Organization") [ZI80].

O modelo OSI possui sete camadas, descritas a seguir em ordem decrescente de nível:

> APLICAÇÃO:

Nessa camada vários protocolos estão envolvidos, dependendo da aplicação. A principal função dessa camada é dar suporte às aplicações, oferecendo um mecanismo de interface com o protocolo da camada de aplicação. Exemplos de aplicações incluem terminais de vídeo virtuais, transferência de arquivos e correio eletrônico.

> APRESENTAÇÃO:

A camada de apresentação tem como principal objetivo a conversão de dados entre a representação padrão utilizada na rede e aquela utilizada no sistema conectado. Essa camada cuida de serviços utilizados frequentemente, para os quais é interessante que sejam adotadas soluções gerais.

> SESSÃO:

A camada de sessão permite que usuários em diferentes máquinas possam estabelecer sessões. Seu efeito é aumentar o nível dos serviços disponíveis na camada de transporte, e suas atribuições incluem o controle de tráfego e sincronização. A camada de sessão permite que o usuário possa ter acesso a um

sistema remoto de tempo compartilhado ou transferir arquivos entre duas máquinas.

> TRANSPORTE:

A função básica da camada de transporte é receber dados da camada de sessão, dividir esses dados em pacotes menores, se necessário, e enviá-los para a camada de rede. Nesse processo a camada de transporte deve isolar a camada de sessão de eventuais mudanças no hardware. Os pacotes recebidos da camada de rede são recompostos e passados para a camada de sessão. A camada de transporte deve garantir a consistência dos dados transmitidos e recebidos. Também cabe à camada de transporte o estabelecimento de múltiplos canais virtuais, no caso de sistemas multiusuário.

> REDE:

Esta camada cuida do roteamento de pacotes entre fonte e destino, geralmente entre redes heterogêneas. Esse roteamento pode ser estático ou dinâmico. Em redes do tipo "broadcast", o problema de roteamento é simples e a camada de rede pode ser fina ou não existir.

> DADOS:

A camada de dados é responsável pela transmissão, livre de erros, dos pacotes de dados provenientes da camada de rede. Cabe a esta camada o controle dos pacotes duplicados ou perdidos.

> FÍSICA:

O meio físico da rede e o mecanismo de transmissão de bits são aspectos tratados nessa camada.

A camada de dados é subdividida em duas subcamadas:

> LLC - "Logical Link Control":

> MAC - "Medium Access Control":

Uma série de padrões ISO está disponível, especificando protocolos e serviços para cada uma das camadas apresentadas [TA88].

2.5.2 Rede Local Ethernet

A rede local padrão Ethernet começou a ser definida em 1972 [ME76], no centro de desenvolvimento da Xerox em Palo Alto. Em 1980, um esforço conjunto da Intel, Dec e Xerox resultou na especificação conhecida hoje como Blue Book Ethernet, ou padrão DEC/INTEL/XEROX [SH82] [FT89]. Em 1985 essa especificação foi transformada, com algumas alterações, no padrão IEEE 802.3 [IE85a], embora o padrão DEC/INTEL/XEROX continue sendo o mais utilizado. A estrutura dos pacotes de dados definidos nas especificações acima está apresentada na figura 2.11. Deve-se observar que a única diferença reside na interpretação dos bytes 21 e 22. Na especificação inicial esse campo é interpretado como o tipo do pacote, e pode ser utilizado para a implementação de mecanismos de multiplexação [FT89]. Na especificação mais recente, esse campo é interpretado como o tamanho do campo de dados.

As principais características da rede Ethernet são:

- > Taxa de transmissão: 10 Mbits/s
- > Topologia: barramento
- > Controle de acesso ao meio de transmissão: CSMA-CD ("Carrier Sense Multiple Access - Collision Detection")

2.5.3 Modelo Cliente-Servidor e Mecanismo RPC

O modelo cliente-servidor representa uma alternativa natural para a implementação da comunicação em sistemas computacionais distribuídos, particularmente aqueles baseados em redes locais de computadores [TA85]. Nesse modelo, serviços são requisitados por processos cliente para processos servidores. Uma vez requisitado um

2.11 Pacote de Dados Ethernet

serviço, o processo cliente aguarda uma resposta. Vários mecanismos podem ser adotados para a implementação desse modelo.

O mecanismo de Chamadas a Procedimentos Remotos, RPC tem se destacado como um mecanismo simples e transparente para a implementação de sistemas distribuídos. O paradigma de chamadas a procedimentos locais, implementado na maioria das linguagens de programação, dá suporte teórico ao mecanismo RPC. A transparência ideal acontece quando os procedimentos remotos não podem ser distinguidos dos procedimentos locais. Alguns aspectos, como o acesso a parâmetros passados por referência, limitam o nível de transparência conseguido [WI87].

A operação do mecanismo RPC reflete o modelo cliente-servidor. O cliente faz as requisições através de um procedimento de interface ("stub" do cliente) que tem como função isolar a visão do usuário dos procedimentos utilizados para a conversão, empacotamento e transmissão dos dados para o servidor. Uma vez feita a requisição, o procedimento de interface aguarda a resposta do servidor. No servidor a requisição é recebida, identificada e tratada por um procedimento de interface ("stub" do servidor), com função análoga ao procedimento de interface do cliente. Deve-se observar que existe apenas um fluxo de execução, embora todo o processo envolva duas ou mais máquinas.

Uma chamada a um procedimento remoto pode não ter sucesso, caso o cliente, o servidor ou o meio de comunicação entre eles venha a falhar. Falhas temporárias podem ser contornadas pela repetição dos procedimentos, normalmente associadas a mecanismos de temporização ("time out"). Quatro casos básicos são possíveis [WI87], conforme pode ser observado na figura 2.12.

A semântica de chamada determina a frequência com que a função remota deve ser executada. As quatro possibilidades descritas a seguir, se encontram em ordem crescente de rigidez:

- > Possivelmente uma execução - A requisição é feita assumindo-se que ela será atendida. Esse modo não atende aos requisitos do mecanismo RPC.

2.12 Tratamento de Falhas em RPCs

- > Pelo menos uma execução - Repete-se a requisição até que uma resposta seja recebida. É possível, no caso de perda da resposta, que a primitiva seja executada mais de uma vez. É importante que a primitiva tenha resposta e efeito idêntico, no caso de uma ou várias execuções.
- > No máximo uma execução - O servidor controla as requisições dos clientes de tal forma que uma primitiva não seja executada mais que uma vez. Uma requisição repetida, motivada pela perda da resposta, ocasiona a retransmissão da última resposta calculada.
- > Exatamente uma execução - Nesse caso garante-se a execução da primitiva, uma única vez. É importante dispor de informações armazenadas em dispositivos não voláteis e múltiplos recursos para contornar falhas no hardware.

Outros aspectos relacionados com a operação do mecanismo RPC são descritos em [WI87]. Entre esses aspectos, a operação com múltiplos servidores introduz versatilidade ao sistema. Nesse caso, mecanismos de ligação ("bind"), estáticos ou dinâmicos, devem ser implementados para permitir a seleção dos servidores. Em [BI84] é descrita com detalhes a implementação de um sistema RPC.

2.6 Considerações Finais

Neste capítulo procurou-se dar uma visão geral dos tópicos relacionados com o desenvolvimento do trabalho proposto.

Os modelos computacionais e arquiteturas para processamento paralelo foram revistos no sentido de situar o sistema servidor de processamento paralelo no contexto atual de desenvolvimento dessa classe de sistemas.

Uma introdução aos processadores Transputer, à linguagem Occam e ao sistema de desenvolvimento INMOS TDS foi apresentada, uma vez que o servidor está diretamente relacionado com esses itens. Essa introdução tem importância na medida em que ela é adequada para o entendimento dos capítulos seguintes.

O sistema proposto neste trabalho é fisicamente constituído por um servidor de um sistema distribuído baseado em uma rede local de computadores. Redes de computadores e sistemas distribuídos são revistos, com ênfase maior em redes locais de computadores. No capítulo seguinte são apresentadas a arquitetura e a definição do sistema servidor de processamento paralelo.

CAPÍTULO 3

ARQUITETURA E ESPECIFICAÇÃO DO SISTEMA

3.1 Considerações Iniciais

No capítulo anterior foram introduzidos os diversos tópicos que compõem o contexto dentro do qual está inserido este trabalho. Esses tópicos envolvem diversas áreas da informática, e a apresentação feita procurou, sem a pretensão de ser completa, ressaltar os aspectos mais importantes e relacionados com os objetivos propostos.

Neste capítulo o sistema servidor de processamento paralelo é especificado. São apresentados os diversos aspectos considerados na definição da sua arquitetura. As informações contidas no capítulo anterior são revistas, motivando e respaldando as definições e decisões apresentadas.

No item 3.2 são definidos os requisitos do sistema. Essa definição é baseada nas finalidades previstas para o sistema. São feitas algumas considerações para que o sistema possa ser utilizado de forma transparente aos detalhes da sua arquitetura, projeto e implementação. O ambiente de utilização do sistema é descrito no item 3.3, onde são discutidos os aspectos relevantes na visão do usuário.

Continuando, no item 3.4 apresenta-se a definição global do sistema. É discutida a interação do sistema com a rede local adotada. A estrutura básica do servidor de processamento paralelo é introduzida nesse item. A especificação do hardware do sistema é apresentada no item 3.5. No item 3.6 é apresentada a especificação do software do sistema, bem como a metodologia utilizada no seu desenvolvimento.

3.2 Requisitos do Sistema

Os objetivos e a motivação para o desenvolvimento deste trabalho foram discutidos no capítulo 1. Baseada neles, uma série de requisitos foi definida. Porém, as características do sistema e sua utilização, indicam que esses requisitos não podem ser considerados definitivos. Acredita-se que a utilização efetiva do sistema provocará revisões periódicas, possibilitando a evolução do sistema e o cumprimento da sua principal finalidade, a de possibilitar novas pesquisas nas áreas envolvidas.

3.2.1 Finalidade

O Servidor de Processamento Paralelo tem como pré-requisito atender duas finalidades principais:

- > Constituir um instrumento de pesquisa em sistemas distribuídos e arquiteturas paralelas.
- > Oferecer suporte de hardware e software para processamento paralelo.

Este trabalho representa uma contribuição inicial para as duas finalidades. Outros trabalhos correlatos deverão ser motivados pelo servidor. Esses trabalhos deverão explorar algum aspecto da sua arquitetura ou organização, seja pela evolução ou reconsideração de etapas do seu desenvolvimento, seja pela avaliação do seu desempenho ou adequação.

A segunda finalidade consiste em tornar disponível uma máquina paralela aos seus potenciais usuários. Considera-se necessário para essa finalidade que o servidor disponha de ferramentas adequadas para o desenvolvimento de programas paralelos, incluindo compiladores e depuradores.

3.2.2 Requisitos Básicos

Para cumprir as finalidades mencionadas, é importante que o servidor de processamento paralelo atenda aos seguintes requisitos:

- > Possibilidade de utilização do sistema de desenvolvimento INMOS TDS.
- > Possibilidade de utilização de compiladores para as linguagens C, Pascal e Fortran, entre outras.
- > Operação transparente aos detalhes de arquitetura e organização do sistema.
- > Conectividade total entre os processadores do servidor.

O sistema de desenvolvimento TDS proporciona uma série de recursos voltados para o desenvolvimento de programas na linguagem OCCAM. A linguagem OCCAM oferece recursos normalmente disponíveis em outras linguagens de alto e baixo níveis, além de recursos especializados para programação paralela. Exibe ainda desempenho comparável ao conseguido com a utilização de linguagens montadoras [BU88]. Na programação de novos sistemas essa linguagem representa uma alternativa importante. Por outro lado, a transposição de sistemas antigos programados em outras linguagens, na maioria das vezes de forma seqüencial, não é tarefa simples. Esses sistemas podem tirar algum proveito do servidor descrito utilizando o código original convenientemente adaptado e acrescido de algumas rotinas. Dessa forma, a exploração de algum paralelismo pode ser conseguida [HE89a]. A disponibilidade de compiladores para outras linguagens possibilita tal utilização do servidor. Isso permite também que programadores reticentes em adotar uma nova linguagem possam utilizar o servidor sem grande esforço adicional.

Programação paralela é uma atividade mais complexa que a programação seqüencial. Isso é um dos fatores que inibem a utilização de arquiteturas paralelas. A operação do servidor de forma transparente, quanto aos detalhes da sua arquitetura e organização, contribui no sentido de não introduzir dificuldades adicionais. Isso deve incentivar sua utilização por pesquisadores não familiarizados com a operação dessa classe de sistemas, interessados apenas em minimizar o tempo de processamento dos seus programas.

Foi considerada neste trabalho a conectividade total entre os processadores disponíveis no servidor. Essa conectividade é necessária para a definição dos subgrafos de processadores correspondentes aos lotes, além de permitir maior flexibilidade ao usuário na definição da topologia da rede de processadores adequada para a sua aplicação.

3.2.3 Requisitos Adicionais

Uma das principais motivações deste trabalho é o compartilhamento de recursos para processamento paralelo através da operação multiusuária do sistema. Para isso, optou-se pela utilização de uma rede local de computadores. Redes locais de computadores têm se mostrado eficazes no compartilhamento de recursos de alto custo entre usuários geograficamente separados por distâncias moderadas. Além disso, propõe-se o compartilhamento do meio físico da rede local entre diferentes sistemas. A figura 3.1 ilustra a situação descrita.

O compartilhamento do meio físico da rede local permite que diferentes classes de estações de trabalho e servidores possam se comunicar, e conseqüentemente fazer uso do servidor de processamento paralelo. Pontes, ("bridges"), podem ser utilizadas para minimizar o tráfego da rede entre segmentos distintos, permitindo uma utilização efetiva da faixa de passagem do meio em cada um dos segmentos [SA88].

A operação multiusuária em rede local de computadores impõe restrições na taxa máxima de transferência de dados entre os processadores do servidor e as estações de trabalho usuárias. Isso limita a utilidade do sistema para aplicações que tenham atividade de E/S intensa. Aplicações gráficas constituem um bom exemplo, onde a simples atualização de uma tela pode envolver a transferência de grandes quantidades de dados. O tempo de resposta dessa atualização pode ser determinante sobre a aplicabilidade do sistema, como no caso de animação gráfica. Para contornar esse problema, o servidor deve possibilitar a conexão de estações de trabalho locais (ETL). Essas estações utilizam canais de comunicação exclusivos, possibilitando taxas elevadas de transferência de dados com dispositivos de E/S. Esse tipo de configuração pode aumentar significativamente a utilidade do servidor na classe de aplicações mencionada. Esses canais também podem ser úteis na depuração de programas paralelos. Conforme sugerido em [LO89b], dispositivos de E/S inteligentes podem ser utilizados para analisar pontos específicos de uma rede

3.1 Rede Local com Segmentos Interligados por Pontes

onde está sendo executado um programa sob depuração. Esse procedimento materializa o conceito de pontas de prova de software, em analogia às utilizadas para análise de hardware.

3.3 Ambiente de Utilização

A partir das considerações feitas até o momento, o ambiente de utilização do sistema toma o aspecto mostrado na figura 3.2. Nessa figura pode-se observar o servidor de processamento paralelo sendo compartilhado em um ambiente multiusuário baseado em uma rede local de computadores. Estações de trabalho locais (ETL) aparecem diretamente acopladas ao servidor. As estações de trabalho podem utilizar o servidor de processamento paralelo através de diferentes programas utilitários, sistemas de desenvolvimento ou linguagens. Essas estações podem estar executando diferentes sistemas operacionais. Outros servidores podem estar presentes na rede local, cada qual vinculado a um ou mais sistemas operacionais que compartilham o mesmo meio físico de transmissão de dados.

O acesso aos recursos do servidor de processamento paralelo é feito através de um conjunto coeso de comandos e ferramentas que escondem os detalhes de implementação, mapeamento físico e existência de múltiplos recursos. O usuário deve ter a ilusão de estar utilizando todos os recursos como se eles estivessem acoplados direta e exclusivamente à sua estação de trabalho. O usuário pode requisitar ao servidor um determinado número de processadores, (lote de Transputers), especificando a topologia da conexão entre eles. O servidor analisa a solicitação baseando-se nos recursos ainda disponíveis, atendendo-a ou não em função dessa análise, permitindo ainda que o usuário permaneça numa fila aguardando a liberação dos recursos solicitados.

O usuário enxerga o servidor de processamento paralelo sob o ponto de vista da ferramenta que está utilizando. Entre essas ferramentas figuram compiladores, carregadores e sistemas de desenvolvimento, tal como o TDS. A grande maioria das ferramentas disponíveis para sistemas com Transputers, incluindo o sistema de desenvolvimento TDS, faz uso de um computador hospedeiro acoplado à rede de Transputers. O computador hospedeiro executa parte dos programas e oferece à rede os serviços de terminal de vídeo, arquivos e impressão. No capítulo 2 podem ser encontrados maiores detalhes a respeito. Esse modo de operação foi mantido para atender os aspectos

3.2 Visão Geral do Sistema

3.3 Servidor de Processamento Paralelo

de transparência desejados. É importante que o projeto de hardware e software do servidor minimize as alterações necessárias no código das ferramentas utilizadas. Isso é fundamental para a utilização de ferramentas de programação comerciais, uma vez que normalmente essas não vêm acompanhadas dos respectivos programas fonte.

3.4 Definição Geral do Sistema

Na definição do sistema, dois aspectos de caráter geral são importantes: a seleção da rede local de computadores e a estrutura básica do servidor. Esses aspectos são considerados neste item..

3.4.1 A Rede Local de Computadores

Devido à evolução ininterrupta dos meios físicos, protocolos de comunicação e sistemas operacionais disponíveis para a implementação das redes locais de computadores, optou-se por tornar o servidor de processamento paralelo o mais independente possível desses fatores. Isso facilitará sua migração para outros ambientes quando isso se tornar atrativo ou necessário. Apesar disso, foi feita uma seleção inicial criteriosa desses itens visando a fácil integração do servidor com os sistemas distribuídos existentes.

O primeiro aspecto considerado na seleção da rede local de computadores foi o meio físico de transmissão de dados. Dentre as diversas possibilidades existentes, optou-se pela utilização do padrão Ethernet operando a 10 Mb/s. Isso justifica-se pela difusão que o padrão Ethernet tem, principalmente entre fabricantes de equipamentos largamente utilizados em processamento científico. Entre esses equipamentos figuram os computadores Digital e as estações de trabalho produzidas pela Sun Microsystems. A adoção do padrão Ethernet não inviabiliza a conexão com equipamentos dotados de acopladores para outros padrões. Existe sempre a possibilidade da conexão de redes locais de padrões diferentes através de dispositivos adequados ("gateways").

Existem comercialmente disponíveis diversos sistemas operacionais e utilitários de comunicação para redes locais de computadores. Esses sistemas e utilitários apresentam características similares, porém são normalmente incompatíveis entre si (figura 3.1). Para

não tornar o servidor de processamento paralelo dependente de um deles, optou-se pela utilização concorrente do meio físico e pelo desenvolvimento das partes críticas do software básico de comunicação. Esse procedimento se justifica se forem considerados os seguintes aspectos:

- > O servidor pode ser utilizado a partir de qualquer uma das estações de trabalho da rede local. Basta para isso que sejam feitas algumas alterações no software de comunicação desenvolvido.
- > A integração com os sistemas operacionais e utilitários de comunicação disponíveis pode ser feita posteriormente, onde for interessante ou necessário, sem as dificuldades que certamente surgiriam da integração de dois sistemas fechados.
- > O desempenho do sistema pode ser otimizado. Primeiro pelo desenvolvimento de um software direcionado para a aplicação. Segundo, pela possibilidade de manutenções evolutivas no software desenvolvido.
- > O sistema resultante é aberto ao desenvolvimento de novas pesquisas.

3.4.2 Estrutura Básica do Servidor

O servidor de processamento paralelo está ligado à rede local através do Processador de Entrada (PE). Esse computador dispõe de uma interface de comunicação com a rede e é responsável pela transferência de pacotes entre a rede local e o Banco de Transputers (BT), como mostra a figura 3.3. O PE também é responsável por alguns serviços que não dependem dos outros componentes do servidor.

O Banco de Transputers constitui o núcleo do servidor de processamento paralelo. Agrega os processadores Transputer, os quais constituem os elementos básicos do recurso de processamento paralelo compartilhado entre os usuários do sistema. O acoplamento entre o PE e o BT é feito através de um canal INMOS. O banco possibilita a interligação dos processadores constituindo redes. Várias redes podem ser definidas simultaneamente. Cada uma delas pode ser alocada para um usuário do servidor, e são denominadas Lotes de Transputers (LT).

No projeto do banco de Transputers o aspecto modularidade tem especial importância, permitindo um crescimento gradual do sistema, tornando o custo inicial baixo, sem limitar a capacidade máxima de processamento.

O banco de Transputers é estruturado hierarquicamente conforme mostra a figura 3.4. Processadores são agrupados constituindo módulos, os quais por sua vez, são agrupados para constituir o banco de Transputers. Note-se que a Rede de Chaveamento (RC) também é hierarquizada em dois níveis: rede de chaveamento do banco e redes de chaveamento dos módulos. Essa estrutura hierárquica pode ser expandida indefinidamente. Pode-se definir um terceiro nível denominado super banco, (um grupo de bancos), e outros níveis superiores se desejado. Uma organização similar é adotada no "supernode" do projeto Esprit [NI88a] [HE88].

A definição do número máximo de usuários do sistema deve satisfazer a seguinte relação:

$$Nu \leq \text{MIN} (\text{Net}, Np)$$

onde:

Nu => número de usuários

Net => número de estações de trabalho monousuárias

Np => número de processadores disponíveis

Essa relação exprime que o número máximo de usuários do sistema deve ser menor que o número de pontos de acesso ao sistema e menor que o número de unidades básicas, processadores, do recurso a ser compartilhado. Esses números podem variar com o crescimento do sistema distribuído ou com a ampliação dos recursos do servidor. O número médio de processadores alocados para cada usuário e o número médio de usuários do servidor também poderiam ser utilizados com critério para a seleção do número máximo de usuários, caso fossem estatisticamente conhecidos. Esses valores dependem, entretanto, fortemente do ambiente de utilização do servidor. Devido às dificuldades encontradas na definição do número de usuários, adotou-se a seguinte relação:

3.4 Estrutura Hierárquica do Banco de Transputers

3.5 Diagrama de Blocos - Processador

$$Nu = Npi/4$$

onde:

Npi --> número de processadores instalados

Essa relação introduz modularidade na definição do número de usuários, permitindo seu aumento com a expansão do servidor. A análise de dados reais de utilização do sistema deverá permitir o julgamento da conveniência dessa decisão.

3.5 Especificação do Hardware

Na definição do hardware do sistema foram considerados, além dos requisitos anteriormente definidos, aspectos relacionados com a sua implementação. Este procedimento é consequência direta da decisão de utilizar-se Transputers no servidor, que é uma decisão a nível de implementação.

3.5.1 Processadores

Cada processador constitui uma unidade básica do recurso computacional a ser compartilhado. Basicamente, como mostra a figura 3.5, o processador dispõe de um microcomputador Transputer INMOS T800, memória e um circuito acoplador com o barramento de controle do módulo. Duas classes de processadores são identificadas dentro do banco. Processadores Raiz de lote (PR) e Processadores de Trabalho (PT). Cada lote dispõe de um processador raiz e normalmente de um ou mais processadores de trabalho. Todos os processadores do lote podem participar diretamente na execução do programa do usuário. O processador raiz é responsável pela comunicação do lote com o meio externo. Além disso, nele são normalmente executados os sistemas para o desenvolvimento de programas paralelos, como é o caso do TDS. A figura 3.6 exemplifica um lote com 3 processadores.

3.6 Lote com 3 Processadores

3.7 Interface do Barramento de Controle

O que diferencia um processador raiz de um processador de trabalho é a quantidade de memória disponível. Os processadores raiz devem ter uma quantidade maior de memória para permitir a execução dos sistemas de desenvolvimento. No caso do TDS, a quantidade mínima de memória é 1 Mbyte. São entretanto recomendados 2 Mbytes para uma utilização plena. Prevendo-se necessidades adicionais de memória, cada processador raiz dispõe de 4 Mbytes de memória. Os processadores de trabalho dispõem de 1 Mbyte de memória. A escolha desses valores depende da utilização dada ao servidor e deverá ser julgada pelos usuários no desenvolvimento das suas aplicações.

O circuito acoplador com o barramento de controle é necessário para a geração dos sinais RESET e ANALYSE e a verificação do sinal ERROR associados a cada lote de Transputers. Esses sinais são importantes para a iniciação dos processadores do lote (RESET), extração de dados para depuração de programas (ANALYSE) e para o tratamento de erros em tempo de execução (ERROR). A figura 3.6 ilustra como esses sinais são logicamente acoplados em um lote de Transputers. Deve-se observar que o sinal ERROR indica a ocorrência de uma condição de erro em um dos processadores da rede. Os sinais RESET e ANALYSE são aplicados simultaneamente a todos os processadores da rede.

Os sinais que constituem o barramento de controle aparecem na figura 3.7. Existem cinco linhas de endereço, (A4..A0), quatro linhas de dados/endereços, (D3..D0), duas linhas de função, (F1 e F0), uma linha de entrada de erro do processador anterior no módulo, (/EI), uma linha de saída de erro para o próximo processador do módulo, (/EO) e uma linha que indica a ocorrência de um erro em cada processador (/EP). As quatro funções de controle implementadas são descritas no próximo item.

Cada processador é identificado dentro do módulo por um endereço com 5 bits variando entre 1 e 30 (EP). Os processadores de um módulo podem pertencer a no máximo 15 lotes distintos. Esses lotes são identificados dentro do módulo por um endereço com 4 bits variando entre 1 e 15 (EL). O endereço EL = 0 define o controlador do módulo. O endereço EP = 0 define um processador não alocado, isto é, um processador que não faz parte de lote algum. O endereço EP é definido para cada processador através de chaves posicionadas manualmente. Para alocar um determinado processador, deve-se armazenar em um registrador o endereço do lote ao qual ele será alocado. Os sinais RESET e ANALYSE são alterados escrevendo-se nos registradores a eles associados. Os

canais de comunicação dos processadores são acoplados à rede de chaveamento do módulo. Essa, juntamente com a rede de chaveamento do banco, constitui a rede de chaveamento do servidor.

3.5.2 Módulos

Os módulos definem o próximo nível hierárquico da arquitetura do sistema e são responsáveis pelo controle e comunicação dos processadores a eles subordinados. Cada módulo dispõe de um computador Controlador de Módulo (CM), uma Rede de Chaveamento (RC) e um circuito acionador do barramento de controle, conforme mostra a figura 3.8.

Devido a aspectos relacionados com a disponibilidade de componentes e outros detalhes a nível de implementação, foram adotados 15 processadores por módulo. Esse número pode ser estendido até um máximo de 30, se o servidor for constituído por apenas um módulo.

O CM é constituído por um processador semelhante a um processador de trabalho do banco. A rede de chaveamento do módulo é configurada pelo controlador através de um dos seus canais de comunicação e de circuitos acopladores adequados. Vale lembrar que essa rede faz parte da estrutura hierárquica que constitui a rede de chaveamento do servidor. A configuração dessa rede depende da informação armazenada no controlador do banco.

O sinais presentes no barramento de controle foram apresentados no item anterior. A tabela 3.1 sintetiza as funções de controle implementadas. A função FC0 permite a alocação de um processador em um determinado lote, ou seja, permite que um processador possa ser utilizado na rede definida por um usuário do sistema. A função FC1 presta-se à aplicação dos sinais RESET e/ou ANALYSE em um determinado lote. A identificação de um processador ou lote na condição de erro é feita examinando-se, inicialmente, o sinal /EO do primeiro processador do módulo. Isso determina se o módulo tem algum processador na condição de erro. A seguir, a função FC2 pode ser utilizada para examinar o sinal /EP, o qual indica uma eventual condição de erro no processador endereçado. A função FC3 possibilita o mascaramento da indicação de erro dos

3.8 Diagrama de Blocos de um Módulo

3.9 Rede de Chaveamento "CrossBar"

processadores de um lote, encadeada no módulo através dos sinais /EI e /EO. Isso permite a identificação de outros lotes com processadores na condição de erro.

Função	F1	F0	A4..A0	D3..D0	/ST	/EO	/EP
FC0	0	0	EP	EL	^	X	X
FC1	0	1	EL	A/R	^	X	X
FC2	1	0	EP	X	X	X	/ERROR
FC3	1	1	EL	I	^	/EI	X

Tabela 3.1 Resumo das funções disponíveis no barramento de controle

3.5.3 Rede de Chaveamento

Conforme definido anteriormente, a rede de chaveamento deve permitir conectividade total entre os processadores do banco de Transputers. Isso implica que o grafo de conexão dos processadores pode apresentar uma topologia qualquer. A formação de subgrafos desconexos está implícita nessa asserção, possibilitando a constituição dos diferentes lotes.

Para atender ao requisito estipulado pode-se empregar uma única chave do tipo "crossbar" com entradas em número suficiente para ligar todos os canais dos processadores disponíveis, conforme pode ser visto na figura 3.9. Apesar de conceitualmente simples e versátil, essa possibilidade apresenta alto custo e não permite o crescimento proporcional da rede de chaveamento com o aumento do número de processadores. O custo da rede de chaveamento é importante porque ele pode superar facilmente o custo dos processadores do sistema. No apêndice A1 são feitas análises de custo dessa e de outras possibilidades para a organização da rede de chaveamento. Essas análises resultaram na escolha da configuração apresentada nas figuras 3.10 e 3.11. A arquitetura da rede de chaveamento escolhida atende ao requisito de conectividade total entre os processadores, conforme [IN89b] [NI88a] [HE89a].

Observa-se que a rede de chaveamento do banco apresenta a mesma estrutura das redes de chaveamento dos módulos. Essa organização modular e hierárquica permite que outros níveis sejam definidos além dos dois previstos neste trabalho. Cada nível inferior

- $n \Rightarrow$ número de processadores
 $CU \Rightarrow$ Canais de Usuário
 $P_i \Rightarrow$ i-ésimo Processador
 $u \Rightarrow$ número de canais de usuário no módulo
 $e \Rightarrow$ número de canais ETL no módulo

3.10 Rede de Chaveamento do Módulo

$m \Rightarrow$ número de módulos
 $CE \Rightarrow$ Canais ETL
 $M_i \Rightarrow$ i-ésimo Módulo

3.11 Rede de Chaveamento do Banco

deve ser ligado ao nível superior com tantos canais quanto forem os canais dos processadores presentes ou provenientes de um nível ainda mais inferior. A estrutura pode ser vista como uma árvore, onde os módulos abrigando os processadores constituem as folhas e as redes de chaveamento de nível superior, os ramos. Cada nível nessa árvore impõe um atraso adicional na comunicação entre os processadores determinado pelo atraso nas redes de chaveamento de cada nó.

A rede de chaveamento proposta permite conectividade total entre os processadores com o ônus da perda de identidade dos canais dos mesmos. Isso significa que não é possível especificar livremente os canais envolvidos nas conexões em um lote de Transputers.

Alguns canais são ligados à rede de chaveamento e acoplados ao controlador do módulo para prover comunicação entre o processador raiz de um lote e a estação de trabalho do usuário. Esses canais são denominados canais de comunicação de usuário. Os canais de comunicação restantes da rede de chaveamento do módulo são ligados à rede de chaveamento do banco para comunicação com as estações de trabalho locais.

Uma restrição adicional pode ser imposta às chaves que compõem a rede de chaveamento do servidor. Conforme [NI88a], não é necessário que essas chaves sejam do tipo "crossbar". Basta que elas permitam que um canal ligado em um dos lados possa ser ligado a qualquer outro do lado oposto. Não é necessário que canais do mesmo lado da chave possam ser ligados. Chaves com essa restrição podem ser implementadas com um número menor de componentes e apresentam menor custo que chaves do tipo "crossbar" convencionais. A adoção dessas chaves agrava ainda mais a perda de identidade dos canais, mencionada anteriormente. A conectividade total é mantida, e em função do seu menor custo, essa foi a opção empregada neste trabalho.

Quatro canais de comunicação são definidos na rede de chaveamento de cada módulo, como canais de comunicação de usuário. Empregando-se duas chaves com 64 canais por módulo, são utilizados 60 canais para a conexão dos 15 processadores, 4 canais de usuário e 64 canais interligados com a rede de chaveamento do banco. Desses últimos, 4 são reservados para as Estações de Trabalho Locais (ETL), sendo denominados canais ETL. A rede de chaveamento do banco, constituída por duas chaves com 2048 canais, comporta 64 módulos, (960 processadores), e 256 canais ETL. Se desejado, cada conjunto

de 64 canais ETL pode interligar um módulo adicional. Caso não sejam desejados canais ETL, quatro módulos podem ser acrescentados, totalizando 1020 processadores. Os canais ETL também podem ser convertidos, a nível de módulo, em canais de usuário.

3.5.4 Comunicação e Controle

Os controladores dos módulos e do banco (CM, CB) são responsáveis pela programação da rede de chaveamento, definindo os lotes de Transputers, e pelo roteamento dos pacotes de dados, trocados entre as estações de trabalho dos usuários e os correspondentes lotes de Transputers. São ainda responsáveis pelo controle de cada lote conforme os pacotes com comandos de controle recebidos.

Considerando dois níveis na rede de chaveamento do servidor de processamento paralelo, a comunicação entre os controladores de módulos e de banco é feita por uma interligação constituindo um anel. Essa configuração permite que o número de módulos do banco seja alterado sem grandes alterações no hardware. Na figura 3.12 é mostrada a configuração descrita. No caso do servidor ser constituído por apenas um módulo, o controlador de banco pode ser suprimido e o controlador do módulo passa a desempenhar também as funções de controlador de banco.

Se a rede de chaveamento do servidor for constituída por mais de dois níveis, a interligação em anel pode ser mantida ou hierarquizada. Nesse último caso devem ser providos, de alguma maneira, os canais de comunicação adicionais que se fizerem necessários.

O diagrama de blocos do controlador de módulo pode ser observado na figura 3.13. Deve-se notar os blocos correspondentes aos circuitos responsáveis pela operação do barramento de controle, pela programação da rede de chaveamento do módulo e pelo atendimento dos canais de comunicação com os lotes. Os canais de comunicação de cada lote, ligados diretamente aos seus processadores raiz, são atendidos pelo controlador em

3.12 Anel Formado pelos Controladores dos Módulos e do Banco

3.13 Diagrama de Blocos - Controlador de Banco

3.14 Diagrama de Blocos - Controlador de Módulo

tempo compartilhado. O sincronismo na comunicação é provido pelo hardware, conforme será visto no capítulo seguinte.

O controlador de banco apresenta uma estrutura similar ao CM, mostrada na figura 3.14, cabendo a ele a programação da rede de chaveamento do banco. Esse controlador centraliza toda a comunicação com o banco de Transputers, recebendo e enviando pacotes para o PE e cada um dos controladores de módulo. Cabe aos CM a troca de pacotes entre o CB e os lotes correspondentes aos diversos usuários do sistema.

3.6 Especificação do Software

Este trabalho não tem a pretensão de cobrir todos os itens de software necessários para a operação irrestrita do servidor de processamento paralelo. Muitos desses itens serão desenvolvidos concomitantemente com a operação do sistema. Nesta seção é especificada uma versão inicial do sistema, com um mínimo de funcionalidade.

A primeira consideração em relação ao software do sistema SPP procura avaliar a utilização de software existente, comercial ou de domínio público. Conforme discutido em 3.4.1, optou-se por tornar esse software desvinculado dos comercialmente disponíveis. Por outro lado, considerou-se importante o compartilhamento do meio físico da rede local, como ilustra a figura 3.1. Nesse sentido foi adotado um mecanismo para acesso ao meio físico e multiplexação de pacotes que possibilita essa facilidade. Esse mecanismo é de domínio público, e constitui a única parte da primeira versão do sistema SPP não especificada neste trabalho.

Especial atenção foi dada à utilização de ferramentas de engenharia de software no desenvolvimento dos programas. Isso se reveste de fundamental importância, uma vez que somente o núcleo do sistema foi implementado neste trabalho. Muitos outros módulos adicionais deverão ser acrescentados ao sistema durante sua utilização, além das manutenções de caráter corretivo ou evolutivo que provavelmente ocorrerão.

3.6.1 Metodologia de Desenvolvimento do Software

Diversas abordagens e metodologias para o desenvolvimento de software podem ser encontradas na literatura disponível. Especificamente para o tipo de sistema envolvido neste trabalho, isto é, de tempo real com características de sistema reativo, podem ser citados os métodos de Jackson [MA89] [CA86], Warnier [WA84], Mascot [SI79], DARTS [GO84] [GO86] e o descrito em [LO90]. Mapas de estado também aparecem como uma ferramenta importante na especificação dessa classe de sistemas [HA87] [HA88a] [HA88b]. Cada um desses métodos apresenta vantagens e restrições, não cabendo neste trabalho uma análise detalhada a respeito. Sob o ponto de vista deste trabalho é importante o benefício que a utilização de qualquer um deles proporciona, uma vez que tornam disponíveis técnicas e diretrizes para apoio às fases de desenvolvimento e manutenção do sistema.

Devido a ser uma extensão da análise e projeto estruturados [GA83] [PA88], bastante utilizados e difundidos, optou-se pela utilização da metodologia DARTS (Design Approach for Real Time Systems). Na utilização dessa metodologia, os diagramas de transição de estados, originalmente previstos, foram substituídos por mapas de estado. Mapas de estado podem representar de maneira mais clara e elegante a relação entre os módulos do sistema, particularmente aqueles que operam concorrentemente, além de permitir uma estruturação hierárquica do diagrama resultante [HA88a]. Na figura 3.15 são definidos os símbolos que aparecem nos diversos diagramas utilizados no desenvolvimento do software.

3.6.2 Mecanismo de Acesso ao Meio

O mecanismo de acesso ao meio físico da rede local deve prover funções que permitam sua utilização pelos diferentes sistemas que o utilizam para transmissão de dados. Esses sistemas são normalmente incompatíveis entre si, e alguma forma de multiplexação dos pacotes de dados deve ser implementada. O endereço físico das estações destino e origem não pode suprir essa necessidade, no caso de redes locais Ethernet. Os pacotes podem ser transmitidos com endereços múltiplos, por exemplo, pacotes tipo "broadcast". Um mecanismo dessa classe, denominado "Packet Driver" (PD),

	ME	DFD	DET	DEM
Estado	Processo	Tarefa	-
-	Depósito de Dados	Depósito de Dados	Módulo
-	Entidade Externa	Entidade Externa	-
Transição	Fluxo de Dados	Fluxo de Dados	Fluxo de Controle
-	-	Comunicação Síncrona Unidirecional	-
-	-	Comunicação Síncrona Bidirecional	-
-	-	Comunicação Assíncrona c/ Armazen.	-
-	-	-	Execução Condicional
-	-	-	Execução Alternativa
-	-	-	Iteração

3.15 Símbolos Utilizados nos Diagramas Relativos ao Software

está disponível como software de domínio público na Universidade de Clarkson - NY - USA [FT89] [CL89a]. Na verdade, o PD é um módulo integrante de uma implementação do conjunto de protocolos TCP/IP. Além da necessidade mencionada, as seguintes características tornam a utilização desse software atrativa:

- > Possibilita uma interface padrão com as camadas superiores do software do sistema, tornando-as independentes das características do meio físico.
- > Permite a execução de múltiplas aplicações simultaneamente utilizando a mesma interface com o meio físico da rede local.
- > Os pacotes de dados não relevantes para a aplicação sendo executada são rapidamente descartados pelo PD. A aplicação somente recebe os pacotes destinados a ela.

O "packet driver" associa um tipo a cada pacote de dados. Um pacote somente será processado se a recepção para pacotes do seu tipo tiver sido previamente autorizada. Isso é feito por uma função do PD a qual associa ao tipo de pacote um identificador e uma rotina de recepção específica. A especificação das funções disponíveis no PD pode ser vista no apêndice A2.

Na figura 3.16 é mostrado o mapa de estados que espelha a operação básica do "packet driver". Basicamente, três módulos de software estão envolvidos: o PD, uma rotina para recepção e tratamento de pacotes (ReceberPkt) e o aplicativo. Quando um pacote válido é recebido pelo hardware de interface com o meio físico da rede, uma interrupção de hardware (int) é gerada e a execução é transferida para o PD. Esse analisa o tipo do pacote, descartando-o ou prosseguindo com a sua transferência em função dessa análise. A rotina de recepção pode aceitar a transferência do pacote ou descartá-lo. No primeiro caso, o PD transfere o pacote para o "buffer" especificado pela rotina de recepção, devolvendo em seguida o fluxo de execução novamente para a rotina de recepção, para um eventual processamento do pacote. Uma vez processado, o fluxo de controle volta novamente para o PD que finalmente transfere a execução para o aplicativo. No segundo caso, a execução é transferida para o PD, que por sua vez, transfere a execução para o aplicativo.

3.16 Mapa de Estados - PD

Apesar de não ser convencional, os fluxos de dados envolvidos estão indicados nesse diagrama entre parênteses. BufLen indica o tamanho necessário para o "buffer" de recepção. AX (registrador), indica para a rotina de recepção a primeira ($AX = 0$), ou a segunda chamada ($AX \neq 0$) feita pelo PD. A rotina de recepção deve devolver o endereço do buffer $\neq 0$ para que o pacote seja transferido e não descartado.

3.6.3 Visão Cliente-Servidor do Sistema

O software do sistema foi projetado respeitando-se o modelo cliente-servidor considerado em [TA85]. Na implementação desse modelo foi utilizado um mecanismo RPC ("Remote Procedure Call") [WI87] [TA88].

Uma visão geral do sistema, sob o ponto de vista dos aspectos mencionados acima, é mostrada na figura 3.17. Existem cinco elementos ativos principais que participam do processo de comunicação: as estações de trabalho, o processador de entrada, o controlador de banco, os controladores de módulo e os processadores raiz de cada lote. Neste trabalho, o controlador de banco foi excluído, como justificado em 3.5.4.

As diversas primitivas disponíveis no sistema são executadas pelo processador de entrada e pelo controlador de módulo. Essas primitivas são descritas no próximo capítulo. O servidor do cliente "estação de trabalho" está fisicamente distribuído no processador de entrada e no controlador de módulo. O processador de entrada atende as primitivas responsáveis pelas funções básicas do sistema, tais como as de abertura e fechamento de uma sessão. O controlador de módulo atende as primitivas relacionadas com a utilização do banco de Transputers, tais como alocação de processadores e controle dos lotes.

Durante a execução do TDS, alguns elementos do sistema têm a sua função alterada. Sob esse ponto de vista, os clientes deixam de ser as estações de trabalho e passam a ser representados pelos processadores raiz de cada lote. O processador de entrada e o controlador de módulo mantêm a função de servidor, atendendo agora também às requisições provenientes dos lotes. As estações comportam-se como servidores para o TDS, apresentando comportamento dual, ora de cliente, ora de servidor. Todo o processo descrito não contradiz o modelo cliente-servidor e o mecanismo RPC adotados.

ET estação de trabalho
 PE processador de entrada
 CM controlador de módulo
 PR processador raiz de lote
 ETH rede ethernet
 LNK canal INMOS
 SPP protocolo do servidor
 SP protocolo iserver - INMOS
 SP* protocolo iserver - INMOS modificado

3.17 Organização dos Pares Cliente-Servidor

Deve-se notar que a execução do TDS pode ser considerada como a execução de uma primitiva com tempo de execução indeterminado.

3.6.4 Especificação do Software - ET

O mapa de estados do programa executado nas estações de trabalho aparece na figura 3.18. Nesse diagrama, para efeito de clareza, não foi incluído o tratamento das condições de exceção. Após a iniciação da estação, está prevista a execução de aplicativos correspondentes às seguintes funções do sistema:

- > Abertura de sessão (login)
- > Encerramento de sessão (logout)
- > Alocação de processadores (lote)
- > Configuração dos processadores do lote (config)
- > Execução do sistema de desenvolvimento TDS (tds)

A execução de cada aplicativo corresponde a uma ou mais funções primitivas implementadas a nível do mecanismo de chamada a procedimentos remotos (RPC).

Na abertura de uma sessão, o usuário deverá informar:

- > Nome
- > Senha

Caso os dados estejam corretos, uma sessão será aberta para o usuário. Se algum dado estiver incorreto, a abertura de sessão será negada. O usuário pode, uma vez aberta a sessão, executar um dos outros utilitários.

3.18 Mapa de Estados - ET (ME-ET)

Para a utilização efetiva do banco é necessário que o usuário requisi-te a alocação de um certo número de processadores do banco. Isso é feito pelo utilitário LOTE, especificando-se o número de processadores desejado. Caso os processadores solicitados estejam disponíveis, o usuário será atendido. Caso contrário, o usuário perderá os processadores anteriormente alocados, se existirem. Esse procedimento foi adotado para a prevenção de "deadlock" na requisição dos recursos. Numa segunda versão do sistema, as requisições dos usuários serão enfileiradas. À medida que os recursos se tornem disponíveis, os usuários serão atendidos segundo prioridades a serem definidas.

A interligação dos processadores alocados pode ser feita pelo utilitário CONFIG. A especificação do grafo de conexões entre processadores é feita da seguinte forma:

0	C	1.1	1.0	-
1	0.2	0.1	2.0	2.3
2	1.2	2.2	2.1	1.3

Cada linha corresponde a um Transputer da rede. A primeira coluna contém um número que identifica o Transputer, variando entre 0 e 999. O número 0 corresponde ao Transputer raiz. As quatro colunas seguintes especificam as ligações dos quatro canais do Transputer, na seqüência 0, 1, 2 e 3, conforme os seguintes casos:

> n1.n2

n1 é o número do Transputer vizinho e n2 o número do seu canal envolvido na conexão.

> C

Indica o canal utilizado para comunicação com a rede. Apenas um canal, do Transputer raiz, é utilizado para comunicação.

> -

Indica um canal sem conexão.

> En

Indica uma conexão com o canal ETL de número n.

O exemplo apresentado acima corresponde ao lote da figura 3.6.

Deve-se observar que a identificação dos canais físicos dos processadores é ignorada na configuração do lote, pois a rede de chaveamento do banco não permite tal facilidade. A execução desse utilitário resulta na confirmação da configuração realizada, a qual pode ser armazenada num arquivo para subsequente utilização. Um trabalho correntemente em curso utiliza essa informação em um utilitário que atualiza automaticamente a configuração de programas fonte OCCAM.

O usuário pode carregar um aplicativo na rede de processadores configurada. Um desses aplicativos é o sistema de desenvolvimento INMOS TDS. Além desse, outros aplicativos deverão ser adaptados para a utilização de outras linguagens e utilitários.

Ao final da sessão, o usuário deverá encerrá-la explicitamente utilizando o comando LOGOUT.

A partir das atividades mostradas no mapa de estados e outras considerações, foi definido o diagrama de fluxo de dados ilustrado na figura 3.19. Nesse diagrama foram excluídas as funções relativas à iniciação das estações de trabalho. Essas funções deverão incluir o teste e a configuração das estações e serão incluídas posteriormente. O recebimento dos pacotes PKT é realizado pelo PD ("packet driver"). Deve-se notar que as áreas sombreadas na figura correspondem às tarefas que são executadas concorrentemente. Admitiu-se que a tarefa de recebimento dos pacotes, iniciada por interrupção de hardware, pertence a essa classe. Na verdade, deveriam pertencer somente aquelas onde a concorrência fosse implementada por tempo compartilhado ou alguma forma de execução paralela.

Na figura 3.20 aparece o diagrama de estrutura de tarefas dos programas executados nas estações de trabalho. Existem nesse diagrama duas tarefas. Uma das tarefas, "receber_PKT", é acionada por uma interrupção de hardware através do mecanismo descrito no funcionamento do PD. Uma estrutura de dados global é utilizada para a comunicação entre as duas tarefas.

3.19 Diagrama de Fluxo de Dados - ET (DFD-ET)

3.20 Diagrama de Estrutura de Tarefas ET (DET-ET)

3.21 Diagrama de Estrutura de Tarefas PE (DET-PE)

3.6.5 Especificação do Software - PE

O processador de entrada constitui um dos componentes do servidor do sistema. O PE atende requisições provenientes das estações de trabalho e dos lotes de processadores, conforme pode ser observado na figura 3.17.

O mapa de estados do processador de entrada é mostrado na figura 3.22. Deve-se observar que não foi incluído o tratamento de exceções. Também não aparecem a origem das requisições e o destino das respostas. Tal como nas estações de trabalho, foi utilizado o mecanismo descrito em 3.6.1 para acesso ao meio físico da rede e multiplexação de pacotes.

O diagrama de fluxo de dados do processador de entrada é mostrado na figura 3.23. Nesse diagrama podem ser observadas as funções desempenhadas pelo mesmo. As funções relativas à iniciação do processador de entrada foram omitidas nessa versão inicial do sistema. A estrutura das funções implementadas e os fluxos de dados envolvidos sugerem várias tarefas como candidatas para execução concorrente. Alguns aspectos a nível de projeto e implementação, tais como particularidades de operação da interface entre o PE e o CM, sugeriram uma simplificação na estrutura das tarefas nessa primeira versão do sistema. No capítulo seguinte esses aspectos são discutidos detalhadamente. Dessa forma, optou-se pela estrutura de tarefas delineada na figura 3.23 e mostrada no diagrama de estrutura de tarefas da figura 3.21. Observa-se que esse último apresenta grande similaridade com o DET - ET mostrado na figura 3.20. No caso do processador de entrada, entretanto, foi utilizada uma fila de recepção para comunicação entre as tarefas. Essa fila é considerada um módulo de comunicação entre tarefas MCM ("Message Communication Module"), e implementada como um módulo IHM ("Information Hiding Module"), segundo a metodologia DARTS [GO84] [LI86] [PR87].

Uma restrição adicional para a operação do PE, tal como descrita acima, se faz necessária. Como as requisições são feitas através de dois canais físicos distintos, ou seja, a interface com a rede local e a interface PE/CM, é preciso garantir que nenhuma requisição demore muito para ser atendida. É necessário que o tempo de atendimento de requisições pelo PE seja limitado e previsível, ou seja, que no PE somente sejam implementadas primitivas com essas características.

3.22 Mapa de Estados - PE (ME-PE)

3.23 Diagrama de Fluxo de Dados - PE (DFD-PE)

Os protocolos de comunicação envolvidos e o formato dos pacotes serão descritos no próximo capítulo.

3.6.6 Especificação do Software - CM

O controlador de módulo constitui o outro elemento que compõe o servidor do sistema SPP. O software do CM possui uma estrutura similar ao PE. Porém, existem nesse caso, mais canais físicos por onde transitam os pacotes de dados.

Na figura 3.24 o mapa de estados do software do controlador de módulo é mostrado. Novamente não foram incluídas as condições de exceção e não foram explicitados os canais físicos por onde circulam os pacotes de dados.

Na figura 3.25 é mostrado o diagrama de fluxo de dados correspondente. Não foram incluídas as atividades relacionadas com o procedimento de iniciação, a serem consideradas numa segunda versão do sistema. Estão previstos quatro usuários por módulo do sistema. As áreas sombreadas na figura correspondem às diversas tarefas executadas concorrentemente, detalhadas na figura 3.26. Deve-se notar que a maior parte da comunicação entre tarefas é feita de forma fortemente acoplada, conforme descrito em [GO84]. A notação original foi ligeiramente alterada nos casos de comunicação unilateral. Essa forma de comunicação é tratada diretamente pelo hardware e software do Transputer utilizado na implementação do controlador de módulo. Não se faz necessária, nesse caso, a implementação de um módulo específico de comunicação entre tarefas (MCM), previsto na metodologia DARTS.

A comunicação entre as tarefas responsáveis pela interface com o processador de entrada e as demais tarefas do controlador de módulo foi implementada através de filas constituídas por módulos IHM.

3.24 Mapa de Estados - CM (ME-CM)

3.25 Diagrama de Fluxo de Dados - CM (DFD-CM)

3.26 Diagrama de Estrutura de Tarefas CM (DET-CM)

3.6.7 Correlação com o Modelo ISO-OSI

A utilização do modelo ISO-OSI ("International Standard Organization - Open Systems Interconnection") [TA88], tem sido defendida por diversos autores como uma alternativa viável no sentido da padronização do software de comunicação utilizado em redes de computadores. Apesar disso, o que se observa é sua pequena utilização prática. Justifica-se a não utilização do modelo considerando-se entre outros, aspectos como eficiência e adequação. No capítulo 2 pode ser encontrada uma discussão mais detalhada a respeito.

O modelo ISO-OSI também não dá suporte adequado ao mecanismo RPC. Alguns autores consideram que ele deve ser implementado na terceira camada do modelo ISO-OSI, enquanto outros consideram a camada 5 mais conveniente [TA88].

A figura 3.27 e 3.28 mostram uma correlação entre as camadas previstas no modelo ISO-OSI e as implementadas no sistema SPP. A interface Ethernet implementa até a subcamada MAC ("Medium Access Control") da camada de dados. O "packet driver" implementa as funções restantes da camada de dados. A camada de rede e transporte não foram implementadas pois não houve necessidade das funções ali previstas. Em versões futuras do sistema, com a utilização de outras linguagens e aplicativos, pode surgir a necessidade de implementação da camada de transporte para possibilitar a transferência de pacotes de dados maiores que os permitidos pela camada de dados. O mecanismo RPC foi implementado na camada de sessão, conforme sugere [TA88]. As primitivas e aplicativos foram implementados numa camada mais elevada, englobando as camadas de apresentação e aplicação do modelo ISO-OSI.

A estrutura em camadas correspondente ao software do controlador de módulo é apresentada na figura 3.29. Nesse caso, nenhuma correlação pode ser feita com o modelo ISO-OSI.

SP - protocolo iserver INMOS

SPP - protocolo servidor de processamento paralelo

PKT - protocolo packet driver NCSA

ETH - protocolo ethernet

3.27 Estrutura em Camadas do Software - ET

SPP - protocolo servidor de processamento paralelo

PKT - protocolo packet driver NCSA

ETH - protocolo ethernet

3.28 Estrutura em Camadas do Software - PE

SP - protocolo iserver INMOS

SP* - protocolo iserver INMOS modificado

SPP - protocolo servidor de processamento paralelo

3.29 Estrutura em Camadas do Software - CM

3.7 Considerações Finais

Este capítulo cuidou da especificação do sistema servidor de processamento paralelo. Foram discutidas as diversas alternativas consideradas na definição da versão inicial do sistema. Alguns aspectos a nível de implementação foram mencionados, uma vez que o sistema está baseado na família de dispositivos INMOS Transputer.

Foram levadas em consideração a facilidade de utilização e a versatilidade do sistema para atender suas finalidades principais, ou seja, servir como objeto de pesquisa e ferramenta de trabalho nas áreas de processamento paralelo e sistemas distribuídos. Apesar do cuidado tomado na especificação do sistema, alguns aspectos poderão exigir revisões adicionais antes de serem implementados. Isso resulta da complexidade do sistema e da característica dinâmica do processo de desenvolvimento.

No capítulo seguinte apresenta-se as partes especificadas do sistema que são tratadas a nível de projeto e implementação.

CAPÍTULO 4

PROJETO E IMPLEMENTAÇÃO DO SISTEMA

4.1 Considerações Iniciais

No capítulo anterior foram apresentados os diversos aspectos relativos à arquitetura e especificação da versão inicial do sistema servidor de processamento paralelo. Neste capítulo, são abordados os aspectos relacionados com o seu projeto e implementação.

No item 4.2 é apresentado o projeto e implementação do hardware do sistema. São discutidas as decisões tomadas a nível de projeto, em função da tecnologia utilizada na implementação.

O projeto do software do sistema é apresentado no item 4.3, onde são introduzidos os diagramas de estrutura modular do software desenvolvido. As primitivas do servidor também são introduzidas nesse item.

O item 4.4 trata da instalação do sistema de desenvolvimento TDS no servidor de processamento paralelo. É apresentada a organização do TDS e as alterações que se fizeram necessárias para que sua instalação fosse possível.

4.2 Projeto e Implementação do Hardware

O projeto e implementação do hardware do sistema SPP foi baseado nos componentes comercialmente disponíveis da família INMOS Transputer. Procurou-se utilizar tecnologia de baixo custo, disponível no mercado nacional.

Utilizou-se para dar suporte físico dos módulos de hardware um bastidor padrão eurocard duplo (placas com 220 x 233.4 mm). As placas de circuito impresso foram projetadas em dupla face. A utilização de placas multi-face tornaria mais simples o projeto nos aspectos relacionados com o roteamento e a eliminação de ruídos dificultando, entretanto, sua confecção, uma vez que existem atualmente no Brasil poucas indústrias equipadas para a produção desse tipo de placa. Por outro lado, existe um grande número de indústrias qualificadas para a produção de placas de dupla face. Algumas universidades também dispõem desse tipo de serviço. A adoção de dispositivos SMD ("Surface Mounted Devices") e módulos de memória, no lugar dos dispositivos convencionais utilizados, permitiria reduzir a um quarto a ocupação das placas projetadas. Esse fatores comprometeriam, entretanto, a facilidade de reprodução do sistema, o que dificultaria sua divulgação mais ágil no meio acadêmico, incentivando novas pesquisas nas áreas de conhecimento envolvidas.

4.2.1 Processadores

Conforme descrito em 3.5.1, as placas processadoras constam basicamente de um microcomputador Transputer T800, 1 a 4 Mbytes de memória adicional e um circuito responsável pelo acoplamento com o barramento de controle do sistema (ver figura 3.5).

O Transputer utilizado é acionado por um "clock" a 20 Mhz, apresentando as seguintes características:

- > 10 MIPS (processador de 32 bits)
- > 1.5 MFlops (processador de ponto flutuante, em regime)
- > 4 Kbytes de memória interna (ciclo 50 ns)
- > 4 canais de comunicação bidirecionais (20 Mbits/s)

A figura 4.1 mostra o diagrama de blocos simplificado do Transputer T800. Podem-se observar as duas unidades de processamento (inteiros e ponto flutuante), e as diversas unidades funcionais operando em paralelo.

A memória externa utilizada foi implementada com circuitos integrados organizados em 256K posições x 4 bits, 80 ns de tempo de acesso. Foram previstos 4 bancos de 256K posições x 32 bits, permitindo processadores com 1, 2, 3 ou 4 Mbytes de memória externa. O tempo para leitura e escrita ficou reduzido ao mínimo de três ciclos permitidos pelo processador, resultando, para o "clock" adotado, em 150 ns.

O circuito de interface com o barramento de controle do módulo foi implementado utilizando-se dois dispositivos PAL ("Programmable Array Logic"). O diagrama lógico equivalente pode ser observado na figura 4.3. Os sinais e as funções desempenhadas por esse circuito são descritos no item 3.5.1. Os circuitos contidos dentro das linhas tracejadas correspondem às duas PAL projetadas.

A cada processador estão associados dois números:

- > endereço do processador ($1 \leq EP \leq 30$)
- > número do conector

É conveniente que o endereço do processador seja posicionado de acordo com o número do conector que ele ocupa. Nada impede, entretanto, que outros valores possam ser utilizados. Na figura 4.4 pode-se ver a disposição dos componentes em uma placa processadora. Dois processadores estão implementados em cada placa.

4.2.2 Rede de Chaveamento

No projeto da rede de chaveamento do banco foi utilizada a chave INMOS C004. Essa chave é constituída por um único circuito integrado, o qual implementa uma chave do tipo "crossbar" para 32 canais INMOS, operando em 10 ou 20 Mb/s. Na figura 4.5 é mostrado o diagrama de blocos dessa chave. Observa-se nessa figura a disponibilidade de 32 entradas e 32 saídas. Um canal de configuração é utilizado para a programação das

4.1 Diagrama de Blocos - Transputer T800

4.2 Diagrama de Blocos - C011/12 (modo 2)

4.3 Interface com Barramento de Controle Processador

4.4 Placa de Processadores

interligações entre as entradas e as saídas e para ler a configuração atual. Na figura 4.5 também são mostrados os comandos de configuração disponíveis.

Para a implementação da rede de chaveamento do módulo são utilizadas 4 chaves C004. Elas são conectadas duas a duas em anti-paralelo, conforme mostra a figura 4.6. Deve-se notar a presença de quatro canais de usuário e quatro canais ETL. Essa configuração implementa as chaves descritas no item 3.5.3, com a restrição adicional de conectividade ali mencionada, ou seja, é permitida a interligação de cada um dos canais de um lado com qualquer canal do outro lado da chave. Não é permitida, entretanto, a interligação de canais situados em um mesmo lado. As interligações possíveis entre dois processadores de um módulo são mostradas na tabela 4.1.

		canais P_i			
		0	1	2	3
canais P_j	0	#	*	*	x
	1	*	#	x	*
	2	*	x	#	*
	3	x	*	*	#

x - conexão possível
 * - conexão impossível (diferentes chaves)
 # - conexão impossível (mesmo lado da chave)

Tabela 4.1 Conexões possíveis entre dois processadores de um módulo.

O projeto da rede de chaveamento do banco segue uma organização similar à organização da rede de chaveamento dos módulos. Para o número máximo previsto de processadores no banco, conforme item 3.5.3, são necessárias 4 chaves "crossbar" com 1024 canais. Para implementar cada uma dessas chaves, são necessários 96 circuitos integrados C004, totalizando 384 para a implementação da rede de chaveamento do banco. Considerando-se as redes de chaveamento dos módulos, 640 chaves C004 são necessárias para a interligação de 960 processadores (ver apêndice A1).

A chave C004 faz a regeneração dos dados recebidos, possibilitando a interligação de inúmeras chaves em cascata sem deterioração do sinal. Por outro lado, isso implica

[0] [ENTRADA] [SAÍDA]	Ligar ENTRADA com SAÍDA
[1] [CANAL1] [CANAL2]	Ligar canal 1 com canal 2
[2] [SAÍDA]	Determinar qual entrada está conectada com SAÍDA
[3]	Habilitar sequência de configuração
[4]	Inicializar a chave
[5] [SAÍDA]	Desligar SAÍDA
[6] [CANAL1] [CANAL2]	Desligar canais

4.5 Diagrama de Blocos - C004

4.6 Rede de Chaveamento do Módulo

numa redução da taxa de transferência de dados. A taxa de comunicação mínima ocorre entre processadores situados em módulos distintos. Nesse caso, cada mensagem atravessa 5 chaves C004. Mensagens entre processadores em um mesmo módulo atravessam apenas uma chave C004. Essa característica é convenientemente explorada pelo algoritmo de alocação dos processadores. São alocados processadores situados no menor número de módulos possível, no caso ideal em um único módulo.

Na implementação da rede de chaveamento do módulo foram utilizadas duas placas de circuito impresso tamanho eurocard duplo (220 x 233.4 mm). Apesar das placas apresentarem pequena densidade de componentes, como pode ser observado na figura 4.7, o número de vias necessárias nos conectores foi um fator limitante para uma maior compactação.

4.2.3 Controlador de Módulo

O diagrama de blocos do controlador de módulo é mostrado na figura 3.14. O núcleo desse controlador é composto por um circuito similar ao dos processadores disponíveis no banco de Transputers. Pela não disponibilidade do Transputer T425 foi utilizado um Transputer T800 em seu lugar.

O controlador de barramento e o controlador do multiplexador, responsável pelo desdobramento do canal de comunicação utilizado na programação da rede de chaveamento do módulo, foram implementados como portas de E/S mapeadas no espaço de endereçamento de memória do processador, utilizando-se circuitos integrados TTL.

A interface de comunicação com os canais de usuários foi implementada utilizando-se circuitos integrados INMOS C011/12. O diagrama de blocos desse componente pode ser visto na figura 4.2. Uma característica importante é o sincronismo por hardware na comunicação. Para a transmissão de um novo dado é necessária uma confirmação de recebimento do dado anteriormente transmitido, através de um sinal de reconhecimento, conforme o protocolo de comunicação de canais INMOS descrito no capítulo 2. A leitura de um dado recebido pelo dispositivo ocasiona a transmissão do sinal de reconhecimento

4.7 Placa de Chaves

para o dispositivo de origem. É possível deixar de ler o dado recebido por tempo indeterminado sem a ocorrência de erro de sobreposição de dados.

Na figura 4.8 pode ser vista a disposição dos componentes na placa do controlador de módulo.

4.2.4 Interface de Comunicação PE/BT

A interface de comunicação PE/BT é responsável pela transferência de dados entre o barramento do processador de entrada (um microcomputador IBM/AT compatível) e o canal INMOS interligado ao banco de Transputers.

Foram utilizados circuitos integrados INMOS C011/12 na implementação da interface PE/BT. O diagrama de blocos desse CI aparece na figura 4.2. O controle do fluxo de dados foi feito por software, através da leitura dos registradores de estado dos circuitos integrados.

A decodificação dos endereços do PE foi feita utilizando-se um dispositivo PAL, resultando uma placa com poucos componentes e pequenas dimensões, conforme pode ser observado na figura 4.9.

4.3 Projeto e Implementação do Software

Conforme discutido no capítulo anterior, foi adotada a metodologia DARTS no desenvolvimento do software do sistema SPP. Neste item prossegue-se com o desenvolvimento abordando-se as fases de projeto e implementação. O projeto segue as diretrizes do projeto estruturado de sistemas [PA88], que tem como principal ferramenta o diagrama de estrutura modular. Para maior clareza nos diagramas, não estão representados os parâmetros de interface dos módulos. Esses parâmetros podem ser obtidos nas descrições de interface individuais de cada módulo, apresentadas nos apêndices A3, A4 e A5.

4.8 Placa do Controlador de Módulo

4.9 Placa da Interface PE/BT

Apesar do projeto se desenvolver no sentido dos módulos hierarquicamente inferiores ("top down"), a implementação e testes do sistema seguiu caminho inverso ("bottom up"). Os testes a nível modular seguiram imediatamente a implementação dos módulos do sistema. Esse procedimento é justificado considerando-se algumas peculiaridades do sistema. A interação entre hardware e software, por exemplo, dificulta bastante as fases de implementação e teste. Os testes do hardware, na maioria dos casos, foram feitos a partir de programas cuja função é exercitar o hardware. Esses programas apresentam uma estrutura modular e foram reutilizados na implementação do sistema. Em [MI89] é discutida uma metodologia para o desenvolvimento estruturado de sistemas, envolvendo hardware e software básico, que segue as diretrizes básicas discutidas acima.

Três linguagens de programação foram utilizadas. Como linguagem principal foi adotada a linguagem C pela sua difusão e pelos recursos para programação em baixo nível disponíveis. O "packet driver" está implementado em linguagem montadora da família de microprocessadores INTEL 8086/286/386. Foram codificados em linguagem montadora módulos de interface entre os programas C e as funções disponíveis no PD. Os programas do controlador de módulo foram implementados em Occam. Nesse caso, a opção deve-se às características adequadas para manipulação de recursos de hardware, eficiência e mecanismos inerentes para programação concorrente que essa linguagem apresenta quando executada em processadores Transputer.

4.3.1 Descrição dos Protocolos, Pacotes e Estruturas de Dados

O formato dos diversos tipos de pacotes de dados utilizados pode ser observado na figura 4.10. O pacote de mais baixo nível que circula pela rede local Ethernet é mostrado na figura 2.9. Esse pacote, eliminados os campos de preâmbulo e CRC ("Cyclic Redundancy Code"), resulta no pacote tipo PKT, empregado na interface com o "packet driver" NCSA. Deve-se observar que foi adotado o padrão antigo do pacote Ethernet, conhecido como padrão DEC/INTEL/XEROX ou "blue book Ethernet". Esse padrão apresenta maior difusão que o padrão IEEE 802.3 [FT89] [CL89a], permitindo maior facilidade no compartilhamento do meio físico da rede local entre o sistema SPP e os outros sistemas existentes [CL89b].

PACOTE PKT

		ED			EO		TP	DADOS
--	--	----	--	--	----	--	----	-------

ED - endereço ethernet destino

EO - endereço ethernet origem

TP - tipo do pacote (blue-book ethernet)

PACOTE SPP

TP	DT	PR	ET	TM	PARÂMETROS
----	----	----	----	----	------------

TP - tipo do protocolo (b7 -> requisição / resposta)

DT - destino (processo / sessão)

PR - identificador da primitiva

ET - estado

TM - tamanho do pacote SPP

PACOTE SP*

B0	B1	B2	MENSAGEM / PARÂMETROS
----	----	----	-----------------------

$B0 \neq 0$ $B0 + 256*B1$ = tamanho da mensagem SP

$B2$ = identificador de primitiva TDS ou resultado da execução

$B0 = 0$ $B1$ = identificador da primitiva SPP

$B2 + 256*B3$ = tamanho do pacote SP*

4.10 Definição dos Pacotes de Dados

O protocolo de comunicação do modelo cliente-servidor foi baseado em um mecanismo RPC que utiliza o pacote de dados SPP. Esse pacote é transmitido dentro do pacote PKT, apresentando os seguintes campos:

- > TP - Tipo do Protocolo. Esse campo permite a utilização de mais de um protocolo por tipo de pacote em futuras expansões do sistema. O tipo do protocolo SPP foi definido igual a 10. O bit mais significativo indica: 1 => requisição ou 0 => resposta de primitiva. Esse campo é útil na implementação de chamadas a procedimentos remotos aninhadas.
- > DT - Destino. Indica o destino do pacote. No sentido cliente -> servidor contém o identificador da sessão do usuário (ide_sessão). No sentido servidor -> cliente contém o identificador do processo do usuário (ide_proc). A identificação do processo do usuário será útil quando o sistema for trasladado para estações de trabalho multiusuárias. Esse campo contém ide_sessão nos pacotes trocados entre o PE e o BT.
- > PR - Identificador da primitiva. Código da primitiva do sistema SPP. As seguintes faixas de código estão definidas:
 - > 0x00 - 0x3f primitivas atendidas pelo PE
 - > 0x40 - 0x7f primitivas atendidas pelo CM
- > ET - Estado. Informações diversas sobre a requisição ou resposta da primitiva, codificadas bit a bit:
 - > b7 -> sequência de chamada da primitiva
 - > b6 -> uso futuro
 - > b5 -> erro na execução da primitiva
 - > b4 -> requisição não válida
 - > b3 -> primitiva não existe ou permissão negada
 - > b2 -> sessão não existe
 - > b1 -> "time out" na execução da primitiva
 - > b0 -> uso futuro
- > TM - Tamanho do pacote SPP.

Para evitar conflito entre as diferentes representações dos tipos de dados nas estações de trabalho e servidores do sistema, os dados devem ser representados, em um pacote SPP, pelos tipos primitivos relacionados abaixo:

- > BYTE - 1 byte, valor entre 0x00 e 0xff
- > INT16 - 2 bytes, complemento de dois
- > INT32 - 4 bytes, complemento de dois
- > INT64 - 8 bytes, complemento de dois
- > REAL32 - 4 bytes, IEEE 754
- > REAL64 - 8 bytes, IEEE 754

Os bytes menos significativos devem ser transmitidos primeiro. Cabe aos programas de interface ("stubs"), nos clientes e servidores, a troca, se necessário, da representação dos tipos de dados.

O protocolo SPP dá suporte direto ao mecanismo RPC. Não são utilizadas chamadas aninhadas de procedimento, apesar de suportadas. Cada requisição e cada resposta ocupam um único pacote SPP, em ordem estrita, isso é, somente podem ser feitas novas requisições uma vez recebida a resposta da requisição anterior. A política adotada para o tratamento de falhas é "no máximo uma execução" [WI87]. O paradigma intrínseco do mecanismo RPC e as restrições acima simplificam a tarefa de validar o protocolo utilizado.

O protocolo SP* é baseado no protocolo INMOS SP, utilizado pelo sistema de desenvolvimento TDS. No item 4.4.2 esse protocolo é descrito com detalhes adicionais. As alterações feitas permitem a execução de primitivas do sistema SPP a partir de um programa do usuário, executado em um lote de Transputers. Essa possibilidade implica numa série de facilidades adicionais, destacando-se a possibilidade de configuração

dinâmica da topologia do lote de Transputers do usuário. A configuração dinâmica de redes de processadores constitui uma alternativa entre a situação ideal, redes totalmente conectadas, e redes com topologia fixa.

Diversas estruturas de dados foram utilizadas na implementação do software do sistema SPP. Apresenta-se a seguir aquelas relacionadas com a operação básica do sistema e que podem contribuir para o seu entendimento.

Estruturas de dados na estação de trabalho:

Tabela de servidores - TSERVER (indexada pelo `ide_servidor`)

<code>end_serv[]</code>	.endereço do servidor
<code>ide_sessão[]</code>	.identificador da sessão do usuário
<code>seq[]</code>	.seqüência de requisição para cada primitiva
<code>to[]</code>	.contador de "time out" para cada primitiva

Estruturas de dados no processador de entrada:

Tabela de sessões - TSES (indexada pelo `ide_sessão`)

<code>cod_usu</code>	.código de identificação do usuário
<code>end_est[]</code>	.endereço da estação
<code>ide_proc</code>	.identificador do processo do usuário
<code>seq[]</code>	.seqüência de requisição de cada primitiva
<code>tam_resp[]</code>	.tamanho máximo da resposta para cada primitiva
<code>*resp[]</code>	.ponteiros para as últimas repostas de cada primitiva

Tabela de Usuários - TU (indexada pelo `cod_usu`)

<code>nome_usu[]</code>	.nome do usuário
-------------------------	------------------

senha[]	.senha do usuário
perm[]	.permissão de execução para cada primitiva
extensão	.estatística e histórico de utilização (uso futuro)

Estruturas de dados no controlador de módulo:

Tabela de Processadores - TP (indexada pelo número do conector do módulo onde cada processador está fisicamente alojado)

EP	.endereço do processador
EL	.endereço do lote
	= 0x00 => processador não alocado
	= 0xff => processador não disponível
cond_op	.condição operacional do processador
	= 0x01 => operacional, 1 Mbyte memória
	= 0x02 => operacional, 2 Mbytes memória
	= 0x03 => operacional, 3 Mbytes memória
	= 0x04 => operacional, 4 Mbytes memória
	= 0x80 => não operacional
	= 0xff => não existente

Tabela de lotes - TLOT (indexada pelo ide_lote)

ide_sessão	.identificador da sessão
conf_atual[]	.configuração do lote, conforme o formato (para cada processador pertencente ao lote):
	byte 0 = conector do processador
	byte 1 = conector ligado ao canal 0
	byte 2 = conector ligado ao canal 1
	byte 3 = conector ligado ao canal 2
	byte 4 = conector ligado ao canal 3

4.11 Diagramas de Estrutura Modular ET (DEM-ET)

4.3.2 Estações de Trabalho

Os diagramas de estrutura modular correspondentes às tarefas executadas nas estações de trabalho aparecem na figura 4.11. As correspondentes descrições de interface dos módulos aparecem no apêndice A3. Observa-se nessa figura a estrutura de chamada de uma primitiva genérica. A execução do sistema de desenvolvimento TDS é considerada no item 4.4.3.

Cada primitiva tem associado um valor de tempo ("time out"), após o qual é feita uma nova requisição, conforme a política de tratamento de falhas implementada. Após um certo número de falhas, especificado pela constante do sistema `TIME_OUT_SPP`, considera-se que o servidor não pode atender a requisição e as tentativas são interrompidas.

4.3.3 Processador de Entrada

O processador de entrada está diretamente acoplado à rede local, comunicando-se através dela com as estações de trabalho. Também se comunica com o banco de Transputers. Sua principal função consiste no roteamento de pacotes entre as estações de trabalho e o banco.

Os diagramas de estrutura modular correspondentes às tarefas executadas no PE, aparecem na figura 4.12. A descrição de interface dos módulos componentes desses diagramas é apresentada no apêndice A4.

Algumas primitivas do mecanismo RPC são atendidas pelo processador de entrada. Entre essas, figuram as primitivas de controle de acesso ao sistema:

- > Login (nome_usuario, senha, *ide_sessão) - Inicia uma sessão no servidor. (*)
- > Logout (ide_sessão) - Encerra uma sessão no servidor. (*)

(*) projetado

4.12 Diagramas de Estrutura Modular PE (DEM-PE)

A lógica de controle no PE é expressa pelo algoritmo:

```

ENQUANTO (atender_requisições)
  SE (ReceberSPP_ET)
    SE (pacote com requisição válida)
      SE (requisição atendida pelo PE)
        atender requisição
      SENÃO
        rotear o pacote para o BT
    SENÃO
      descartar pacote
  SE (ReceberSPP_BT)
    SE (pacote com requisição/resposta válida)
      SE (requisição atendida pelo PE)
        atender requisição
      SENÃO
        rotear pacote para a ET
    SENÃO
      descartar pacote

```

4.3.4 Controlador de Módulo

Os módulos correspondentes às tarefas do CM podem ser vistos na figura 4.13. A descrição de interface desses módulos é apresentada no apêndice A5. As diversas tarefas são executadas concorrentemente utilizando-se os mecanismos intrínsecos da linguagem Occam.

As seguintes primitivas são atendidas pelo controlador de módulo:

- > IniciarLote (ide_lote) - Aplica sinal "reset" no lote de Transputers ide_lote. (+)
- > AnalisarLote (ide_lote) - Aplica sinais "reset" e "analyse" no lote de Transputers ide_lote. (+)
- > LerLote (ide_lote, nro_bytes) - Lê "nro_bytes" bytes do lote de Transputers ide_lote. (+)
- > EscreverLote (ide_lote) - Escreve "nro_bytes" bytes no lote de Transputers ide_lote. (+)

4.13 Diagramas de Estrutura Modular CM (DEM-CM)

4.13 Diagramas de Estrutura Modular CM (DEM-CM) cont ...

4.13 Diagramas de Estrutura Modular CM (DEM-CM) cont ...

4.13 Diagramas de Estrutura Modular CM (DEM-CM) cont ...

- > TestarErroLote (ide_lote) - Testa condição de erro no lote de Transputers ide_lote. (+)
- > AlocarLote (nro_proc, *ide_lote) - Aloca "nro_proc" Transputers. No caso de sucesso, devolve o endereço do lote. (+)
- > ConfigLote (ide_lote, *config) - Configura o lote ide_lote conforme a especificação dada em "config". Devolve a configuração real do lote, isto é, a configuração com os canais efetivamente interligados. (+)

(+) implementado

As cinco primeiras primitivas correspondem às rotinas necessárias à instalação do sistema de desenvolvimento TDS, substituindo as descritas no item 4.4.3. As duas primitivas restantes permitem o controle sobre os lotes de processadores. Somente usuários credenciados devem ter acesso direto à primitiva AlocarLote.

4.4 Instalação do Sistema de Desenvolvimento TDS

A instalação do sistema de desenvolvimento TDS introduz no sistema SPP ferramentas para o desenvolvimento de programas na linguagem Occam. Isso possibilita um mínimo de funcionalidade para o sistema.

4.4.1 Organização do Sistema de Desenvolvimento TDS

O sistema de desenvolvimento TDS é constituído de duas partes, uma delas executada no computador hospedeiro, o ISERVER, e a principal, o TDS, executada no processador raiz de uma rede de Transputers, conforme mostrado na figura 2.5. O ISERVER possibilita o carregamento de programas objeto no processador raiz da rede e pode permanecer ativo, atendendo requisições de serviços relativos a arquivos, vídeo e teclado provenientes do TDS ou de um programa do usuário.

A estrutura modular do ISERVER pode ser observada na figura 4.14. Os módulos representados por caixas com contornos tracejados são responsáveis pela comunicação do ISERVER com o processador raiz. Para comunicação através da rede local esses módulos sofreram as modificações descritas no item 4.4.3.

4.4.2 Protocolo SP*

O TDS comunica-se com o ISERVER através do protocolo SP e pacotes de dados com o formato apresentado na figura 4.10. Algumas regras desse protocolo incluem [IN90]:

- > No sentido TDS -> ISERVER o menor pacote tem 8 bytes e o maior 512 bytes.
- > O número de bytes no pacote deve ser par.
- > Cada requisição enviada para o ISERVER recebe uma resposta no mesmo protocolo. Nenhuma nova requisição é aceita antes que a resposta seja enviada.
- > Nos pacotes resposta o primeiro byte indica se a função foi executada com sucesso (byte = 0). Se a operação não foi executada com sucesso o byte retornado é diferente de zero. No caso da função não estar implementada, retorna-se o valor 1. Nas requisições, o primeiro byte indica a primitiva a ser executada.

As seguintes alterações no protocolo SP originam o protocolo SP*:

- > Modificação do ISERVER para garantir que o tamanho máximo do pacote SP* no sentido ISERVER -> TDS possa ser transmitido em um único pacote SPP.
- > Interpretação do primeiro byte do pacote SP*, proveniente de um lote, no controlador de módulo. No caso desse byte ser igual a zero, o segundo byte indica uma primitiva SPP, conforme é mostrado na figura 4.10.

4.14 Diagrama de Estrutura Modular ISERVER

4.4.3 Alterações no ISERVER-INMOS

As alterações necessárias no ISERVER, para a operação no TDS no sistema SPP, são:

- > alteração do protocolo SP, conforme descrito no item 4.4.2.
- > alteração dos módulos de comunicação representados por caixas com contornos tracejados na figura 4.14.

A função original dos módulos do ISERVER foi na maioria dos casos preservada, conforme descrição abaixo. Para os módulos OpenLink e CloseLink, a função inicial perde parcialmente o sentido. Isso se deve ao fato desses módulos estarem relacionados com o endereçamento físico dos circuitos de interface, originalmente previstos, entre o computador hospedeiro e a rede de Transputers. Os módulos TestRead e TestWrite perdem completamente o sentido, uma vez que o mecanismo de controle de fluxo de dados foi completamente alterado. Apesar de codificados, esses módulos não são utilizados na versão original do ISERVER, conforme pode-se observar na figura 4.14.

- > LINK OpenLink (char * Name) - Associa um identificador de canal ("link") a "Name" e correspondentes endereços de E/S. (x)
- > int CloseLink (LINK LinkId) - Termina a associação feita por OpenLink. (x)
- > int ResetLink (LINK LinkId) - Aplica sinal de "reset" no processador raiz da rede de Transputers. (x)
- > int AnalyseLink (LINK LinkId) - Aplica sinais de "reset" e "analyse" no processador raiz da rede de Transputers. (x)
- > int ReadLink (LINK LinkId, char *Buffer, unsigned int Count, int TimeOut) - Lê "Count" bytes do canal "LinkId" em "Buffer". Retorna o número de bytes lidos. (x)

- > int WriteLink (LINK LinkId, char *Buffer, unsigned int Count, int TimeOut) - Escreve "Count" bytes de "Buffer" no canal "LinkId". Retorna o número de bytes escritos. (x)
- > int TestError (LINK LinkId) - Testa condição de erro na rede de Transputers. (x)
- > int TestRead (LINK LinkId) - Testa se existe dado recebido pelo canal "LinkId". (-)
- > int TestWrite (LINK LinkId) - Testa se o canal "LinkId" está pronto para transmissão. (-)

(x) modificado

(-) suprimido

Conforme mostrado na figura 3.17, ao executar o TDS o servidor do sistema aparenta estar fisicamente alocado na estação de trabalho do usuário. Essa aparente contradição com o mecanismo RPC foi discutida no item 3.6.3 considerando-se o TDS uma primitiva com duração indeterminada. A nível de implementação, somente a primitiva "ReadLink" apresenta essa característica.

4.5 Considerações Finais

As fases de projeto e implementação do sistema servidor de processamento paralelo foram abordadas neste capítulo. Deve-se observar que a especificação apresentada no capítulo anterior não está totalmente projetada. Dos itens projetados alguns não passaram à fase de implementação.

Considerando-se os itens de hardware previstos no capítulo 3, não estão implementados o controlador e a rede de chaveamento do banco. Conforme discutido, esses itens são opcionais para sistemas com até 30 processadores. Dos itens de software previstos, estão implementados aqueles necessários para a operação do núcleo do servidor.

Tal como no restante do trabalho, foram utilizadas técnicas estruturadas no projeto e implementação do sistema. A documentação resultante dessas técnicas apresenta

especial relevância para a continuidade do desenvolvimento e para a fase de manutenção do sistema, uma vez que outros pesquisadores estão envolvidos. Parte dessa documentação, tal como as listagens dos programas fonte, não foi incluída neste trabalho para não torná-lo volumoso e de difícil leitura [TR91a] [TR91b] [TR91c].

No capítulo seguinte são apresentados os resultados obtidos nos testes do sistema. A sequência de testes foi estruturada de forma coerente com a metodologia empregada em todo o desenvolvimento. É apresentado um exemplo de utilização do núcleo básico do sistema.

CAPÍTULO 5

VALIDAÇÃO DO SISTEMA

5.1 Considerações Iniciais

As etapas de projeto e implementação do sistema servidor de processamento paralelo foram apresentadas no capítulo anterior. Não é possível, via de regra, definir nitidamente as fronteiras entre cada uma dessas etapas. O sistema servidor de processamento paralelo não constitui uma exceção.

Neste capítulo é feita a validação do sistema. Os testes realizados procuraram validar funcionalmente o sistema, inicialmente a nível modular. Testes de integração e operação completaram os testes modulares. Os testes dos principais subsistemas são apresentados, alguns desses diretamente relacionados com a implementação do software, particularmente com os módulos de mais baixo nível. Alguns desses módulos foram inicialmente desenvolvidos para os testes de hardware e, após pequenas adaptações, transformados em componentes do software.

No item 5.2 são apresentados os testes relativos ao hardware do sistema. Esses testes foram aplicados a partir dos subsistemas menores, a exemplo dos testes de software. Subsistemas maiores foram definidos a partir da combinação dos subsistemas já testados, culminando com o teste do sistema como um todo.

Os testes de software são apresentados no item 5.3. A seqüência dos testes inicia-se com os testes modulares, passa pelos testes de integração e termina com alguns testes de operação dos subsistemas principais.

O teste de operação do núcleo básico do sistema é apresentado no item 5.4, na forma de um exemplo de utilização. A seqüência das operações pode ser utilizada como um roteiro para a utilização do servidor.

Uma comparação do sistema proposto com outros sistemas baseados em Transputers é apresentada no item 5.5. No item 5.6 comenta-se a seqüência prevista de implementação. Também são feitas considerações sobre a evolução e a influência da nova série de componentes da família INMOS Transputer sobre o servidor de processamento paralelo proposto neste trabalho.

5.2 Testes do Hardware

O resultado dos testes dos principais subsistemas de hardware são apresentados. A realização desses testes foi realizada a nível funcional com a utilização de equipamentos e programas. Entre os equipamentos figuram osciloscópios digitais com memória e analisadores lógicos. Na maioria dos casos foram utilizados programas para excitar os circuitos sob teste, sendo as respostas observadas com os equipamentos mencionados.

5.2.1 Processador

Os testes de hardware aplicados nos processadores procuraram verificar o correto funcionamento dos seguintes subsistemas:

- > Processador Transputer
- > Canais de comunicação
- > Memória externa
- > Circuito de interface com o barramento de controle

Para o teste funcional do processador e canais de comunicação foi utilizado o programa CHECKOUT, fornecido pela Microway. Esse programa acompanha as placas compatíveis com a INMOS B004 comercializadas por essa companhia [MI87]. Esses testes foram repetidos em todas as placas implementadas para verificação de funcionamento.

O teste da memória externa dos processadores foi feito utilizando-se o programa TM fornecido pela empresa Logical Systems a título de exemplo de codificação de programas em linguagem montadora T800. Esse programa faz parte do sistema de desenvolvimento para linguagem C em Transputers [LO89a] [LO89b]. Algumas alterações foram efetuadas no código original para contabilização global dos erros. As seguintes operações de leitura e escrita em memória são realizadas em cada laço do teste:

- > preenchimento com o endereço da posição, leitura e comparação.
- > preenchimento com o complemento do endereço da posição, leitura e comparação.
- > preenchimento com 0x00000000, leitura e comparação
- > preenchimento com 0xFFFFFFFF, leitura e comparação
- > preenchimento com 0x55555555, leitura e comparação
- > preenchimento com 0xAAAAAAAA, leitura e comparação

Os testes da memória externa são importantes, uma vez que essa constitui o subsistema de projeto mais crítico do processador. Foram realizados diversos testes, em diferentes temperaturas e com diferentes conjuntos de circuitos integrados. As condições de teste e resultados obtidos estão sumarizados abaixo:

Teste 1 (1ro conjunto CIs, temperatura ambiente)

Tempo aproximado de teste (horas)	10
Número de Laços de teste	1587
Número de operações de leitura/escrita	1.997×10
Número de operações com erro	0

Teste 2 (2do conjunto CIs, temperatura ambiente)

Tempo aproximado de teste (horas)	11,5
Número de Laços de teste	1776
Número de operações de leitura/escrita	2.235×10
Número de operações com erro	0

Teste 3 (2do conjunto CIs, +/- 50 graus celsius)

Tempo aproximado de teste (horas)	5
Número de Laços de teste	795
Número de operações de leitura/escrita	1.000×10
Número de operações com erro	0

Totais nos 3 testes

Tempo aproximado de teste (horas)	26,5
Número de Laços de teste	4158
Número de operações de leitura/escrita	5.232×10
Número de operações com erro	0

Os circuitos de interface com o barramento foram implementados em dois dispositivos PAL. Esses dispositivos foram simulados no processo de compilação das equações lógicas utilizando-se o software PALASM, distribuído pela empresa AMD (Advanced Micro Devices). Uma vez programados, foi feita uma simulação elétrica para confirmar a correta operação dos mesmos.

5.2.2 Controlador de Módulo e Rede de Chaveamento

Os teste funcionais do controlador de módulo e da rede de chaveamento foram realizados a partir do programa TMOD executado no controlador de módulo. Esse programa foi desenvolvido na linguagem Occam, especificamente para a finalidade mencionada. Ele promove a excitação das partes do hardware sob teste. Quando possível, a resposta ao estímulo aplicado foi avaliada pelo próprio programa. Nos demais casos a observação da resposta foi feita utilizando-se instrumentos.

O usuário pode selecionar as funções de teste no TMOD através de menus. O menu principal apresenta as seguintes funções:

- 1-> AlocarProcessador
- 2-> ResetarLote
- 3-> AnalisarLote
- 4-> ResetarAnalisarLote

- 5-> TestarErroMódulo
- 6-> TestarErroProcessador
- 7-> HabilitarErroLote
- 8-> InibirErroLote
- 9-> SelecionarChave
- A-> ConectarCanais
- B-> DesconectarCanais
- C-> LerConexão
- D-> ResetarChave
- E-> MapearConexões
- F-> TestarCanalUsuário

As funções de teste 1, 2, 3, 4, 5, 6, 7 e 8 estão relacionadas com o barramento de controle. As funções 9, A, B, C, D e E estão relacionadas com a rede de chaveamento. A função F permite o teste de transmissão/recepção ("loop back") nos canais de usuário.

5.2.3 Interface de Comunicação PE/BT

O circuito da interface de comunicação PE/BT foi testado através do programa TLINK, desenvolvido na linguagem C para essa finalidade. Esse programa foi posteriormente utilizado em alguns testes do software do sistema.

As funções de teste disponíveis no TLINK são selecionadas através de um menu, contendo as seguintes opções:

- 1-> IniciarTeste
- 2-> InterromperTeste
- 3-> PulsarReset
- 4-> PulsarAnalyse
- 5-> LerErro
- 6-> TaxaTransferência
- 0-> Terminar

A função 1 executa um teste de transmissão/recepção ("loop back"), pelo canal implementado. As funções 3, 4 e 5 testam as portas de E/S relacionadas com os sinais RESET, ERROR e ANALYSE. A função 6 permite medir a taxa de transferência através do canal implementado ("loop back"). Note que a taxa medida depende da implementação do programa de teste utilizado para medi-la. Os valores apresentados são aproximados. Programas escritos em linguagem montadora, por exemplo, podem atingir maiores taxas de transferência de dados. O trecho de programa C que implementa o processo de transmissão/recepção no TLINK é mostrado abaixo:

```
for (i=0; i<1000000; i++)
{
    while (!(inp (0x153) & 0x01));
    outp (0x151, tbyte);
    while (!(inp (0x152) & 0x01));
    rbyte= (char) inp (0x150);
}
```

As seguintes taxas de transferência foram obtidas em algumas máquinas IBM/PC/AT compatíveis:

Tipo da Máquina		TT Kbytes/s	TT Kbits/s
IBM/PC	4.77 MHz	4.9	53.9
IBM/AT	286 8 MHz	18.5	203.5
IBM/AT	386 25 MHz (PE)	62.5	687.5

Essas taxas de transferência estão bem abaixo da taxa máxima para canais INMOS. Isso sugere a necessidade de um novo projeto para a interface de comunicação, a ser considerado em trabalhos futuros.

5.3 Testes do Software

Os testes efetuados no software do sistema procuraram, dentro das limitações existentes, validar funcionalmente cada módulo imediatamente após a implementação. A sequência de testes apresentada reflete a sequência de implementação do sistema. Essa abordagem foi adotada tendo em vista as dificuldades encontradas na implementação dos módulos de baixo nível, os quais interagem com o mecanismo de acesso ao meio físico da

rede local, a comunicação PE/BT, o barramento de controle do módulo de Transputers, a comunicação do CM com os diversos lotes de Transputers e a rede de chaveamento.

5.3.1 Software de Comunicação

O teste de comunicação através da rede local foi realizado visando a avaliar o mecanismo de acesso ao meio físico da rede local de computadores, "packet driver", a interface com esse mecanismo e a fila circular de recepção de pacotes implementada no PE. Foram utilizados dois computadores tipo IBM/PC conectados por interfaces para rede local Ethernet, como mostrado na figura 5.1.

Diversas dificuldades tiveram que ser contornadas para que o módulo de interface com o PD funcionasse a contento. A maior delas foi encontrada na passagem de parâmetros entre as tarefas executadas na ocorrência de uma interrupção de hardware, ocasionada pelo recebimento de um pacote, conforme ilustra a figura 3.16. Para a depuração dos módulos de software relacionados, foi utilizado um analisador lógico, os códigos em linguagem montadora correspondentes aos programas C envolvidos e utilitários para depuração de programas C.

O primeiro programa desenvolvido foi executado nos dois computadores utilizados no teste, podendo ser descrito pelo algoritmo abaixo:

```
ENQUANTO  character <> @
  SE character digitado
    transmite character
  SE character recebido
    ecoa character
```

O segundo programa permite, mediante seleção, a transmissão de cinco pacotes de dados de tamanhos distintos, conforme mostrado abaixo:

- 1-> dado = 'A' tamanho pacote = 125 bytes
- 2-> dado = 'B' tamanho pacote = 250 bytes
- 3-> dado = 'C' tamanho pacote = 500 bytes
- 4-> dado = 'D' tamanho pacote = 1000 bytes
- 5-> dado = 'E' tamanho pacote = 2000 bytes

5.1 Configuração de Hardware Utilizada para Testes de Software

ET	Estação de Trabalho
PE	Processador de Entrada
CM	Controlador de Módulo
PR	Processador Raiz de Lote
PE/BT	Interface PE/BT
T800	Processador
A	Operação Remota do TDS
B	Software do Controlador de Módulo

5.2 Subdivisão do Sistema para Implementação e Testes de Software

Esse programa foi utilizado para verificar a operação da fila circular de recepção. A Opção 5 foi utilizada para verificar a rejeição de pacotes pelo PD motivada por tamanho excessivo.

5.3.2 Mecanismo RPC

O esquema apresentado na figura 5.1 foi novamente utilizado para o teste do mecanismo RPC. Neste caso, porém, um dos computadores exerceu a função de servidor e o outro de cliente do sistema.

O mecanismo RPC foi testado utilizando-se um conjunto de programas que implementa o mecanismo e duas primitivas para tratamento de cadeias de caracteres. O menu de acesso a essas primitivas é mostrado abaixo:

- 1-> IntroduzirString
- 2-> InverterString
- 3-> EstatísticaString
- 4-> ComplementarSequência
- 5-> Terminar

Além de demonstrar a execução remota das primitivas para manipulação de cadeias, modificações no código dos programas de teste permitiram exercitar os seguintes aspectos do mecanismo RPC:

- > Verificação do mecanismo de "no máximo uma execução" ("at most once").
- > Verificação do mecanismo de não execução por "time out" após repetidas tentativas de requisição sem resposta.
- > Verificação no protocolo SPP da validade dos campos: (figura 4.10)
 - .tipo do protocolo
 - .requisição/resposta

5.3.3 Operação Remota do TDS

Uma vez testada a operação do mecanismo RPC, o sistema teve sua implementação dividida em duas etapas:

- > operação remota do TDS.
- > software do controlador de módulo.

Conforme pode ser observado na figura 5.2, essas etapas têm no processador de entrada o ponto comum para posterior integração.

A operação remota do sistema de desenvolvimento TDS utilizou um computador, ET, executando uma versão modificada do ISERVER e outro, PE, conectado por uma interface PE/BT a um processador Transputer executando o TDS. Os testes de operação remota do TDS possibilitaram a verificação das alterações no ISERVER. As primitivas relacionadas com a operação do TDS foram codificadas na linguagem C. Essas rotinas foram traduzidas para a linguagem Occam e transportadas para o controlador de módulo na integração do software relativo às etapas acima relacionadas.

5.3.4 Software do Controlador de Módulo

Para implementação e testes do software do controlador de módulo foi empregado o esquema apresentado na figura 5.2. Foram envolvidos o PE, o controlador de módulo e os processadores raiz dos lotes. Para contornar as dificuldades existentes, foi adotada uma série de procedimentos preliminares. Esses procedimentos visaram a validar o mecanismo de comunicação entre os diversos componentes do sistema, conforme ilustrado na figura 5.3. Cinco níveis de comunicação foram testados e as taxas de transferência observadas em cada um desses níveis estão indicadas na figura 5.3. Os seguintes programas foram utilizados:

PE **Processador de Entrada**

CM **Controlador de Módulo**

PR **Processador Raiz de Lote**

PE/BT **Interface PE/BT**

T800 **Processador**

Teste	Taxa de Transf.	Programas Utilizados
1	62.5 Kbytes/s	TLINK
2	43.5 Kbytes/s	TLINK/SS (modificado)
3	33.3 Kbytes/s	TLINK/SS
4	33.3 Kbytes/s	TLINK/SS
5	26.3 Kbytes/s	TLINK/SS/ECO

5.3 Esquema para Testes do Software do Controlador de Módulo

- > TLINK executado no PE e descrito no item 5.2.3
- > SS executado no CM, descrito pelo trecho de código:

```

PAR
  WHILE dado.dir <> '@'
    SEQ
      from.PE ? dado.dir
      TransmitirByteUsuário (dado.dir)
  WHILE dado.inv <> '@'
    SEQ
      ReceberByteUsuário (dado.inv)
      to.PE ! dado.inv

```

- > ECO - executado nos PR, descrito pelo trecho de código:

```

  WHILE dado <> '#'
    SEQ
      from.CM ? dado
      to.CM ! dado

```

Para a realização do teste de nível 5 foi adotada a seguinte sequência de passos para o carregamento e execução dos programas:

- > execução do programa SS, a partir do ISERVER, configurando a rede de chaveamento do módulo para fazer conexão entre um canal de usuário com um processador raiz.
- > interrupção do ISERVER (ctrl-c).
- > carregamento do programa echo no PR, utilizando o ISERVER sem a opção /SR ("reset"), roteado através da rede de chaveamento pelo programa SS, em execução no CM.
- > utilização do programa TLINK.

Procedimento similar foi adotado para a implementação e teste de todo software do controlador de módulo. Alterações sucessivas no programa SS deram origem ao software mencionado. Em cada etapa, um novo módulo foi introduzido no sistema, sendo

executados testes a nível modular e de integração. Esse procedimento está de acordo com a filosofia de implementação e teste adotada.

5.4 Exemplo de Utilização do Sistema em Ambiente TDS

Os procedimentos necessários para a utilização do servidor de processamento paralelo, utilizando o sistema de desenvolvimento TDS, são sumarizados neste item. Na verdade, uma vez alocados os processadores e configurado o lote de Transputers, os procedimentos são análogos aos adotados na operação do sistema TDS utilizando-se uma rede de processadores Transputers e um computador hospedeiro dedicados.

Deve-se observar que o software descrito não completa a versão inicial do sistema definida no capítulo 3, uma vez que não foram implementados as primitivas para controle de abertura e encerramento das sessões do sistema. Os identificadores de sessão foram codificados como constantes, para a execução do teste.

As primitivas descritas no capítulo 4 foram testadas individualmente e integradas para a execução do exemplo descrito em seguida.

Conforme descrito no capítulo 2, o sistema de desenvolvimento TDS, na versão TDS3 [IN90], dá suporte aos seguintes tipos de programas:

> EXE

> PROGRAM

> PSP

> SHP

Foi utilizado um programa ordenador de cadeias, apresentado como exemplo em [IN90], para o teste de operação do sistema servidor de processamento paralelo. Essa opção é justificada uma vez que o programa é simples, atende os objetivos propostos e está codificado em cada um dos tipos de programa mencionados acima. Representa também

um bom exemplo de programação paralela utilizando Occam e o TDS, principalmente nos aspectos relacionados com E/S e encerramento controlado de programas paralelos.

O programa de ordenação faz uso de um "pipeline" de processos idênticos para a ordenação de uma cadeia de caracteres. Cada elemento do "pipeline" executa o trecho de código reproduzido abaixo:

```
PROC element (CHAN OF letters input, output)
  INT highest, next:
  BOOL going, inline:
  SEQ
    going := TRUE
    WHILE going
      input ? CASE
        terminate
          going := FALSE
          letter; highest
          SEQ
            inline := TRUE
            WHILE inline
              input ? CASE
                letter; next
                IF
                  next > highest
                    SEQ
                      output ! letter; highest
                      highest := next
                    TRUE
                      output ! letter; next
                end.of.letters
              SEQ
                inline := FALSE
                output ! letter; highest
            output ! end.of.letters
          output ! terminate
  :
```

Observa-se no código apresentado o algoritmo de ordenação utilizado. Também pode-se observar a utilização de um protocolo com campos alternativos ("variant protocol"), para a implementação de controle do fim de cada cadeia e do fim da execução do programa.

O teste de operação do sistema foi realizado a partir de quatro estações de trabalho, utilizando o servidor de processamento paralelo para a edição, compilação e execução do programa ordenador nos seguintes modos:

- > Estação 1 -> EXE (1 processador)
- > Estação 2 -> PSP (4 processadores)
- > Estação 3 -> SHP (1 processador)
- > Estação 4 -> SHP (4 processadores)

O desenvolvimento de programas "PROGRAM" somente é possível se forem adotados os procedimentos descritos em 5.6.1.

A sequência inicial de primitivas, descritas no capítulo 4, executadas nas estações foi:

- > AlocarLote
- > ConfigLote (somente estações 2 e 4)

A configuração resultante do lote alocado pelo utilitário ConfigLote, foi utilizada para a alteração dos programas fonte nas estações 2 e 4.

Uma vez alocado o lote, foi executado o sistema de desenvolvimento TDS. A sequência procedimentos para edição, geração de código objeto, carregamento e execução de cada um dos tipos de programas não sofre alterações em relação àquela descrita no manual do TDS [IN90].

5.5 Comparação com Outros Sistemas Baseados em Transputers

Arquiteturas MIMD com memória distribuída, representadas pelos denominados multicomputadores, constituem o principal objetivo de sistemas computacionais baseados

em Transputers. A principal característica dessa classe de arquiteturas reside na implementação do mecanismo para comunicação entre os processadores. A utilização de redes de comunicação tem como aspecto mais importante a topologia das conexões, que pode variar desde o caso ideal, redes totalmente conectadas, até redes fixas, como é o caso de hipercubos.

O número de sistemas acadêmicos ou comerciais utilizando Transputers tem aumentado. A facilidade para conexão dos processadores permite que o usuário, familiarizado, possa implementar uma arquitetura específica para a sua aplicação, a partir de módulos básicos, tais como processadores, redes de chaveamento e interfaces de comunicação. Essa filosofia é seguida por alguns sistemas comerciais, entre os quais o MEIKO [ME90]. Muitos usuários potenciais não têm interesse específico em processamento paralelo. Nesse sentido, a versatilidade do sistema perde importância em função da simplicidade de utilização.

A elevada especialização dos componentes da família Transputer não permite muitas variações na maneira como podem ser utilizados, fazendo com que as arquiteturas disponíveis apresentem muitas características em comum.

A utilização de sistemas baseados em Transputers através de redes locais tem sido relatada na bibliografia [NI88a] [ME90] [HA90]. Normalmente esse serviço é provido pela utilização de um computador hospedeiro multiusuário conectado em uma rede local. O servidor apresentado em [HA90] está associado ao paradigma de programação "processor farm" [HE89a], perdendo a característica de sistema para uso geral.

Na tabela 5.1 é feita uma comparação do sistema SPP com os dois sistemas baseados em Transputers descritos no capítulo 1.

Da tabela 5.1 e das considerações feitas, pode-se observar que o servidor de processamento paralelo incorpora as características dos sistemas existentes, numa arquitetura definida. Deve-se notar que a filosofia modular em que se baseia o sistema computacional MEIKO, possibilita a definição de sistemas semelhantes ao servidor de processamento paralelo, bem como qualquer outro baseado em Transputers.

	Sistema SPP	Esprit P1085	Meiko CS
Número Máximo de T800	960 Expansível	1024 Expansível	depende da config. (1)
Quant. Memória por Processador	1 / 4 Mb	256 Kb / 4 Mb	256 Kb / 64 Mb
Topologia Rede Chaveamento	programável	programável	depende da config.
Computador Hospedeiro	IBM/AT compatível	diversos	diversos
Natureza de Utilização	Propósito Geral	Propósito Geral	Propósito Geral
Canais para ET Locais	Padrão	--	Possível
Operação Multiusuária	Padrão	Prevista	Possível
Operação em Rede Local	Software Próprio	--	Software Hospedeiro
Linguagens	Diversas (2)	Diversas	Diversas CS tools

(1) Apesar do número não ser limitado, a interligação entre os processadores apresenta os mesmos compromissos que nas outras arquiteturas.

(2) Somente Occam já instalada.

Tabela 5.1 Comparação Entre Sistemas Baseados em Transputers.

5.6 Evolução do Sistema

Além da evolução prevista para o sistema, consequência direta das partes ainda não implementadas e eventuais manutenções, uma série de alterações deve ser considerada com a introdução dos novos componentes da família Transputer. Esses componentes,

conforme apresentado no capítulo 2, incluem um novo processador (T9000), e apresentam características que têm grande influência sobre a arquitetura apresentada neste trabalho.

5.6.1 Aspectos de Hardware

A expansão do sistema SPP figura como a primeira consideração quanto à evolução do hardware. Novos processadores deverão ser construídos até o limite de 30 processadores por módulo. A adição de processadores acima desse limite implica na implementação de um novo controlador de módulo e do controlador de banco. Isso provocará modificações no software atualmente desenvolvido.

O controle do balanço de carga entre os processadores pode ser útil na avaliação de programas executados em lotes de Transputers. Pretende-se incluir no sistema hardware específico para essa finalidade.

A nova série de componentes da família INMOS - Transputer inclui uma série de características que influenciarão futuras versões da arquitetura do sistema SPP apresentada neste trabalho. Entre essas características, destacam-se [IN91]:

- > Implementação de canais virtuais gerenciados por um processador dedicado interno ao Transputer T9000.
- > Canais específicos nos dispositivos para controle e configuração.

A primeira característica possibilita isolar a topologia física da rede de Transputers da sua topologia lógica. A adoção de redes com topologia fixa, tal como hipercubos, parece ser bastante atrativa. Uma análise mais cuidadosa deve entretanto considerar os atrasos introduzidos em cada operação de roteamento.

A segunda característica mencionada torna inadequado o barramento de controle utilizado na versão atual do sistema, baseada em Transputers T800.

A implicação desses fatores sobre a organização da rede de chaveamento adotada merece estudos adicionais. As chaves empregadas são incompatíveis com o novo protocolo

de comunicação dos canais e novos dispositivos, tal como a chave C104, devem ser utilizados.

Levando-se em consideração as observações acima, pode-se definir uma arquitetura de um novo banco, específica para os novos Transputers. Esse banco deverá ser conectado ao banco desenvolvido utilizando-se redes de chaveamento apropriadas e conversores de protocolo, tal como o disponível no circuito integrado C100. Essa solução é sugerida em [IN91] e permitirá que o sistema e os usuários incorporem de maneira gradual os novos dispositivos.

O servidor de processamento paralelo não permite o controle dos sinais RESET, ERROR e ANALYSE de um lote de Transputers no modo subsistema. Conseqüentemente, programas do tipo PROGRAM não podem ser executados. É possível o desenvolvimento desse tipo de programas utilizando-se um processador local, executando TDS, e um lote de transputers remoto, controlado como um subsistema. Essa configuração é mostrada na figura 5.4. O projeto da placa de processadores do banco prevê essa utilização. Mediante substituição das PALs que implementam a interface com o barramento de controle, a placa processadora se torna compatível com a placa INMOS B004 [IN86], podendo ser utilizada como processador local. O programa ISERVER deve ser convenientemente modificado e o mecanismo RPC incluir primitivas que possibilitem esse modo de operação.

5.6.2 Aspectos de Software

Duas versões de software deverão coexistir ao longo da vida útil do sistema SPP. Uma adequada para sistemas com até 30 processadores, sem controlador de banco, e outra para sistemas com mais de um módulo de Transputers. A segunda versão será implementada tão logo o número de processadores disponíveis a torne indispensável.

Para tornar o sistema SPP plenamente operacional, diversas primitivas deverão ser introduzidas ao longo da sua operação. Essas primitivas podem ser agrupadas nas seguintes classes:

- 1 Processador Transputer**
- 2 Interface PE/BT**
- 3 Interface Ethernet**

5.4 Configuração para Execução de Programas em Lotes Controlados pelo TDS

- > Manutenção do Sistema - Funções de configuração do sistema efetuadas por usuários com permissões especiais (gerentes), tal como o cadastramento de usuários. (#)
- > Controle do Sistema - Funções de controle do sistema, tais como interrupção de operação e reiniciação do sistema ou partes desse efetuadas por usuários com permissões especiais. (#)
- > Estatística do Sistema - Funções para análise de dados estatísticos da utilização do sistema, tais como número de horas de operação, número médio de processadores alocados por usuário, horário de login e logout, entre outras. (#)
- > Consultas - Parâmetros do sistema, tais como, usuários em sessão e número de processadores disponíveis. (#)

(#) previsto

Algumas das novas primitivas atendidas pelo PE podem não apresentar tempo de execução compatível com o mecanismo de controle atualmente implementado. Uma das maneiras de contornar esse problema consiste na adoção de um sistema operacional multitarefas para o PE, tal como o Microsoft OS/2 [MI90]. Uma outra alternativa consiste na implementação de um núcleo multitarefas específico. Essa solução permite otimizar o desempenho do sistema. Núcleos multitarefas dessa classe podem ser encontrados em [TA91] e [GR89]. Essas duas implementações diferem quanto ao mecanismo de controle de concorrência adotado. Em [TA91] esse mecanismo é implementado de forma não preemptiva, exigindo que as tarefas tenham um comportamento cooperativo. Um mecanismo preemptivo tradicional é apresentado em [GR89].

O processo de encerramento de uma sessão do sistema é determinado unicamente por um comando explícito do usuário. Nos casos de falha de hardware ou mesmo esquecimento do usuário, é interessante dispor de um mecanismo que proceda o encerramento da sessão por iniciativa do sistema. Uma das formas de implementar um mecanismo desse tipo faz uso de pacotes transmitidos simultaneamente pelo servidor, em intervalos de tempo regulares, para todas estações ("broadcast"). As estações devem, ao receber um pacote desse tipo, fazer acesso a uma primitiva do sistema informando sua

atividade. Caso o servidor não receba a resposta de alguma estação, a sessão correspondente é compulsoriamente encerrada por ocasião da próxima transmissão. A implementação desse mecanismo é facilitada com a utilização de um núcleo multitarefas no processador de entrada.

Não existe na implementação atual do sistema um mecanismo definido para a sua iniciação. As seguintes seqüências de iniciação automáticas são sugeridas e deverão ser futuramente implementadas:

Estações de Trabalho

- > Auto Teste
- > Carga do sistema operacional da estação
- > Teste da interface Ethernet e determinação do endereço físico
- > Carga do PD
- > Carga do sistema operacional da rede local (opcional)

Processador de Entrada

- > Auto Teste
- > Carga do sistema operacional
- > Teste da interface Ethernet e determinação do endereço físico
- > Carga do PD
- > Carga do sistema operacional da rede local (opcional)
- > Carga de software para teste, incluindo CB, CM, rede de chaveamento e condição operacional dos processadores
- > Iniciação das variáveis do sistema em função do resultado dos testes do BT

5.7 Considerações Finais

Neste capítulo são descritos os testes aplicados nos principais subsistemas do servidor de processamento paralelo e os resultados obtidos. A seqüência de aplicação dos testes espelha a seqüência de implementação do sistema, consequência direta da metodologia empregada nessas etapas do desenvolvimento.

Os testes de hardware e software foram feitos a nível funcional. Na maioria dos casos, os testes resultaram em dados qualitativos, tal como a verificação de funcionamento do módulo e a ausência de efeitos colaterais.

Entre os aspectos de hardware considerados em futuras versões do sistema, a interface de comunicação PE/BT tem especial importância. A implementação atual resulta em um gargalo na taxa geral de comunicação do sistema, conforme os resultados apresentados no item 5.2.3. É necessária uma nova implementação dessa interface para contornar esse inconveniente. Essa implementação deverá dispor de um processador dedicado e utilizar canais de acesso direto à memória para maximizar a taxa de transferência de pacotes.

Conforme pode ser observado neste capítulo, o sistema servidor de processamento paralelo não apresenta nesse estágio do seu desenvolvimento uma interface adequada com o usuário. Sua utilização, por usuários não diretamente ligados ao projeto, deverá acontecer tão logo sejam implementadas as rotinas para controle do acesso ao sistema. Espera-se com isso contar, já nas etapas finais de implementação do sistema, com a análise crítica dos usuários.

A comparação do sistema proposto com outros sistemas e as considerações feitas quanto a sua evolução, permitem afirmar que o servidor de processamento paralelo proposto reflete o estágio atual de desenvolvimento de sistemas baseados na atual geração de dispositivos da família Transputer.

No próximo capítulo são apresentadas as contribuições, continuidade, possibilidades para novas pesquisas e comentários finais deste trabalho.

CAPÍTULO 6

CONCLUSÕES

6.1 Considerações Iniciais

Nos capítulos anteriores foram apresentadas as diversas etapas de desenvolvimento deste trabalho que culminaram com a validação do protótipo do sistema servidor de processamento paralelo. O processo de desenvolvimento do sistema foi baseado na especificação inicial, grande parte da qual passou pela etapa de projeto. A implementação de um núcleo básico permitiu a avaliação da exequibilidade do sistema.

Neste capítulo apresentam-se as conclusões obtidas após a análise das atividades e resultados de cada uma das etapas do desenvolvimento. Especial atenção é dada à definição de trabalhos complementares e adicionais ao término deste, de acordo com um dos objetivos básicos do sistema, ou seja, incentivar pesquisas nas áreas de conhecimento envolvidas.

No item 6.2 são salientadas as contribuições dadas por este trabalho. Os projetos complementares, adicionais e motivados pela conclusão deste estão relacionados no item 6.3.

No item 6.4 apresentam-se os comentários finais do trabalho. Procura-se nesse item avaliar os resultados obtidos, de acordo com os objetivos e a motivação inicialmente estabelecidos. O impacto do trabalho sobre novas possibilidades de pesquisa também é avaliado.

6.2 Contribuições Deste Trabalho

A desenvolvimento do servidor de processamento paralelo representa uma solução alternativa em relação aos sistemas baseados em Transputers existentes, ressaltando-se os seguintes aspectos:

- > O desenvolvimento de um projeto com as características do descrito neste trabalho constitui uma iniciativa de vanguarda no âmbito nacional. O domínio de cada fase do desenvolvimento permitirá a evolução contínua do sistema, equiparável àquela resultante do esforço empreendido, em outros países, na pesquisa de sistemas baseados em Transputers.
- > O sistema comporta-se como um todo bem definido, sem entretanto apresentar restrições de utilização dignas de nota, ao contrário de sistemas com arquiteturas rígidas ou excessivamente flexíveis e indefinidas.
- > A disponibilidade de canais ETL permite associar uma interface consistente com necessidades particulares de E/S, principalmente aquelas que demandam taxas elevadas de transferência de dados. Cada canal ETL possibilita uma taxa de transferência de até 20 Mb/s. Essa característica contribui para tornar o sistema flexível, contribuindo também para a facilidade de operação.

Outra contribuição importante do trabalho reside na abordagem empregada no processo de desenvolvimento, integrando os aspectos de hardware e software. Apesar de diluída no texto, essa abordagem fez uso de diversas técnicas, convenientemente adaptadas e integradas, formando um arcabouço que pode ser utilizado com igual sucesso no desenvolvimento de sistemas com características similares. Dois aspectos dessa abordagem são dignos de nota:

- > A utilização de metodologias estruturadas em todo o processo de desenvolvimento. Essas metodologias resultaram, além da otimização de todo o desenvolvimento, em um conjunto de documentos considerados essenciais para a continuidade do projeto. É importante mencionar que, além de complexo, participam do desenvolvimento do sistema diversos pesquisadores. A documentação gerada é fundamental para o desenvolvimento em equipe.

- > A seqüência e interação das etapas de desenvolvimento integrado de hardware e software. A especificação foi feita tendo em vista uma versão inicial do sistema. O projeto seguiu a especificação, com poucas omissões. A implementação e testes, entretanto, procuraram validar a especificação e projeto através da implementação de parte da especificação, criando-se condições para revisões e melhoramentos do sistema. Técnicas "top down" foram utilizadas nas etapas de especificação e projeto, e "bottom up" nas etapas de implementação e testes, permitindo o desenvolvimento conjunto e reutilizável de programas para teste de hardware e software.

A abordagem semântica dada no tratamento do mecanismo RPC, permitindo a existência de elementos com comportamento dual (cliente/servidor), sem contrariar o paradigma envolvido, estende o conceito para casos similares ao deste trabalho.

6.3 Sugestões Para a Continuidade do Trabalho

Conforme mencionado anteriormente, a atividade de pesquisa relacionada com as áreas de conhecimento associadas ao desenvolvimento do servidor de processamento paralelo, apenas se inicia com o encerramento deste trabalho. Outros trabalhos deverão ser desenvolvidos para que o servidor apresente as características operacionais desejadas. Alguns trabalhos devem fazer uso dos recursos computacionais tornados disponíveis. Outros trabalhos de pesquisa, não diretamente relacionados com o servidor, foram idealizados durante o desenvolvimento e estão previstos. Neste item são apresentadas descrições sucintas desses trabalhos. Enquanto alguns deles estão em andamento, outros deverão ser desenvolvidos dentro de programas de iniciação científica, mestrado e doutorado. Muitos desses trabalhos estão baseados no item 5.6, que deve ser consultado como referência adicional.

6.3.1 Interface PE/BT via DMA com Processamento Local de Pacotes

Conforme comentado no capítulo 5, a interface PE/BT correntemente em uso não apresenta um desempenho compatível com a velocidade do canal INMOS de comunicação entre o PE e o BT. Esse projeto envolve o desenvolvimento do hardware e do software de

uma interface PE/BT dotada de processamento local de pacotes, utilizando um canal de DMA do PE (IBM/AT compatível), para a transferência de dados.

6.3.2 Instalação de Linguagens Adicionais

Algumas restrições da linguagem Occam tornam difícil a sua utilização para a solução de determinados problemas, por exemplo, os que apresentam tipos de dados complexos. A sua utilização também não é adequada quando o usuário dispõe de programas escritos de forma seqüencial em C, PASCAL ou FORTRAN e deseja modificar ligeiramente o código original para exploração de algum paralelismo. Nesses casos é importante que o servidor disponha de compiladores para essas linguagens.

6.3.3 Desenvolvimento de Mecanismos para Controle de Sessões, Acesso e Supervisão do Sistema

Os mecanismos básicos, abertura e encerramento das sessões com o servidor, estão correntemente sendo implementados. Outros mecanismos, tais como o encerramento compulsório das sessões e utilitários para gerenciamento do sistema, mencionados no capítulo 4, deverão ser implementados para melhorar as condições de operação e supervisão do sistema.

6.3.4 Utilização do PE como Servidor de Arquivos Dedicado ao Sistema

A disponibilidade de memória de massa no PE pode ser convenientemente aproveitada para o armazenamento dos sistemas de desenvolvimento e linguagens disponíveis, liberando o espaço ocupado nas estações de trabalho. Outro benefício é a redução de tráfego na rede motivada pelo carregamento remoto dos utilitários mencionados.

6.3.5 Gerador de Aplicações Orientado a Objetos

A programação orientada a objetos se situa em um nível semântico superior ao de outros paradigmas, tal como a programação com tipos abstratos de dados. Objetos modelam entidades reais, desempenhando ações essencialmente paralelas. Esta proposta de trabalho envolve a investigação de ferramentas para geração automática de código, em uma linguagem paralela adequada, de programas paralelos especificados segundo as técnicas de programação orientada a objetos. Em princípio, o sistema deve cuidar da identificação do paralelismo em cada aplicação e da distribuição ótima das tarefas entre os processadores disponíveis.

6.3.6 Subsistema para Avaliação do Balanço de Carga Entre os Processadores de um Lote

O balanço de carga entre os processadores de um lote é um fator importante para a obtenção de máximo desempenho [NI88b]. A investigação desse parâmetro exige o projeto de hardware e software específicos.

6.3.7 Módulo de Transputers Baseado no T9000

Conforme mencionado no capítulo 5, a utilização dos componentes da família Transputer T9000 implica no projeto de um novo módulo, conectado ao original através de dispositivos adequados para conversão de protocolo. Esse novo módulo deve ser utilizado de forma integrada ao sistema desenvolvido.

6.3.8 Estação de Trabalho Local Gráfica

A disponibilidade de canais ETL possibilita a conexão, sem restrição na velocidade de comunicação, de dispositivos que demandem altas taxas de transferência de E/S. Entre esses dispositivos, uma estação de trabalho local gráfica permite a utilização do servidor na pesquisa e desenvolvimento de algoritmos paralelos para computação gráfica.

6.3.9 Canais de Comunicação RS232 Acoplados a Canais ETL

A disponibilidade de canais RS232 ligados diretamente ao servidor permite que os usuários possam ter acesso a um grande número de dispositivos, tais como impressoras, traçadores de gráficos, mesas digitalizadoras, entre outros. Um canal ETL poderá dar origem a diversos canais RS232, através de dispositivos adequados para multiplexação de canais.

6.3.10 Canais de Comunicação Ethernet Acoplados a Canais ETL

A comunicação do banco de Transputers diretamente com outros sistemas dispondo de interface Ethernet pode ser feita pela conversão de canais ETL em canais Ethernet. Essa conversão é possível com o desenvolvimento de hardware e software específicos.

6.3.11 Mecanismos para Iniciação, Testes e Tratamento Tolerante a Falhas

O processo de iniciação dos parâmetros operacionais das estações de trabalho e servidores do sistema deve ser feita após a realização de testes de operação nos subsistemas componentes, conforme descrito no capítulo 5. A ocorrência de erros no banco de Transputers, durante sua operação, pode receber tratamento tolerante a falhas, aproveitando a redundância intrínseca existente para permitir a degradação controlada do sistema.

6.3.12 Ferramentas de Apoio para a Utilização de Canais Virtuais

A disponibilidade de canais virtuais para programação de lotes de Transputer elimina a restrição imposta pelo número limitado de canais físicos e pela topologia da rede. Apesar da influência sobre o desempenho que esses expedientes podem apresentar, a tarefa do programador fica bastante simplificada com a utilização dessas ferramentas [SU90].

6.3.13 Ferramenta para Remapeamento de Canais em Programas Fonte

A rede de chaveamento utilizada no servidor de processamento paralelo apresenta o inconveniente de não permitir a identificação dos canais físicos envolvidos nas conexões entre processadores de um lote de Transputers. Isso significa que a requisição de uma mesma topologia pode resultar em lotes diferentes, em sessões distintas no servidor. Os programas devem ser compilados a cada sessão, com os comandos de mapeamento de canais físicos alterados de acordo com o lote alocado. Esse procedimento pode ser simplificado com a disponibilidade de uma ferramenta que promove a alteração automática dos programas fonte.

6.3.14 Levantamento e Análise de Dados Estatísticos da Operação do Sistema

Para que seja possível a análise estatística do sistema são necessários dados estatísticos sobre sua operação, que deverão ser coletados por software especificamente desenvolvido para essa finalidade.

6.3.15 Avaliação e Otimização do Desempenho do Sistema

Um passo subsequente à análise de dados do sistema consiste na avaliação e otimização do seu desempenho. Diversas partes do sistema são susceptíveis a esse estudo. Algumas dessas partes podem ter seus parâmetros operacionais otimizados enquanto outras podem exigir um novo projeto e implementação.

6.3.16 Ferramentas de Apoio à Programação Baseadas em Paradigmas

A adoção de paradigmas de programação [HE89a] pode simplificar a utilização do sistema e o desenvolvimento de ferramentas para a geração automática de código. Esses paradigmas introduzem critérios para paralelização, sendo adequados para amplas classes de problemas.

6.3.17 Suporte de Software para Reconfiguração Dinâmica da Topologia dos Lotes

Conforme mencionado no capítulo 4, a reconfiguração dinâmica de redes de Transputers pode representar uma alternativa importante na busca de desempenho ótimo e merece ser investigada. A adição de primitivas específicas no mecanismo RPC possibilita que o servidor atenda a esse requisito.

6.3.18 Núcleo Multitarefa para o PE

Muitas funções implementadas ou previstas para o PE podem ser otimizadas pela disponibilidade de um sistema multitarefa. Um núcleo multitarefa representa uma alternativa à adoção de um sistema operacional multitarefa, condizente com a filosofia de desenvolvimento de manter o controle sobre cada um dos componentes do servidor de processamento paralelo.

6.3.19 Instalação e Testes do Mecanismo para Compartilhamento do Meio Físico

O compartilhamento do meio físico da rede local foi previsto e implementado com a utilização do "Packet Driver" NCSA. Tão logo os sistemas envolvidos estejam totalmente operacionais, um esforço deverá ser empreendido para avaliar o comportamento global do sistema.

6.3.20 Desenvolvimento de uma Interface de Comunicação com Dispositivos de E/S Padrão Através de Canais ETL

Canais ETL representam uma particularidade importante do sistema. A especificação de uma interface genérica para comunicação com dispositivos de E/S, através do desenvolvimento de um conjunto de primitivas apropriadas, permite a utilização de inúmeros dispositivos de forma homogênea e transparente. Alguns dispositivos, tal como terminais de vídeo gráficos, podem ser considerados recursos permanentes do sistema sujeitos a compartilhamento.

6.4 Comentários Finais

Uma arquitetura baseada em Transputers, adequada para processamento paralelo foi proposta implementada e testada com sucesso. Durante o desenvolvimento do projeto, cuidou-se de aspectos relacionados com a facilidade de utilização do sistema e compartilhamento de recursos. Dois objetivos básicos foram especificados. Gerar recursos computacionais para pesquisa nas áreas de sistemas distribuídos e processamento paralelo, representa o primeiro deles. O segundo objetivo consiste em tornar disponível uma máquina paralela aos seus potenciais usuários. Baixo custo e domínio tecnológico, propiciados pelo desenvolvimento integral do projeto, constituem objetivos complementares.

Analisando-se os resultados obtidos, acredita-se que foram materializados os objetivos que levaram à realização desse trabalho. O compromisso assumido com o prosseguimento da pesquisa, um dos objetivos do servidor, está estabelecido pelo grande número de trabalhos sugeridos no item anterior, alguns dos quais em andamento. Deve-se notar que somente foram incluídos trabalhos nas áreas de conhecimento relacionadas com o desenvolvimento do servidor. O interesse inicial demonstrado por pesquisadores de outras áreas indica que o projeto atingiu o segundo objetivo acima mencionado.

REFERÊNCIAS BIBLIOGRÁFICAS

- AL89 ALMASI, G.S.; GOTTLIEB, A. *Highly Parallel Computing*. US, Benjamin/Cummings, 1989.
- AM90 AMERICA, P.H.M.; HULSHOF, B.J.A.; ODIJK, E.A.M.; et al. Parallel Computers for Advanced Information Processing. *IEEE Micro*, v.10, n6, p.12-16, Dec., 1990.
- AN90 ANGENIOL, B. PYGMALION: ESPRIT II Project 2059, Neurocomputing. *IEEE Micro*, v.10, n.6, p.28-31, 99-102, Dec., 1990.
- AS86 ASKEW, C. R. *Parallel Computing Using Transputers or Occam (or Both)*. Southampton, Department of Physics/Un. of Southampton, 1986. (Internal Report)
- AS87 ASKEW, C. *What can the Transputer do for me? "Introduction to Transputer Hardware"*. Southampton, Un. of Southampton, 1987. (Internal Report)
- BA89a BAL, H.E.; STEINER, J.G.; TANENBAUM, A.S. Programming Languages for Distributed Systems. *ACM Computing Surveys*, v.21, n.3, p. 261-322, Sept., 1989.
- BA89b BACON, J.M.; LESLIE, I.M.; NEEDHAM, R.M. *Distributed Computing With a Processor Bank*. Computer Laboratory/Cambridge University, 1989. (Technical Report)
- BE89 BELL, G. The Future of High Performance Computers in Science and Engineering. *Communications of the ACM*, v.32, n.9, p.1091-101, Sept., 1989.
- BI84 BIRREL, A.; NELSON, B.J. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, v.2, n.1, p.39-59, Feb., 1984.
- BR89 BROOKES, G.R.; STEWART, A.J. *Introduction to OCCAM 2 on the Transputer*. Macmillan, 1989.
- BR90a BRAWER, S. *Introduction to Parallel Programming*. New York, Academic Press., 1989.
- BU88 BURNS, A. *Programming in OCCAM 2*. New York, Addison Wesley, 1988.
- CA86 CAMERON, J.R. An Overview of JSD. *IEEE Transactions on Software Engineering*, v.SE-12, n.2, p.222-40, Feb., 1986.
- CA89 CARRIERO, N. GELERNTER, D. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Computing Surveys*, v.21, n.3, p.323-57, Sept., 1989.
- CH90 CHEN, D.K.; SU, H.M.; YEW, Pen-Chung. The Impact of Synchronization and Granularity on Parallel Systems. *IEEE*, p.239-48, 1990.
- CL78 CLARK, D.D.; POGRAN, K.T.; REED, D.P. An Introduction to Local Area Networks. *Proceedings of the IEEE*, v.66 n.11, p.1497-517, Nov., 1978.

- CL89a USER Documentation for the Packet Driver Collection. Version 1.09, US. Clarkson University, 1989.
- CL89b CONCURRENT Novell Netware and TCP/IP Usage. A Case Study. Packet Driver Collection v.1.09, US. Clarkson University, 1989.
- CL90 CLARE, R.C. *Parallel Computing at UEA with the Meiko Computing Surface. Version 1.01.* East Anglia, University of East Anglia, Computing Centre, 1990. (Internal Report)
- CO88 COULOURIS, G.F; DOLLIMORE, J. *Distributed Systems - Concepts and Design.* Londres, Addison-Wesley, 1988.
- CO91 CORCORAN, E. Calculating Reality. *Scientific American*, v.264, n.1, p.100-9, Jan., 1991.
- CU88 CURRIE, W.S. LANs Explained - *A Guide to Local Area Networks.* U.K. Ellis Horwood, 1988.
- DU90 DUNCAN, R. A Survey of Parallel Computer Architectures. *IEEE Computer*, p.5-16, Feb., 1990.
- FI86 FISHER, A. J. A Multi-processor Implementation of Occam. *Software - Practice and Experience*, v.16 n.10, p.875-92, oct., 1986.
- FT89 FTP Software Inc. PC/TCP Packet Driver Specification. Version 1.09. US, FTP Software Inc., 1989.
- FU91 FUKUDA A.; MURAKAMI, K. Toward Advanced Parallel Processing - Exploiting Parallelism at Task and Instruction Levels. *IEEE Micro*, v.11, n.4, p.16-9,50-61, Aug, 1991.
- GA83 GANE, C.; SARSON, T. *Análise Estruturada de Sistemas.* São Paulo. Livros Técnicos e Científicos, 1983.
- GL87 GLENDINNING, I. *The Transputer Development System.* Southampton, Transputer Support Centre, 1987. (Internal Report)
- GO84 GOMAA, H. A Software Design Method for Real Time Systems. *Communications of the ACM*, v.27, n.9, p.938-49, Sept., 1984.
- GO86 GOMAA, H. Software Development of Real Time Systems. *Communications of the ACM*, v.29, n.7, p.657-68, July, 1986.
- GO89 GORTON, I.; Kerridge, J.; Jervis, B. Simulating Microprocessor Systems Using Occam and a Network of Transputers. *IEEE Proceedings*, v.136, n.1, Jan., 1989.
- GR89 GREEN, T. A C++ Multitasking Kernel. *Dr. Dobbs's Journal*, v.14, n.2, p.45-51; 84-90, Feb. 1989.
- HA87 Harel,D.; Pnueli,A.; Schmidt,J.P.; Sherman,R. On the Formal Semantics of Statecharts. In: *Proceedings of 2nd Symposium on Logic in Computer Science*, Ithaca, NY, 1987. p.54-64.

- HA88a HAREL, D. et al. StateMate: A Working Environment for the Development of Complex Reactive Systems. In: *Proceedings of the 10th Conference on Software Engineering*, Singapore, 1988. p.
- HA88b HAREL, D. On Visual Formalisms. *Communications of the ACM*, v. 31, n.5, p.514-31, May, 1988.
- HA90 HARDY, I. D.; JORDAN, A. P. H. The Application of Transputers as a Network 'Compute' Server. In: Pritchard, D.J.; Scott, C.J. (eds), *Applications of Transputers 2*, Amsterdam, IOS-Press, 1990. p.311-18.
- HE87 HEY, A.J.G. *Parallel Decomposition of Large Scale Simulations in Science and Engineering*. Southampton, Southampton University, 1987. (Internal Report)
- HE88 HEY, A.J.G.; PRITCHARD, D.J. Parallel Applications on the RTP Supernode Machine. In: Kartshev, L.P.; Kartshev, S.I. *Proceedings of 3rd International Conference on Supercomputing*, v.II, 1988. p.264-270.
- HE89a HEY, A.J.G. Reconfigurable Transputer Networks: Practical Concurrent Computation. In: Elliot R.; C.A.R. Hoare (eds), *Scientific Applications of Multiprocessors*, New York. Prentice Hall, 1989. p.39-54.
- HE89b HEY, A. J. G. Experiments in Mimd Parallelism. Lecture Notes in Computer Science, Proceedings of PARLE 89 Parallel Architectures and Languages Europe, v.II, p.28-42, 1989.
- HE90a HEY, A.J.G. Supercomputing with Transputers - Past, Present and Future. *Proceedings of the International Conference on Supercomputing*, v.18, n.3, p.479-89, June, 1990.
- HE90b HEY, A.J.G. Concurrent Supercomputing in Europe. In: *Distributed Memory Computing Conference*, 5, Charleston, 1990.
- HE91 HEY, A.J.G. Experience with MIMD Message-Passing Systems - Towards General Purpose Parallel Computing. In: *Jornada EPUSP/IEEE em Sistemas de Computação de Alto Desempenho*, São Paulo, 1991.
- HO78 HOARE, C.A.R. Communicating Sequential Processes. *Communications ACM*, v.21, n.11, p.666-77, Aug., 1978.
- HO86 HOPPER, A.; TEMPLE, S. and WILLIAMSON, R. *Local Area Network Design*. Londres, Addison-Wesley, 1986.
- HO87 HOMEWOOD, M.; MAY, D.; SHEPHERD, D.; SHEPHERD, R. The IMS T800 Transputer. *IEEE MICRO*, v.7, n.5, p.10-26, Oct. 1987.
- HW84 HWANG, K.; BRIGGS, F. A. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.
- HW87 HWANG, K. Advanced Parallel Processing with Supercomputer Architectures. *Proceedings of the IEEE*, v.75, n.10, p.1348-1349, Oct., 1987.

- IE85a ANSI/IEEE. Standard 802.5. *Carrier Sense Multiple Access with Collision Detection*. California, IEEE, 1985.
- IE85b ANSI/IEEE. Standard 754. *Binary Floating-Point Arithmetic*. California, IEEE, 1985.
- IN85 INMOS Limited. *User's Manual. Transputer Development System - IBM/PC Version*. INMOS document number 72 TDS 041 00, 1985.
- IN86 INMOS Limited. *IMS B004 Evaluation Board - User Manual*, U.K., 1986.
- IN88 INMOS Limited. *Occam 2 - Reference Manual*. U.K., Prentice Hall, 1988.
- IN89a INMOS Limited. *The Transputer Databook*, 1989, (INMOS Databook Series, INMOS document number 72 TRN 203 00).
- IN89b INMOS Limited. *The Transputer Applications Notebook - Systems and Performance*, 1989. (INMOS Databook Series, 72 TRN 205 00).
- IN89c INMOS Limited. *3L Parallel C - User Guide*, 1989. (INMOS document number 72 TDS 179 00).
- IN89d INMOS Limited. *3L Pascal - User Manual*, 1989. (INMOS document number 72 TDS 145 00).
- IN89e INMOS Limited. *3L Pascal - Reference Manual*, 1989. (INMOS document number 72 TDS 148 00).
- IN89f INMOS Limited. *The Transputer Development and iq systems Databook*, 1989. (INMOS document number 72 TRN 219 00).
- IN90 INMOS Limited. *Transputer Development System - User Guide and Reference Manual*. U.K., Prentice Hall, U.K., 1990.
- IN91 INMOS Limited. *The T9000 Transputer Products Overview Manual*, 1991. (INMOS Databook Series, 72 TRN 228 00).
- JO88 JONES, G.; GOLDSMITH, M. *Programming in Occam2*. New York, Prentice Hall, 1988. (Prentice Hall International Series in Computer Science).
- KU82 KUNG, H.T. Why Systolic Architectures? *IEEE Computer*, v.15, n.1, p.37-46, Jan., 1982.
- KU89 KUNG, H.T. Computacional Models for Parallel Computers. In: Elliot, R.J.; Hoare. *Scientific Applications of Multiprocessors*, New York, Prentice Hall, 1989, p.1-15
- LI86 LINGS, B.J. *Information Structures - A Uniform Approach Using Pascal*. U.K. Chapman and Hall, 1986.
- LI87 LIPPMANN, R. An Introduction to Computing With Neural Nets. *IEEE ASSP Magazine*, v.4, n.2, p.4-22, Apr., 1987.
- LO89a Logical Systems. *TCX "C" Compiler User Guide*. Version 88.4. 1990.

- LO89b Logical Systems. LD - NET, Network Loader User Guide. Version 88.4. 1990.
- LO90 LOVENGREEN, H. H.; RAVN, A. P.; RISCHER, H. Design of Embedded, Real Time Systems: Developing a Method for Practical Software Engineering. In: *Comp Euro 90*, Tel-Aviv, 1990.
- MA89 MASIERO, P.C. *Uma Visão Geral do Método JSD*. São Carlos, ICMSC-USP, 1989. (Notas Didáticas do ICMSC - USP, 3)
- MC89 MCDOWELL, C. E.; HELMBOLD, D. P. Debugging Concurrent Programs. *ACM Computing Surveys*, v.21, n.4, p.593-623, Dec. 1989.
- ME76 METCALFE, R.M.; BOGGS, D.R. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, v.19 n.7, p.395-403, Jul, 1976.
- ME90 Meiko scientific Ltd. Computing Surface Hardware Reference Manual. Version 1.0, 1990.
- MI87 Microway, Inc. Monoputer 2 - Owner's Manual, Massachusetts, 1987.
- MI89 Minghim, R.; Germano, F.S.R.; Trindade, O.T. Microcomputador Z80 com Memória Extendida. Congresso de Iniciação Científica e Tecnológica em Engenharia, 8, São Carlos, 1989. *Anais*.
- MI90 Microsoft Corporation. Microsoft Operating System/2 - User's Guide. Version 1.2, Document Number SY13238-0490 OEMO 786-1Z, USA, 1990.
- MU89 MULLENDER, S.J. (ed). *Distributed Systems*. Londres, Addison-Wesley, 1989.
- NA89 NAVAUX, P.O.A. Introdução ao Processamento Paralelo. *Revista Brasileira de Computação*, v.5, n.2, p.31-43, out./dez. 1989.
- NI88a NICOLE, D. A. *Esprit Project 1085 - Reconfigurable Transputer Processor Architecture*. Private communication, U.K., Southampton University, 1989.
- NI88b NICOL, D. M.; Saltz, J. H. Dynamic Remapping of Parallel Computations with Varying Resource Demands. *IEEE Transactions on Computers*, v.37, n.9, p.1073-1087, Sept., 1988.
- PA88 PAGE-JONES, M. *Projeto Estruturado de Sistemas*. São Paulo, McGraw Hill, 1988.
- PE91 PETERSON, C.; SUTTON, J.; WILEY, P. IWarp: A 100 Mops LIW Microprocessor for Multicomputers. *IEEE Micro*, v. 11, n.3, p.26-29; 81-87, June 1991.
- PO87 POUNTAIN, D.; MAY, D. *A Tutorial Introduction to OCCAM Programming*. U.K., BSP Professional Books, 1987.
- PR87 PRESSMAN, R.S.; *Software Engineering - A Practitioner's Approach*. McGraw Hill, 1987.
- PR88 PRITCHARD, D.J. Mathematical Models of Distributed Computation. In: Muntean, T. *7th Occam User Group Technical Meeting*. Amsterdam, IOS Press, 1988.

- RO90 ROUNCE, P, DELGADO J. Architectures Within the ESPRIT SPAN Project. *IEEE MICRO*, v.10, n.6, p.24-7, 88-97, Dec. 1990.
- SA88 SANTANA, M.J.; SANTANA, R.H.C.; ZALUSKA, E.J. The TRICE Fast Bridge For Efficient LAN-LAN Coupling. *IEE Proceedings E*, v.5, n.135, p.273-7, Sep., 1988.
- SH82 SHOCH, J. F.; Dalal, Y.K.; Redell, D.D. Evolution of the Ethernet Local Computer Network, *IEEE Computer*, v.15, n.8, p.10-27, Aug., 1982.
- SI79 SIMPSON, H.R.; Jackson, K.L. Process Synchronization in MASCOT. *The Computer Journal*, v.22, n.4, p.332-45, 1979.
- ST82 STTOTS Jr., P.D. A Comparative Survey of Concurrent Programming Languages. *ACM Sigplan Notes*, v.17, n.9, p.76-87, Sep. 1982.
- ST84 STALLINGS, W. Local Networks. *ACM Computer Surveys*, v.16 n.1, p.3-41, March, 1984.
- SU90 SURRIDGE, M. A General Communications Harness and Configuration Language. In: Pritchard, D.J.; Scott, C.J. (eds), *Applications of Transputers 2*, Amsterdam, IOS - Press, 1990, p.340-354.
- TA85 TANENBAUM, A.S.; van RENESSE, R. Distributed Operating Systems. *ACM Computing Surveys*, v.17(4), p.419-70, Dez. 1985.
- TA88 TANENBAUM, A.S. *Computer Networks*. 2.ed., New-York, Prentice-Hall, 1988.
- TA91 TARPENNING, M. Cooperative Multitasking in C++ . *Dr. Dobbs's Journal*, p.54-59; 96-99, April, 1991.
- TR90 TRINDADE Jr, O.; SANTANA, M.J. Um servidor de Processamento Paralelo Baseado em Transputers - Requisitos e Definição. In: Simpósio Brasileiro de Arquitetura de Computadores e Processamento Paralelo 3, Rio de Janeiro, 1990. *Anais* Rio de Janeiro, SBC - PUC-Rio, 1990. p.225-37.
- TR91a TRINDADE Jr, O.; Sistema SPP - Especificação Geral. Documento de trabalho, versão preliminar. SCE-ICMSC-USP, São Carlos, 1991.
- TR91b TRINDADE Jr, O.; Sistema SPP - Documentação de Software. Documento de trabalho, versão preliminar. SCE-ICMSC-USP, São Carlos, 1991.
- TR91c TRINDADE Jr, O.; Sistema SPP Documentação de Hardware. Documento de trabalho, versão preliminar. SCE-ICMSC-USP, São Carlos, 1991.
- VA89 VALIANT, L. G. Optimally Universal Parallel Computers. Scientific Applications of Multiprocessors. In: Elliot, R.J.; Hoare, C.A.R. New York, Prentice Hall, 1989, p.17-20.
- WA84 LCS - Lógica de Construção de Sistemas. Rio de Janeiro, Campus, 1984.
- WE89 WEXLER, J. *Concurrent Programming in Occam2*. Ellis Horwood, 1989.
- WH90 WHITBY-Strevens, C. Transputers - Past, Present and Future. *IEEE MICRO*, v.10, n.6, p.16-19; 76-82, Dec. 1990.

- WI87 WILBUR, S.; Bacarisse, B. Building Distributed Systems with Remote Procedure Call. *Software Engineering Journal*, p.148-159, Sept., 1987.
- ZI80 ZIMMERMANN, H. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, *IEEE Transactions on Communications*, v.28, n.4, p.425-32, April, 1980.

APÊNDICE A1

ANÁLISE DA REDE DE CHAVEAMENTO

As duas características exigidas para rede de chaveamento do sistema servidor de processamento paralelo são:

- > Possibilidade de constituição de lotes de Transputers, isto é, a definição de redes de processadores formando subgrafos desconexos.
- > Conectividade total entre os processadores de cada lote.

Entre as possíveis arquiteturas de chaveamento que apresentam essas características, as seguintes foram investigadas:

- 1- Uma única rede constituída por uma chave do tipo "crossbar".
- 2- Rede estruturada em árvore, constituída por subredes com chaves do tipo "crossbar".
- 3- Rede estruturada em árvore, constituída por subredes com chaves do tipo quase "crossbar", organizadas bidimensionalmente.

A arquitetura 1 representa o caso ideal para o servidor de processamento paralelo sob o ponto de vista operacional (figura 3.9). Não apresenta, entretanto, uma estrutura modular, podendo representar o item mais caro em sistemas médios e pequenos. As arquiteturas 2 e 3 têm estrutura modular, permitindo a definição de sistemas pequenos, a baixo custo (figura 3.4). A arquitetura 3 apresenta o menor custo entre as três

alternativas, utilizando chaves com as restrições de conectividade mencionadas no item 3.5.3 (figuras 3.10 e 3.11) [IN89b].

Cada uma dessas arquiteturas exige o emprego de chaves com grande número de canais. Normalmente essas chaves são constituídas a partir da associação de chaves menores. Em [IN89b] duas possibilidades são apresentadas. A segunda possibilidade de associação é mais geral, e por isso foi utilizada nesse trabalho. Supondo a associação de chaves com M canais, constituindo uma chave com N_v canais, tem-se:

assumindo:

$$M = 2^K \quad N = 2^L \quad K, L \text{ inteiros positivos quaisquer}$$

$$N_c = 3.N \text{ (número de chaves)}$$

1 - para $N \leq M$ (roteamento através de 3 chaves)

$$V_1 = M$$

$$N_{v1} = V_1.N_1 = M.N_1 = M.(N_c/3)$$

caso $N_1 = M$

$$N_{v1} = M^2$$

2 - para $M < N \leq M^2$ (cada chave composta por uma chave anterior, roteamento através de 9 chaves)

$$V2 = M.Nc1/3 \quad N2 = Nc2/3 \quad Nv2 = V2.N2 = M.(Nc1/3).(Nc2/3)$$

Pode-se verificar facilmente que a equação geral recorrente para o caso geral com I níveis de associação é:

$$Nci = (3.Nvi)/Nvi-1 \quad Nv0 = M$$

Utilizando-se $M = 32$ nessas equações, o que equivale a implementar as chaves com circuitos integrados C004, tem-se:

para associação em um nível:

$$Nv1 = 32.(Nc1/3) = > Nc1 = (3.Nv1)/32$$

para associação em dois níveis:

$$Nv2 = 32.(Nc1/3).(Nc2/3) = > Nc2 = (3.Nv2)/Nv1$$

A avaliação das arquiteturas acima quanto à relação do número de chaves C004 pelo número de processadores, para dois casos particulares de interesse, leva aos seguintes valores:

arquitetura 1

um nível, 256 Transputers = > 1024 canais

$$Nc/Np = 96/256 = 0.375$$

dois níveis, 1024 Transputers = > 4096 canais

$$Nc/Np = 1152/1024 = 1,125$$

arquitetura 2

um nível, 256 Transputers = > 1024 canais

$$Nc/Np = 288/256 = 1.125$$

dois níveis, 1024 Transputers = > 4096 canais

$$Nc/Np = 1920/1024 = 1,875$$

arquitetura 3

um nível, 256 Transputers => 1024 canais

$$N_c/N_p = 160/256 = 0.625$$

dois níveis, 1024 Transputers => 4096 canais

$$N_c/N_p = 640/1024 = 0.625$$

Observa-se que a arquitetura 1 apresenta a melhor relação N_c/N_p para sistemas dispondo de 153 a 256 processadores. Fora dessa faixa, para os casos descritos, a arquitetura 3 apresenta uma relação mais favorável. Com 32 processadores, por exemplo, a relação N_c/N_p vale 0.125. Uma análise detalhada do comportamento dessa relação pode ser encontrada em [TR91a].

APÊNDICE A2

DESCRIÇÃO DE INTERFACE DAS FUNÇÕES - PD

Neste apêndice são apresentados extratos das descrições de interface das funções básicas integrantes do "Packet Driver" NCSA. A descrição detalhada dos módulos restantes e maiores informações podem ser encontradas em [CL89a].

```
driver_info(handle) AH == 1 AL == FF
    int handle; BX
Retorna informações sobre a interface com o meio físico da rede
local.
retorno com erro:
    flag carry posicionado
    código de erro DH
erros possíveis:
    BAD_HANDLE
retorno sem erro:
    flag de carry zerado
    version          BX
    class            CH
    type             DX
    number           CL
    name             DS:SI
    basic/extended  AL
                    1 == básica, 2 == extendida,
                    FF == não disponível
```



```
int access_type(if_class, if_type, if_number, type, typelen,
               receiver) AH == 2
```

```
    int if_class; AL
    int if_type; BX
    int if_number; DL
    char far *type; DS:SI
    unsigned typelen; CX
    int (far *receiver)(); ES:DI
```

Inicia o processamento de pacotes do tipo especificado.

retorno com erro:

```
    flag carry posicionado
    código de erro DH
```

erros possíveis:

```
    NO_CLASS
    NO_TYPE
    NO_NUMBER
    BAD_TYPE
    NO_SPACE
    TYPE_INUSE
```

retorno sem erro:

```
    flag de carry zerado
    handle AX
```

chamada à rotina de recepção:

```
    (*receiver)(handle, flag, len [, buffer])
    int handle; BX
    int flag; AX
    unsigned len; CX
    if AX == 1, char far *buffer; DS:SI
```

```
int release_type(handle) AH == 3
```

```
    int handle; BX
```

Esta função termina o acesso à pacotes associados com o handle.

retorno com erro:

```
    flag carry posicionado
    código de erro DH
```

erros possíveis:

```
    BAD_HANDLE
```

retorno sem erro:

```
    flag de carry zerado
```

```
int send_pkt(buffer, length) AH == 4
char far *buffer; DS:SI
unsigned length; CX
Transmite length bytes de dados, apontados por buffer.
retorno com erro:
```

```
    flag carry posicionado
    código de erro DH
```

```
erros possíveis:
```

```
    CANT_SEND
```

```
retorno sem erro:
```

```
    flag de carry zerado
```

```
terminate(handle) AH == 5
    int handle; BX
Encerra o driver associado com o handle.
retorno com erro:
```

```
    flag carry posicionado
    código de erro DH
```

```
erros possíveis:
```

```
    BAD_HANDLE
    CANT_TERMINATE
```

```
retorno sem erro:
```

```
    flag de carry zerado
```

```
get_address(handle, buf, len) AH == 6
    int handle; BX
    char far *buf; ES:DI
    int len; CX
Copia para buf o endereço de rede associado com o handle.
retorno com erro:
```

```
    flag carry posicionado
    código de erro DH
```

```
erros possíveis:
```

```
    BAD_HANDLE
    NO_SPACE
```

```
retorno sem erro:
```

```
    flag de carry zerado
    length CX
```

reset_interface(handle) AH == 7

int handle; BX

Reinicia a interface associada com o handle para um estado default.

retorno com erro:

flag carry posicionado

código de erro DH

erros possíveis:

BAD_HANDLE

retorno sem erro:

flag de carry zerado

APÊNDICE A3

DESCRIÇÃO DE INTERFACE DOS MÓDULOS - ET

Neste apêndice são apresentadas as especificações de interface dos módulos correspondentes ao software executado nas estações de trabalho, mostrados na figura 4.11. A documentação adicional pode ser encontrada em [TR91b]. A primitiva `InverterCadeia` foi incluída como exemplo.

```
InverterCadeia (int *tam, BYTE *str)
Func : Inverte uma cadeia de caracteres terminada por /0.
Ent  : tam  -> tamanho da cadeia a ser invertida
      str  -> cadeia a ser invertida
Sai  : str  -> cadeia já invertida
Possíveis Erros: 212- Erro na transmissão
                  315- Servidor não responde

ObterIdeServ (int *serv, BYTE prot, int prim)
Func : Obtem o identificador de um servidor para a execução
      da primitiva prim.
Ent  : prot -> identificador do protocolo
      prim -> identificador da primitiva
Sai  : serv -> identificador do servidor

MontarPacoteSPP (int serv, BYTE prim, int tam_dad,
                 BYTE *dados, BYTE *pac_spp)
Func : Monta um pacote SPP
Ent  : serv  -> identificador do servidor
      prim  -> identificador da primitiva
      tam_dad -> tamanho dos dados
      dados  -> dados a serem empacotados
Sai  : pac_spp -> pacote SPP
```

InvocarProcedimentoRemoto (int serv, BYTE prim, BYTE *pac_spp)
Func : Realiza a chamada de um procedimento remoto e aguarda resposta. Aborta o processo com código de erro xx se o servidor não responder.
Ent : pac_spp -> pacote SPP solicitado
serv -> servidor
prim -> primitiva
Sai : pac_spp -> pacote SPP resposta se retorno = 0
Possíveis Erros: 212- Erro na transmissão
315- Servidor não responde

DesmontarPacoteSPP (BYTE *bufe, int *tam_bufs, BYTE *bufs)
Func : Retira o header SPP de um pacote e retorna o campo de dados e o seu tamanho
Ent : bufe -> pacote SPP recebido
Sai : bufs -> dados
tam_bufs -> tamanho dos dados
Possíveis Erros: 316- Término do time-out
311- Tipo do protocolo não confere
312- Erro na transmissão
314- Pacote não é requisição
322- Erro na execução da primitiva

AnalisarResposta (BYTE prim, BYTE *bufe, int serv)
Func : Verifica se a resposta recebida tem os mesmos dados da requisição feita.
Ent : prim -> identificador da primitiva
bufe -> pacote SPP recebido
serv -> identificador do servidor
Possíveis Erros: 325- Sequência não confere
324- Primitiva não confere
323- Processo não confere
321- Pacote não é resposta

TransmitirSPP (int servidor, BYTE *buf)
Func : Monta e transmite um pacote PKT para um servidor especificado
Ent : servidor -> identificador do servidor
buf -> pacote SPP fornecido
Sai : buf -> pacote PKT a ser transmitido
Possíveis Erros: 12- Erro na transmissão

ReceberSPP (BYTE *buf)
 Func : Recebe pacotes PKT e converte para SPP
 Sai : buf -> pacote SPP

ReceberPKT()
 Func : Receber e armazenar um pacote PKT.

ObterParametros (int intno,int handle,int *version,int *class,
 int *type,int *number,int *basic)
 Func : Esta função retorna informações sobre a interface
 com a rede local.
 Ent : intno -> número da interrupção
 handle -> identificador do processo de recepção
 Sai : version -> versão do "packet driver"
 class -> classe da interface
 type -> tipo da interface
 number -> número da interface
 basic -> tipo de funções disponíveis
 Possíveis Erros: 1- handle inválido

HabilitarAcessoTipoPacote (int intno,int *handle, int if_class,
 int if_type, int if_number,
 BYTE far *type, unsigned short
 typelen, void (_far *receiver) ())
 Func : Inicia o acesso a um pacote do tipo especificado.
 Ent : intno -> número da interrupção
 if_class -> classe da interface
 if_type -> tipo de interface
 if_number -> número da interface
 type -> tipo do pacote
 typelen -> comprimento do pacote
 receiver -> rotina de recepção
 Sai : handle -> identificador do processo de recepção
 Possíveis Erros: 2- classe inválida
 3- tipo inválido
 4- número inválido
 5- tipo do pacote inválido
 9- espaço insuficiente
 10- tipo em uso

TerminarAcessoTipoPacote (int intno, int handle)

Func : Esta função finaliza o acesso a um pacote associado ao handle.

Ent : intno -> número da interrupção

handle -> identificador do processo de recepção

Possíveis Erros: 1- handle inválido

TransmitirPkt (int intno, BYTE _far *buffer, int length)

Func : Transmite o buffer de tamanho length.

Ent : intno -> número da interrupção

buffer -> buffer a ser transmitido

length -> tamanho do buffer a ser transmitido

Possíveis Erros: 12- Erro na transmissão

APÊNDICE A4

DESCRIÇÃO DE INTERFACE DOS MÓDULOS - PE

Neste apêndice são apresentadas as especificações de interface dos módulos correspondentes ao software executado no processador de entrada, mostrados na figura 4.12. A documentação adicional pode ser encontrada em [TR91b]. A primitiva InverterCadeia foi incluída como exemplo.

InverterCadeia (int *tam, BYTE *str)
Func : Inverte uma cadeia de caracteres terminada por /0.
Ent : tam -> tamanho da cadeia a ser invertida
 str -> cadeia a ser invertida
Sai : str -> cadeia já invertida

AtenderProcedimentoRemoto ()
Func : Atender a chamada de um procedimento remoto e enviar resposta.
Possíveis Erros: 12 - Erro na transmissão
 0x08 - Primitiva não existe ou usuário sem permissão
 0x04 - Sessão não existe
 0x10 - Requisição não é válida

ProcessarSPP (int sessão, BYTE *bufe)
Func : Verificar a possibilidade de execução da primitiva requisitada.
Ent : sessão -> identificador da sessão
 bufe -> pacote SPP recebido
Sai : buf -> estado do pacote SPP
Possíveis Erros: 0x08 - Primitiva não existe ou usuário sem permissão
 0x04 - Sessão não existe
 0x10 - Requisição não é válida

MontarPacoteSPP (int sessão, BYTE prim, int tam_dad,
BYTE *dados, BYTE *pac_spp, BYTE est)

Func : Montar um pacote SPP

Ent : sessão -> identificador da sessão
prim -> identificador da primitiva
tam_dad -> tamanho do campo de dados
dados -> dados do pacote

Sai : pac_spp -> pacote SPP

DesmontarPacoteSPP (BYTE *bufe, int *tam_bufs, BYTE *bufs)

Func : Retirar o header SPP de um pacote e retornar
o campo de dados e o seu tamanho

Ent : bufe -> pacote SPP recebido

Sai : tam_bufs -> tamanho dos dados
bufs -> dados

ReceberPKT()

Func : Receber e armazenar um pacote PKT.
Rejeitar o pacote se Fila_Rec cheia.

TransmitirSPP_ET (int sessão, BYTE *buffer)

Func : Montar e transmitir um pacote PKT para um processo
especificado

Ent : sessão -> identificador da sessão
buffer -> pacote SPP fornecido

Sai : buffer -> pacote PKT a ser transmitido

Possíveis Erros: 12- Erro na transmissão

ReceberSPP_ET (BYTE *buf)

Func : Receber pacotes PKT e converter para SPP

Sai : buf -> pacote SPP

Possíveis Erros: -1 - Fila Vazia, não pode retirar pacote

DefinirFilaRec (FILAREC _far *fila)

Func : Definir uma fila de recepção.

Sai : fila -> fila de recepção

InserirFilaRec()

Func : Inserir um pacote PKT na fila de recepção.

Sai : fila -> fila de recepção

```
RetirarFilaRec (FILAREC _far *fila, BYTE *buffer,
               int *buf_len)
Func : Retirar um pacote PKT da fila de recepção.
Ent  : fila      -> fila de recepção
Sai  : buffer    -> pacote PKT retirado da fila
      : buf_len  -> tamanho do pacote PKT retirado
      : fila     -> fila de recepção
Possíveis Erros:  -1 - Fila Vazia, não pode ser retirado
                  nenhum pacote
```

```
ObterParametros (int intno,int handle,int *version,int *class,
                 int *type,int *number,int *basic)
Func : Esta função retorna informacoes sobre a interface
      com a rede local.
Ent  : intno     -> número da interrupção
      : handle   -> identificador do processo de recepção
Sai  : version  -> versão do "packet driver"
      : class   -> classe da interface
      : type    -> tipo da interface
      : number  -> número da interface
      : basic   -> tipo de funções disponíveis
Possíveis Erros:  1- handle inválido
```

```
HabilitarAcessoTipoPacote (int intno,int *handle, int if_class,
                           int if_type, int if_number,
                           BYTE _far *type, unsigned short
                           typelen, void (_far *receiver) ())
Func : Inicia o acesso a um pacote do tipo especificado.
Ent  : intno     -> número da interrupção
      : if_class -> classe da interface
      : if_type  -> tipo de interface
      : if_number -> número da interface
      : type     -> tipo do pacote
      : typelen  -> comprimento do pacote
      : receiver -> rotina de recepção
Sai  : handle   -> identificador do processo de recepção
Possíveis Erros:  2- classe inválida
                  3- tipo inválido
                  4- número inválido
                  5- tipo do pacote inválido
                  9- espaço insuficiente
                  10- tipo em uso
```

TerminarAcessoTipoPacote (int intno, int handle)
Func : Esta função finaliza o acesso a um pacote associado
ao handle.
Ent : intno -> número da interrupção
handle -> identificador do processo de recepção
Possíveis Erros: 1- handle inválido

TransmitirPkt (int intno, BYTE _far *buffer, int length)
Func : Transmite o buffer de tamanho length.
Ent : intno -> número da interrupção
buffer -> buffer a ser transmitido
length -> tamanho do buffer a ser transmitido
Possíveis Erros: 12- Erro na transmissão

TransmitirSPP_BT (int sessão, BYTE *buffer)
Func : Transmitir um pacote SPP para o BT
Ent : sessão -> identificador da sessão
buffer -> pacote SPP fornecido

ReceberSPP_BT (BYTE *buf)
Func : Receber pacotes SPP provenientes do BT
Sai : buf -> pacote SPP
Possíveis Erros: -1 - Pacote não disponível

APÊNDICE A5

DESCRIÇÃO DE INTERFACE DOS MÓDULOS - CM

Neste apêndice são apresentadas as especificações de interface dos módulos correspondentes ao software executado no controlador de módulo, mostrados na figura 4.13. A documentação adicional pode ser encontrada em [TR91b].

```
RotearSPP (CHAN OF SPP from.PE, to.PE, from.lote, to.lote)
Func : Roteia pacotes SPP entre o PE e a filas
      de entrada e saída
Ent  : from.PE  -> pacote SPP proveniente do PE
      from.lote -> pacote SPP proveniente do lote
Sai  : to.PE    -> pacote SPP para o PE
      to.lote   -> pacote SPP para o lote
```

```
ReceberSPP.PE ([]BYTE pspp ent)
Func : Recebe pacote SPP do PE e insere na fila de entrada
Ent  : ent  -> pacote SPP
```

```
TransmitirSPP.PE ([]BYTE pspp ent, INT est)
Func : Transmite pacote SPP para o PE
Ent  : ent  -> pacote SPP
Sai  : est  -> 0x00 -> operação realizada
      0xff -> erro na transmissão
```

```
InserirSPP ([]BYTE pspp ent, INT est)
Func : Insere pacote SPP na fila de entrada
Ent  : ent  -> pacote SPP
Sai   : est  -> 0x00 -> operação realizada
        0xff -> fila cheia
```

```
RetirarSPP ([]BYTE pspp sai, INT est)
Func : Retira pacote SPP na fila de saída
Sai   : sai  -> pacote SPP
        est  -> 0x00 -> operação realizada
        0xff -> fila vazia
```

```
RotearSP* (CHAN OF INT ide.lote.e, ide.lote.s,
           CHAN OF SP* from.CM, to.CM,
           from.lote0, to.lote0, from.lote1, to.lote1,
           from.lote2, to.lote2, from.lote3, to.lote3,)
Func : Roteia pacotes SP* entre o CM e os módulos de
       comunicação com cada lote
Ent   : from.CM    -> pacote SP* proveniente do CM
       from.lote0  -> pacote SP* proveniente do lote0
       from.lote1  -> pacote SP* proveniente do lote1
       from.lote2  -> pacote SP* proveniente do lote2
       from.lote3  -> pacote SP* proveniente do lote3
       ide.lote.e  -> identificador do lote
Sai   : to.CM      -> pacote SP* para o CM
       to.lote0    -> pacote SP* para o lote0
       to.lote1    -> pacote SP* para o lote1
       to.lote2    -> pacote SP* para o lote2
       to.lote3    -> pacote SP* para o lote3
       ide.lote.s  -> identificador do lote
```

```
ReceberSP* ([]BYTE psp* ent)
Func : Recebe pacote SP* do roteador e envia
       para processamento
Ent   : ent  -> pacote SP*
```

```
TransmitirSP* ([]BYTE psp* ent)
Func : Transmite pacote SP* para o interpretador
Ent   : ent  -> pacote SP*
```

```

ProcessarSP*pLote0 ([]BYTE psp* ent, INT est)
Func  : Valida pacote SPP e envia para processamento
        do lote 0
Ent   : ent  -> pacote SP*
Sai   : est  -> código de erro

```

```

ProcessarSP*doLote0 ([]BYTE psp* ent, INT est)
Func  : Valida pacote SP* proveniente do lote0
Ent   : ent  -> pacote SP*
Sai   : est  -> código de erro

```

```

InterpretarSP* (CHAN OF SP* from.lote, to.PE,
                CHAN OF SPP resp.prim, req.prim)
Func  : Recebe pacotes provenientes do lote ou correspondentes
        à respostas de primitivas e insere para o PE ou envia
        para execução de primitiva de controle.
Ent   : resp.prim -> pacote SP* resposta de primitiva
        from.lote -> pacote SP* proveniente do lote
Sai   : to.PE     -> pacote SP* para o PE
        req.prim  -> pacote SP* requisição de primitiva

```

```

ReceberSP* ([]BYTE psp* from.lote)
Func  : Recebe pacote SP* proveniente do lote
Sai   : from.lote -> pacote SP*

```

```

ReceberResposta ([]BYTE psp* resp.prim)
Func  : Recebe pacote SP* (resposta de primitiva)
Sai   : resp.prim -> pacote SP*

```

```

MontarSPP ([]BYTE psp* ent, []BYTE pspp sai)
Func  : Monta pacote SPP a partir de pacote SP*
Ent   : ent  -> pacote SP*
Sai   : sai  -> pacote SPP

```

```

InterpretarSPP ([]BYTE pspp, INT est)
Func  : Valida pacote SPP
Ent   : ent  -> pacote SPP
Sai   : est  -> código de erro

```

InserirSPP ([]BYTE pspp ent)

Func : Insere pacote SPP na fila de saída

Ent : ent -> pacote SPP

Sai : est -> 0x00 -> operação realizada
0xff -> fila cheia

TransmitirRequisição ([]BYTE pspp req.prim)

Func : Envia pacote SPP para execução

Ent : req.prim -> pacote SPP (requisição de primitiva)

InterpretarSPP (CHAN OF SPP from.PE, to.lote, resp.prim,
req.prim)

Func : Retira pacotes SPP provenientes do PE ou resposta de
primitiva, e envia para execução (primitiva de controle)
ou no sentido do lote.

Ent : from.PE -> pacote SPP proveniente do PE

resp.prim -> pacote SPP resposta de primitiva

Sai : to.lote -> pacote SPP para o lote

req.prim -> pacote SPP para execução

RetirarSPP ([]BYTE pspp sai, INT est)

Func : Retira pacote SPP na fila de entrada

Sai : sai -> pacote SPP

est -> 0x00 -> operação realizada

0xff -> fila vazia

ReceberResposta ([]BYTE pspp resp.prim)

Func : Recebe pacote SPP (resposta de primitiva)

Sai : resp.prim -> pacote SPP

InterpretarSPP ([]BYTE pspp, INT est)

Func : Valida pacote SPP

Ent : ent -> pacote SPP

Sai : est -> código de erro

MontarSP* ([]BYTE pspp ent, []BYTE psp* sai)

Func : Monta pacote SP* a partir de pacote SPP

Ent : ent -> pacote SPP

Sai : sai -> pacote SP*

TransmitirSP* ([]BYTE psp* to.lote)
Func : Transmite pacote SP* para lote
Ent : ent -> pacote SP*

TransmitirRequisição ([]BYTE pspp req.prim)
Func : Envia pacote SPP para execução
Ent : req.prim -> pacote SPP (requisição de primitiva)

ProcessarSP*Lote0 (CHAN OF SPP ent, sai,
 CHAN OF BYTE entb, saib)
Func : Gerencia comunicação com o lote0
Ent : ent -> pacote SP* proveniente do roteador
 entb -> byte proveniente do lote
Sai : sai -> pacote SP* para o roteador
 saib -> byte para o lote0

ReceberSP* ([]BYTE psp* sai)
Func : Recebe pacote SP* do roteador
Sai : sai -> pacote SP* recebido

ReceberByte (BYTE dado)
Func : Recebe um byte proveniente de lote
Sai : dado -> byte recebido

ExecutarPrimComunicação ([]BYTE psp* ent, sai)
Func : executar primitiva de comunicação
Ent : ent -> pacote SP* (requisição)
Sai : sai -> pacote SP* (resposta)

MontarSP* (BYTE dado)
Func : Monta e transmite pacote SP*
Ent : dado -> bytes para montagem do pacote SP*

TransmitirSP* ([]BYTE psp* ent)
Func : Transmite pacote SP* para lote
Ent : ent -> pacote SP*


```

AtenderPrimControle (CHAN OF SPP from.PE, to.lote,
                    from.lote, to.PE)
Func : Atender primitivas de controle
Ent  : from.PE    -> requisição proveniente do PE
      from.lote   -> requisição proveniente do lote
Sai   : to.PE     -> resposta para o PE
      to.lote     -> resposta para o lote

```

```

ReceberRequisiçãoPE ([]BYTE pspp sai)
Func : Recebe requisição proveniente do PE
Sai   : sai    -> requisição recebida

```

```

ReceberRequisiçãoLote ([]BYTE pspp sai)
Func : Recebe requisição proveniente de lote
Sai   : sai    -> requisição recebida

```

```

ExecutarPrimitiva ([]BYTE pspp ent, sai)
Func : Executar primitivas de controle
Ent   : ent    -> requisição
Sai   : sai    -> resposta

```

```

TransmitirRespostaPE ([]BYTE pspp ent)
Func : Transmite resposta para o PE
Ent   : ent    -> pacote SPP (resposta)

```

```

TransmitirRespostaLote ([]BYTE pspp ent)
Func : Transmite resposta para o lote
Ent   : ent    -> pacote SPP (resposta)

```