ALGORITMO DE RECONFIGURAÇÃO NA MÁQUINA T-NODE EM CASO DE FALHAS

Raul Ceretta Nunes[*]
Philippe Olivier Alexandre Navaux[**]
Ingrid Jansch-Pôrto[***]

RESUMO

Neste artigo é apresentado um algoritmo para a execução de reconfiguração na máquina T-NODE na ocorrência de falhas. T-NODE [TEL91] é uma máquina paralela que usa transputers como blocos básicos; foi projetada para aplicações de alto desempenho e não apresenta, em seu projeto original, qualquer característica especial de tolerância a falhas.

No texto, são apresentados brevemente alguns conceitos básicos de sistemas tolerantes a falhas, o ambiente arquitetural da máquina T-NODE, e a motivação para o desenvolvimento da pesquisa. Na seqüência, são descritas as principais características dos transputers e da máquina T-NODE; para esta descrição, são consideradas as necessidades e o ponto de vista de reconfiguração. Então é apresentado o algoritmo para reconfigurar a T-NODE quando forem detectadas falhas nos módulos básicos, que correspondem aos transputers e suas memórias locais. O artigo é concluído com uma discussão sobre melhoramentos possíveis e com as conclusões obtidas a partir deste trabalho.

ABSTRACT

In this paper is presented an algorithm to the execution of reconfiguration in the T-NODE machine in presence of fault. T-NODE [TEL91] is a parallel machine which uses transputers as basic building blocks; it has been designed for high performance applications and does not present, at its origin, any special behavior concerning fault-tolerance.

In the following, we briefly present some basic concepts of fault-tolerant systems, the architecture environment of the T-NODE machine, besides explaining the motivation for the developed research. In the sequence, the main characteristics of transputers and of the T-NODE machine are described; for this description, the needs and the point-of-view of reconfiguration are considered. The algorithm to reconfigure the T-NODE under fault occurrence is then presented; detection is done considering the transputers and their local memories as basic modules. The paper finishes with the discussion of possible improvements and conclusions taken from this work.

[*]Mestrando em Ciência da Computação (UFRGS), [**]Doutor-Eng. em Informática (INPG, França), [***]Doutor-Eng. em Microeletrônica (INPG, França).
Endereço: Pós-Graduação em Ciência da Computação e Instituto de Informática - UFRGS - Caixa Postal 15064 91501.970 Porto Alegre - Brasil - Telefone: 55-51-3368399 ou 2281633; fax: 55-51-3365576
E-mails: ceretta@inf.ufrgs.br, navaux@inf.ufrgs.br, ingrid@inf.ufrgs.br

1. INTRODUÇÃO

Visando atingir alto desempenho, os computadores modernos incluem vários recursos que permitem concorrência e paralelismo. A existência destes recursos modificou bastante a estrutura convencional antes empregada, embasada no modelo de Von Neumann. A máquina T-NODE [TEL91], concebida para processar algoritmos paralelos com alto desempenho, é um exemplo desta tendência: é uma máquina altamente paralela, fracamente acoplada, baseada em *transputers* (maiores detalhes são tratados na seção 2). Uma de suas principais características é sua habilidade em alterar a topologia da rede de interconexão, entre os diversos elementos processadores, através de uma chave eletrônica (programável), o que lhe confere a reconfigurabilidade como característica.

A diversidade de soluções em arquitetura motivou alguns pesquisadores a proporem taxonomias para as arquiteturas paralelas. Uma taxonomia bastante conhecida, apresentada por Michel J. Flynn [FLY66], classifica as máquinas de acordo com o controle de instruções e fluxo de dados, incluíndo-as em um dos seguintes grupos: SISD, SIMD, MISD e MIMD. De acordo com a classificação de Flynn, a T-NODE constitui-se em um sistema MIMD, embora suas propriedades de reconfigurabilidade também permitam operação similar a dos sistemas SIMD e MISD.

Além do desenvolvimento da arquitetura de computadores pela inserção de paralelismo, outras propriedades como confiabilidade[1] e disponibilidade[2] também surgem como necessidades, devido à dependência crescente dos usuários com relação aos sistemas computacionais. O aumento de confiabilidade e de disponibilidade resulta em maior segurança de funcionamento (ou "dependabilidade", como começa a ser traduzido o termo *dependability*). A segurança de funcionamento de um sistema é a propriedade que permite depositar confiança justificada no serviço que ele fornece [LAP85]. O desenvolvimento de técnicas de tolerância a falhas para sistemas computacionais iniciou com as aplicações críticas como, por exemplo, controle de tráfego aéreo, onde falhas no sistema podem causar prejuízos irreparáveis. Este não é o caso na aplicação prevista para a T-NODE; mas o uso de técnicas de tolerância a falhas evita longos tempos de processamento com o uso de dados incorretos ou que possam produzir saídas incorretas. Na análise da arquitetura da máquina em questão [NUN92], verificou-se que não houve preocupação específica com aspectos de tolerância a falhas, prevalecendo aspectos de desempenho, conforme já exposto.

O projeto de sistemas tolerantes a falhas adota diferentes estratégias. Em [SIE82] é apresentada uma taxonomia para estas diferentes estratégias, onde o autor classifica os sistemas tolerantes a falhas em três grupos (cuja divisão não é rígida): detecção de falhas, redundância para mascaramento e redundância dinâmica (ver figura 1). Para detecção de falhas, são previstas

[1]Confiabilidade de um sistema, R(t), como função do tempo, é a probabilidade de que o sistema permaneça em funcionamento durante o intervalo [0,t], considerando que ele esteja operacional em t=0 [SIE82].
[2]Disponibilidade de um sistema, A(t), como função do tempo, é a probabilidade de que o sistema esteja operacional no instante de tempo t [SIE82].

técnicas tais como códigos de detecção de erros, lógica auto-testável e livre de falhas, temporizadores *watch-dog* e limites de tempo, verificações de consistência e de capacidades, e duplicação. O uso de redundância para mascaramento inclui técnicas como redundância n-modular (RnM) com votação, códigos de correção de erros e lógica de mascaramento. Entre as técnicas de redundância dinâmica, pode-se encontrar o uso de duplicação reconfigurável, RnM reconfigurável, unidades-estepe ou reservas para substituição, degradação gradual, reconfiguração e recuperação.

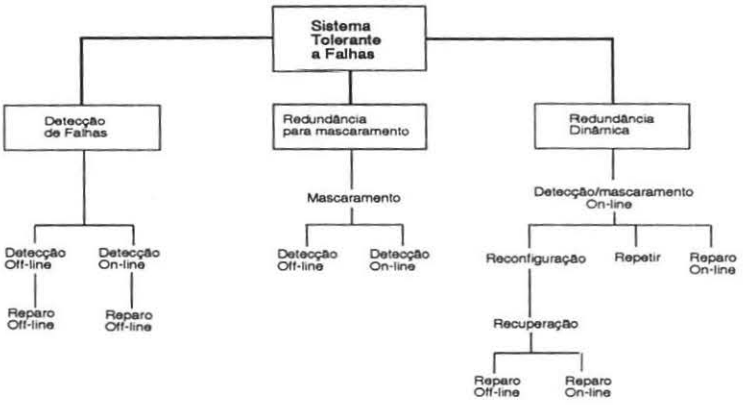

Figura 1: Taxonomia de estratégias para sistemas tolerantes a falhas

As técnicas de detecção de falhas fornecem meios para que se perceba a presença potencial de erros no sistema digital. Adicionalmente, a detecção de falhas proporciona um acréscimo na disponibilidade do sistema, graças ao diagnóstico mais rápido das falhas. Entretanto, isoladamente, a detecção de falhas não aumenta o parâmetro confiabilidade do sistema, considerando o valor matemático calculado para a função; na prática, esta idéia fica mais clara se for expressa como: a simples detecção de falhas não aumenta a confiabilidade ou a correção das respostas obtidas do sistema. Mas as técnicas de mascaramento de falhas aumentam a confiabilidade do sistema, uma vez que permitem o seu funcionamento correto mesmo em presença de falhas. Uma forma alternativa para se obter aumento na confiabilidade utiliza redundância dinâmica. As técnicas de redundância dinâmica envolvem a reconfiguração dos componentes do sistema, em resposta à ocorrência de falhas. A reconfiguração evita que as falhas produzam efeitos nocivos sobre a operação do sistema. Em vários casos, a reconfiguração se estende à desconexão das unidades afetadas no sistemas; com o uso de uma associação entre técnicas de mascaramento e reconfiguração dinâmica, a remoção de componentes falhos pode ser adiada até que um determinado número de falhas tenha se acumulado e, se não for removidas, comecem a ameaçar a segurança do sistema [SIE82].

Considerando-se uma máquina de múltiplos nodos interconectados por uma rede de chaves programáveis, como é o caso da T-NODE, as técnicas de tolerância a falhas que mais se adequam são as de redundância dinâmica, pois elas podem tirar proveito das características de reconfiguração dinâmica da rede.

Conforme exposto anteriormente, a máquina T-NODE não foi desenvolvida para atividades críticas: o objetivo principal, em sua especificação, foi a obtenção de alto desempenho. Mesmo assim, o uso de alguns princípios de tolerância a falhas pode dar bons resultados sem afetar significantemente seu desempenho e o custo da máquina.

Vários autores propuseram algoritmos de reconfiguração, visando obter tolerância a falhas, aplicáveis a sistemas multiprocessadores. Entretanto, grande parte destes algoritmos é projetada para topologias específicas de redes como, por exemplo, árvores ([DUT88], [LOW87]) ou *arrays* ([NEG86], [HAS88]). Neste artigo, é proposto um algoritmo para efetuar a reconfiguração da máquina T-NODE em seqüência a detecção de falhas, o que visa aumentar sua segurança de funcionamento ou dependabilidade. Este algoritmo é independente de topologia. As técnicas de detecção empregadas asseguram a cobertura das falhas permanentes do processador; a detecção de falhas em interconexões não é totalmente coberta pois depende da possibilidade de modelagem destas como falhas de comunicação de algum processador. Os procedimentos foram todos especificados de modo a não alterar significativamente o desempenho do sistema original.

2. *TRANSPUTERS* E A MÁQUINA T-NODE

Dentre as arquitetura paralelas, é a arquitetura multiprocessadora que mais interesse têm atraído nos últimos tempos. Esta arquitetura caracteriza-se por possuir diversos processadores comunicando-se através de uma memória global ou um barramento de mensagens coordenados por um sistema operacional único com o objetivo de processar uma determinada tarefa.

Os multiprocessadores dividem-se em diversos tipos, dependendo da topologia de interconexão, sendo que ultimamente a forma cúbica tem sido objeto de maiores pesquisas gerando as máquinas hipercúbicas.

A máquina T-NODE enquadra-se com uma máquina multiprocessadora que se comunica por troca de mensagens, sendo portanto fracamente acoplada. Quanto à interconexão, a T-NODE pode se configurar segundo diversos tipos de topologias; no entanto ela se adapta bastante facilmente à topologia hipercúbica. Sua capacidade de reconfiguração é dinâmica permitindo que a máquina altere sua topologia em estado de processamento, mas o que, na prática, não é usual. Normalmente a máquina T-NODE é empregada como configuração estática, isto é, antes de processar uma determinada tarefa, a estrutura de interconexão é configurada e permanece assim até o final da execução, quando sua interconexão pode ser reprogramada.

Nesta seção, são descritas as principais características dos *transputers* e da máquina T-NODE, visando possibilitar o entendimento posterior das opções efetuadas a nível do algoritmo.

2.1 O *transputer*

O *transputer* [INM88a] consiste de um circuito VLSI composto de processador, memória e canais de comunicação com outros *transputers*. O *transputer* de 32-bits, IMS T800, cuja estrutura básica é mostrada na figura 2, e que se constitui em uma versão melhorada do IMS T414 pelo acréscimo de um processador inteiro de ponto-flutuante que opera em alta velocidade, é usado como nodo básico da máquina T-NODE. Seu projeto foi desenvolvido no contexto do projeto ESPRIT número P1085, com a finalidade de se tornar o bloco básico da máquina SUPERNODE ([NIC88], [INM84]).

Os *transputers* podem ser programados através da maior parte das linguagens de alto nível, e são projetados para assegurar que os programas compilados serão eficientes. Quando se faz necessário explorar concorrência, a linguagem OCCAM pode ser usada como uma ferramenta para ligar módulos escritos nas linguagens escolhidas.

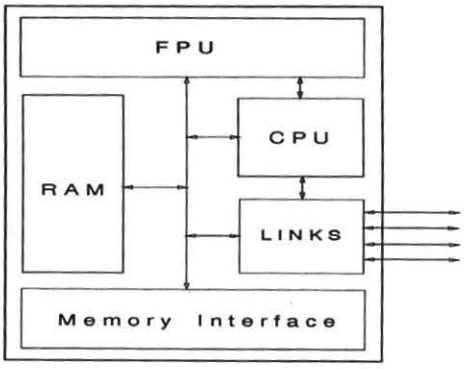

Figura 2: Diagrama em blocos do *transputer* IMS T800

O OCCAM [POU88] é uma linguagem de programação de alto nível baseada na teoria de processos seqüenciais comunicantes (*communicating sequential processes*, CSP) proposto por Hoare em [HOA78]. Nesta teoria, Hoare supôs um conjunto de processos seqüenciais sendo executados concorrentemente e comunicando-se com os demais através de mensagens. Em OCCAM, cada par de processos concorrentes comunica-se através de um canal unidirecional comum. A comunicação é sincronizada e ocorre quando ambos estão prontos (de forma similar ao *handshake* , usado como método de comunicação em sistemas de hardware).

A linguagem OCCAM fornece um ambiente para o projeto de sistemas concorrentes que usem *transputers*. Para tirar proveito da arquitetura do *transputer*, o sistema completo pode ser programado em OCCAM. Esta opção proporciona todas as vantagens de uma linguagem de alto-nível, eficiência máxima na execução do programa e a possibilidade de usar características especiais do *transputer*. O trabalho do projetista do sistema é facilitado devido à relação entre a linguagem e a arquitetura do processador.

Em OCCAM, cada processo é tipicamente um conjunto de instruções seqüenciais que inicia, realiza um certo número de ações e então termina. Uma ação pode corresponder a um conjunto de processos seqüenciais ou a processos paralelos. Como um processo é por si só composto por processos, sendo que alguns deles podem ser executados em paralelo, um processo pode conter concorrência interna (esta situação é ilustrada esquematicamente na figura 3). Um programa rodando no *transputer* é formalmente equivalente a um processo OCCAM; portanto, uma rede de *transputers* pode ser descrita diretamente como um programa OCCAM.

Cada *transputer* implementa em hardware os conceitos de concorrência e de comunicação do OCCAM. Assim, determinado número de processos é habilitado para execução conjunta, partilhando o tempo do processador, por meio de um escalonador implementado em microcódigo no *transputer*. Os processos podem ser definidos como de alta- ou de baixa prioridade. Os

processos de alta prioridade preemptam os de baixa prioridade. Em geral, os processos de alta prioridade executam em pequenos intervalos de tempo; durante sua execução, eles são processados até que necessitem esperar por uma comunicação, um período de tempo ou até que sejam concluídos. Os processos de baixa prioridade são periodicamente particionados no tempo para proporcionar uma distribuição eqüitativa do tempo do processador entre as tarefas computacionalmente intensas. O escalonador controla os dois níveis através de filas de processos, uma para cada nível de prioridade.

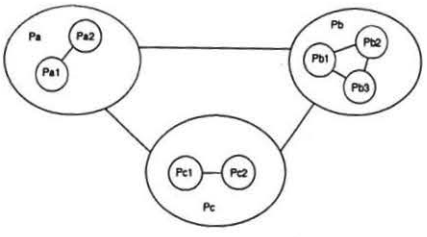

Figura 3: Processos OCCAM

A comunicação entre processos é efetuada por meio de canais, que podem ser implementados por uma única palavra de memória (*soft*), quando os processos estão executando no mesmo *transputer*, ou implementados em ligações ponto-a-ponto (*hard*), quando os processos estão executando em diferentes *transputers*. O número de canais *soft* é limitado somente pelo tamanho da memória, enquanto que o número de canais *hard* é fixo e igual a 4 (quatro).

2.2 A MÁQUINA T-NODE

T-NODE é uma máquina multiprocessadora de alto desempenho baseada em *transputers*, com alta modularidade permitindo fácil expansão do sistema. Sua arquitetura pode ser reconfigurada em modo estático ou dinâmico. A reconfiguração é feita pela programação da rede de interconexão existente entre os processadores. Esta rede faz o chaveamento entre os canais *hard*, permitindo a construção de diferentes topologias como *arrays*, *pipelines*, árvores, hipercubos ou de outras combinações quaisquer. A seguir, são apresentadas as principais características desta máquina e uma descrição detalhada da rede de interconexão.

A T-NODE pode ser construída usando de 8 a 1024 *transputers* de trabalho[3]. O módulo básico é uma rede reconfigurável com 36 *transputers*, sendo que um destes é o *transputer* controlador (ver figura 4). Este módulo tem uma estrutura de controle e facilidades de comunicação com o meio externo, e é denominada de nodo. Para construir máquinas de maior porte, a partir desta estrutura do nodo básico, basta trocar algumas conexões da rede de interconexão e/ou substituir alguns (normalmente 2) grupos de 8 *transputers* de trabalho por conjuntos de *buffers* que servirão a uma rede internodo. Uma visão geral esquemática desta

[3]Os *transputers* de trabalho (*worker transputers*) são empregados pelo usuário para rodar os seus processos; os *transputers* controladores, referidos no texto apenas como controladores, tem a tarefa de gerenciar o sistema.

organização arquitetural pode ser vista na figura 5. Este é o princípio da arquitetura hierárquica usada para produzir máquinas de múltiplos nodos. A hierarquia é implementada em hardware através de um barramento de controle com um protocolo (*handshake*) mestre-escravo (isto corresponde à linha representada em negrito na figura 5).

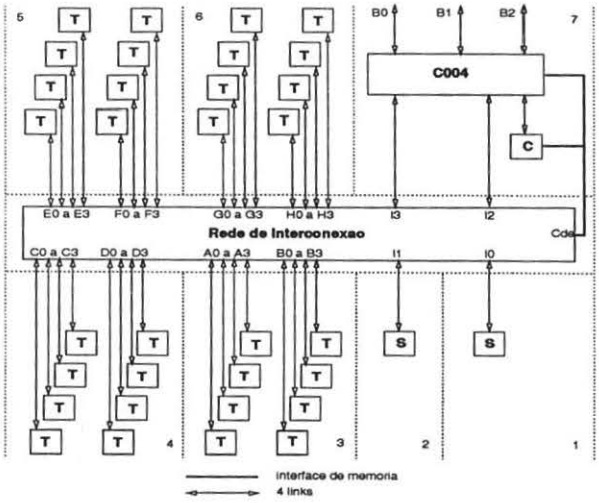

Figura 4: Organização do nodo básico

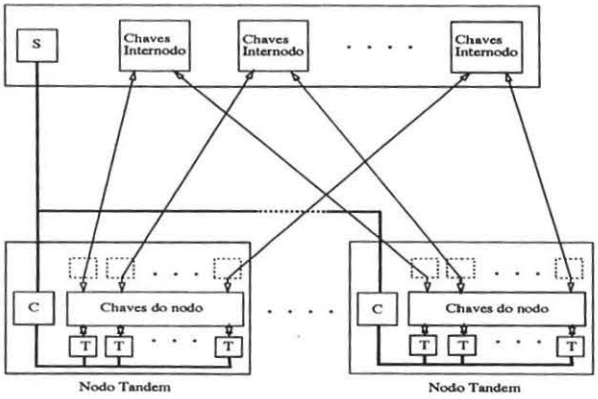

Figura 5: Arquitetura da T-NODE

Cada nodo tem um controlador que controla as mensagens do sistema e as chaves de interconexão programadas dentro do nodo. O controlador pode operar como escravo ou como mestre de outro controlador, dependendo de sua posição hierárquica na estrutura, mas somente como mestre com relação a trabalhadores. Várias configurações desta arquitetura foram produzidas mantendo-se o conceito de nodo: o nodo básico, o nodo TANDEM e o MEGA-NODE.

O nodo básico pode ter de 8 a 32 *transputers* de trabalho ligados por uma rede gerenciada pelo *transputer* controlador. O nodo TANDEM é composto por dois nodos básicos, aceitando até 64 nodos de trabalho. Uma das placas controladoras é definida como sendo escrava da outra. O MEGA-NODE é baseado em um conjunto de nodos (até 32 nodos TANDEM) interconectados por uma chave internodos; é possível construir máquinas de até 1024 *transputers* de trabalho. Um *transputer* supervisor gerencia todos os controladores-mestre dos nodos TANDEM.

Em um nodo básico, a rede de interconexão consiste de um par de chaves, sendo que cada uma é funcionalmente equivalente a uma *crossbar* de 72x36. Esta chave é capaz de implementar qualquer topologia de rede entre *transputers* de uma maneira rearranjável. Ambos circuitos de chaveamento são controlados pelo controlador.

O *transputer* tem quatro canais usados para a comunicação com os seus vizinhos, os quais recebem as seguintes denominações: Norte, Leste, Oeste e Sul. Todas as saídas dos canais norte e leste (36 de cada) dos *transputers* do nodo básico são ligados às entradas dos circuitos de chaveamento, e as saídas deste circuito são ligadas às entradas dos canais sul e oeste (36 de cada um). Todas as saídas dos canais sul e oeste dos *transputers* do nodo básico são ligados às entradas de outro circuito de chaveamento, e as saídas deste circuito são ligadas às entradas dos canais norte e leste. Aplicando este modelo, as seguintes conexões podem ser estabelecidas em um nodo básico: Norte↔Sul, Norte↔Oeste, Sul↔Leste, Leste↔Oeste. Na figura 6, são mostradas estas conexões. As linhas pontilhadas mostram os *straps*, ou *swap-plugs* (conexões diretas de chave-a-chave) que são usados para compor o nodo TANDEM.

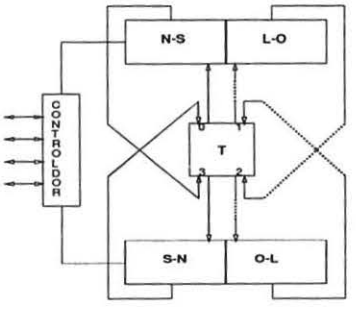

Figura 6: Circuito de chaveamento do nodo básico

A rede de interconexão do nodo TANDEM é semelhante a do nodo básico: a diferença que existe é a limitação do número de conexões. Isto provém do fato de que no nodo TANDEM, os conectores do tipo *swap-plugs* são usados para ligar a metade dos canais em cada nodo básico. A

mudança de um conjunto para o outro causa a troca entre os grupos de canais. Ainda, há uma divisão entre as redes norte-sul e leste-oeste que impede as conexões N↔O e S↔L.

Em ambos, no nodo básico e no nodo TANDEM, as redes são totalmente rearranjáveis, uma vez que são usadas ligações *crossbar*. No MEGA-NODE, a rede de interconexão dos *transputers* é baseada na teoria de multiestágios de Clos [NIC88], que a torna totalmente rearranjável mas bloqueante. A chave MEGA-NODE permite somente as conexões Norte-Sul e Leste-Oeste. Isto resultará em diferentes características (considerando as características do T-NODE e do nodo TANDEM como parâmetros básicos) com relação ao processo de reinicialização, que pode ser executado após a reconfiguração.

3. O ALGORITMO DE RECONFIGURAÇÃO

Um sistema pode ser reparado ou pela substituição do módulo falho ou pela reconfiguração da estrutura do sistema / distribuição de carga de trabalho, eliminando logicamente o módulo. A substituição do módulo recompõe o funcionamento do sistema para operação plena, mas necessita de módulos adicionais, não usados em operação normal.

O procedimento de reconfiguração proposto neste artigo usa a característica de total reconfigurabilidade de máquina T-NODE, a fim de eliminar as falhas que ocorreram no sistema pelo isolamento do módulo falho. O procedimento de chaveamento é efetuado no nível de processadores com suas respectivas memórias - este par é considerado aqui como um módulo - uma vez que a T-NODE é uma máquina fracamente acoplada, sendo a detecção de falhas executada a nível destes módulos. A seguir, são descritos os principais objetivos e os problemas encontrados no decorrer deste trabalho, enfatizando-se as decisões tomadas para resolvê-los.

Uma vez que se pretende tolerar k falhas sem que ocorra degradação de desempenho, são necessários k módulos-reserva[4] (denominados a seguir de reservas, simplesmente). Para não restringir o número de processadores disponíveis para o usuário, permite-se que o número de reservas seja definido por ele próprio quando o sistema é inicializado. No limite inferior, com o número de reservas igual a zero, pode ser considerada a possibilidade de redistribuir a carga dos módulos falhos entre os demais; entretanto este caso não é estudado neste artigo. Considera-se que a máquina está operando em modo monousuário, pois só se pode admitir o uso da máquina para aplicações críticas neste modo. Para outras aplicações, pode-se prever a operação em modo multiusuário mas, neste caso, o gerenciador é responsável pela definição dos módulos-reserva pois, nesta hipótese, a especificação deste número é dependente do conjunto de aplicações.

O algoritmo é executado em dois passos diferentes: um passo testa os módulos com o objetivo de detectar falhas; o outro passo é o de reconfiguração, que isola a falha e reestrutura o sistema. O passo de teste é implementado como um conjunto de processos idênticos (processos testadores) que são executados em paralelo, um em cada *transputer* de trabalho. A existência de diversos processos testadores tornou necessária a implementação de um processo (supervisor) para supervisioná-los e inicializar o passo de reconfiguração que é implementado por outro processo (reconfigurador). O algoritmo executa os três processos concorrentemente no sistema;

[4]Módulos-reserva são módulos que não participam de estrutura lógica disponível da máquina, mas que existem na estrutura física; eles podem assumir uma função lógica quando for necessário.

sua implementação foi realizada para uma configuração de nodo básico, e pode ser facilmente adaptada para o nodo TANDEM ou para o MEGA-NODE.

O **processo testador** (a seguir denominado de "testador") roda um conjunto de testes curtos, para evitar degradação significativa no desempenho do sistema, que verificam a possível ocorrência de falhas. Existe número idêntico de testadores e de módulos da rede; cada testador opera em um módulo distinto visando assegurar cobertura de falhas para toda a rede. O tamanho destes testes pode ser modificado, dependendo dos níveis de cobertura desejados. Testes longos podem resultar em degradação de desempenho excessiva, uma vez que este processo compete com os outros processos do usuário no mesmo módulo. Também por esta razão, os processos testadores são programados para serem executados somente em intervalos de tempo pré-determinados. Adicionalmente e pelas razões já expostas, o tempo que eles permanecem ativos afeta o atraso global de processamento do sistema. O gerenciamento destes períodos de tempo é feito por meio de um temporizador da linguagem, que no trabalho aqui relatado é a OCCAM2 [INM88b]. O algoritmo que implementa o **processo testador** é descrito conforme segue. A forma escolhida para a descrição apresenta estrutura correspondente à implementada em OCCAM.

```
-- Processo testador
SEQ
   .. inicializa alta prioridade
   WHILE modulo_ok
      SEQ
         ... testa a ucp                                    --ucp = unid.central de proces.
         IF
            ucp_nao_ok
               SEQ
                  ... repete o teste
                  IF
                     ucp_nao_ok
                        canal_supervisor_testador ! msg_de_nao_ok      --envia msg ao superv.
                     TRUE
                        SKIP
            TRUE
               SEQ                                                     --se a ucp não está ok
                  ... testa a upf                                      --upf = unid. ponto flut.
                  IF
                     upf_nao_ok
                        SEQ
                           ... repete o teste
                           IF
                              upf_nao_ok
                                 canal_supervisor_testador ! msg_de_nao_ok
                              TRUE
                                 SKIP
                     TRUE
                        SEQ
                           ... testa a memoria
                           IF
                              memoria_nao_ok
                                 SEQ
                                    ... repete o teste
                                    IF
                                       memoria_nao_ok
                                          canal_supervisor_testador ! msg_de_nao_ok
                                       TRUE
                                          canal_supervisor_testador ! msg_de_ok
                              TRUE
                                 SKIP
      ... aguarda periodo de tempo especificado
```

Quando um erro é detectado por um dos testes, o teste é repetido a fim de eliminar a hipótese de que ele poderia ter sido causado por uma falha transitória, antes de informar ao supervisor que o módulo está falho. Os procedimentos previstos para os processos supervisor e reconfigurador consideram a falha detectada como permanente.

O **processo supervisor** é o responsável pelo monitoramento das mensagens enviadas pelo testador, as quais informam acerca do estado do módulo. Ele associa a cada entrada uma variável de limite de tempo (*timeout*), a qual permite a detecção de falhas que poderiam impedir que o testador instalado no módulo falho enviasse mensagens de estado. A fim de evitar a ocorrência de um limite de tempo quando um processo está esperando na fila do escalonador para ser executado[5], o testador é executado em alta prioridade, pois nesta classe os processos são curtos e não são desescalonados por outros, nem mesmo devido a limites de tempo.

Existe somente um processo supervisor e é executado no controlador do nodo. Esta idéia de centralização é contrária aos princípios da tolerância a falhas mas é inevitável porque o hardware original da máquina implementa uma estrutura de controle hierárquica (como visto anteriormente na seção 3).

O supervisor chama o processo reconfigurador quando ocorre uma detecção, informando qual módulo está falho.

A seguir, é apresentado o algoritmo que descreve o **processo supervisor**:

```
-- Processo supervisor
WHILE TRUE
   ALT
      watchdog & canal_supervisor_testador? msg_testador          --espera msg dos testadores
         IF
            msg_testador = ok
               ... inicializa contador de timeout
            TRUE
               SEQ
                  ... marca modulo como defeituoso
                  canal_supervisor_reconfigurador ! msg_modulo_defeit.     --emite msg p/ reconf
      watchdog ? canal_timeout ? AFTER hora_atual + periodo_timeout        --espera timeout testador
         SEQ
            ... marca modulo com defeituoso
            canal_supervisor_reconfigurador ! msg_do_modulo_defeit.        --emite msg p/ reconf.
                                                                           --ident. modulo falho
```

O **processo reconfigurador**, responsável pela programação das chaves, é chamado pelo supervisor quando é detectado um módulo falho. Na chamada, o supervisor informa ao reconfigurador qual módulo está defeituoso.

Após a detecção de uma falha e do reconfigurador ser informado sobre a localização desta, é necessário proceder ao isolamento da falha e substituir o módulo por um reserva. Muitos algoritmos de reconfiguração preocupam-se com a obtenção de um grande número de reservas pelo incremento de arcos redundantes[6] ([DUT88],[LOW87]); esta não é a opção adequada para o caso aqui tratado porque a rede de interconexão é totalmente rearranjável, logo a redundância dos

[5]Variáveis correspondentes a limites de tempo podem ser inicialmente estimadas; elas podem ser ajustadas posteriormente com base na experiência prática. Elas não são críticas com a hipótese de inicialização completa do sistema após a detecção de falha, mas tem a tendência a serem mais críticas quando são executados os procedimentos de recuperação, devido ao espalhamento dos danos.

[6]Este tipo de enfoques adicionam ligações e/ou arcos redundantes na estrutura básica. Arcos redundantes podem assegurar a conectividade da estrutura.

arcos está implícita pela possibilidade de reprogramar a rede. Esta característica da rede permite o uso do reserva como substituto para qualquer módulo ativo no sistema sem custo adicional.

O algoritmo que implementa o **processo reconfigurador** corresponde à seguinte descrição:

```
-- Processo reconfigurador
SEQ
  canal_supervisor_reconfigurador ? msg_supervisor          --aguarda msg do superv.
  IF
    nodos-reserva > 0                                       --verifica se existe reserva
      SEQ
        ... modifica tabela de recursos
        ... modifica tabela de roteamento
        ... reprograma chaves de acordo com as tabelas
        ... reinicializa o sistema
    TRUE
      ... informa ao usuario sobre a ocorrencia de falha e da impossibilidade de reconfiguracao
```

Devido às características do OCCAM, o processo de espera de mensagens é facilmente implementado com uma instrução de recepção por um canal de comunicação, uma vez que esta bloqueia o processo inteiro até que o remetente da mensagem proceda à sua atividade.

Após receber uma mensagem informando uma falha, o processo verifica se há módulos-reserva. Se a resposta for negativa, o usuário é informado a respeito da falha e da impossibilidade do sistema em se recuperar. Se há módulo-reserva, o reconfigurador busca o arquivo que contém a descrição dos recursos disponíveis, e executa a substituição do módulo falho por um reserva modificando suas tabelas (informação mais detalhada sobre este arquivo pode ser encontrada em [TEL90]).

A modificação das tabelas de roteamento e a reprogramação de chaves é uma conseqüência da combinação dos arquivos de configuração definidos pelo usuário, com o arquivo de recursos disponíveis da máquina, resolvidos pelo sistema operacional.

Com a reinicialização completa do sistema, que corresponde à situação aqui considerada, o rearranjo completo da rede de interconexão do T-NODE é suficiente para assegurar a conexão da nova rede, após a substituição do módulo falho pelo reserva. Entretanto, nos casos em que a recuperação a partir de um ponto intermediário for desejável, sem reinicializar o sistema, será importante considerar o bloqueio da rede existente no MEGA-NODE quando forem definidas novas conexões entre os módulos.

4. CONCLUSÕES

O algoritmo proposto tem duas características principais: suporta k falhas no sistema, onde k corresponde ao número de módulos-reserva, e opera diferentes tipos de topologias, pois os processadores estão interconectados por uma rede programável. A principal diferença desta proposição para outros algoritmos existentes é o fato de suportar topologias múltiplas.

Nesta versão do algoritmo, o processo de reinicialização após uma ocorrência de falha e a reconfiguração são feitas a partir do estado inicial, embora perca todos os resultados intermediários, pois isto resulta em um comportamento adequado considerando-se parâmetros de confiabilidade. Entretanto, do ponto de vista de desempenho da reconfiguração, isto pode ser

encarado como inconveniente. Este problema pode ser minimizado com o uso de procedimentos de recuperação como, por exemplo, a inclusão de pontos de verificação (*checkpoints*) e o uso de técnicas de *rollback* em sistemas multiprocessadores; para este caso, a degradação do sistema, quando é necessária reinicialização, será definida pela consistência dos pontos de recuperação.

Adicionalmente, um outro ponto a explorar, é o uso do barramento de controle da máquina para a troca de mensagens entre os processos do algoritmo, o qual pode minimizar o nível de interferência com outras atividades.

REFERÊNCIAS

[DUT88] DUTT, S. and HAYES, J. P. Design and Reconfiguration Strategies for Near-Optimal *K*-Fault-Tolerant Tree Architectures. **International Symp. On Fault-Tolerant Computing**, FTCS-18. New York: IEEE, 1988.

[FLY66] FLYNN, M. J. Very High-Speed Computing Systems. **Proceedings of the IEEE**, n. 54, Dezembro 1966.

[HAS88] HASAN, N. and LIU, L. Minimum Fault Coverage in Reconfigurable Arrays. **International Symp. On Fault-Tolerant Computing,** FTCS-18. New York: IEEE,1988. p.348-53

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

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

[INM88a] INMOS LIMITED. IMS T800 Transputer. In: **Transputer databook**. Bath: Bath, 1988. p.43-111

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

[LAP85] LAPRIE, J.C. Dependable computing and fault-tolerance: concepts and terminology. In: **International Symp. On Fault-Tolerant Computing**, FTCS-15. New York: IEEE, 1985. p.2-11

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

[NEG86] NEGRINI, R.; SAMI, M.; STEFANELLI, R. Fault tolerance techniques for array structures used in supercomputing. **Computer**, v.19, n.2, Fevereiro 1986. p.78-87

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

[NUN92] NUNES, R.C. **Um estudo de confiabilidade da arquitetura do T-NODE.** Porto Alegre: CPGCC da UFRGS, 1992. 62p. (Trabalho Individual, n.252)

[POU88] POUNTAIN, D. and MAY, D. **A tutorial introduction to OCCAM programming.** BSP Professional Books, 1988.

[SIE82] SIEWIOREK, D.; SWARZ, R. **The theory and practice of reliable system design.** Bedford: Digital, 1982. 772p.

[TEL90] TELMAT INFORMATIQUE. **The Configuration File for Standalone Enviroments.** Technical Report n.2, Maio 1990. 9p. (In: T-NODE Technical Reports)

[TEL91a] TELMAT INFORMATIQUE. **T-NODE hardware manual.** Soultz: Telmat Informatique, 1991. v.1