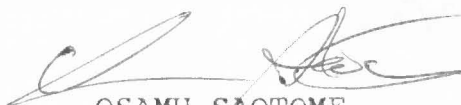


Tese apresentada à Divisão de Pós-Graduação do Instituto Tecnológico de Aeronáutica como parte dos requisitos para obtenção do título de Mestre em Ciência na área de Dispositivos e Sistemas Eletrônicos do Curso de Engenharia Eletrônica e Computação.


VANIA VIEIRA ESTRÊLA

AVALIACÃO EXPERIMENTAL DE ALGORITMOS DE VISÃO COMPUTACIONAL EM  
MAQUINAS MIMD BASEADAS EM "TRANSPUTERS"

Tese aprovada em sua versão final pelos abaixo assinados.



OSAMU SAOTOME  
ORIENTADOR



MARCO A. G. CECCHINI  
CHEFE DA DIVISÃO DE PÓS-GRADUAÇÃO

CAMPO MONTENEGRO  
SÃO JOSÉ DOS CAMPOS, SP, BRASIL  
1991

AVALIAÇÃO EXPERIMENTAL DE ALGORITMOS DE VISÃO COMPUTACIONAL  
EM MÁQUINAS BASEADAS EM "TRANSPUTERS"

Vania Vieira Estrêla.

Composição da Banca Examinadora:

Wagner Chiepa Cunha,	PhD., PRESIDENTE
Osamu Saotome,	PhD., ORIENTADOR
Elder M. Hemerly,	PhD.
Nelson D.A. Mascarenhas,	PhD.
Osvaldo Katsumi Imamura,	PhD.

## ÍNDICE DE ASSUNTOS.

I - INTRODUÇÃO .....	1
II - IDENTIFICAÇÃO DO PROBLEMA	
II.1 - PROCESSAMENTO DE IMAGENS .....	2
II.1.1 - Conceitos Básicos .....	2
II.1.2 - Classificação dos Algoritmos .....	4
II.1.3 - Algoritmos para Processamento de Pontos .....	6
II.1.4 - Algoritmos para Processamento de Área .....	9
II.1.4.1 - Convolução .....	9
II.1.4.2 - Processamento Não-linear de Área .....	14
II.1.4.3 - Classificação .....	15
II.1.5 - Algoritmos Geométricos .....	16
II.1.6 - Algoritmos para Processamento de Quadros .....	17
II.2 - VISÃO POR COMPUTADOR .....	18
II.3 - DETECÇÃO DE CURVAS USANDO TRANSFORMADA DE HOUGH .....	20
II.3.1 - Caracterização do Problema .....	20
II.3.2 - Transformada de Hough para Linhas Retas .....	21
II.3.3 - Transformada de Hough para Curvas Analíticas .....	25
II.3.3.1 - Efeito do Uso da Informação sobre o Sentido de Variação das Intensidades nos Contornos .....	26
II.3.3.2 - Algoritmo Genérico para Curvas Analíticas .	27
II.3.3.3 - Compensação de Erros .....	28

### III - TAXIONOMIA DAS ARQUITETURAS DE COMPUTADORES DESTINADAS AO PROCESSAMENTO DE IMAGENS E A VISÃO COMPUTACIONAL.

III.1 - CARACTERÍSTICAS .....	31
III.2 - TIPOS DE ARQUITETURA MAIS USADOS .....	34
III.2.1 - Arranjo de Processadores .....	34
III.2.2 - Arquiteturas em "Pipeline" .....	37
III.2.3 - Sistemas MIMD (multiprocessadores) .....	38
III.2.4 - Sistemas com Unidades Funcionais .....	40

### IV - MÁQUINAS M.I.M.D.

IV.1 - TOPOLOGIA .....	41
IV.1.1 - Acoplamento .....	42
IV.1.1.1 - Acoplamento Forte .....	42
IV.1.1.2 - Acoplamento Fraco .....	42
IV.1.2 - Tipos de Interconexão .....	45
IV.1.2.1 - Barramento Compartilhado .....	45
IV.1.2.2 - Rede de Barras Cruzadas .....	46
IV.1.2.3 - Memória Multiporta .....	48
IV.1.3 - Características de um Processador Destinado à Operação em Máquinas M.I.M.D. ....	49
IV.1.3.1 - Capacidade de Recuperação de Processos ....	49
IV.1.3.2 - Chaveamento de Contexto Eficiente .....	50
IV.1.3.3 - Espaços de Endereçamento Virtual e Físico Grandes .....	51
IV.1.3.4 - Primitivas de Sincronização Eficientes ....	51
IV.1.3.5 - Mecanismo de Comunicação entre Processadores .....	52
IV.1.3.6 - Conjunto de Instruções .....	52

IV.1.4 - Organizações de Memória .....	53
IV.2 - CONTROLE E SINCRONIZAÇÃO .....	54
IV.2.1 - Granularidade .....	54
IV.2.1.1 - Escalonamento por Lista .....	55
IV.2.1.2 - Escalonamento por Fluxo de Dados entre Processos de Grão Grosso .....	56
IV.3 - PROGRAMAÇÃO DE MÁQUINAS PARALELAS .....	56
IV.3.1 - Linguagens Paralelas .....	56
IV.3.1.1 - Classes de Linguagens para Programação Concorrente .....	57
IV.3.1.2 - Características de Linguagens para Programação de Sistemas M.I.M.D. ....	60
IV.3.2 - Sistemas Operacionais .....	63
IV.3.2.1 - Classificação dos Sistemas Operacionais para Multiprocessadores .....	63
V - O "TRANSPUTER" E A LINGUAGEM OCCAM	
V.1 - O "TRANSPUTER" .....	66
V.1.1 - Concepção .....	66
V.1.2 - Arquitetura do T800 .....	69
V.1.2.1 - Registradores .....	73
V.1.2.2 - Implementação de Processos Concorrentes ....	73
V.1.2.3 - Comunicação .....	75
V.1.2.4 - Prioridades .....	77
V.1.2.5 - Temporizadores .....	78
V.1.2.6 - A Unidade de Ponto Flutuante (FPU) ....	78
V.1.2.7 - Seção de Serviços de Sistema .....	81
V.1.2.8 - Memória e Interface com Periféricos .....	84

V.1.2.9 - Linhas de Comunicação .....	88
V.2 - A LINGUAGEM OCCAM .....	89
V.2.1 - Concorrência .....	90
V.2.2 - Localização e Distribuição de Processos .....	94
V.2.3 - Comunicação .....	95
V.2.4 - Estruturação dos Programas .....	97
V.2.5 - Tipos e Estruturas de Dados .....	99
V.2.6 - Abreviações .....	100
V.2.7 - Protocolos dos Canais .....	101
V.2.8 - Construções de Controle .....	103
V.2.9 - Programação em Tempo Real .....	106
V.2.10 - Operação em Sistemas com Múltiplos "Transputers" .....	108
VI - IMPLEMENTAÇÃO DE ALGORITMOS DE PROCESSAMENTO DE IMAGENS E DE VISÃO COMPUTACIONAL NUMA REDE DE "TRANSPUTERS".	
VI.1 - DESCRIÇÃO E ANÁLISE DAS FERRAMENTAS DISPONÍVEIS PARA O DESENVOLVIMENTO DE APLICAÇÕES PARA "TRANSPUTERS" .....	109
VI.2 - O USO DO "TRANSPUTER" EM OPERAÇÕES DE PROCESSAMENTO DE IMAGENS DE BAIXO NÍVEL .....	111
VI.3 - O USO DO "TRANSPUTER" EM VISÃO COMPUTACIONAL .....	114
VI.3.1 - Análise das Alternativas para a Implementação da Transformada de Hough numa Rede de "Transputers" .....	114
VI.3.1.1 - Interface Gráfica .....	115
VI.3.1.2 - Servidor de Rede .....	117
VI.3.1.3 - Aplicação .....	119

VI.3.2 - Detecção de Retas .....	121
VI.3.2.1 - Alternativas para a Implementação .....	122
VI.3.2.2 - Discussão dos Resultados .....	128
VI.3.3 - Detecção de Círculos .....	131
VI.3.3.1 - Alternativas para a Implementação .....	132
VII - CONCLUSÕES.	
VII.1 - CONSIDERAÇÕES RELATIVAS AO SISTEMA LÓGICO .....	140
VII.2 - LIMITAÇÕES DOS PROGRAMAS DE APLICAÇÃO .....	140
VII.3 - VANTAGENS DA ARQUITETURA UTILIZADA .....	143
BIBLIOGRAFIA .....	145
APÊNDICE I - O COMPUTADOR NCP I .....	150

## ÍNDICE DE FIGURAS

## CAPÍTULO II:

II.1 - Mapeamento do pixel (0,0) em (2,0) .....	16
II.2 - Representação paramétrica de uma linha reta .....	22
II.3 - Lugar geométrico dos parâmetros de um círculo .....	26
II.4 - Lugar geométrico dos parâmetros de um círculo com informação sobre o sentido de variação da intensidade no contorno .....	27
II.5 - Região onde é feita a compensação de erros .....	29

## CAPÍTULO III:

III.1 - Arquitetura SIMD .....	34
III.2 - Distribuição de EPs sobre uma imagem, com mapeamento EP-pixel .....	35
III.3 - Distribuição de EPs sobre uma imagem, com mapeamento EP-área .....	36
III.4 - Diagrama em blocos do Cytocomputer .....	37
III.5 - Organização de uma máquina M.I.M.D. ....	39

## CAPÍTULO IV:

IV.1 - Exemplos de arquiteturas fracamente acopladas .....	44
IV.2 - Organização típica de um barramento .....	45
IV.3 - Rede de barras cruzadas .....	47
IV.4 - Rede borboleta .....	47
IV.5 - Organização de um sistema de memória multiporta .....	48
IV.6 - Chaveamento de contexto num processador com vários conjuntos de registradores .....	50



## CAPÍTULO V:

V.1 - Arquitetura do T800 [26] .....	71
V.2 - Registradores do processador do T800 [27] .....	73
V.3 - Comunicação entre processos [26] .....	76
V.4 - Registradores do timer 0 [27] .....	78
V.5 - Diagrama em blocos da FPU [26] .....	79
V.6 - Relógio dos "transputers" [26] .....	81
V.7 - Gerenciamento de erro num sistema multi-transputer [27] .....	82
V.8 - Transputer com periféricos de controle .....	86
V.9 - Transputer com adaptadores de "link" .....	86
V.10 - Rede com comunicação ponto-a-ponto .....	88
V.11 - Interconexão entre "transputers" com 4 "links" cada um .....	88
V.12 - Representação dos processos que compõem o editor .....	92
V.13 - Diagrama do pacote gráfico interativo do exemplo anterior ....	93

## CAPÍTULO VI:

VI.1 - Comportamento do "transputer" em operações de processamento de imagem de baixo nível (operador de Sobel) ...	113
VI.2 - Diagrama básico mostrando as relações entre os programas desenvolvidos para o estudo da Transformada de Hough .....	115
VI.3 - Interface gráfica .....	116
VI.4 - Rede hipercúbica de dimensão 3 .....	118
VI.5 - Anel com 4 "transputers" mapeados na rede hipercúbica .....	122
VI.6 - Estrutura lógica básica dos programas para detecção de retas .....	122
VI.7 - Estrutura lógica básica dos processos de aplicação .....	123
VI.8 - Caso 1. (a) Estrutura lógica do processo de aplicação.	

(b) Divisão da imagem em duas partições .....	124
VI.9 - Configuração final da rede para o caso 1 .....	127
VI.10 - Caso 2.	
(a) Estrutura lógica do processo de aplicação.	
(b) Divisão do espaço de Hough em duas partições .....	126
VI.11 - Imagens usadas para comparação.	
(a) Imagem original.	
(b) Retas detectadas .....	130
VI.12 - Matriz acumuladora dividida em três partições .....	132
VI.13 - Mapeamento de um anel na rede hipercúbica .....	133
VI.14 - Mapeamento de uma árvore na rede hipercúbica .....	134
VI.15 - Cena utilizada para a confecção das tabelas VI.4 e VI.5	
(a) Imagem original.	
(b) Resultado da detecção .....	135
VI.16 - Comparação entre os dados das tabelas VI.4 e VI.5 .....	136
VI.17 - Comparação entre as estratégias da tabela VI.6.	
(a) Caso ideal ("Speed-up" linear).	
(b) Carga balanceada.	
(c) Carga desequilibrada .....	139

#### APÊNDICE I:

A.1 - Sistema NCP I .....	150
A2 - Arquitetura de um nó do NCP I .....	151

# ÍNDICE DE TABELAS

## CAPÍTULO V:

V.1 - Designações de pinos [27] .....	70
V.2 - Desempenho da FPU [30] .....	80
V.3 - Desempenho segundo o padrão de Wheatstone [30] .....	80

## CAPÍTULO VI:

VI.1 - Comparação entre o tempo de execução de rotinas de processamento de imagens escritas em "C" e OCCAM .....	113
VI.2 - Resultados obtidos para o caso 1 .....	128
VI.3 - Resultados obtidos para o caso 2 .....	129
VI.4 - Desempenho do Anel .....	136
VI.5 - Desempenho da Árvore .....	137
VI.6 - Comparação entre as distribuições de carga proposta anteriormente (partições iguais) e a nova (baseada no esforço computacional) .....	137

ÍNDICE DE ALGORITMOS.

CAPÍTULO II:

ALGORITMO 1 : ajuste.de.brilho .....	7
ALGORITMO 2: modulacao.de.contraste .....	7
ALGORITMO 3: realce.de.bordas .....	13
ALGORITMO 4: deteccao.de.retas .....	24
ALGORITMO 5: deteccao.de.circulos .....	30

## SUMÁRIO.

Este trabalho visa a avaliação de uma classe de algoritmos de visão computacional conhecida genericamente como Transformada de Hough em máquinas distribuídas baseadas em "transputers". Esta técnica pode ser utilizada para detectar curvas e formas em imagens digitalizadas, demandando grande quantidade de memória e considerável esforço computacional.

Optou-se por multiprocessadores baseados em "transputers" pelo fato desta tecnologia ter se mostrado bastante promissora em aplicações envolvendo computação gráfica. Adicionalmente, há poucos trabalhos no Brasil abordando a utilização de "transputers" em processamento de imagem e visão computacional.

Em primeiro lugar, esta pesquisa faz um levantamento dos aspectos teóricos envolvidos, tais como a caracterização dos problemas a serem tratados e dos tipos de máquinas usados em processamento de imagem. A seguir, são abordadas questões referentes à implementação da técnica em estudo e são discutidos os resultados obtidos com diversas topologias e organizações de dados e tarefas.

**ABSTRACT.**

This thesis intends to evaluate a class of computer vision algorithms known as Hough Transform in distributed machines based on transputers. This technique can be used for detecting curves and shapes in digitized images. It demands a lot of memory and computational effort.

Multiprocessors architectures based on transputers have been chosen because good results were obtained in computer graphics applications. Furthermore, there are few works in Brazil showing how transputers can be used in image processing and computer vision.

First of all, this research treats theoretical aspects related to problem characterization and types of image processing machines. In the second part, aspects concerning implementation of algorithms and practical results are discussed. The results obtained were generated by programs which exploited several kinds of topologies and process organizations. Different data organizations were also considered.

## AGRADECIMENTOS.

A Deus que me ajudou a superar todas as dificuldades.

Ao povo brasileiro por ter financiado este trabalho, através da CAPES.

Ao Marcos pelo amor, paciência, dedicação e carinho.

À minha mãe pelo amor e compreensão.

À Maria Helena pela grande amizade.

Aos colegas do LIN ( Laboratório de Instrumentação Nuclear)/COPPE/UFRJ pelo apoio técnico, pela utilização de seus equipamentos e pelos litros de café consumidos. Em particular, gostaria de agradecer ao engenheiro Nacao e aos físicos Ricardo e Ana.

Ao Núcleo de Computação Paralela/COPPE-Sistemas/UFRJ que forneceu os elementos fundamentais à realização deste trabalho.

Aos colegas da COPPE-Sistemas que prestaram uma grande contribuição ao meu trabalho, através de discussões, além de terem me incorporado ao grupo, concordando em dividir comigo uma escala de horários tão apertada. Em particular, gostaria de agradecer às seguintes pessoas: Anna, Ricardo Citro, Ricardo Corrêa, Lúcia, Clícia, Cristina e Cláudio Amorim.

Aos colegas do CBPF (Centro Brasileiro de Pesquisas Físicas) pelo auxílio durante a confecção da documentação deste trabalho.

Ao prof. Darcy Domingues Novo pela compreensão e apoio fraterno.

Ao PhD. Osamu Saotome pela orientação recebida e confiança.

Ao PhD. Flávio R. D. Velasco pela sugestão do tema da tese e pelas orientações iniciais.

Ao engenheiro Celso Luiz Mendes pelo auxílio durante a etapa inicial deste trabalho.



À Maria de Lourdes onde quer que  
esteja.

## CAPÍTULO I

### INTRODUÇÃO

Máquinas capazes de captar uma imagem, analisá-la e controlar processos a partir das informações visuais obtidas são bastante úteis em aplicações industriais e científicas. Associando-se conhecimentos de robótica, inteligência artificial e computação gráfica à visão por computador, pode-se obter sistemas melhores e mais funcionais. O estudo da visão computacional requer a identificação das características presentes na imagem e a representação simbólica das mesmas. Os principais problemas envolvidos são a compreensão do processo de visão e a sua posterior implementação num computador.

Uma das questões mais importantes relacionadas com a visão por computador é o processamento em tempo real. Sistemas capazes de operar em tempo real devem reconhecer padrões com suficiente rapidez para controlar processos industriais. Computadores de propósito geral são lentos, em se tratando da resolução de problemas complexos, devido ao esforço computacional necessário para o reconhecimento. Além do mais, algumas aplicações envolvem movimento de objetos, requerendo uma captura e um armazenamento rápidos de imagens.

Os problemas de processamento de imagens e visão por computador são caracterizados no Capítulo II. O estudo se concentrará numa técnica de visão conhecida como Transformada de Hough. Basicamente, esta classe de algoritmos aproxima contornos pertencentes a uma imagem digitalizada por curvas. Serão analisados os principais aspectos desta técnica, juntamente com alguns

algoritmos para processamento de baixo nível.

O uso de uma arquitetura convencional consome muito tempo de processamento e memória. Uma alternativa eficiente consiste na utilização de arquiteturas paralelas, onde vários processadores trabalham concorrentemente. A maior parte das máquinas paralelas existentes para processamento de imagens opera no modo SIMD, onde vários elementos de processamento (EPs) executam a mesma instrução sobre conjuntos de dados diferentes. Este tipo de arquitetura é mais adequado ao processamento de baixo nível, conforme será visto no Capítulo III. No Capítulo IV são definidos os aspectos lógicos e físicos das máquinas MIMD, pois esta é a classe de máquinas a ser estudada. Como as máquinas alvo são baseadas em "transputers", as características que motivaram tal escolha são discutidas no Capítulo V.

No Capítulo VI são discutidas as alternativas existentes para a implementação paralela dos algoritmos do Capítulo II, aproveitando as características apresentadas nos Capítulos III, IV e V. Explorou-se o paralelismo a nível de imagem e de tarefa. Também é feita uma análise do desempenho dos algoritmos vistos anteriormente, tomando-se como base o resultado de programas desenvolvidos para placas aceleradoras e para o computador NCP I. Neste trabalho - ao contrário do que foi proposto por BOWYER e LAKE [37] - as quantidades de memória local necessárias para cada nó foram reduzidas, pois a leitura dos "pixels" se dá concorrentemente com o processamento.

No Capítulo VII são mostradas as principais conclusões referentes ao uso da classe de máquinas estudada.

## CAPÍTULO II

### IDENTIFICAÇÃO DO PROBLEMA

#### II.1 - PROCESSAMENTO DE IMAGENS.

##### II.1.1 - Conceitos Básicos.

Por processamento digital de imagens entende-se a manipulação de uma imagem por computador de modo que a entrada e a saída sejam imagens. Na área de reconhecimento de padrões a entrada do processo é uma imagem e a saída constitui-se numa classificação ou descrição da mesma. O surgimento de arquiteturas de computadores poderosas, assim como o desenvolvimento de novos algoritmos destinados ao tratamento de sinais bidimensionais, está ampliando sensivelmente a gama de aplicações do processamento digital de sinais. Estes algoritmos podem ser aplicados em diversas áreas a saber: sensoriamento remoto, tomografia computadorizada, imageamento por ultra-som, transmissão de sinais digitais, automação industrial, ensaios não-destrutivos, etc.

Uma imagem pode ser definida como uma função  $I(x,y)$  [1] especificada para uma dada região do plano. Geralmente esta região é um subconjunto limitado do plano e os valores assumidos pela função são números inteiros, limitados e positivos. Para a maior parte das imagens, essa função mede, de alguma forma, a energia refletida por objetos e captada por um sistema de imageamento. Uma imagem digital pode ser vista como uma matriz de pontos com  $n$  linhas e  $m$  colunas, onde cada ponto assume um valor entre 0 e  $k-1$ . Usualmente, os valores de  $k$  são potências de 2. Ao valor  $I(n,m)$  dá-se o nome

de nível de cinza. Para ser passível de manipulação por computadores é necessário que as imagens tenham um formato digital. Há várias técnicas empregadas para melhorar a qualidade de uma imagem, tais como realce de bordas e manipulação de contraste [2,3].

O realce de bordas compreende operações de estreitamento e alargamento das bordas de uma imagem, a partir da informação contida na vizinhança de um pixel (elemento de imagem), e portanto alteram o contraste espacial desta.

A manipulação de contraste consiste numa transformação radiométrica de cada elemento de imagem, com o objetivo de aumentar a discriminação visual de imagens de baixo contraste.

Uma forma simples de análise de imagens é através do estudo de seu histograma. Ele descreve a distribuição estatística dos níveis de cinza em termos do número de amostras (quantidade de pixels) associado a cada nível. A distribuição pode também ser dada em termos da percentagem do número total de pixels presentes na imagem. Pode ser estabelecida uma analogia entre o histograma de uma imagem e a função densidade de probabilidade, que é o modelo matemático da distribuição de tons de cinza de uma classe de imagens.

#### **II.1.2 - Classificação dos Algoritmos.**

O processamento de imagens é uma técnica que se encarrega de modificar e analisar figuras. Existem várias classes de algoritmos utilizados para este fim:

- 1) Algoritmos para processamento de pontos : mudam o valor de um pixel, baseando-se apenas no valor atual do mesmo.
- 2) Algoritmos para processamento de área : mudam o valor de um pixel, baseando-se no seu valor e nos valores dos pixels vizinhos.
- 3) Algoritmos para processamento geométrico : mudam a posição ou o arranjo dos pixels.
- 4) Algoritmos para processamento de quadro : mudam o valor dos pixels a partir de uma comparação entre duas ou mais imagens (quadros).

O processamento de imagens pode ser empregado com várias finalidades, dentre as quais podem ser mencionadas:

- o realce ou a modificação da imagem de modo a melhorar o seu aspecto ou ressaltar determinadas informações;
- a medição de elementos de imagem;
- a classificação;
- o casamento de elementos de imagem;
- o reconhecimento de itens presentes na mesma.

As medições feitas numa imagem baseiam-se em algumas hipóteses a respeito do conteúdo de uma determinada figura, enquanto os processos de classificação e reconhecimento requerem bastante conhecimento sobre o que pode aparecer numa imagem. Por exemplo, pode-se medir quantos pixels pertencentes à uma imagem proveniente de um satélite estão contidos dentro

de uma determinada faixa de valores. Caso estes valores estejam associados a uma cultura de trigo, a imagem pode ser classificada como sendo composta de duas classes de elementos: trigo e não-trigo. Se a máquina for dotada de algum conhecimento referente à estrutura de um campo de trigo, ela poderá ser capaz de reconhecer estes campos numa imagem. Pode-se utilizar uma falsa cor para ressaltar as áreas reconhecidas. Outras formas de classificação dos algoritmos de processamento de imagens são:

- algoritmos baseados em imagem e em métodos simbólicos;
- algoritmos lineares e não-lineares;
- algoritmos que realizam a classificação segundo o nível de conhecimento utilizado.

Os algoritmos baseados em imagens transformam os valores atuais dos pixels em outros valores ou posições, por meio de operações lógicas ou numéricas, enquanto os algoritmos simbólicos manipulam conhecimento a respeito da estrutura dos pixels. O grau de conhecimento usado por um algoritmo pode variar desde simples suposições acerca da formação da imagem até o conhecimento específico do mundo em questão e dos diversos itens que compõem uma cena. O exemplo dado acima, de medição, classificação e reconhecimento de um campo de trigo, ilustra a dimensão do problema dos níveis de conhecimento envolvidos.

### **II.1.3 - Algoritmos para Processamento de Pontos.**

Estes algoritmos varrem a imagem ponto por ponto e usam o valor e/ou o

endereço do pixel para o cálculo do novo valor do mesmo.

O ALGORITMO 1, por exemplo, altera o brilho de uma imagem adicionando uma constante ao valor de cada um de seus pixels.

**ALGORITMO 1 : ajuste.de.brilho**

```
lin = contador de linha
col = contador de coluna
fator = fator de brilho
repita
    pixel[lin][col] = pixel[lin][col] + fator
ate final da imagem
```

O contraste entre áreas pode ser ajustado ou realçado através de mudanças suaves dos valores dos pixels. O ALGORITMO 2 modula o contraste de uma imagem, segundo a função abaixo:

$$f(x,y) = I(x,y).k.e^{(-(x^2 + y^2)/l)} - m$$

**ALGORITMO 2: modulacao.de.contraste**

```
lin = contador de linha
col = contador de coluna
repita
    calcular f(lin,col) para o pixel[lin][col]
ate o final da imagem
```



Se o valor do pixel e a sua localização forem utilizados, então estes algoritmos poderão ser empregados para corrigir o sombreamento ou suavizar mudanças abruptas nos valores dos pixels. Sombreamento é um artifício provocado por pequenos deslocamentos na iluminação da cena ou pelas características da câmera usada. Um processo que aplica aos pontos uma função inversa a de sombreamento pode corrigir o seu efeito.

O histograma é uma técnica de medição e pode ser classificado como um processamento de ponto, pois envolve o exame de um pixel de cada vez, embora os valores dos pixels em si não sejam alterados. A informação fornecida por um histograma é útil para classificar e realçar uma imagem. Para gerar um histograma de intensidades, conta-se o número de vezes que uma intensidade ocorre, numa dada área de imagem. A título de exemplo, utilizando-se uma palavra de 8 bits por pixel serão possíveis 256 valores de intensidade.

O contraste também pode ser melhorado a partir da análise de um histograma. Em primeiro lugar, estabelece-se um certo intervalo de variação de intensidades, que no caso de uma imagem com 256 tons de cinza pode ser, por exemplo, de 0 a 30. Se a maior parte dos pixels estiver contida nesta faixa, então os demais poderão ser descartados, de modo que os pixels remanescentes possam ser redistribuídos ao longo do intervalo de 0 a 255. A imagem resultante apresentará um contraste melhor. Convém ressaltar que, em geral, as operações de processamento de imagem implicam numa perda de informação, como consequência da seleção ou acentuação de uma característica ou dado.

Os pixels que estão compreendidos num certo intervalo de intensidades podem ser realçados se for atribuída uma dada cor aos mesmos, enquanto os

outros são mostrados em tons de cinza. Este processo é conhecido como pseudo-coloração.

#### III.1.4 - Algoritmos para Processamento de Área.

O processamento de área trabalha com a informação disponível na vizinhança de um pixel ou verifica a existência de alguma propriedade envolvendo os pontos da imagem. Como exemplos deste tipo de processamento podem ser citadas as seguintes operações:

- filtragem espacial;
- realce de contornos;
- busca de características;
- remoção de ruído;
- suavização;
- borramento.

##### III.1.4.1 - Convolução.

A convolução é um exemplo clássico deste tipo de algoritmo. Ela é comumente utilizada para filtragem espacial e na busca de características da imagem. Este procedimento requer um grande esforço computacional e, portanto, a sua implementação deve ser feita levando-se em consideração todos os aspectos que possam ser usados para minimizar os cálculos. A convolução é uma operação que substitui o valor de um pixel pela soma do valor atual do mesmo com os valores dos seus vizinhos, cada qual multiplicado por um fator. Os

fatores usados são comumente agrupados numa máscara. Seja uma máscara 3 X 3, cujos elementos são  $k(x,y)$  e o pixel  $p(x,y)$ . A equação que define tal operação é:

$$p(x,y) = \sum_{m,n=0}^2 k(m,n)*p(x+m,y+n)$$

Tal processo pode ser melhor visualizado, observando-se que ele equivale ao deslizamento da máscara sobre a matriz que compõe a imagem. Para cada um dos pontos, os valores que se situam sob a máscara são multiplicados pelos valores correspondentes da mesma e a seguir somados. O resultado é o novo valor do pixel.

Para efetuar a convolução de uma área de dimensões  $X \times Y$  com uma máscara de tamanho  $n \times m$  serão necessárias  $X*Y*n*m$  multiplicações e somas. A convolução de uma imagem de 256 por 256 pontos com uma máscara 3 X 3 resultará em 589824 multiplicações e somas. Isto requer muito tempo de processamento, caso não seja utilizado um hardware rápido.

Em se tratando da implementação deste algoritmo, devem ser consideradas várias questões. Primeiramente, o resultado de uma convolução não pode ser colocado na memória tão logo seja obtido, pois o efeito resultante seria análogo ao de um filtro de resposta impulsiva. Isto se deve ao fato de que o valor atual de um pixel poderá ser usado em outras convoluções, já que faz parte da vizinhança de outros pixels. Para resolver este problema, emprega-se um "buffer", o qual armazenará temporariamente os valores de saída dos pixels, substituindo-os na área a ser modificada assim que a convolução for concluída. Um outro procedimento seria a escrita dos elementos numa área de memória que já tenha sido submetida à convolução.

Um outro problema diz respeito às bordas da imagem. Por exemplo, seja uma máscara 3 X 3. Quando o centro da máscara estiver sobre um pixel da borda da imagem, haverá um trecho desta sob o qual não existirão pixels. O resultado da convolução nas bordas será sempre uma borda de conteúdo indeterminado e que pode ser ignorada fazendo-se todos os pixels iguais a 0, ou pode-se estimar valores para os mesmos. O tamanho da máscara geralmente é ímpar. Caso não seja, as bordas serão assimétricas, pois não é possível a substituição de um valor no centro de um conjunto par de pixels.

Uma terceira questão diz respeito ao resultado da convolução em si, o qual pode resultar num número cuja representação excede aquela permitida pelo número de bits alocado para cada pixel. A probabilidade de ocorrência deste problema cresce a medida que o número de elementos da máscara aumenta. Tomando-se o exemplo anterior e usando-se uma máscara 3 X 3, o valor máximo possível é  $3 \cdot 3 \cdot 256 = 2304$ , o que implica no uso de uma palavra de 12 bits. Como se supôs se que um pixel seria representado por uma palavra de 8 bits, é necessário algum recurso para trabalhar-se com tal valor. Uma alternativa seria aumentar o tamanho das palavras usadas no cálculo, de modo a aumentar a precisão dos mesmos. Outra possibilidade seria fazer uma transformação em escala, como por exemplo, dividindo-se os resultados por 2.

Os fatores presentes na máscara podem ser positivos e negativos, o que implica num resultado positivo ou negativo para a convolução.

Concluindo, a convolução é uma operação relativamente simples, mas que envolve problemas relacionados com a sua implementação. A questão relativa ao tratamento das bordas está presente na maioria dos problemas de processamento de área. A aplicação da convolução a uma imagem pode ser

interpretada como sendo um filtro casado ou um filtro espacial. No filtro casado, a máscara usada na convolução é, essencialmente, uma pequena imagem daquilo que se deseja ampliar ou detectar.

A segunda interpretação da convolução é aquela que a vê como sendo uma filtragem espacial. Em se tratando de imagens, frequência espacial é o número de vezes que um dado padrão se repete por unidade de distância. Uma imagem pode ser representada por uma série de senóides e cossenóides com frequências espaciais distintas. Isto pode ser efetuado através de uma FFT, a qual deve ser aplicada horizontalmente e verticalmente, pois há frequências espaciais em ambas as direções.

A seleção de uma certa banda de frequências pode ser efetuada por uma máscara. Baixas frequências espaciais podem ser associadas a mudanças suaves de intensidade, enquanto alterações bruscas equivalem a altas frequências. Estas últimas podem ser detectadas pela máscara abaixo:

-1 -1 -1

-1 8 -1

-1 -1 -1

Esta máscara é chamada de filtro laplaciano, pois se aproxima de uma derivada de segunda ordem e pode ser usada para extrair contornos de uma imagem, já que a sua saída são os pixels associados às bordas. Se o filtro laplaciano for ligeiramente modificado, de forma que o elemento central seja 9, ao invés de 8, o resultado será equivalente à soma da saída de um filtro laplaciano com a imagem original. Isto ocorre devido ao fato de uma máscara

cujo elemento central é igual a 1 e os demais são nulos fornecer como saída a própria imagem. A mesma resultante apresenta contornos mais nítidos e contém ruído adicional. O ALGORITMO 3 pode ser usado para exibir no vídeo uma imagem com as altas frequências espaciais (bordas) realçadas.

**ALGORITMO 3: realce.de.bordas**

```
lin = contador de linha
col = contador de coluna
masc = mascara (elemento central 9 e demais iguais a 1)
repita
  inicio
    temp = convolucao de masc com o pixel[lin][col]
    mostra a nova intensidade
  fim
ate o final da imagem
```

O poder da convolução reside no uso da informação presente numa dada área de imagem para a alteração da característica de um ponto. Em resumo, os operadores de bordas caracterizam um pixel como pertencente ou não a um contorno, partindo da premissa de que uma borda se estende ao longo de certa distância. O ponto chave de uma operação de convolução reside na escolha da máscara correta.

#### II.1.4.2 - Processamento Não-linear de Área.

A convolução é uma operação simples, pois é linear, ou seja, requer apenas somas de produtos de grau um. Operações não-lineares, embora sejam um pouco mais complicadas, são também úteis e podem proporcionar uma relação sinal-ruído melhor. Podem ser usadas para detectar elementos característicos, com menor esforço computacional. A seguir são considerados dois exemplos de processamentos desta natureza.

##### (1) Filtro de Sobel

Este filtro compara o resultado de duas convoluções, para então estimar a orientação ( $\theta$ ) e o módulo ( $m$ ) da variação das intensidades nas bordas. As duas máscaras usadas nestas convoluções são:

$$\begin{array}{ccc} & -1 & 0 & 1 \\ \text{Direção X :} & -2 & 0 & 2 \\ & -1 & 0 & 1 \end{array} \quad \text{e} \quad \begin{array}{ccc} & 1 & 2 & 1 \\ \text{Direção Y :} & 0 & 0 & 0 \\ & -1 & -2 & -1 \end{array}$$

Os valores de  $m$  e  $\theta$  são obtidos através das funções abaixo:

$$m = \sqrt{X * X + Y * Y} \quad \text{e} \\ \theta = \arctan(Y/X),$$

onde  $X$  e  $Y$  são as coordenadas do pixel.

Este filtro é um bom detetor de bordas e é freqüentemente usado como um dos primeiros passos em se tratando de algoritmos de visão por computador.

Apresenta as seguintes desvantagens: é computacionalmente dispendioso e é afetado por ruído.

#### (11) Filtro de Mediana.

Este filtro ordena os pontos da vizinhança de  $P(x,y)$  de forma ascendente e assume como novo valor para  $P(x,y)$  o valor intermediário desta ordenação. O efeito resultante é a remoção do ruído.

#### II.1.4.3 - Classificação.

Para classificar os elementos de uma imagem é necessário algum conhecimento a respeito dos mesmos. A título de exemplo, seja um elemento constituído por a um grupo conexo de pixels com o mesmo valor. Assumir-se-á que um pixel é conexo quando este possuir o mesmo valor que seus vizinhos situados a  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  e  $270^\circ$ . Para simplificar os cálculos, a imagem pode ser transformada em binária, ou seja, todos os pixels cujos valores forem inferiores a um dado limiar serão considerados nulos. Os demais serão igualados a 255. Este procedimento é freqüentemente usado em problemas de visão por computador. Fixa-se um número mínimo de pixels conexos acima do qual um grupo será considerado como um elemento. Assim procedendo, evita-se manchas introduzidas na imagem por ruído e pelo limiar adotado. Pode-se então separar os elementos atribuindo-se a eles uma falsa cor e ainda armazenar o número de pixels por elemento. Esta informação pode ser usada em futuras classificações.

A maior dificuldade para a implementação deste exemplo é a determinação dos grupos de pixels conexos. Uma das formas mais simples de se implementar



tal operação é através de uma busca recursiva. O procedimento descrito é um processamento de área, mas existem outros que utilizam uma combinação de processamento de ponto e de área.

#### II.1.5 - Algoritmos Geométricos.

Esta classe de algoritmos muda o arranjo espacial dos pixels. São geralmente empregados para corrigir as distorções provocadas pelo sistema ótico da câmera ou pelo ponto de vista do observador e para aumentar uma certa área de imagem ("zoom"). Como exemplos de processamentos de imagens geométricos típicos pode-se mencionar as seguintes operações: rotação, translação, alongamento e distorção da imagem. Os algoritmos geométricos podem ser expressos através de um conjunto de equações dispostas numa matriz. Desta forma, um pixel localizado no ponto (x,y) será mapeado em uma nova posição (x',y'). Por exemplo, uma área quadrada pode ser girada de 90°, utilizando-se o seguinte mapeamento:

$$x' = (\text{LARGURA} - 1) - y$$

$$y' = x$$

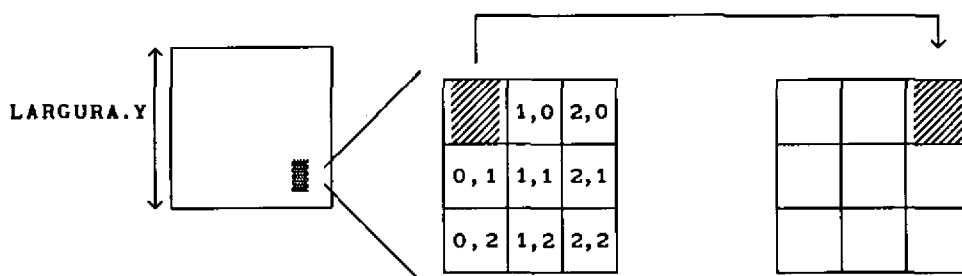


Figura II.1 - Mapeamento do pixel (0,0) em (2,0)

Uma outra implementação para a rotação é a leitura das linhas horizontais de pixels da imagem fonte, que seriam rescritas segundo uma nova inclinação.

Muitas transformações geométricas apresentam como efeito colateral o aparecimento de lacunas entre os pixels da imagem de saída, caso seja feito um mapeamento direto a partir das equações que as definem. Trata-se de um problema típico de imagens digitalizadas. Por exemplo, seja uma certa área de imagem, que terá as suas dimensões  $x$  e  $y$  duplicadas. Os pixels da imagem fonte serão mapeados em outras posições, o que acarretará o aparecimento de lacunas. A solução consiste na inversão das equações de mapeamento, de modo que a área de destino seja varrida ponto-a-ponto. Para cada ponto de destino, as equações invertidas serão analisadas.

Quando é usada uma aproximação baseada no vizinho mais próximo, produzir-se-á uma mudança repentina na intensidade da imagem, conferindo-lhe um aspecto de bloco. A interpolação produz resultados mais suaves.

#### **II.1.6 - Algoritmos para Processamento de Quadros.**

São aqueles que processam mais de uma imagem. Um exemplo bastante simples é a subtração de imagens. As diferenças resultantes podem ser usadas para comparar duas cenas, o que é útil em se tratando da detecção de peças ou partes ausentes numa dada máquina. O processamento de quadro também pode ser empregado na detecção de movimento e na melhoria da qualidade da imagem.

A detecção de movimento pode ser realizada a partir do resultado de uma

subtração de quadros. Esta operação se assemelha a uma derivação em relação ao tempo e pode ser útil em controle de qualidade e para avaliação de movimento. Como a subtração de imagens requer um esforço computacional muito grande, é necessário um hardware dedicado a esse processamento e que seja capaz de realizá-lo em tempo real, ou seja, que trabalhe segundo as taxas de vídeo.

## II.2 - VISÃO POR COMPUTADOR.

O termo visão por computador refere-se à extração de informação sobre uma cena, através da análise da mesma. Em realidade, trata-se de um processamento de informação cuja entrada é um conjunto de imagens e a saída é uma descrição da cena [4]. O problema consiste na recuperação da descrição da cena, a partir de um conjunto incompleto de dados. Isto é feito inferindo-se a informação perdida através de algum conhecimento do mundo em estudo. Existem basicamente cinco processos associados à visão por computador [40]:

- a entrada (captação da imagem);
- amostragem e quantização;
- pré-processamento;
- extração de características;
- reconhecimento de objetos.

A fonte de informação pode ser um equipamento de raios X, tomografia, câmeras de vídeo, radar, etc.

A amostragem e a quantização são críticas e dependem da velocidade e da qualidade dos conversores analógico-digitais utilizados para a digitalização dos sinais de entrada.

O pré-processamento é usado para melhorar a imagem de modo que a mesma possa ser tratada pelos estágios subsequentes. As operações de ajuste de contraste e remoção de ruído são exemplos típicos.

A extração de características e o reconhecimento de objetos são algumas das funções inteligentes desempenhadas por um sistema de visão por computador. O presente trabalho possui aplicações nesta etapa de processamento, pois aborda uma classe de algoritmos conhecidos como Transformada de Hough, destinados à detecção de curvas e formas arbitrárias. Outras funções que podem ser mencionadas são: identificação, descrição e mapeamento tridimensional.

Os principais problemas relacionados a visão computacional são:

- as condições ambientais (que são difíceis de serem controladas, principalmente em ambientes industriais);
- a falta de flexibilidade das máquinas desenvolvidas para este fim (os sistemas existentes são, em geral, destinados a aplicações específicas);
- o "software", que deve ser capaz de resolver situações ambíguas, tais como a identificação de partes com cores diferentes ou com defeitos de fabricação [5,6].

## II.3 - DETECÇÃO DE CURVAS USANDO TRANSFORMADA DE HOUGH.

### II.3.1 - Caracterização do Problema.

Um problema comum na área de visão por computador é a identificação de curvas em imagens digitalizadas. No caso mais simples, a figura contém um número fixo de pontos claros sobre um fundo escuro e o problema consiste na detecção de grupos de pontos colineares ou quase colineares. A solução pode ser obtida testando-se as linhas formadas por todos os pares de pontos. O esforço computacional necessário é proporcional a  $n^2$ , para uma imagem com  $n \times n$  pixels, o que torna o procedimento proibitivo para valores de  $n$  muito grandes.

A maior parte da informação pertinente a um objeto reside no seu contorno. Imagens cujos contornos estão codificados sob uma forma bruta são em geral suficientes para fins de reconhecimento de objetos.

Inicialmente, a imagem a ser processada pode ser codificada de modo a conter apenas as informações sobre os contornos dos objetos presentes na mesma (intensidades locais ou gradientes de cor). Esta representação é chamada de esboço primário ("primal sketch") e é o primeiro passo em se tratando de processamento de imagens de alto nível. Há algoritmos bastante genéricos para detecção de objetos de uma determinada forma, a partir de uma imagem que contenha apenas informação relativa aos seus contornos. Neste tipo de representação, os pontos amostrados não contêm os valores dos níveis de cinza, mas a informação sobre o módulo e a direção do vetor gradiente em cada ponto. Tais características indicam, respectivamente, a severidade e a orientação de uma mudança local de nível de cinza.

### II.3.2 - Transformada de Hough para Linhas Retas.

Hough desenvolveu um método que substitui o problema da busca de pontos colineares por um outro, matematicamente equivalente, de busca de linhas concorrentes [7]. Este método mapeia os pontos da figura que estão contidos numa mesma linha reta num único ponto, pertencente a um espaço de parâmetros (espaço de Hough), cujas dimensões são os parâmetros usados para descrever as linhas no plano cartesiano. Hough escolheu como parâmetros a inclinação da reta e o ponto no qual a mesma intercepta o eixo y. O fato dos parâmetros não serem limitados dificulta o uso desta técnica [8].

A transformada de Hough é aplicada a pixels de borda, que podem ser obtidos utilizando-se operadores de borda. O detector de bordas escolhido foi o filtro de Sobel, que calcula valores aproximados para o módulo  $m$  e a direção  $\theta$  do vetor gradiente de intensidade (ou nível de cinza) em cada pixel da imagem. Seja o pixel P, cujos vizinhos possuem os níveis de cinza dados a seguir:

A B C

D P E

F G H

Em primeiro lugar, os valores das diferenças de Sobel ( $\Delta x$  e  $\Delta y$ ) devem ser calculados:

$$\Delta x \equiv \frac{1}{4} [(C + 2E + H) - (A + 2D + F)]$$

$$\Delta y \equiv \frac{1}{4} [(A + 2B + C) - (F + 2G + H)]$$

O módulo e a direção são definidos como sendo (item II.1.4.2):

$$m \equiv \sqrt{\Delta x^2 + \Delta y^2} \quad \text{e} \quad \theta \equiv \text{tg}^{-1}(\Delta y/\Delta x) \quad (\text{II.1})$$

P será considerado um pixel de borda se o valor de  $m$  em P exceder um limiar pré-estabelecido pelo programador.

Uma linha reta pode ser caracterizada pela sua inclinação  $\alpha$  e pela distância  $\rho$  (figura II.2), que é o comprimento da perpendicular à mesma, passando pela origem. Para uma linha com inclinação  $\alpha$  e que passa pelo ponto  $(x,y)$ , tem-se que:

$$\rho = x \cdot \text{sen}\alpha - y \cdot \text{cos}\alpha \quad (\text{II.2})$$

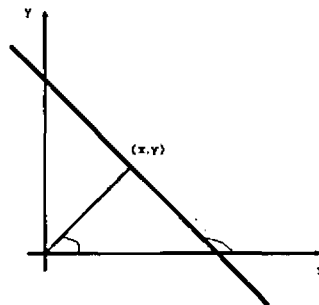


Figura II.2 - Representação paramétrica de uma linha reta.

Convém observar que  $\rho$  possui sinal. Se  $\alpha$  estiver no primeiro e no quarto quadrantes,  $\rho$  será negativo.

Considere uma imagem que contenha uma longa linha reta, cuja inclinação é  $\alpha$  e que dista  $\rho$  em relação à origem. Então um pixel pertencente a esta borda, e com coordenadas  $(x,y)$ , deve ter uma direção  $\theta$  para o gradiente, perpendicular a  $\alpha$ , ou seja,  $\alpha = \theta \pm \pi/2$  (figura II.2). Reescrevendo-se a equação de  $\rho$  em termos de  $\theta$  tem-se:

$$\rho = x.\cos\theta + y.\sen\theta \quad (\text{II.3})$$

Na prática, os grupos de pixels obtidos não correspondem precisamente às bordas da imagem em virtude do uso de uma vizinhança 3x3. Uma alternativa para a correção desta distorção é variar  $\theta$  em torno do valor obtido com o operador de Sobel. Para um dado par  $(x,y)$  e um intervalo  $\Delta\theta$ , os pares  $(\rho,\theta)$  resultantes situam-se sobre um arco, no plano  $(\rho,\theta)$ , o qual passa pelo valor de  $(\rho,\theta)$  nominal. Caso a imagem contenha uma longa linha reta, estes arcos se interceptarão num ponto. O par  $(\rho,\theta)$  associado ao mesmo é o conjunto de parâmetros procurado.

O maior valor possível para  $\rho$  numa imagem com  $n \times n$  pixels é  $n\sqrt{2}$ . O número de linhas digitalizadas distintas que podem passar através de um dado ponto é também da ordem de  $n$ , ou seja, essas linhas possuem  $O(n)$  inclinações diferentes. Portanto, quando se trabalha com uma imagem de entrada de  $n \times n$  pontos, pode-se utilizar um espaço digital  $(\rho,\theta)$  com dimensões da ordem de  $n \times n$  (matriz de Hough ou matriz acumuladora). O processo para o cálculo da Transformada de Hough está resumido no ALGORITMO 4. Para efeito de correção de erros (item II.3.3.3), toma-se o intervalo  $[\theta(x,y) - \theta_0, \theta(x,y) + \theta_0]$ ,



onde  $\theta_0$  é uma constante dada. Após o cálculo da transformada, cada posição da matriz conterá o número de vezes que um par  $(\rho, \theta)$  foi obtido como resultado do processamento dos pixels de contorno; assumiu-se que existem  $k$  valores de  $\theta$  no intervalo  $[\theta(x,y) - \theta_0, \theta(x,y) + \theta_0]$ .

**ALGORITMO 4: deteccao.de.retas**

```

P(x,y) = pixel da posicao (x,y)
 $\theta_0$  = metade do intervalo de variacao do angulo  $\theta(x,y)$ 
inicio
  repita
    calcular modulo do vetor gradiente ( $m(x,y)$ ) em P(x,y)
    calcular a direcao do vetor gradiente em P(x,y)
    se o P(x,y) pertencer a um contorno, entao
      inicio
        repita
          calcular  $\rho$ 
          incrementar a celula  $(\rho, \theta)$ 
        ate varrer todo o intervalo  $2\theta_0$ 
      fim
    ate o final da imagem
  localizar as curvas presentes na imagem
fim

```

O valor de  $k$  não depende somente de  $\theta_0$ , mas também de  $n$ , pois  $k = 2.\theta_0/360/n = n.\theta_0/180$  e é igual ao número de incrementos a serem dados a  $\theta$  dentro de um intervalo  $2.\theta_0$ . Por exemplo, se  $\theta_0 = 15^\circ$  e  $n = 128$ ,  $k$  será

igual 10. Quando  $\theta$  é estimado através do operador de Sobel, um valor de  $\theta_0$  igual a  $15^\circ$  parece ser razoável, pois conduz a  $k = n/12$ . Pode-se empregar um operador gradiente com uma máscara maior para tornar  $k$  uma fração menor de  $n$ , o que não aumentará significativamente o tempo de processamento, já que a maior parte dele é gasta computando a transformada e não com o cálculo do gradiente.

### II.3.3 - Transformada de Hough para Curvas Analíticas.

Curvas analíticas são aquelas que podem ser expressas sob a forma  $f(\mathbf{x}, \mathbf{a}) = 0$ , onde  $\mathbf{x}$  é um ponto da imagem e  $\mathbf{a}$  é um vetor de parâmetros [9]. Seja o problema da detecção de bordas circulares. A equação de um círculo em coordenadas cartesianas é dada por:

$$(x - a)^2 + (y - b)^2 = r^2$$

Suponha-se também que a imagem codificada contenha apenas informações acerca dos seus contornos, ou seja, somente os módulos das mudanças locais de intensidade são conhecidos. Caso um pixel pertença a um círculo, o lugar geométrico dos parâmetros que o descrevem é um cone circular (figura II.3), que pode ser obtido fixando-se  $x$  e  $y$  e variando-se  $a$ ,  $b$  e  $r$ . Se um conjunto dos pixels pertencentes a um contorno residir sobre um círculo, cujos parâmetros são  $a_0$ ,  $b_0$  e  $r_0$ , os lugares geométricos dos parâmetros de cada par  $(x, y)$  se interceptarão num mesmo ponto do espaço paramétrico. Isto equivale à intersecção dos correspondentes cones circulares num ponto  $(a_0, b_0, r_0)$  dentro do referido espaço.

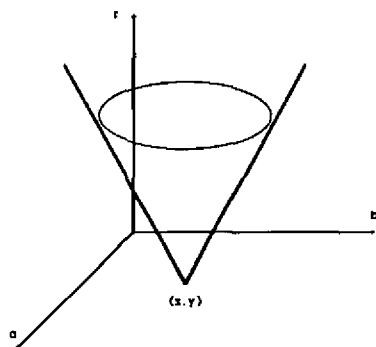


Figura II.3 - Lugar geométrico dos parâmetros de um círculo.

#### II.3.3.1 - Efeito do Uso da Informação sobre o Sentido de Variação das Intensidades nos Contornos.

O lugar geométrico dos parâmetros de um círculo será reduzido a uma linha (figura II.4), caso seja acrescentada alguma informação sobre o sentido de variação das intensidades nos contornos. Formalmente o círculo requer três parâmetros. Se a equação que o descreve for usada em conjunto com a sua derivada, simplifica-se o cálculo:

$$\frac{df(\mathbf{x}, \mathbf{a})}{dx} = 0$$

Esta operação introduz um termo em  $dy/dx$ , dado por:

$$\frac{dy}{dx} = \text{tg}[\phi(\mathbf{x}) - \pi/2], \quad \text{onde } \phi(\mathbf{x}) \text{ é a direção do gradiente.}$$

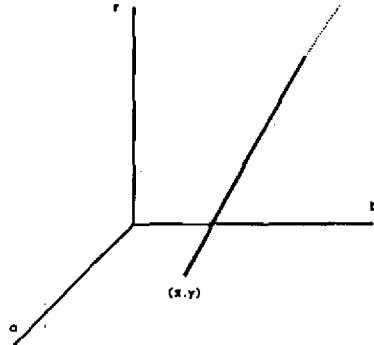


Figura II.4 - Lugar geométrico dos parâmetros de um círculo com informação sobre o sentido de variação da intensidade.

#### II.3.3.2 - Algoritmo Genérico para Curvas Analíticas.

Dada uma curva  $f(x, a) = 0$ , onde  $a$  é o vetor de parâmetros, forma-se uma matriz  $A(a)$ , cujos elementos são inicialmente nulos (matriz acumuladora). Para cada pixel pertencente a um contorno, calcula-se todos os valores de  $a$ , para os quais  $f(x, a) = 0$  e  $df(x, a)/dx = 0$ , incrementando-se as correspondentes entradas da matriz acumuladora:

$$A(a) = A(a) + 1$$

Após a repetição do procedimento acima para todos os pixels pertencentes aos contornos, os pontos da matriz acumuladora onde forem registrados máximos corresponderão a curvas na imagem.

Usando-se apenas a equação  $f(\mathbf{x}, \mathbf{a}) = 0$ , o número de cálculos necessários crescerá exponencialmente com o número de parâmetros menos 1. Seja  $m$  o número de parâmetros, os quais podem assumir  $M$  valores, logo a quantidade de cálculos é proporcional a  $M^{m-1}$ , pois a equação da curva é usada para determinar o último parâmetro. O uso do vetor gradiente diminui o número de cálculos. O esforço computacional é proporcional a  $M^{m-2}$ , para  $m \geq 2$ .

O método pode ser estendido a curvas e formas arbitrárias, bastando escolher um conjunto adequado de parâmetros. Por exemplo, a altura, a largura e as coordenadas do centro de um retângulo podem ser usadas para caracterizá-lo. Quanto mais sofisticadas forem as formas a serem detectadas, maior será o número de parâmetros a serem levados em conta. O processamento em tempo real desta transformada requer uma grande quantidade de memória e esforço computacional considerável.

#### II.3.3.3 - Compensação de Erros.

O maior problema apresentado pela Transformada de Hough é a determinação dos máximos da matriz acumuladora ( $A(\mathbf{a})$ ). Há muitas fontes de erro que podem afetar o vetor de parâmetros  $\mathbf{a}$ , de modo que vários pontos na vizinhança do ponto ideal sejam incrementados. Uma maneira de resolver este problema é modelar formalmente os erros na etapa de incrementação. O modelo especificará um conjunto de pontos próximos, em vez de apenas um ponto. Uma outra solução é tentar compensar os valores contidos na matriz acumuladora através de uma função, após concluída a etapa de incrementação. O efeito desta operação é suavizar o conteúdo da matriz. Estes métodos são equivalentes, supondo-se erros isotrópicos.

No caso da detecção de círculos, a suavização da matriz acumuladora é equivalente a uma alteração no procedimento de incrementação, de modo a introduzir os efeitos das incertezas no gradiente  $\phi$  e no raio  $r$ :

$$\phi(x) \pm \Delta\phi \quad \text{e} \quad r \pm \Delta r(r)$$

Todos os valores de  $a$  localizados no interior da faixa sombreada da figura II.5 serão incrementados. A incerteza  $\Delta r$  é proporcional a  $r$  (percentagem de  $r$ ). A idéia básica é modular os vetores de parâmetros  $A'(a)$  localizados no interior de um domínio quadrado por uma função de suavização  $h$ , que leve em conta o espalhamento dos pontos. O ALGORITMO 5 leva em consideração este procedimento.

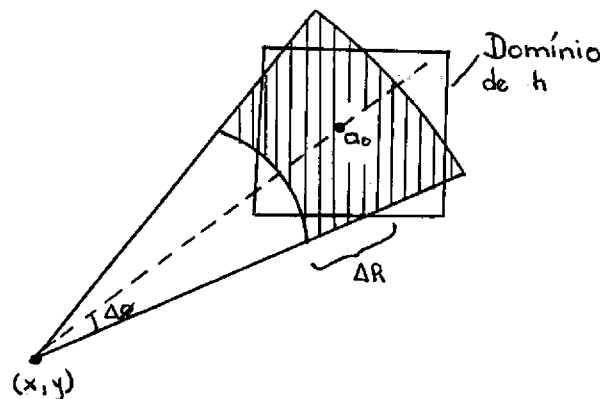


Figura II.5 - Região onde é feita a compensação de erros.

**ALGORITMO 5: deteccao.de.circulos**

$P(x,y)$  = pixel da posicao  $(x,y)$

inicio

  repita

    calcular o modulo do gradiente

    calcular a direcao do gradiente

    se o pixel  $P(x,y)$  pertencer a um contorno, entao

      repita

        repita

          repita

            calcular as coordenadas do centro do circulo

            incrementar a celula  $(x,y,r)$

            ate varrer o intervalo  $\pm\Delta\phi$

            ate varrer o intervalo  $\pm\Delta r$

            ate pesquisar todos os raios

  ate o final da imagem

  localizar curvas

### CAPÍTULO III

## TAXIONOMIA DAS ARQUITETURAS DE COMPUTADORES DESTINADAS AO PROCESSAMENTO DE IMAGENS E À VISÃO COMPUTACIONAL

Neste capítulo será feito um levantamento das características mais importantes a serem consideradas no projeto de uma máquina destinada ao processamento digital de imagens.

### III.1 - CARACTERÍSTICAS.

As áreas de processamento de imagens e de visão computacional tratam problemas cujas propriedades dificultam a obtenção de soluções através do uso de computadores de propósito geral. A maior parte dos algoritmos envolve a execução de um mesmo conjunto de operações para todos os pixels de uma imagem relativamente grande, o que requer grandes velocidades de processamento. No capítulo anterior foram discriminadas duas classes de problemas [10,11]:

#### a) Processamento de imagens ou processamento de imagens de baixo nível.

Lida com operações tais como: correção geométrica, detecção de contornos, remoção de ruído, extração de características, contraste, etc. Estas tarefas apresentam as seguintes propriedades:

- a matriz de saída possui o mesmo tamanho da matriz de entrada;



- uma mesma operação é aplicada a todos os pixels da imagem;
- os cálculos feitos para um pixel envolvem apenas os pixels pertencentes à sua vizinhança local.

#### **b) Visão computacional ou processamento de imagens de alto nível.**

Os problemas de visão incluem, entre outros, a classificação de segmentos ou de características da imagem, separando-os em classes conhecidas. Em muitos casos, a imagem a ser processada não se encontra mais sob a forma de uma matriz de pixels. Neste nível, as estruturas mais adequadas à representação de segmentos de imagem usam um conjunto de parâmetros ou baseiam-se em grafos. A identificação dos processos seqüenciais, que podem ser executados em paralelo, é feita através de uma análise de grafos.

As arquiteturas destinadas ao processamento de imagens e à visão devem levar em consideração os fatores abaixo [12]:

#### **1) Aplicações.**

Deve-se ter em mente qual é a classe de problemas a ser tratada pela arquitetura, ou seja, se o processamento é de baixo ou de alto nível. Não há uma separação abrupta entre estes dois níveis e, freqüentemente, existe grande interação entre eles.

#### **2) Especificações da imagem.**

Uma imagem pode ser descrita por um conjunto de parâmetros, tais como:

- número de pixels;

- número de níveis de cinza;
- presença ou ausência de cor;
- se a imagem é multiespectral ou não é;
- se a mesma deve ou não ser tratada em tempo real.

### 3) Relação entre o tempo de processamento e o custo do sistema.

O tempo de processamento depende da aplicação da arquitetura. Em certos casos, deve-se fazê-lo em tempo real, o que exige maior velocidade. Isto implica num aumento do custo do "hardware". As verdadeiras necessidades de um sistema de processamento de imagens devem ser levadas em conta, de modo que seja obtida a melhor solução para um dado problema, com um custo razoável.

### 4) Os algoritmos existentes para a resolução do problema e as restrições impostas pelos mesmos.

Visando a solução de um dado problema, escolher-se-á o algoritmo mais adequado, dentre os vários algoritmos serials e paralelos disponíveis. A partir do mesmo, será feita a seleção da arquitetura mais apropriada à sua implementação.

### 5) Limitações de ordem técnica.

Pode-se optar entre sistemas cujo "hardware" é totalmente dedicado à resolução do problema e sistemas mais genéricos, onde o "software" é o principal responsável pela execução das tarefas. O número de processadores a ser usado e a forma pela qual estes serão interconectados são outros fatores importantes, que devem ser considerados.

### III.2 - TIPOS DE ARQUITETURA MAIS USADOS.

Foram desenvolvidos vários tipos de arquiteturas paralelas destinadas ao processamento de imagens. Estas máquinas podem ser classificadas em quatro grandes grupos a saber: arranjos de processadores, arquiteturas em "pipeline", sistemas MIMD e sistemas de unidades funcionais [11,12].

#### III.2.1 - Arranjo de Processadores.

Trata-se essencialmente de uma máquina SIMD, ou seja, possui uma unidade de controle que transmite instruções, endereços e sinais de controle para todos os elementos de processamento (EPs), conforme pode ser visto na figura III.1 [13]. A complexidade do controle é constante com o aumento do número de EPs

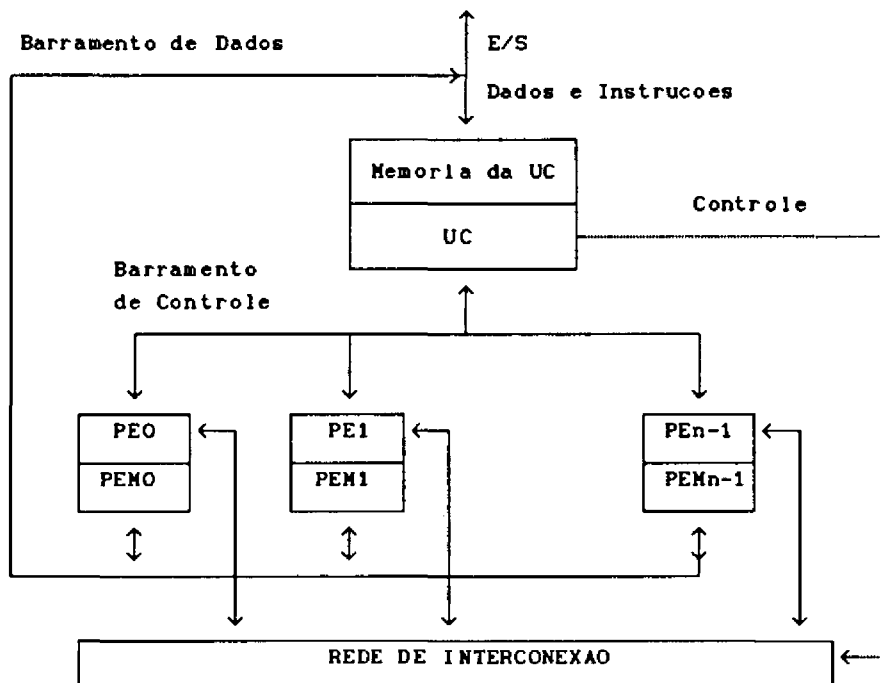


Figura III.1 - Arquitetura SIMD.

e permite um alto grau de paralelismo. Geralmente os EPs são do tipo serial por bit, dotados de uma unidade lógica e aritmética de um bit apenas e capazes de estabelecer comunicação com os seus vizinhos mais próximos.

As máquinas SIMD são mais adequadas à implementação de processamento de imagens a nível de pixel, ou seja, à execução de algoritmos que trabalham com a vizinhança local dos pixels. Neste tipo de arquitetura, a imagem é dividida em subimagens, que são mapeadas nos EPs, conforme é mostrado nas figuras III.2 e III.3.

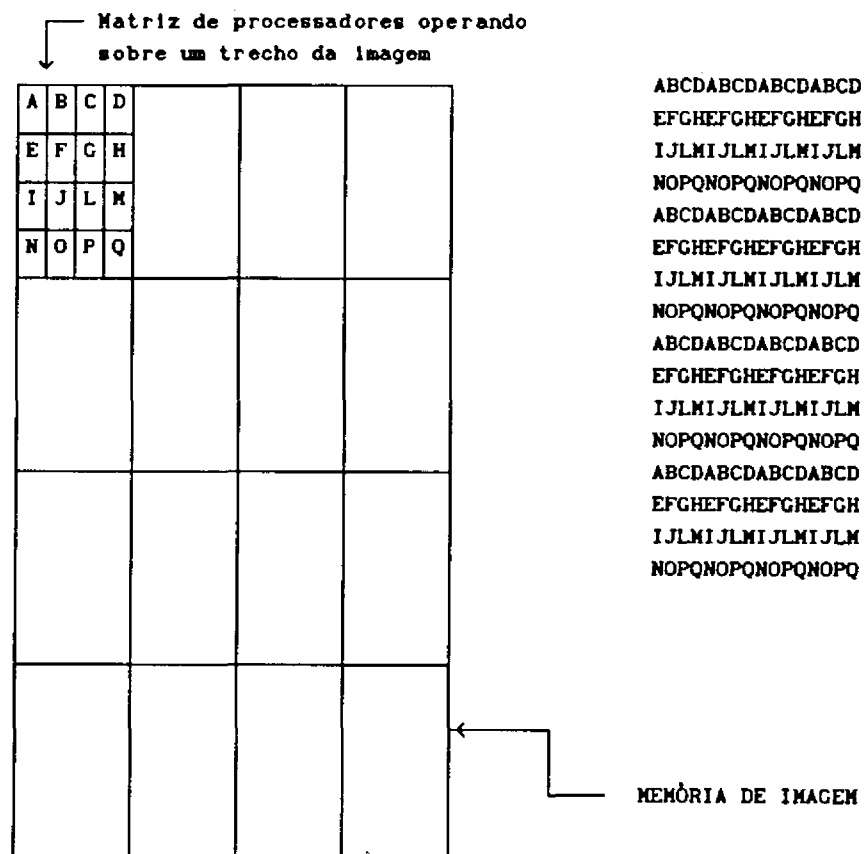


Figura III.2 - Distribuição de EPs sobre uma imagem, com mapeamento EP-pixel.

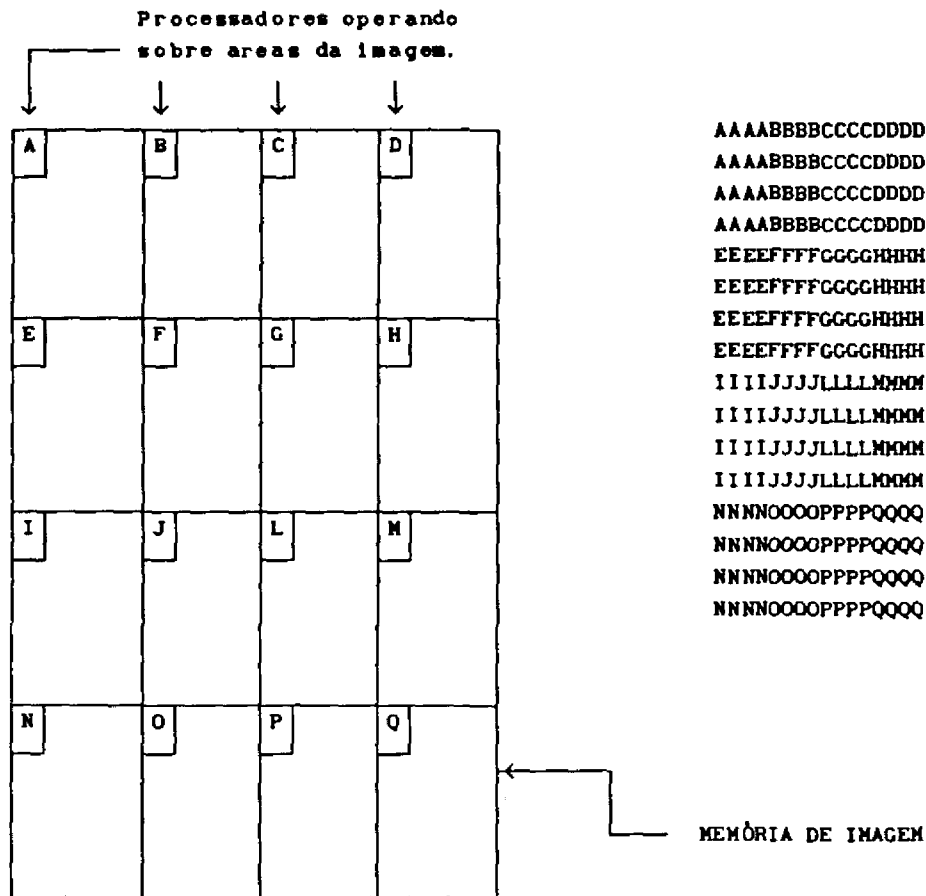


Figura III.3 - Distribuição de EPs sobre uma imagem, com mapeamento EP-área.

No limite, há um processador para cada pixel, ou seja, a matriz de processadores possui o mesmo tamanho da imagem. Desta maneira, o tempo de processamento dependerá apenas da complexidade dos cálculos efetuados e não do tamanho da imagem. Como exemplos de arquiteturas SIMD, podem ser citados: MPP, Illiac-IV, Clip-IV, BASE8 e GAPP.

### III.2.2 - Arquiteturas em "Pipeline".

Nestas máquinas, a imagem é armazenada em um "buffer", sendo a seguir lida linha a linha. A seqüência de dados gerada percorre um conjunto de estágios de modo que seja processada em "pipeline" (figura III.4). Cada estágio

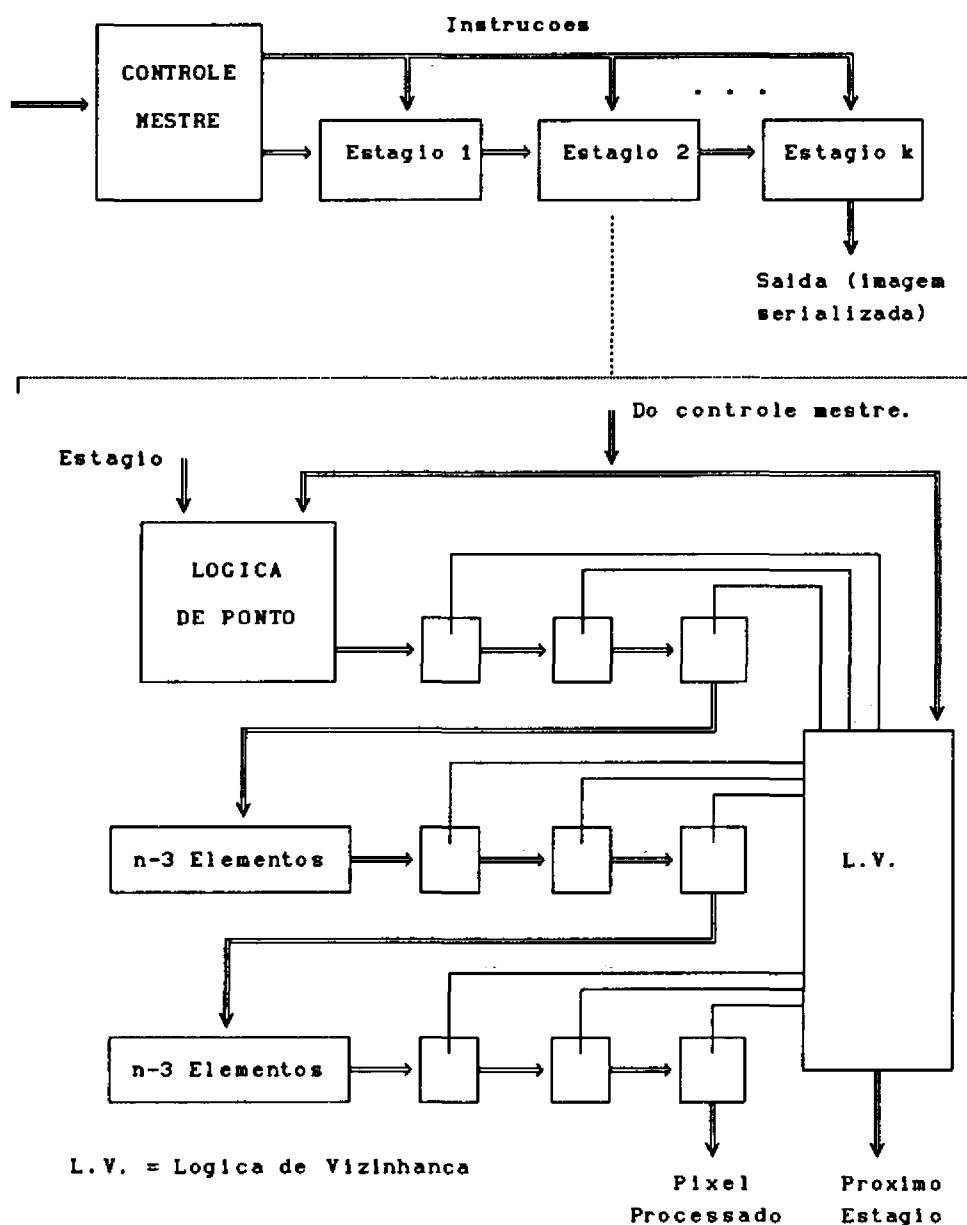


Figura III.4 - Diagrama em blocos do Cytocomputer.

produz um valor transformado de pixel por ciclo de relógio. Os vizinhos locais do pixel a ser processado são obtidos através de mecanismos de atraso (memória de atraso de linha).

Ignorando-se os atrasos iniciais (necessários para a organização da informação pertinente à vizinhança do primeiro pixel da imagem), pode-se dizer que o fluxo de informações é constante, ou seja, à medida que os dados entram, os dados processados saem. A desvantagem deste tipo de arquitetura reside no fato de que o tempo de processamento cresce linearmente com o número de pixels da imagem, ao invés de aumentar conforme cresce a complexidade dos cálculos. Como exemplos deste tipo de arquitetura podem ser citados o Cytocomputer, o FLIP e PICAP II.

### III.2.3 - Sistemas MIMD (multiprocessadores).

No modo MIMD, cada processador possui uma unidade de controle e executa o seu próprio programa. Torna-se necessária a existência de um elemento responsável pela coordenação das tarefas realizadas por todos os EPS, visando a obtenção do máximo desempenho possível (figura III.5). Assim como no modo SIMD, é importante que haja mecanismos de interconexão adequados ao estabelecimento da comunicação entre os EPs e os módulos de memórias. Esta classe de máquinas será estudada detalhadamente no próximo capítulo.

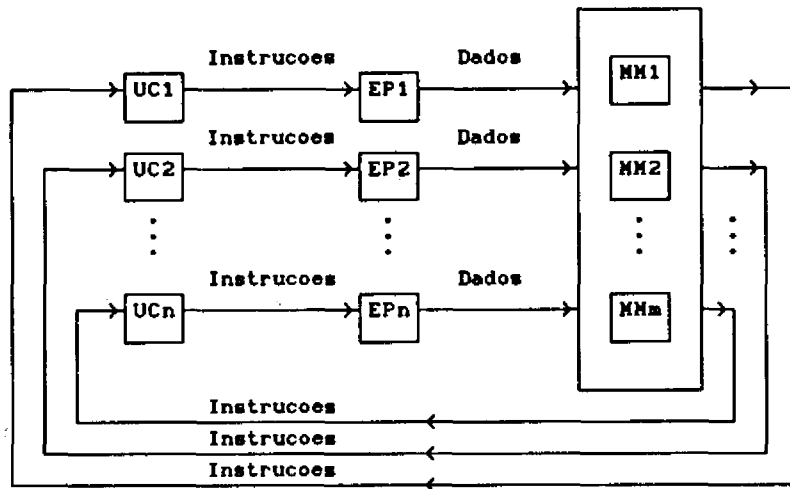


Figura III.5 - Organização de uma máquina MIMD.

As máquinas MIMD são pouco eficientes quando utilizadas em processamento de baixo nível. Isto se deve ao fato de que é necessário um "hardware" mais complexo para gerenciamento das unidades de controle individuais e para garantir a confiabilidade da comunicação entre EPs. Estes podem, ainda, trocar informações entre si referentes aos pixels da vizinhança local, o que resulta num outro problema: perda de tempo devido à comunicação.

Este tipo de arquitetura é mais adequado ao processamento de imagens a nível de região, como por exemplo, os algoritmos para identificação de contornos e classificação de objetos. Tarefas de reconhecimento de padrões podem ser realizadas em paralelo, usando-se para tal muitas cópias de uma imagem ou ainda uma ou mais imagens compartilhadas. Por exemplo, seja o caso em que se deseja localizar um ou mais objetos dentro de uma mesma imagem. Pode-se destinar um processador ou vários deles à busca de um dado objeto [14]. Atualmente há diversas máquinas operando neste modo e que tratam de



problemas de processamento de imagem tais como: PASM, ZMOB, PUMPS e C.m.m.p.

#### **III.2.4 - Sistemas com Unidades Funcionais.**

Estes sistemas são formados por um computador hospedeiro e um conjunto de unidades de processamento dedicadas a funções especiais. Cada unidade de processamento é responsável pela execução de apenas uma função (ou conjunto de funções afins), possuindo um "hardware" totalmente voltado para a aplicação a qual se destina. A idéia básica é reunir numa mesma máquina a velocidade de processamento das unidades funcionais e as características de propósito geral do computador hospedeiro (edição, compilação, etc).

As unidades funcionais podem ser implementadas, por exemplo, com os circuitos integrados VLSI usados no projeto de arquiteturas sistólicas, os quais são bastante adequados à realização de operações tais como filtragem, correlação e FFT. Outro exemplo, seria a implementação dos circuitos para a etapa de sensoriamento da imagem através de CCDs e que já incluíssem alguma forma de processamento de baixo nível. ESL, TOPICS e DSR são máquinas que se enquadram nesta categoria de arquiteturas.

Existem outros tipos de arquiteturas em estudo e que não são classificadas em nenhuma das categorias discutidas anteriormente. Dentre elas, podem ser citadas aquelas baseadas na concepção "data-flow". Estas arquiteturas não serão abordadas neste trabalho, embora haja autores [15] que as classifiquem como sendo máquinas MIMD.

## CAPÍTULO IV

### MÁQUINAS M.I.M.D.

Neste capítulo, serão vistos alguns aspectos referentes às máquinas M.I.M.D., pois este é o modo de operação dos computadores baseados em "transputers". Este modo de operação apresenta as seguintes vantagens [15,16]:

- descentralização do controle e dos recursos o que tende a produzir uma resposta mais rápida;
- maior disponibilidade e capacidade de compartilhamento de recursos;
- maior adaptabilidade a mudanças na carga de trabalho; e
- é mais geral e flexível.

Estas máquinas podem ser caracterizadas por três fatores: topologia, granularidade e a forma como as unidades são controladas e sincronizadas.

#### IV.1 - TOPOLOGIA.

Daqui em diante, será considerado apenas o processamento de imagens a nível de região, ou seja, cada região é tratada por um processador, havendo necessidade de trocar partições da imagem entre os processadores. Isto torna indispensável a existência de formas adequadas de interconexão de EPs (elementos de processamento) e bancos de memória. Um outro aspecto a ser levado em conta é o fato de que, em se tratando de algoritmos complexos voltados para visão por computador, deve-se procurar sistemas de

máquinas com ampla gama de aplicações.

#### **IV.1.1 - Acoplamento.**

As topologias existentes para o estabelecimento de conexões entre os EPs e a memória podem ser classificadas, quanto ao acoplamento em dois tipos básicos: forte e fraco.

##### **IV.1.1.1 - Acoplamento Forte.**

Num sistema fortemente acoplado, os EPs e a memória global compartilham um mesmo espaço de endereçamento. Os elementos do sistema comunicam-se através da memória comum. Este tipo de acoplamento tolera um alto grau de interações entre tarefas, sem que haja significativa deterioração do desempenho.

##### **IV.1.1.2 - Acoplamento Fraco.**

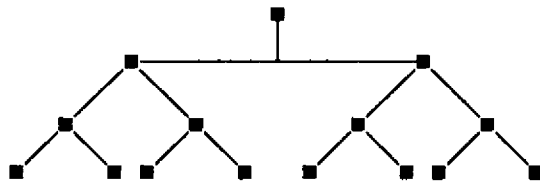
Neste tipo de arquitetura, os EPs comunicam-se através de troca de mensagens. Estes sistemas são muitas vezes chamados de distribuídos. As mensagens destinadas a EPs não-adjacentes são transmitidas através dos EPs intermediários, pois um processador se comunica mais facilmente com um vizinho. Logo, a topologia do sistema é um fator muito importante para que a comunicação seja eficiente. Em geral, sistemas com acoplamento fraco são mais eficientes quando há poucas interações.

Há vários padrões de interconexão de EPs. Na figura IV.1 são mostradas

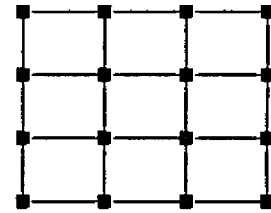
diversas topologias com acoplamento fraco. A escolha depende da aplicação. Uma estrutura em árvore presta-se ao tratamento de problemas de inteligência artificial, como por exemplo, classificação. A malha é a topologia mais empregada em PI, devido à facilidade de mapeamento da imagem na matriz de processadores. Porém, a estrutura mais flexível é o hipercubo, pois minimiza o número de processadores intermediários necessários à transmissão de mensagens. Um hipercubo de dimensão  $n$  possui  $N$  EPs interconectados, de modo que  $N = 2^n$ . Além disto, anéis, malhas e árvores podem ser mapeados no mesmo, resultando numa gama maior de aplicações (vide Apêndice I). A flexibilidade desta estrutura a torna bastante apropriada ao desenvolvimento de computadores de grande porte.

A confiabilidade de um sistema M.I.M.D. pode ser aumentada com a introdução de mecanismos de tolerância a falhas [13,17]. Caso circuitos especiais sejam acrescentados para teste dos resultados de processamento obtidos, as falhas podem ser detectadas durante o tempo de execução. A lógica de correção será acionada ou não, dependendo do resultado. Há dois tipos de soluções:

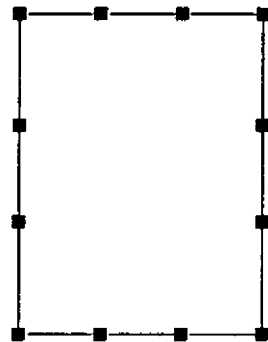
- (i) se a arquitetura possuir um pequeno número de EPs, a correção deverá ser feita por "software";
- (ii) caso a arquitetura apresente uma estrutura regular, pode-se estabelecer esquemas de reconfiguração, de modo que os EPs defeituosos venham a ser substituídos por EPs de reserva e as conexões possam ser refeitas.



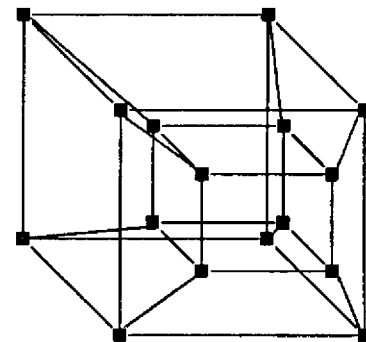
(a) Árvore



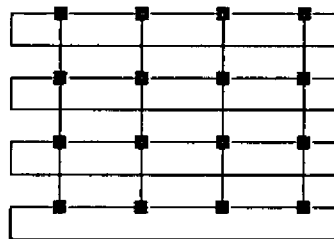
(b) Malha



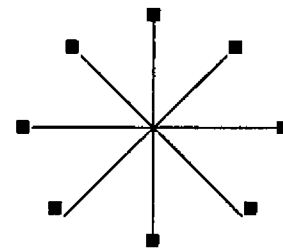
(c) Anel



(d) Hipercubo



(f) Torus



(g) Estrela

■ = Elemento de processamento  
 — = Conexão

Figura IV.1 - Exemplos de arquiteturas fracamente acopladas.

#### IV.1.2 - Tipos de Interconexão.

Um fator determinante para a organização de sistemas M.I.M.D. é a forma como os processadores, a memória e os canais de E/S são interligados. Há três diferentes tipos de interconexão [13]:

- barramento compartilhado;
- rede de barras cruzadas ("crossbar");
- memórias multiporta.

##### IV.1.2.1 - Barramento Compartilhado.

Os EPs são conectados à memória compartilhada por meio de um barramento de alta velocidade (fig. IV.2).

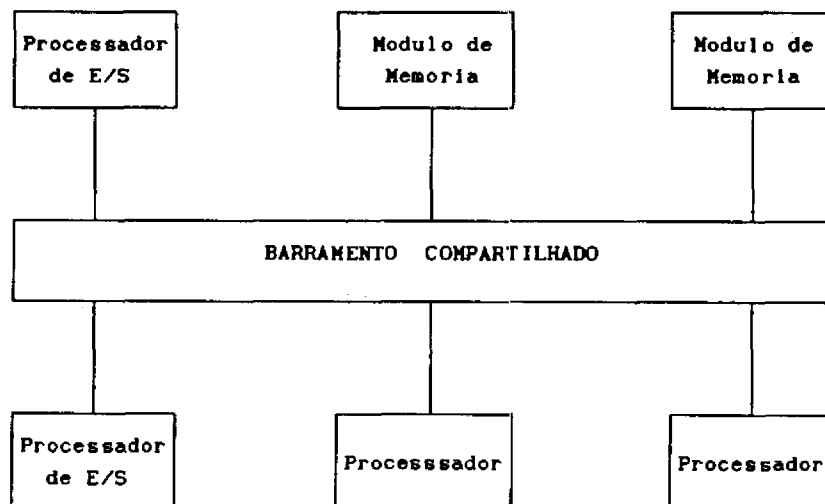


Figura IV.2 - Organização típica de um barramento.

Este tipo de interconexão apresenta um custo total baixo, em relação às demais, sendo mais apropriada para pequenas máquinas. A configuração de um sistema pode ser facilmente modificada através da adição ou da remoção de unidades funcionais, porém deve-se levar em consideração que o desempenho é degradado quando novas unidades são adicionadas. A capacidade total do sistema é limitada pela taxa de transferência do barramento e caso ocorra uma falha no mesmo, esta será catastrófica. A eficiência do sistema é pequena, quando comparada com aquela apresentada pelos outros tipos de interconexão.

#### IV.1.2.2 - Rede de Barras Cruzadas.

Os sistemas nos quais as ligações são feitas por intermédio de chaves situam-se na fronteira entre o acoplamento fraco e o forte. Os mesmos empregam um mecanismo de endereçamento que requer informações adicionais para o estabelecimento da rota correta. Na figura IV.3 é mostrada uma rede de barras cruzadas. O custo destes sistemas cresce linearmente com o número de chaves, o que é um fator limitante.

Uma solução para este problema é a utilização de uma chave do tipo borboleta (figura IV.4), a qual reduz o número de chaves necessárias à conexão de  $N$  nós para  $N \cdot \log_4 N$ . O termo nó será usado para designar o conjunto formado por um processador e a sua memória local.

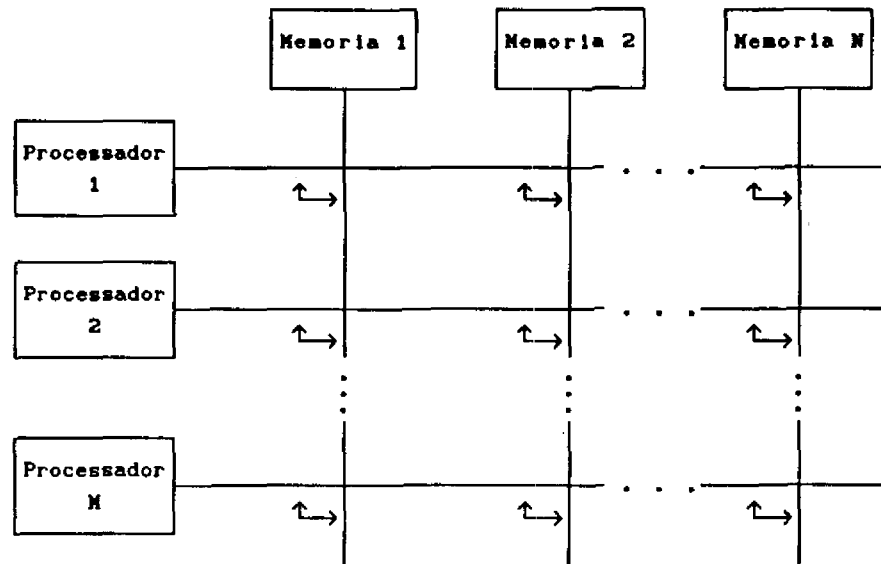


Figura IV.3 - Rede de barras cruzadas.

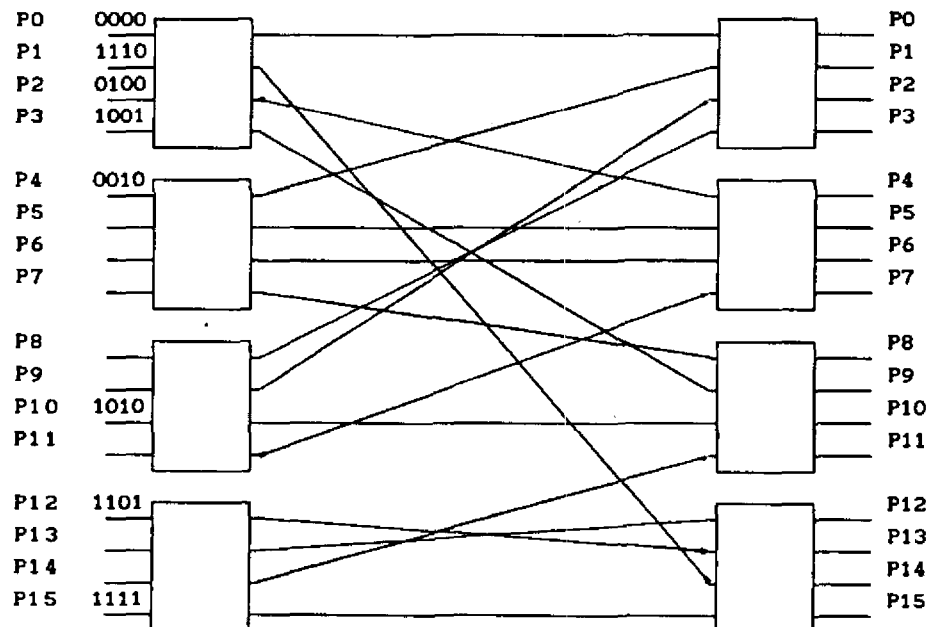


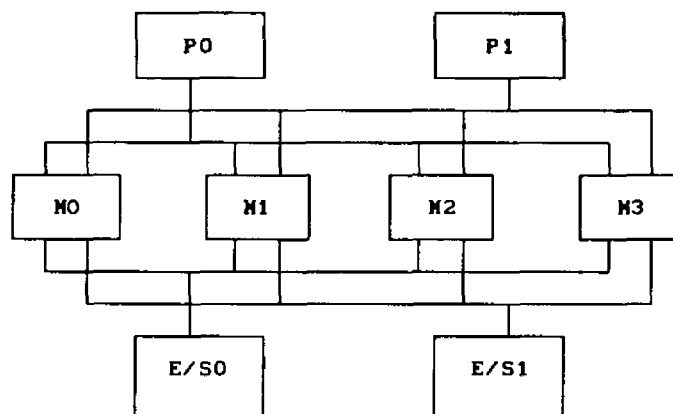
Figura IV.4 - Rede borboleta.



Este é o sistema de interconexão mais complexo, mas permite que se obtenha uma alta taxa de transferência. As unidades funcionais são simples e baratas, uma vez que o controle e a lógica de chaveamento estão na chave. Em geral, a expansão do sistema aumenta o seu desempenho total. Não é necessário reprogramar o sistema operacional quando a máquina é expandida. Teoricamente, a expansão do sistema é limitada apenas pelo tamanho da matriz de chaves, a qual pode ser aumentada de forma modular, respeitando um projeto inicial ou qualquer outra limitação técnica. A confiabilidade do sistema pode ser aumentada através de segmentação e/ou redundância dentro da chave.

#### IV.1.2.3 - Memória Multiporta.

Neste caso, o controle, o chaveamento e a lógica de arbitração de prioridade acham-se distribuídos pelas interfaces para os módulos de memória (figura IV.5).



$P_i$  = processador  $i$ ,  $i=0,1$   
 $M_j$  = unidade de memória,  $j=0,\dots,3$   
 $E/S_n$  = módulo de entrada/saída,  $n=0,1$

Figura IV.5 - Organização de um sistema de memória multiporta.

Este tipo de sistema requer unidades de memória mais caras, pois a maior parte do controle e da lógica de chaveamento reside na unidade de memória, porém isto permite que processadores de baixo custo sejam usados e é possível obter uma grande taxa de transferência.

As opções de tamanho e configuração são limitadas pela quantidade e pelo tipo de portas de memória disponíveis; estas escolhas devem ser feitas no início do projeto, pois é difícil fazer modificações. Uma outra desvantagem deste tipo de interconexão é a quantidade de cabos e conectores necessários.

#### **IV.1.3 - Características de um Processador Destinado à Operação em Máquinas M.I.M.D.**

Muitas máquinas M.I.M.D., como por exemplo o C.m.m.p., foram construídas com processadores cujos projetos não eram adequados a este fim. A seguir, são mencionadas as características desejáveis para que um processador apresente bom rendimento em multiprocessadores [13,17].

##### **IV 1.3.1 - Capacidade de Recuperação de Processos.**

A arquitetura de um processador deve refletir o fato de que processo e processador são entidades diferentes. Se o processador falhar, deve ser substituído por outro, de modo que a execução do processo não seja interrompida. Sem esta característica, a capacidade de recuperação do sistema é substancialmente reduzida. Muitos processadores armazenam o estado

do processo corrente em seus registradores, mas estes não são acessíveis externamente nem são escritos na memória no momento de uma falta. Com a tecnologia atual, é possível separar os registradores de propósito geral daqueles usados para o controle do contexto, sem perda significativa de velocidade. É desejável que se tenha uma cópia (arquivo) dos registradores compartilhada por todos os processadores no momento de falha do sistema.

#### IV.1.3.2 - Chaveamento de Contexto Eficiente.

Um outro motivo para a existência de um arquivo de registradores é a possibilidade de usá-lo para chaveamento de contexto (figura IV.6), que é

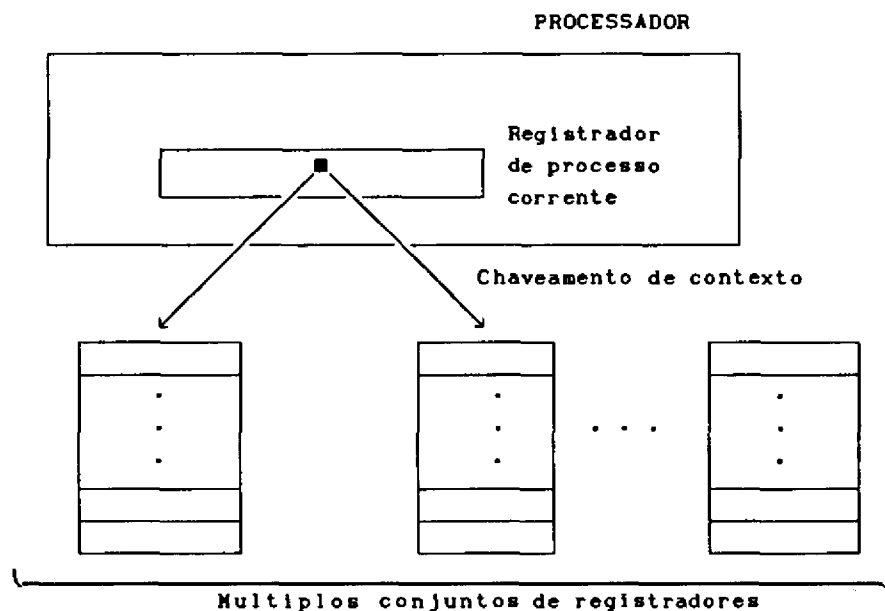


Figura IV.6 - Chaveamento de contexto num processador com vários conjuntos de registradores.

uma operação de salvamento do estado do processo corrente e substituição do mesmo pelo estado do processo que está pronto para ser executado. O estado de um processo em execução é indicado pelos conteúdos dos registradores do processador. Esta operação exige o uso intensivo de pilhas e filas, sendo importante que haja um conjunto de instruções que suporte este tipo de mecanismo.

#### **IV.1.3.3 - Espaços de Endereçamento Virtual e Físico Grandes.**

Um processador destinado ao uso em multiprocessadores de médio e grande porte deve possuir um amplo espaço de endereçamento físico. Mesmo quando um algoritmo é implementado com um código pequeno, podem existir processos que acessem grandes quantidades de dados. Um espaço virtual grande, e se possível segmentado, também é desejável.

#### **IV.1.3.4 - Primitivas de Sincronização Eficientes.**

O processador deve fornecer alguma implementação de ações indivisíveis, as quais servirão como base para primitivas de sincronização. Estas primitivas requerem mecanismos de mútua exclusão eficientes, a qual é necessária quando dois ou mais processos estão sendo executados concorrentemente e devem trocar informações entre si durante a execução de um programa. Um exemplo deste mecanismo é o semáforo. Algumas instruções usadas para este fim são: "test-and-set" e "compare-and-swap".

#### **IV.1.3.5 - Mecanismo de Comunicação entre Processadores.**

O conjunto de processadores deve comunicar-se de forma eficiente, preferencialmente por "hardware". Esta necessidade é mais visível em sistemas onde há troca frequente de serviços entre processadores diferentes. O mecanismo de comunicação também pode facilitar a sincronização e correção de falhas.

No caso de sistemas fortemente acoplados, é possível implementar a comunicação via "software". Este método é ineficiente, pois cada processador deverá atualizar periodicamente a sua "caixa postal" para averiguar a existência de uma mensagem endereçada a ele. Isto aumenta os tempos de resposta, além de exigir mecanismos de prevenção de acesso simultâneo aos canais de comunicação.

#### **IV.1.3.6 - Conjunto de Instruções.**

O conjunto de instruções de um processador deve possuir características adequadas à implementação de linguagens concorrentes de alto nível. Devem existir instruções para ligação de procedimentos, construções que permitam a criação de laços, capacidade de manipulação de parâmetros, cálculo de índices multidimensionais e verificação dos limites dos endereços. Além do mais, devem existir instruções que criem e terminem processos paralelos dentro de um programa. Também é desejável que existam vários modos de endereçamento. Devem ser fornecidos contadores e relógios de tempo real para que sejam gerados os números de identificação dos processos e os sinais de tempo de espera ("time-out") necessários ao gerenciamento dos processos.

Estes temporizadores também podem ser utilizados para detectar erros, caso os mesmos sejam associados a recursos importantes. Um multiprocessador fornece um ambiente natural onde cada um dos componentes podem monitorar os outros de forma relativamente fácil.

#### **IV.1.4 - Organizações de Memória.**

Numa máquina M.I.M.D., a degradação do desempenho provocada por acessos simultâneos à memória deve ser minimizada. Em sistemas fortemente acoplados, é desejável que a memória seja dividida em vários módulos independentes. Esta técnica é conhecida como entrelaçamento e apresenta como principal vantagem o fato de permitir acessos concorrentes a mais de um módulo de memória.

Há dois tipos básicos de entrelaçamento:

##### **(i) Alta ordem.**

Os módulos contêm endereços consecutivos, proporcionando maior confiabilidade ao sistema. Se houver falha, apenas uma parte do espaço de endereçamento será afetada e a degradação do desempenho será mais lenta. O módulo defeituoso poderá ser isolado, de modo que nenhum processo use a área afetada. Este método é muito sujeito a conflitos de barramento, pois os processos compartilham instruções e dados, além de ocorrerem interações entre eles.

##### **(ii) Baixa ordem.**

Neste caso, endereços consecutivos estão em módulos consecutivos. Este

método permite a busca simultânea de elementos, o que reduz sensivelmente a interferência na memória.

Mesmo quando o entrelaçamento de memória é usado, ocorrem atrasos na transmissão e conflitos na memória. Com o objetivo de minimizar estes problemas, pode-se associar uma memória cache a cada EP. Assim procedendo, a maior parte das instruções e dados referenciadas por um EP pode ser encontrada na cache. Contudo, a largura de banda do barramento pode afetar o custo e o tempo de transferência de um bloco de dados para a cache.

#### IV.2 - CONTROLE E SINCRONIZAÇÃO.

Uma questão relevante, em se tratando do projeto de uma máquina M.I.M.D., é o controle e a sincronização dos processos executados pelos EPs. Uma distribuição equilibrada de tarefas contribui para o aumento do desempenho do sistema, diminuindo, inclusive, o tempo gasto com a comunicação entre processos. O controle e a sincronização são geralmente feitos via "software", o qual contém primitivas destinadas à coordenação de tarefas, além de fornecer mecanismos para a proteção de recursos, tais como semáforos. A complexidade do controle tende a crescer muito rápido, à medida que o número de processadores aumenta.

##### IV.2.1 - Granularidade.

Granularidade é uma medida quantitativa do potencial de processamento de cada EP. Se a capacidade de um EP é pequena, diz-se que o

atrasos relacionados com a comunicação. A implementação deste método é mais complexa, pois o escalonador de processos deve examinar o instante em que foi iniciada a execução de cada tarefa, em cada processador disponível, de modo a escolher o mais adequado para a execução de um determinado processo. O resultado obtido é mais eficiente.

#### **IV.2.1.2 - Escalonamento por Fluxo de Dados entre Processos de Grão Grosso.**

Esta técnica procura considerar os atrasos introduzidos pela comunicação entre processos que estão sendo executados em processadores diferentes. Um determinado programa é dividido em processos, de tal forma que o tempo de execução de uma tarefa ou grupo de tarefas é bem maior do que o tempo gasto para a comunicação entre elas. O maior problema neste caso, é a determinação do tamanho do grão e o fato da otimização ser realizada apenas através da análise da comunicação.

### **IV.3 - PROGRAMAÇÃO DE MÁQUINAS PARALELAS.**

Nesta seção, serão discutidos alguns aspectos relativos ao "software" dos multiprocessadores.

#### **IV.3.1 - Linguagens Paralelas.**

Um programa paralelo, desenvolvido para uma máquina M.I.M.D., consiste de dois ou mais processos interativos. O primeiro passo no desenvolvimento de



mesmo apresenta paralelismo de grão fino, caso contrário, tem-se paralelismo de grão grosso. Um grão é definido como sendo um módulo de programa que pode ser executado concorrentemente. O tempo de execução de um programa num sistema M.I.M.D. pode ser sensivelmente reduzido, dependendo da forma como as diversas tarefas que o compõem são distribuídas pelos EPs [18].

A questão da granularidade envolve dois aspectos:

- escalonamento por lista; e
- escalonamento por fluxo de dados entre processos de grão grosso.

#### IV.2.1.1 - Escalonamento por Lista.

É feita a atribuição de uma tarefa, cujas tarefas predecessoras tenham sido completadas, ao próximo processador disponível. Caso um processador esteja livre, é porque não há uma tarefa em condições de ser atribuída ao mesmo [18]. Neste tipo de escalonamento de processos, o tamanho do grão pode ser obtido por balanceamento de carga ou através de análise de máximos e mínimos.

Na primeira situação, é feita uma distribuição eqüitativa da carga por todos os EPs , de modo a minimizar o tempo de execução das tarefas. A principal desvantagem apresentada por esta estratégia é o fato de não levar em conta os atrasos provocados pela comunicação. Portanto, nem sempre é conseguido um bom desempenho do sistema.

A segunda estratégia procura maximizar o paralelismo, minimizando os

um programa paralelo é a identificação dos processos e dos objetos por eles compartilhados. Há linguagens onde o programador especifica explicitamente a concorrência, através de construções das mesmas. Em outras, o próprio compilador determina o que será executado em paralelo (concorrência implícita).

#### **IV.3.1.1 - Classes de Linguagens para Programação Concorrente.**

As linguagens utilizadas para programar máquinas M.I.M.D. enquadram-se numa das seguintes categorias [19,20]:

- a) linguagens sequenciais ampliadas com rotinas escritas em "Assembly" para tratamento do "hardware";
- b) linguagens concorrentes baseadas em variáveis compartilhadas;
- c) linguagens concorrentes baseadas em troca de mensagens explícita ou implícita.

#### **a) Classificação de linguagens quanto à comunicação.**

Um aspecto de grande importância é o modo como é feita a comunicação entre os processos. Há duas formas básicas: através de chamadas de procedimentos e através de troca de mensagens. Estas formas de comunicação podem ser ainda síncronas ou assíncronas.

A troca de mensagens requer o uso de primitivas do tipo "envia" e "recebe" num ambiente onde os objetos do sistema são colocados em processos.

Na troca de mensagens assíncrona [21], a execução das primitivas acontecerá de maneira não bloqueada, ou seja, os processos que contiverem as primitivas terão continuidade, mesmo que não estejam prontos para a comunicação. Na troca de mensagens síncrona [22], a execução das primitivas dependerá da situação dos processos emissores e receptores. Estes serão bloqueados quando os processos parceiros não estiverem prontos para a comunicação.

Na chamada de procedimento assíncrona, onde situa-se o mecanismo conhecido por monitor, os procedimentos fazem parte de uma estrutura passiva. Quando uma instrução de um processo for executada, o processo será suspenso e a chamada de procedimento será encaminhada, mesmo que o procedimento não possa ser executado imediatamente. A chamada final ficará pendente numa fila e a sua execução será realizada tão logo seja possível. Na chamada de procedimento síncrona, onde se situa a chamada remota de procedimento [23], os procedimentos estão associados a processos que gerenciam as suas chamadas. É necessário que haja uma sincronização entre processo chamador do procedimento, no ponto da chamada, e o processo gerenciador dos procedimentos, no ponto de recepção da chamada, para que a execução do procedimento seja iniciada. O princípio básico de uma chamada remota de procedimento é o fato de uma chamada de procedimento com seus parâmetros ser interceptada, e, em vez de ser executada no mesmo EP, ser transmitida para execução em outro EP. Enquanto isso, o programa que tiver chamado o procedimento permanecerá bloqueado. Quando o procedimento terminar no elemento remoto, o retorno com seus parâmetros será interceptado e remetido para o EP que originou a chamada. Aí, então, o programa que fez a chamada será desbloqueado, e continuará como se a chamada tivesse sido executada localmente.

**b) Classificação de linguagens quanto aos mecanismos de interação entre processos.**

As linguagens podem ser orientadas a procedimentos, a troca de mensagens e a operações [24].

As linguagens orientadas a procedimentos são aquelas que incorporam chamada assíncrona de procedimento e tratam processos e objetos passivos compartilhados (módulos, monitores, etc). A interação entre processos baseia-se em variáveis compartilhadas que representam os objetos passivos. A manipulação dos objetos pelos processos dá-se através de chamada de procedimento. Como exemplos desta classe, podem ser citados as seguintes linguagens: Pascal Concorrente, Módulo e Mesa.

Nas linguagens orientadas a mensagens, a interação entre processos que constituem os elementos do sistema baseia-se na troca de mensagens entre eles. As linguagens OCCAM, CSP e PLITS são exemplos desta classe.

As linguagens orientadas a operação combinam alguns aspectos das duas classes anteriores. Cada objeto está associado a um processo gerente, e as operações são realizadas sobre o objeto através da execução de um procedimento. A interação entre processos ocorre pela troca de mensagens, precedendo e sucedendo a execução de um procedimento acessado pelo respectivo processo gerente. As linguagens ADA e DP são exemplos desta classe.

#### **IV.3.1.2 - Características de Linguagens para Programação de Sistemas M.I.M.D.**

A utilização dessas linguagens está relacionada com a arquitetura do sistema. As linguagens orientadas a procedimento são mais adequadas a sistemas com memória compartilhada. Já as orientadas a mensagens são mais apropriadas para sistemas distribuídos, embora apresentem boa adequação para sistemas com memória compartilhada. As linguagens orientadas a operação incorporam as vantagens das duas classes anteriores; elas poderão ser orientadas a procedimento, quando existir memória compartilhada, ou poderão usar troca de mensagens, no caso de sistemas distribuídos. De qualquer forma, as três classes de linguagem podem ser adaptadas aos vários tipos de ambiente, apresentando ou não uma aderência natural aos mesmos e implicando sobrecargas diferentes de execução. Uma linguagem para programação de máquinas M.I.M.D. pode possuir algumas ou todas as características abaixo [20]:

##### **a) Método de comunicação.**

Corresponde à maneira pela qual os processos trocam informações. Os métodos normalmente utilizados são trocas de mensagens, variáveis globais, dados compartilhados protegidos por monitores e chamada remota de procedimento.

##### **b) Método de sincronização.**

Refere-se ao controle dos processos. Pode-se citar aqueles métodos que utilizam sinais, troca de mensagens síncrona, "buffers", eventos, filas de condição, regiões com guarda, "rendez-vous" estendido e chamada remota de procedimento.

**c) Uso de variáveis globais.**

É a forma mais antiga de comunicação de dados e, possivelmente, de sincronização entre processos.

**d) Uso de "buffers" com comprimento definido pelo usuário.**

Os "buffers" de mensagem, podem existir como um tipo de dado predefinido, podendo ser criados como variáveis de programa com comprimento especificado pelo usuário.

**e) Criação de processos.**

Os processos são criados estaticamente no início da execução do programa ou dinamicamente por ocasião de sua chamada durante a execução. Os processos estáticos continuam a existir mesmo que não sejam mais chamados; os dinâmicos desaparecem depois de terminados.

**f) Rede de interconexão de processos.**

Na interconexão estática, as conexões entre os processos não mudam durante a execução, ou, se houver criação e destruição dinâmicas de processos, a colocação e retirada das ligações estará prevista em tempo de compilação. Na interconexão dinâmica, as ligações entre processos podem ser alteradas durante a execução.

**g) Execução indeterminada/expressão explícita para a indeterminação.**

Indica a capacidade de uma linguagem expressar possíveis execuções indeterminadas, isto é, para várias execuções sobre os mesmos dados de entrada, pode-se obter conjuntos diferentes de dados de saída. Algumas linguagens possuem declarações especiais para expressar explicitamente execução indeterminada, enquanto em outras a indeterminação acontece através

algum escalonador de processos ou de mecanismo de fila, os quais são inacessíveis ao programador.

**h) Compilação em separado.**

Corresponde à possibilidade de compilar um módulo de código, mesmo sem o conhecimento do código do restante do programa.

**i) Suporte para operações em tempo real.**

Significa que a linguagem permite que o programador controle explicitamente a ordem de execução dos processos, manipule saídas por tempo na fase de comunicação entre processos, comunique-se diretamente com dispositivos de "hardware", etc. Tais características tornam a programação de aplicações em tempo real mais fácil e confiável.

**j) Facilidade de abstração.**

Significa que a linguagem pode manipular itens como monitores, pacotes, tipos abstratos de dados, etc.

**l) Tratamento de exceções.**

Corresponde à existência de facilidades para ações definidas pelo programador, a serem tomadas quando ocorrerem condições excepcionais durante o tempo de execução.

**m) Suporte para verificação de programas.**

A linguagem pode possuir características que ajudem os programadores ou verificadores automáticos a determinarem a correção lógica do programa. Essas características podem estar relacionadas com construções sintáticas, incluindo especificação formal de processos e suportes semânticos como, por

exemplo, convenções de chamada de processos e de monitores.

#### **IV.3.2 - Sistemas Operacionais.**

As características necessárias para um sistema operacional de um microprocessador são:

- alocação e gerenciamento de recursos;
- proteção de memória e banco de dados;
- prevenção de "deadlock" e terminação anormal de processos;
- manipulação de exceções;
- gerenciamento de E/S;
- esquemas de balanceamento de carga;
- utilização eficiente de recursos.

Uma das razões para o uso de máquinas com vários processadores é a confiabilidade e a redução da degradação de desempenho no caso de falhas, implicando numa carga adicional para o sistema operacional. O número de processadores e a forma como eles estão interconectados também influenciam o projeto de um sistema operacional.

##### **IV.3.2.1 - Classificação dos Sistemas Operacionais para Multiprocessadores.**

Há basicamente três tipos de sistemas operacionais: mestre-escravo, supervisor separado e supervisor flutuante. Na prática, a maior parte dos sistemas operacionais existentes não são exemplos puros destas três classes,



porém este esquema de classificação é bastante adequado para fins de estudo.

**a) Modo mestre-escravo.**

O modo mestre-escravo é o mais simples de ser implementado. Nesta configuração, um processador chamado mestre mantém o estado e coordena todos os outros processadores do sistema (escravos). O supervisor é sempre executado no mesmo processador. Um exemplo deste tipo de sistema operacional é o Cyber-170. Além disto, este tipo de sistema pode ser obtido através de extensões de sistemas operacionais de máquinas uniprocessadas que apresentem multiprogramação. Embora o tipo mestre-escravo seja simples, este é ineficiente para o controle e utilização dos recursos do sistema. É eficiente apenas em aplicações especiais, onde a carga de trabalho é bem definida ou em sistemas assimétricos, onde os escravos possuem uma capacidade menor do que o processador mestre. Este modo é inferior aos outros dois e é frequentemente usado quando existem poucos processadores envolvidos.

**b) Modo supervisor separado.**

Nesta configuração, há um sistema supervisor separado (núcleo) sendo executado em cada processador. Este modo se assemelha aos sistemas operacionais usados em redes de computadores, onde cada processador contém uma cópia do núcleo. O compartilhamento de recursos ocorre num nível mais alto, como por exemplo, através de uma estrutura de arquivos compartilhados. Cada processador atende às suas próprias necessidades. Como ocorrerão interações entre os processadores, haverá necessidade de que parte do código supervisor seja replicado ou reentrante, de modo que existam cópias

disponíveis em todos eles. Embora cada supervisor tenha o seu próprio conjunto de tabelas, algumas delas são de uso comum e compartilhadas por todo o sistema. Esta característica cria vários problemas de acesso a tabelas. O método empregado para o acesso dos recursos compartilhados depende do grau de acoplamento entre os processadores. Este modo não é tão sensível a falhas catastróficas quanto o anterior. Cada processador possui o seu próprio conjunto de arquivos e dispositivos de E/S. Qualquer reconfiguração de E/S requer intervenção manual e, em alguns casos, chaveamento manual.

**c) Modo supervisor flutuante.**

Este esquema trata todos os processadores da mesma forma que os outros recursos. Isto se dá de maneira simétrica ou como um grupo ("pool") de recursos. Este modo é o mais difícil de ser implementado e o mais flexível. Neste caso, a rotina supervisora flutua de um processador para outro, embora outros processadores possam estar executando rotinas supervisoras simultaneamente. O balanceamento de carga oferecido é melhor do que aquele apresentado pelos outros métodos. Conflitos na requisições de serviços são resolvidos através do estabelecimento de prioridades estabelecidas sob bases estatísticas ou sob controle dinâmico. Como há um considerável grau de compartilhamento de recursos, a maior parte do código do supervisor deve ser reentrante. Neste tipo de sistema, os conflitos de acesso a tabelas e os atrasos provocados pela busca nas mesmas não podem ser evitados. É importante controlar estes acessos para resguardar a integridade do sistema. Este modo apresenta as seguintes vantagens: menor degradação do desempenho, aumento da disponibilidade de recursos e redundância real. Os sistemas operacionais Hydra do C.m.m.p. e MVS do IBM 3081 são exemplos deste modo.

## CAPÍTULO V

### O "TRANSPUTER" E A LINGUAGEM OCCAM

Os "transputers" são circuitos integrados VLSI, produzidos pela INMOS, que combinam em uma única pastilha unidades de processamento, comunicação e memória.

#### V.1 - O "TRANSPUTER".

##### V.1.1 - Concepção.

O "transputer" foi concebido como um bloco destinado à construção de sistemas multicomputadores, sendo, a grosso modo, um microcomputador dotado de memória local e "links" seriais para comunicação [25-30]. Pode-se construir um nó de processamento com este componente, acrescentando-se uns poucos circuitos de suporte. O uso do mesmo em conjunto com as regras formais da linguagem OCCAM, fornece toda uma metodologia de projeto para sistemas concorrentes, baseada nos trabalhos de HOARE [22].

Em realidade, o termo "transputer" designa uma família de dispositivos VLSI programáveis, dentre os quais podem ser citados: controladores de disco, processssadores gráficos, dispositivos para processamento de sinais e microcomputadores de 16/32 bits.

O IMS T414 possui um processador de 32 bits, 2K bytes de memória interna e uma interface de 32 bits para memória externa, além de 4 links

para conexão com outros "transputers". O IMS T212 difere do IMS T414 apenas pelo comprimento de palavra usado para a interface para memória externa, que é 16 bits. O IMS M212 é um controlador de disco dotado de um processador de 16 bits, RAM, ROM, dois links seriais e circuitos especiais para controle de um disco rígido e discos flexíveis. Há também produtos destinados ao interfaciamento com componentes que não seguem o padrão INMOS.

O "transputer" viabiliza a implementação de um programa dividido em processos. Um processo é um programa independente, com seu próprio espaço de programa e dados, o qual pode comunicar-se com outros processos, executados concorrentemente. A comunicação entre os processadores é feita através de troca de mensagens, usando a definição de canais. Instruções especiais dividem o tempo do processador e realizam a comunicação entre processos. Existe uma correspondência entre os canais do programa e os "links" do "transputer". Cada canal fornece uma conexão entre dois processos concorrentes. A comunicação é feita por troca de mensagens e sincronizada, implica, obrigatoriamente, na existência de um canal de entrada e outro de saída.

O comportamento externo de um "transputer" corresponde ao modelo formal de um processo. Como consequência, é possível programar sistemas contendo vários "transputers" interconectados, atribuindo um conjunto de processos a cada um deles. Uma vez que um programa é definido como sendo um conjunto de processos, o mesmo pode ser mapeado numa máquina M.I.M.D. de várias formas: minimizando-se o custo, otimizando-se o desempenho, melhorando-se a resposta a determinados eventos, etc.

A programação destes componentes pode ser feita em muitas linguagens,

mas as suas características especiais são melhor exploradas pela OCCAM [29-32], que é tão eficiente quanto o "Assembly" e simplifica a elaboração de programas. Esta linguagem fornece facilidades para a exploração da concorrência através do encapsulamento das noções de entrada, saída e interrupções dentro de um formalismo. Programas desenvolvidos utilizando a mesma podem ser ligados a outros módulos, escritos em outras linguagens de alto nível.

A arquitetura do "transputer" é independente do tamanho da palavra, ou seja, processadores com comprimentos de palavra diferentes podem ser interconectados livremente e programados como sendo um único sistema [26]. O programa em OCCAM estabelece uma correspondência entre os aspectos lógicos e os físicos. O comportamento lógico do programa não será alterado pelo modo como os processos são mapeados ou pela velocidade de processamento e comunicação. Um mesmo programa escrito em OCCAM pode ser executado tanto por um ou mais "transputers". Conseqüentemente, um programa escrito para uma rede de "transputers" pode ser compilado e executado por um componente apenas, usado para fins de desenvolvimento.

Em aplicações que requeiram somente um elemento de processamento, o "transputer" tem a oferecer várias vantagens em relação aos processadores convencionais. Dentre elas, pode-se citar:

- velocidade;
- redução do número de componentes presentes no sistema; e
- facilidade de programação.

O desempenho de um sistema pode ser aumentado colocando-se um

"transputer" (ou uma rede deles) dedicado a uma função particular, dentro de um dado programa. Em geral, este é o melhor método para assegurar que a resposta em tempo real será satisfatória.

Um outro aspecto que pode ser considerado, quando do estudo da arquitetura do "transputer", é o da relação entre o desempenho e a área de silício utilizada. Um conjunto de 4 pastilhas T800-30 ocupam praticamente a mesma área que um i80386 com um co-processador Weitek [27] numa placa de circuito impresso. Porém, o desempenho do primeiro sistema em operações de ponto flutuante (aplicações concorrentes) é cerca de 6 vezes o do segundo.

#### V.1.2 - Arquitetura do T800.

Os algoritmos desenvolvidos neste trabalho serão executados em placas com 1 e 4 "transputers" e no multiprocessador hipercúbico NCPI (Apêndice I). Estes três sistemas têm em comum o fato de serem baseados no componente T800, o qual será descrito a seguir.

O T800 é um microcomputador de 32 bits com uma unidade de ponto flutuante de 64 bits (figura V.1, tabela V.1) [27]. Existem dois modelos: o T800-20 (20 MHz) e o T800-30 (30 MHz). O conjunto de instruções permite uma implementação eficiente das linguagens de alto nível, principalmente OCCAM, em virtude do modelo de concorrência usado por esta linguagem.

Tabela V.1 - Descrições de pinos [27].

Pino	E/S	Função
VCC, GND	-	Alimentação e terra.
CapPlus, CapMinus	-	Capacitor externo.
ClockIn	E	Clock
ProcSpeedSelect0-2	E	Seletores de frequência
Reset	E	Reset do sistema
Error	S	Indicador de erro
ErrorIn	E	Entrada de erro para "daisy chain"
Analyse	E	Análise de erros
BootFromRom	E	"Boot" pela ROM ou link
HoldToGND	-	Deve ser conectado à terra
DoNotWire	-	Não deve ser conectado
ProcClockOut	S	Clock do processador
MemnotWrD0	E/S	Bit 0 de dado e aviso de escrita
MemnotRfD1	E/S	Bit 1 de dado e sinal de refresh
MemAD2-31	E/S	Dados e endereços multiplexados
notMemRd	S	Sinal de leitura
notMemWrB0-3	S	Sinais de habilitação de escrita
notMemS0-4	S	Sinais para uso geral
notMemRf	S	Indicador de refresh
MemWait	E	Extende o ciclo de memória
MemReq	E	Requisição de DMA
MemGranted	S	Permissão para realizar DMA
EventReq	E	Requisição de evento
EventAck	S	Reconhecimento de evento
LinkIn0-3	E	Canais seriais de entrada
LinkOut0-3	S	Canais seriais de saída
LinkSpecial	E	Velocidade: 5 ou 20 Mbits/s
Link0Special	E	Velocidade especial para o link 0
Link123Special	E	Velocidade especial (links 1,2,3)

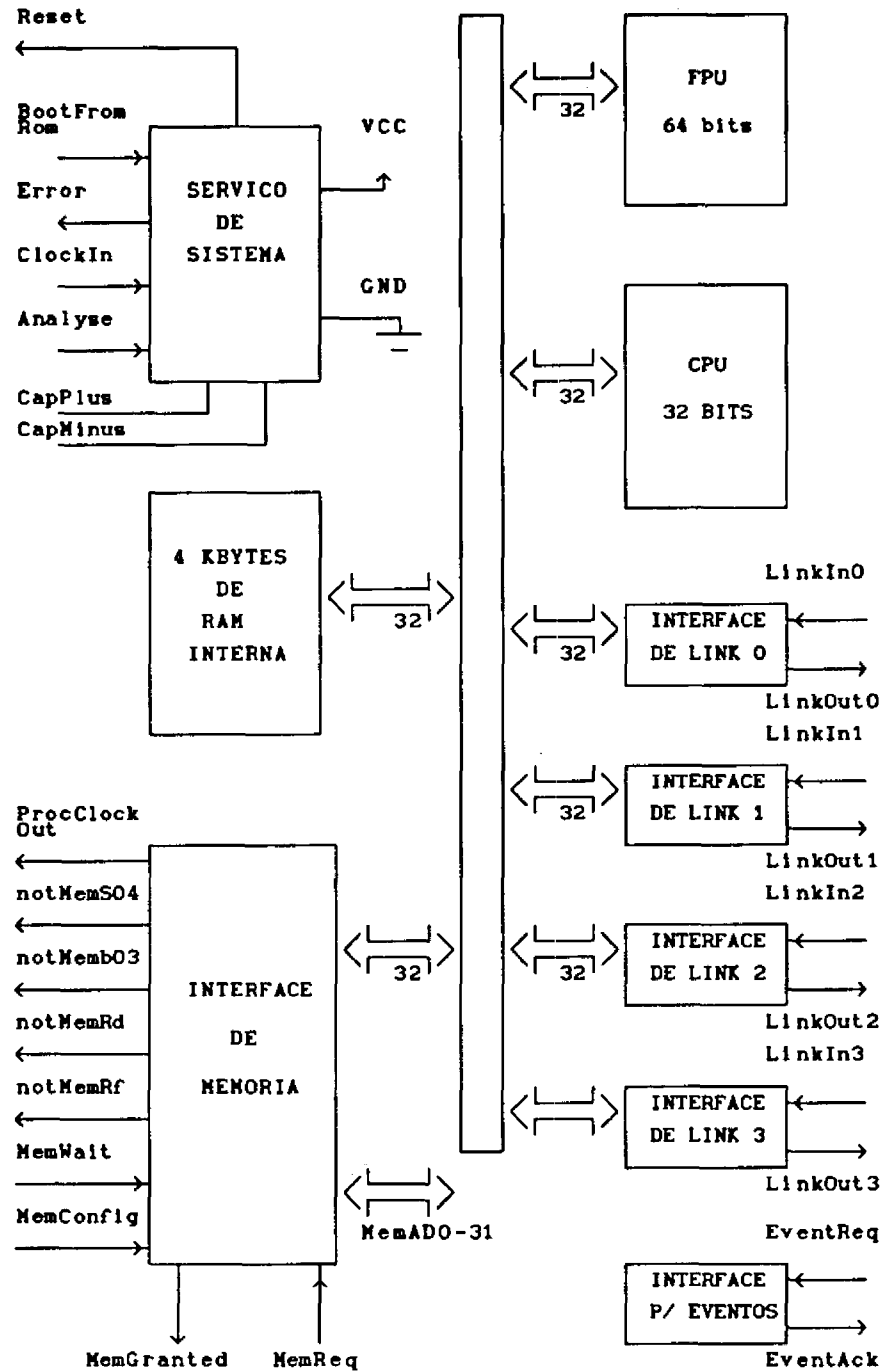


Figura V.1 - Arquitetura do T800 [26].



O microcomputador possui suporte gráfico no seu microcódigo, o qual opera na velocidade da memória [28]. Estas instruções podem ser usadas em operações gráficas tais como janelamento, rolamento e atualização de tela. Estão disponíveis instruções envolvendo CRC, que são úteis quando se lida com seqüências de dados seriais de tamanhos arbitrários. Desta forma, garante-se a integridade da informação, já que existe um mecanismo de detecção de erros. Outra característica bastante útil, em se tratando de reconhecimento de padrões, é a capacidade de contar bits iguais a "1" numa palavra.

#### V.1.2.1 - Registradores.

Devido à disponibilidade de memória rápida no "transputer", existem apenas seis registradores (figura V.2). O reduzido número de registradores, e o conjunto de instruções fazem com que o processador tenha as vias de dados e a lógica de controle simples e rápidas.

O espaço de trabalho de um processo consiste de um vetor de palavras de memória e é usado para manter as variáveis locais e os valores temporários manipulados pelo mesmo. É organizado sob a forma de uma pilha decrescente, cujo fim é endereçado pelo apontador de espaço de trabalho. As variáveis locais são endereçadas como deslocamentos em relação a este apontador.

O apontador de instruções aponta para a próxima instrução a ser executada. O registrador de operando é usado na formação dos operandos das instruções. A, B e C formam uma pilha de avaliação de expressões e são usados pela maioria das instruções aritméticas e lógicas. O carregamento de

um valor na pilha, faz com que o conteúdo de B seja colocado em C e o de A em B, antes do carregamento de A. A remoção de um valor de A faz com que o conteúdo de B seja armazenado em A e o de C em B. Não há mecanismo de "hardware" para detecção de erro no caso de uma tentativa de carregamento de mais de três valores na pilha. O compilador deve encarregar-se de corrigir este problema.

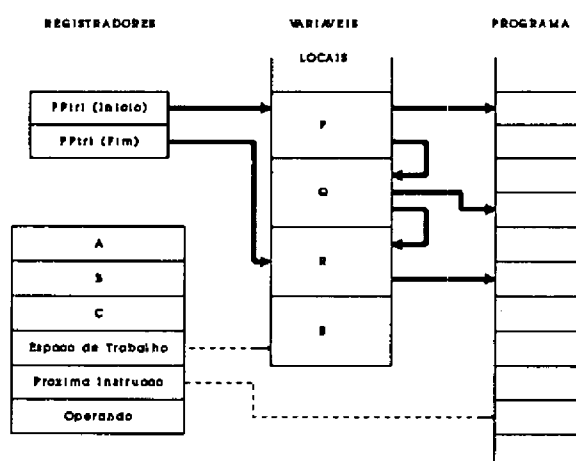


Figura V.2 - Registradores do processador do T800 [27].

#### V.1.2.2 - Implementação de Processos Concorrentes.

Um processo possui início, meio e fim e nada mais é do que uma seqüência de instruções. Os processos podem ter prioridades atribuídas, sendo possível a existência de um número qualquer deles. A CPU possui um escalonador em seu microcódigo, o qual habilita a execução de qualquer número de processos concorrentes, através de multiplexação de tempo. Isto elimina a necessidade de um núcleo no "software". Num dado tempo, um processo concorrente pode estar em um dos seguintes estados:

- .Ativo: sendo executado ou numa lista esperando execução;
- .Inativo: pronto para realizar uma operação de E/S ou esperando até que seja atingido um determinado instante de tempo.

A alocação de espaço para processos concorrentes é estática e, geralmente, feita pelo compilador. O escalonador trabalha de tal forma que os processos inativos não consomem qualquer tempo do processador. Os processos ativos que estão esperando para serem executados são mantidos em duas listas encadeadas de espaços de trabalho. Uma delas relaciona os processos de alta prioridade, a outra os de baixa prioridade. Cada lista é implementada usando-se dois registradores que apontam, respectivamente, para o primeiro e o último processo.

Registradores de alta prioridade:

FPtr0 - apontador de início da lista.

BPtr0 - apontador de fim da lista.

**Registradores de baixa prioridade:**

**FPtr1** - apontador de início da lista.

**BPtr1** - apontador de fim da lista.

Na lista encadeada de processos mostrada na figura V.2, S está em execução; P, Q e R estão ativos, aguardando por execução. Somente os registradores relacionados com os processos de baixa prioridade são mostrados.

Cada processo é executado até que tenha completado a sua ação, mas é desescalonado durante uma espera por comunicação com outro "transputer" ou processo, ou ainda, até que transcorra um determinado intervalo de tempo. Um processo de baixa prioridade pode ser executado, no máximo, durante dois intervalos de tempo ("time slices") antes de ser forçosamente desescalonado. Sempre que um processo for desabilitado, o seu apontador de instrução será salvo no espaço de trabalho e o próximo processo será removido da lista. Os apontadores de escalonamento são atualizados, por instruções apropriadas e não devem ser alterados diretamente. O tempo de chaveamento de processos é menor do que 1µs como consequência da necessidade de salvamento do contexto e pelo fato de não ser preciso salvar a pilha de avaliação no caso de reescalonamento.

**V.1.2.3 - Comunicação.**

Os "links" bidirecionais tornam possível a comunicação direta entre processos residentes em "transputers" distintos (figura V.3).

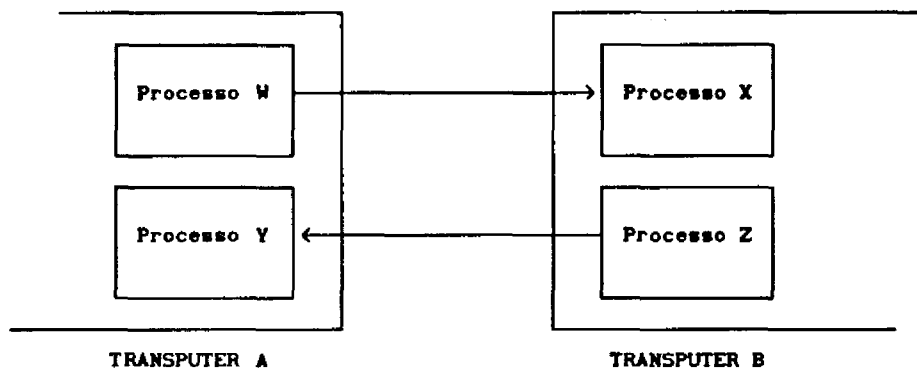


Figura V.3 - Comunicação entre processos [26].

Para que a comunicação seja sincronizada, cada mensagem deve ser reconhecida. Conseqüentemente, um "link" requer pelo menos um pino para cada direção. Existe uma correspondência entre a entrada e a saída de um "link" e os canais de OCCAM. O uso de um protocolo para a transmissão de uma seqüência arbitrária de bytes permite a interconexão de "transputers" cujas palavras tenham comprimentos diferentes. Cada mensagem é transmitida como uma seqüência de comunicações envolvendo um byte de cada vez. Portanto, o "buffer" de recepção precisa ter o tamanho de apenas 1 byte, para garantir que não haja perda de informação.

Um canal entre dois processos que estão sendo executados num mesmo "transputer" é implementado utilizando-se uma palavra da memória interna. O processador possui várias operações de suporte à comunicação por troca de mensagens, que usam o endereço do canal para determinar se o mesmo é interno ou externo. Logo, a mesma seqüência de instruções pode servir para ambos os casos, permitindo que um processo seja escrito e compilado sem conhecimento de onde os seus canais serão ligados.

O processo que ficar pronto primeiro deve esperar até que o segundo também esteja. Um processo realiza uma entrada ou saída carregando na pilha o apontador para a mensagem, o endereço de canal e o tamanho da mensagem. Os dados são transferidos se o outro processo estiver pronto, caso contrário, o processo será desescalonado.

#### V.1.2.4 - Prioridades.

Há dois níveis de prioridade: 1 (baixa) e 0 (alta). Espera-se que os processos de alta prioridade sejam executados num curto período de tempo. Se um ou mais processos de alta prioridade estiverem habilitados, então um deles será seleccionado e executado. Um processo só será interrompido no caso de ter que esperar por comunicação, entrada para o *timer* ou ainda se o mesmo for concluído. Os processos de baixa prioridade são multiplexados visando uma distribuição eqüitativa do tempo do processador entre as tarefas.

#### V.1.2.5 - Temporizadores.

O T800 possui dois temporizadores de 32 bits, que permitem um controle preciso dos processos e o desescalonamento destes após um intervalo de tempo específico. Um deles é acessado apenas por processos de alta prioridade e é incrementado a cada microsegundo. O seu ciclo completo dura 4295 segundos. O outro é acessado apenas pelos processos de baixa prioridade e é incrementado a cada 64 microsegundos, o que resulta em 15625 contagens por segundo. O seu ciclo dura 76 horas. A figura V.4 mostra dois processos esperando numa fila, um pelo instante 21 e o outro pelo instante 31.

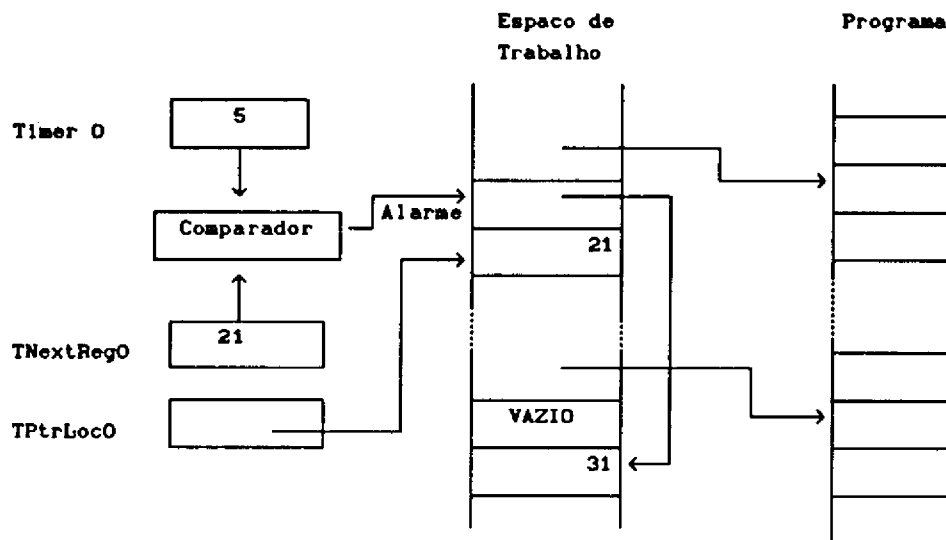


Figura V.4 - Registradores do timer 0 [27].

#### V.1.2.6 - A Unidade de Ponto Flutuante (FPU).

A FPU possui uma pilha de avaliação de três posições (registradores AF, BF e CF), semelhante a da CPU. Os endereços dos valores em ponto flutuante são formados na pilha da CPU, que também é responsável pela transferência de valores entre a memória e a pilha da FPU. O tamanho de palavra da CPU é independente do utilizado na FPU. É possível usar a mesma FPU com, por exemplo, uma CPU de 16 bits como a usada no T212.

A FPU trabalha com 64 bits, comprimento simples (32 bits) e duplo (64 bits), seguindo o padrão ANSI-IEEE 754-1985. Atinge 1,5 Mflops com 20 MHz e 2,25 Mflops com 30 MHz (figura V.5). Contém um microcódigo para manipulação de números em ponto flutuante que usa a pilha de avaliação. Os números denormalizados são completamente suportados pelo "hardware". Da mesma maneira que na pilha da CPU, a pilha da FPU não é salva quando ocorre um

reescalonamento. A FPU pode ser utilizada em processos de baixa e de alta prioridade. Quando um processo de alta prioridade interrompe um de baixa, o estado da FPU é salvo no seu interior. A CPU servirá imediatamente a interrupção, assim que completar a operação atual. O processo de alta prioridade não será iniciado antes que a FPU tenha completado a sua operação corrente.

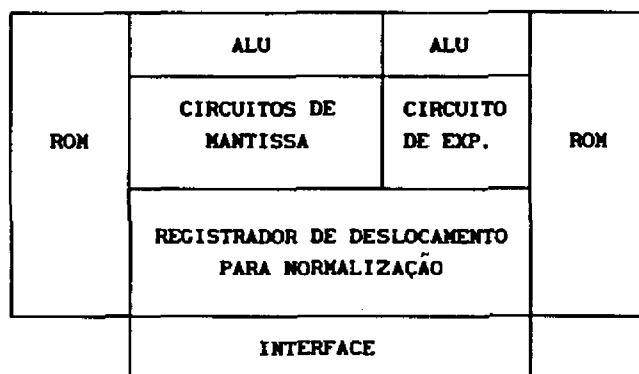


Figura V.5 - Diagrama em blocos da FPU [26].

Os pontos numa seqüência de instruções onde os dados devem ser transferidos de/para a FPU são chamados pontos de sincronização. Num ponto de sincronização, a primeira unidade de processamento a ficar pronta irá esperar que a outra também esteja. Então, a transferência de dados ocorrerá e ambos os processadores continuarão a operar concorrentemente. Os endereços necessários são calculados pela CPU, para explorar ao máximo a concorrência. Assim sendo, o desempenho do sistema é otimizado, através da minimização do tempo de inatividade da CPU e da FPU.

As operações básicas são realizadas por instruções únicas. Contudo,



algumas instruções pouco usadas são selecionadas por um valor contido no registrador A. Os tempos de operação para o T800 em operações de ponto flutuante são mostrados na tabela V.2. O tempo de duração de uma operação não é uma medida confiável do desempenho de um sistema. Usando-se o padrão de desempenho de Wheatstone, o qual é um típico programa científico, tem-se os resultados mostrados na tabela V.3.

Tabela V.2 - Desempenho da FPU [30].

OPERAÇÃO	T800-30		T800-20	
	COMP. SIMPLES	COMP. DUPLO	COMP. SIMPLES	COMP. DUPLO
add	230 ns	230 ns	350 ns	350 ns
subtract	233 ns	233 ns	350 ns	350 ns
multiply	433 ns	700 ns	650 ns	1050 ns
divide	633 ns	1133 ns	950 ns	1700 ns

Tabela V.3 - Desempenho segundo o padrão de Wheatstone [30].

SISTEMA	WHEATSTONES/S, PRECISÃO SIMPLES
IMS T414-20	704K
NS 32332-32081 15 MHz	728K
MC 68020/68881 (SUN3) 16/12.5 MHz	860K
Micro VAX II com FPA	925K
Intel 80386/80387 20 MHz	1860K
Fairchild Clipper 33 MHz	2220K
IMS T800-20	4548K
IMS T800-30	6800K

#### V.1.2.7 - Seção de Serviços de Sistema.

A Seção de Serviços de Sistema contém toda a lógica necessária à inicialização e operação do "transputer", além de facilidades para detecção e análise de erros. A seguir serão vistos alguns aspectos referentes à mesma.

##### a) Temporização.

O T800 usa um relógio externo de 5 MHz. Os relógios internos possuem frequências mais altas e são derivados do externo. Isto simplifica o projeto de sistemas e evita os problemas resultantes da distribuição de relógios externos de alta frequência. Os processadores podem ser conectados a um relógio comum ou podem possuir relógios individuais (figura V.6), desde que se garanta a estabilidade dos mesmos. No caso de sistemas com vários relógios, o defasamento existente entre as diversas entradas não é importante, devido à natureza assíncrona dos "links".

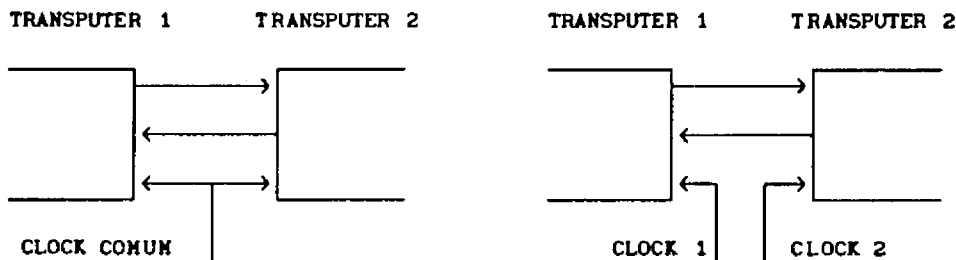


Figura V.6 - Relógio dos "transputers" [26].

**b) Autocarregamento.**

O programa que será executado após o "reset" pode residir dentro do espaço de endereçamento do "transputer", ou pode ser carregado através de qualquer um dos "links" seriais.

**c) Execução Passo-a-Passo.**

O estado de uma rede de "transputers" pode ser investigado utilizando-se para tal o pino Analyse. Isto é feito colocando-se o sistema em estado de parada (*halt*), de modo que cada "transputer" do mesmo possa ser examinado individualmente (figura V.7). Esta última característica facilita o desenvolvimento de sistemas.

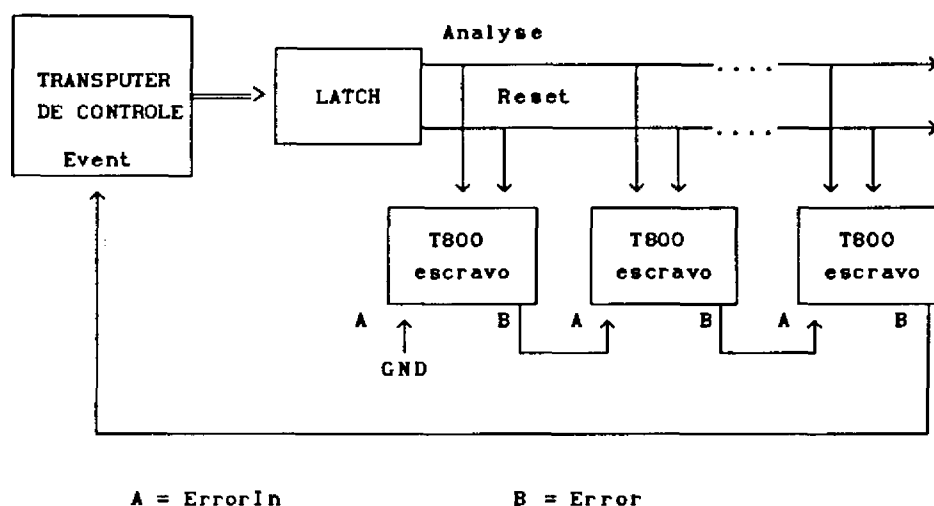


Figura V.7 - Gerenciamento de erro num sistema multi-transputer [27].

Assim sendo, pode-se acompanhar a situação dos processos que estão sendo executados em um "transputer". Os processos ativos estarão nas filas de processos e poderão ser encontrados através de listas encadeadas. Os processos que estão aguardando a sua vez, de acordo com o tempo marcado pelos temporizadores, serão encontrados numa pilha, onde são guardadas informações sobre as filas dos temporizadores. Os processos que estão aguardando comunicação podem ser encontrados examinando-se as palavras de controle de canal disponíveis na memória (figura V.7).

#### d) Detecção de erros.

A saída do pino Error fornece o resultado do OU lógico entre o "flag" de erro e a entrada do pino ErrorIn. Se Error estiver em nível alto pode significar que o sinal ErrorIn está em nível 1 ou que foi detectado um erro em um processo. Um erro interno pode ser causado, por exemplo, por um "overflow" aritmético, uma divisão por zero, violação dos limites de um arranjo ou pelo fato do "flag" ter sido colocado em nível 1 diretamente. O sinal Error também pode ser alterado pela FPU, sob certas circunstâncias. Um processo pode ser programado de modo a parar, se Error for igual a 1. Desta forma, o mesmo não poderá transmitir dados errados. Os processos relacionados com o processo que apresenta erro pararão, devido ao comprometimento das informações. ErrorIn não afeta o "status" do processador.

Um método alternativo é fazer com que o processo ou o "transputer" em erro interrompa o funcionamento dos restantes, através do "daisy chain" dos sinais Errorin e Error pertencentes aos processadores.

#### **e) Seleção de Frequência.**

A frequência do processador pode ser selecionada através de 3 pinos (ProcSpeedSelect0-2), em passos discretos, a partir de 17,5 MHz até o máximo permitido, que depende do componente (20 MHz para o T800-20 e 30 MHz para o T800-30). Um dispositivo de 30 MHz atinge uma taxa de desempenho de 15 MIPS.

#### **V.1.2.8 - Memória e Interface com Periféricos.**

##### **a) Memória.**

Dado um sistema, cada "transputer" tem a sua memória local. A largura de banda total da memória é proporcional ao número de "transputers" presentes. Este comportamento é diferente de um sistema no qual haja memória global, onde os processadores adicionais devem compartilhar do mesmo barramento e, portanto, a largura de banda decresce à medida que outros processadores são adicionados.

As linhas de dados são multiplexadas com as de endereços, oferecendo uma taxa de leitura de dados de 4 bytes a cada 100 ns (40 Mbytes/s) para um dispositivo de 30 MHz. Um controlador de memória configurável faz a temporização, o controle e fornece os sinais de "refresh" para DRAM.

O T800 possui internamente 4 Kbytes de memória estática rápida, o que possibilita altas taxas de transmissão de dados. Cada acesso interno à memória dura um ciclo de processador (ProcClockOut). O "transputer" também pode acessar 4 Gbytes de memória externa. Tanto a memória externa quanto a

interna fazem parte do mesmo espaço de endereçamento linear. A memória interna é endereçada byte a byte, com palavras compostas de 4 bytes, onde o menos significativo é o de endereço mais baixo. Devido ao fato das interfaces de memória não serem compartilhadas e estarem dissociadas das interfaces de comunicação, estas podem ser individualmente aperfeiçoadas. Deste modo, diferentes "transputers" podem ser produzidos para serem usados em projetos com o mínimo de componentes externos. A EMI (Interface de Memória Externa) permite o acesso a palavras de 32 bits e pode ser implementada com memórias RAM dinâmicas e estáticas, além de ROM e EPROM. Existem 13 configurações de memória que podem ser selecionadas através de um único pino.

#### **b) Interfaceamento com Periféricos.**

Em geral, uma aplicação baseada em "transputers" conterá vários deles comunicando-se através dos "links". Há três métodos para a comunicação, os quais são descritos a seguir.

O primeiro deles é o uso de "transputers" de controle (figura V.8) que são conectados diretamente ao respectivo periférico, pois possuem um "hardware" especialmente desenvolvido para a aplicação. O "software" do "transputer" pode ser desenvolvido em OCCAM e o controle da interface feito pelos canais de OCCAM. O mesmo estabelece uma comunicação entre o processador e o "hardware" específico.

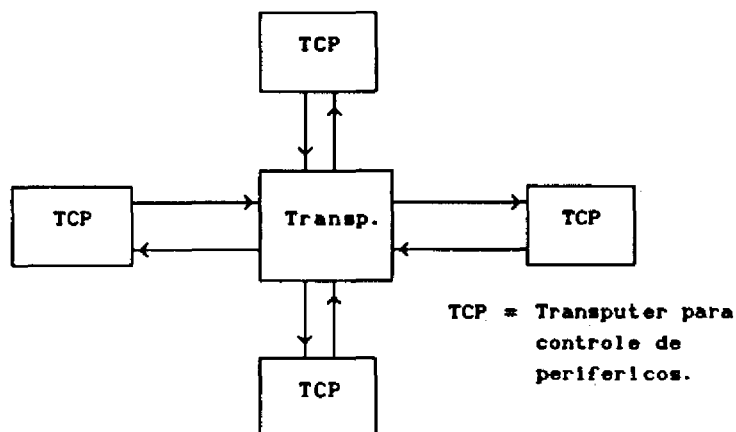


Figura V.8 - Transputer com periféricos de controle.

O segundo método usa adaptadores de "link" (figura V.9), que são circuitos integrados destinados à comunicação com interfaces especializadas e que não seguem o padrão da INMOS. Eles fazem a conversão paralelo/serial e serial/paralelo dos dados. Estas unidades permitem que os "transputers" sejam conectados a sistemas convencionais baseados em barramentos.

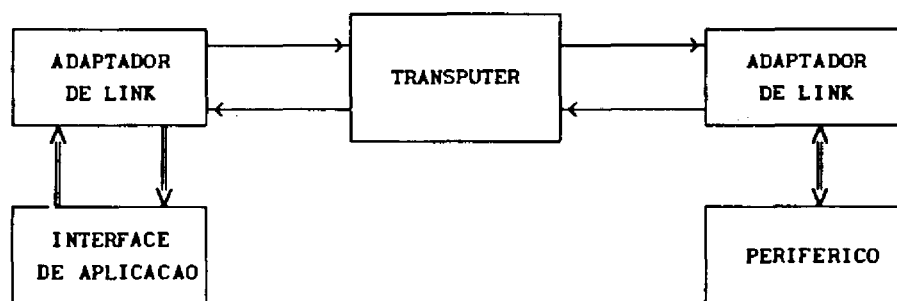


Figura V.9 - Transputer com adaptadores de "link".

O terceiro método mapeia o periférico em memória e controla o mesmo através de instruções de entrada e saída.

#### V.1.2.9 - LINHAS DE COMUNICAÇÃO.

##### a) "Links".

O uso de "links" para a comunicação, simplifica o projeto de um sistema, pois permite a construção de redes de tamanho e topologias arbitrários (figuras V.10 e V.11). A comunicação ponto-a-ponto oferece muitas vantagens em relação ao uso de barramentos:

- a) não existe contenção do sistema devido à comunicação;
- b) há poucas perdas provocadas por capacitâncias, à medida que outros "transputers" são adicionados ao sistema;
- c) a largura da banda passante cresce com o aumento do número de componentes;

O protocolo dos "links" e as suas características elétricas obedecem a um padrão da INMOS. Todos os "transputers" suportam uma velocidade de comunicação de 10 Mbits/s, que facilita a comunicação entre dispositivos diferentes. O T800 trabalha com outras velocidades (5 e 20 Mbits/s), cuja seleção é feita através dos pinos LinkSpecial, Link0Special e Link123Special. Cada "link" consiste de dois canais unidirecionais, nos quais são transmitidos serialmente bits de dados e de controle. Os sinais são compatíveis com o nível TTL e a sua capacidade pode ser aumentada inserindo-se "buffers".



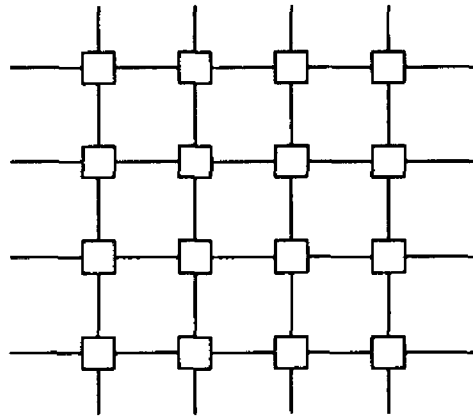


Figura V.10 - Rede com comunicação ponto-a-ponto.

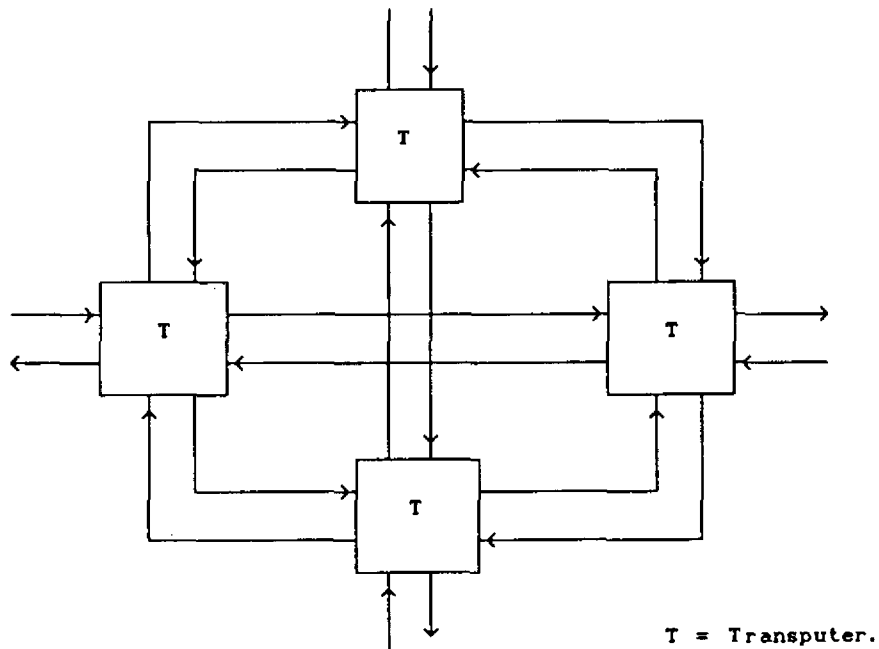


Figura V.11 - Interconexão entre "transputers" com 4 "links" cada um.

**b) Interrupções.**

Os pinos EventReq e EventAck são canais para interfaceamento entre um evento externo e um processo interno. Quando um evento externo coloca o pino EventReq alto, o canal do evento externo é preparado para se comunicar com um processo. O processador afirmará o pino EventAck quando ambos os canais estiverem prontos. O processo, caso esteja em espera é escalonado. EventAck é removido após EventReq ser desativado. Somente um processo de cada vez pode usar o canal de evento. Uma forma de se interromper o programa de um "transputer" é deixar uma tarefa de alta prioridade esperando por um evento.

**V.2 - A LINGUAGEM OCCAM.**

A versão inicial da linguagem (OCCAM 1) foi introduzida pela INMOS em 1983. A mesma evoluiu, com a inclusão de tipos e outras características, para uma nova versão conhecida como OCCAM 2 [31-33]. Ela difere de outras em muitos aspectos. As linguagens MODULA 2 e ADA, por exemplo, possuem características que permitem decompor um programa em processos semi-independentes. Ambas suportam uma forma de concorrência onde se supõe que os processos compartilham de um mesmo computador. A linguagem OCCAM baseia-se no fato do "transputer" ter sido concebido para trabalhar cooperativamente com outros numa mesma tarefa, ou seja, foi desenvolvida para ser utilizada em sistemas distribuídos com múltiplos "transputers".

### V.2.1 - Concorrência.

A linguagem OCCAM 2 é a mais indicada para trabalhar com o "transputer", pois o seu desenvolvimento esteve diretamente ligado a ele e além disso o seu "Assembly" é bastante complexo [27,28]. A mesma permite uma programação seqüencial fortemente estruturada, assim como um controle completo de processos concorrentes. O modelo de concorrência da linguagem OCCAM baseia-se na idéia de processo, que pode ser definido como uma unidade de um programa.

Uma aproximação bastante interessante, consiste em pensar num processo como sendo constituído por vários subprocessos concorrentes. Um programa escrito em OCCAM é capaz de descrever vários processos que devem ser executados concorrentemente e que se comunicam por troca de mensagens. Estes processos não levam em conta as primitivas e os conceitos relacionados com a sincronização dos mesmos (variáveis compartilhadas, semáforos e regiões críticas, etc). A linguagem OCCAM não requer e não suporta compartilhamento de variáveis. A troca de mensagens sempre ocorre entre processos. Não há múltiplos receptores e transmissores, nem difusão ou incertezas associadas às mensagens. Estas por sua vez não necessitam de armazenamento prévio, portanto, o envio e a recepção de uma mensagem envolve a sincronização momentânea entre os dois processos participantes.

O modelo de concorrência adotado é aplicável a processos que estejam sendo executados num único processador ou numa rede deles. Como cada processador é controlado por uma seqüência única de instruções, torna-se evidente que os processos que estão sendo executados em um mesmo processador não são verdadeiramente concorrentes. Contudo, eles podem ser

multiprogramados e os efeitos da concorrência podem ser reproduzidos (concorrência virtual). No "transputer", esta multiprogramação é gerenciada por "hardware", sem que haja necessidade de um sistema operacional.

A principal característica da linguagem OCCAM é a facilidade com a qual a concorrência pode ser expressa, o que permite a decomposição de uma aplicação numa coleção de processos paralelos, que se comunicam através de canais. Um canal é um objeto usado para fins de comunicação entre dois processos, onde um deles é usado para entrada de dados e o outro para saída. Não é possível a comunicação entre processos por compartilhamento de dados, em vez disso, usa-se trocas de mensagens através de canais. Os compiladores OCCAM geralmente fazem uma verificação do programa, para confirmar se os canais e as variáveis estão sendo utilizados adequadamente pelos processos paralelos. Portanto, o modelo de concorrência adotado é o mesmo, em se tratando de um único processador ou de vários deles. Isto traz duas vantagens:

- elimina os erros de programação associados à comunicação entre variáveis compartilhadas; e
- simplifica a distribuição de processos entre os processadores.

Este comportamento traz algumas implicações, uma vez que não existe um ambiente global de programa. Um processo somente se comunica com o meio exterior via canais. A construção paralela PAR indica que alguns processos devem ser executados em conjunto. O exemplo abaixo mostra a decomposição em alto nível de um editor de textos, onde há três processos, com as seguintes finalidades: edição, controle de teclado e atualização da tela (figura V.12). Os canais entrada e saída controlam a troca de informações entre

processos.

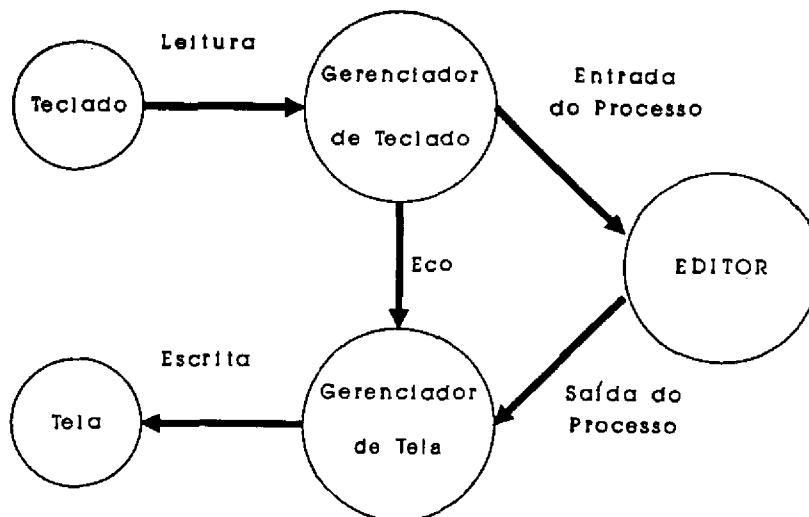


Figura V.12 - Representação dos processos que compõem o editor.

O trecho de programa abaixo corresponde à descrição em OCCAM do editor de textos:

**PAR**

**editor(entrada, saida)**

**gerenciador.de.teclado(teclado, entrada)**

**gerenciador.de.tela(saida, tela)**

Os processos que compõem o editor são representados por círculos e a interação entre eles por setas. Tal construção paralela difere daquelas encontradas em outras linguagens, devido à granularidade do paralelismo em questão. Deve-se explorar formas de paralelismo naturais, o que somente pode

ser feito através de uma escolha adequada de construções da linguagem de modo a assegurar a eficiência da concorrência e da comunicação.

Em qualquer instante de tempo, um programa em OCCAM pode ser encarado como uma coleção de processos paralelos, com dados fluindo pelos canais. Os mesmos são combinados de forma que após a execução de um programa, a estrutura paralela do mesmo será alterada à medida que os processos se ramificam e se reúnem. O exemplo abaixo mostra como processos paralelos e seqüenciais podem ser combinados num pacote gráfico implementado em "pipeline". O processo *vista* lê o banco de dados, extraíndo a informação referente ao ponto de vista do observador e enviando-a para um canal chamado *saida.vista*. O restante do "pipeline" de processos encarrega-se das transformações dos dados, até que a informação processada seja trabalhada pelo último processo, *mostrador*, que envia os resultados para o monitor de vídeo.

```

...Declaracao dos dados

SEQ

...inicializacao dos parametros

WHILE flag

    SEQ

        PAR

            vista(banco.de.dados,

                    ponto.de.vista,saida.vista)

            recorte(saida.vista,saida.recorte)

            perspectiva(saida.recorte,saida.persp)

            superf.escondida(saida.persp,saida.superf)

```

**mostrador(saida.superf,tela)**  
**...atualizacao do banco de dados**

A figura V.13 mostra a representação esquemática deste programa. Foi utilizada a convenção (...) para designar uma seção de programa que não é mostrada na íntegra. Os processos paralelos são simbolizados por círculos claros e os seqüenciais por círculos escuros.

#### **V.2.2 - Localização e Distribuição de Processos.**

A descrição em OCCAM de um sistema permite que as questões relacionadas com a decomposição do algoritmo em vários processos concorrentes fiquem independentes do "hardware" no qual será executado. Cada implementação requer uma especificação de quantos processadores serão necessários, qual a topologia utilizada e de como os processos serão distribuídos pelos processadores. Estas especificações não devem resultar em alterações no programa.

Na prática, os programas desenvolvidos para "transputers" não são completamente independentes das especificações do sistema. A menos que seja cuidadosamente elaborado, um programa poderá ser executado em apenas um tipo de rede ou por um conjunto pequeno de redes de topologias similares. A execução do programa por outros tipos de redes implica numa mudança do programa. Às vezes as alterações são mínimas, em outros casos podem exigir modificações profundas. Os programadores devem procurar desenvolver programas que possam ser transportados para configurações variadas.

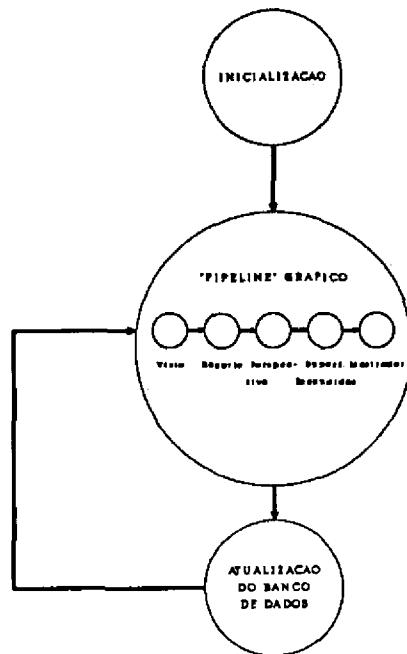


Figura V.13 - Diagrama do pacote gráfico interativo do exemplo anterior.

### V.2.3 - Comunicação.

Para que ocorra uma comunicação num canal que liga dois processos, um deles:



deve transmitir o dado e o outro deve recebê-lo. Em OCCAM, a comunicação é sincronizada, ou seja, antes da ocorrência da mesma, o processo transmissor deve ter alcançado o trecho de programa que realiza a ação de saída, enquanto o processo receptor deve ter alcançado o trecho correspondente à entrada. O processo que primeiro alcançar a sua etapa de comunicação é bloqueado, até que o outro chegue à ação correspondente. As ações de entrada e saída em OCCAM são representadas da seguinte maneira:

- entrada: um processo realiza uma entrada, por exemplo,  $c ? v$ , atribuindo o valor lido no canal  $c$  à variável  $v$ ;
- saída: um processo executa uma saída, por exemplo,  $c ! e$ , colocando o valor da expressão  $e$  no canal  $c$ .

Muitas vezes é necessário efetuar uma entrada a partir de um processo que possa ser selecionado dentro de um conjunto de processos concorrentes. O objetivo é a execução do primeiro processo que apresente um valor na entrada do seu respectivo canal. Para tal, há uma construção chamada ALT, que seleciona um dentre vários processos alternativos, dependendo da resposta de um canal pertencente a um conjunto de canais de entrada.

```
WHILE TRUE
```

```
  ALT
```

```
    canal.esquerdo ? x
```

```
      saida ! x
```

```
    canal.direito ? x
```

```
      saida ! x
```

No exemplo anterior, um processo amostra continuamente dois canais: `canal.esquerdo` e `canal.direito`. A informação recebida por um deles é transmitida para o canal `saída`. Cada ramo do ALT consiste numa ação de entrada (condição de guarda) e um processo a ser executado, caso esta condição seja satisfeita. Se mais de uma condição de guarda for satisfeita, é feita uma seleção aleatória (não-determinística). O programador não tem controle direto sobre a condição de guarda, a menos que inclua outras condições de controle, tais como expressões booleanas, que introduzam um controle adicional. Neste caso, somente aquelas condições de guarda que possuírem as suas respectivas expressões booleanas com valor `TRUE` serão avaliadas.

#### V.2.4 - Estruturação dos Programas.

A estrutura dos programas escritos em OCCAM é determinada por dois fatores:

- **estrutura de processos:** diz respeito à maneira como os processos podem ser combinados em paralelo, com o objetivo de produzir processos mais complexos; e
- **abstração procedural:** diz respeito à criação de subrotinas ou procedimentos.

As construções `SEQ`, `PAR` e `ALT` são usadas para criar estruturas complexas a partir de outras mais simples. As declarações que compõem uma dada estrutura são indicadas por indentação, que corresponde às chaves utilizadas em "C" e ao par `begin-end` do PASCAL. A indentação é considerada

como parte da linguagem e não apenas como um estilo, sendo inclusive verificada pelo compilador.

As declarações de canais e de variáveis podem ser feitas em qualquer trecho de um programa, desde que precedam o código associado às mesmas.

Uma declaração de procedimento consiste do seu nome e de um conjunto de parâmetros formais. Os parâmetros a serem passados para um procedimento podem ser: valores de variáveis, canais, ou ainda um arranjo de qualquer um destes elementos. Um procedimento não pode ser passado como parâmetro para outro. Parâmetros podem ser passados por valor ou por referência.

Uma função é um procedimento que retorna valores, os quais podem ser usados em expressões e que não apresenta efeitos colaterais, pois não altera os valores das variáveis, nem compromete a comunicação.

Não é permitida a chamada recursiva de procedimentos e funções, devido a certas restrições quanto ao tempo de execução de um programa e uso da memória [32]. A RAM interna do "transputer" é usada pelo compilador para armazenar os endereços de base de arranjos e as variáveis locais dos procedimentos. O tamanho e a natureza de um arranjo são fixados durante a etapa de compilação, assim sendo, um simples cálculo pode ser usado para determinar o endereço de um elemento. A RAM interna também serve como pilha, ou seja, uma área onde as variáveis usadas repetidamente são colocadas com o objetivo de aumentar a velocidade de execução do programa. Uma variável deste tipo é acessada num único ciclo, enquanto aquelas que residem na RAM externa requerem vários ciclos. O uso da RAM interna da maneira descrita se reflete na especificação da linguagem. Se a recursividade fosse possível, o

empilhamento repetitivo de variáveis locais poderia provocar confusão. A inserção de recursividade requer a manutenção e o controle da pilha, além de alocação dinâmica de memória.

#### V.2.5 - Tipos e Estruturas de Dados.

OCCAM 2 é uma linguagem "tipada", onde os canais e variáveis devem ter seus tipos declarados. Os tipos básicos são:

BOOL → booleano

INT → inteiro

BYTE → byte

REAL → real

Os inteiros podem possuir os seguintes comprimentos em bits: 16, 32 e 64. Já os reais, podem ser representados internamente por palavras com 32 ou 64 bits. Em OCCAM 2 não existe conversão automática de tipos no processo de avaliação de expressões. Esta deve ser feita explicitamente. No exemplo dado abaixo, o valor em ASCII de "a" (tipo byte) é convertido para inteiro.

```
x := INT 'a'
```

No caso da conversão de números reais para inteiros, há operadores que especificam o modo pelo qual isto será feito. São eles ROUND e TRUNC, onde o primeiro arredonda o valor numérico e o segundo faz o truncamento do mesmo.

Declarações de arranjos de canais e de variáveis também são suportadas. O tamanho de um arranjo é fixo e deve ser estabelecido quando da sua declaração. A seguir são mostrados alguns exemplos:

```
[20]INT maria:          -- Vetor de inteiros
[90]CHAN OF INT chave:   -- Vetor de canais
[8][8]BYTE tela:        -- Arranjo bidimensional de bytes
```

As variáveis do tipo inteiro podem ser tratadas como palavras de máquina, permitindo o acesso em baixo nível à CPU. Tipos definidos pelo usuário ou estruturas do tipo "record" não são suportados.

#### V.2.6 - Abreviações.

Uma abreviação (OCCAM2) é uma declaração que introduz um novo nome para um objeto já existente ou para o valor de uma expressão. Isto é particularmente interessante quando se deseja trabalhar com um elemento isolado de um arranjo. A abreviação abaixo permite que se use o nome `no.corrente` em lugar de `nos[x.indice][y.indice]`.

```
no.corrente IS nos[x.indice][y.indice];
```

Uma abreviação pode também introduzir um novo nome para uma porção de um arranjo já existente. Por exemplo, um vetor pode ser decomposto em duas partes menores através do uso deste recurso:

```
[100]INT  vetor:
VAL      metade IS (vetor)/2:
metade inferior IS [vetor FROM 0 FOR metade]:
metade superior IS [vetor FROM metade FOR metade]:
```

O emprego de novos nomes para objetos já existentes é uma causa comum de erros de programação, uma vez que uma dada porção do programa pode aparentar estar trabalhando com dois objetos totalmente diferentes, embora, esteja lidando com apenas um. Isto é conhecido por "aliasing". Tal fenômeno não ocorre em OCCAM2, ou seja, quando um objeto recebe um novo nome como resultado de uma abreviação, o nome anterior do objeto não pode ser usado dentro do programa.

#### V.2.7 - Protocolos dos Canais.

Quando ocorre uma comunicação entre canais é importante que o processo receptor e o transmissor concordem entre si quanto ao formato e ao tamanho dos dados que trocam. A isto dá-se o nome de protocolo de canal. Por exemplo, declarando-se um canal como sendo **CHAN OF INT**, o mesmo estará habilitado a transportar valores inteiros. É desejável que se possa especificar canais capazes de transportar valores de vários tipos. Um protocolo seqüencial especifica que uma certa seqüência de valores, possivelmente de tipos mesclados, podem ser enviados através do canal. A sentença de saída consiste de uma lista de valores com os tipos apropriados. A de entrada é composta por outra lista, com os nomes das variáveis:

**PROTOCOL PAR.BYTE.INT IS BYTE; INT:**

**CHAN OF PAR.BYTE.INT c:**

**BYTE b:**

**INT i:**

**PAR**

```
c ! 'a';23
```

```
c ? b;i
```

Os protocolos sequenciais são úteis quando as mensagens têm um formato fixo. Algumas vezes é útil que a comunicação possua vários formatos. O protocolo abaixo define várias possibilidades para o formato das mensagens, cada qual iniciando por um rótulo (caracter, numero, cadeia e nada). Na definição, cada rótulo é seguido por um tipo, que corresponde ao tipo dos elementos da mensagem.

#### PROTOCOLO DADOS.VARIAVEIS

##### CASE

```
caracter; BYTE
```

```
numero; INT
```

```
cadeia; [256]BYTE
```

```
nada
```

Uma entrada CASE para um canal com este protocolo tem o aspecto mostrado no programa listado abaixo. Cada ramo de entrada consiste de uma lista de nomes de variáveis que guardarão a mensagem que se segue ao rótulo e uma ação a ser executada, caso este elemento seja escolhido. A entrada CASE receberá um rótulo e então escolherá o ramo que deve ser executado.

```

CHAN OF DADOS.VARIAVEIS canal.de.dados:

BYTE ch:

INT i:

[256]BYTE str:

SEQ

...Faca algo

canal.de.dados ? CASE

    character; ch

        ...Acao a ser realizada com o character

    numero; i

        ...Acao a ser realizada com o numero

    cadeia; str

        ...Acao a ser realizada com a cadeia

    nada

        Acao a ser realizada, caso nada ocorra

```

#### V.2.8 - Construções de Controle.

Existe um conjunto pequeno, porém suficiente, de estruturas de controle [32]:

```

IF    → indica condicao
CASE  → indica selecao
REP   → operador de repeticao
WHILE → repeticao com teste

```

As regras para as declarações permitem uma estrutura hierárquica de



programa, na qual é possível declarar funções e procedimentos com qualquer nível de aninhamento, logo os programas são uma combinação de processos e procedimentos [33].

O IF da linguagem OCCAM permite que um programa execute um processo pertencente a um certo conjunto, dependendo do resultado de um teste. Um IF pode possuir qualquer número de processos e seus componentes são avaliados um após o outro. A primeira expressão cuja condição for satisfeita terá a ação correspondente executada. Se nenhuma das condições for verdadeira, o IF não terminará. Logo, o mesmo é diferente do encontrado em outras linguagens. O programador deve analisar todas as possibilidades, verificando se são cobertas pelos ramos do IF. Seja o exemplo abaixo:

```
IF
    x = 1
        canal1 ! y
    x = 2
        canal2 ! y
TRUE
    canal3 ! y
```

No programa anterior, o programa colocará o valor de y no canal3 caso nenhuma das expressões booleanas ( $x = 1$  e  $x = 2$ ) seja satisfeita.

A construção CASE permite que se escolha um processo pertencente a um conjunto de processos, dependendo do valor de uma dada expressão. Se nenhuma delas for satisfeita, o CASE comportar-se-á como um STOP (primitiva de OCCAM

que faz com que um processo nada faça e nunca termine). Os componentes de um CASE que se seguem a um ELSE são analisados se as opções que o antecederam não tiverem sido satisfeitas.

Em OCCAM, são permitidas duas formas de repetição: por teste e por contagem.

No primeiro caso, uma ação é repetida enquanto uma condição de teste não é satisfeita. Há somente um tipo de estrutura de controle usada para teste com repetição. Trata-se do WHILE. O programa abaixo contém um laço com quatro iterações e exemplifica esta forma de repetição:

```

INT x,i:
SEQ
  i := 0
  WHILE i < 4
    SEQ
      entrada ? x
      i := i + 1

```

No segundo caso, a repetição é controlada por um contador, que é usado em conjunto com um endereço de base. Neste caso, processos similares são colocados num arranjo, de modo que possam ser indexados e controlados por um contador. A construção típica é mostrada a seguir, onde REP pode ser uma construção do tipo SEQ, ALT, PAR ou IF (construção replicada).

```

REP indice = base FOR contador
    processo

```

Empregando-se uma construção replicada, a notação fica mais concisa e não é necessário incrementar a variável de indexação, nem tão pouco testar o seu valor várias vezes. O valor da variável de índice é incrementado de um passo a partir do endereço de base. O número de vezes que isto ocorre é especificado pelo valor contido no contador. Um arranjo de qualquer tamanho pode ser preenchido com valores de entrada da seguinte maneira:

```

SEQ i = 0 FOR SIZE arranjo.grande
    entrada ? arranjo.grande[i]

```

#### V.2.9 - Programação em Tempo Real.

Dois aspectos podem ser considerados. O primeiro deles relaciona-se com o fato de que um processo pode ler o valor do relógio interno do "transputer" e o seu comportamento pode ser regulado pelo mesmo. Este acesso é feito através do uso de objetos do tipo **TIMER**. Um "timer" se comporta como se fosse um canal, no qual somente é possível a operação de entrada e todos os valores lidos são do tipo **INT**. Um processo pode investigar qual é o instante atual usando este recurso e solicitar a suspensão de sua execução por um intervalo de tempo predeterminado. Por exemplo, o procedimento abaixo espera por um atraso especificado pelo parâmetro **atraso**:

```

PROC espera(VAL INT atraso)

    INT agora:

```

**TIMER    relógio:**

**SEQ**

**relógio ? agora**

**relógio ? AFTER agora PLUS atraso**

O outro está relacionado com a atribuição de prioridades aos processos, o que pode resultar num aumento de desempenho. Pode-se introduzir prioridades de processos através do uso da construção **PRI PAR**. Esta garante que um processo terá prioridade sobre outro, se ambos estiverem em condições de prosseguir. Isto assegura que processos críticos em relação ao tempo, tenham prioridade sobre processos que não o são. Seja o programa abaixo:

**PRI    PAR**

**gerenciador.de.eventos(entrada.dados,dados.armazenados)**

**processador.de.dados(dados.armazenados,saida.de.dados)**

Neste caso, o processo **gerenciador.de.eventos** tem prioridade sobre o processo **processador.de.dados**, pelo fato de ter sido colocado no início da lista. Desta maneira, ele pode responder aos dados que estão chegando e armazená-los, se necessário.

Uma construção **ALT** também pode ser priorizada (**PRI ALT**), onde as condições de guarda que se encontram no início da lista têm preferência sobre as subseqüentes.

**V.2.10 - Operação em Sistemas com Múltiplos "Transputers".**

O desenvolvimento da linguagem OCCAM teve como um de seus principais objetivos a independência em relação à topologia do sistema. As mesmas técnicas de programação concorrente podem ser usadas em sistemas com um ou mais processadores. Isto significa que a decisão acerca da forma pela qual um programa será mapeado na arquitetura pode ser feita no final do processo de desenvolvimento. Os programas escritos em OCCAM podem ser mapeados numa rede de processadores através da construção PLACED PAR, cuja função é atribuir processos aos processadores, além de atribuir canais aos "links" de comunicação. Este procedimento é chamado de configuração da rede.

## CAPÍTULO VI

### IMPLEMENTAÇÃO DE ALGORITMOS DE PROCESSAMENTO DE IMAGENS E DE VISÃO COMPUTACIONAL NUMA REDE DE "TRANSPUTERS".

Nos capítulos anteriores, caracterizou-se o tipo de problema a ser resolvido (Transformada de Hough para curvas analíticas) e fez-se um levantamento das características das máquinas a serem estudadas neste trabalho (multiprocessadores baseados em "transputers").

Neste capítulo, são descritas e discutidas várias alternativas para a implementação da classe de algoritmos escolhida. Também são feitas avaliações de desempenho dos mesmos em redes de "transputers", cujos resultados foram obtidos com diferentes topologias e distribuições de dados e tarefas.

#### VI.1 - DESCRIÇÃO E ANÁLISE DAS FERRAMENTAS DISPONÍVEIS PARA O DESENVOLVIMENTO DE APLICAÇÕES PARA "TRANSPUTERS".

Os equipamentos disponíveis são as placas TSB04/08 (1 T800), TSB44/48 (4 processadores T800), TSB07 (placa gráfica com um T800) e o supercomputador NCP I (Apêndice I). Todos foram cedidos pelo Núcleo de Computação Paralela do Programa de Engenharia de Sistemas da COPPE/UFRJ.

Atualmente, estão disponíveis os seguintes compiladores paralelos: FORTRAN, Pascal, "C" e OCCAM. Também está disponível um sistema operacional desenvolvido para multiprocessadores baseados em "transputers" chamado

Helios, que segue o padrão UNIX.

Embora os compiladores para FORTRAN, "C" e Pascal ofereçam ambientes de programação mais confortáveis, optou-se pela linguagem OCCAM2, por ser a mesma bastante próxima do comportamento de um "transputer". Ela permite o desenvolvimento de programas mais rápidos, já que é considerada uma linguagem de montagem. Pode-se dizer que o "transputer" nada mais é do que uma implementação em VLSI dos conceitos de concorrência e comunicação de OCCAM.

O maior problema apresentado por esta linguagem, quando o projeto foi iniciado, era a não-existência de ambientes integrados autônomos para desenvolvimento. Naquela época havia apenas duas versões disponíveis do TDS (Transputer Development System): o TDS1 e o TDS2.

O TDS é basicamente um sistema de desenvolvimento integrado para IBM-PC que pode ser usado para edição, compilação, ligação e execução de programas OCCAM. A sua característica mais notável é o fato do editor de textos empregar o conceito de "fold". Esta estrutura é utilizada para encapsular trechos de programa, o que permite a confecção de programas com hierarquias de processos bem definidas.

O TDS1 pode ser executado com ou sem uma rede de "transputers". O fato deste ambiente permitir o desenvolvimento de programas no modo autônomo (standalone) facilita o aprendizado e o exercício de técnicas de programação para máquinas que se comuniquem por troca de mensagens. Porém, este sistema não possibilita uma avaliação criteriosa do desempenho e do comportamento real de uma rede, quando usado no modo autônomo, já que neste caso o mesmo

opera como simulador. Além deste problema, pode-se mencionar outros:

- implementa uma versão antiga da linguagem OCCAM (OCCAM1);
- não dispõe de boas bibliotecas de entrada e saída;
- não possui recursos eficientes de depuração;
- está muito mal documentado;
- só pode ser usado sob o sistema operacional DOS 3.1. ou versão inferior.

O TDS2 não suporta o modo autônomo, porém dispõe de bibliotecas mais elaboradas. Contém um depurador e pode ser usado sob os sistemas operacionais DOS e HELIOS. São fornecidos servidores para uso dentro e fora do ambiente de desenvolvimento. Estes programas têm por finalidade carregar a rede de "transputers" com o código resultante das etapas de compilação e ligação. Os servidores foram escritos em "C" e seus códigos estão disponíveis. Desta maneira, eles podem ser utilizados como ponto de partida para o desenvolvimento de servidores adequados às necessidades do usuário.

#### VI.2 - O USO DO "TRANSPUTER" EM OPERAÇÕES DE PROCESSAMENTO DE IMAGENS DE BAIXO NÍVEL.

O uso de uma arquitetura convencional requer muito tempo de processamento e memória. Uma alternativa eficiente consiste na utilização de paralelismo.

Embora a maior parte das máquinas paralelas existentes para processamento de imagens opere no modo SIMD, o modo MIMD permite a



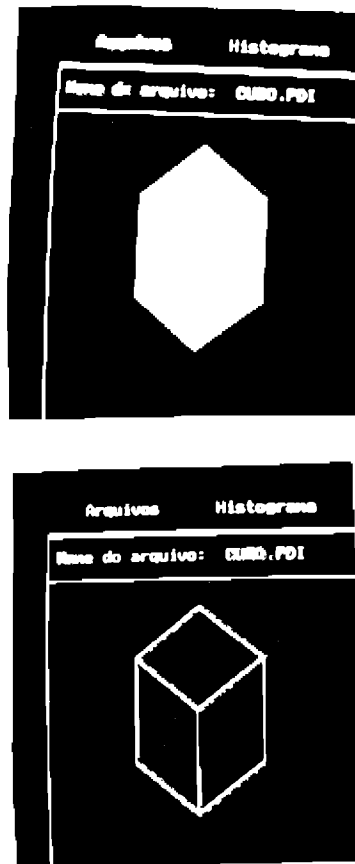
implementação de programas mais complexos e melhor estruturados [34].

As operações de processamento de imagem de baixo nível são, em sua grande maioria, de granularidade fina. Sendo assim, a implementação das mesmas em máquinas SIMD é mais simples e os desempenhos obtidos têm sido altos. Porém, estas máquinas possuem estruturas fixas, geralmente dedicadas aos algoritmos ou classe de algoritmos que executam, além de serem pouco flexíveis no que tange a sua programação. Deve-se ressaltar que as diferenças obtidas são devidas não só às características dos processadores, mas também aos diferentes tipos de compiladores usados. Os programas não estão otimizados.

As arquiteturas baseadas em "transputers" tornam possível embutir diversas configurações de processadores numa rede. Também facilitam o desenvolvimento de programas e permitem a integração de tarefas de processamento de imagens de baixo e alto níveis numa mesma máquina. Tal característica viabiliza o desenvolvimento de máquinas paralelas mais complexas e gerais.

A figura VI.1 mostra o tempo de execução de uma operação de extração de contornos (operador de Sobel), realizada num microprocessador T800-20 e num IBM-PC AT (386/387). O primeiro programa foi escrito em OCCAM e o segundo em "C". Foi utilizada uma imagem de 128 X 256 pixels, 8 níveis de cinza. Os resultados obtidos para outras operações são mostrados na tabela VI.1. Os algoritmos utilizados são descritos no Capítulo II.

Os algoritmos de processamento de imagens podem ser paralelizados a nível de imagem, de tarefa ou combinando-se os dois métodos.



T800-20	0,32
286/287	2,70
386/387	0,81

OBS: Tempos medidos em segundos.

Figura VI.1 - Comportamento do "transputer" em operações de processamento de imagem de baixo nível (operador de Sobel).

Tabela VI.1 - Comparação entre o tempo de execução de rotinas de processamento de imagens escritas em "C" e OCCAM.

Operação (*)	T800-20	286/287 (10MHz)	386/387 (25MHz)
Ajuste de brilho	0,13	0,40	0,10
Modulação de contraste	1,34	7,43	2,25
Realce de bordas	0,51	1,20	0,73

(\*) Tempos medidos em segundos.  
Imagem com 128X256 pixels,  
8 níveis de cinza.

Em se tratando de programas de granularidade fina, é mais eficiente que estes sejam paralelizados a nível de imagem, ou seja, vários processadores executam a mesma tarefa, empregando partições de imagem diferentes como dados de entrada. Os programas concorrentes devem ser escritos de modo que as operações de E/S se dêem concorrentemente com as de cálculo. Em OCCAM isto deve ser feito explicitamente [35].

Cada vez que um item é transferido através de um "link", o microprocessador necessita de cerca de 1 microsegundo para preparar o canal, mas uma vez iniciada a comunicação, esta se dá autonomamente. Assim sendo, o desempenho pode ser aumentado se o número de mensagens trocadas for diminuído. Isto pode ser conseguido agrupando-se os itens em pacotes. No caso do processamento de imagem de baixo nível, o ideal é transferir uma ou mais linhas de cada vez, reduzindo o tempo necessário para a preparação do canal.

### VI.3 - O USO DO "TRANSPUTER" EM VISÃO COMPUTACIONAL.

#### VI.3.1 - Análise das Alternativas para a Implementação da Transformada de Hough numa Rede de "Transputers".

O sistema lógico proposto para fins de elaboração deste estudo é mostrado na figura VI.2 e é composto pelos seguintes módulos:

- (i) interface gráfica;
- (ii) servidor;
- (iii) aplicação.

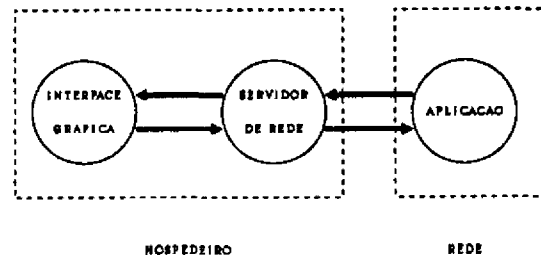


Figura VI.2 - Diagrama básico mostrando as relações entre os programas desenvolvidos para o estudo da Transformada de Hough.

O hospedeiro é um microcomputador IBM-PC AT que se comunica com o processador zero da rede hipercúbica (nó raiz). Atua como um terminal através do qual o usuário interage com o multiprocessador. É responsável pela execução de dois programas: a interface gráfica e o servidor.

A rede hipercúbica é o computador NCP I, onde será executada a aplicação. Nos estágios iniciais do projeto, as placas com 1 e 4 "transputers" serão utilizadas para fins de desenvolvimento e depuração. O sistema operacional usado para fins de desenvolvimento foi o MS-DOS.

#### VI.3.1.1 - Interface Gráfica.

Havia duas possibilidades para a implementação deste módulo: uma usando a placa gráfica Harlequin e outra através de chamadas ao sistema operacional.

Como o objetivo deste trabalho não é avaliar o desempenho do "transputer" em operações de computação gráfica, optou-se pela segunda alternativa.

Este programa foi escrito em "C" e visa o estabelecimento de uma interface amigável entre o usuário e a rede de "transputers". É basicamente um ambiente integrado, composto por várias camadas de menus e que permite ao usuário (figura VI.3):

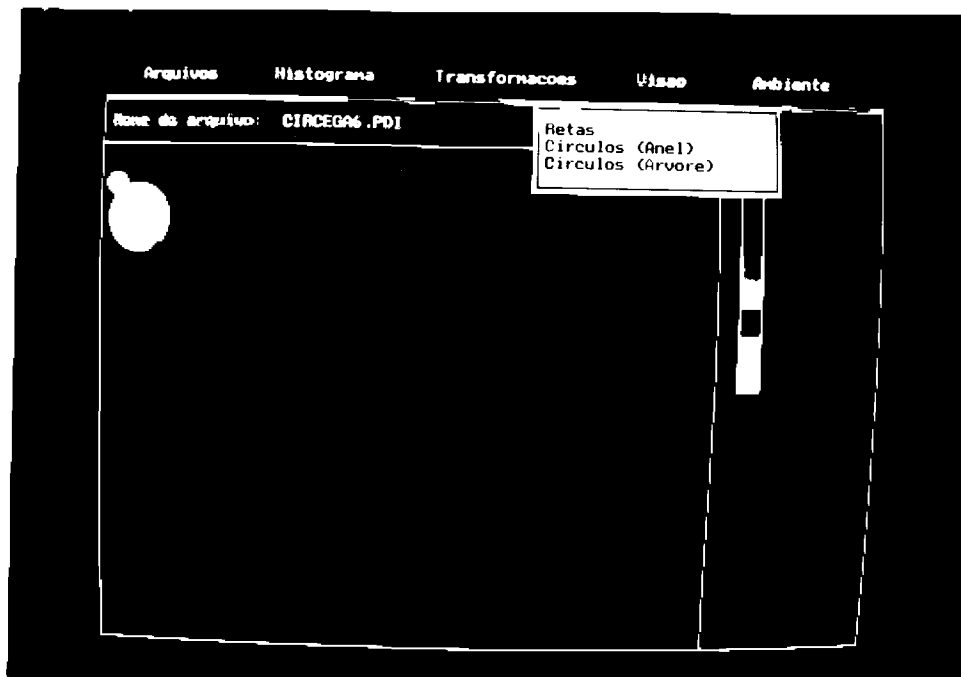


Figura VI.3 - Interface gráfica.

- executar as rotinas de processamento de imagens e visão computacional desenvolvidas para este trabalho (em "C" e OCCAM);
- carregar os programas desenvolvidos em OCCAM na rede, seleccionando o tipo de configuração e a quantidade de "transputers" a ser usada;
- visualizar alguns resultados do processamento, tais como os tempos de processamento, mensagens do servidor, imagens processadas e curvas reconhecidas.

#### VI.3.1.2 - Servidor de Rede.

Os programas compilados gerados pelo TDS2 podem ser de quatro tipos: LIB, SC, PROGRAM e EXE, os quais são descritos a seguir:

LIB : contém uma biblioteca de funções e/ou declarações de variáveis, constantes e protocolos de canais.

SC ("separate compilation unit") : usualmente contém procedimentos e funções e fica contido em "folds" do tipo LIB, PROGRAM ou EXE.

PROGRAM : é um processo cuja execução é feita fora do TDS2 e permite o desenvolvimento de programas autônomos. Também é utilizado para agrupar os comandos que descrevem as interconexões entre processadores ("fold" de configuração) e para atribuir procedimentos aos mesmos.

EXE : é um processo cuja execução é feita dentro do TDS2, facilitando assim o desenvolvimento e depuração de programas.

Toda a comunicação entre o hospedeiro e a rede é realizada através do nó 0. A configuração do multiprocessador hipercúbico é mostrada na figura VI.4.

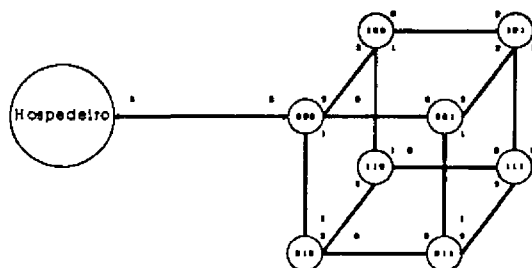


Figura VI.4 - Rede hipercúbica de dimensão 3.

As aplicações desenvolvidas trabalham no modo autônomo, ou seja, as rotinas foram compiladas com o tipo PROGRAM. Desta forma, os programas poderão comunicar-se com a interface gráfica. Porém, a comunicação não é feita diretamente, uma vez que o protocolo da rede deve ser compatibilizado com o do hospedeiro.

O servidor da rede é um programa executável, residente no hospedeiro e escrito em "C", que pode ser chamado pela interface gráfica. O mesmo é uma adaptação da tarefa AFSERVER, cujo código é fornecido com o pacote TDS2 e é executado em paralelo com o programa carregado na rede. Possui as seguintes características:

- carrega na rede os programas executáveis e as configurações gerados em OCCAM;
- é responsável pela troca de informações entre a rede e o hospedeiro;
- permite que a rede faça chamadas ao sistema operacional do hospedeiro (MS-DOS).

#### VI.3.1.3 - Aplicação.

São os programas de processamento de imagens e visão computacional desenvolvidos em OCCAM. Os primeiros são, em geral, de granularidade fina e já foram suficientemente explorados em outros trabalhos [36]. No item VI.2 são mostrados e comentados alguns resultados obtidos para um "transputer". Daqui em diante, serão analisados apenas algoritmos de visão computacional.



No que diz respeito à Transformada de Hough, pode-se adotar uma combinação do paralelismo de tarefas com o de imagem. Os espaços de Hough e de imagem devem ser distribuídos pelos nós de modo a extrair o máximo de concorrência possível. BOWYER e LAKE [37] propuseram as seguintes alternativas para máquinas MIMD:

**a) Imagem Completa/partição do espaço de Hough.**

Cada processador acessa uma cópia local da imagem e uma partição local do espaço de Hough. A quantidade de memória usada é de moderada a alta.

**b) Partição da imagem/espaço de Hough completo.**

Cada processador acessa uma partição local da imagem e incrementa células na sua matriz acumuladora (espaço de Hough local). Uma vez que o espaço de Hough pode ser uma estrutura de dados muito grande, é necessária bastante memória local, especialmente para formas parametrizadas. Os resultados produzidos são parciais. O cálculo completo pode ser obtido de duas formas: os espaços de Hough são difundidos ao longo da rede de "transputers" e acumulados em cada nó ou as partições da imagem são trocadas até que cada nó possua uma cópia completa da matriz acumuladora.

**c) Partição da imagem/partição do espaço de Hough.**

Cada processador varre a sua partição da imagem e atualiza a sua partição do espaço de Hough. Estes resultados são parciais e o cálculo total pode ser obtido difundindo-se as partições da imagem ou do espaço de Hough.

Este trabalho abordará a Transformada de Hough para a detecção de retas e círculos. Os procedimentos de detecção compreendem as seguintes ações:

- incrementação da matriz acumuladora;
- busca de máximos;
- empacotamento e envio dos máximos detectados para o nó raiz, que se encarrega de transmitir os resultados para o hospedeiro.

### VI.3.2 - Detecção de Retas.

A detecção de retas é uma tarefa computacionalmente mais intensiva do que a detecção de contornos através do operador de Sobel, caso seja empregado um procedimento de suavização da matriz acumuladora. O tempo necessário à execução da mesma varia de 2 a 4 vezes o tempo de filtragem, dependendo da complexidade da cena. Levando-se em conta estas considerações, um anel com 4 "transputers" é suficiente para efeito de avaliação do algoritmo de detecção de retas (figura VI.5). Serão usadas imagens com 128 x 256 "pixels" e 8 tons de cinza.

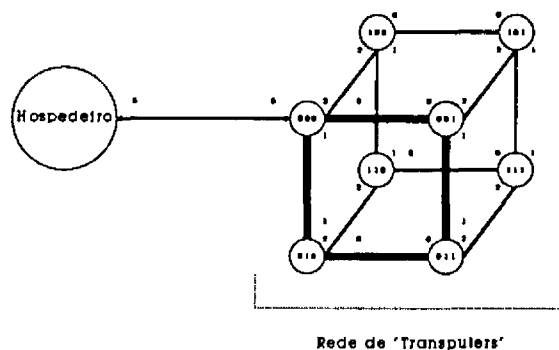


Figura VI.5 - Anel com 4 "transputers" mapeados na rede hiper-cúbica.

**VI.3.2.1 - Alternativas para a Implementação.**

A estrutura lógica mais adequada aos programas distribuídos necessários à detecção de retas consiste de um processo monitor e três processos de aplicação (figura VI.6).

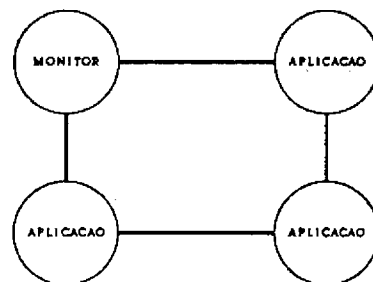


Figura VI.6 - Estrutura lógica básica dos programas para detecção de retas.

O processo monitor desempenha as seguintes funções:

- leitura em disco da imagem;
- distribuição da mesma ao longo do anel;
- coleta e escrita em disco dos máximos detectados;
- escrita em disco dos tempos de processamento.

Os processos de aplicação realizam basicamente duas funções: roteamento para os seus vizinhos e detecção. A sua estrutura é mostrada na figura VI.7.

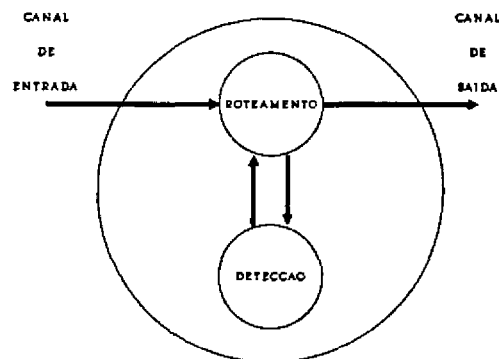


Figura VI.7 - Estrutura lógica básica dos processos de aplicação.

Tendo em vista as granularidades envolvidas, as alternativas mais

... viáveis para a organização dos dados e das tarefas são as discutidas a seguir.

**a) CASO 1 : Extração de contornos distribuída.**

Cada nó trabalha com uma partição da imagem e possui um espaço de Hough completo.

O processo monitor envia linhas da imagem para os nós e estes extraem a partição da mesma que deve ser analisada (figura VI.8.b).

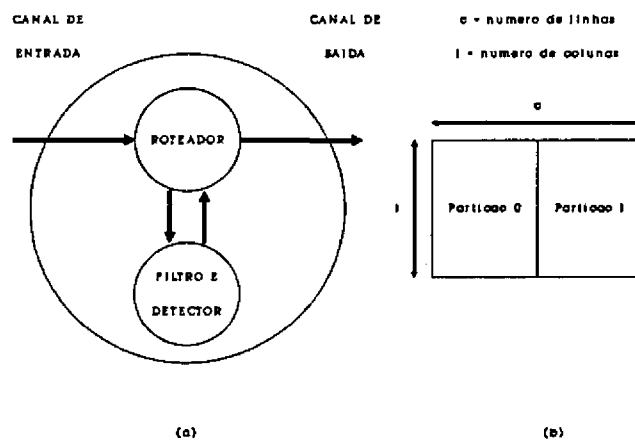


Figura VI.8 - Caso 1.

(a) Estrutura lógica do processo de aplicação.

(b) Divisão da imagem em duas partições.

Os contornos são extraídos localmente e para cada pixel de contorno é

calculada a Transformada de Hough correspondente. Porém, o resultado gerado será parcial e deverá ser somado ao dos outros nós. Para tanto, os espaços de Hough parciais serão difundidos e somados, até que um dos nós (nó 1) contenha o resultado final. Neste nó também serão detectados e empacotados os máximos, para que sejam posteriormente empacotados e enviados ao monitor. A configuração final do multiprocessador hipercúbico para o caso 1 é mostrada na figura VI.9.

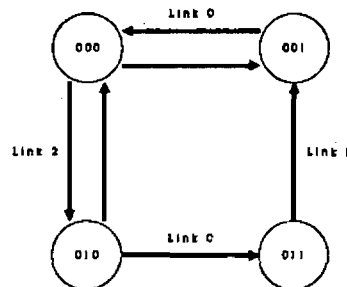


Figura VI.9 - Configuração final da rede para o caso 1.

Nesta situação, trocar matrizes acumuladoras ainda é razoável, se for levado em conta o tamanho da mesma (arbitrou-se uma matriz com 256 x 256 células de 16 bits).

§. **b) CASO 2 : Extração de contornos centralizada.**

CB

Existe um processo responsável pela extração de contornos e envio dos valores dos módulos e das direções dos vetores gradiente nos "pixels" reconhecidos para os processos de aplicação. Este processo pode residir no nó 0, otimizando desta forma a distribuição de carga na rede, uma vez que a transmissão dos contornos extraídos pode se dar concorrentemente com a extração. Para que tal ocorra, basta estabelecer um esquema de rodízio de "buffers".

Em cada nó haverá uma partição da imagem, para a qual serão executados os procedimentos de cálculo da Transformada de Hough, determinação de máximos e empacotamento de parâmetros.

A figura VI.10 mostra a organização dos dados e tarefas. As partições são divididas segundo a direção  $\theta$ .

**c) CASO 3 : Extração de contornos centralizada, com troca de espaços de Hough incompletos.**

Neste situação, o processo monitor também se encarrega de detectar as bordas e empacotar os dados relativos aos vetores gradiente, de maneira análoga ao caso 2. Cada pacote contém o endereço do nó destino. Os espaços de Hough seriam trocados como no caso 1.

Esta estratégia reduz o esforço computacional necessário à filtragem em cada nó de aplicação, porém a seção do programa responsável pelo roteamento

torna-se mais lenta. Tal fato é consequência do aumento da complexidade da mesma (o nó 2 deve reconhecer quais são os seus pacotes e quais pertencem ao nó 3). Logo, pode-se prever que o desempenho do sistema será semelhante ao do caso 1 e, portanto, não valeria a pena implementá-lo.

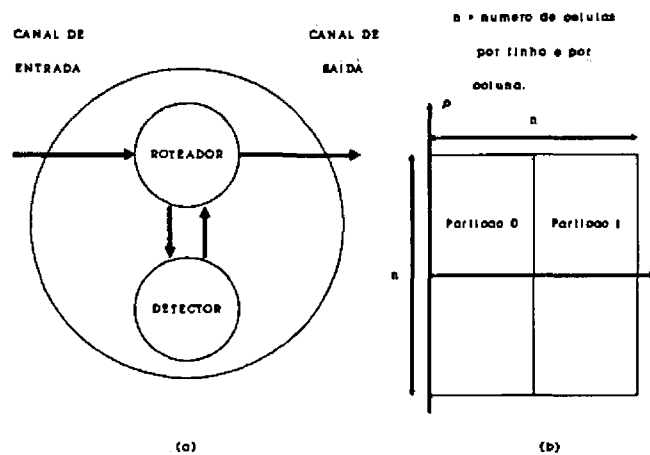


Figura VI.10 - Caso 2.

(a) Estrutura lógica do processo de aplicação.

(b) Divisão do espaço de Hough em duas partições.



### VI.3.2.2 - Discussão dos Resultados.

Um aspecto importante a ser destacado em relação às propostas de implementação mencionadas anteriormente é o fato de que há apenas uma cópia da imagem na rede (processo monitor). Este fato difere das três alternativas sugeridas por BOWYER e LAKE, onde cada nó possui cópia de pelo menos uma parte da imagem, além da matriz acumuladora local. As propostas deste trabalho têm o mérito de reduzirem as quantidades de memória local necessárias em cada nó, uma vez que a leitura dos "pixels" é concorrente com o processamento.

Na figura VI.11.a é mostrada uma imagem que será submetida à detecção de retas e na figura VI.11.b as retas encontradas na mesma. Nas tabelas VI.2 e VI.3 são mostrados os resultados para a detecção de retas (casos 1 e 2). Observando-se as mesmas, conclui-se que a distribuição de dados e tarefas do caso 2 resulta num tempo de execução menor. Isto é válido para qualquer imagem, uma vez que as tarefas estão melhor equilibradas do que na situação anterior.

Tabela VI.2 - Resultados obtidos para o caso 1.

Número de nós	Tempo(s)	"Speed-up"	Eficiência
1	3,43	1	-
2	2,18	1,57	0,79
3	2,35	1,46	0,49

Tabela VI.3 - Resultados obtidos para o caso 2.

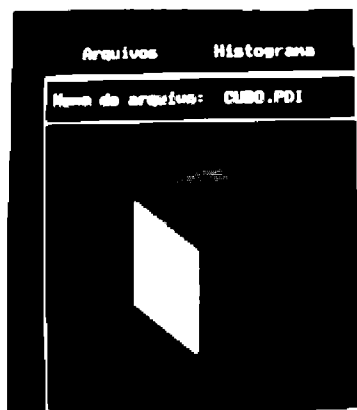
Número de nós	Tempo(s)	"Speed-up"	Eficiência
1	2,08	1	-
2	1,10	1,89	0,95
3	1,05	1,98	0,66

Em relação ao desempenho do caso 1, pode-se observar que o uso de três "transputers" produz um resultado pior do que aquele obtido com dois. Este comportamento é consequência da distribuição de partições de mesmo comprimento pelos os nós da rede. Quando a cena analisada for avaliada por três "transputers", haverá uma sobrecarga do processador responsável pelo cálculo da Transformada de Hough da partição central. Ao contrário, quando são usados dois "transputers", a carga está balanceada.

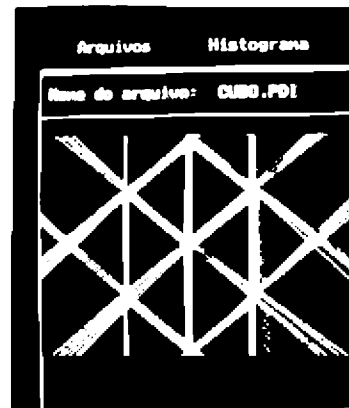
O fato do caso 1 requerer uma comunicação intensiva, para trocar espaços de Hough entre os nós, faz com que os valores de "speed-up" sejam menores do que os conseguidos para o caso 2.

Nas duas situações são necessárias políticas de balanceamento de carga. Uma alternativa para o desequilíbrio apresentado pelo caso 1 seria adotar uma versão modificada do caso 3, onde em lugar de dividir uma linha em partes iguais, seria feita uma distribuição de cargas por demanda. Este procedimento resultaria num balanceamento da carga durante a etapa de cálculo da transformada, já que a quantidade de trabalho ("pixels" pertencentes aos contornos) seria a mesma em todos os nós. Já no segundo caso, a carga pode ser equilibrada com o auxílio de "buffers" de partição.

Cada nó receberia os pixels processados e se houvesse uma grande carga para um nó, parte da mesma seria armazenada para ser processada num momento de menor esforço computacional. De modo geral, a inserção de um terceiro "transputer" não traz muitas vantagens, pois as granularidades envolvidas são pequenas.



(a)



(b)

Figura VI.11 - Imagens usadas para comparação.

(a) Imagem original.

(b) Retas detectadas.

### VI.3.3 - Detecção de Círculos.

A detecção de círculos é uma tarefa cuja granularidade é muito maior do que os procedimentos de extração de contornos e de detecção de retas. Tal fato é consequência da aplicação de uma função de suavização que melhora o espectro da matriz acumuladora tridimensional, após o cálculo da Transformada de Hough (capítulo II, item II.3.3.3). Este procedimento tem por objetivo facilitar a detecção dos pontos de máximo da mesma. Para tornar possível uma comparação com um microcomputador IBM-PC AT, serão usadas imagens com 64 x 64 "pixels" e 8 tons de cinza.

A abordagem utilizada é semelhante à do caso 2 do item VI.3.2.1. O procedimento de extração de contornos ficará mapeado no nó raiz e somente existirá na rede uma cópia da imagem de entrada, a qual ficará residente no mesmo. A rotina Sobel será ligeiramente modificada, de modo que envie também a informação acerca do sentido da variação de nível de cinza no "pixel" de contorno.

A matriz acumuladora será fatiada conforme indicado na figura VI.12, segundo planos paralelos ao XY e igualmente espaçados na direção r. Cada processador será responsável pela detecção de um conjunto de círculos cujos raios estarão contidos no intervalo associado à partição.

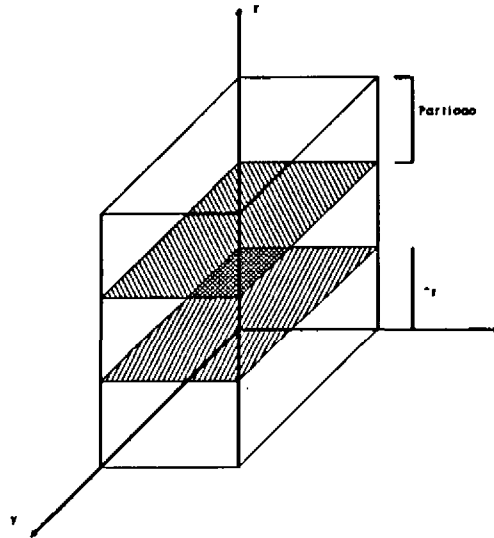
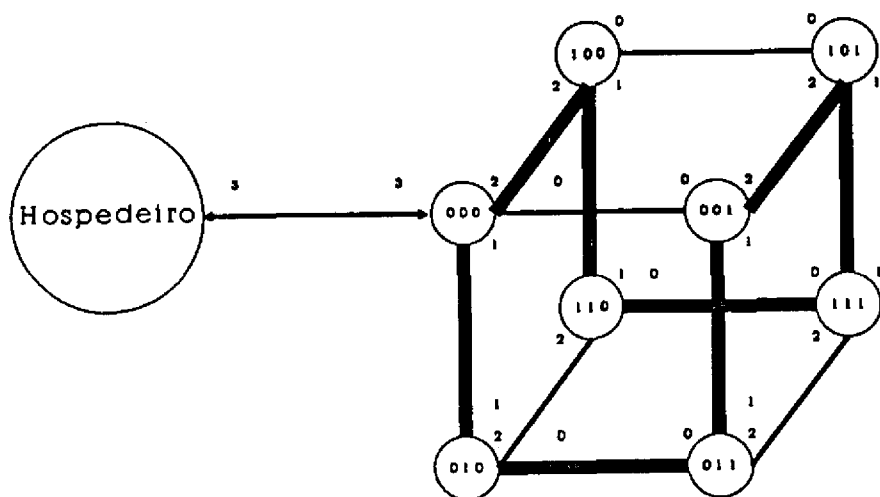


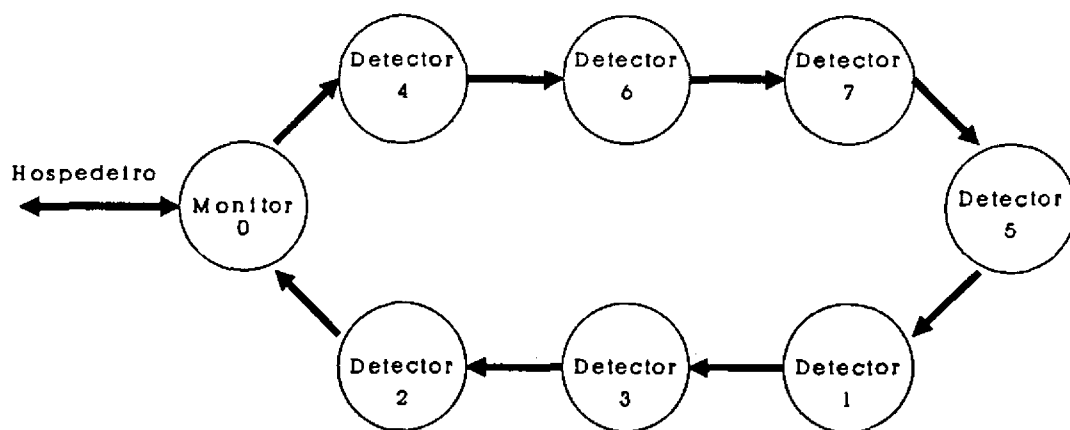
Figura VI.12 - Matriz acumuladora dividida em três partições.

#### VI.3.3.1 - Alternativas para a Implementação.

Serão mapeadas no multiprocessador as topologias anel e árvore (figuras VI.13 e VI.14).

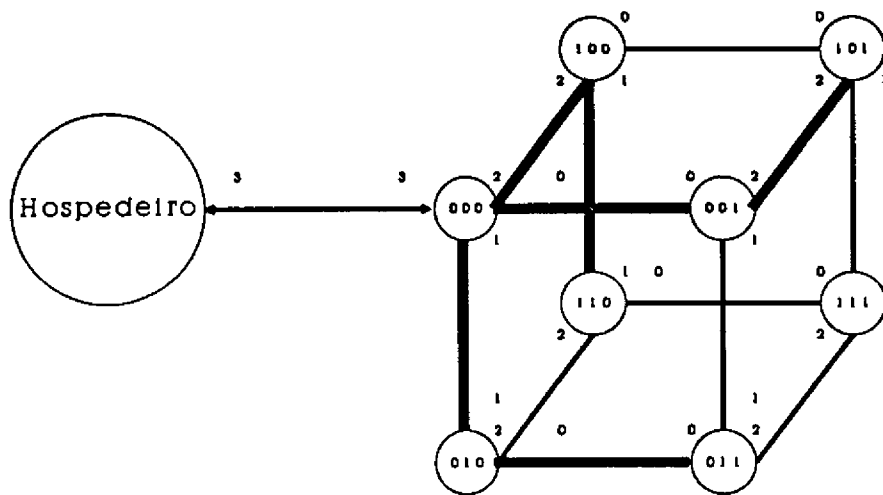


(a) Anel Físico

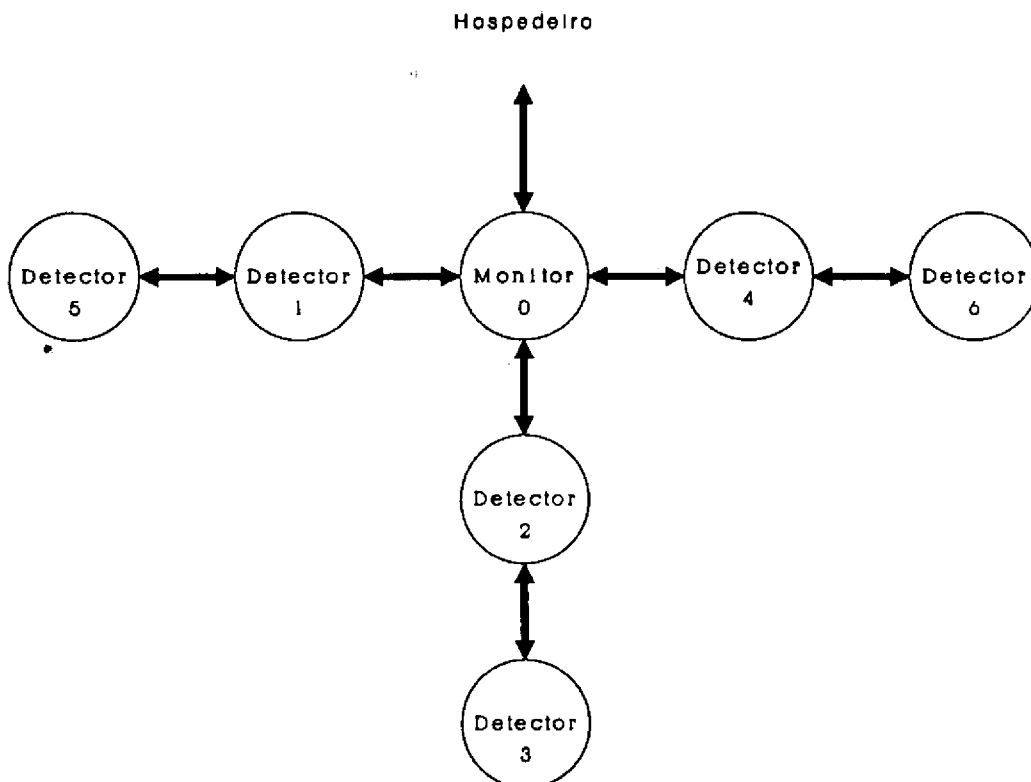


(b) Anel Lógico

Figura VI.13 - Mapeamento de um anel na rede hipercúbica.



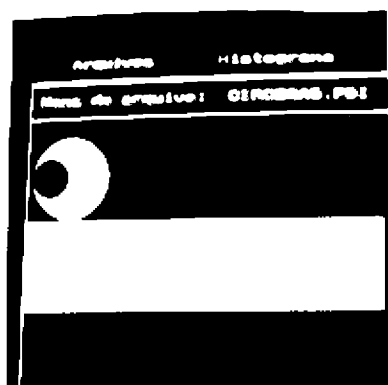
(a) Árvore Física



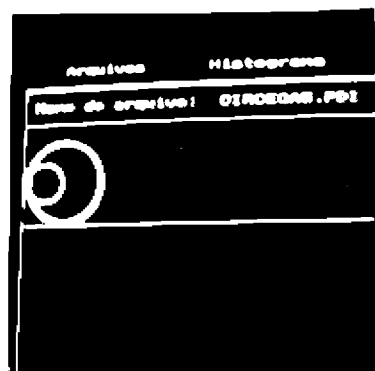
(b) Árvore Lógica

Figura VI.14 - Mapeamento de uma árvore na rede hipercúbica.

O universo pesquisado para os raios é o intervalo  $5 < r < 30$ , o qual será distribuído equitativamente pelos nós. Cada nó ativo aplicará a Transformada de Hough e, dependendo do resultado obtido, atualizará ou não a partição do espaço de paramétrico local. O número de nós ativos é igual à quantidade de processadores escolhida pelo programador. A figura VI.15 mostra a cena que será analisada a seguir. Nas tabelas VI.4 e VI.5 e a figura VI.16 são mostrados os resultados obtidos.



(a)



(b)

Figura VI.15 - Cena utilizada para a confecção das tabelas VI.4 e VI.5.

(a) Imagem original.

(a) Resultado da detecção.



Tabela VI.4 - Desempenho do Anel.

Nº de nós	Tempo(s)	"Speed-up"	Eficiência
1	60,6	1	-
2	35,6	1,7	0,85
3	24,2	2,5	0,84
4	18,9	3,2	0,80
5	16,0	3,8	0,76
6	13,4	4,5	0,75

Tabela VI.5 - Desempenho da Árvore.

Nº de nós	Tempo(s)	"Speed-up"	Eficiência
1	60,6	1	-
2	35,6	1,7	0,85
3	24,1	2,5	0,84
4	18,6	3,3	0,83
5	15,6	3,9	0,78
6	12,9	4,7	0,78

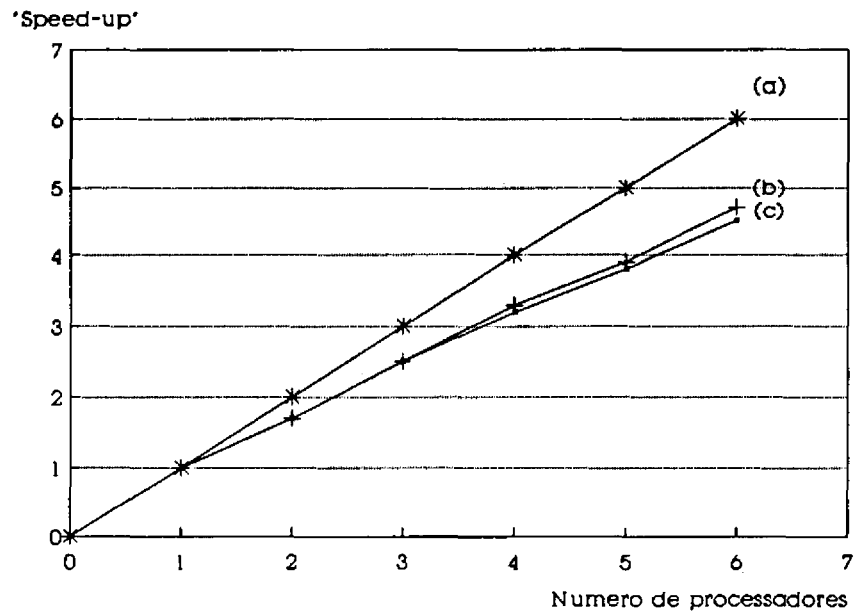


Figura VI.16 - Comparação entre os dados das tabelas VI.4 e VI.5

(a) Situação ideal.

(b) Árvore.

(c) Anel.

Como era esperado, a árvore é um pouco mais rápida que o anel. Esta topologia é mais adequada para processos onde os dados de entrada não são dependentes dos processos vizinhos. A detecção de círculos, conforme implementada, se enquadra neste caso. O anel é uma topologia indicada quando

estão envolvidas tarefas que são naturalmente executadas em sequência, como por exemplo o "pipeline" gráfico da figura V.13. A diferença entre as duas topologias somente se torna perceptível quando há um grande número de nós, pois o peso das operações de roteamento se torna mais evidente. Logo, quando são usados anéis, as tarefas de computação e comunicação devem ser balanceadas com extremo cuidado, para que não haja retenções desnecessárias.

A árvore é mais eficiente do que o anel, mas a medida que o número de processadores aumenta, ambas as topologias sofrem uma degradação de velocidade. Tal fato é consequência de uma distribuição de carga ineficiente, uma vez que os nós encarregados de processador a primeira e a última partições estão submetidos a uma carga menor. Alterando-se a distribuição de raios por partição de modo a tentar corrigir esta distorção, tem-se o resultado da tabela VI.6. A topologia usada foi a árvore.

Tabela VI.6 - Comparação entre as distribuições de carga proposta anteriormente (partições iguais) e a nova (baseada no esforço computacional).

Nº de nós	Tempos(s)		"Speed-up"		Eficiência	
	Anterior	Atual	Anterior	Atual	Anterior	Atual
1	60,6	60,6	1	1	-	-
2	35,6	31,6	1,7	1,9	0,85	0,95
3	24,1	21,7	2,5	2,8	0,84	0,93
4	18,6	16,4	3,3	3,7	0,83	0,93
5	15,6	13,5	3,9	4,5	0,78	0,90
6	12,9	11,3	4,7	5,4	0,78	0,90

As duas estratégias são comparadas graficamente com a situação ideal na figura VI.17.

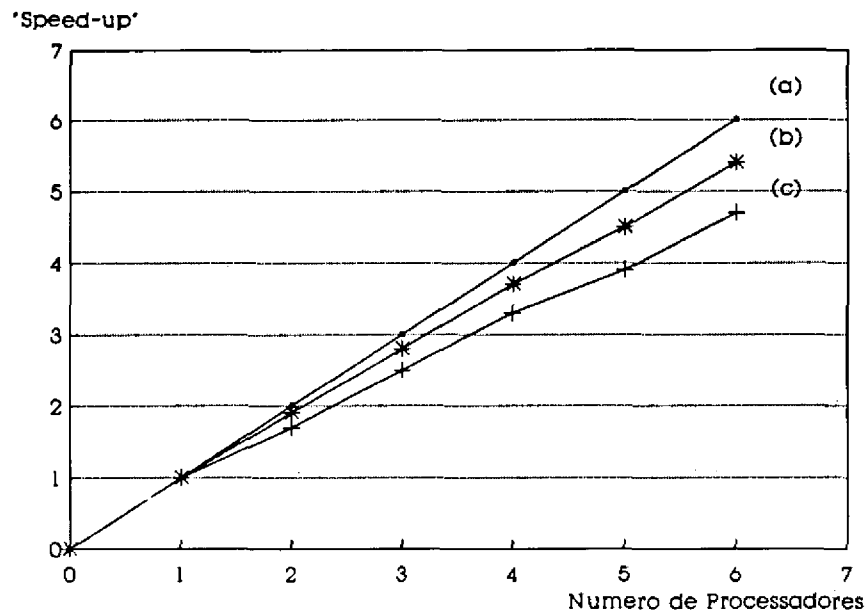


Figura VI.17 - Comparação entre as estratégias da tabela VI.6.

(a) Caso ideal ("Speed-up" linear).

(b) Carga balanceada.

(c) Carga desequilibrada.

## CAPÍTULO VII. CONCLUSÕES.

### VII.1 - CONSIDERAÇÕES RELATIVAS AO SISTEMA LÓGICO.

O conjunto interface gráfica-servidor funciona de modo ineficiente. A maior parte do tempo decorrido entre a chamada de uma rotina OCCAM e o aparecimento do resultado final na tela é gasto com acessos ao disco rígido e à memória. Tal comportamento é indesejável em sistemas de reconhecimento de padrões que porventura precisem operar em tempo real.

A alternativa mais eficiente neste caso seria a utilização de um "transputer" dedicado às funções gráficas. A memória de vídeo seria acessada diretamente através de chamadas a um servidor instalado no hospedeiro.

Esta implementação resultaria numa interface bastante rápida, já que o "transputer" tem apresentado um bom desempenho em aplicações gráficas.

Outro aspecto que poderia ser explorado é o desenvolvimento de estratégias de configuração da rede por intermédio de recursos gráficos. Estas ferramentas seriam muito úteis, pois facilitariam e automatizariam os processos de configuração e atribuição dos canais.

### VII.2 - LIMITAÇÕES DOS PROGRAMAS DE APLICAÇÃO.

De uma forma geral, o desempenho apresentado pela rede de "transputers" foi

muito bom, mas a comunicação entre o nós mostrou-se um ponto crítico. Para algoritmos de granularidade um pouco mais fina, como é o caso da detecção de retas, a comunicação pode impor limitações que algumas vezes são maiores do que as impostas pelo processamento [38,39].

Estruturando-se o processo detector de retas conforme é visto no EXEMPLO 1, haverá uma degradação acentuada do desempenho. O "speed-up" máximo obtido com este tipo de implementação foi 1,14.

#### EXEMPLO 1:

```
PROC exemplo1(CHAN OF PROTOCOLO entrada, saida)
  ... Declarações das constantes e variáveis
  WHILE condicao
    SEQ
      entrada ? dados
      aplicacao(dados)
      saida ! dados
```

#### EXEMPLO 2:

```
PROC exemplo2(CHAN OF PROTOCOLO entrada, saida)
  ... Declarações de constantes e variáveis
  ... PROC leitura
  ... PROC escrita
  ... PROC aplicacao
  SEQ
    leitura(entrada, buffer1)
```

```

PAR
    leitura(entrada, buffer2)
    aplicacao(buffer1)
WHILE condicao
    SEQ
        PAR
            leitura(entrada, buffer3)
            aplicacao(buffer2)
            escrita(saida, buffer1)
        PAR
            leitura(entrada, buffer1)
            aplicacao(buffer3)
            escrita(saida, buffer2)
        PAR
            leitura(entrada, buffer1)
            aplicacao(buffer1)
            escrita(saida, buffer3)

```

A eficiência do processamento poderá ser sensivelmente aumentada se for levado em conta o fato de que as interfaces de comunicação do "transputer" são unidades de DMA autônomas. Esta característica pode ser explorada por estratégias baseadas no fluxo de dados na rede. Os resultados apresentados no capítulo anterior, para o caso da detecção de retas, foram obtidos com rotinas que usam esta abordagem. O EXEMPLO 2 mostra como o EXEMPLO 1 pode ser reescrito seguindo esta filosofia. As ações de entrada, saída e processamento são realizadas concorrentemente, o que reduz a latência da

comunicação.

Processadores de comunicação têm sido pesquisados para uso em sistemas distribuídos, visando a redução do custo da comunicação [39]. O objetivo destes dispositivos é separar as funções de processamento daquelas relativas à comunicação. O roteamento e as instruções de propagação de mensagens são exemplos típicos de tarefas que podem ser executadas por estes elementos.

Outra questão relevante é a distribuição de carga. Na detecção de círculos, a comunicação não afeta tanto o desempenho, em virtude da granularidade do processamento envolvido. Já a forma como é feita a distribuição das partições de imagem e/ou do espaço de Hough tem implicações diretas sobre a eficiência da computação, conforme pôde ser constatado no capítulo VI.

Uma característica desejável, mas que não está disponível no TDS2, é a alocação dinâmica de memória. Este mecanismo facilitaria o desenvolvimento de algumas rotinas.

### VII.3 - VANTAGENS DA ARQUITETURA UTILIZADA.

A seguir são citadas algumas características que tornam o uso de multiprocessadores baseados em "transputers" nas áreas de processamento de imagens e visão computacional bastante promissor:

- custo moderado;
- a família "transputer" é versátil e poderosa;



- as arquiteturas apresentam grande modularidade;
- permite a implementação de mecanismos síncronos e assíncronos de comunicação por troca de mensagens;
- a computação realizada pelos nós é de granularidade média, ou seja, a relação entre o tempo de processamento e a soma dos tempos de comunicação é inferior a dos sistemas de memória compartilhada e superior a de máquinas SIMD.

Como consequência das características mencionadas acima, estas máquinas podem ser consideradas computadores paralelos de propósito geral.

## BIBLIOGRAFIA

- [1] Mascarenhas, N.D.A. ; Velasco, F.R.D. - "Processamento Digital de Imagens", IV Escola Brasileiro-Argentina de Informática, Universidad Catolica de Santiago del Estero, Argentina, Janeiro, 1989.
- [2] Dawson, B. M. - "Introduction to Image Processing Algorithms", BYTE, March, 1987.
- [3] Dawson, B.M.; Spalding, M. - "Finding the Titanic", BYTE, March, 1986.
- [4] Maresca, M. & Lavin, M.A. - "Parallel Architectures for Vision", Proceedings of the IEEE/Special Issue on Computer Graphics, Vol. 76, N° 8, August, 1988, pp. 970-981.
- [5] L. Uhr - "Layered Recognition Cone Networks that Preprocess, Classify and Describe", IEEE Transactions on Computers, Vol. C-21, 1985, pp. 758-768.
- [6] Brady, M. - "Computacional Approach to Image Understanding", ACM Computing Surveys, Vol. 14, pp. 3-70, 1982.
- [7] Duda, R.O. & Hart, P.E. - "Use of the Hough Transformation to Detect Lines and Curves in Pictures", Comm. ACM, N° 15, 1972, pp. 11-15.
- [8] Rosenfeld, A. et alli - "Hough Transform Algorithms for Mesh-Connect SIMD Parallel Processors", Computer Vision, Graphics and Image Processing, N° 41, 1988, pp. 293-305.

- [9] Ballard, D.H. - "Generalizing the Hough Transform to Detect Arbitrary Shapes", Pattern Recognition, Vol. 13, N° 2, 1981, pp. 111-122.
- [10] Danielsson, P.E. - "Vices and Virtues of Image Parallel Machines", Digital Image Analysis, Pitman Publishing Inc., Great Britain, 1984, pp.47-59.
- [11] Cantoni, V. & Levialdi, S. - "Matching the Task to an IP Architecture", Proc. VI Int. Conf. on Pattern Recognition, Munich, 1982, pp. 254-257.
- [12] Danielsson, P.E. - "What Computer Architecture?", Digital Image Analysis, Pitman Publishing Inc., Great Britain, 1984, pp.47-59.
- [13] Hwang, K. & Briggs, F. A. - "Computer Architecture and Parallel Processing", McGraw-Hill Book Company, 1987.
- [14] Siegel, H.J. et alli - "PASM: A Partionable SIMD/MIMD System for Image Processing and Pattern Recognition", IEEE Transactions on Computers, vol. c.30, No. 12, December, 1981.
- [15] Gallant, J. - "Parallel Processing Ushers in a Revolution on Computing", EDN, September, 1988, pp. 86-100.
- [16] Basille, I. L. & Castan, S. - "The M.I.M.D. Level of the SY.MP.A.T.I. Simulation and Performances Expected", Digital Image Analysis, Pitman Publishing Inc., Great Britain, 1984, p. 239-250.
- [17] Stefanelli, R. - "VLSI Components and Fault-Tolerance Methodologies for Image Processors", Digital Image Analysis, Pitman Publishing Inc., Great Britain, 1984, pp. 225-228.

- [18] Kruatrachue, B. & Lewis, T. - "Grain Size Determination for Parallel Processing", IEEE Software, pp. 23-32.
- [19] Andrews, G.R. - "Distributed Programming Languages", Procedures of ACM'82 Conference, Dallas, Texas, ACM, oct. 1982, p. 113-117.
- [20] Kirner, C. & Mendes, S.B.T. - "Sistemas Operacionais Distribuídos - Aspectos Gerais de sua Estrutura", Ed. Campus Ltda., 1988, p. 37-44.
- [21] Staunstrup, J. - "Message Passing Communication Versus Procedure Call Communication", Software-Practice and Experience, Vol. 12, N° 3, March, 1982, p. 223-234.
- [22] Hoare, C.A.R. - "Communicating Sequential Process", Communications of the ACM, VOL. 21, N° 8, August, 1978, p. 666-677.
- [23] Brinch Hansen, P. - "Distributed Processes: A Concurrent Programming Concept", Communications of the ACM, Vol. 21, N° 11, November, 1978, p. 934-941.
- [24] Whiddett, D. - "Distributed Programs: an Overview of Implementations", Microprocessors and Microsystems, Vol. 10, N° 9, November, 1986.
- [25] INMOS Limited - "The Transputer Family", Bristol, June, 1986, pp. 4-13.
- [26] INMOS Limited - "Transputer Architecture", Bristol, June, 1986.
- [27] INMOS Limited - "IMS T800 Transputer - Preliminary", Bristol, March,

1988, pp. 1-51.

- [28] INMOS Limited - "The Transputer Instruction Set - A Compiler Writer's Guide", Bristol, February, 1987.
- [29] Stein, R. M. - "T800 and Counting", BYTE, November, 1988, pp. 287-296.
- [30] INMOS Limited - "Lies, Damned Lies and Benchmarks", in Technical note 13, Bristol, 1987.
- [31] Wexler, J. e Prior, D. - "Solving Problems with Transputers: Background and Experience", Microprocessors and Microsystems, vol. 13, N<sup>o</sup> 2, March, 1989, pp. 67-78.
- [32] Wayman, R. - "OCCAM2 : an Overview from a Software Engineering Perspective", Microprocessors and Microsystems, vol. 11, N<sup>o</sup> 8, October, 1987, pp. 413-422.
- [33] Pountain, D. - "A Tutorial Introduction to OCCAM Programming", INMOS Limited, 1986, pp. 49-58.
- [34] Maresca, M. & Lavin, M.A. - "Parallel Architectures for Vision", Proceedings of the IEEE/Special Issue on Computer Graphics, Vol. 76, N<sup>o</sup> 8, August, 1988, pp. 970-981.
- [35] INMOS Limited - "Performance Maximisation", in Technical Note 14, Bristol, 1987.
- [36] Mendes, C. L. - "Arquitetura Paralela para Processamento de Imagens",

tese de mestrado, ITA, São José dos Campos, 1988.

- [37] Bowyer, K. e Lake, C. - "Computing the Hough Transform on MIMD Parallel Architectures (extended abstract)", University of South Florida, 1989, pp. 1-11.
  
- [38] INMOS Limited - "Performance Maximisation", Technical Note 14, Bristol, 1987.
  
- [39] Drummond, L.M.A. - "Projeto e Implementação de um Processador Virtual de Comunicação", dissertação de mestrado, COPPE/Sistemas, UFRJ, Rio de Janeiro, 1990.
  
- [40] Fu, K.S.; Gonzalez, R.C. e Lee, C.S.G. - "Robotics: Control, Sensing, Vision and Intelligence", McGraw-Hill, Singapore, 1987.

## APÊNDICE I

### O COMPUTADOR NCP I.

O NCP I é uma máquina MIMD que usa comunicação por troca de mensagens. Seus nós são idênticos e baseiam-se no T-800. Um microcomputador IBM-PC AT é utilizado como hospedeiro (figura A.1).

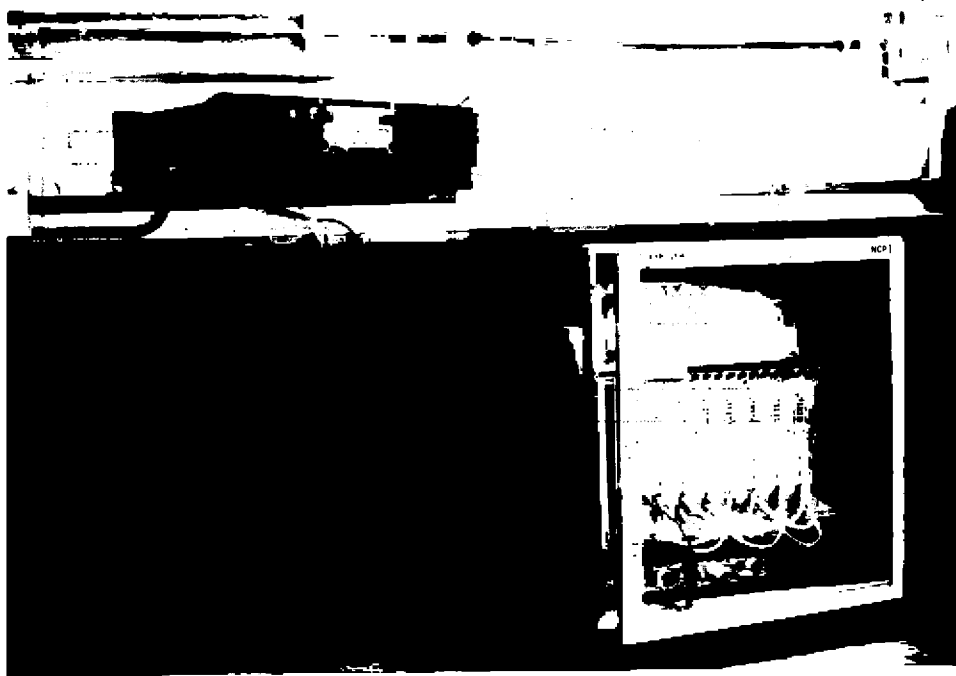


Figura A.1 - Sistema NCP I.

Um nó deste multiprocessador, conforme concebido, pode ser visto na figura A.2. Há basicamente dois módulos: um dedicado ao processamento e à comunicação e outro destinado a operações vetoriais. Estes módulos são geridos respectivamente pelo T800 e pelo i86C, que por sua vez compartilham uma certa quantidade de memória.

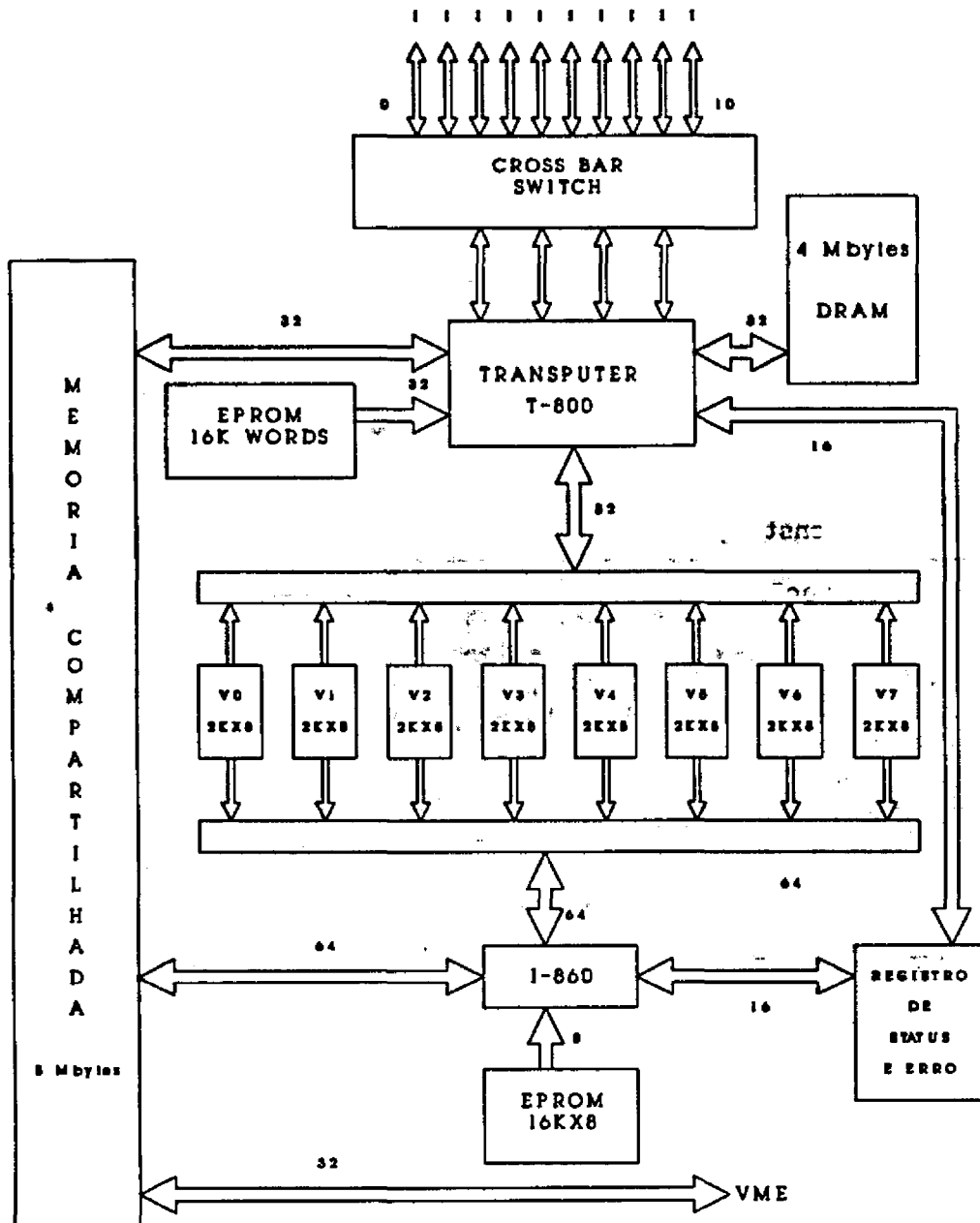


Figura A2 - Arquitetura de um nó do NCP I.

Atualmente o NCP I está configurado como um hipercubo de dimensão três e o módulo vetorial ainda não foi instalado. Esta máquina pode utilizar os



sistemas operacionais DOS e HELIOS (UNIX).

O T800 acessa 16 Kwords de memória EPROM, onde residem rotinas para o tratamento de interrupções, erros internos e diagnóstico, além de informações relacionadas com a configuração do sistema. O T800 também pode acessar 4 Mbytes de memória dinâmica, na qual serão carregados os programas a serem executados pelo mesmo.

Um "transputer" possui apenas quatro "links", o que a princípio somente viabilizaria a construção de um cubo com dimensão máxima igual a quatro (16 nós). Esta limitação pôde ser superada incluindo-se no projeto do nó uma chave "crossbar", que além de interligar os nós, permite a reconfiguração física da rede. Logo, este dispositivo possibilita a utilização de diversos tipos de topologia, numa mesma máquina.

O i860 é um processador vetorial de 64 bits, com arquitetura RISC, fabricado pela INTEL. O seu desempenho é bastante alto graças à operação concorrente de suas várias unidades: núcleo, somador de ponto flutuante e multiplicador de ponto flutuante. Uma soma, uma multiplicação e uma instrução inteira podem ser realizadas simultaneamente a cada ciclo de relógio. Os programas desenvolvidos para o i860 podem alternar os modos vetorial (com "pipeline") e escalar (sem "pipeline") com facilidade. Existem duas caches internas, uma para dados e outra para instruções, que proporcionam uma largura de banda de 960 Mbytes.