UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Reconfiguração no T-NODE em caso de falhas

por

Raul Ceretta Nunes

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Philippe Olivier Alexandre Navaux
Orientador

Prof. Ingrid Jansch Pôrto
Co-orientadora

Porto Alegre, setembro de 1993

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Ceretta Nunes, Raul

Reconfiguração no T-NODE em caso de falhas / Raul Ceretta Nunes. - Porto Alegre: CPGCC da UFRGS, 1993.

116 p. : il.

Dissertação (mestrado)—Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1993. Orientador: Navaux, Philippe Olivier Alexandre; Co-orientador: Pôrto, Ingrid Jansch

Dissertação: Tolerância a Falhas, Processamento Paralelo Reconfiguração, Transputer, T-node, Arquiteturas Paralelas e Tolerância a Falhas

AGRADECIMENTOS

O trabalho desenvolvido neste texto, fruto de mais de dois anos e meio de estudo, dedicação, conversas, alegrias e preocupações, marca uma fase inesquecível de minha vida. Nesta fase salientaram-se os laços de amizade e de trabalho. Assim, quero através de poucas mas sinceras palavras agradecer:

- a minha esposa Viviany pelo apoio e compreensão com os quais sempre pude contar nos momentos difíceis;
- à minha filha Rauani que mesmo à quilômetros de distância sempre me preserva em seu coração como um grande pai, e na certeza que breve vamos estar juntos;
- aos meus pais Inocêncio e Irma pelo impagável apoio sem o qual chegar onde estou seria muito difícil, para não dizer impossível;
- a minha sogra Anilza pela grande ajuda que sempre pude contar;
- aos meus orientadores que, sempre com muita vontade e dedicação, procuraram fazer críticas e sugestões enriquecedoras ao trabalho;
- aos meus amigos e colegas: Adenauer, Otilia, De Rose, Soraia, Krug, Renata, Rafael, Zancanella e Gerson, pelas conversas e brincadeiras sempre acompanhadas de de uma boa dose de humor e simpatia, e em especial aos amigos Sérgio, Benedito e Cláudia, com os quais tive a feliz oportunidade de compartilhar prazeres, angústias, anseios, alegrias e até mesmo teto, no qual sempre pude sentir-me em casa;
- a todos os colegas do CPGCC que de uma forma ou de outra sempre estiverão presentes no espaço de meus sentimentos e lembranças;
- aos meus velhos e sempre amigos Roberto, Hanelore, Luis Cláudio e Luciana, pelas alegrias que sempre me trouxeram;

- em especial a minha querida tia Carlota pela acolhida incansável e carinhosa em sua casa;

- a CAPES, pelo apoio financeiro.

SUMÁRIO

GLOSSÁRIO

Broadcast ..... Envio de uma mensagem para todos os possíveis recepientes;

CPU ..... *Central Processing Unit*. Unidade de processamento de números inteiros;

CSP ..... *Communicating Sequential Processes*. Linguagem desenvolvida por Hoare para especificação de sistemas concorrentes;

flops ..... *floating-point operations per second*. Operações de ponto flutuante por segundo; usada como medida de desempenho na avaliação de sistemas de computação;

FPU ..... *Floating Point Unit*. Unidade de processamento de números reais;

HELIOS ..... Sistema operacional para máquinas *transputer* desenvolvido pela Perihelion Software;

IEEE754 ..... Padrão de unidades de ponto flutuante estabelecido pela IEEE;

Link ..... Conexão "um para um" entre dois processadores;

MIMD ..... *Multiple Instruction Stream and Multiple Data Stream*;

Mips ..... *Million Instruction per Second*. Milhões de instruções por segundo; usada como medida de desempenho na avaliação de sistemas de computação;

MISD ..... *Multiple Instruction Stream and Single Data Stream*;

OCCAM ..... Linguagem de alto nível baseada na linguagem CSP. Dispõe de mecanismos intrínsecos para a sincronização de processos concorrentes;

RAM ..... *Random Access Memory*. Memória de leitura/escrita;

RISC ..... *Reduced Instruction Set Computer*. Computador que tem somente um conjunto de instruções simples, mas que as executa de forma extremamentemente rápida;

ROM ..... *Read Only Memory*. Memória não volátil somente de leitura;

RSRE ..... *Royal Signal and Radar Etablishment*;

SIMD ..... *Single Instruction Stream and Multiple Data Stream*;

SISD ..... *Single Instruction Stream and Single Data Stream*;

Swap-plugs.... *Jumps* para estabelecimento ou não de conexões físicas entre conjuntos de chaves localizados em placas diferentes;

TDS..... *Transputer Development System*. Sistema para o desenvolvimento de software em transputers. Inclui editor de textos baseado em dobras (*folds*), compilador OCCAM, editor de ligação, depurador, carregador e diversos outros programas utilitários;

TELMAT..... Fabricante de supercomputadores;

timeout..... Tempo máximo permitido para espera de alguma entrada associada;

T-NODE..... Supercomputador produzido pela TELMAT;

Transputer..... Processador fabricado pela INMOS. Possui características que o torna indicado como elemento básico na implementação de máquinas paralelas;

T800..... Processador *transputer*. Caracteriza-se principalmente por possuir fpu;

VLSI..... *Very Large Scale Integration*. Denominação dada para *chips* com muito alta escala de integração.

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Procedimentos de reconfiguração são usados em diversos sistemas para isolar módulos falhos e recuperar o sistema após a ocorrência de erros. Em ambientes multiprocessadores, onde existe redundância implicita de nodos processadores, vários algoritmos de reconfiguração já foram propostos. Entretanto a maior parte destes algoritmos destina-se a topologias específicas bastante exploradas como, por exemplo, arquiteturas na forma de *arrays* e árvores.

Neste trabalho é apresentada uma estratégia de detecção / reconfiguração para tolerar falhas na máquina T-NODE. Esta máquina possui uma arquitetura multiprocessadora fracamente acoplada, que tem como processador base o *transputer*. Sua arquitetura de interconexão é definida pelo usuário; a organização de barramentos implementada com base em uma chave *crossbar*, a qual permite uma variada e fácil gama de opções. Assim, os algoritmos tradicionais de reconfiguração não se aplicam pois são excessivamente restritivos. A análise da arquitetura e do *software* de baixo nível existentes para a T-NODE revelou recursos praticamente inexistentes a nível de controle de falhas nos processadores e erros no processamento. Mesmo considerando-se que o principal objetivo desta máquina é a obtenção de alto desempenho, é possível implementar procedimentos que melhorem suas características de confiabilidade. Neste estudo é apresentada uma maneira de melhorar o nível de tolerância a falhas da máquina de modo que ela possa ser usada em tarefas mais exigentes do ponto de vista de confiabilidade, sem perda excessiva de desempenho.

A estratégia definda usa a técnica de redundância dinâmica com detecção de falhas *on-line* e recuperação do sistema através do isolamento da falha por reconfiguração e conseqüente reinicialização do sistema.

A validação da estratégia foi feita pela construção de um protótipo utilizando a linguagem OCCAM2 e um processador *transputer* conectado ao bar-

ramento de um microcomputador PC. No protótipo foram implementados três processos distintos: o testador, o supervisor e o reconfigurador. Estes processos têm respectivamente, as funções de testar os nodos processadores, supervisionar os resultados dos testes e reconfigurar o sistema quando da ocorrência de uma falha.

PALAVRAS-CHAVE: Reconfiguração, *Transputer*, T-NODE, Arquiteturas Paralelas e Tolerância a Falhas.

TITLE: "RECONFIGURATION ON THE T-NODE MACHINE UNDER FAULT"

ABSTRACT

In many systems, reconfiguration strategies are used to remove failed components and to recuperate system from the resulting errors. Various reconfiguration algorithms have been proposed with the goal of covering faults in multiprocessing systems, but most of them support only specific architecture styles, as arrays or trees.

In this study, a reconfiguration algorithm is proposed whose goal is to tolerate faults in the T-NODE machine. The T-NODE is a loosed coupled, multiprocessor machine based on transputers. The analysis of the architecture and of the system software existing for the T-NODE has shown that, in practice, there were not special resources aiming to control processor faults and processing errors. Even considering that the main goal of this machine is processing with high performance, it is possible to implement alternative procedures which result in better reliability characteristics. By other way, the interconnection architecture of this machine is defined by the user; its bus organization implemented with the aid of a crossbar switch allows choices among several possibilities. Consequently, traditional algorithms do not apply because they are too restrictive.

Therefore, the research here related aims to improve the fault-tolerance parameters of this machine without changing significantly its original performance.

The strategy here presented uses a dynamic redundancy technique with on-line fault detection; system recovery is get by logically isolating the faulty module, reconfiguring the others and restarting the system.

The validation of the strategy has been done with the construction of a prototype using the OCCAM2 language and a transputer processor connected to the bus of a microcomputer (PC). Three different processes have been implemented in the prototype: the tester, the supervisior and the reconfigurator. These processes have respectively the functions of: testing the processing nodes, to supervise tests results and to reconfigure the system under fault occurrence.

KEYWORDS: Reconfiguration, Transputer, T-NODE, Parallel Architecture and Fault Tolerance.

1 INTRODUÇÃO

Desde que Von Neumann estabeleceu seu modelo de arquitetura, grande parte dos computadores digitais tem sido projetados de uma maneira fundamentalmente similar, ou seja, possuem um processador, conectado a uma memória, que pode armazenar instruções e dados. Neste modelo, a execução do programa é estritamente seqüencial.

Para muitas aplicações, como por exemplo aplicações comerciais, este tipo de construção pode ser satisfatório; entretanto, em aplicações que possuem elevado volume de dados e/ou que exijam um alto desempenho do sistema, ele se torna insatisfatório.

Essa necessidade de alto desempenho levou vários pesquisadores a estudarem e projetarem máquinas que fazem uso de paralelismo. Assim, várias máquinas com topologias distintas, envolvendo vários processadores e memórias, tornaram-se usuais.

Michael J. Flynn [FLY66], David B. Skillicorn [SKI88] e Ralph Duncan [DUN90], propuseram taxonomias para classificar estas novas máquinas. Entretanto, a classificação mais conhecida é a de Flynn, que classificou as arquiteturas paralelas quanto ao controle, no mesmo intervalo de tempo, tanto do fluxo de instruções como do fluxo de dados.

Sua classificação divide as arquiteturas de computadores em quatro grandes grupos:

- SISD (*Single Instruction Stream and Single Data Stream*): neste grupo, a unidade de controle controla tanto o fluxo de instruções como o fluxo de dados. A clássica arquitetura de Von Neumann pertence a este grupo;

- SIMD (*Single Instruction Stream and Multiple Data Stream*): todos os processadores recebem o mesmo conjunto de instruções, mas executam-nas sobre dados diferentes, no mesmo intervalo de tempo. Processadores matriciais são exemplos de arquiteturas deste grupo;

- MISD (*Multiple Instruction Stream and Single Data Stream*): cada processador recebe diferentes conjuntos de instruções que atuam sobre um único conjunto de dados no mesmo intervalo de tempo;

- MIMD (*Multiple Instruction Stream and Multiple Data Stream*): cada processador da máquina recebe diferentes conjuntos de instruções e dados e operam sobre estes no mesmo intervalo de tempo. Neste grupo, encontram-se as arquiteturas fortemente acopladas (com memória comum) ou fracamente acopladas (com memória distribuída).

A arquitetura alvo deste trabalho, a máquina T-NODE[1] (que será descrita no capítulo 2), caracteriza-se por ser altamente paralela, reconfigurável, fracamente acoplada e baseada no processador *transputer*. Segundo a classificação de Flynn, ela pertence aos sistemas MIMD, embora sua característica de reconfigurabilidade também permita sua operação de maneira similar aos sistemas SIMD ou MISD. A reconfigurabilidade da T-NODE é conseguida com o uso de uma rede de interconexão programável entre os diversos elementos processadores.

Segundo [CLO53], uma rede de interconexão é um dispositivo que estabelece as ligações entre um conjunto de entradas e um conjunto de saídas, e que, de acordo com as características de estabelecimento destas conexões, podem ser classificadas como rearranjável e/ou não bloqueante. Uma rede é **rearranjável** se, ao reorganizar-se a rede, todas as possíveis conexões entre as entradas e as saídas são factíveis. Ela é **não bloqueante** se qualquer conexão entre duas portas

[1] T-NODE é a denominação comercial da máquina multiprocessadora fabricada pela empresa francesa Telmat Informatique.

livres (uma de entrada e uma de saída) pode ser estabelecida sem que se interfira em outra conexão pré-existente.

1.1 Arquiteturas Paralelas e Tolerância a Falhas

Concomitantemente ao desenvolvimento das arquiteturas paralelas para obtenção de maior desempenho, existe também a preocupação com a tolerância a falhas dos sistemas de computação. Esta preocupação deve-se ao crescente uso de computadores nos mais diversos ramos de atividades, dentre os quais encontram-se atividades que estabelecem parâmetros de disponibilidade e confiabilidade bastante rígidos, fazendo com que os computadores que controlam algumas atividades sejam mais confiáveis.

A terminologia empregada neste volume usa as traduções "falha, erro e defeito" para os termos "*fault*, *error* e *failure*" definidos em comum pelo comitê 10.4 da IFIP (designado *Reliable Computing and Fault-Tolerance*) e pelo grupo de terminologia do IEEE (em *Fault-Tolerant Computing*) e apresentados por Laprie em [LAP85].

Em se tratando de tolerância a falhas, um componente é considerado defeituoso somente se ele apresentar um comportamento diferente do especificado. Schneider [SCH90] considera duas classes representativas de comportamento defeituoso: defeitos bizantinos, onde o componente pode apresentar um comportamento arbitrário e malicioso, gerando mensagens que podem confundir outros componentes do sistema; ou defeitos *fail-stop*, onde em resposta a um defeito o componente muda para um estado que permite com que outros componentes detectem o defeito e parem o sistema.

Embora em sistemas multiprocessadores e em sistemas distribuídos possam ocorrer defeitos bizantinos, para muitas aplicações é suficiente assumir

defeitos *fail-stop* para se conseguir manter um bom nível de tolerância a falhas no sistema.

Um sistema multiprocessador é t-tolerante a falhas se ele satisfaz as especificações de suportar t defeitos em um dado intervalo de interesse[2]. Tradicionalmente, tolerância a falhas tem sido especificada em termos de MTBF (*Mean Time Between Failures*), probabilidade da manifestação de um defeito em um dado intervalo de tempo, ou através de medidas percentuais de confiabilidade dos componentes ou do sistema como um todo. Entretanto, para os usuários, a descrição da tolerância a falhas do sistema em termos do número máximo de componentes defeituosos que pode ser tolerado num dado intervalo de tempo pode ser mais interessante, pois desta forma, o número de falhas suportada pelo sistema é explicitado. Este tipo de forma de avaliação é prática, do ponto de vista de manutenção dos sistemas, e mais fácil à compreensão do usuário. Entretanto, neste método, a tolerância é definida também pela arquitetura do sistema, e não apenas pela confiabilidade dos componentes utilizados na construção da máquina.

O presente trabalho expressa os níveis de tolerância a falhas na T-NODE em termos do número máximo de processadores defeituosos, não se preocupando diretamente com os parâmetros de confiabilidade dos processadores *Transputer*. Deste modo, os níveis de tolerância a falhas assumidos neste trabalho estão estritamente relacionados com a arquitetura da T-NODE, a qual possui características que facilitam a implementação de arquiteturas com módulos redundantes.

O grande desenvolvimento de técnicas de tolerância a falhas para sistemas computacionais teve como impulso os sistemas críticos, onde uma falha no sistema pode provocar danos irreparáveis, como por exemplo controle de tráfego aéreo. Embora a T-NODE não tenha sido desenvolvida para missões críticas (ver seção 1.2), o uso de técnicas de tolerância a falhas sobre sua arquitetura pode melhorar sua disponibilidade e/ou confiabilidade.

[2]Pode ser que um sistema t-tolerante continue operando corretamente mesmo após a ocorrência de mais do que t falhas, mas a operação correta não pode ser garantida.

Na construção de sistemas tolerantes a falhas pode se adotar várias estratégias. Em [SIE82] é apresentada uma taxonomia para estas diferentes estratégias, onde o autor divide o nível de tolerância a falhas empregada nos sistemas em três sub-regiões: detecção de falhas, redundância para mascaramento de falhas e redundância dinâmica (ver figura 1.1).

Na região de detecção de falhas são usadas técnicas como: códigos de detecção de erros, controle de tempo e *timeout*, verificações de consistência, e outras; na região de redundância para mascaramento de falhas encontram-se as técnicas de redundância n-modular (nMR) com votação, códigos de correção de erros, e outras; na região de redundância dinâmica estão as técnicas que usam módulos sobressalentes que podem eliminar defeitos através de reconfiguração e recuperação.

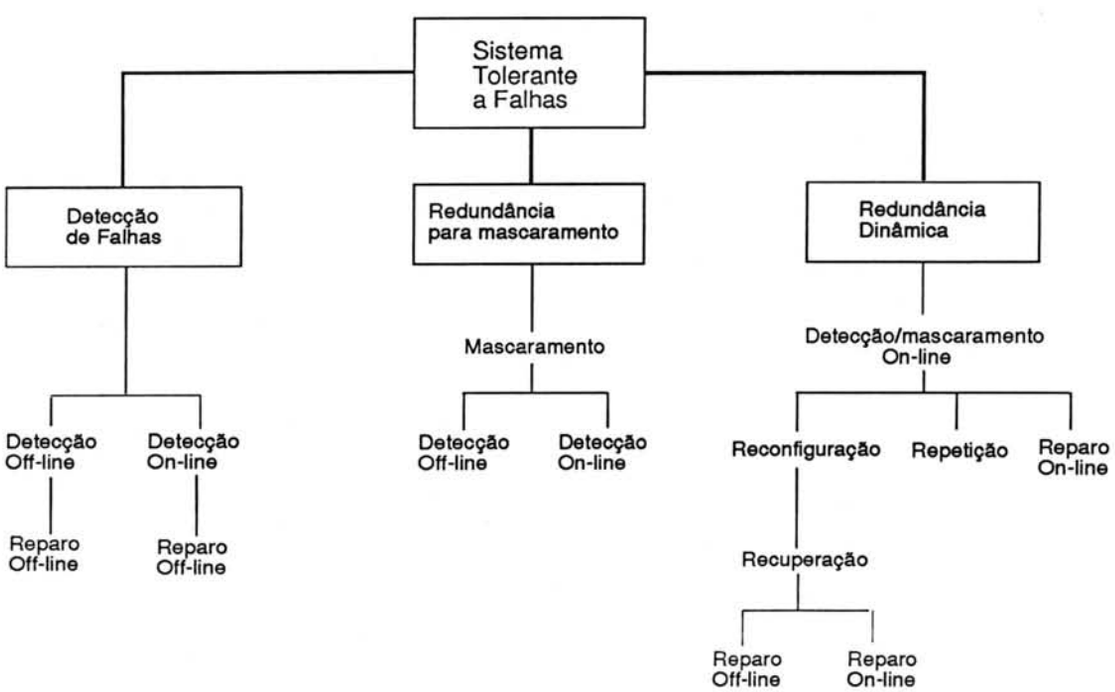

Figura 1.1: Taxonomia de estratégias em sistemas tolerantes a falhas.

Destas técnicas, as que melhor se adaptam às arquiteturas com vários nodos processadores interconectados através de uma rede de chaves programáveis, como é o caso da máquina T-NODE, são as de redundância dinâmica, pois permitem que a falha seja detectada e isolada através do uso de algoritmos de

reconfiguração. A vantagem em relação à redundância para mascaramento é que ela consegue isolar a falha, livrando o sistema de possíveis efeitos colaterais produzidos por um elemento falho.

Kartashev [KAR89] com a intenção de propor uma taxonomia para reconfiguração, separou os algoritmos de reconfiguração em dois grandes conjuntos:

- reconfiguração arquitetural; e
- reconfiguração tolerante a falhas.

No primeiro conjunto, os algoritmos resolvem mudanças de um estado arquitetural[3] para outro, com a finalidade de melhorar o desempenho do sistema na resolução de tarefas. No segundo conjunto, os algoritmos eliminam elementos defeituosos da arquitetura a fim de garantir algum nível de tolerância a falhas no sistema. Dependendo da disponibilidade de módulos no sistema, existem duas estratégias distintas:

- relocação dinâmica; ou
- degradação gradual.

A estratégia de relocação dinâmica é usada quando existem componentes sobressalentes no ambiente arquitetural, de modo a serem usados em caso de falhas. Nesse caso, quando ocorre um defeito em algum componente, o algoritmo de reconfiguração troca o mesmo por um sobressalente sem defeito, mantendo-se deste modo a topologia e o desempenho do sistema.

A estratégia de degradação gradual, parte do pressuposto que não existe componente sobressalente. Deste modo, após a eliminação do defeito, um

[3] Estado arquitetural são configurações que podem estar sendo utilizadas no momento de execução de um programa. Por exemplo, um *array*, um *pipeline*, são estados arquiteturais.

componente ativo e não defeituoso assume, além de suas próprias tarefas, as tarefas do defeituoso, causando no sistema uma alteração da topologia e degradação no desempenho. Seu objetivo é manter o sistema funcionando livre de falhas, mesmo com um nível reduzido de desempenho.

As duas estratégias acima podem ainda ser combinadas, ou seja, o sistema pode usar a primeira até que não existam mais componentes sobressalentes, passando em seguida para a segunda estratégia.

Como sistemas com muitos processadores e estruturas de interconexão específicas estão sendo cada vez mais utilizados e incrementados, e como a probabilidade de defeito no sistema aumenta com o aumento do número de processadores, computadores de média e grande escala incorporam tolerância a falhas estruturais. Dutt [DUT92] define tolerância a falhas estruturais como sendo a habilidade de reconfigurar os processadores da rede, mantendo a estrutura de interconexão do multiprocessador. Deste modo, vários algoritmos de reconfiguração para arquiteturas multiprocessadoras já foram propostos, sendo que a maioria destes atende arquiteturas específicas.

1.2 O Projeto Supernode

O projeto Esprit I P1085 "SUPERNODE" [FLI89], tinha como objetivo de desenvolver uma arquitetura altamente paralela baseada em *transputers*, que obtivesse um alto desempenho a baixo custo. Como resultado deste projeto foi desenvolvida a máquina T-NODE. Esta máquina foi produzida pela Telmat Informatique, com uma arquitetura modular reconfigurável e tendo como processador básico o *transputer* T800.

A T-NODE é uma máquina multiprocessadora fracamente acoplada, que tem como uma de suas principais características a possibilidade de alteração

dinâmica da topologia da rede de processadores, quando necessário, através da reprogramação de chaves eletrônicas. Sua arquitetura pode suportar de 8 a 1024 processadores de trabalho.

Como o objetivo principal do projeto visava obter uma máquina com alto desempenho, mas com custo não muito elevado, a única técnica de tolerância a falhas implementada foi o código de paridade no armazenamento em memória. Entretanto, a característica de reconfigurabilidade da máquina sugere que técnicas de reconfiguração tolerante a falhas venham a ser implementadas por software.

1.3 Objetivos da dissertação

Esta dissertação tem como objetivo propor uma estratégia de detecção de falhas / reconfiguração na máquina T-NODE, considerando como módulos básicos o par processador-memória. O trabalho apoia-se em uma análise das características da máquina T-NODE no tocante a aspectos de reconfigurabilidade, de modo que a confiabilidade e a disponibilidade da máquina possam ser melhoradas.

A estratégia considera somente falhas pertencentes a classe *fail-stop* e propõe-se a tolerar s falhas sem degradação e $n-1$ falhas com degradação, onde s é o número de módulos sobressalentes e n o número de processadores de trabalho da máquina. A detecção e o tratamento de falhas bizantinas é excessivamente caro e complexo do ponto de vista de implementação, diante do tipo de aplicações visado (que não são críticas).

O texto está organizado da seguinte forma: no capítulo 2 são descritas as características gerais do processador *transputer* e os detalhes da arquitetura e da rede de interconexão da T-NODE. No capítulo 3 são brevemente descritas as ferramentas de *software* desenvolvidas pelo fabricante da T-NODE (Telmat) espe-

cialmente para sua arquitetura, e o sistema operacional HELIOS, o qual serve de referência para as ferramentas Telmat mais recentes. No capítulo 4 são apresentadas algumas estratégias de reconfiguração para arquiteturas em árvores e *array*, as quais envolvem a maioria dos algoritmos de reconfiguração já propostos pela literatura. No capítulo 5 são avaliadas as possibilidades de reconfiguração na T-NODE e apresentada uma estratégia de reconfiguração proposta para esta arquitetura. No capítulo 6 é descrito o ambiente arquitetural e de programação que foi usado para implementar um simulador para esta estratégia, e avaliados os resultados obtidos.

2 CARACTERÍSTICAS DA ARQUITETURA T-NODE

Este capítulo apresenta as características mais relevantes do processador usado na T-NODE, o *transputer*, seguido de uma descrição detalhada da arquitetura desta máquina.

2.1 Transputer

O *transputer* é um microprocessador de 32 bits fabricado pela INMOS. Sua filosofia de projeto está baseada no conceito de processos seqüenciais comunicantes e concorrência (conforme será visto na seção 2.1.1). Vários modelos já estão disponíveis comercialmente, como por exemplo, o M212, o IMS T414, o IMS T800 e o T805. O modelo mais recente é o T9000 [INM91] que possui mudanças arquiteturais profundas como, por exemplo, canais específicos para serviços do sistema. Entretanto, este último modelo ainda não está disponível comercialmente por problemas de projeto.

O modelo IMS T800 difere do IMS T414 por possuir uma unidade de ponto flutuante, e é o mais usado na T-NODE, pois corresponde a todos os *transputers* de trabalho[1] da máquina. Sua concepção foi desenvolvida através do projeto Esprit I P1085 [FLI89], com a finalidade de ser o processador básico do SUPERNODE.

A arquitetura do *transputer* T800 é composta por:

[1] *Transputers* de trabalho são os que ficam a disposição do usuário. Controladores e servidores não podem resolver tarefas do usuário, somente do sistema (ver seção 2.2).

- uma unidade central de processamento (CPU) de 32 bits, com um ciclo de instrução de 33ns;
- uma unidade para cálculo em ponto flutuante (FPU) de 64 bits (padrão IEEE 754) capaz de resolver 2 milhões de operações de ponto flutuante por segundo (MFLOPS);
- 4KBytes de memória RAM estática (SRAM) com taxa de acesso de 120 Mbits por segundo;
- uma interface para memória externa com velocidade de 150ns, capaz de endereçar até 2^{30} palavras;
- uma memória ROM contendo o microcódigo de um escalonador de instruções OCCAM, que se parece com um pequeno núcleo de sistema operacional de tempo real;
- quatro canais seriais assíncronos (*links*) com conexões ponto a ponto, capazes de suportar transferências simultâneas em ambas as direções numa taxa de 5/10/20 Mbits por segundo.

Os blocos básicos do IMS T800 e suas interconexões são mostradas na figura 2.1.

2.1.1 Modelo de Programação

2.1.1.1 Modelo OCCAM

OCCAM é uma linguagem de programação de alto nível, construída com base na teoria de processos seqüenciais comunicantes (CSP) proposta por Hoare em [HOA78]. Em CSP os processos seqüenciais executam concorrentemente e se comunicam através da troca de mensagens. Em OCCAM, a comunica-

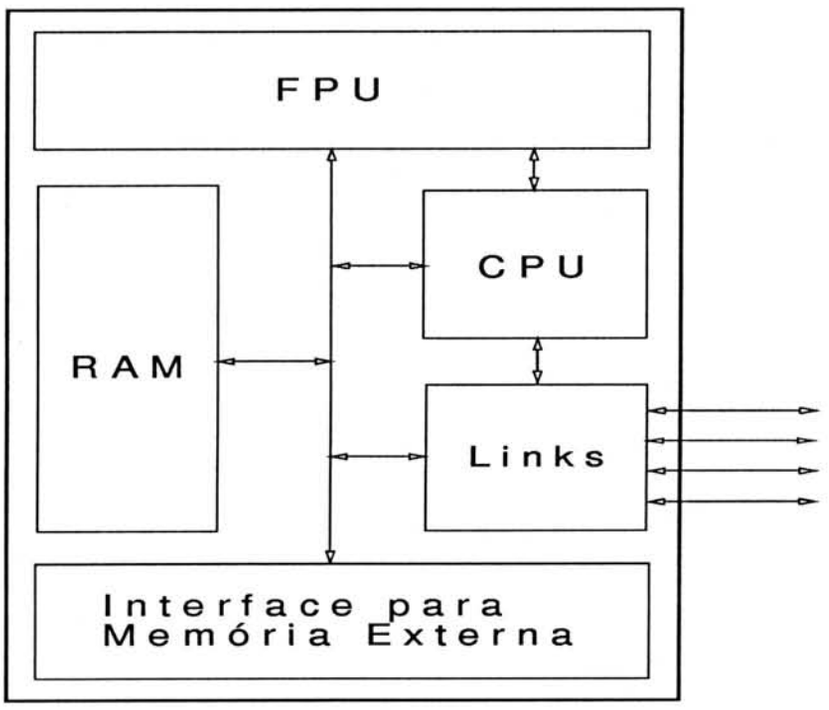

Figura 2.1: Diagrama de blocos do *transputer* IMS T800.

ção entre processos é implementada através de canais que fornecem um mecanismo de comunicação entre dois processos concorrentes. A comunicação através deste canal é síncrona e somente se efetiva quando os processos emissor e receptor estiverem prontos para a troca de mensagens. Esse protocolo sempre bloqueia um dos processos pela espera do sincronismo (*rendez-vous*). Após a troca de mensagens os processos continuam suas execuções. Este protocolo é similar ao protocolo *handshake* usado em hardware.

No modelo OCCAM pode-se ter um conjunto de processos operando concorrentemente, que podem ser compostos por outros processos também operando concorrentemente. A comunicação entre os processos é feita pela troca de mensagens através de canais bidirecionais. Na figura 2.2 é esquematizada uma situação possível, onde três processos (Pa, Pb e Pc) operam concorrentemente entre si comunicando-se através dos canais 1, 2 e 3. O processo Pa e Pc são compostos por dois subprocessos (Pa1, Pa2 e Pc1, Pc2, respectivamente) eo processo Pb é composto por três subprocessos (Pb1, Pb2 e Pb3).

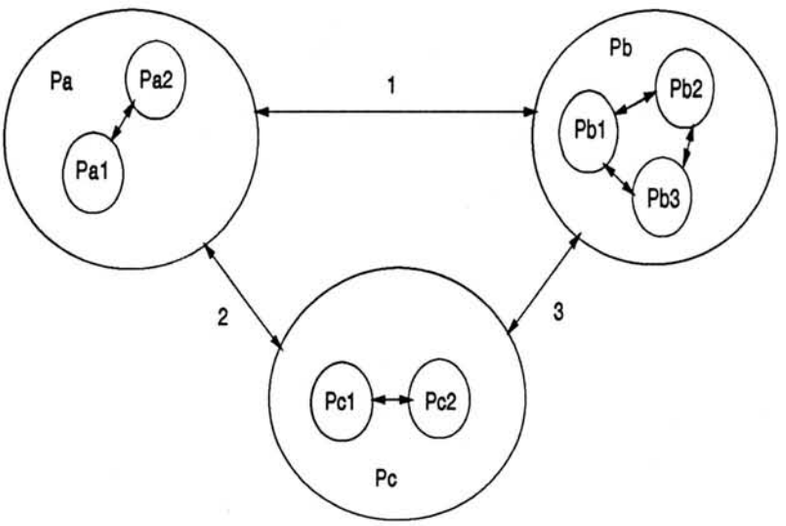

Figura 2.2: Processos OCCAM.

2.1.1.2 Modelo OCCAM Implementado no Transputer

Como foi construído para suportar o modelo de programação OCCAM, o *transputer* implementa em seu *hardware* conceitos de concorrência e comunicação desse modelo.

O suporte a um grande número de processos concorrendo pela CPU é implementado no *transputer* por meio de um escalonador de processos em *hardware*, que gerencia processos classificados em dois níveis de prioridade, os quais serão estudados na seção 2.1.2.

Processos localizados em *transputers* distintos comunicam-se através de canais seriais implementados em *hardware*, enquanto que processos localizados em um mesmo *transputer* trocam mensagens através de canais mapeados na memória. Em ambos os casos a comunicação obedece um protocolo *rendez-vous*.

Diversas linguagens de programação podem ser utilizadas para programar o *transputer*, como por exemplo, PASCAL, C, FORTRAN e OCCAM. Destas, a que oferece melhor desempenho é a linguagem OCCAM, desenvolvida também

pela INMOS, pois o *transputer* implementa diretamente instruções geradas pelo compilador desta linguagem.

2.1.2 Prioridade e Escalonamento de Processos

No *transputer* podem rodar muitos processos em paralelo (concorrentemente), pois ele possui um escalonador de processos implementado em *hardware* que gerencia o compartilhamento do tempo do processador. Os processos podem ser de alta ou de baixa prioridade, sendo que os de alta prioridade "preemptam" os de baixa prioridade. O escalonador implementa duas filas de processos, uma para cada nível de prioridade. Processos de alta prioridade executam até alcançar um ponto de desescalonamento. Pontos de desescalonamento são pontos em que o processador fica bloqueado, esperando por uma comunicação com outro processo (devido ao protocolo utilizado), ou por uma entrada de tempo específica (um processo TIMER no OCCAM). Já processos de baixa prioridade executam no máximo durante duas fatias de tempo, ou são forçados, pela existência de um processo de alta prioridade apto a executar, a serem desescalonados no próximo ponto de desescalonamento. O período de cada fatia de tempo é de 5120 ciclos do relógio externo de 5MHz. Mais detalhes sobre *transputers* podem ser encontrados em [INM84] [INM88].

2.2 T-NODE

Subproduto do projeto Esprit I P1085, a T-NODE é uma máquina multiprocessadora fracamente acoplada, que utiliza como processador-base o *transputer*. Sua arquitetura é modular, o que permite sua expanção e reconfiguração (estática ou dinamicamente).

A reconfiguração da arquitetura é feita através da reprogramação de uma rede de interconexão programável, que chaveia os canais *hard* dos processadores que compõem a máquina, permitindo deste modo a construção de diversas topologias arquiteturais como, por exemplo, *array*, *pipeline*, árvore e hipercubo, fazendo desta a característica mais importante da máquina.

A máquina suporta de 8 à 1024 processadores de trabalho, podendo alcançar um desempenho máximo de 12 MFLOPS com 8 processadores e 1500 MFLOPS com 1024. Entretanto, para conseguir este desempenho teve que ser adotada uma estrutura hierárquica na arquitetura, fixada pela implementação de um barramento de controle que pode gerenciar todas as informações do sistema, reduzindo o número de mensagens de controle pelos canais de comunicação dos *transputers* de trabalho.

A seguir são apresentadas as principais características do barramento de controle, que estabelece uma hierarquia em *hardware*, e da rede de interconexão da T-NODE.

2.2.1 Barramento de Controle

Todos os *transputers* da T-NODE estão conectados ao barramento de controle, o qual está mapeado na memória. A função do barramento é liberar os canais *transputers* (*links*) das mensagens do sistema. Entretanto, este barramento tem um protocolo mestre-escravo, onde o mestre é o *transputer* controlador e os escravos, os *transputers* de trabalho. A vantagem deste barramento é que ele permite: uma rápida sincronização entre *transputers*, uma reinicialização seletiva, e um rápido *broadcast* de mensagens.

A hierarquia da T-NODE inserida pelo protocolo mestre-escravo utilizado pelo barramento de controle, e a modularidade da máquina, sugere a análise de três nodos distintos: o nodo básico, o nodo TANDEM e o MEGANODE.

O nodo básico é composto por uma placa que contém o circuito de chaveamento, e sete *slots* (ver figura 2.3) aos quais podem ser conectados de 8 a 32 *transputers* de trabalho (em placas com 8), dois servidores e um controlador, que gerencia as mensagens do sistema e a programação das chaves de interconexão do nodo. Este controlador pode operar tanto como escravo ou como mestre de um outro controlador, dependendo do nível hierárquico em que ele estiver perante outros nodos.

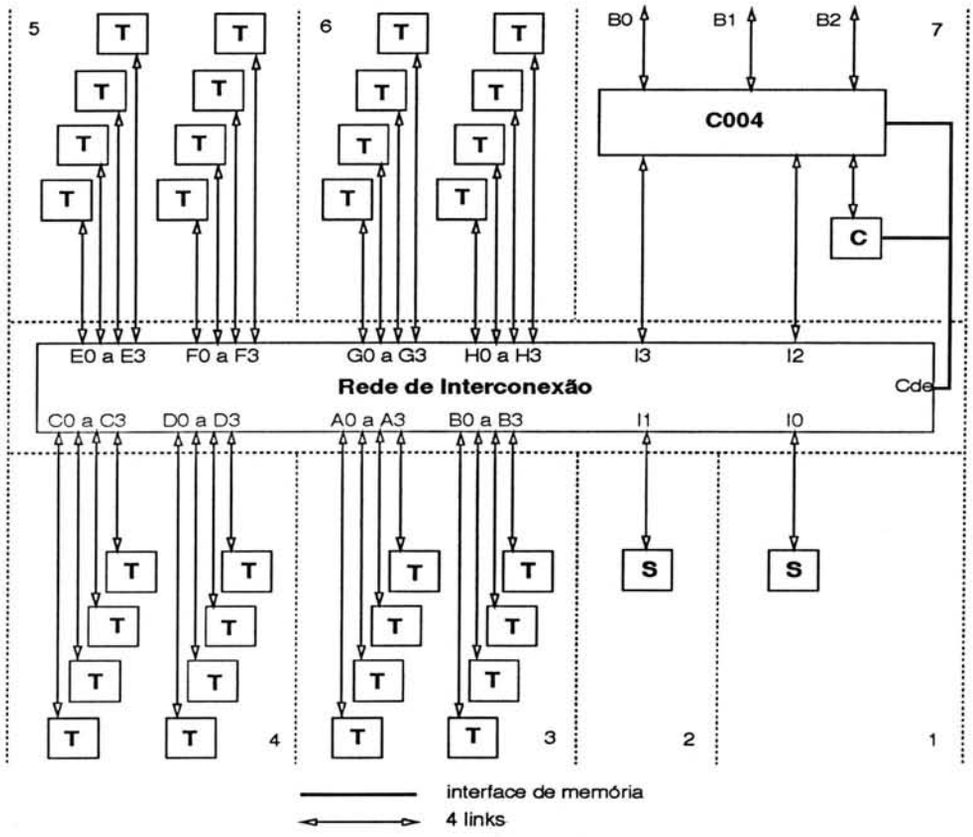

Figura 2.3: Organização do nodo básico.

O nodo TANDEM é composto pela conexão de dois nodos básicos, podendo ter um máximo de 64 *transputers* de trabalho. O controlador de um dos nodos básicos será escravo do outro.

O MEGANODE agrupa até 32 nodos TANDEM através de chaves internodos, podendo formar uma rede de até 1024 *transputers* de trabalho (1024 porque metade dos *transputers* de trabalho de cada nodo TANDEM são substituídos por pla-

cas internodos). Um *transputer* supervisor gerencia todos os controladores mestres dos nodos TANDEM. A figura 2.4 apresenta uma visão geral da organização arquitetural.

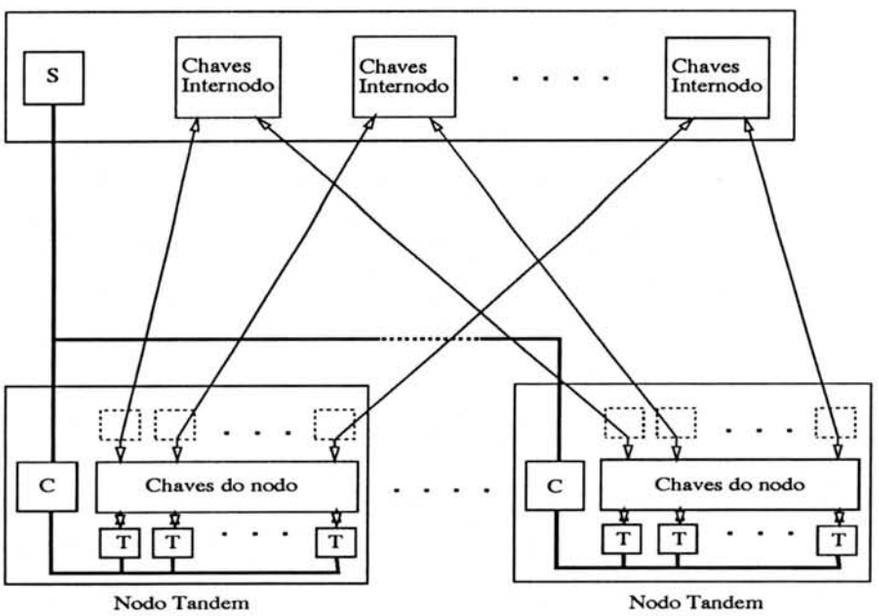

Figura 2.4: Arquitetura da máquina T-NODE.

2.2.2 Rede de Interconexão

Para descrever a implementação da rede de interconexão da T-NODE, serão analisadas separadamente as principais configurações permitidas pela estrutura da máquina, ou seja, o nodo básico, o nodo TANDEM e o MEGANODE. Nesta análise os quatro *links* do *transputer* são nomeados como sendo Norte (N), Leste (L), Oeste (O) e Sul (S).

2.2.2.1 Nodo Básico

No nodo básico, a rede de interconexão é formada por dois circuitos de chaveamento. Cada circuito é composto por duas chaves *crossbar* 72x36, com as entradas conectadas conforme esquematizado na figura 2.5. Ambos os circuitos

de chaveamento são controlados (programados) pelo *transputer* de controle do nodo.

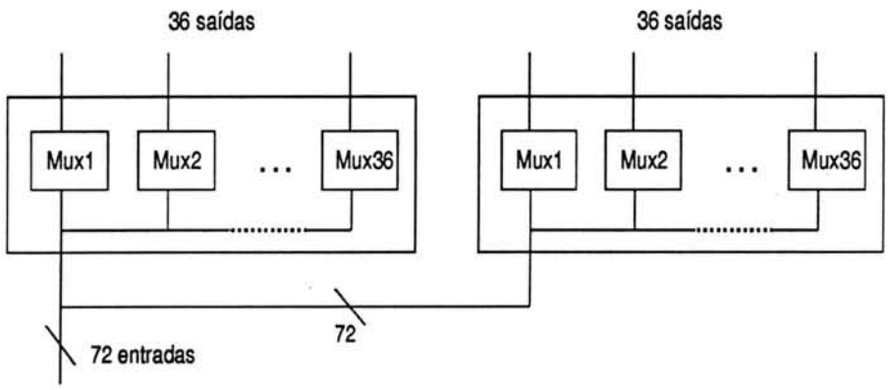

Figura 2.5: Composição do circuito de chaveamento.

Todas as saídas dos canais norte e leste (36 de cada) dos *transputers* do nodo básico são conectadas à entrada de um dos circuitos de chaveamento. As saídas deste circuito são conectadas às entradas dos canais sul e oeste (36 de cada). Todas as saídas dos canais sul e oeste dos *transputers* do nodo básico são conectadas à entrada do outro circuito de chaveamento. As saídas deste circuito são conectadas às entradas dos canais norte e leste. Deste modo, em um nodo básico podem ser estabelecidas as seguintes conexões: Norte<—>Sul, Norte<—>Oeste, Sul<—>Leste e Leste<—>Oeste. Na figura 2.6, são mostradas estas conexões, onde as linhas pontilhadas indicam os *swap-plugs*[2] que são utilizados para formar o nodo TANDEM.

2.2.2.2 Nodo TANDEM

A rede de interconexão do nodo TANDEM é similar a do nodo básico, mas com a diferença de que é inserida uma limitação no número de conexões, ou seja, as ligações feitas entre as duas redes (de cada nodo base) pelos *swap-plugs* fazem com que haja uma divisão entre as redes norte-sul e leste-oeste (ver figura 2.7), impedindo as conexões N<—>O e S<—>L.

[2] *Swap-plugs* são chaves para conectar dois nodos básicos.

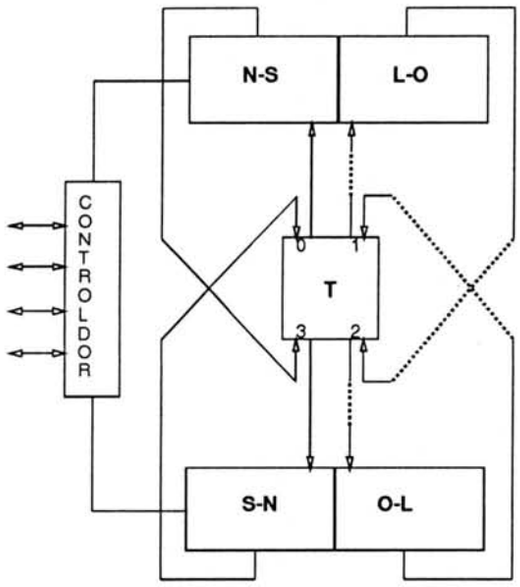

Figura 2.6: Circuito de chaveamento do nodo base.

Tanto no nodo básico como no nodo TANDEM, as redes são totalmente rearranjáveis porque usam *crossbars*.

2.2.2.3 MEGANODE

No MEGANODE, a rede de interconexão entre *transputers* baseia-se na teoria das redes multiestágios de Clos [NIC88], que define uma rede com um estágio intermediário de q *crossbars* com p entradas e p saídas, um estágio de entrada com p *crossbars* com m entradas e q saídas, e um estágio simétrico de saída composto por p *crossbars* com q entradas e m saídas. Esta rede possui a propriedade de ser rearranjável se $q \geq m$ e não-bloqueante se $q \geq 2m - 1$.

A viabilidade de transformação da rede multiestágio de Clos para uma rede unilateral de Clos mantendo as mesmas propriedades, possibilitou o uso de *crossbars* para chavear diretamente entradas e saídas.

Uma análise da rede implementada no MEGANODE mostra que $m = 32$ e $q = 32$, pois a rede no nodo básico tem o mesmo número de entradas e saídas, 32. Como o segundo estágio é composto por chaves *crossbar* C004, produzidas

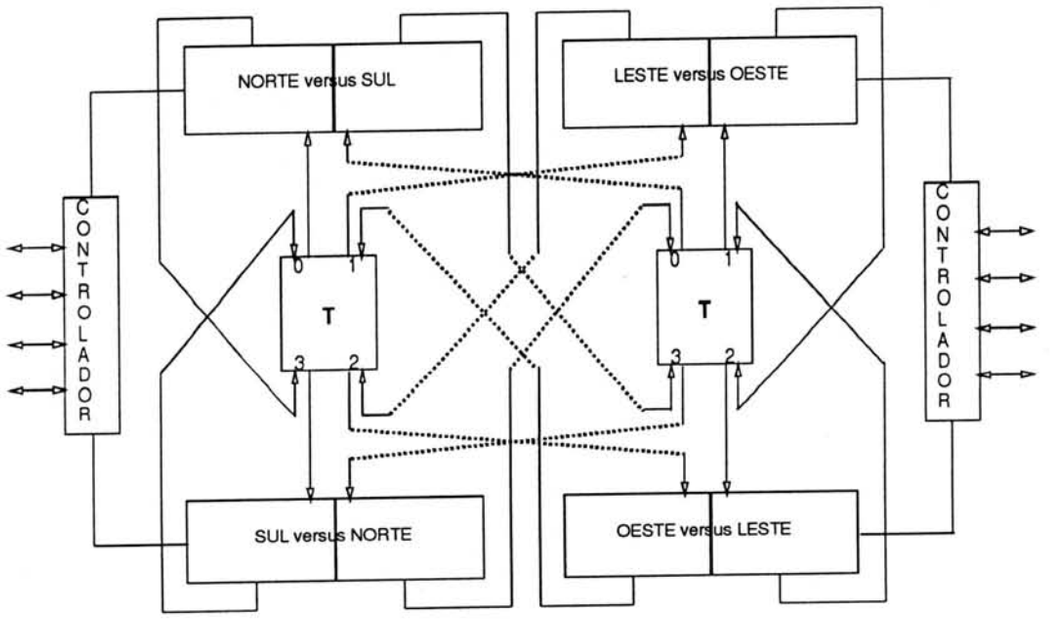

Figura 2.7: Circuitos de chaveamento do nodo TANDEM.

pela INMOS, $p = 32$, pois ela é uma *crossbar* 32x32. Deste modo, pode-se notar que, no caso do MEGANODE, a rede se comporta como totalmente rearranjável mas bloqueante pois não satisfaz a condição $q \geq 2m - 1$.

3 CARACTERíSTICAS DO SOFTWARE T-NODE

Este capítulo descreve as principais características do *software* básico da máquina T-NODE. É descrita a versão 3.0 das ferramentas desenvolvidas pelo fabricante da máquina, a Telmat SA, que foram projetadas para executarem em conjunto com o sistema operacional HELIOS versão 1.2, e também as características do próprio sistema HELIOS 1.2.

3.1 Software Telmat

Esta seção apresenta os *softwares* produzidos pela Telmat S.A. especificamente para gerenciamento da máquina T-node e também uma breve descrição do arquivo *snconfig.src* que descreve o *hardware* disponível na T-NODE.

3.1.1 Ferramentas T-NODE

Inicialmente, logo que a T-NODE foi lançada ao mercado, ela vinha com um *software* residente composto por três programas: o T.Configurer, o T.Switcher e o T.Kernel, os quais formam a versão 2.2 das ferramentas T-NODE, que eram utilizadas por aplicações *standalone*. Este *software* continha sérias implicações no que se refere a aspectos de gerenciamento da máquina, principalmente porque permitia que um usuário pudesse definir diretamente o chaveamento da máquina. Deste modo, na versão 3.0 as ferramentas T-NODE foram separadas em dois grupos, o T.Kernel[1], que executa sobre os *transputers* controladores, e o T.Library, que executa sobre os *transputers* de trabalho. No que se refere ao chaveamento da T-NODE, o antigo T.Switcher 2.2 foi reescrito sob a forma de dois servidores

[1]O T.Kernel é um *software* de sistema usado para lidar com os acessos ao *hardware*. Todos os comandos de *hardware* tem que ser transmitidos ao módulo T.Kernel para serem executados.

do sistema operacional HELIOS 1.2 ([PER90] [TEL91e]): o SwMan (*Switch Manager*) acessível pelo usuário via servidor de rede, pode manusear os comandos de chaveamento, em alto nível, e o SwDrv (*Switch Driver*), que é distribuído (uma cópia em cada) pelos controladores da T-NODE e que pode manusear comandos de *hardware*, em baixo nível. Inicialmente estes servidores só foram projetados para executarem sobre a T-NODE nodo básico e TANDEM.

As vantagens da versão 3.0 em relação à versão 2.2 são:

1. que somente aplicações HELIOS podem comunicar-se com os servidores do T.Kernel;
2. que a parte de chaveamento do T.Kernel 3.0 (HELIOS Switcher 3.0) trabalha com os números reais de identificação de processadores (*Processors Identification Numbers* - PIN);
3. que o gerenciamento de recursos é agora feito pelo sistema HELIOS 1.2, o que libera os servidores de chaveamento destes aspectos, sendo que a requisição tem que ser enviada ao servidor HELIOS antes de ser enviada ao servidor de chaveamento;
4. que o mesmo servidor de rede manipula funções de chaveamento para as diversas configurações da T-NODE.

Estas características, aliadas a grande integração do *software* básico da T-NODE com o sistema operacional HELIOS, sugerem que o algoritmo de detecção e recuperação de falhas proposto seja incorporado ao ambiente de programação HELIOS, embora existam vários outros sistemas operacionais para máquinas *transputer* como, por exemplo, o TRIX [PAZ91], que está sendo desenvolvido no CPGCC-UFRGS, e o PARIX, que executa em máquinas Parsytec, mas ambos não possuem *drivers* específicos para o caso desta máquina.

No TRIX, o *boot* do sistema inicializa uma rede de processadores baseando-se em tabelas de roteamento previamente estabelecidas. Estas tabelas, defi-

nidas em tempo de compilação, utilizam variáveis que podem alterar as possibilidades de roteamento de mensagens entre os nodos. Entretanto, a alteração do roteamento só pode ser feita por *software*, porque é o *kernel* do sistema quem gerencia os canais *hard* do *transputer*. Logo, dinamicamente só é possível alterar conexões lógicas entre processadores e não físicas.

Recentemente, Bart Veer e Nick Garnett [VER93], também preocupados com tolerância a falhas a nível do sistema operacional, propuseram várias alterações no sistema HELIOS, que fazem com que este possua recursos em termos de tolerância a falhas como, por exemplo, detecção de defeitos, roteamento de mensagens através de nodos defeituosos e reinicialização de nodos defeituosos se possível. Estes recursos estão presentes na versão 1.3 do HELIOS.

3.1.2 Arquivo *snconfig.src*

O arquivo *snconfig.src* [TEL91a] especifica a configuração de *hardware* da máquina T-NODE e é utilizado pelos servidores SwMan e CbMan para a determinação das conexões da máquina. Seu conteúdo é formado por um conjunto de tabelas que descrevem:

- a estrutura da máquina em termos de Snodos, onde cada Snodo equivale a um nodo básico;
- os *transputers* com suas próprias características;
- os canais de interface com o hospedeiro;
- os canais entre os Snodos.

Para o propósito deste trabalho são descritas aqui somente as tabelas que contêm as características dos processadores da máquina, as tabelas *Transputers*. Cada tabela especifica os dados dos *transputers* com o tipo de função

respectivo, com seus canais, números de identificação, tamanho da memória, tipo de *transputer*, número do Snodo, endereço das chaves e do barramento de controle, e a entrada na tabela *twiddle* (descrita a seguir).

Cada tabela começa com uma palavra chave, que identifica o tipo do processador que é descrito por ela, seguida das descrições dos processadores, uma em cada linha. A tabela 3.1 relaciona os tipos possíveis e a tabela 3.2 mostra a estrutura de colunas das tabelas *transputers*.

Tabela 3.1: Tipos de *transputer* quanto à sua função.

Tipo	Função
WRK	transputer de trabalho
MEM	servidor de memória
DSK	servidor de disco
CTL	controlador
SPC	outra especialidade
BUF	buffer

Tabela 3.2: Campos das tabelas *Transputers*.

Direções				Características			End. chaves		End. bar. controle			Twi
N	L	O	S	PIN	mem	tipo	Snode	grupo	grupo	mybit	ben	order
							. . .					
0	1	2	3	4	5	6	7	8	9	10	11	12
Colunas												

As colunas das tabelas *transputers*, mostradas esquematicamente através da tabela 3.2, fornecem as seguintes informações:

- **colunas 0,1,2 e 3** especificam as direções dos quatro canais *transputer*;
- **coluna 4** especifica o número de identificação do processsador (PIN);
- coluna 5 especifica o tamanho da memória associada ao *transputer*;
- **coluna 6** especifica o tipo de *transputer* (por exemplo, T414, T800, ...);
- **coluna 7** especifica a que Snodo pertence o processador;

- **coluna 8** especifica o endereço do processador para a chave *crossbar*, o qual é utilizado pelas ferramentas T-NODE para conectar processadores;
- **coluna 9** especifica o endereço do grupo do processador no barramento de controle;
- **coluna 10** especifica o número do *transputer* dentro do grupo;
- **coluna 11** especifica se o processador pertence ao Snodo escravo ou mestre;
- **coluna 12** especifica o número de entrada na tabela Twiddle, que contém dados para as conexões das chaves C004.

3.2 Sistema Operacional HELIOS

O sistema operacional HELIOS ([PER90] [TEL91e]), desenvolvido pela Perihelion Software para redes de *transputers*, é um sistema operacional distribuído, para ambientes multiprocessadores do tipo MIMD, que se baseia no paradigma cliente-servidor. Neste sistema, os processadores são considerados recursos que podem ser inicializados por um *kernel* mínimo chamado *nucleus*, o qual controla a alocação de processadores e de memória. Uma cópia idêntica do *nucleus* está distribuída por todos os processadores da rede.

O suporte de E/S é dado por um servidor de E/S que executa num processador hospedeiro como, por exemplo, um computador pessoal ou estação de trabalho, que serve como interface com o usuário. Neste sistema nenhuma topologia particular é assumida, permitindo, deste modo, que topologias diversas sejam implementadas sobre a máquina T-NODE.

As próximas subseções descrevem os principais aspectos do *software* básico da T-NODE integrante do HELIOS que influenciam no momento da execução de uma alteração nas conexões da máquina.

3.2.1 Drivers e Servidores para Controle de Inicialização e Configuração

Na versão 1.2 do HELIOS, as placas controladoras da T-NODE fazem parte de uma rede HELIOS especial, a rede do sistema, que é formada por todos os *transputers* controladores, servidores e *roots* (mais de um no caso de operação multiusuário). A rede do sistema é utilizada para executar ferramentas específicas de apoio ao *boot*, ao interfaceamento T-NODE/hospedeiro e ao gerenciamento da máquina.

O HELIOS possui um servidor de rede responsável pela alocação de processadores a usuários e pelo controle do roteamento de mensagens entre os processadores alocados a um usuário específico e o seu processador *root*.

As ferramentas que fazem a interface entre o sistema operacional HELIOS e a máquina T-NODE foram desenvolvidas pela Telmat SA. Estas ferramentas incluem *drivers* e servidores, os quais são brevemente descritos a seguir (maiores detalhes sobre estes *drivers* e servidores podem ser encontrados em [PER90], [TEL91e] e [TEL91b] [TEL91c]):

- **telmat_r.d driver**: gerencia as requisições de inicialização. Este código é carregado no momento do *boot* pelo servidor de rede. As chamadas de função de inicialização feitas pelo servidor de rede (via o telmat_r.d driver) é passada ao CbMan (*Control Bus Manager*);

- **telmat_c.d driver**: gerencia as requisições de configuração. Ele é encarregado de executar a fase de configuração. Este *drive* também é

carregado pelo servidor de rede no momento do *boot*. As requisições são então passadas para o SwMan (*Switch Manager*);

- **CbMan server**: este servidor recebe requisições dos clientes HELIOS através do servidor de rede (via o telmat_r.d driver), e gerencia os acessos ao barramento de controle da máquina, baseando-se nos dados fornecidos pelo arquivo *snconfig.src*. As requisições são do tipo: reinicialização de um *transputer*, envio ou recebimento de dados através do barramento de controle, etc. Encarregado da inicialização da T-NODE, ele envia as requisições de baixo nível ao CbDrv (*Control Bus Driver*);

- **CbDrv driver**: é um servidor HELIOS encarregado de gerenciar o *hardware* do barramento de controle. Este servidor é executado sobre a placa controladora mestre;

- **SwMan server**: este servidor também recebe requisições dos clientes HELIOS através do servidor de rede (via o telmat_c.d driver), e gerencia as chaves do *backplane*, que também baseia-se nos dados fornecidos pelo arquivo *snconfig.src*. Os tipos de requisições que ele manipula são, entre outras, o ajuste das conexões dos canais *transputer* e a verificação de conexão. Encarregado de estabelecer a configuração inicial da rede, ele envia as requisições de baixo nível ao SwDrv (*Switch Driver*);

- **SwDrv driver**: é um servidor HELIOS encarregado de manusear os comandos de chaveamento de baixo nível, interagindo diretamente com o *hardware*.

Os drivers de dispositivos, tanto para configuração das chaves eletrônicas como para controle de inicialização, garantem a independência do *hardware* e do *software* no sistema. A figura 3.1 mostra a localização e interação destas ferramentas.

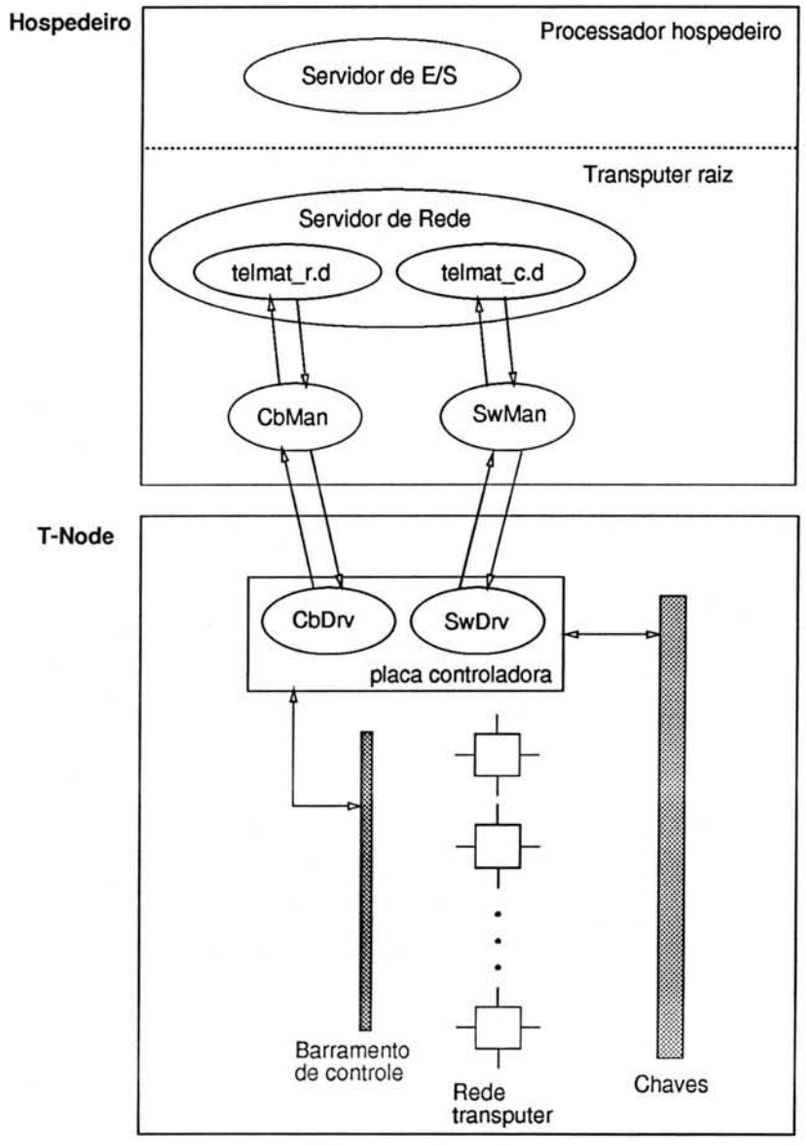

Figura 3.1: Localização e interação dos *drivers* e servidores que compõem o *software* básico da T-NODE.

3.2.2 Arquivos de Configuração

O sistema operacional HELIOS possui vários arquivos de configuração, que estabelecem parâmetros para auxiliar os diferentes tipos de *boot*, configurações e características de seções do usuário.

Dentre os diversos arquivos de configuração, destacam-se:

- **initrc**, que lido pelo programa **init**, inicia o software de rede específico para o T-NODE;
- **nsrc**, que lido pelo servidor de rede, indica opções de acordo com o modo de operação do T-NODE (mono ou multi-usuário);
- **snconfig.src**, que contém a descrição do *hardware* disponível na máquina, e é utilizado pelas ferramentas de rede Telmat;
- **mapa de recursos**, que contém a descrição da topologia de rede a ser configurada sobre a máquina, e é lido pelo servidor de rede HELIOS.

O mapa de recursos é um arquivo texto que descreve a topologia de rede em uma linguagem especializada projetada para este propósito. Este arquivo é compilado pelo compilador *rmgen* que verifica algumas validações de correspondência com outros arquivos de inicialização, e que gera o código objeto a ser usado pelo servidor de rede.

O mapa de recursos pode conter as seguintes informações:

1. o nome da rede ou a hierarquia dos nomes da rede;
2. a descrição dos processadores;
3. a especificação do *driver* de inicialização;
4. a especificação do *driver* de configuração;

5. as facilidades adicionais de inicialização que podem ser disponíveis.

Destas a que merece maior atenção é a que descreve os processadores e, em especial, a conectividade dos processadores. Esta descrição é feita da seguinte forma:

processador <nome> <conexões>; <opções>

Geralmente a topologia especificada no mapa de recursos através do campo <conexões> é a topologia atual da rede. Entretanto, para a máquina T-NODE, que tem restrições no chaveamento de canal em *hardware* por ela somente conectar um canal a outros dois canais de qualquer outro *transputer*, a topologia física (implementada) pode não corresponder à lógica, constante nos mapas, ou seja, se o mapa de recursos indica que o *link* 0 de algum processador é conectado a outro processador, algum *link* está conectado àquele processador, mas não necessariamente o *link* 0.

No campo <opções> são definidos: o modo de operação do processador (Helios, IO, System ou Nativo), seu tipo (T400, T800, ...), tamanho da memória e outras informações adicionais. No modo de operação Helios, o processador é encarado como um *transputer* trabalhador. No modo IO, como um processador de E/S, não podendo executar aplicações e nem ser inicializado pelo servidor de rede. No modo System, como processador de uso exclusivo do sistema. No modo Nativo, como não utilizado, não sendo inicializado e nem usado pelo sistema, mas poderá ser incluído no mapa de recursos, com suas respectivas conexões, se o *hardware* puder chavear os canais através de chaves de canal.

3.2.3 Inicialização e Configuração da Rede pelo HELIOS

Quando a máquina T-NODE está operando com o sistema operacional HELIOS, a configuração da rede de interconexão é feita de acordo com o mapa

de recursos definido pelo usuário. O mapa de recursos deve ser compilado pelo compilador *rmgen*, o qual não conhece o *hardware* real, mas acredita que o mapa de recursos tenha correspondência com este.

Na fase de *boot* da T-NODE o servidor de rede, ao ser inicializado, carrega os *drivers* de inicialização e configuração, que estabelecem conexões com o CbMan e SwMan, e lê o mapa de recursos. Através dos *drivers*, o servidor de rede envia requisições ao CbMan e SwMan para inicializar e configurar os processadores conforme o mapa de recursos. Neste ponto o CbMan e SwMan lêem o arquivo *snconfig.src* e então executam as requisições feitas pelo servidor de rede. Se o mapa de recursos solicitar mais processadores do que o *hardware* contiver, mensagens de erro serão geradas. A fase de *boot* termina somente com os primeiros processadores (correspondente ao número de processadores físicos) inicializados e configurados, sendo os processadores restantes (após esgotar-se os recursos físicos da máquina) do mapa de recursos ignorados pelo HELIOS.

3.2.4 Suporte a Linguagens de Alto Nível

O HELIOS suporta várias linguagens de alto nível, entre elas Parallel C e OCCAM, sendo que os compiladores de todas elas geram um código padrão convencionado pelas chamadas do sistema.

Neste sistema o usuário pode determinar em que processadores físicos da rede seus processos devem ser executados, o que facilita o uso de linguagens de alto nível para aplicações onde exista a real necessidade de definir a posição dos processos.

4 ALGORITMOS DE RECONFIGURAÇÃO

Algoritmos de reconfiguração em arquiteturas multiprocessadoras podem ter dois fins distintos: modificar a topologia de uma rede qualquer a fim de aumentar o desempenho do sistema na resolução de tarefas, ou modificar a topologia da rede para obter algum grau de tolerância a falhas no sistema através do isolamento de um nodo defeituoso.

Com base em um estudo da arquitetura do T-NODE [NUN92], do ponto de vista confiabilidade, verificou-se que a propriedade de reconfigurabilidade disponível nesta máquina pode servir para dar-lhe algum grau de tolerância a falhas, além de permitir o uso de topologias variadas para aumentar seu desempenho.

Normalmente a reconfiguração para tolerância a falhas em arquiteturas multiprocessadoras é feita através de algoritmos que atendem arquiteturas específicas. Estes algoritmos utilizam o conceito de nodos redundantes na rede de processadores, sendo que esta redundância pode ser de dois tipos: de *hardware* ou temporal [CHE90].

A redundância de *hardware* indica que existem mais nodos no sistema do que a configuração lógica utiliza, permitindo assim que os nodos falhos sejam substituídos por nodos sobressalentes não falhos. Já a redundância temporal indica que a configuração lógica ocupa todos os nodos do sistema, permitindo somente degradação gradual de desempenho, ou seja, as tarefas dos nodos falhos são redistribuídas aos nodos não falhos aumentando com isso o tempo total de processamento.

Neste capítulo serão analisadas algumas técnicas de reconfiguração sob falhas propostas para arquiteturas multiprocessadoras específicas. Esta análise procura identificar as principais características destes algoritmos a fim de usá-las na elaboração da estratégia de reconfiguração sob falha da arquitetura da

T-NODE. As arquiteturas específicas escolhidas para esta análise foram as nas formas de *array* e árvore visto que ambas são alvos da maior parte dos algoritmos de reconfiguração propostos na literatura, como conseqüência da escolha destas topologias para maioria das máquinas multiprocessadoras.

As estratégias analisadas neste capítulo são todas para arquiteturas homogêneas, ou seja, que possuem todos os processadores (ou nodos) idênticos. Logo, apresentam características facilmente portáveis para a máquina T-NODE. Estratégias para arquiteturas não-homogêneas não são consideradas.

4.1 Técnicas para Reconfiguração de Arquiteturas Array

Geralmente as arquiteturas do tipo *array* utilizam um grande número de processadores e são implementadas em *chips*, os quais estão tendo um grande aumento no número de seus processadores internos, bem como de suas tarefas. Estes parâmetros implicam também em um aumento da probabilidade de defeito, que pode se tornar um fator limitante importante para a fabricação destes componentes. Deste modo, vários pesquisadores, dentre eles [FOR85][CHE90], desenvolveram técnicas para recuperação de erros a fim de tolerar falhas na produção ou em sua vida ativa. Em geral, estas técnicas se baseiam em elementos sobressalentes para casos onde a estrutura é homogênea, a fim de poderem relocá-los na arquitetura de forma a substituir algum elemento defeituoso.

Os *chip-arrays* normalmente possuem um conjunto homogêneo de processadores, ou seja, o mesmo tipo de processadores em todos os nodos. Esta característica é um ponto em comum com a T-NODE, que também pode ser configurada na forma de um *array* homogêneo (máximo 32x32).

Com a intenção de propor uma taxonomia, Chean [CHE90] dividiu as técnicas de reconfiguração existentes para arquiteturas do tipo *array* em quatro grandes classes:

1. **chaveamento de conjunto** - onde a redundância de *hardware* é global, e grupos de processadores são chaveados. Um exemplo desta classe é a substituição de uma linha ou coluna inteira quando um dos processadores de uma linha ou coluna está defeituoso;

2. **redundância local** - onde existe um processador redundante para cada grupo de processadores ativos. Um exemplo deste caso é a existência de um processador para cada grupo de quatro processadores;

3. **chaveamento de processador** - onde existe um número qualquer de processadores redundantes que podem substituir qualquer um dos processadores ativos sob falha;

4. **redundância temporal** - onde processadores redundantes não são necessários, mas tem-se degradação gradual no desempenho do sistema. Processadores ativos bons assumem a tarefa dos processadores ativos sob falhas.

A primeira classe não é de interesse para este trabalho pois exige um grande número de processadores redundantes para suportar pequeno número de falhas. Isto põe em risco parâmetros de desempenho da máquina T-NODE uma vez que ocorre diminuição significativa no número de processadores disponíveis. Por exemplo, numa máquina T-NODE contendo 128 elementos processadores de trabalho pode-se implementar no máximo um *array* 11x11, ocupando 121 dos 128 processadores da rede. Para tolerar duas falhas, usando um algoritmo pertencente à classe 1, deve-se ter dois elementos redundantes por linha ou coluna, o que impede o uso de um array 11x11, pois só existem 7 processadores sobressalentes. Deste modo, o array máximo será o 10x10, sobrando 28 processadores. Neste caso

pode-se usar 20 dos 28 restantes para suportar duas falhas por linha no array. A degradação, em relação ao número de processadores disponíveis, ocorrida neste caso é de $28 * 100/128 = 22\%$.

A segunda classe pode utilizar, para o exemplo anterior, um processador redundante para cada grupo de quatro processadores, o que melhora o nível de tolerância em relação a primeira classe, que tinha dois sobressalentes para cada 10 processadores. Entretanto, as proporções de redundância são semelhantes a primeira, com a desvantagem ainda de duas falhas num mesmo grupo não serem toleradas.

A terceira e a quarta classe são as que possuem as melhores características para serem aplicadas em ambientes tipo o da T-NODE. Deste modo, a seguir são apresentadas as principais características de algumas estratégias para construção de algoritmos pertencentes a estas duas últimas classes.

As técnicas que são apresentadas a seguir, apesar de terem sido desenvolvidas para processadores matriciais, aplicam-se perfeitamente a qualquer sistema com um grande número de elementos de processamento idênticos regularmente conectados, como é o caso da T-NODE.

4.1.1 Técnicas por Chaveamento de Processador

O chaveamento de processador é uma técnica de reconfiguração que busca reduzir o número de processadores redundantes em *hardware*, substituindo direta ou indiretamente somente o processador defeituoso. Existe uma abordagem que substitui o processador defeituoso por seu vizinho, que por sua vez é substituído por seu outro vizinho, e assim sucessivamente até alcançar um sobressalente da rede. As estratégias *fault-stealing* e *Full Use of Suitable Spares* (FUSS) são exemplos de estratégias que usam essa abordagem. Uma outra abordagem é

a que conecta e desconecta diretamente os módulos que compõem o *array*. Como exemplo desta estratégia tem-se a estratégia Diogenes [ROS83].

Como a T-NODE possui a característica de poder chavear diretamente seus nodos pela programação da rede de interconexão, semelhantemente a segunda abordagem, a seguir é apresentada a estratégia Diogenes, que chaveia diretamente um sobressalente.

4.1.1.1 Estratégia Diogenes

Nesta estratégia o conjunto de nodos é conectado por um mecanismo de chaveamento composto por chaves programáveis e um conjunto de linhas que passam por todos os nodos, inclusive os sobressalentes. Os nodos sobressalentes ficam desconectados e posicionados à esquerda dos nodos ativos. Sempre que um nodo falha, o sobressalente mais da direita é conectado, através das chaves programáveis, e o defeituoso desconectado do conjunto de linhas, resultando num deslocamento lógico de nodos.

Este procedimento de chaveamento permite que a estratégia Diogenes alcance 100% dos sobressalentes. A figura 4.1 mostra uma visão simplificada de como é implementada esta estratégia.

A parte mais importante da estratégia Diogenes é o projeto do mecanismo de barramento e das chaves. Segundo Chean [CHE90] e Dutt [DUT92], a vantagem desta estratégia é a possibilidade de implementar diferentes topologias (linearizando-as) e conseguir o uso ótimo dos sobressalentes.

Comparando esta estrutura com a da T-NODE, se vê que ambas possuem chaves programáveis que permitem alcançar 100% dos sobressalentes. A diferença é que na T-NODE as topologias não são linearizadas, porque a rede de interconexão faz o papel do conjunto de linhas e a possibilidade de chavear um processador com qualquer outro dispensa a necessidade do deslocamento lógico.

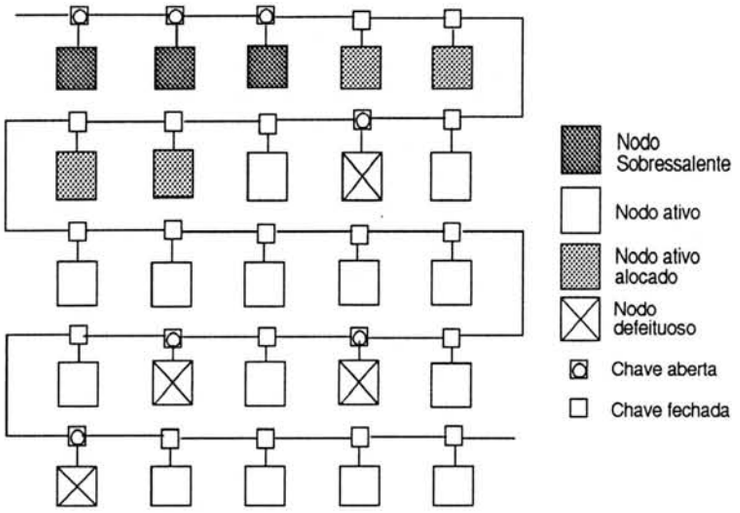

Figura 4.1: Implementação da estratégia Diogenes.

4.1.2 Técnicas por Redundância Temporal

A reconfiguração por redundância temporal não usa processadores sobressalentes. Esta técnica faz com que um processador da estrutura de *hardware* assuma, adicionalmente às suas, as funções do processador defeituoso. Entretanto, há casos em que as técnicas de redundância temporal podem ser usadas em conjunto com outras técnicas. Por exemplo, se um sistema opera com s processadores sobressalentes em *hardware* e d processadores falham, onde $d > s$, o(s) $d - s$ defeito(s) que excedem o número de processadores físicos podem ser reconfigurados sobre processadores lógicos, mantendo-se assim a funcionalidade do sistema, mas ocorrendo conseqüentemente uma degradação de desempenho.

Como exemplo deste tipo de estratégia pode-se citar a abordagem para o projeto de processadores *array* com degradação gradual sugerida por Fortes e Raghavendra [FOR85], onde estratégias de eliminação sucessiva de linhas (SRE) e eliminação alternativa de linhas e/ou colunas (ARCE) são usadas. Entretanto, em arquiteturas multiprocessadoras como a da T-NODE, que não têm o compromisso de manter exatamente a estrutura física, esta estratégia degrada muito o sistema.

4.2 Técnicas para Reconfiguração de Arquiteturas em Árvore

Uma arquitetura em árvore, assim como um *array*, é constituída de nodos e arcos, onde os nodos representam recursos (computadores, processadores, ou dispositivos de I/O) e os arcos representam os canais de comunicação entre os recursos.

Do ponto de vista redundância, existem duas abordagens para se conseguir tolerância a falhas em arquiteturas em árvores [DUT88]. A primeira é usar arcos redundantes para garantir a conectividade em caso de falha, mas por outro lado isto não garante a preservação da estrutura topológica do sistema, que é muito importante em alguns casos, como por exemplo, em aplicações de tempo real (veja exemplo de uma estrutura com somente arcos redundantes na figura 4.2). Nesta abordagem, após a detecção de um defeito, o sistema elimina o nodo defeituoso, passando a funcionar num modo degradado. A segunda abordagem é usar arcos e nodos redundantes, o que garante a conectividade e a estrutura do sistema após a reconfiguração (veja exemplo de uma estrutura com arcos e nodos redundantes na figura 4.3), ou seja, ocorrida uma falha, a arquitetura não só irá achar um novo caminho, eliminando a falha, mas também substituirá o nodo defeituoso por outro não defeituoso (redundante). Em ambas as abordagens, falhas nos arcos são tratadas como se fossem falhas nos nodos.

Na máquina T-NODE existem, virtualmente, arcos sobressalentes, pois a rede de interconexão é totalmente rearranjável, o que permite a alteração das conexões entre os nodos. Entretanto, para esta máquina não se pode garantir falhas nos arcos (na rede de interconexão), como nas abordagens citadas, sem alterar o *hardware*, porque uma falha na rede de interconexão pode levar o sistema todo a um colapso. Já falhas em nodos podem ser tratadas semelhantemente à maneira feita nas estratégias acima, ou seja, fazendo uso de redundância. Deste modo, sobrevém a necessidade de estabelecer-se uma restrição no presente traba-

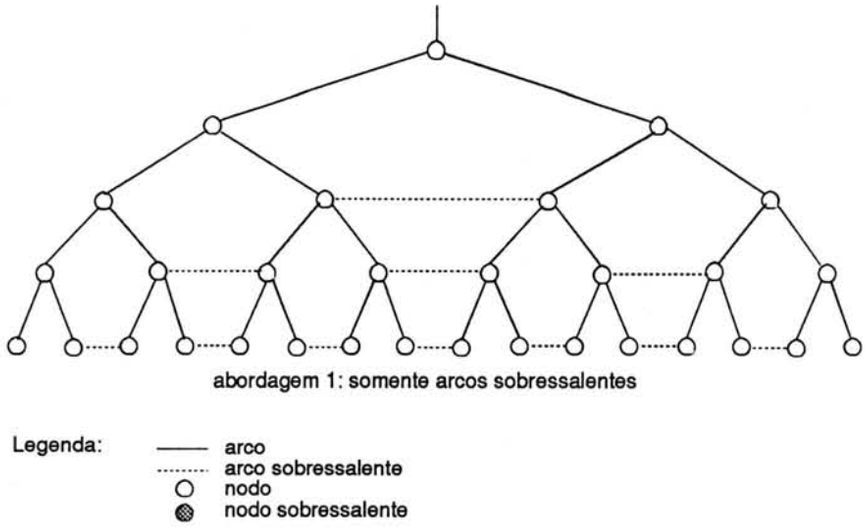

Figura 4.2: Estrutura para tolerar falhas em arquiteturas tipo árvore usando somente arcos redundantes.

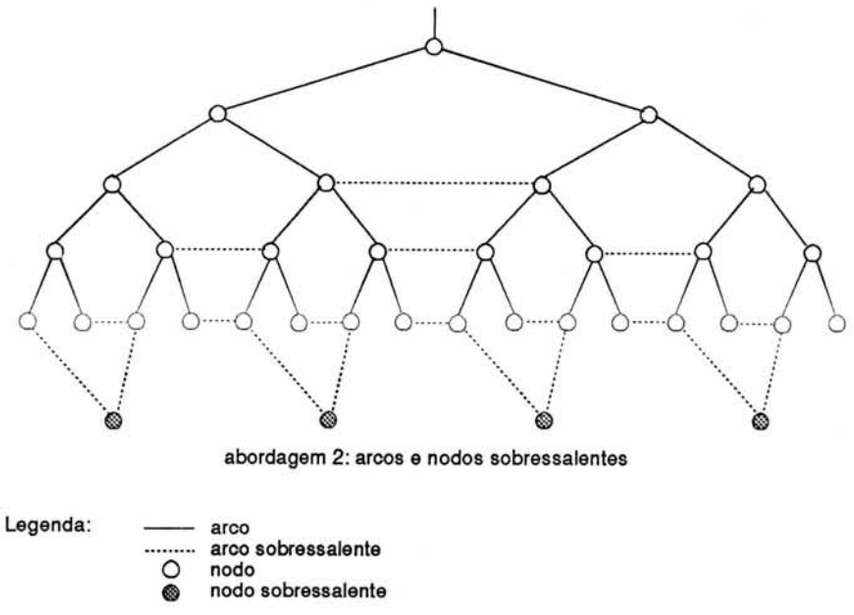

Figura 4.3: Estrutura para tolerar falhas em arquiteturas tipo árvore usando arcos e nodos redundantes.

lho, eliminando-se da classe de falhas as ocorridas na rede de interconexão. Do mesmo modo que na seção 4.1, a redundância tanto pode ser em *hardware* como temporal.

Várias técnicas para cobertura de falhas em árvores já foram desenvolvidas por diversos autores ([DUT90] [DUT88] [RAG84] [HAS85] [LOW87]). Dutt, em [DUT88] e [DUT90], propõe estratégias de projeto e reconfiguração para árvores simétricas não homogêneas (ASNH) tolerantes a s falhas, baseadas no modelo de cobertura de grafos introduzido por [HAY76]. Raghavendra, Avizienis e Ercegovac, em [RAG84], descrevem um método que suporta múltiplas falhas em árvores homogêneas. O método protege contra uma falha por nível da árvore, associando um nodo sobressalente para cada grupo de 2^i nodos para algum valor de i, onde i é o número de níveis da árvore, e protege contra i falhas simultâneas desde que elas estejam em diferentes níveis da ávore. A cobertura dada por este método necessita de um reserva por nível da árvore, onde o nível pode ser comparado com uma linha do *array*. É apresentada uma abordagem com redundância por *hardware* e outra com redundância temporal. Lowrie e Fuchs [LOW87] descrevem outra técnica para projeto de árvores tolerantes a falhas, em que um nodo sobressalente é associado às folhas de uma subárvore da árvore real, e também é compartilhado por duas subárvores. Esta técnica é chamada Tolerância a Falhas Orientada a Subárvores (TFOS).

Abaixo é analisada a estratégia proposta por Raghavendra, em [RAG84], a qual suporta múltiplas falhas em árvores homogêneas, pois apresenta características de controle da reconfiguração semelhantes as pretendidas para a T-NODE e a estratégia TFOS proposta por Lowrie, em [LOW87], pois tolera s falhas na árvore se existirem s processadores sobressalentes.

4.2.1 Tolerância a Falhas em Árvores Binárias Homogêneas

O esquema de tolerância a falhas proposto por Raghavendra [RAG84] protege cada nível de uma árvore binária contra falhas simples. Deste modo, o sistema pode tolerar múltiplos defeitos desde que eles ocorram em diferentes níveis da árvore.

Usando esta estratégia, existem duas maneiras de tolerar falhas em árvores: a primeira é usar um ou mais nodos sobressalentes por nível, com canais redundantes para manter a conectividade do nível com os sobressalentes; a segunda é proteger as custas de perda de desempenho. Ambas são analisadas brevemente a seguir.

4.2.1.1 Estratégia com Sobressalentes

Esta estratégia considera o uso de arcos e nodos redundantes para poder manter a estrutura inicial da árvore binária em caso de falhas. A cobertura de falhas é feita por nível da árvore, que a princípio suporta uma falha simples no nível e falhas múltiplas desde que em níveis distintos. A figura 4.4 mostra como fica a estrutura lógica de uma árvore com 5 níveis, na qual pode-se notar que a probabilidade de ocorrer falha nos níveis mais baixos é maior, porque o número de processadores destes níveis é maior. Para minimizar este problema, pode-se adicionar mais do que um nodo sobressalente nestes níveis, a fim de assegurar a cobertura de mais do que uma falha.

A estrutura de árvore binária redundante, com sobressalentes, pode ser considerada como um sistema tolerante a falhas que consiste de uma série de subsistemas homogêneos S_i, onde $i = 1, 2, 3, \ldots, N$. Deste modo, o i-ésimo nível da árvore pode ser encarado como um subsistema homogêneo com 2^i nodos ativos e um sobressalente.

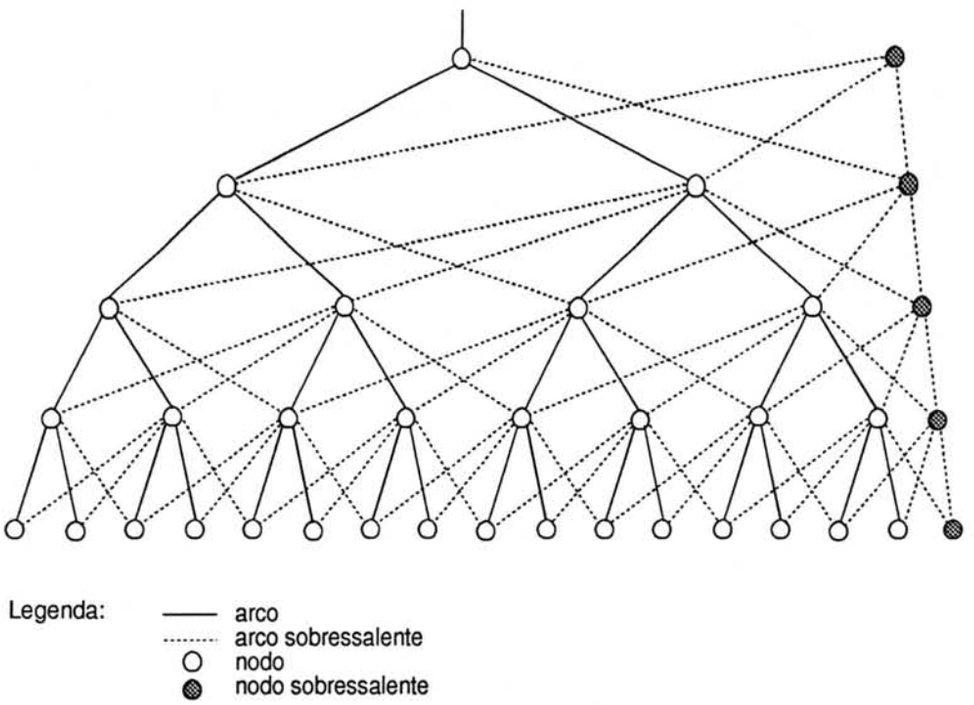

Figura 4.4: Estrutura em árvore de cinco níveis com sobressalentes.

Como o número de canais redundantes inseridos por esta estratégia é relativamente alto, em relação aos ativos, os autores utilizaram duas redes *decoupling*, que são redes programáveis usadas para fazer deslocamentos laterais como mostra a figura 4.5, com a intenção de facilitar a integração da rede. Quando um processador falha, todos os canais dos processadores à direita dele são reajustados para os vizinhos da direita, como mostra a figura 4.6. Quem executa a reconfiguração é um processador monitor, que resolve tanto a reprogramação das chaves como os procedimentos de *rollback* e *recovery*, após ser informado de um defeito. Analisando esta estratégia de reconfiguração, observa-se que ela contém características que podem ser mapeadas para o caso da T-NODE como, por exemplo, o processador monitor, que detém o controle de programação da rede de interconexão, pode ser comparado ao transputer controlador, e a rede que também é programável e permite trajetos alternativos entre processadores diferentes, mas com a desvantagem de não ser totalmente rearranjável como a da T-NODE.

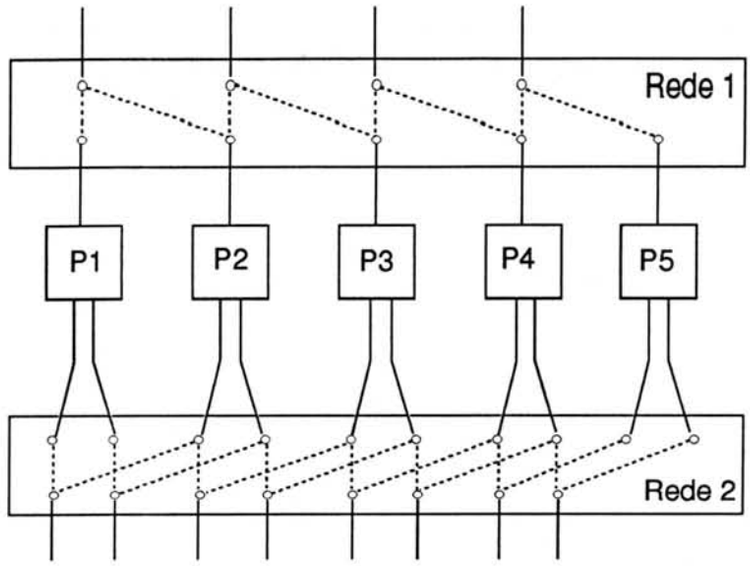

Figura 4.5: Organização do subsistema de nível 2 usando redes *decoupling*.

4.2.1.2 Estratégia com Degradação de Desempenho

Nesta estratégia, o sistema opera com uma degradação de desempenho quando há um nodo defeituoso. A redundância é temporal e feita pelo vizinho direito de cada nodo. As únicas redundâncias físicas empregadas são a de um nodo sobressalente para o nodo raiz da árvore, pois este nodo não possui vizinho, e de canais extras para cada um dos outros nodos, como mostrado na figura 4.7, para que o nodo defeituoso possa ser isolado.

Quando um nodo falha, seu vizinho assume suas tarefas, ficando conseqüentemente mais carregado e contribuindo para a degradação de desempenho do sistema. Esta é a grande desvantagem desta estratégia, principalmente quando ocorrem múltiplas falhas em nodos adjacentes, pois um dos vizinhos assume a carga dos dois nodos falhos. Entretanto, os autores assumem que a probabilidade de tal evento ocorrer é pequena quando se consideram falhas independentes.

O procedimento de reconfiguração e recuperação do sistema envolve uma etapa de informação ao nodo vizinho à falha, notificando-o de que ele deve resolver tarefas adicionais e usar os canais sobressalentes para comunicação com seus novos parentes.

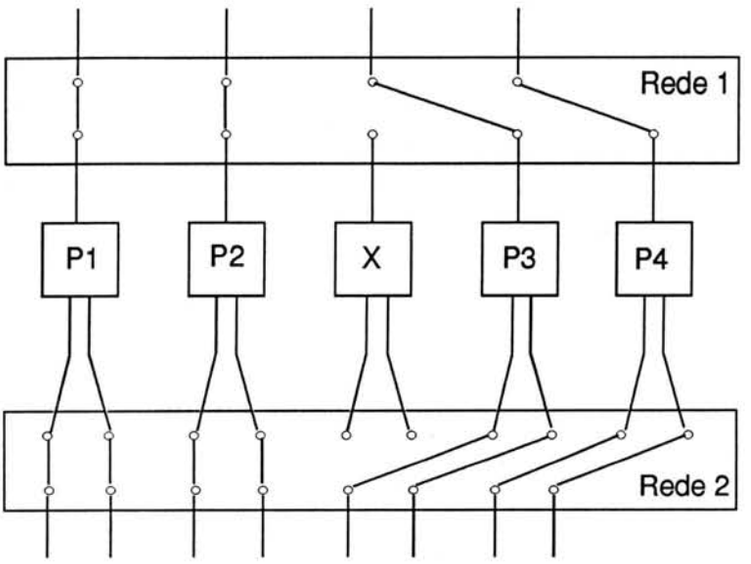

Figura 4.6: Reconfiguração do subsistema de nível 2 após o processador 3 falhar.

Analisando esta estratégia, verifica-se que um defeito perto do nodo raiz causa uma degradação bem maior do que um defeito perto dos nodos folha. Deste modo, uma boa solução é usar nodos sobressalentes e degradação de desempenho nos níveis mais altos (raiz), e usar somente degradação de desempenho nos níveis mais baixos (folhas). Quando se pretende manter a velocidade de resposta da computação, então a opção é usar a estratégia com sobressalentes.

4.2.2 Tolerância a Falhas Orientada a Subárvores (TFOS)

O método desenvolvido por Raghavendra para tolerar falhas em árvores binárias homogêneas, visto na seção anterior, baseia-se na premissa de que os arcos redundantes garantem a conectividade da árvore, ou topologia, após a ocorrência de falhas [RAG84]. Contudo, este método possui uma alta redundância de arcos na estrutura, podendo chegar a 200%.

Com o objetivo de reduzir o nível de redundância e fazer uma utilização eficiente dos recursos disponíveis, ou seja, tolerar s processadores defeituosos se

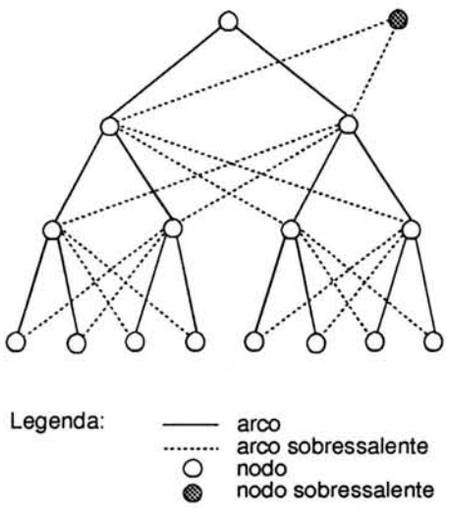

Figura 4.7: Esquema com degradação de desempenho.

existirem s processadores sobressalentes na arquitetura, Lowrie e Fuchs [LOW87] propuseram uma estratégia de Tolerância a Falhas Orientada a Subárvores (TFOS).

Dado que uma árvore contém $i + 1$ níveis, onde o nível da raiz é 0 e o nível das folhas é i, e que o nodo raiz seja numerado como 1, o filho esquerdo de algum nodo n como $2n$ e o filho direito como $2n + 1$, define-se uma subárvore como uma árvore de j níveis, com $j < i$, contida na árvore original tal que as folhas da subárvore correspondem às folhas da árvore original.

A estratégia TFOS conecta processadores sobressalentes nas folhas da árvore, compartilhando-os entre subárvores adjacentes[1], garantindo com isto uma flexibilidade considerável na reconfiguração. Canais sobressalentes também são usados na conexão entre processadores com o objetivo de manter a topologia da árvore após a ocorrência de falha.

A alocação de nodos sobressalentes na árvore, segue um algoritmo descrito em [LOW87], que define o número de nodos sobressalentes em 2^c, onde

[1]Subárvores adjacentes são aquelas conectadas por um canal de comunicação redundante ou não.

c é um inteiro entre $0 \leq c \leq i-1$. A figura 4.8.a mostra um exemplo de alocação quando $c = 2$ e $i = 4$. Uma subárvore com folhas numeradas de $x+k2^{iSAS}$ à $x+(k+1)2^{iSAS}-1$, onde x é a folha mais da esquerda da raiz e $0 \leq k \leq 2^c - 1$ é referida como uma subárvore sobressalente, ou SAS. Cada SAS tem um sobressalente associado que é adjacente à sua folha mais da direita. O sobressalente adjacente à folha mais à esquerda da SAS é referido como sobressalente não associado.

A reconfiguração baseia-se numa técnica de deslocamentos virtuais entre diferentes níveis da árvore implementada por um *bypass* do pai do defeituoso para um dos filhos do defeituoso, até alcançar os nodos sobressalentes conectados às folhas da árvore. A figura 4.8.b mostra um exemplo de reconfiguração usando esta estratégia. Nos níveis não folha da árvore, o defeito em um nodo resulta em sua eliminação, ou seja, na conexão direta entre seu pai e um de seus filhos. O filho do nodo defeituoso assume as tarefas alocadas a seu pai (defeituoso), e conseqüentemente, um outro processador tem que ser encontrado para assumir as tarefas do filho. De maneira similar, um dos filhos do filho assume suas responsabilidades, e assim sucessivamente. Este deslocamento virtual continua até um sobressalente ser configurado para assumir as tarefas de um nodo folha.

Embora este método tenha sido desenvolvido para arquiteturas na forma de árvores, pode-se verificar a possibilidade de suportar no sistema tantas falhas quanto for o número de processadores sobressalentes. Outra característica desta estratégia é que processadores sobressalentes podem ser usados para cobrir falhas em qualquer ponto da rede, embora esta estratégia não cubra todos os grupos de falhas que podem ocorrer. Para o caso da T-NODE isto não é tão significativo, porque todos os nodos podem ser conectados a qualquer nodo da rede.

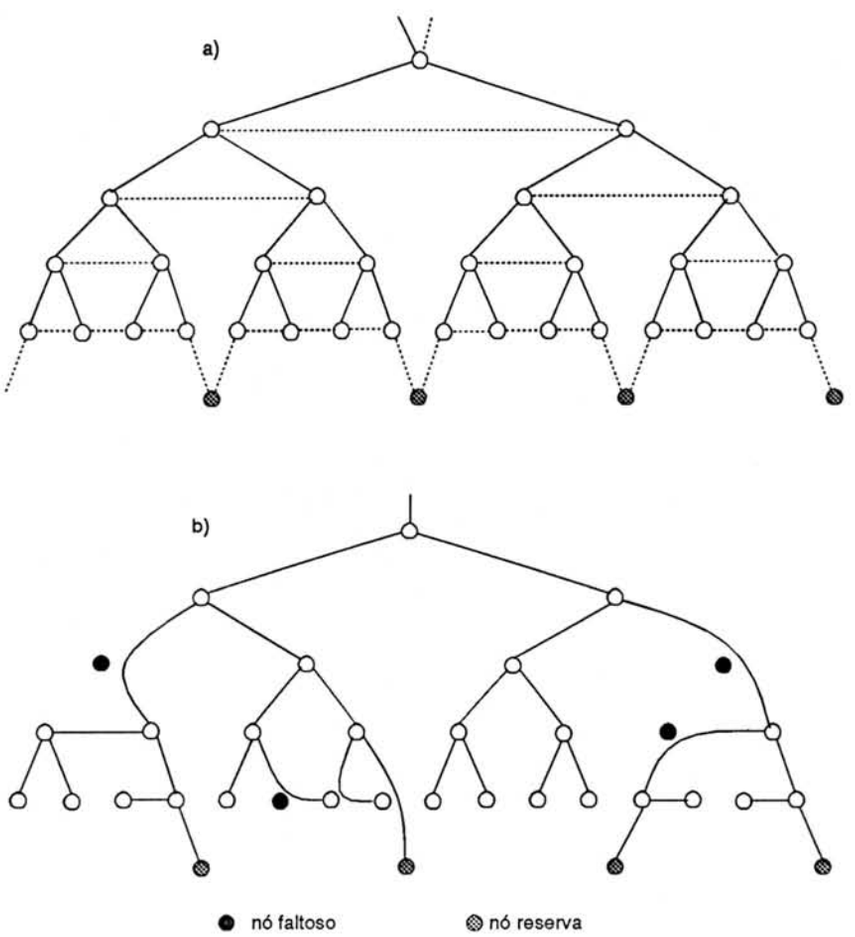

Figura 4.8: a) Arquitetura TFOS com $c = 2$ e $i = 4$, b) reconfiguração com quatro falhas.

5 ALGORITMO DE RECONFIGURAÇÃO PROPOSTO

No capítulo anterior foi visto que um sistema contendo um módulo defeituoso pode prosseguir em funcionamento pela retirada do defeituoso através de reconfiguração, e pela atribuição das suas tarefas a outro módulo não defeituoso, que poderá ser um módulo redundante em *hardware* (sobressalente) ou um outro módulo que já fazia parte da topologia utilizada.

Quando são usados módulos sobressalentes em arquiteturas totalmente rearranjáveis, é possível garantir que, após a ocorrência de falhas, o sistema manterá a mesma estrutura topológica básica, e que seu desempenho não será degradado dentro dos parâmetros previstos. Entretanto, quando não há sobressalentes, um dos módulos já utilizados terá que assumir as tarefas do módulo defeituoso adicionalmente às suas, o que degrada o desempenho do sistema e aumenta o *overhead* de comunicação, visto que a estrutura física inicial não é mantida.

Neste capítulo, é apresentado um procedimento para detecção de falhas e reconfiguração da máquina T-NODE que utiliza sua característica de total reconfigurabilidade (conforme visto na seção 2.2.2) para eliminar possíveis falhas no sistema através do isolamento e substituição do módulo defeituoso. Como o chaveamento é feito a nível de processadores com suas respectivas memórias locais, pois a T-NODE é uma máquina multiprocessadora fracamente acoplada, tanto falhas no processador como em sua memória local são consideradas como falhas no módulo, o qual corresponde ao par "processador-memória".

Para o desenvolvimento deste trabalho considerou-se algumas hipóteses quanto à cobertura de falhas e aos parâmetros de aplicação, as quais são vistas a seguir.

5.1 Hipóteses Consideradas

5.1.1 Quanto à Cobertura de Falhas

Por hipótese, a máquina T-NODE pode tolerar:

- s falhas sem degradação de desempenho do sistema, onde s é o número de módulos sobressalentes em *hardware*; ou
- $n-1$ falhas com degradação, onde n é o número total de processadores de trabalho, incluindo os s processadores sobressalentes, caso estes existam, sendo que $(n-1) > s \geq 0$.

Para satisfazer a primeira hipótese, o custo adicional será o de necessariamente ter s módulos sobressalentes no sistema, o que pode restringir o número de módulos disponíveis ao usuário, uma vez que ele não terá acesso a todos os processadores da máquina.

Já na segunda hipótese, o sistema poderá tolerar $n-1$ falhas, mas as custas de degradação no desempenho do sistema, uma vez que sua topologia e balanceamento de carga inicial não serão mantidos.

O algoritmo de reconfiguração aqui proposto implementa o uso de uma técnica de redundância mista, ou seja, com e sem o uso de sobressalentes, sobre o nodo básico de uma máquina T-NODE. Deste modo, consegue-se tolerar as primeiras s falhas sem degradação, pois a reconfiguração neste caso faz uso dos s módulos sobressalentes, e as próximas $(n-1)-s$ falhas com degradação, pois nesta situação o sistema já não mantém sua estrutura inicial e passa a operar com menos módulos físicos do que lógicos.

5.1.2 Quanto aos Parâmetros de Aplicação

Também por hipótese, para o algoritmo aqui proposto:

- a máquina opera num modo mono-usuário; e
- é o usuário quem define, no momento de inicialização do sistema, o número de módulos sobressalentes que a máquina vai usar para tolerar falhas.

A hipótese de operação num modo mono-usuário é aqui considerada porque esta modalidade de operação corresponde a generalidade das situações nas quais se manipulam aplicações críticas. A operação em modo multi-usuário não é considerada neste trabalho, mas pode ser tratada de maneira similar. Todavia, ela não pode ser associada à hipótese de que o usuário será o responsável pela definição do número de sobressalentes, mas sim a de ser o administrador da rede o responsável pela definição do número de módulos sobressalentes. Entretanto, a especificação deste número é mais complexa, pois dependerá das características do conjunto de aplicações na rede.

Do ponto de vista de redundância, a restrição ao número de processadores disponíveis no sistema terá sua influência determinada pelo tipo de topologia escolhida pelo usuário. Para avaliar esta influência, define-se como taxa de ociosidade aparente (T_{oc-ap}) a relação entre o número de sobressalentes (s) e o número total de processadores de trabalho da máquina (n), inclusive os s sobressalentes, e como taxa de ociosidade real (T_{oc-re}) a relação entre o número de processadores não utilizados (p) e o número total de processadores de trabalho da máquina (n). A taxa de ociosidade aparente analisa a influência do uso de processadores sobressalentes, considerando que todos os n processadores de trabalho poderiam estar disponíveis, mas a taxa de ociosidade real analisa a influência, considerando o uso real dos processadores em cada caso.

Por exemplo, considerando que a máquina esteja operando com uma configuração de 16 processadores, para poder tolerar uma falha sem degradar o desempenho do sistema, precisa-se utilizar um sobressalente ($s = 1$), o que produz uma taxa de ociosidade aparente de:

$$T_{oc-ap} = s/n = 1/16$$

Entretanto, se o usuário deseja trabalhar com uma árvore binária de 4 níveis (incluindo o nível raiz), o número de processadores utilizados pela topologia (u) será de $u = 2^4 - 1 = 15$, o que faz com que um dos 16 processadores fique ocioso, podendo ser utilizado como sobressalente. Deste modo, tem-se:

$$Toc - re = 0$$

pois:

$$p = n - u - s = 16 - 15 - 1 = 0$$

o que indica um custo adicional igual a zero para tolerar uma falha. Mas se o usuário desejar trabalhar com uma matriz quadrada de processadores com dimensão 4 x 4, o número de processadores utilizados pela topologia será $u = 4 * 4 = 16$. Logo, neste caso, o sobressalente impede o uso de uma matriz 4 x 4, pois limita a 15 o número de processadores disponíveis, permitindo somente o uso de uma matriz 3 x 3. Isto faz com que o custo para tolerar uma falha seja o mesmo que o de tolerar 7 falhas, pois deixará de utilizar 6 processadores para ter um sobressalente, fazendo a taxa de ociosidade real ser de:

$$T_{oc-re} = (n - u - s)/n = (16 - 9 - 1)/16 = 6/16$$

o que parece inaceitável para tolerar uma falha, do ponto de vista da maior parte das aplicações, já que esta taxa foi provocada pelo número de sobressalentes escolhido.

Deste modo, como a topologia adotada pelo usuário tem uma forte relação com o número de módulos sobressalentes, a hipótese de permitir que o próprio usuário defina, no momento da inicialização do sistema, o número de módulos sobressalentes que ele deseja, é justificada pelo reconhecimento de que o usuário é quem possui as melhores condições de definir, em função da sua aplicação, qual é o preço possível de ser suportado:

- dispor de menor número de módulos para execução da aplicação; ou
- perder desempenho apenas sob ocorrência de falhas.

5.2 Estratégia do Algoritmo

Pode-se dividir o algoritmo em duas etapas distintas: uma de testes dos módulos (de trabalho, servidores e controladores), que faz a detecção de possíveis falhas nos módulos, e uma de reconfiguração, que isola as falhas e recupera o sistema.

A etapa de testes é resolvida por um conjunto de processos idênticos (processos testadores) executados em paralelo, um em cada módulo da máquina. A existência de vários processos testadores executados em paralelo gera a necessidade de implementação de um processo (o supervisor) para supervisionar o estado dos vários módulos e disparar, quando detectado um erro em quaisquer dos módulos, a etapa de reconfiguração, que é resolvida por um outro processo (o reconfigurador).

Do ponto de vista segurança do sistema, o processo supervisor deve estar distribuído ou replicado por vários nodos do sistema. Entretanto, dada a característica centralizada da máquina T-NODE (conforme visto no capítulo 2), onde o processador controlador é que comanda o barramento de controle e a programação da rede de interconexão entre processadores, optou-se por implementar um processo supervisor centralizado, pois torna a estratégia mais simples e eficiente em termos de desempenho não comprometendo a segurança do sistema. Uma falha no processador controlador leva a máquina a um estado defeituoso independente da maneira em que o processo supervisor for implementado.

A monitoração dos processos testadores pelo processo supervisor pode ser feita de três maneiras distintas:

1. solicitando periodicamente informações de estado a cada módulo do sistema e associando um tempo máximo (*timeout*) para resposta, fora do qual o módulo é considerado defeituoso;

2. esperando que cada módulo informe seu estado dentro de um tempo máximo de espera, fora do qual o módulo é considerado defeituoso; e

3. esperando que cada módulo informe seu estado dentro de um tempo máximo de espera e quando este extrapolar, solicitar as informações de estado ao módulo, associando um novo tempo máximo para resposta fora do qual o módulo é considerado defeituoso.

A diferença básica entre as três maneiras de implementação do processo supervisor é que a primeira possibilita que o supervisor determine o momento que o processo testador deve informar o estado do módulo, enquanto na segunda e terceira é o testador quem estabelece este momento. A segunda e terceira diferem pela não confirmação e confirmação, respectivamente, de um estado defeituoso ocorrido por *timeout*. Entretanto, a confirmação feita na terceira maneira pode ser encarada como um simples aumento no *timeout*, o que não é necessário se o

primeiro *timeout* for bem dimensionado. Intrinsico às três maneiras, o problema principal é a determinação do *timeout* a ser considerado.

Neste trabalho optou-se por utilizar a segunda maneira de monitoração dos testadores, por entender-se ser esta a que estabelece o maior grau de independência entre processos, dado que a comunicação entre testadores e supervisor se faz num único sentido (testador -> supervisor), e por adaptar-se perfeitamente ao modelo de comunicação adotado pelo *transputer*.

As próximas seções apresentam e discutem os três processos que compõem o algoritmo.

5.2.1 Processo Testador

Este processo é o responsável pela detecção de erros nos diversos módulos da máquina. Deste modo, existem tantos processos testadores quantos forem os módulos na rede, sendo que todos operam em módulos distintos, garantindo a cobertura individual. Este processo executa uma bateria de testes rápidos, para não prejudicar significativamente o desempenho do sistema. Esses testes verificam as condições do módulo. A duração desta bateria de testes poderá variar de acordo com o nível de cobertura que se deseja, sendo que quanto mais detalhada ela for maior será a degradação de desempenho inserida no sistema, pois esse processo concorre com os processos do usuário num mesmo módulo.

Os processos testadores são programados para executarem somente em períodos de tempo pré-determinados, pois como eles executam concorrentemente com os processos do usuário e do sistema operacional (nos módulos de trabalho), quanto mais tempo eles permanecerem ativos maior será a influência destes no tempo total do sistema (influência no desempenho).

Quando um erro é detectado por um dos testes pertencentes a este processo, o teste é refeito antes de informar ao processo supervisor de que o módulo está defeituoso. Este procedimento visa eliminar a possibilidade do testador detectar uma falha transitória e informar ao processo supervisor que o módulo está defeituoso, pois os processos supervisor e reconfigurador reconhecem todas as falhas detectadas como sendo falhas permanentes (*fail-stop*).

Neste trabalho, considera-se somente nodos defeituosos e não nodos e arcos, como nos algoritmos vistos no capítulo anterior. Esta escolha foi adotada porque a rede de interconexão é única e qualquer defeito nela leva o sistema todo a uma condição defeituosa. Este problema pode ser resolvido pela substituição da rede de interconexão atual por uma tolerante a falhas que mantenha as características de reconfigurabilidade e não-bloqueio.

Na figura 5.1 é apresentada a estrutura funcional do processo testador. Este processo executa em alta prioridade. Sua função é testar todas as unidades funcionais do *transputer* pela execução de um conjunto de instruções. Se nenhum erro foi detectado o processo envia uma mensagem de "ok" para o supervisor e se bloqueia por um tempo especificado, retornando a testar as unidades funcionais após o desbloqueio. Quando uma falha é detectada, uma mensagem de "não ok" é enviada ao supervisor e o processo termina. Este término garante que sempre que ocorrer a detecção de uma falha pelo testador, o supervisor irá perceber, pois mesmo que uma mensagem de "não ok" enviada ao supervisor seja corrompida para uma mensagem de "ok", esta será a última, e na escuta da próxima mensagem proveniente do processador defeituoso ocorrerá *timeout*.

5.2.2 Processo Supervisor

Este processo é executado sobre o *transputer* controlador e é o responsável pela monitoração das mensagens enviadas pelos processos testadores,

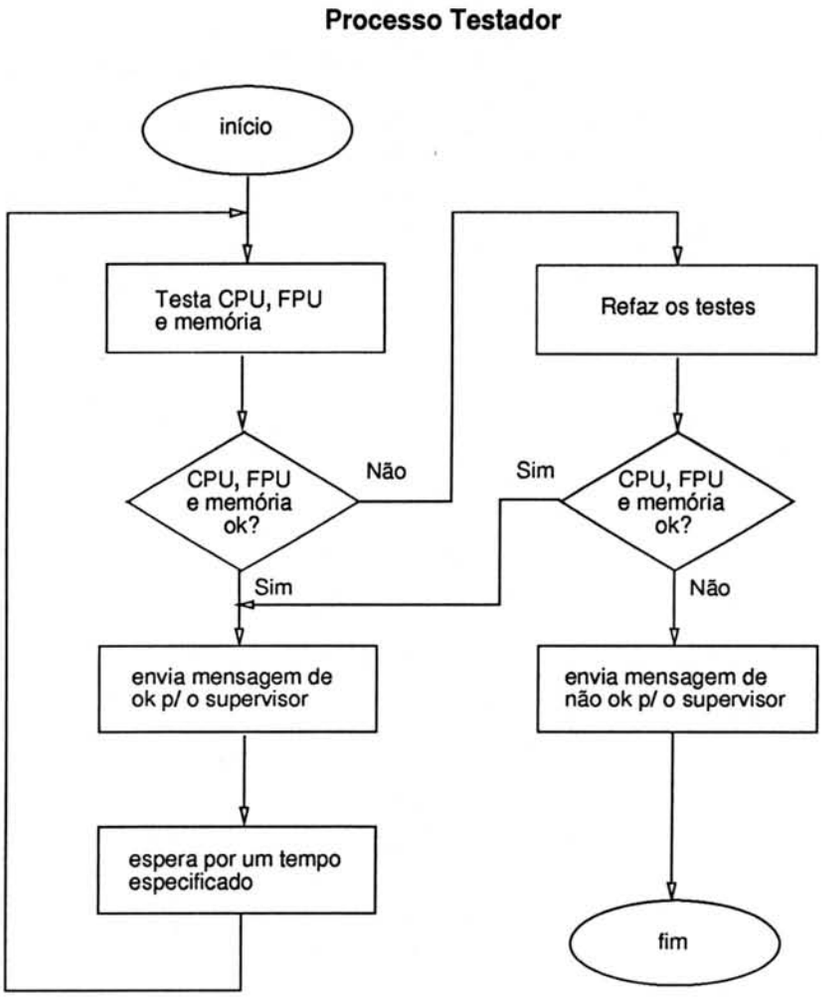

Figura 5.1: Fluxograma do processo testador.

os quais periodicamente informam o estado de seus módulos, e pelo envio ao processo reconfigurador da identificação do módulo defeituoso. A monitoração dos processos testadores é feita pela espera de mensagens de estado sobre vários canais de entrada (um para cada módulo).

Como a comunicação dos módulos também pode ser afetada pela ocorrência de uma falha, deixando-os incomunicáveis, a cada canal de entrada do supervisor é associado um tempo máximo de espera (*timeout*), que deve ser estabelecido considerando a periodicidade de execução do processo testador, a duração dos testes, as estatísticas do número e tipo de processos que concorrem pela CPU, e os tempos de roteamento de mensagens através da rede de processadores. Este *timeout* garante a detecção de falhas que impedem o envio de

mensagens de estado pelo processo testador localizado no módulo defeituoso. O atraso ou bloqueio da mensagem pode ocorrer principalmente por dois motivos:

- ocorrência de um defeito no módulo que impeça o envio de mensagens; ou
- congestionamento de processos na fila do escalonador esperando para serem executados.

No primeiro caso, a ocorrência de *timeout* é inevitável, mas no segundo caso deve ser evitada sua ocorrência, pois o processador será visto como defeituoso pelo reconfigurador. Uma maneira de diminuir o risco de ocorrência do segundo caso é executar o processo testador em alta prioridade, pois nesta prioridade os processos, preemptam os de baixa prioridade e não são desescalonados nem por outros processos de alta prioridade (conforme explicado na seção 2.1.2). Estas características fazem com que o processo testador seja executado o mais rápido possível e que durante a execução não seja desescalonado. Uma situação possível é de a fila de processos de alta prioridade estar congestionada ocasionando a ocorrência do *timeout*. Entretanto, a probabilidade de que isto ocorra é baixa, dado que o estabelecimento do *timeout* leva em consideração o número e tipo de processos que concorrem pela CPU.

A mensagem que o processo supervisor recebe do testador pode ter três formas distintas:

- mensagem de "ok";
- mensagem de "não ok"; ou
- mensagem contendo um valor qualquer decorrente da manipulação causada por um defeito.

Qualquer mensagem diferente de "ok" é tratada pelo supervisor como uma mensagem de "não ok", indicando um defeito no módulo correspondente.

A identificação do módulo defeituoso é resolvida implicitamente pelo processo supervisor, pois para cada processo testador existe um canal de entrada associado. Isto também acontece para as entradas de *timeout*.

A figura 5.2 apresenta o fluxograma que descreve a estrutura funcional do processo supervisor. Este processo baseia-se no construtor alternativo (ALT) proposto por Hoare [HOA78] e utilizado na linguagem de programação OCCAM2 [INM88b]. Este construtor espera um dos vários canais de entrada associados a ele tornar-se pronto. O primeiro canal de entrada que se tornar pronto é atendido e o conjunto de processos associados ao canal é executado. No início do fluxograma é utilizado a representação ALT para simbolizar a execução alternativa entre os diferentes canais de escuta dos testadores. A cada canal com um testador está associado através de outro contrutor alternativo (ALT1 até ALTn) o canal de tempo (*timeout*). Na figura 5.2, abaixo da primeira opção, ALT1, encontra-se um conjunto de blocos que são replicados para cada ALTx.

5.2.3 Processo Reconfigurador

O processo reconfigurador, responsável pela reprogramação das chaves, é disparado simultaneamente aos processos testadores e supervisor, mas fica bloqueado (como processo inativo) à espera de uma mensagem do processo supervisor.

O bloqueio do processo, pela espera de uma mensagem, é possível porque o modelo OCCAM (abordado na seção 2.1.1.1) estabelece um protocolo *rendez-vous* entre a entrada e a saída de um canal. Deste modo, o recebimento de mensagem do supervisor, decorrente da detecção de falha em um dos módulos, desbloqueia este processo para que a reconfiguração do sistema possa ser feita. O conteúdo desta mensagem é o número do módulo que está defeituoso, o que agiliza o procedimento de identificação.

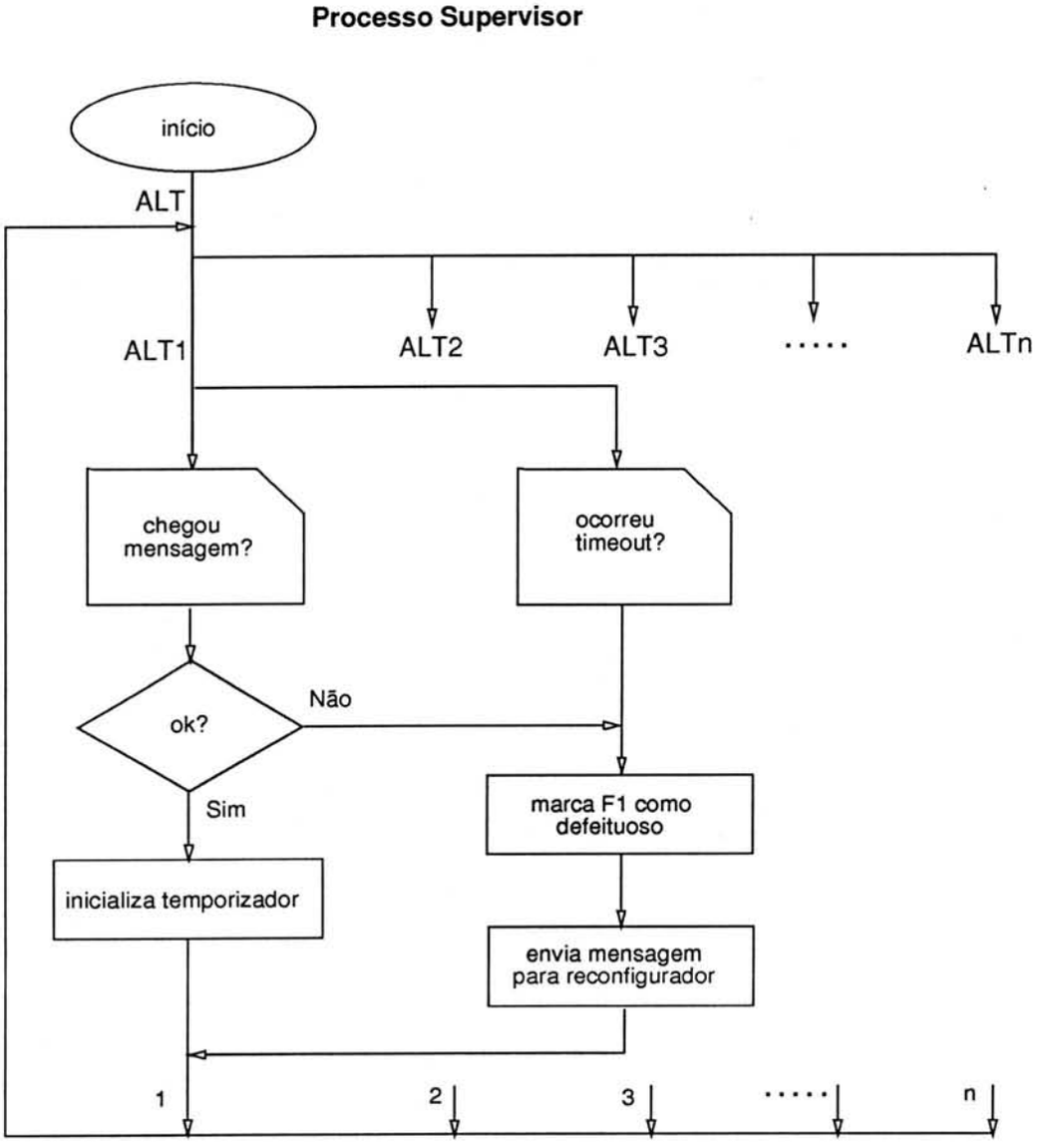

Figura 5.2: Fluxograma do processo supervisor.

Se a falha detectada for no controlador ou em um dos servidores, este processo relata ao usuário que houve uma falha fatal e que não pode recuperar o sistema através de reconfiguração modular. Após enviar esta informação o processo avisa o sistema operacional que o processamento deve terminar.

Ao invés disto, se a falha detectada for em um dos módulos de trabalho da máquina, este é isolado e substituído por um outro sobressalente, caso exista, ou suas tarefas atribuídas a um dos módulos ativos (redundância temporal). Segundo Horowitz e Despain, citado por [DUT88], a preocupação que existe, em

vários algoritmos de reconfiguração (dentre os quais os abordados no capítulo 4), de alcançar o maior número de sobressalentes, é resolvido na maioria dos casos pelo adição de arcos redundantes, o que garante a conectividade na presença de falhas. Entretanto, no caso da máquina T-NODE o uso de arcos redundantes para manter a conectividade não é necessário porque a rede de interconexão é totalmente rearranjável através de reprogramação (conforme visto no capítulo 2). Esta característica faz com que um módulo sobressalente qualquer possa ser usado para substituir qualquer módulo de trabalho defeituoso do sistema, a um custo incremental nulo.

De acordo com as características da máquina, a reprogramação da rede de interconexão pode ser feita estática ou dinamicamente. Neste trabalho considera-se somente a reprogramação estática, visto que o suporte à dinamicidade da máquina ainda não está totalmente resolvido. No caso dinâmico problemas de sincronização entre processos e validade de uma recuperação a partir de estados intermediários do processamento serão freqüentes, aumentando consideravelmente a complexidade do algoritmo pela necessidade do uso de técnicas de recuperação baseadas em pontos de recuperação intermediários que possuam um estado consistente.

Caso existam módulos sobressalentes, o reconfigurador substitui o módulo defeituoso pelo sobressalente, modificando as conexões entre módulos através da reprogramação da rede de interconexão e da solicitação ao sistema operacional de uma reinicialização da aplicação do usuário. Este procedimento mantém a mesma topologia inicial e por isso não degrada o desempenho do sistema. As conexões lógicas são definidas pelo mapa de recursos, que descreve a topologia pretendida pelo usuário, e as conexões físicas são definidas pelo casamento do mapa de recursos com o arquivo *snconfig.src* que descreve o *hardware* da máquina (conforme visto no capítulo 3).

Caso não existam módulos sobressalentes disponíveis para serem usados na recuperação do sistema, o algoritmo testa se o número de módulos de

trabalho é maior do que um, pois caso contrário não há como recuperar o sistema pelo uso de reconfiguração porque não existem mais módulos redundantes. Caso a resposta seja positiva, o processo isola o módulo defeituoso desconectando-o, e solicita ao sistema operacional a reinicialização da aplicação, o que fará com que as tarefas do processador defeituoso sejam atribuídas a um outro módulo de trabalho ativo não defeituoso. Este procedimento implica na perda de desempenho do sistema após sua reinicialização por dois motivos: sobrecarga no módulo que recebe as tarefas do defeituoso, pois as acumula com as suas tarefas, e o aumento no *overhead* de comunicação causado pela perda de topologia. Caso o sistema operacional possua um dispositivo de balanceamento de carga dinâmico, o problema da sobrecarga é automaticamente minimizado, mas caso contrário, podem ser implementadas verificações no balanceamento de carga da rede, a fim de atribuir as tarefas do defeituoso a um módulo com pouca carga, e verificações do aumento de tráfego causado por novas hipóteses de topologias. Entretanto, neste trabalho não são analisados procedimentos para implementar estas minimizações na degradação do sistema.

Após o isolamento da falha através da substituição do módulo defeituoso ou da degradação da topologia, o processo reconfigurador sempre solicita ao sistema operacional que a aplicação do usuário e o conseqüente reestabelecimento do sistema seja feito. Este procedimento faz com que todo o processamento da aplicação já executado até o momento seja perdido, ou seja, o processamento é retomado do ponto inicial.

Do ponto de vista desempenho, para aplicações não críticas, procedimentos para recuperação a partir de pontos intermediários são mais onerosos do que procedimentos para recuperação a partir do ponto inicial. Isto porque a degradação de desempenho em tempo de execução inserida pelo primeiro é maior, e porque a probabilidade de ocorrência de falha é pequena, justificando-se o seu uso somente em aplicações mais críticas (geralmente sistemas de tempo real).

Com a reinicialização completa do sistema, que corresponde a situação aqui considerada, o fato da rede de interconexão do T-NODE ser totalmente rearranjável é suficiente para garantir as novas conexões da rede, após a troca do módulo defeituoso pelo sobressalente. Entretanto, para casos em que a recuperação possa ocorrer a partir de um ponto intermediário, sem reinicializar completamente o sistema, é importante considerar também os casos de bloqueio da rede no MEGANODE quando estiverem sendo definidos os novos canais entre módulos.

A figura 5.3 mostra como o processo reconfigurador está estruturado funcionalmente.

5.2.4 Posicionamento dos Processos

Como visto, o algoritmo de reconfiguração é composto por três processos distintos (testador, supervisor e reconfigurador), e que sua estrutura é centralizada sobre o *transputer* controlador. A figura 5.4 mostra como estes processos estão distribuídos pelos diversos módulos do nodo básico da T-NODE.

Os processos testadores encontram-se em todos os módulos do nodo, sendo que sobre os servidores e controlador sua função é somente detectar erros para serem informados ao usuário de que houve uma falha no módulo, pois não há como realocar tarefas destes módulos a outros, devido à maneira pela qual o *hardware* da máquina foi construído.

O processo supervisor e reconfigurador são únicos e localizados sobre o controlador que, por hipótese, não pode falhar, pois neste caso leva o sistema todo a um estado defeituoso irreparável por *software*.

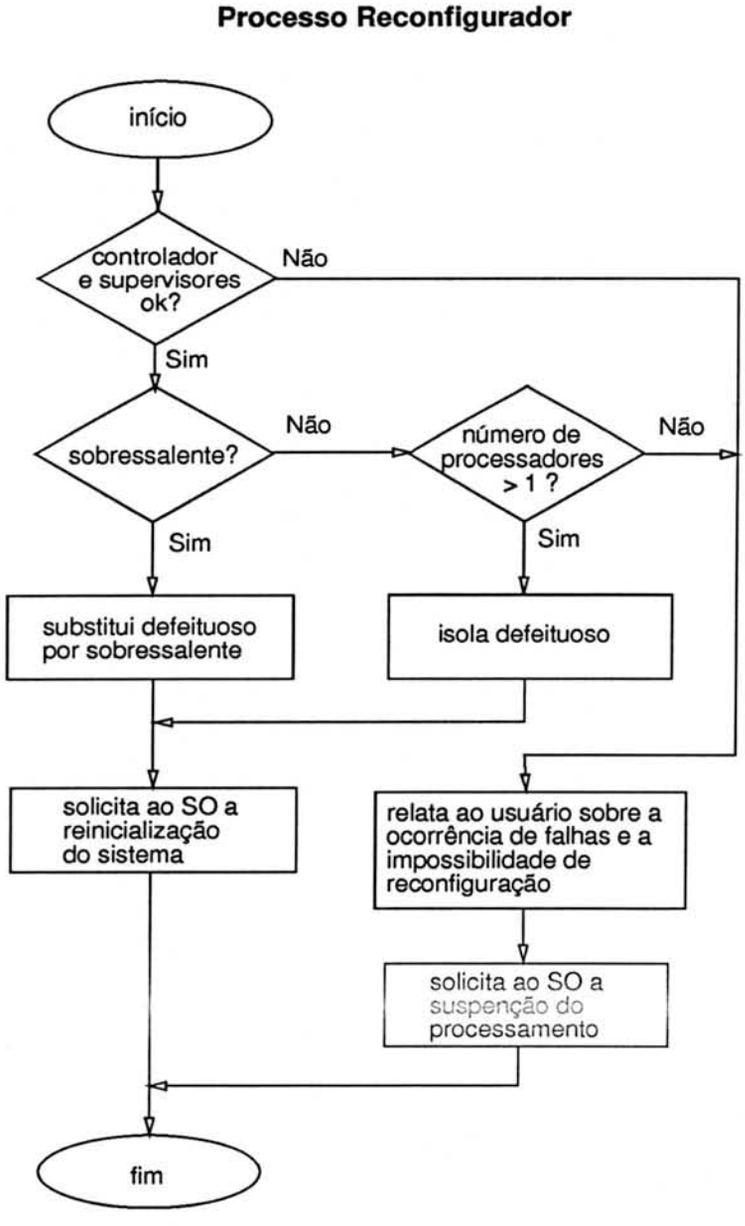

Figura 5.3: Fluxograma do processo reconfigurador.

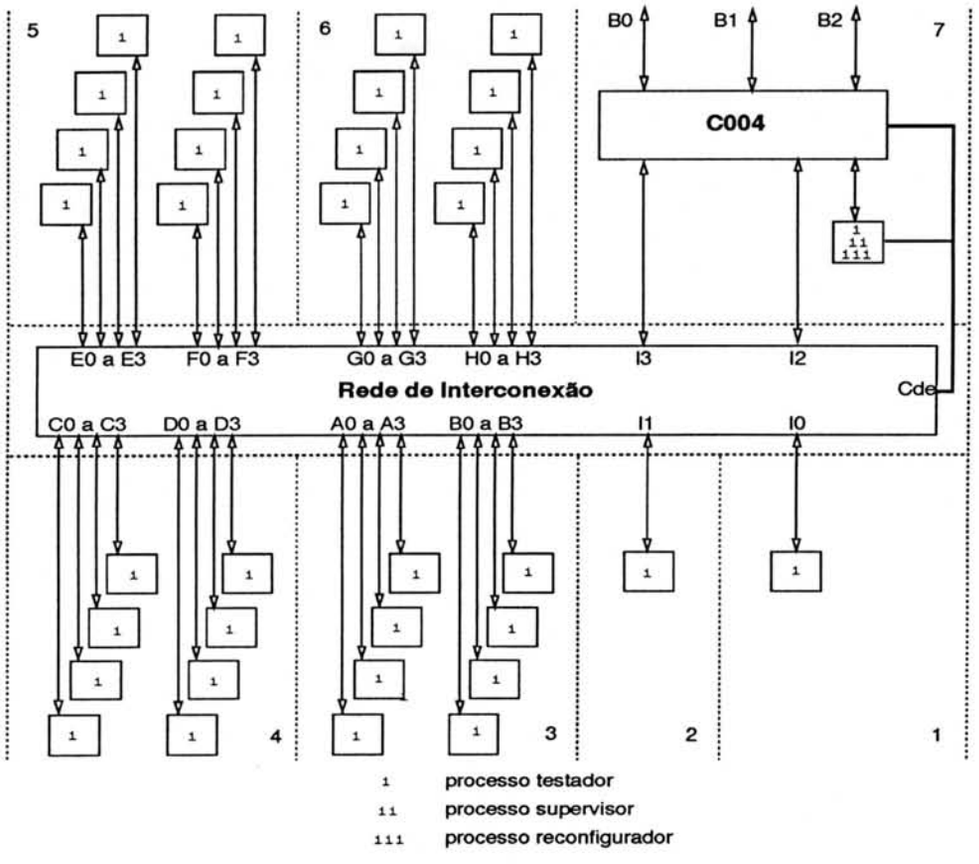

Figura 5.4: Localização dos processos nos processadores do nodo básico.

5.3 Posicionamento do Algoritmo

Para que a estratégia de reconfiguração vista na seção anterior possa ser implementada, é necessário que o algoritmo tenha permissão para mapear processos sobre processadores adequados, de modo que os processos testadores fiquem localizados exatamente um em cada processador da rede, pois do contrário a cobertura por processador fica comprometida. Esta preocupação é relevante uma vez que alguns sistemas operacionais para ambientes *transputer* como, por exemplo, o TRIX [PAZ91], que é uma adaptação do MINIX para *Transputer*, fazem com que o usuário não tenha acesso direto ao *hardware* da máquina, ficando a cargo do sistema operacional alocar processos a processadores. Já o sistema operacional HELIOS permite que o usuário envie processos a processadores específicos (similarmente ao SunOS), pois suporta o ambiente 3L e TDS, os quais exigem este tipo de alocação. Esta dificuldade de alocação física de processadores inserida

por alguns sistemas operacionais, impede que o algoritmo proposto opere num nível superior ao do sistema operacional.

No momento em que ocorre uma falha, o arquivo *snconfig.src*, que descreve o *hardware* disponível no T-NODE através de várias tabelas de recursos, é alterado. Esta alteração faz com que o processador defeituoso torne-se não disponível pela máquina, já que o sistema é reinicializado, e que um dos processadores sobressalentes tome suas atribuições. Deste modo, consegue-se o isolamento do defeito.

O sistema operacional é quem resolve o novo mapeamento dos processos a partir do arquivo de configuração do usuário (*config.usr*) e do novo *snconfig.src*, impedindo, deste modo, que o algoritmo proposto opere num nível abaixo do sistema operacional, pois necessita ocupar parte deste.

O posicionamento e interação do algoritmo proposto em relação ao sistema operacional é mostrado pela figura 5.5.

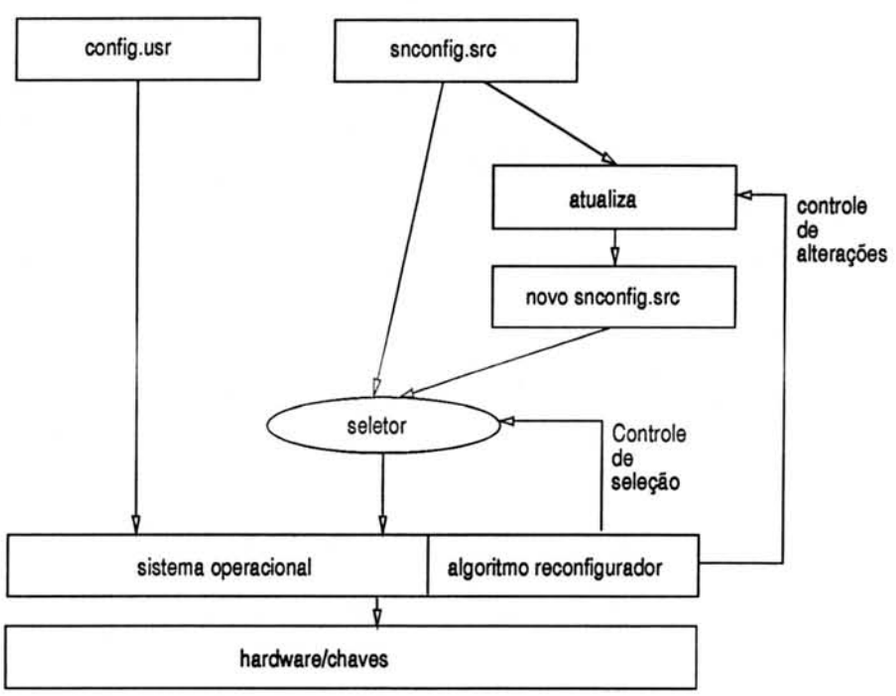

Figura 5.5: Posicionamento e interação do algoritmo com o sistema operacional.

A implementação do algoritmo a nível do sistema operacional (SO) necessita que a parte do SO que realiza o casamento entre o arquivo de configuração do usuário (*config.usr*) e o arquivo de descrição do *hardware* (*snconfig.src*), suporte casos em que o usuário solicite mais processadores do que a configuração física oferece, ou seja, se o arquivo de configuração do usuário necessita de, por exemplo, 10 processadores e existem somente 9 disponíveis, o SO deve alocar os 9 e adotar uma política de redistribuição de carga para o décimo. Esta política poderá ser de:

- transferir toda a carga ao processador menos carregado; ou
- distribuir a carga através da rede.

Do ponto de vista de desempenho, a segunda política é a mais adequada, pois fornece um melhor balanceamento de carga na rede, o que prejudica menos o desempenho do sistema. Entretanto esta escolha fica a cargo do projetista do SO.

Esta situação de falta de processadores pode ocorrer quando um sistema está operando com redundância temporal ou quando o número de falhas extrapola o número de sobressalentes.

6 IMPLEMENTAÇÃO

Neste capítulo são descritos os aspectos inerentes à implementação do algoritmo proposto no capítulo anterior e uma análise de como este algoritmo pode ser integrado ao sistema operacional HELIOS.

6.1 Ambiente de desenvolvimento

Para a implementação do algoritmo de reconfiguração proposto, foi utilizada uma placa B004 da INMOS, a qual contém um processador *transputer* T414 com 2 MBytes de RAM, conectada a um microcomputador PC que serve como computador hospedeiro.

A indisponibilidade de mais *transputers* não comprometeu a implementação, pois a possibilidade de executar processos concorrentemente sobre um mesmo *transputer* permitiu a construção de um protótipo, que executa concorrentemente os processos do algoritmo.

A implementação foi feita utilizando o sistema de desenvolvimento *transputer* (TDS), que contém um compilador OCCAM2, um editor de páginas (*folder editor*), um depurador e vários utilitários. A escolha da linguagem de programação OCCAM2, em preferência ao 3L Parallel C (versão de 1988), foi principalmente pela primeira possuir o construtor alternativo (ALT), que permite a escuta de várias entradas simultâneamente (sem bloqueio por espera) através de canais de comunicação, e por ser a linguagem própria do *transputer*, o que permite que o código gerado pelo compilador OCCAM seja um código de máquina *transputer* bastante otimizado.

6.2 Processos implementados

Esta seção descreve a implementação dos três processos que compõem o algoritmo de reconfiguração proposto: o testador, o supervisor e o reconfigurador, sendo mostrada as partes mais importantes do código. O anexo A-1 contém a listagem completa do código implementado.

O algoritmo implementado solicita, no início de sua execução, qual o número de processadores sobressalentes que o usuário pretende utilizar. O número de processadores físicos da máquina é fixo em tempo de compilação, pois isto não muda dinamicamente numa máquina real.

De posse do número de processadores físicos e do número de processadores sobressalentes desejados pelo usuário, a máquina pode ser configurada. No ambiente real o sistema pode ler um arquivo que define a configuração pretendida pelo usuário. No HELIOS esta fase equivale a leitura do mapa de recursos definido pelo usuário (mais detalhes sobre o HELIOS foram descritos no capítulo 3). Na atual implementação a configuração é definida automaticamente, se o usuário optar por configurar a rede em *mesh*, ou manualmente, para qualquer outra topologia.

Para o estabelecimento destas conexões foi adotada a seguinte convenção de numeração para os canais *transputer*:

1. todos os canais (bidirecionais) possuem números exclusivos;
2. o número do canal depende do número do processador e da posição relativa sobre o mesmo (norte, sul, leste ou oeste).

Para calcular o número de um canal y pertencente a um processador n, deve-se multiplicar o número do processador por 4, pois este é o número de canais por processador, e adicionar 1, 2, 3 ou 4, dependendo da posição relativa

norte, leste, sul ou oeste. A figura 6.1 exemplifica este cálculo de numeração dos canais.

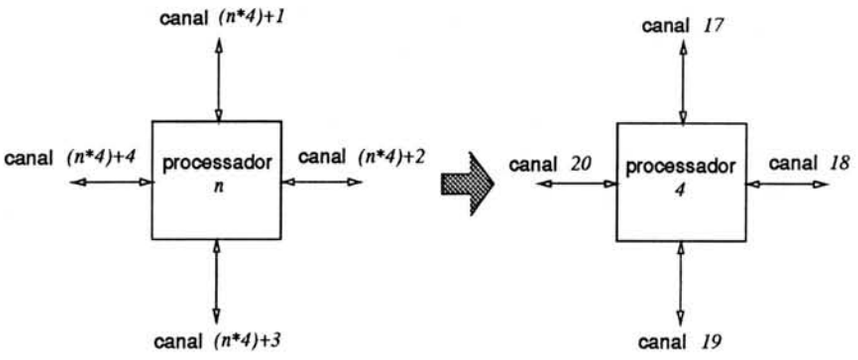

Figura 6.1: Numeração dos canais *transputer*.

Esta numeração é importante para o estabelecimento manual das conexões, que é feito informando estes números. Conexões ao canal zero (canal <—> 0) indicam a desconexão do canal.

As conexões entre os processadores da rede foram implementadas através de uma tabela de conexões que se utiliza da numeração dos canais conforme descrito acima. Esta tabela descreve as conexões físicas da rede de interconexão, e as alterações feitas sobre ela equivalem às requisições enviadas ao *software* que gerencia a programação das chaves, o SwDrv.

Após a configuração da rede, os processos testadores, supervisor e reconfigurador são disparados em paralelo. A seguir são descritos cada um destes processos.

6.2.1 Testador

Este processo, responsável pela detecção de falhas no *hardware* do *transputer*, não inclui a definição do conjunto de testes a ser executado, por exceder ao escopo inicial deste trabalho. Apesar disto, a elaboração deste conjunto merece maiores comentários, pois ele tem reflexos sobre parâmetros de desempenho e

degradação do sistema, uma vez que a relação custo-desempenho adequada é obtida com a sua execução em curto intervalo de tempo, exercitando o maior número de registradores e blocos funcionais, ou seja, verificando seu funcionamento.

De acordo com [WAG88], os testes podem ser gerados nos seguintes níveis: lógico, no nível de portas lógicas, funcional, comportamental e de microprocessadores; o enquadramento em um destes grupos depende do nível de abstração em relação ao *hardware*. No caso dos *transputers*, a aplicação dos testes necessita ser efetivada através do conjunto de instruções do processador, que é a única forma disponível para acesso aos seus pontos internos. Este tipo de testes pode ser enquadrado na classe dos testes funcionais, pois prevê o uso dos registradores, ou na classe dos testes de microprocessadores, pois são aplicados sob a forma de um conjunto de instruções. A opção entre um destes níveis vai depender de uma definição complementar: do modelo de falhas empregado.

O modelo de falhas pode ser, por sua vez, funcional ou estrutural. No modelo de falhas funcional, os testes são, em teoria, determinados a partir da função especificada para o circuito; a esta função é associado o modelo de falhas. No modelo de falhas estrutural, os vetores de teste são determinados a partir da estrutura do circuito; à estrutura do circuito são associados os modelos de falhas. Em ambos os casos, a maior ou menor precisão dos modelos depende do nível de descrição e de detalhamento da especificação do circuito.

No caso do *transputer*, dispõe-se de uma descrição de sua funcionalidade apenas. Nos manuais, há esboços de sua organização, a nível funcional; entretanto estes esboços correspondem ao funcionamento lógico, não se dispondo de qualquer indicativo de que exista correspondência com a implementação real. Assim, embasando-se nesta organização procedimentos estruturais de teste, obtém-se resultados pouco confiáveis. Outro aspecto que sugere a não utilização do teste estrutural, é o nível de detalhamento dos modelos de falhas que facilmente conduzirá a conjuntos de testes longos, o que é indesejável na presente aplicação.

Assim, os testes a serem aplicados sobre o *transputer*, que devem fornecer informação do tipo funciona/ não funciona (*go/ not go*) deverão ser realizados com base em um modelo de falhas funcional, de forma similar aos sugeridos por Thatte e Abraham em [THA80] ou por Chantal e Saucier em [CHA80], independente portanto de detalhes de implementação. Esta mesma filosofia está sendo empregada por Ecker e Marz, em artigo recente publicado no CHDL'93 [ECK93]; neste, os autores propõem uma estratégia de testes baseados em modelos funcionais para validar o projeto de circuitos usando VHDL. O processador desenvolvido foi testado utilizando programas armazenados numa ROM.

A determinação do tempo gasto pelo processo poderá ser obtida quando o conjunto mínimo de instruções for definido, pois cada instrução consome um número fixo de ciclos de máquina. Este parâmetro possibilitará obter uma avaliação da degradação inserida pela execução periódica do processo testador.

Na implementação, os testes são simulados por geradores de erros aleatórios com uma distribuição uniforme, um em cada processo testador, os quais permitem que o algoritmo de reconfiguração possa ser ativado.

Como a seqüência de números gerados por cada processo testador não pode ser a mesma, pois neste caso todos os processos falhariam numa seqüência fixa, foi utilizado como semente para cada gerador a leitura da hora atual do sistema, o que garante uma aleatoriedade nos números entre os geradores, dado que os processos não estão sincronizados no tempo. A escolha deste método para simular a falha tornou desnecessária a tentativa de re-geração de erro no momento de refazer o teste, pois o erro gerado é considerado um erro permanente. Deste modo, na implementação atual o processo somente foi preparado para receber o conjunto mínimo de instruções que realizarão os testes reais (linhas ... *testa cpu, fpu e memoria* e ... *repete o teste* da figura 6.2).

A mensagem que implementa um estado "ok" (*msg_ok*) é uma mensagem TRUE, enquanto a de "não ok" (*msg_nao_ok*) é FALSE. Deste modo, quando

um erro for gerado, o processo envia a mensagem FALSE para o supervisor e termina. Quando não for detectado defeito uma mensagem TRUE é enviada pelo *canal_supervisor_testador*. Em seguida, a hora atual do sistema é lida, e o processo bloqueado por um tempo igual ao do período de execução dos testadores. Nesta implementação este período possui um valor arbitrário, pois seu valor ótimo depende da relação custo/benefício desejada por cada usuário, onde o custo é a perda de desempenho decorrente de um período menor e do tamanho do conjunto de testes (cobertura de falhas), e o benefício é o aumento de confiabilidade nas saídas do sistema. O código que implementa o processo testador está descrito brevemente na figura 6.2.

```
-- Processo testador
... declarações e inicializações
  SEQ
    WHILE modulo_ok
      SEQ
        ... testa cpu, fpu e memória
        IF
          nao_ok
            SEQ
              ... repete o teste
              IF
                nao_ok
                  SEQ
                    canal_supervisor_testador ! msg_nao_ok
                    modulo_ok := FALSE  -- termina processo
                TRUE
                  SEQ
                    canal_supervisor_testador ! msg_ok
                    relogio ? agora     -- lê hora do sistema
                    -- bloqueia processo por período de tempo especif.
                    relogio ? AFTER agora PLUS periodo
          TRUE
            SEQ
              canal_supervisor_testador ! msg_ok
              relogio ? agora           -- lê hora do sistema
              -- bloqueia processo por período de tempo especificado
              relogio ? AFTER agora PLUS periodo
```

Figura 6.2: Processo testador.

6.2.2 Supervisor

A existência de vários processos testadores executando ao mesmo tempo em diversos processadores da máquina tornou necessária a implementação

deste processo, o qual monitora as mensagens de estado provenientes dos diversos processos testadores, associando a cada uma um tempo máximo de espera (*timeout*), conforme proposto na seção 5.2.2.

Na implementação deste processo constatou-se que o construtor ALT do OCCAM, não implementa fielmente o construtor *alternative* proposto por Hoare em sua teoria de processos seqüenciais comunicantes (CSP), onde a seqüência de verificações de condição associada a este construtor é aleatória em particular, a ordem dada. O que o construtor ALT do OCCAM implementa é um pseudo *alternative*, pois possui uma ordem seqüencial de verificação das condições, sendo a de índice mais baixo a de maior prioridade (a primeira). Esta característica, que não se encontra descrita no manual do usuário da linguagem [INM88b], não possibilita o uso de dois ALTs aninhados (conforme proposto na seção 5.2.2) na escuta dos diversos canais com os testadores. Deste modo, a escuta dos canais foi implementada de maneira seqüencial para garantir que todos os canais sejam verificados pois, no caso contrário, os últimos teriam grandes chances de não serem escutados em tempo hábil, dado que existe um *timeout* associado a cada canal de entrada. A figura 6.3 mostra o código deste processo.

```
-- Processo supervisor
... declarações e inicializações
WHILE nenhum_erro
  SEQ
    proc := 0
    -- verifica entrada e timeout por processador
    WHILE nenhum_erro AND proc <> ultimo_processador
      SEQ
        ALT
          canal_supervisor_testador[proc] ? msg_testador[proc]
            IF
              msg_testador[proc] = ok
                ... inicializa contador de timeout
              TRUE
                SEQ
                  nenhum_erro := FALSE
                  canal_supervisor_reconfigurador ! msg_modulo_nao_ok
          canal_timeout[proc] ? AFTER hora_atual[proc] + periodo_timeout
            SEQ
              ... dá mais uma chance esperando mais 1/3 do periodo_timeout
              nenhum_erro := FALSE
              canal_supervisor_reconfigurador ! msg_modulo_nao_ok
        proc := proc + 1  -- passa para o próximo processador
```

Figura 6.3: Processo supervisor.

É importante salientar que para um funcionamento ideal o processo supervisor deve ser implementado com um construtor alternativo que corresponda à proposta de Hoare.

O envio de mensagem para o processo reconfigurador somente acontece quando é recebida uma mensagem de estado FALSE (defeituoso). O número do processador defeituoso é automaticamente determinado porque o supervisor possui um índice associado ao canal de recebimento da mensagem. Este índice é passado ao reconfigurador para que este já conheça, no momento do seu disparo, qual o processador defeituoso.

6.2.3 Reconfigurador

Este processo, responsável pelo isolamento e tratamento da falha, implementa as alterações do arquivo *snconfig.src*, que descreve todos os detalhes da configuração física da máquina, e o estabelecimento das novas conexões após a ocorrência de uma falha.

A ativação do processo reconfigurador é feita em paralelo com os processos testadores e o supervisor, mas logo no início de sua execução ele é bloqueado para que não execute a reconfiguração antes que uma falha seja detectada por um dos testadores. Este bloqueio do processo é implementado pela execução de uma instrução de espera por recebimento de mensagem proveniente do supervisor ($canal\_supervisor\_reconfigurador?msg\_supervisor$, conforme mostra a figura 6.4), pois a comunicação entre processos na linguagem OCCAM obedece a um protocolo onde ambos (tanto emissor como receptor) devem estar prontos para troca de mensagens sobre o canal especificado, sendo que o primeiro a tornar-se pronto fica bloqueado. O desbloqueio do reconfigurador somente acontece quando o supervisor recebe uma mensagem de defeito e informa o reconfigurador (conforme visto na subseção anterior).

Ao receber a mensagem do supervisor contendo o número do processador defeituoso, o reconfigurador verifica se é possível reconfigurar o sistema. Se a falha ocorreu num controlador ou num servidor (de memória ou disco), não é possível fazer a reconfiguração do sistema, pois estes processadores possuem funções específicas fixas no *hardware* da T-NODE, conforme descrito na seção 2.2.

```
-- Processo reconfigurador
... declarações e inicializações
SEQ
  canal_supervisor_reconfigurador ? msg_supervisor
  IF
    ... verifica se processador defeituoso é o controlador
      ... informa que falha é irrecuperável
    ... verifica se processador defeituoso é um servidor
      ... informa que falha é irrecuperável
    TRUE                          -- processador é de trabalho
      SEQ
        IF
          -- reconfiguração com sobressalente
          numsobr > 0             -- existe sobressalente
            SEQ
              ... faz a permuta do processador defeituoso pelo sobressalente
              ... isola processador defeituoso
          -- reconfiguração sem sobressalente
          TRUE                    -- não tem sobressalente
            IF
              numproc > 1         -- verifica se tem mais de um processador bom
                SEQ
                  ... faz o bypass dos canais conectados ao processador def.
                  ... isola processador defeituoso
              TRUE
                ... informa que falha é irrecuperável
        ... mostra nova configuração
```

Figura 6.4: Processo reconfigurador.

Se a falha ocorreu num processador de trabalho, uma verificação no número de processadores sobressalentes é feita. O resultado da verificação define o tipo de reconfiguração a ser seguido pelo algoritmo: reconfiguração com ou sem sobressalentes. Cada um destes caminhos é analisado a seguir.

6.2.3.1 Reconfiguração com Sobressalentes

Quando o sistema opera com sobressalentes, a ocorrência de falhas em qualquer um dos processadores de trabalho não implica na perda de topologia, pois o tratamento dado pelo reconfigurador faz com que o canal conectado ao

canal norte do processador defeituoso, seja desconectado deste e conectado ao canal norte do processador sobressalente. O mesmo procedimento é feito para os demais canais, estabelecendo deste modo a permuta do processador defeituoso pelo sobressalente, como mostra a figura 6.5. O uso desta metodologia de reconfiguração somente foi possível porque a rede de interconexão da T-NODE é do tipo *crossbar*. A divisão da rede na T-NODE em N <—> S e L <—> O não interfere no uso desta metodologia, pois em nenhum caso ela solicitará ligações S, N <—> L, O.

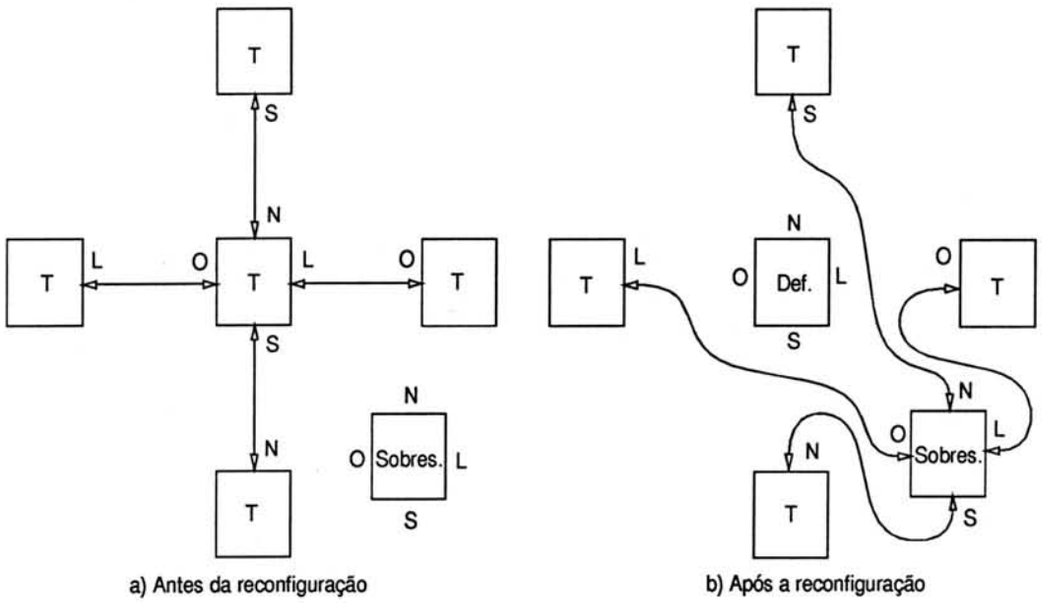

Figura 6.5: Chaveamento de canais do sobressalente.

6.2.3.2 Reconfiguração sem Sobressalentes

Quando o sistema não usa sobressalentes a perda de topologia após a reconfiguração é inevitável, uma vez que o número de processadores disponíveis é menor do que o número solicitado pelo usuário. Na implementação atual, a técnica de *bypass* é utilizada para conectar processadores vizinhos do defeituoso, semelhantemente a maneira utilizada na estratégia TFOS que conecta o processador pai do defeituoso ao filho do mesmo (conforme visto no capítulo 4).

A reconfiguração sem sobressalentes exige que o servidor de rede do HELIOS reconheça todos os processadores lógicos definidos no mapa de recursos do usuário, independentemente de não existir um processador físico para cada. Quando o servidor SwMan receber uma requisição de um novo processador e retornar um erro, um outro processador já alocado deve ser encarado pelo sistema como sendo um processador físico mas também como dois lógicos.

Como a T-NODE possui limitação nas conexões dos canais, por exemplo no nodo TANDEM e SUPERNODE somente conecxões N <—> S e L <—> O, a técnica de *bypass* utilizada nesta implementação isola o processador defeituoso dos seus vizinhos da seguinte maneira: o canal conectado ao canal oeste do processador defeituoso é desconectado deste e conectado ao canal que está conectado ao canal leste deste processador, e vice-versa. A mesma regra é utilizada para estabelecer as conexões N <—> S. A figura 6.6 mostra um exemplo do como um sistema é reconfigurado utilizando esta técnica.

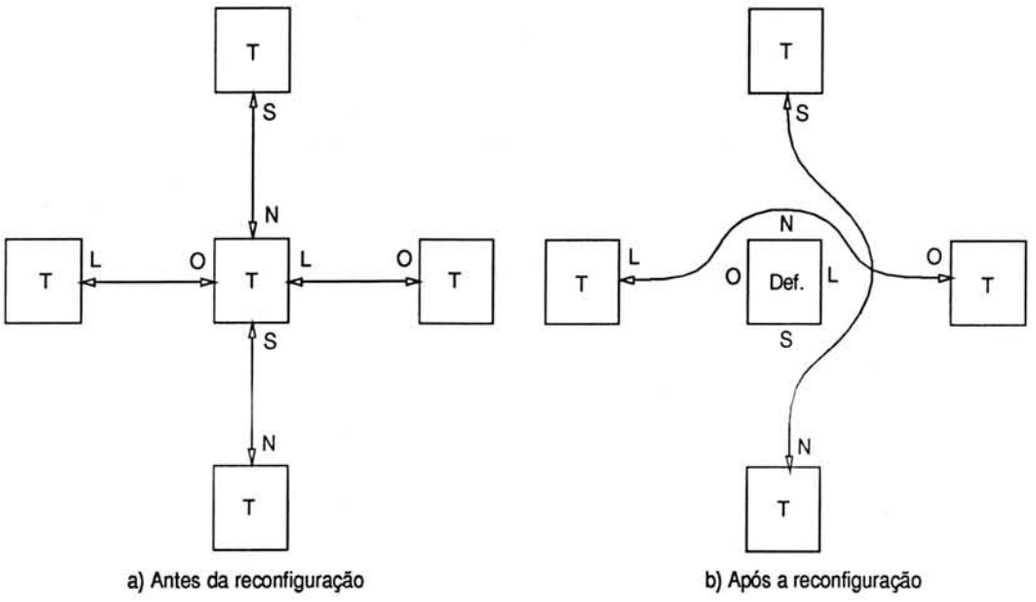

Figura 6.6: *Bypass* no processador defeituoso.

A técnica de *bypass* descrita acima é eficiente para estruturas onde há uma simetria nas conexões envolvendo os canais N,S e L,O dos *transputers* como, por exemplo *array*, hipercubo, *mesh*, estrela e anel. Para casos onde não existe

simetria nas conexões de um mesmo *transputer* como, por exemplo em árvores, poderá haver processadores (ou ramos) que serão isolados do restante da rede. Para estes casos especiais, devem ser incorporadas ao presente algoritmo técnicas de reconfiguração para topologias específicas já propostas na bibliografia. Por exemplo, se o usuário trabalhar com uma topologia em árvore, a estratégia de conectar processadores sobressalentes nas folhas da árvore, proposta por Lowrie, em [LOW87], pode ser empregada, dado que o principal problema da definição das novas conexões em topologias não simétricas é a limitação das conexões entre canais em N <—> S e L <—> O, e do número total de canais (quatro por *transputer*).

6.3 Comentários sobre Desempenho

A degradação no desempenho do sistema T-NODE pela inserção do algoritmo de tolerância a falhas apresentado neste trabalho depende principalmente:

- da duração do conjunto de testes a ser executado pelo processo testador;
- do período de execução do conjunto de testes;
- do número de mensagens entre os processos componentes do algoritmo; e
- do tempo necessário para o reestabelecimento do sistema após a ocorrência de uma falha.

A duração do conjunto de testes (testes funcionais) executados pelo testador é determinado pela cobertura de falhas pretendida pelo usuário. Dado que os testes devem exercitar as unidades funcionais do processador, a determinação do conjunto ótimo de instruções a serem executadas deve ser feita a partir da análise detalhada das micro-operações de cada instrução que compõem o proces-

sador. O tempo de execução destas instruções fica automaticamente determinado pelo tempo de execução de cada instrução em ciclos de máquina (fixo).

O período de execução do conjunto de testes, que determina quanto tempo o processo testador deve ficar inativo, também é determinado pela cobertura de falhas pretendida pelo usuário. Na determinação deste período devem ser levados em conta o MTBF do processador, as condições ambientais de operação do mesmo e os níveis de confiabilidade desejáveis, uma vez que são consideradas somente falhas no *hardware*.

O número do mensagens acrescidas na rede pela inserção deste algoritmo é insignificante, pois a geração de uma mensagem que deve ser roteada através da rede só acontece uma vez a cada ciclo de execução do processo testador, fazendo o número de mensagens por ciclo de execução do algoritmo ser igual ao número de processadores da rede. A figura 6.7 mostra a interação entre os processos que compõem o algoritmo através dos seus respectivos canais de comunicação.

O tempo de reestabelecimento do sistema é o ponto mais crítico relativo à degradação. Isto acontece porque uma falha pode corromper informação por diversos processadores da rede quando há cooperação entre eles. Neste trabalho a recuperação é feita do estado inicial de processamento, perdendo-se todo processamento já feito. Este método de reconfiguração baseia-se na baixa probabilidade de falha encontrada nos processadores da linha *transputer*. Métodos de recuperação de sistemas a partir de pontos intermediários minimizam o tempo de recuperação, mas em contrapartida exigem um tempo de execução bem mais elevado (em comparação com o adotado neste trabalho) pois necessitam fazer salvamentos periódicos de pontos consistentes de recuperação.

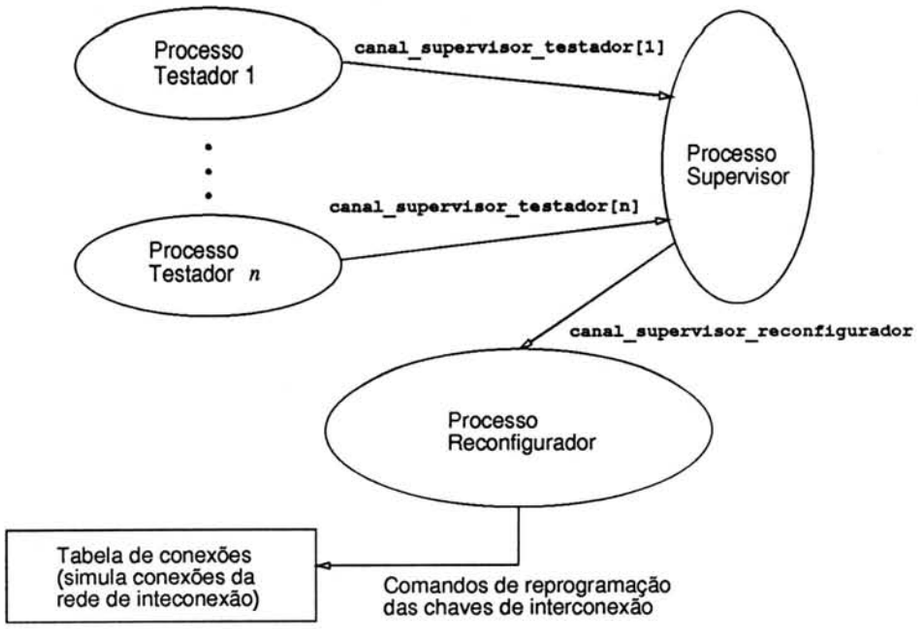

Figura 6.7: Interação entre os processos.

6.4 Integração com o *Software* T-NODE

A análise feita nesta seção baseia-se somente em dados teóricos sobre o sistema operacional HELIOS e as ferramentas Telmat apresentados no capítulo 3, pois o ambiente de desenvolvimento utilizado não contempla estes *softwares*.

A integração necessária do algoritmo com o HELIOS durante a fase de execução livre de falhas fica restrita ao roteamento de mensagens do processo testador ao processo supervisor, o qual é gerenciado pelo servidor de rede do sistema. Já na fase de execução com falhas a necessidade de isolamento e tratamento da falha requer o uso de recursos fornecidos pelo sistema, tal como alteração do arquivo de descrição do *hardware*.

Conforme proposto no capítulo anterior, o algoritmo deve estar implementado a nível do sistema operacional para que tanto funções de baixo como de alto nível possam ser executadas.

O sistema operacional HELIOS, através de seu servidor de rede, interage com os servidores produzidos pela Telmat (conforme visto no capítulo 3) para inicializar ou configurar uma rede. Os servidores Telmat utilizam as informações contidas no arquivo *snconfig.src*, o qual descreve o *hardware* da T-NODE, para alterar o chaveamento entre processadores.

Como o HELIOS desconhece os recursos físicos da máquina e não tem como prevenir que um processador num estado defeituoso seja utilizado pela máquina, é necessário que existam alterações no *software* básico produzido pela Telmat.

Para evitar que um processador defeituoso seja utilizado pela máquina, o arquivo *snconfig.src*, que descreve as características de cada processador, deve indicar também qual o estado de cada um dos processadores (com falha / sem falha). Deste modo, quando existirem processadores defeituosos no sistema, estes processadores estarão marcados como indisponíveis neste arquivo, evitando que sejam utilizados pelo sistema na, ou após a, inicialização do mesmo.

Na inicialização do sistema, uma variável inicializada no arquivo de configuração *nsrc*, pode determinar o número de processadores sobressalentes que o sistema deve conter. Esta variável é utilizada para indicar ao servidor de rede, que terá que ser capaz de alterar o arquivo de descrição do *hardware*, quantos processadores devem ser marcados como sobressalentes (opção feita pelo usuário).

A implementação destas alterações no arquivo *snconfig.src* pode ser feita pela inserção, na tabela de descrição dos processadores, de mais uma coluna (coluna 13) que indique o modo de operação do processador. Deste modo, as tabelas *transputers* ficam com a forma da tabela 6.1.

Os modos de operação permitidos pelo novo arquivo *snconfig.src* são:

Tabela 6.1: Campos das Novas tabelas *Transputers*.

Direções				Carecterísticas								Twi	Operação
N	L	O	S	PIN	mem	tipo						order	modo
							. . .						
0	1	2	3	4	5	6	7	8	9	10	11	12	13
Colunas													

- **normal**, no qual o processador pode ser utilizado segundo as regras definidas pelo sistema (no caso do HELIOS, as opções: Helios, Native, System ou IO);
- **defeituoso**, no qual o processador está com defeito e não pode ser alocado ao sistema;
- **sobressalente**, no qual o processador foi reservado para uso exclusivo em caso de falhas de processador, e também não pode ser alocado ao sistema.

Estas alterações permitem que tanto o *software* de rede do HELIOS como os servidores Telmat saibam o número de processadores disponíveis e o número de processadores sobressalentes da rede. Em caso de falha, a mudança no modo de operação do processador defeituoso e a reprogramação da rede faz com que ele seja isolado do sistema e não mais utilizado por este.

Quanto ao mapa de recursos, nenhuma alteração é necessária porque este somente trabalha num nível lógico, não influenciando a rede física propriamente dita.

7 CONCLUSÃO

O aumento considerável no uso de computadores em diferentes áreas de aplicação estimulou o desenvolvimento de arquiteturas paralelas com alto grau de desempenho e confiabilidade. Alvo deste trabalho, a máquina T-NODE possui uma arquitetura especialmente projetada para obtenção de alto desempenho, mas carente no que se refere a aspectos de tolerância a falhas em *hardware*.

Neste trabalho foi estudado a arquitetura da máquina T-NODE, a qual se caracteriza por ser do tipo multiprocessadora e totalmente reconfigurável, e também as características de diversos algoritmos de reconfiguração para arquiteturas multiprocessadoras, a fim de propor, com base nestes estudos, uma estratégia de reconfiguração adequada para a máquina T-NODE. As hipóteses de falhas consideradas foram as listadas na seção 5.1.

O desenvolvimento das técnicas de reconfiguração para arquiteturas multiprocessadoras normalmente visam arquiteturas com topologias específicas como, por exemplo, *arrays* e árvores. Embora a máquina T-NODE não apresente estas restrições topológicas, neste trabalho usou-se conceitos importantes pertencentes a estas técnicas, os quais são analisados no capítulo 4.

As técnicas de reconfiguração tradicionalmente destinam-se ao tratamento de falhas após a sua detecção e localização. A algoritmo apresentado, ao contrário das técnicas tradicionais, propõe também uma estratégia para controle da detecção de falhas nos módulos e localização dos módulos falhos, usando conceitos de processamento paralelo, pois distribui processos testadores por todos os módulos da máquina e implementa-os utilizando recursos de uma linguagem de programação paralela (OCCAM2).

Quanto à cobertura de falhas, o algoritmo pode tratar a reconfiguração em qualquer tipo de topologia e permite o suporte à s falhas, onde s é o número

de processadores sobressalentes escolhido pelo usuário, sem inserir degradação de desempenho, pois faz uso de 100% dos processadores sobressalentes. Quando não existir sobressalente, o algoritmo suporta $n-1$ falhas, onde n é o número total de processadores disponíveis na máquina. O suporte de $n-1$ falhas está condicionado à capacidade do sistema operacional, ao qual o algoritmo está integrado, em suportar a alocação lógica de processadores, pois é necessário distribuir as tarefas do defeituoso para um ou mais processadores não defeituosos.

Topologias com simetria de conexões (em relação ao nodo) são tratadas satisfatoriamente. Entretanto, para topologias sem simetria de conexões o tratamento não é eficiente devido à limitação de conexões (somente N <—> S e L <—> O) e ao número reduzido de canais por *transputer*. Para estes casos devem ser consideradas as características específicas da topologia em questão.

A tarefa de definir o número de processadores sobressalentes é executada pelo usuário, pois esta decisão tem reflexos na usabilidade da máquina, uma vez que o número de processadores disponíveis para a aplicação é uma função da quantidade de sobressalentes.

Um resultado bastante satisfatório que pôde ser observado é que a máquina T-NODE consegue suportar múltiplas falhas a um custo relativamente baixo, tanto a nível de implementação quanto de degradação de desempenho. Ao nível de implementação de tolerância a falhas, a estratégia apresentada neste trabalho é eficiente e fácil de ser incorporada ao sistema. Quanto à degradação de desempenho, fazendo uma comparação dos resultados alcançados por Bart Veer [VER93], onde um período de 30 segundos para a busca de alguns defeitos sobre a rede de processadores causa um impacto menor do que 0.1% no desempenho do sistema, com a característica do *transputer*, que possui uma arquitetura RISC com somente 167 instruções que podem ser executadas (todas) em aproximadamente 33 micro segundos, garante-se que um conjunto de testes funcionais sobre o *transputer* insere menos do que 0.1% de degradação de desempenho.

Como sugestão para trabalhos futuros fica a otimização do algoritmo proposto, definição e implementação do conjunto de testes, e uso de técnicas para recuperação a partir de um estado intermediário.

Para a otimização do algoritmo proposto é interessante que a estratégia de troca de mensagens entre os processos do algoritmo seja portada para o barramento de controle da T-NODE, permitindo deste modo uma menor degradação no desempenho global do sistema. Esta nova estratégia de troca de mensagens diminui o fluxo de mensagens, através da rede de interconexão, mesmo que atualmente seja pequeno, evitando que os roteadores de mensagens entre processadores necessitem rotear também mensagens de controle do algoritmo.

A definição e implementação do conjunto de testes deverá englobar o modelo de falhas funcional, objetivando exercitar a partir de um conjunto mínimo de instruções a maior parte das unidades funcionais do *transputer*.

A reinicialização do sistema sob falhas, ou recuperação, de um estado consistente intermediário necessitará de uma avaliação e adequação de técnicas de salvamento de pontos de recuperação (*checkpoints*) e conseqüente restauração (*rollback*). A maior implicação neste sentido é a complexidade para achar um ponto consistente no momento da recuperação, dado que em sistemas multiprocessados existem vários processos se comunicando simultâneamente. Entretanto, o uso destas técnicas evitaria que longos tempos de processamento livres de falhas, executados antes da ocorrência da falha, sejam perdidos.

ANEXO A-1 LISTAGEM DO CÓDIGO IMPLEMENTADO

Este anexo apresenta a listagem do código em OCCAM2 que implementa a estratégia de reconfiguração descrita no capítulo 6.

```
--*************************************************************************
--
--        ALGORITMO DE RECONFIGURACAO DA T-NODE EM CASO DE FALHAS
--
--*************************************************************************

--
-- chamada de bibliotecas
--
#USE uservals                -- biblioteca com constantes do TDS
#USE userio                  -- biblioteca com procedimentos de I/O
#USE t4utils                 -- utilizada no gerador de erros
#USE realpds                 -- utilizada para a funcao SQRT()

--
-- declaracao de constantes
--
VAL numproc IS 16 :          -- define numero de processadores ativos
VAL numespec IS 03 :         -- define numero de processadores especiais

--
-- declaracoes de variaveis
--
BOOL NoErro :                -- controle de terminacao de processos
                             -- quando ocorrer erro
INT numsobr :                -- numero de processadores sobressalentes
BOOL optionc :               -- variavel auxiliar
INT option :                 -- opcao de estabelecimento de nova config.
[((numproc+numespec)*4)+1]INT gconfigtab : -- tabela conexoes
                                           -- (equivalente ao snconfig)

--
-- declaracoes de canais interprocessos
--
CHAN OF INT suprec:          -- canal entre supervisor e reconfigurador
[(numproc+numespec)+1]CHAN OF BOOL suptes : -- canais testador /
                                            -- supervisor

--
-- procedimentos
--

--*************************************************************************
--
```

```
-- Nome         : mostra
--
-- Funcao       : mostra configuracao atual da rede de processadores
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : julho de 1993
--
-- Observacoes :
--
--*****************************************************************************
PROC mostra ([]INT configtab, CHAN OF ANY screen)
  SEQ
    newline (screen)
    SEQ i=1 FOR (numproc*4)
      SEQ
        write.full.string (screen, "        conexao ")
        write.int (screen, i, 0)
        write.full.string (screen, "-")
        write.int (screen, configtab[i], 0)
    newline (screen)
:

--*****************************************************************************
--
-- Nome         : configrede
--
-- Funcao       : configurar a rede de processadores em mesh
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : julho de 1993
--
-- Observacoes : Este procedimento e' chamado pelo processo principal.
--               Na implementacao sobre a T-node, este  procedimento
--               nao  sera mais necessario, dado a existencia de um
--               arquivo  que  descreve as conexoes da maquina (mapa de
--               recursos).
--
--*****************************************************************************
PROC configrede (CHAN OF ANY screen, CHAN OF INT keyboard, []INT
                 configtab)
  --
  -- Declaracoes locais
  --
  INT n :                    -- numero do processador
  INT raiz :                 -- numero de linhas/colunas do mesh
  REAL32 tmp :               -- variavel auxiliar
  --
  -- corpo do procedimento
  --
  SEQ
    tmp := SQRT(REAL32 TRUNC numproc) -- calcula a dimencao do mesh
    raiz := 1(INT) * (INT TRUNC tmp)
    SEQ
```

```
      SEQ n=0 FOR numproc-1          -- conexoes L <---> O
        SEQ
          configtab[(n*4)+2] := ((n+1)*4)+4
          configtab[((n+1)*4)+4] := (n*4)+2
      SEQ n=0 FOR (numproc-raiz)     -- conexoes N <---> S
        SEQ
          configtab[(n*4)+3] := ((n+raiz)*4)+1
          configtab[((n+raiz)*4)+1] := (n*4)+3
      SEQ n=(numproc-raiz) FOR raiz-1
        SEQ
          configtab[(n*4)+3] := ((n-((numproc-raiz)-1))*4)+1
          configtab[((n-((numproc-raiz)-1))*4)+1] := (n*4)+3
      n := numproc-1                 -- determina n como ultimo processador
      configtab[(n*4)+2] := 4
      configtab[(n*4)+3] := 0
      configtab[4] := (n*4)+2
      configtab[1] := 0
      mostra (configtab, screen)     -- mostra conexoes da rede no video
      write.text.line (screen, "Rede conectada em mesh. ")
      newline (screen)
:

--=============================================================================
--
-- Nome         : changeconfig
--
-- Funcao       : muda a configuracao inicial da rede por alguma outra
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : julho de 1993
--
-- Observacoes :
--
--=============================================================================
PROC changeconfig ([]INT configtab, CHAN OF ANY screen)
  SEQ
    newline (screen)
    write.text.line (screen, "Entre com a ligacao dos canais: ")
    newline (screen)
    SEQ i = 1 FOR ((numproc - numsobr)*4)
      INT char :
      SEQ
        write.int (screen, i, 0)
        write.full.string (screen, " - ")
        char := INT' '
        read.echo.int (keyboard, screen, configtab[i], char)
    newline (screen)
    write.text.line (screen, "Configuracao terminada.")
    newline (screen)
:

--
-- processos
--
```

```
--=========================================================================
--
-- Nome         : Testador
--
-- Funcao       : Testar CPU, FPU e Memoria interna do transputer
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : marco de 1993
--
-- Observacoes : Informa a ocorrencia ou nao de falhas ao supervisor.
--               Roda em periodos de tempo pre-determinados.
--               Os testes nao foram implementados.
--
--=========================================================================
PROC testador (CHAN OF BOOL ssuptes, INT i, VAL INT tnumproc,
               INT tnumsobr)
  --
  -- funcao geradora de numeros randomicos entre 0 e 1
  --
  REAL32, INT FUNCTION RANDOMIC (VAL INT SeedRan)
    REAL32 RanOut :
    INT NewSeed :
    VALOF
      INT RanOut RETYPES RanOut :
      VAL A IS 1664525(INT) :
      SEQ
        NewSeed := (SeedRan TIMES A) PLUS 1(INT)
        RanOut := ROUNDSN (RealXcess - 1, INT NewSeed, 0)
      RESULT RanOut, NewSeed
  :
  --
  -- Declaracoes locais
  --
  BOOL ok :                     -- estado do processador
  TIMER relogio :               -- canal de tempo
  INT agora :                   -- hora de inicio do wait
  VAL periodo IS 10000 :        -- define periodo de execucao do processo
  REAL32 erro :                 -- varianel auxiliar do gerador randomico
  --
  -- corpo do processo
  --
  IF
    (i >= tnumproc) OR (i < (tnumproc - tnumsobr))
      SEQ
        ok := TRUE              -- marca processador como bom
        relogio ? agora         -- le hora do sistema
        erro := 0.0 (REAL32)
        WHILE ok                -- executa processo enquanto proc. bom
          SEQ
            --
            -- rotina de teste da CPU, FPU e memoria
            --
            erro, agora := RANDOMIC (agora)
```

```
            IF
              erro < 0.05 (REAL32)
                SEQ
                  ok := FALSE
              TRUE
                SKIP
            IF
              ok                        -- verifica resultado do teste
                SEQ
                  ssuptes ! ok          -- informa estado para supervisor
                  relogio ? agora       -- le hora do sistema
                   bloqueia processo p/ periodo especificado
                  relogio ? AFTER agora + periodo
              TRUE
                SEQ
                  ssuptes ! ok -- informa nao ok detectado no teste CPU
    TRUE
      SKIP                          -- se sobressalente, nao faz nada
:

--=============================================================================
--
-- Nome         : Supervisor
--
-- Funcao       : Supervisionar o estado dos processadores da rede e
--                  disparar o reconfigurador na ocorrencia de falhas.
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : marco de 1993
--
-- Observacoes : Quando dispara o reconfigurador ja informa qual
--                  processador esta defeituoso.
--
--=============================================================================
PROC supervisor ([]CHAN OF BOOL ssuptes)
  --
  -- Declaracoes locais
  --
  INT proc :                    -- indica o numero do processador
  VAL periodo IS 30000 :        -- especifica periodo do timeout
  BOOL ErroTimeout :            -- ocorrencia de erro por timeout
  INT contador :                -- controle de fim de roteamento
  [72]BOOL infotes :            -- mensagens de informacao dos testes
  [72]TIMER timeout :           -- canais para controle do timeout
  [72]INT agora :               -- hora inicial dos timeouts
  [72]INT falha :               -- ocorrencia de falhas por proc.
  --
  -- corpo do processo
  --
  SEQ
    ErroTimeout := FALSE
    contador := 0
    PAR i = 0 FOR numproc + numespec      -- inicializa variaveis de
                                          -- controle
```

```
  falha[i] := 0
WHILE contador <> ((numproc + numespec) - numsobr)
  SEQ
    proc := 0
    -- verifica entrada e timeout por processador
    WHILE (contador <> ((numproc + numespec) - numsobr)) AND
          (proc <> (numproc + numespec))
      SEQ
        IF
          (proc >= numproc) OR (proc < (numproc - numsobr))
            ALT
              -- espera mensagem testador. Recebe TRUE se
              -- processador bom
              ssuptes[proc] ? infotes[proc]
                IF
                  -- se ja ocorreu erro incrementa contador
                  ((contador <> 0) OR ErroTimeout)
                    IF
                      infotes[proc]
                        SKIP
                      TRUE
                        contador := contador + 1
                  TRUE
                    IF
                      infotes[proc]    -- testa se processador ok
                        -- inicializa periodo para timeout
                        timeout[proc] ? agora[proc]
                      TRUE
                        SEQ
                          -- informa reconfigurador que P_proc esta
  -- defeituoso
                          suprec ! proc
                          contador := contador + 1
              -- verifica se ocorreu timeout
              timeout[proc] ? AFTER agora[proc] + periodo
                SEQ
                  -- indica ocorrencia de uma falha
                  falha[proc] := falha[proc] + 1
                  IF
                    falha[proc] >= 2     -- se falhou mais do que 2
                                         -- vezes
                      SEQ
                        -- elimina hipotese de falha transitoria
                        ErroTimeout := TRUE
                        -- informa reconfigurador que P_proc esta
-- defeituoso
                        suprec ! proc
                    falha[proc] = 1 -- em caso de falha transitoria
                      SEQ           -- segunda chance
                        -- le hora do sistema
                        timeout[proc] ? agora[proc]
                        -- soma mais 1/3 do periodo de timeout
                        agora[proc] := agora[proc] + (periodo *
                                       (1/3))
              TRUE & SKIP
```

```
                SKIP
            TRUE
              SKIP
          proc := proc + 1
:

--============================================================================
--
-- Nome         : Reconfigurador
--
-- Funcao       : Reconfigurar a rede de processadores no caso de falha,
--                isolando o processador defeituoso e alocando um novo
--                processador para assumir suas tarefas.
--
-- Autor        : Raul Ceretta Nunes
--
-- Data         : julho de 1993
--
-- Observacoes :
--
--============================================================================
PROC reconfigurador ([]INT configtab, CHAN OF ANY screen)
  --
  -- Declaracoes locais
  --
  INT procdef :                -- processador defeituoso
  INT ncontr :                 -- numero do processador controlador
  INT any :                    -- variavel auxiliar
  --
  -- corpo do processo
  --
  SEQ
    suprec ? procdef           -- fica bloqueado ate alguem falhar
    newline (screen)
    write.text.line (screen, "Falha sinalizada ao reconfigurador.")
    beep (screen)              -- sinalizacao sonora da falha
    write.full.string (screen, "Defeito no processador  ")
    write.int (screen, procdef, 0)
    newline (screen)
    IF
      -- verifica se processador defeituoso e' o controlador
      (procdef = numproc) OR (procdef = (numproc + numespec))
        SEQ
          write.text.line (screen, "Ocorreu falha no controlador. ")
          write.text.line (screen, "Falha irrecuperavel atraves de
   reconfiguracao.")
      -- verifica se processador defeituoso e' um supervisor
      ((procdef > numproc) AND (procdef < (numproc + numespec))) OR
      (procdef > (numproc + numespec))
        SEQ
          write.text.line (screen, "Ocorreu falha num servidor. ")
          write.text.line (screen, "Falha irrecuperavel atraves de
   reconfiguracao.")
      TRUE
        SEQ
```

```
      -- testa se existe processador reserva
      -- (opcao feita pelo usuario na hora do boot)
      IF
        -- reconfiguracao com sobressalente
        numsobr > 0           -- existe sobressalente
          SEQ
            -- faz a permuta do processador defeituoso pelo
            -- sobressalente
            configtab[configtab[(procdef * 4) + 1]] :=
                                  ((numproc - numsobr) * 4) + 1
            configtab[configtab[(procdef * 4) + 2]] :=
                                  ((numproc - numsobr) * 4) + 2
            configtab[configtab[(procdef * 4) + 3]] :=
                                  ((numproc - numsobr) * 4) + 3
            configtab[configtab[(procdef * 4) + 4]] :=
                                  ((numproc - numsobr) * 4) + 4
            configtab[((numproc - numsobr) * 4) + 1] :=
                                  configtab[(procdef*4) + 1]
            configtab[((numproc - numsobr) * 4) + 2] :=
                                  configtab[(procdef*4) + 2]
            configtab[((numproc - numsobr) * 4) + 3] :=
                                  configtab[(procdef*4) + 3]
            configtab[((numproc - numsobr) * 4) + 4] :=
                                  configtab[(procdef*4) + 4]
            -- isola processador defeituoso
            configtab[(procdef*4) + 1] := 0
            configtab[(procdef*4) + 2] := 0
            configtab[(procdef*4) + 3] := 0
            configtab[(procdef*4) + 4] := 0
        -- reconfiguracao sem sobressalente
        TRUE                  -- nao tem sobressalente
          IF
            numproc > 1       -- verifica se tem mais de um
                              -- processador bom
              SEQ
                -- faz o bypass dos canais conectados ao
                -- processador defeituoso
                configtab[configtab[(procdef*4) + 1]] :=
                                  configtab[(procdef*4) + 3]
                configtab[configtab[(procdef*4) + 3]] :=
                                  configtab[(procdef*4) + 1]
                configtab[configtab[(procdef*4) + 2]] :=
                                  configtab[(procdef*4) + 4]
                configtab[configtab[(procdef*4) + 4]] :=
                                  configtab[(procdef*4) + 2]
                -- isola processador defeituoso
                configtab[(procdef*4) + 1] := 0
                configtab[(procdef*4) + 2] := 0
                configtab[(procdef*4) + 3] := 0
                configtab[(procdef*4) + 4] := 0
            TRUE
              SEQ
                write.text.line (screen, "Ocorreu falha no unico
processador ativo. ")
                write.text.line (screen, "Falha irrecuperavel
```

```
                                        atraves de reconfiguracao.")
          -- mostra nova configuracao
          mostra (configtab, screen)
:

--=====================================================================
--
-- Processo principal
--
--=====================================================================
[numproc + numespec]INT xnumsobr : -- variavel auxiliar p/ o PAR de
                                   -- testadores
[numproc + numespec]INT proc :     -- variavel auxiliar p/ o PAR de
                                   -- testadores
INT Char :                         -- variavel auxiliar
INT numtotal :                     -- numero total de processadores
BOOL invalido :
SEQ
  Char := INT' '
  invalido := TRUE
  NoErro := TRUE
  optionc := TRUE
  numtotal := numproc + numespec
  PAR i=0 FOR ((numproc + numespec)*4)+1 -- inicializa tabela de
                                         -- configuracao
    gconfigtab[i] := 0
  write.text.line (screen,"-----------------------------------
                   ----------------------------------------")
  write.text.line (screen,"-----------------------------------
                   ----------------------------------------")
  write.full.string (screen, " CONFIGURACAO INICIAL: ")
  write.full.string (screen, "Numero de processadores de trabalho: ")
  write.int (screen, numproc, 0)
  newline (screen)
  write.full.string (screen, "                          Numero de
                     processadores especiais: ")
  write.int (screen, numespec, 0)
  newline (screen)
  write.text.line (screen,"-----------------------------------
                   ----------------------------------------")
  write.text.line (screen,"-----------------------------------
                   ----------------------------------------")
  WHILE invalido
    SEQ
      write.full.string (screen, "Entre com o numero de processadores
 sobressalentes da rede: ")
      read.echo.int (keyboard, screen, numsobr, Char)
      newline (screen)
      IF                   -- testa se numero de processadores e' valido
        (numsobr < 0) OR (numsobr >= numproc)
          write.text.line (screen, " Numero invalido, entre com um novo
                           numero. ")
        TRUE
          invalido := FALSE
  write.full.string (screen, "Configurar a rede em Mesh
```

```
                    (automaticamente)? Ou numa outra topologia? (m/o)")
read.echo.char (keyboard, screen, option)  -- le a option do usuario
newline (screen)
WHILE optionc
  IF
    option = (INT'o')      -- outra topologia
      SEQ
        changeconfig (gconfigtab, screen)
        mostra (gconfigtab, screen)
        optionc := FALSE
    option = (INT'm')      -- topologia em mesh
      IF                   -- testa se e' possivel conectar em mesh
        (numsobr = 0) OR ((numproc - numsobr) = 4) OR
        ((numproc - numsobr) = 9) OR ((numproc - numsobr) = 16) OR
        ((numproc - numsobr) = 25) OR ((numproc - numsobr) = 36) OR
        ((numproc - numsobr) = 49) OR ((numproc - numsobr) = 64)
          SEQ
            configrede(screen, keyboard, gconfigtab)
            optionc := FALSE
        TRUE
          SEQ
            write.text.line (screen, "Nao eh possivel conectar a rede
                                em mesh.")
            option := INT'o'
    TRUE
      SEQ
        write.text.line (screen, "Tecla nao valida. ")
        write.full.string (screen, "Deseja configurar a rede em Mesh
         (automaticamente)? Ou numa outra topologia? (m/o) ")
        read.echo.char (keyboard, screen, option) -- le a option do
                                                  -- usuario
        newline (screen)
PAR x = 0 FOR numproc + numespec -- cria varias variaveis contendo o
                          -- numero de processadores sobressalentes,
                          -- para poder usa-las em paralelo
  xnumsobr[x] := numsobr
PAR                       -- dispara processos testadores, supervisor
                          -- e reconfigurador em paralelo
  PAR x = 0 FOR numproc + numespec -- dispara testadores em paralelo
    SEQ
      proc[x] := x               -- define o numero do processador
      testador (suptes[x], proc[x], numproc, xnumsobr[x])
  supervisor (suptes)           -- dispara processo supervisor
  reconfigurador (gconfigtab, screen) -- dispara processo
                                      -- reconfigurador
newline (screen)
write.full.string (screen, "Tecle algo para retornar ao TDS ")
INT any :
read.char (keyboard, any)
newline (screen)
```

BIBLIOGRAFIA

[BLU89] BLUM, Bertrand; BURRER, Caroline. **MEGA Node an Implementation of a Coarse Grain Totally Reconfigurable Parallel Machine.** Soultz-France: Telmat Informatique, 1989. 12p.

[CHA80] CHANTAL, Robach; SAUCIER, Gabrièle. Microprocessor Funtional Testing. **IEEE Test Conference**, p.433-443, 1980.

[CHE89] CHEAN, Mengly; FORTES, Jose A. B. **FUSS: A Reconfiguration Scheme for Fault-Tolerant Processor Arrays.** In: INTERNATIONAL WORKSHOP HARDWARE FAULT TOLERANCE IN MULTIPROCESSORS, June 1989, p.30-32.

[CHE90] CHEAN, Mengly; FORTES, Jose A. B. A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays. **COMPUTER**, Los Alamitos, v.24, n.1, p.55-69, Jan. 1990.

[CLO53] CLOS, Charles. A Study of Non-Blocking Switching Networks. **Bell Technical Journal**, New York, v.32, p.406-424, 1953.

[DES78] DESPAIN, Alvin M.; PATTERSON, David A. X-tree: A Structured Multiprocessor Computer Architecture. In: SYMPOSIUM COMPUTER ARCHITECTURE, 5., Apr. 1978. **Proceedings...** [S.l.:s.n.], 1978. p.144-151.

[DUN90] DUNCAN, Ralph. A Survey of Parallel Computer Architectures. **COMPUTER**, Los Alamitos, v.23, n.2, p.05-16, Feb. 1990.

[DUT88] DUTT, Shantanu; HAYES, John P. Design and Reconfiguration Strategies for Near-Optimal K-Fault-Tolerant Tree Architectures. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 18., June 1988, Tokyo-Japan. **Proceedings...** Tokyo: 1988. p.328-333.

[DUT90] DUTT, Shantanu; HAYES, John P. On Designing and Reconfiguring k-Fault-Tolerant Tree Architectures. **IEEE Transactions on Computers**, v.C-39, n.4, p.490-503, Apr. 1990.

[DUT92] DUTT, Shantanu; HAYES, John P. Some Practical Issues in the Design of Fault-Tolerant Multiprocessors. **IEEE Transactions on Computers**, v.C-41, n.5, p.588-598, May 1992.

[ECK93] ECKER, Wolfgang; MÃRZ, Sabine System-Level Specification and Design Using VHDL: A Case Study. In: IFIP CONFERENCE ON HARDWARE DESCRIPTION LANGUAGES AND THEIR APPLICATIONS, Apr. 1993, Ottawa: Canada. **Proceedings...** Ottawa: [s.n], 1993.

[FLI89] FLIELLER, Sylvain. T-node, Industrial Version of Supernode. **Computer Physics Communications**, n.57, p.492-494, 1989.

[FLY66] FLYNN, M. J. Very High-Speed Computing Systems. **Proceedings of the IEEE**, v.54, p.1901-1909, Dec. 1966.

[FOR85] FORTES, J. A. B.; RAGHAVENDRA, C. S. Gracefully Degradable Processor Arrays. **IEEE Transactions on Computers**, v.C-34, n.11, p.1033-1044, Nov. 1985.

[GRE92] GRECO, Solange P. C. **Reconfiguração em Árvores Tolerantes a Falhas**. Porto Alegre: CPGCC da UFRGS, 1992. 221p. (Dissertação de mestrado)

[HAS88] HASAN, N.; LIU, L. Minimum Fault Coverage in Reconfigurable Arrays. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 18., June 1988, Tokyo-Japan. **Proceedings...** Tokyo: 1988. p.348-353.

[HAS85] HASSAN, A. S. M.; AGARWAL, V. K. A Modular Approach to Fault-Tolerant Binary Tree Architectures. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 15., June 1985, Ann Arbor-Michegan-USA. **Proceedings...** Ann Arbor: [s.n.], 1985. p.344-349.

[HAY76] HAYES, J. P. A Graph Model for Fault-Tolerant Computing System. **IEEE Transactions on Computers**, v.C-25, n.9, p.875-884, Sep. 1976.

[HOA78] HOARE, C. A. R. Communicating Sequencial Process. **Communications ACM** v.21, n.8, p.666-677, Aug. 1978.

[HOR81] HOROWITZ, E.; ZORAT, A. The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and VLSI. **IEEE Transactions on Computers**, v.C-30, n.4, p.247-253, Apr. 1981.

[INM84] INMOS LIMITED. **IMS T424.** Bristol: INMOS, 1984. 31p. Preliminary data.

[INM88b] INMOS LIMITED **OCCAM 2 Reference Manual.** Cambridge: Pretice Hall, 1988. 133p. (Series in Computer Science)

[INM88] INMOS LIMITED IMS T800 Transputer. In: **Transputer databook**. Bath: Bath, 1988. p.43-111

[INM91] INMOS LIMITED **The T9000 Transputer Products Overview Manual** Bristol: INMOS, 1991. 194p. (INMOS Databook Series)

[KAR89] KARTASHEV, Svetlana P.; KARTASHEV, Steven I. **Designing and Programming Modern Computer Systems v. II Supercomputing Systems: Reconfigurable Architectures.** New Jersey: Prentice Hall, 1989. 428p.

[KWA81] KWAN, C. L. **Dinamically Optimal Fault-Tolerant structures of Hierarchical Tree Systems.** Waterloo, Ont., Canada: Univ. of Waterloo, 1981. (Ph.D. Dissertation)

[KWA82] KWAN, C. L.; TOIDA, S. An Optimal 2-Fault Tolerant Realization of Simmetric Hierarchical Tree Systems. **Networks**, v.12, p.231-239, 1982.

[LAP85] LAPRIE, J. C. Dependable computing and fault tolerance: concepts and terminology. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 15., July 1985, Ann Arbor-EUA. **Proceedings...** Ann Arbor: [s.n.], 1985. p.2-11.

[LOW87] LOWRIE, M. B.; FUCHS, W. K. Reconfigurable Tree Architectures Using Subtree Oriented Fault Tolerance. **IEEE Transactions on Computers**, v.C-36, n.10, p.1172-1182, Oct. 1987.

[NEG86] NEGRINI, Roberto; SAMI, Mariagiovanna; STEFANELLI, Renato Fault tolerance techniques for array structures used in supercomputing. **COMPUTER**, Los Alamitos, v.19, n.2, p.78-87, Feb. 1986.

[NIC88] NICOLE, Denis A. **Reconfigurable transputer processor architecture.** Southamptom: Southamptom Transputer Support Centre, 1988. 18p. (ESPRIT Project 1085, Technical Report, 2)

[NUN92] NUNES, Raul Ceretta. **Um Estudo de Confiabilidade da Arquitetura do** T-NODE. Porto Alegre: CPGCC da UFRGS, 1992. 62p. (Trabalho Individual, 252)

[PAZ91] PAZZINI, Marcelo. **Especificação do TRIX**: um sistema operacional multiprocessado para transputers. Porto Alegre: CPGCC da UFRGS, 1991. 60p. (Trabalho Individual, 242)

[PER90] PERIHELION SOFTWARE **The HELIOS operating system.** Somerset: Perihelion Software, 1990. v.1

[RAG84] RAGHAVENDRA, C. S., AVIZIENIS, A.; ERCEGOVAC, M. D. Fault Tolerance in Binary Tree Architectures. **IEEE Transactions on Computers,** v.C-33, n.6, p.568-572, June 1984.

[ROS83] ROSENBERG, A. L. The Diogenes Approach to Testable Fault Tolerance Arrays of Processors. **IEEE Transactions on Computers,** v.C-32, p.902-910, Oct. 1983.

[SCH90] SCHNEIDER, FRED B. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. **ACM Computing Surveys,** v.22, n.4, p.299-319, Dec. 1990.

[SIE82] SIWIOREK, D. P.; SWARZ, R. S. **The theory and practice of reliable system design.** Beolford: Digital, 1982. 772p.

[SKI88] SKILLICORN, D. B. A Taxonomy for Computer Architectures. **COMPUTER,** Los Alamitos, v.21, n.11, p.46-57, Nov. 1988.

[TEL91a] TELMAT INFORMÁTIQUE. **The HELIOS 1.2 & T.Kernel 3.0 Configuration File.** [S.l.]: TELMAT INFORMÁTIQUE, May 1991. 12p. (Technical Report, 3)

[TEL91b] TELMAT INFORMÁTIQUE. T.Switcher 3.0 - The SwMan & SwDrv HELIOS Servers. [S.l.]: TELMAT INFORMÁTIQUE, May 1991. 18p. (Technical Report, 5)

[TEL91c] TELMAT INFORMÁTIQUE. **T.ControlBus 3.0 - The CbMan & CbDrv HELIOS Servers.** [S.l.]: TELMAT INFORMÁTIQUE, May 1991. 32p. (Technical Report, 7)

[TEL91d] TELMAT INFORMATIQUE **T-node hardware manual.** Soultz: Telmat Informatique, 1991. v.1

[TEL91e] TELMAT INFORMATIQUE **T-node software manual.** Soultz: Telmat Informatique, 1991. v.2

[THA80] THATTE, Satish M.; ABRAHAM, Jacob A. Test Generation for Microprocessors. **IEEE Transactions on Computers**, v.C-29, n.6, p.429-441, Jun. 1980.

[VER93] VERR, Bart **Systems Support for Fault Tolerant Applications in Parallel Computing Systems.** Porto Alegre: 11 Aug. 1993. mail Internet. BART@PERIHELION.CO.UK.

[WAG88] WAGNER, Flávio R.; JANSCH-PÔRTO, Ingrid; WEBER, Raul F.; WEBER, Taisy S. **Métodos de Validação de Sistemas Digitais.** In: ESCOLA DE COMPUTAÇÃO, 6., 1988, Campinas, SP. Campinas: 1988.

[WAI91] WAILLE, Philippe **Architectures Paralleles a Connectique Programmable: Reconfiguration et Routage.** Grenoble: Institut National Polytechnique de Grenoble, Sept. 1991. 184p. (Tese de Doutorado).

Informática
UFRGS

Reconfiguração no T-Node em caso de falhas.

Dissertação apresentada aos Senhores:

Prof. Dr. Claudio Luís de Amorim (UFRJ)

Prof. Dr. Celso Maciel da Costa

Prof. Dr. Philippe Olivier Alexandre Navaux

Profa. Dra. Taisy Silva Weber

Vista e permitida a impressão.
Porto Alegre, 13/01/94.

Prof. Dr. Philippe Olivier Alexandre Navaux,
Orientador.

Prof. Dr. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.