UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE TELEMÁTICA

PROCESSAMENTO A FLUXO DE DADOS TOLERANTE A FALHAS EM UM COMPUTADOR PARALELO

Por : JORGE LUIZ E SILVA

Orientador : Prof. Dr. SHUSABURO MOTOYAMA

Co-Orientador : Prof. Dr. CLAUDIO KIRNER

Tese apresentada à Faculdade de Engenharia Elétrica, da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do Título de Doutor em Engenharia Elétrica.

Dezembro de 1992

Este exemplar corresponde à redação final da defendida por Jorge Luiz e Silva e aprovada pela Comissão Julgadora em 18 / 12 / 1992.

Orientador

Entrega o teu caminho ao Senhor,

confia nele, e ele tudo fará

(Sl.37:5)

Crer é tornar possível o impossível

À Beth, Lia e Lyse

que por amor superaram todas

as dificuldades durante este tempo

AGRADECIMENTOS

Ao Prof. Dr. Shusaburo Motoyama pela orientação, amizade, disponibilidade e confiança a mim dedicada.

Ao Prof. Dr. Claudio Kirner, amigo e co-orientador, que de maneira muito natural sempre nos incentivou dizendo: "podemos nos comparar a um caminhão de pedras, que ao longo do caminho vai recebendo e perdendo pedras. Se nos preocuparmos com as que caem, não teremos tempo para apanhar as que virão, que são as que encherão o caminhão no final do caminho". Agradeço portanto ao Claudio o apoio na conquista de mais esta pedrinha.

À Ana Sígoli Fernandes Matheus, pelos desenhos.

Aos amigos que nos momentos mais difíceis ajudaram a carregar o fardo que parecia mais pesado.

Aos colegas do DC que direta ou indiretamente colaboraram para a concretização deste trabalho.

À CAPES, ao Departamento de Computação da UFSCar e ao Departamento de Telemática da FEC-UNICAMP, pelo apoio logístico.

RESUMO

Este trabalho teve como objetivo analisar e desenvolver Programação a Fluxo de Dados Tolerante a Falhas em um computador paralelo estruturado recursivamente (CPER).

O CPER é uma máquina paralela que possui uma estrutura básica constituída de N Elementos de Processamento (EP), interligados através de um barramento paralelo de alta velocidade, correspondendo ao barramento nível 1. Várias destas estruturas básicas interligadas através de um outro barramento constitui a estrutura do barramento nível 2. O uso recursivo desta mesma regra irá compor a estrutura hierárquica do CPER, que permite expansão, flexibilidade e alto grau de Tolerância a Falhas.

Entretanto esta arquitetura pode não ser eficiente, se não utilizarmos conceitos de programação paralela.

Propusemos então a Programação a Fluxo de Dados na estrutura hierárquica do CPER.

Inicialmente apresentamos o hardware do CPER, e suas estruturas de alto desempenho. Em seguida, mostramos como o CPER opera para executar programas a fluxo de dados de forma dinâmica e mostramos também uma solução de software para garantir Tolerância a Falhas na programação a fluxo de Dados.

Finalmente apresentamos um simulador (FDsim), e uma análise estocástica, que tiveram como objetivo mostrar viabilidade e eficiência da máquina paralela proposta.

ABSTRACT

In this work is analyzed the fault tolerant data flow processing in a computer based on hierarchical parallel buses. In this architecture, a set of N processors, each one called processing element (PE), are interconnected to a bus constituting a basic structure or cluster. The set of the buses of these basic structures correspond to the parallel buses level 1. Several basic structures of level 1 are interconnected to a bus constituting a cluster for the parallel buses level 2. The same idea can be recursively used to get parallel buses level M. This computer architecture based on hierarchical parallel buses permits the expansion flexibility and is highly fault tolerant computer.

However, this architecture may not be efficient. If it is not utilized parallel programming concept.

In this work we proposed the data flow processing for the execution of the programs in hierarchical parallel buses machine. First of all, it is proposed hardware solution to get high throughput for inter-clusters and inter-levels communications in hierarchical parallel buses machine. Following, it is discussed how the parallel buses computer can work as a dynamic data flow machine and it is proposed software solution to get fault tolerant processing. Finally, a software simulation is carried out to show the feasibility and efficiency of proposed parallel buses machine.

SUMÁRIO

CAPÍTULO 6

O SIMULADOR DE PROGRAMAS A FLUXO DE DADOS NA MFDD (FDsim)

CAPÍTULO 7

AVALIAÇÃO DA MFDD

RELAÇÃO DAS FIGURAS

RELAÇÃO DAS TABELAS

ABREVIAÇÕES UTILIZADAS

AP	- PROCESSADOR ATÔMICO
AQ	- PILHA DE AGENDA
ASU	- UNIDADE DE ARMAZENAGEM ATÔMICA
CCT	- CENTRO DE CIÊNCIAS E TECNOLOGIA
CP	- CONTADOR DE PROGRAMA
CPER	- COMPUTADOR PARALELO ESTRUTURADO RECURSIVAMENTE
DC	- DEPARTAMENTO DE COMPUTAÇÃO
DOS	- DISK OPERATING SYSTEM
EP	- ELEMENTO DE PROCESSAMENTO
FDsim	- FLUXO DE DADOS SIMULADOR
FIFO	- FIRST IN FIRST OUT
FTG	- FAULT TOLERANT GATEWAY
GB	- GATEWAY BARRAMENTO
IQ	- PILHA DE ENTRADA
LSI	- LARGE SCALA INTEGRATION
MB	- MATRIZ BARRAMENTO
MCE	- MATRIZ CEPER
MEV	- MATRIZ DE EVENTOS
MFDD	- MÁQUINA A FLUXO DE DADOS DINÂMICA
MIMD	- MULTIPLE INSTRUCTION STREAM-MULTIPLE DATA STREAM
MIPS	- MILHÕES DE INSTRUÇÕES POR SEGUNDO
MISD	- MULTIPLE INSTRUCTION STREAM - SINGLE DATA STREAM

MIT	- MASSACHUSETTS INSTITUTE TECHNOLOGY
MP	- MEMÓRIA PRINCIPAL
MSI	- MEDIAN SCALA INTEGRATION
OQ	- PILHA DE SAÍDA
RAM	- RANDOM ACESS MEMORY
SIMD	- SINGLE INSTRUCTION STREAM - MULTIPLE DATA STREAM
SISD	- SINGLE INSTRUCTION STREAM - SINGLE DATA STREAM
TF	- TOLERÂNCIA A FALHAS
UFSCar	- UNIVERSIDADE FEDERAL DE SÃO CARLOS
UC	- UNIDADE DE CONTROLE
UCP	- UNIDADE CENTRAL DE PROCESSAMENTO
ULA	- UNIDADE LÓGICA E ARITMÉTICA
VLSI	- VERY LARGE SCALA INTEGRATION

C A P Í T U L O 1

1 - INTRODUÇÃO

Nos últimos anos temos acompanhado a crescente busca pelo desenvolvimento de arquiteturas paralelas e softwares distribuídos de tal forma a se operar cada vez mais com alta capacidade de armazenamento de informações e também grande velocidade de processamento dessas informações em aplicação tais como: previsão de tempo, exploração de petróleo, diagnóstico médico, inteligência artificial, sistemas especialistas, automação industrial e muitas outras aplicações científicas e de engenharia.

O alto desempenho dos sistemas computacionais depende não somente da utilização de dispositivos de hardware mais confiáveis e mais rápidos, como também da sofisticação da arquitetura dos sistemas computacionais e das técnicas de processamento.

As arquiteturas de alto desempenho atuais estão centradas em conceitos de processamento paralelo, que atualmente podem ser caracterizados em três classes estruturais [HWANG e BRIGGS ,1984]: computadores "pipeline", processadores em arranjo ("array processors") e sistemas multiprocessadores.

O desenvolvimento e aplicação desses sistemas requerem

um grande conhecimento das estruturas de hardware e software, e uma iteração muito forte entre algoritmos de computação paralela e a alocação ótima dos recursos da máquina.

Máquinas a fluxo de dados têm sido uma dessas frentes de pesquisa, por se tratar de sistemas cujo paralelismo está embutido na própria natureza do sistema, uma vez que o processo de execução de programas nestas máquinas é determinado pela disponibilidade dos dados, contrário às estruturas tradicionais onde o fluxo é determinado por um controle explícito [DENNIS, 1974], [DENNIS e MISUNAS, 1975].

Várias arquiteturas de máquina a fluxo de dados foram propostas na literatura [GURD et al, 1985], [ITO, et. al. 1986], [SHIMADA, et. al. 1986], [SATO, 1992]. Estas propostas de arquiteturas objetivaram a idéia de funcionamento e eficiência das máquinas, sem se preocupar com o aspecto da tolerância a falhas dessas máquinas.

Este trabalho tem por objetivo apresentar a Máquina a Fluxo de Dados Dinâmica (MFDD) com características de Tolerância a Falhas na programação a fluxo de dados [SILVA e KIRNER,1989a], [SILVA e KIRNER, 1989b], [SILVA e KIRNER, 1990], baseada na arquitetura do Computador Paralelo Estruturado Recursivamente (CPER) [KIRNER e MARQUES, 1985], [KIRNER e MARQUES, 1986], [KIRNER, 1989], em desenvolvimento no Departamento de Computação no Centro de Ciências e Tecnologia da Universidade Federal de São Carlos .

O CPER é um computador paralelo cuja estrutura é baseada em um barramento hierárquico, interligando Elementos de Processamento (EPs) entre si, a uma máquina hospedeira

responsável por todo o "front-end" do sistema. Esses EPs estão sendo propostos como uma estrutura que utiliza "transputer", memórias, e uma interface com o barramento. Em testes de laboratório foi possível comprovar a taxa de comunicação na ordem de 66 Mbytes/seg no barramento para dados de 32 bits, o que nos leva a uma taxa de 132 Mbytes/seg para dados de 64 bits, bastando aumentar as linhas no barramento, dispensando o uso de componentes de hardware exclusivo [GONÇALVES, 1991].

A Tolerância a Falhas no CPER ocorre através de alguns processos que garantem a nível de hardware o seu funcionamento. No barramento existem algumas linhas sobressalentes para eventual substituição de uma que falhe, em decorrência de um processo de tratamento de falha. Como toda comunicação ocorre através do barramento, os "links" do "transputer" em cada EP foram utilizados para constituírem um canal de comunicação sobressalente estruturado em forma de anel, que poderá vir a ser utilizado em decorrência de uma falha generalizada no barramento, mas que já seria utilizado para envio de algumas mensagens curtas de controle dispensando o uso do barramento. Existe também um processo de Tolerância a Falhas a nível de EP, detectado por um EP supervisor naquele barramento, que averigua constantemente se algum EP falhou, quando é enviado uma notificação para que a máquina hospedeira acione um processo de reconfiguração em função da falha.

A MFDD é uma máquina que executa programas a fluxo de dados dinâmica, sobre a estrutura hierárquica do CPER, com características de Tolerância a Falhas na execução dos programas a fluxo de dados. Na verdade a MFDD é um software de base que

torna o CPER uma máquina a fluxo de dados dinâmica.

Este software de base, a ser instalado em cada EP individualmente, deverá permitir a um EP executar uma ou várias operações a fluxo de dados (granularidade grossa ou fina respectivamente), como parte da execução de um programa a fluxo de dados distribuído entre os EPs do CPER. A este EP caberá gerenciar toda a recepção de dados, execução das operações a fluxo de dados a ele atribuidas e transmissão dos resultados, além de proceder um controle com relação à Tolerância a Falhas dentro da execução do programa a fluxo de dados.

A idéia geral da MFDD é que, a partir do Hospedeiro do CPER, um programa escrito, por exemplo, em PASCAL, seja convertido na forma fluxo de dados. A partir daí, um programa ANALISADOR/"LOADER" [D'AREZZO, 1991] faz a devida análise e carga das operações fluxo de dados na estrutura hierárquica do CPER de forma a encontrar a melhor distribuição, incluindo aqui questões relacionadas com granularidade fina ou grossa na distribuição do programa a fluxo de dados, dependendo da aplicação.

Uma vez distribuídas as operações, a MFDD inicia sua execução, recebendo os dados do programa através do Hospedeiro, processando esses dados em paralelo na forma fluxo de dados, e devolvendo os resultados ao Hospedeiro, garantindo ainda uma reconfiguração do sistema em decorrência de uma falha, aspecto este fundamental nesta estrutura.

Foi desenvolvido um simulador, o FDsim (Fluxo de Dados simulador), que permitiu validar todas as estruturas de programação a fluxo de dados tolerante a falhas na MFDD, e que

deverá ser utilizada como um laboratório para o desenvolvimento do software de base do CPER para que este opere como a MFDD.

Os resultados de desempenho da MFDD obtidos através do FDsim foram comparados com os obtidos pela análise teórica de filas.

A apresentação deste trabalho foi proposta da seguinte forma: No capítulo 2, descrevemos algumas características dos sistemas de alto desempenho com arquiteturas convencionais e não convencionais dirigidos por controle, e algumas características de software nestes sistemas. Ainda no capítulo 2 descrevemos os sistemas dirigidos por dados: a programação, e a organização de máquinas deste tipo. No capítulo 3 descrevemos as características básicas dos softwares para máquinas a fluxo de dados. No capítulo 4 descrevemos a arquitetura do CPER, os EPs, o barramento e as características de Tolerância a Falhas no hardware. Em seguida ainda no capítulo 4 descrevemos a estrutura dos programas a fluxo de dados para que o CPER opere como uma máquina a fluxo de dados dinâmica (MFDD). No capítulo 5 descrevemos os aspectos de Tolerância a Falhas nas arquiteturas a fluxo de dados, e apresentamos a proposta para a MFDD. No capítulo 6 descrevemos o simulador FDsim, suas estruturas e módulos básicos. Finalmente no capítulo 7 fazemos uma análise dos resultados obtidos através do simulador comparados aos resultados obtidos através da análise de filas.

C A P I T U L O 2

PROCESSAMENTO A FLUXO DE DADOS

2.1 - INTRODUÇÃO

As arquiteturas de computadores de alto desempenho citadas no capítulo anterior estão centradas em conceitos de processamento paralelo, que atualmente podem ser caracterizados em três classes estruturais: computadores "pipeline", processadores em arranjo e sistemas multiprocessadores, cuja característica é o processamento dirigido por controle.

Dentre as arquiteturas com alto desempenho estão também aquelas com estrutura a fluxo de dados, cujo processamento é dirigido pelos dados [HWANG e BRIGGS, 1984].

Em seguida faremos uma exposição das características básicas destes dois estilos de processamento.

2.2 - PROCESSAMENTO DIRIGIDO POR CONTROLE

Podemos definir processamento paralelo como uma forma eficiente de processar a informação com ênfase à exploração de eventos concorrentes no processo de computação. Concorrência

implica no paralelismo, simultaneidade e "pipeline". Eventos paralelos podem ocorrer em múltiplos recursos durante o mesmo intervalo de tempo; eventos simultâneos podem ocorrer no mesmo instante de tempo; e "pipeline" pode ocorrer através da sobreposição na extensão do tempo. Estes eventos concorrentes podem estar presentes num sistema computacional em vários níveis de processamento.

Podemos destacar como o nível mais alto de processamento paralelo, aquele obtido entre múltiplos "JOBs" ou PROGRAMAS através de multiprogramação, "time-sharing" e multiprocessamento. Este nível requer o desenvolvimento de algoritmos processáveis de forma paralela. A implementação destes algoritmos dependem da alocação eficiente dos recursos limitados de hardware/software para múltiplos programas.

Em um nível inferior pode-se obter o processamento paralelo entre processos ou tarefas (segmento de programas) dentro de um mesmo programa. Pode-se destacar aqui a utilização de técnicas para programação concorrente [BEN-ARI,1982], [KIRNER e MENDES, 1988] que leva em conta aspectos como: exclusão mútua entre recursos de um sistema computacional; confiabilidade nos programas; uso de métodos como semáforos, monitores, "rendez-vouz", para garantir a exclusão mútua, além de comunicação inter-tarefas.

Um terceiro nível é o que explora concorrência entre múltiplas instruções. Essa concorrência, ou mais especificamente o paralelismo entre as instruções, geralmente é determinada pela dependência dos dados. Isso pode ser observado quando da operação entre vetores, tal que: uma soma por exemplo, pode ser efetuada

simultaneamente entre os elementos do vetor enquanto que em um sistema não paralelo essa mesma soma deveria ser efetuada passo a passo dentro de um "LOOP" tipo "FOR".

Finalmente podemos ter operações concorrentes dentro de cada instrução. Isso pode ser obtido através do "pipeline" das instruções tal que partes destas instruções podem ser executadas simultaneamente, durante um ciclo de máquina.

2.2.1 - ESTRUTURA DOS COMPUTADORES PARALELOS

Em geral, computadores digitais podem ser classificados em quatro categorias, de acordo com a multiplicidade de instruções e dados. Este esquema de classificação foi introduzido por Michael J. Flynn [FLYNN, 1972], ou seja:

- "Single Instruction stream - Single Data stream" (SISD)
- "Single Instruction stream - Multiple Data stream" (SIMD)
- "Multiple Instruction stream - Single Data stream" (MISD)
- "Multiple Instruction stream -Multiple Data stream" (MIMD)

a) Arquiteturas SISD

Conforme Fig.2.1a, vemos a configuração de grande parte dos computadores atuais. Instruções são executadas sequencialmente mas não sobrepostas em seus estágios de execução.

b) Arquiteturas SIMD

Conforme a Fig.2.1b, temos a configuração básica de processadores em arranjo, existindo múltiplos Elementos de Processamento, supervisionados por uma única unidade de controle.

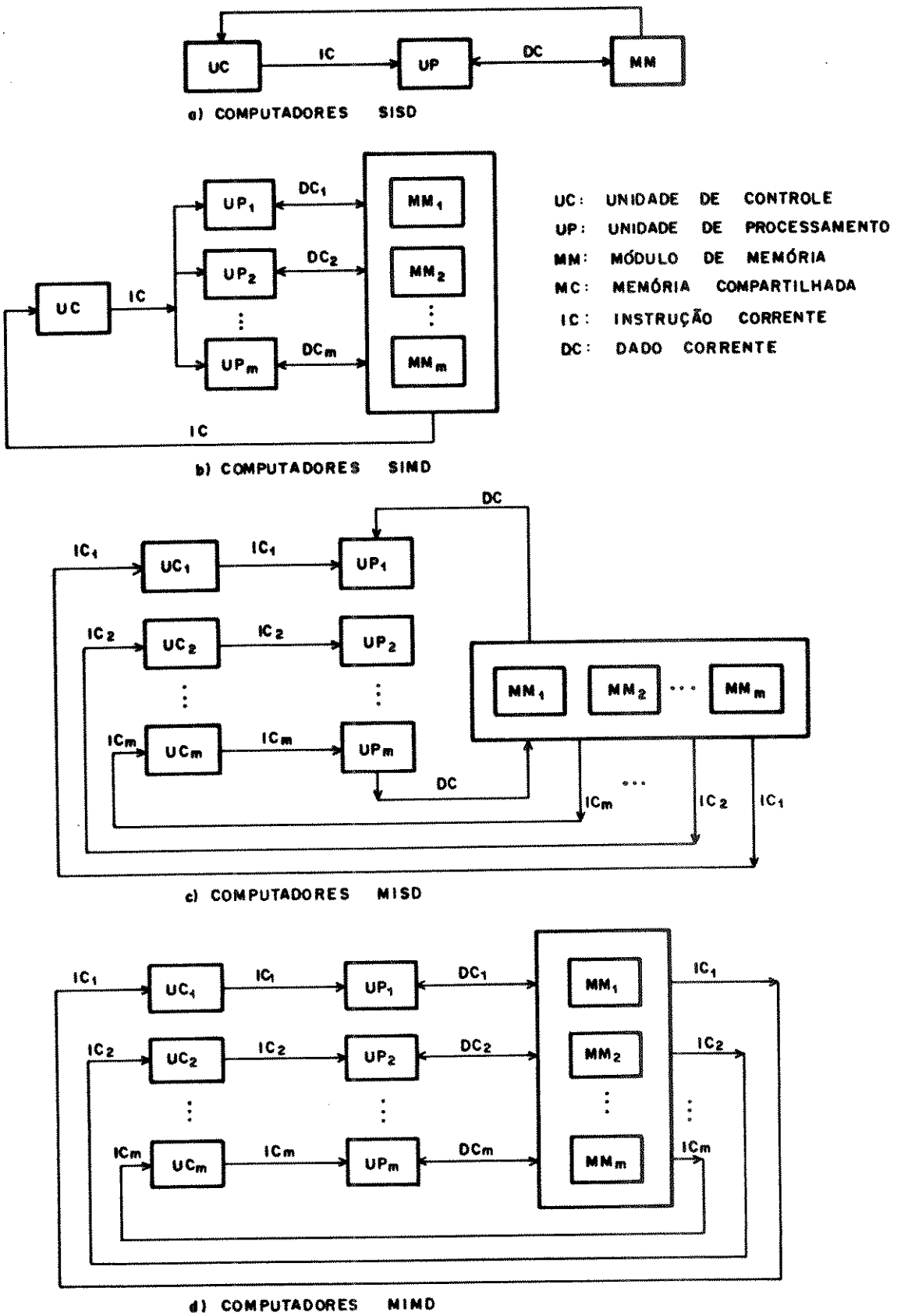


FIGURA 2.1 - CLASSIFICAÇÃO DE FLYNN DAS VÁRIAS ORGANIZAÇÕES DE COMPUTADORES.

c) Arquiteturas MISD

Conforme Fig.2.1c, existem N processadores, cada um recebendo instruções distintas, operando sobre um mesmo conjunto de dados. O resultado de um processador é entrada (operando) para o próximo processador.

d) Arquiteturas MIMD

Conforme Fig.2.1d, um sistema MIMD implica na iteração entre os N processadores, pois toda a memória é derivada de um mesmo espaço de dados compartilhado entre todos os processadores. Pode-se ainda dizer que um sistema MIMD é fortemente acoplado se o grau de iteração entre os processadores é alto, caso contrário ele é fracamente acoplado.

Redes de Computadores são sistemas bastante conhecidos, que podem ser considerados como estruturas do tipo MIMD fracamente acoplado, mas com características próprias, principalmente no que se refere ao software.

2.2.2 - Software nestes Sistemas

Sabemos que um sistema operacional é um conjunto de programas que controlam os recursos de um computador; recursos esses que vão desde o hardware puro como processador, memória principal e secundária, canais de Entrada e Saída, controladores, periférico, etc; até o gerenciamento de Memória Virtual, processos paralelos e outros, oferecendo aos seus usuários uma máquina virtual com mais facilidade de operação do que a máquina real propriamente dita. Os exemplos mais conhecidos de sistemas operacionais são o "DOS" e "UNIX" (conhecidos como sistemas operacionais centralizados) [CLAUDIO e MENDES, 1988].

Um sistema operacional distribuído também olha para seus usuários da mesma forma que o sistema operacional centralizado, porém ele roda sobre várias UCPs (Unidade Central de Processamento) independentes, significando que o controle dos recursos do sistema agora está distribuído entre as várias UCPs que formam o sistema.

Em vários "workshops" ocorridos no final da década de 70 ficou claro que há muitos elementos de um sistema que podem ser distribuídos, tais como: processadores (lógica de processamento); dados; programas; e controle (sistema operacional). Estabeleceu-se no entanto que a principal característica que distingue os sistemas de processamento distribuído dos sistemas de arquiteturas clássicas, está na descentralização do controle.

Sendo assim, podemos utilizar sistemas operacionais

centralizados em todas as estruturas definidas por Michael J. Flynn, sem necessariamente estarmos falando de um Sistema Distribuído, ou podemos distribuir o controle principalmente nas máquinas tipo MISD e MIMD, de tal forma a obter uma máquina cujo sistema operacional é distribuído.

Para fazer um contraste com sistemas operacionais distribuídos, vamos considerar um outro tipo de sistema, que chamamos sistema operacional de redes [TANENBAUM, 1981].

Na maioria dos casos, tais sistemas possuem as características abaixo, diferenciando-o dos sistemas distribuídos:

a) Cada computador possui seu próprio sistema operacional, ao invés de rodar parte de um sistema operacional global.
b) Cada usuário normalmente trabalha sobre sua própria máquina, usando algum tipo de acesso remoto para outras máquinas, ao invés de existir uma alocação dinâmica, PROCESSO - UCP, elaborada por um sistema operacional.
c) Os usuários estão cientes de onde estão armazenados seus arquivos, e que devem usar comandos explícitos de transferência de arquivo para movê-los entre as máquinas, ao invés do sistema operacional gerenciar todo posicionamento dos arquivos.
d) O sistema tem pouca ou nenhuma tolerância a falha, isto é, se 1% dos computadores param, 1% dos usuários não estarão operando, ao invés de cada um poder continuar seus trabalhos, embora com 1% menos no desempenho do sistema.

O grande interesse em sistemas distribuídos se deve principalmente a evolução tecnológica na área de microprocessadores, que os tem tornado cada vez mais poderosos e mais baratos, se comparado aos grandes "mainframes" e minicomputadores existentes.

Isso nos leva a pensar no projeto de grandes sistemas compostos por pequenos processadores, certamente com melhor relação custo/desempenho em relação a sistemas tradicionais, além da relativa simplicidade do projeto de seu software de controle, uma vez que este será distribuído entre os vários processadores cada um com uma função dedicada.

Uma outra consideração importante a ser analisada, no caso de sistemas distribuídos é o crescimento incremental, tal que: quando se deseja aumentar a capacidade computacional do sistema, basta conectar a ele mais processadores. A arquitetura do sistema é crucial para este tipo de crescimento, uma vez que seria difícil obtê-lo em sistemas com arquitetura convencional.

Outra grande vantagem que pode ser obtida deste crescimento incremental é a disponibilidade e confiabilidade no sistema, sendo que algumas partes deste poderão ser desativadas sem perturbar o restante do sistema. Por outro lado, o "overhead" gerado para se obter estas vantagens, pode se tornar um grande problema. Em um sistema centralizado no entanto, não se pode esperar que o computador funcione normalmente se metade da memória está com problema, enquanto que em um sistema distribuído há de se prever um alto grau de tolerância a falha.

Finalmente, o problema fundamental em sistemas

distribuído, é a grande quantidade de informações que compõem o estado global do sistema, não sendo conveniente tratá-las como um todo em uma tabela por exemplo. Por outro lado, dividir essas informações implica na possibilidade de acessar algumas que não são as mais importantes para o momento.

Nesta divisão de informações, e em se tratando de um sistema distribuído, podemos ainda considerar um aspecto importante nesses sistemas que é a granularidade das funções ou sub-tarefas a serem realizadas em paralelo. A granularidade pode ser dividida em fina, média e grossa [KRVATRACHUE e LEWIS, 1988]. Nos modelos que exploram a granularidade grossa, o paralelismo reside na execução de módulos lógicos ou de sub-rotinas de um programa. Quando partes de um mesmo programa são executados paralelamente, tem-se granularidade média. A granularidade fina está no extremo onde instruções são realizadas em paralelo.

2.3 - PROCESSAMENTO DIRIGIDO POR DADOS

A grande maioria dos sistemas tradicionais foram implementados baseados nos princípios básicos de computação von Neumann [TRELEVEAN et al, 1982], que incluem:

1) Um simples elemento de computação incorporando processador, comunicação e memória.
2) Organização linear de células de memória com tamanho fixo.
3) Espaço de endereçamento das células em um único nível.
4) Linguagem de máquina de baixo nível (instruções que executam operações simples sobre operandos elementares).
5) Controle de computação centralizada e sequencial.

Na seção 2.2.1 vimos algumas organizações de computadores, dentre as quais podemos destacar aquelas com uma característica específica que é o processamento paralelo (SIMD, MISD, e MIMD), que possibilitam um aumento da velocidade de processamento, com algum ou nenhum controle distribuído, porém mantendo alguma dependência dos princípios de von Neumann.

Na busca deste paralelismo, vêm sendo desenvolvidos também algumas máquinas cujas organizações são "naturalmente" paralelas, [DENNIS, 1974], [DENNIS e MISUNAS,1975], [DENNIS,1980] [GURD e KIRKHAM, 1985], [ITO et al, 1986], [SHIMADA et al, 1986], [SATO, 1992], motivadas por um trabalho pioneiro sobre Máquinas a Fluxo de Dados , publicado por Jack Dennis [DENNIS, 1974], onde operandos disponíveis disparam a execução de uma operação sobre

um computador, conotação esta que não segue os princípios de von Neumann.

Pode-se destacar nas pesquisas sobre essas máquinas três áreas de grande interesse:

1) O desejo de utilizar concorrência para aumentar o desempenho do computador.
2) O desejo em explorar a integração em escala muito alta (VLSI) no projeto de computadores.
3) O grande interesse em uma nova classe de linguagens de programação a nível muito alto.

Passaremos então a analisar as propostas de arquiteturas a fluxo de dados através de suas características básicas de hardware e software.

2.3.1 - Programação a Fluxo de Dados

Como citado anteriormente, Jack Dennis foi o pioneiro nas pesquisas de computadores a fluxo de dados. A idéia era a implementação de um sistema de computador altamente paralelo, que executasse simultaneamente vários fragmentos de um programa, porém este alto grau de atividades concorrentes deveria ser encontrado sem nenhum sacrifício [DENNIS e MISUNAS, 1975]. Assim, a criação de uma linguagem básica contribuiu significativamente na construção de programas corretos com as características acima, servindo como guia para as pesquisas em arquiteturas de computadores avançados.

A linguagem utiliza grafos dirigidos. Nesta

representação, a execução de um teste ou uma operação é habilitada pela disponibilidade dos valores requisitados. O término de execução de uma operação ou um teste liberam valores ou decisões para os elementos do programa cuja execução dependem deles.

Em computadores convencionais, a análise dos programas geralmente é feita para ganhar tempo de compilação, melhorar a utilização de recursos e otimização dos códigos , e diminuir o tempo de execução através de atividades lógicas e aritméticas concorrentes, tudo para aumentar a vazão do sistema.

Vamos analisar o seguinte programa em PASCAL:

1. P:=X+Y deve esperar por X e Y
2. Q:=P/Y deve esperar a instrução 1 ser completada
3. R:=X*P deve esperar a instrução 1 ser completada
4. S:=R-Q deve esperar as instruções 2 e 3 serem completadas
5. T:=R*P deve esperar a instrução 3 ser completada
6. V:=S/T deve esperar as instruções 4 e 5 serem completadas

Em um computador serial poderíamos ter as seguintes sequências de computação possíveis:

(1,2,3,4,5,6) (1,3,2,5,4,6)

(1,3,5,2,4,6) (1,2,3,5,4,6)

(1,3,2,4,5,6)

Um computador paralelo poderia executar essas seis operações em quatro passos ao invés de seis, na forma:

(1, [2 e 3 simultaneamente],[4 e 5 simultaneamente],6)

O programa acima pode ser representado por um grafo dirigido conforme descrito na Fig.2.2.

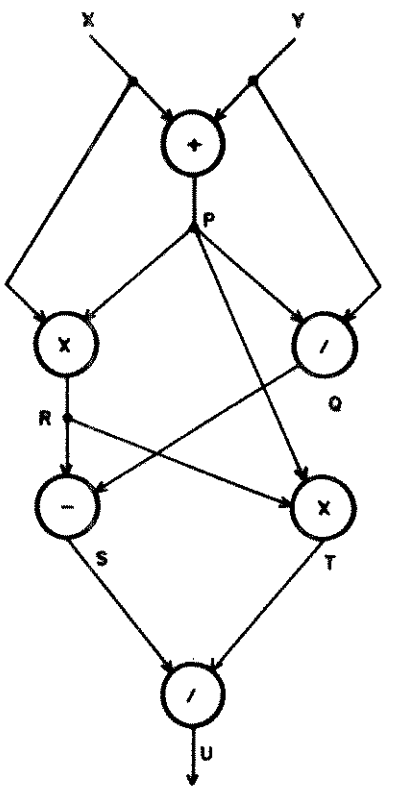


FIGURA 2.2 - UM GRAFO A FLUXO DE DADOS PARA A COMPUTAÇÃO DE U=f(x,y)=(x*(X+Y)-(X+Y)/y)/(X*(X+Y)*(X+Y)).

Essa representação é utilizada para representar programas a fluxo de dados cujos nós correspondem a operadores e os arcos são ponteiros para o envio de pacotes de dados [TRELEVEAN et al, 1982], [DENNIS, 1974], [HWANG e BRIGGS, 1984].

Na Fig.2.3 são descritos dois tipos de enlace sobre um grafo a fluxo de dados, um que transmite números inteiros, reais, ou complexos (LINK DE DADOS), e um que possui somente valores "booleanos", que serão utilizados como controle (LINK LÓGICO).

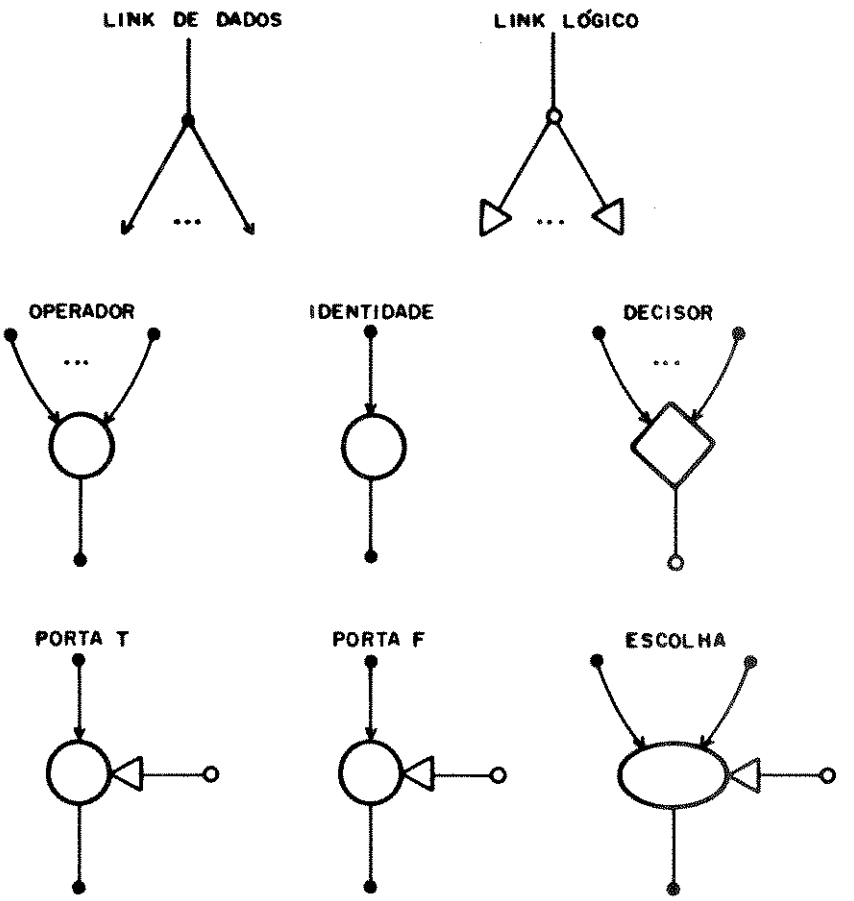


FIGURA 2.3 - OPERADORES (NÓS) E ENLACE (ARCO) PARA CONSTRUÇÃO DE GRAFOS A FLUXO DE DADOS.

Um operador IDENTIDADE é um operador especial que tem um arco de entrada e seu valor de entrada é transmitido sem mudanças. Operadores DECISOR, PORTA, e ESCOLHA são usados para representarem computação iterativa ou condicionais no grafo a fluxo de dados. Um DECISOR requer um valor para cada arco de entrada e produz um valor resultante da aplicação de uma operação lógica sobre os valores recebidos. Um OPERADOR requer também um valor para cada arco de entrada e produz um valor resultante da aplicação de uma operação aritmética sobre os valores recebidos.

Pacotes de controle ("tokens") representam valores "booleanos" que controlam o fluxo dos pacotes de dados através das portas T,F e operadores ESCOLHA. Uma porta T passa um pacote de dado de seu arco de entrada para seu arco de saída quando ele recebe um valor "TRUE" sobre seu arco de entrada de controle. Ele absorverá o pacote de dado do seu arco de entrada e não colocará nada no seu arco de saída, se ele receber um valor "FALSE". Uma porta F tem comportamento similar, exceto que o valor do controle é reverso. Um operador ESCOLHA tem arcos de entrada de dados T e F e um arco de controle. Quando um valor "TRUE" é recebido no arco de controle, o operador ESCOLHA posiciona o pacote de dado do arco de entrada T sobre seu arco de saída. O pacote de informação sobre o arco de entrada não usado é descartado. Da mesma forma, se a entrada no arco de controle é "FALSE", o operador ESCOLHA inverte o processo e o dado na entrada F é passado para a saída.

Como dito acima, a regra de disparo para execução das instruções é baseada na disponibilidade dos dados e uma vez

disparado, os pacotes de dados e de controle utilizados como entrada das operações, são descartados e pacotes de dados ou controle podem ser gerados como resultados das operações.

Através de um outro exemplo, vamos representar um programa iterativo, conforme descrito na Fig.2.4, usando alguns dos operadores e enlaces especificados na Fig.2.3.

O programa calcula $Z = x^n$ para x e n definidos pelo usuário:

```
input x,n
   y=1; i=n
    while i>0 do
     begin y=y*x ; i=i-1 end
   z=y
output z
```

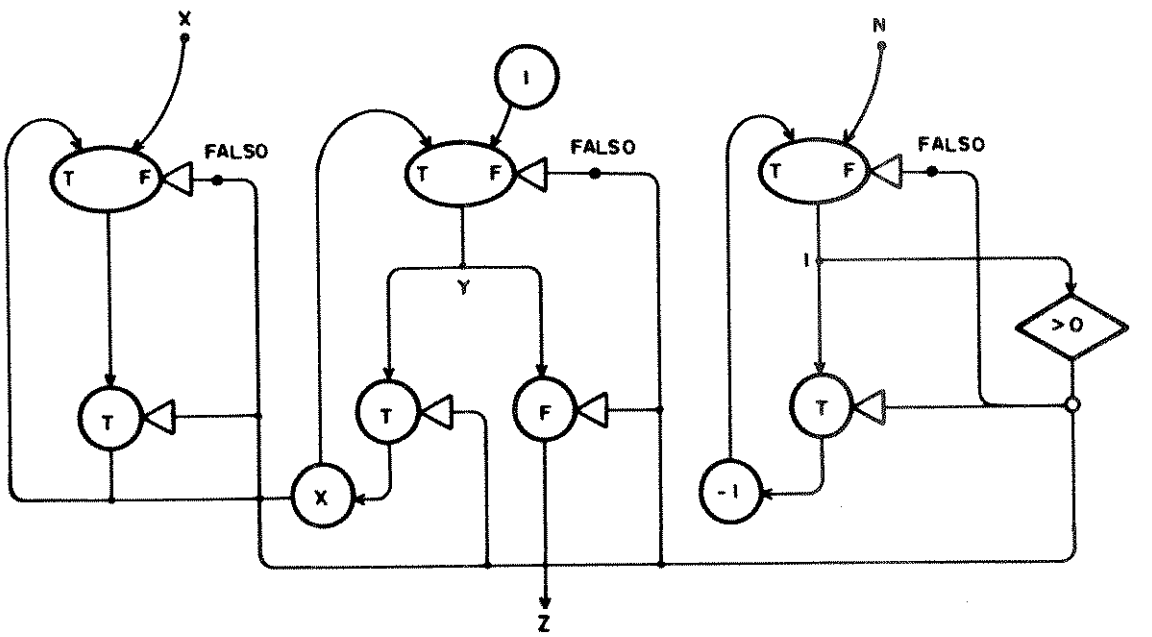


FIGURA 2.4 - REPRESENTAÇÃO DE UM GRAFO A FLUXO DE DADOS SOBRE A COMPUTAÇÃO $Z = x^n$

Os valores sucessivos assumidos pelas variáveis y e i passam através dos enlaces rotulados no grafo do programa. O DECISOR emite um pacote que leva o valor "TRUE" cada vez que for requerido a execução do corpo do "loop". Quando o disparo do DECISOR produz um "FALSE", o valor de y é direcionado para o enlace de saída z.

Na Fig.2.4 os pacotes "FALSE" nos operadores ESCOLHA servem para iniciar a execução dos "loops".

Apesar do alto grau de paralelismo encontrado na execução deste exemplo simples de programação iterativa, pode-se observar um dos maiores problemas associados com grafos cíclicos [DENNIS, 1974], [HWANG e BRIGGS, 1984]. No exemplo acima, se a multiplicação de x por y demorar mais tempo para se efetivar que os outros operadores, possivelmente vários pacotes de controle seriam gerados pelo DECISOR antes que uma multiplicação estivesse concluída. Para que este problema seja resolvido é necessário colocar em "buffers" os pacotes que estão sendo gerados sobre os enlaces e à medida que os dados vão sendo gerados, os pacotes nos "buffers" vão sendo também consumidos.

A utilização ou não de "buffers" é que caracterizam máquina a fluxo de dados dinâmica ou estática respectivamente.

Para as máquinas estáticas não existem os "buffers", o que significa que em cada enlace só pode existir um pacote, impedindo o fluxo natural dos dados pelo programa.

Nas máquinas dinâmicas, os "buffers" permitem que os dados fluam naturalmente, o que significa utilizar toda a potencialidade da programação a fluxo de dados, exigindo no entanto um tratamento mais complexo na implementação do sistema

pois é preciso além de armazenar os dados, identificar os parceiros dos dados que disparam uma operação.

2.3.2 - Organizaçao de Máquinas tipo Fluxo de Dados

Conforme proposto por Dennis [DENNIS e MISUNAS, 1975], foi projetado um processador elementar para utilizar a Linguagem Básica. Nesta linguagem, um operador está habilitado quando todos os pacotes estão presentes sobre os arcos de entrada. O operador habilitado pode disparar a qualquer instante, removendo os pacotes de seus arcos de entrada, identificando os valores dos operandos associados com esses pacotes, e executando uma operação sobre estes valores, cujos resultados são posicionados sobre seus arcos de saída. Um resultado pode ser enviado para mais que um destino. Este esquema originou o projeto de um processador organizado como na Fig. 2.5.

A estrutura da Fig.2.5 descreve quatro blocos funcionais que se comunicam através de mecanismo de comunicação por pacotes.

Um programa a fluxo de dados a ser executado nesta máquina é armazenado na Memória do processador. A Memória é organizada em células de instrução. Cada célula corresponde a um operador do programa a fluxo de dados.

Quando uma célula contém uma instrução e os operandos necessários, ela está habilitada e então sinaliza a Rede de Arbitração que ela está pronta para transmitir seu conteúdo como um pacote de operação para uma Unidade de Operação que pode então executar a função desejada. O pacote de operação flui através da

Rede de Arbitração que o direciona para uma Unidade de Operação apropriada, decodificando um campo da instrução no pacote para esta identificaçao.

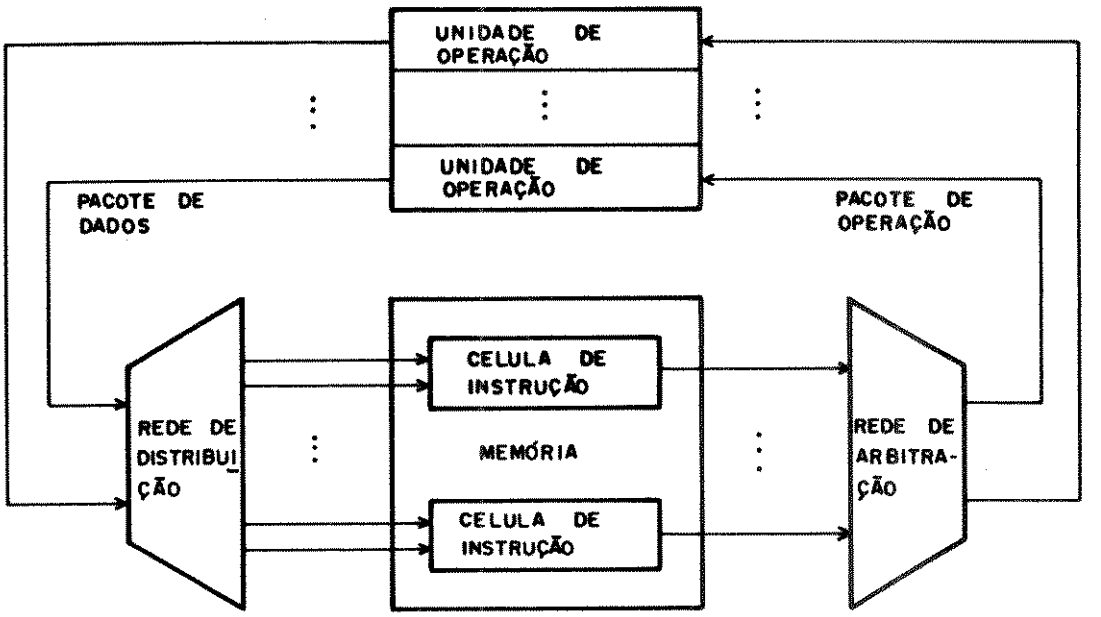


FIGURA 2.5 - ORGANIZAÇÃO DE UM PROCESSADOR A FLUXO DE DADOS ELEMENTAR.

Como resultado de uma operação, a Unidade de Operação libera um ou mais pacotes de dados, consistindo dos valores computados e o endereço dos registros na Memória para os quais os valores serão enviados. A Rede de Distribuição aceita pacotes de

dados das Unidade de Operação e utiliza os endereços de cada um para direcionar os dados através da rede para o registro correto na Memória. A Célula de instrução que contém aquele registro pode então ser habilitada se uma instrução e todos os operandos estão presentes na célula.

As Células de instruções são manuseadas como um conjunto de gabaritos, cada um correspondendo a um nó do programa a fluxo de dados, conforme descrito na Fig. 2.6, onde é mostrado um gabarito correspondendo ao operador soma.

Existem quatro campos no gabarito da Fig.2.6: um para o código de operação especificando a operação a ser executada; dois receptores, que esperam para ser preenchidos com valores de operandos; e campo destino (no caso um), que especifica o que fazer com o resultado da operação sobre os operandos.

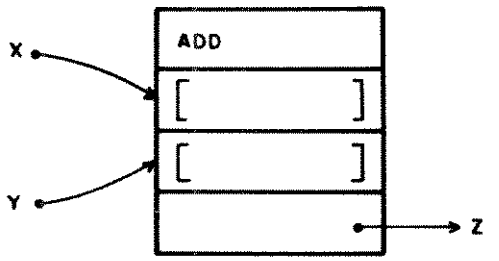


FIGURA 2.6 - UM GABARITO DA OPERAÇÃO SOMA.

Na Fig.2.7 vemos como juntar gabaritos para representar um programa.

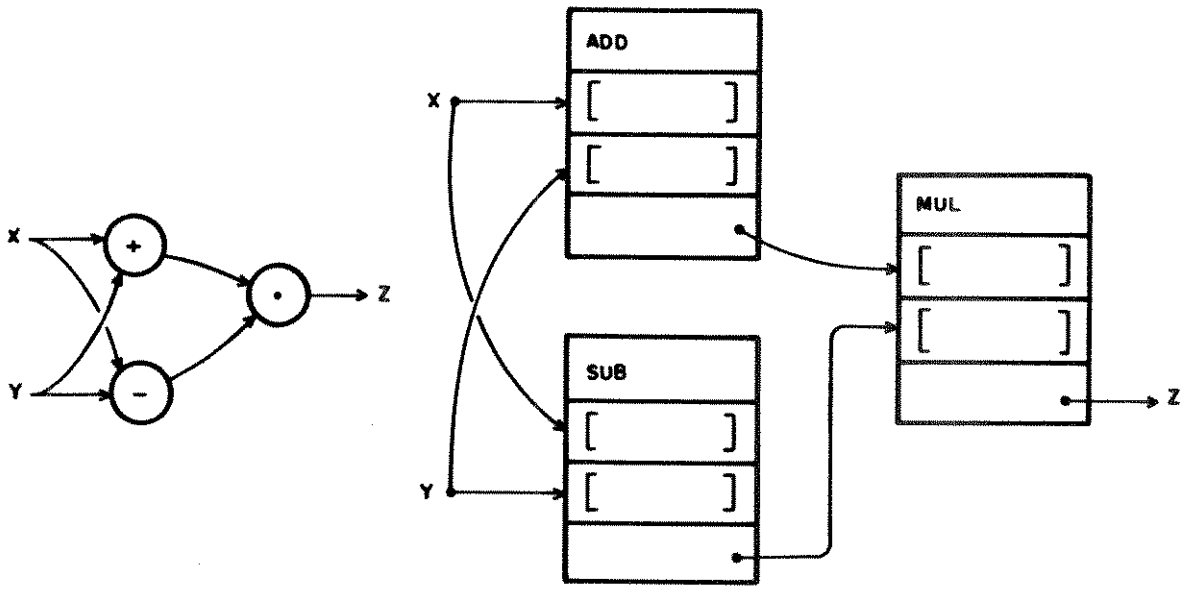


FIGURA 2.7 - EXEMPLO DE UM PROGRAMA REPRESENTADO ATRAVÉS DE GABARITOS.

2.3.2.1 - Descrição das Arquiteturas Existentes

Um grande número de projetos de arquiteturas a fluxo de dados vem sendo desenvolvidos e construídos ao longo dos anos. No entanto um dos primeiros projetos de máquina a fluxo de dados sem dúvida começou com o "Massachusetts Institute of Technology Architecture" - MIT que vem servindo como base para muitos outros projetos, que passamos a descrever.

2.3.2.1.1 - O Computador a Fluxo de Dados do MIT

O computador a fluxo de dados do MIT é o descrito na seção anterior, com a observação que somente um pacote pode ocupar um arco a cada instante. Isto implica que a regra de disparo é tal que uma instrução é habilitada se um pacote de dados está presente sobre cada um de seus arcos de entrada e nenhum pacote está presente sobre qualquer de seus arcos de saída. Na organização de programas da MIT existem pacotes de controle e pacotes de dados, que contribuem para a habilitação de uma instrução. Esses pacotes de controle atuam como sinal de reconhecimento quando os pacotes de dados são removidos dos arcos de saída. Uma descrição mais detalhada da organização de uma máquina MIT é mostrada na Fig.2.8.

Ela consiste de cinco partes conectadas por canais através dos quais são enviados pacotes de informação usando protocolo de transmissão assíncrono. As partes são:

a) Seção de Memória - consistindo de Células de Instrução que contém instruções e seus operandos.
b) Seção de Processamento - consistindo de elementos processadores especiais, que realizam operações sobre pacotes de dados.
c) Rede de Arbitragem - distribui pacotes de instruções executáveis da Seção de Memória para a Seção de Processamento.
d) Rede de Controle - distribui pacotes de controle da Seção

de processamento para a Seção de Memória.

e) Rede de Distribuição - distribui pacotes de dados da Seção de Processamento para a Seção de Memória.

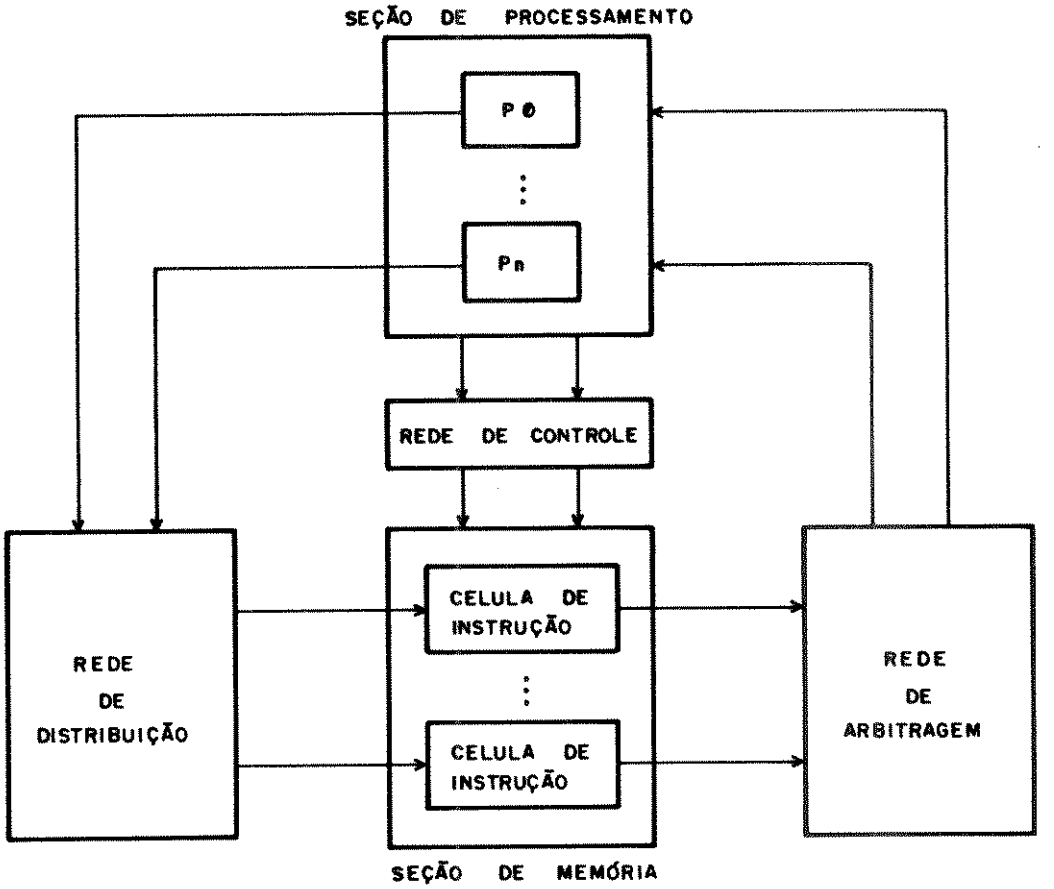


FIGURA 2.8 - COMPUTADOR A FLUXO DE DADOS DO MIT.

2.3.2.1.2 - Processador de Dados Distribuídos da Texas Instruments-DDP

O Processador de Dados Distribuídos (DDP) é um sistema projetado pela "Texas Instruments" para investigar o potencial do processamento a fluxo de dados como base para computadores de alto desempenho. A organização de uma máquina DDP é dada na

Fig.2.9.

Conceitualmente, o computador DDP, como o do MIT, é baseado sobre uma organização similar de programas. Uma instrução é habilitada se um pacote de dado está presente sobre cada arco de entrada e nenhum pacote está presente sobre qualquer arco de saída. Da mesma forma, pacotes de controle são usados como sinais de reconhecimento.

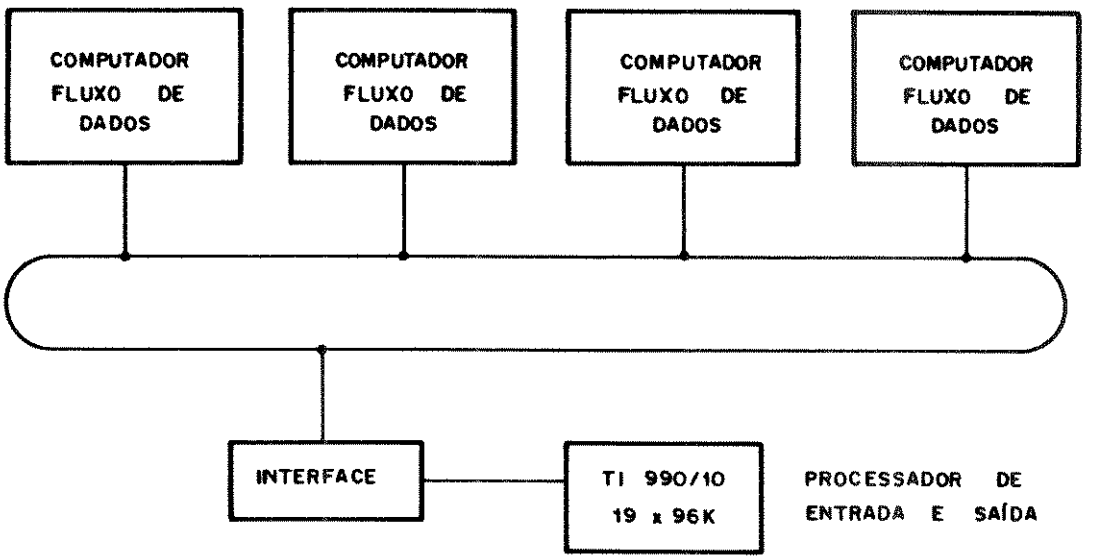


FIGURA 2.9 - ORGANIZAÇÃO DA MÁQUINA DDP DA "TEXAS INSTRUMENTS".

A máquina DDP consiste de cinco elementos computacionais independentes: quatro computadores a fluxo de dados idênticos que cooperam na execução de uma computação e um minicomputador "Texas Instruments" 990/10, atuando como um processador "front-end" para entrada/saída, providenciando suporte de sistemas operacionais, e gerenciando o desempenho do

sistema. Os cinco elementos computacionais no DDP são conectados por um registrador de deslocamento circular com palavras de tamanho grande e de comprimento variável.

Cada computador a fluxo de dados consiste de quatro unidades principais. Estas unidades são:

a) Unidade Aritmética - que processa instruções executáveis e gera pacotes de saída.
b) Memória de Programa - construída com memórias de acesso aleatório (RAM) e que mantém instruções a fluxo de dados.
c) Controle Atualizador - que atualiza instruções com pacotes de dados.
d) Pilha de Instruções Pendentes - que mantém as instruções executáveis que foram habilitadas. Instruções executáveis são removidas desta fila pela Unidade Aritmética e processadas.

Quando termina a execução de uma instrução, uma série de pacotes de dados são enviados para o controlador atualizador. Usando o endereço do pacote, o Controlador Atualizador armazena o operando na instrução e decrementa de seu contador de predecessores. Se esta contagem é zero, a instrução está pronta para executar; uma cópia é posicionada na fila de instruções pendentes e o processo é reiniciado.

2.3.2.1.3 - A Máquina Dirigida por Dados de "UTAH"

A Máquina Dirigida por Dados 1 (DDM1) é um elemento computacional com uma arquitetura a fluxo de dados estruturada

recursivamente, implementada na Universidade de "UTAH" e suportada pela "Burroughs Corporation". A estrutura do elemento computacional DDM1 é mostrado na Fig.2.10.

Tanto o programa como a máquina são organizados utilizando os conceitos de recursão. O computador é composto de elementos computacionais estruturados de forma hierárquica (pares processador-memória) onde cada elemento é lógicamente recursivo e consiste de elementos mais inferiores. Físicamente a arquitetura do computador é estruturada em árvore, com cada elemento computacional sendo conectado a um elemento superior acima e oito elementos inferiores que ele supervisiona.

Na organização de programa a fluxo de dados de "UTAH", não existem pacotes de controle, os pacotes de dados providenciam toda comunicação entre as instruções. Nesta organização, os arcos do grafo dirigido dos programas a fluxo de dados, são vistos como fila "first-in first-out"(FIFO), um modelo que é suportado pela arquitetura, onde os "tokens" de dados são armazenados à medida que vão chegando mas não são imediatamente consumidos pelas instruções. Por isso cada instrução consiste de um código de operação e uma lista de endereços distintos para os resultados, junto com um número variável de conjuntos de pacotes de dados esperando para ser completado ou para ser consumido pela instrução.

A máquina consiste de 6 grandes unidades:

a) Unidade de Armazenagem Atômica (ASU) - que armazena o programa.

b) Processador Atômico (AP) - que executa as instruções.

c) Pilha de Agenda (AQ) - que armazena mensagens para a

unidade de Armazenagem Atômica local.

d) Pilha de Entrada (IQ) - que armazena mensagens de um elemento computacional superior.

e) Pilha de Saída (OQ) - que armazena mensagens para um elemento computacional superior.

f) Comutador - que conecta o elemento computacional com os oito elementos inferiores.

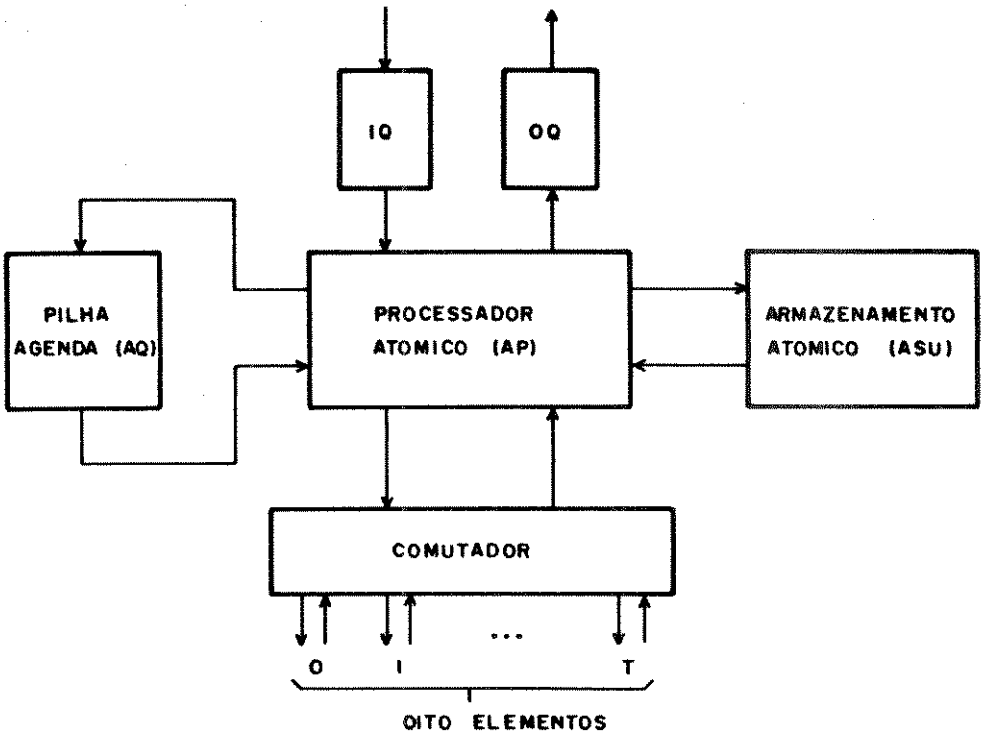


FIGURA 2.10 - MÁQUINA DIRIGIDA POR DADOS DE "UTAH" (DDM1).

2.3.2.1.4 - A Máquina a Fluxo de Dados de "Irvine"

Projeto originado na Universidade da Califórnia "Irvine" e continuou no MIT, explora o potencial de VLSI providenciando uma organização de programa altamente concorrente.

A Máquina "Irvine" é composta por N elementos interligados por uma rede de comunicação N * N. A Fig.2.11 ilustra um Elemento de Processamento da máquina a fluxo de dados de "Irvine".

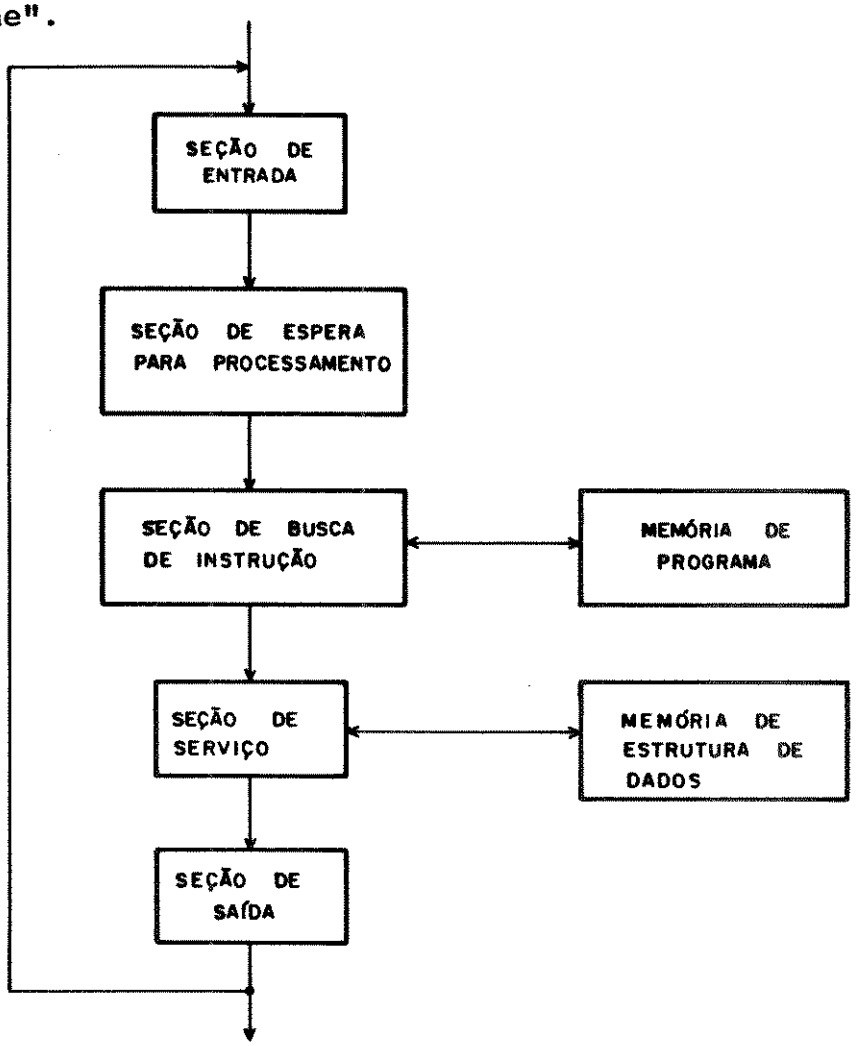


FIGURA 2.11 - ELEMENTO DE PROCESSAMENTO A FLUXO DE DADOS DE "IRVINE".

Cada Elemento de Processamento é essencialmente um computador completo com um conjunto de instruções, com 16k palavras cada uma das memórias de programa e de estrutura de dados, e certos elementos especiais. Estes elementos especiais incluem:

a) Seção de Entrada - que aceita entradas de outros Elementos de Processamento.
b) Seção de espera para Processamento - que forma pacotes de dados para uma instrução consumidora.
c) Seção de Busca de Instruções - que busca instruções da memória de programa.
d) Seção de Serviço - que é uma unidade lógica aritmética de ponto-flutuante.
e) Seção de Saída - que rotula pacotes de dados contendo resultados para os Elementos de Processamento destino.

2.3.2.1.5 - O Computador a Fluxo de Dados de "Manchester"

A organização da máquina é mostrada na Fig.2.12. Ela consiste de cinco unidades principais:

a) Comutador - providencia entrada e saída para o sistema.
b) Pilha Pacote de dado - realiza armazenamento temporário de um pacote de dado.
c) Unidade de Disparo - que iguala par de pacotes de dados.
d) Memória de Instruções - memória que mantém os programas a fluxo de dados.
e) Unidade de Processamento - consistindo de um número de Elementos de Processamento idênticos, que executam as instruções.

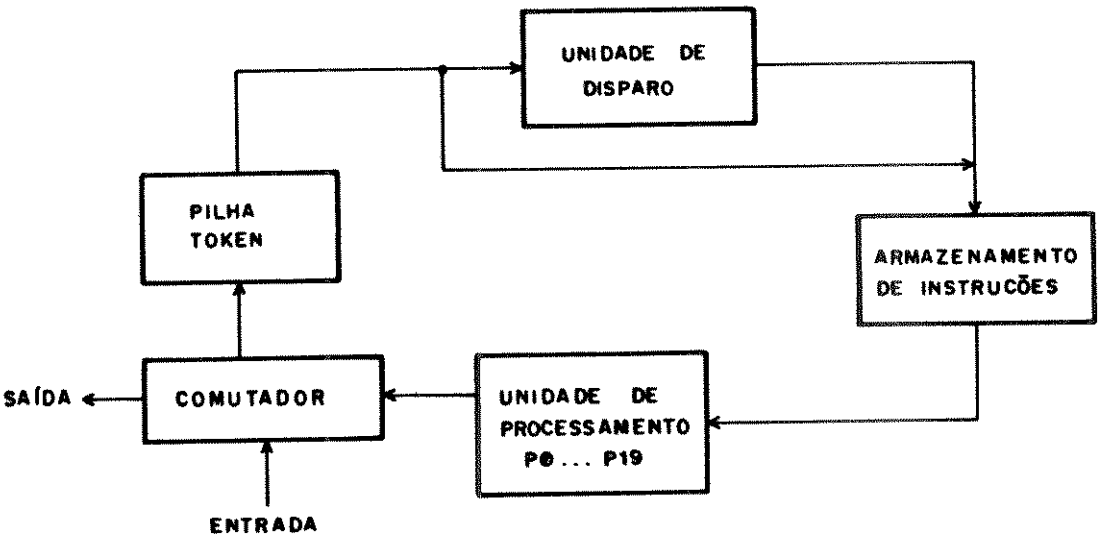


FIGURA 2.12 - COMPUTADOR A FLUXO DE DADOS DE "MANCHESTER".

O comutador é usado para passar pacotes de dados para dentro ou para fora do computador, ou comunicar com periféricos ou outros computadores. Para iniciar a execução de um fragmento de programa, os pacotes de iniciação são inseridos através de comutadores e dirigidos por seus rótulos para as instruções de início da computação. No final do programa são inseridos instruções especiais que geram pacotes de dados de saída.

2.3.2.1.6 - Sistema "LAU de Toulouse"

"Language à Assignation Unique" é a tradução francesa para a frase Linguagem de Atribuição Simples. O projeto LAU foi implementado no laboratório CERT em "Toulouse". A linguagem de

programação LAU tem um modelo a fluxo de dados, mas a organização dos programas é de fato baseado sobre os conceitos de fluxo por controle.

Na Fig.2.13 mostramos a organização do sistema de computador LAU.

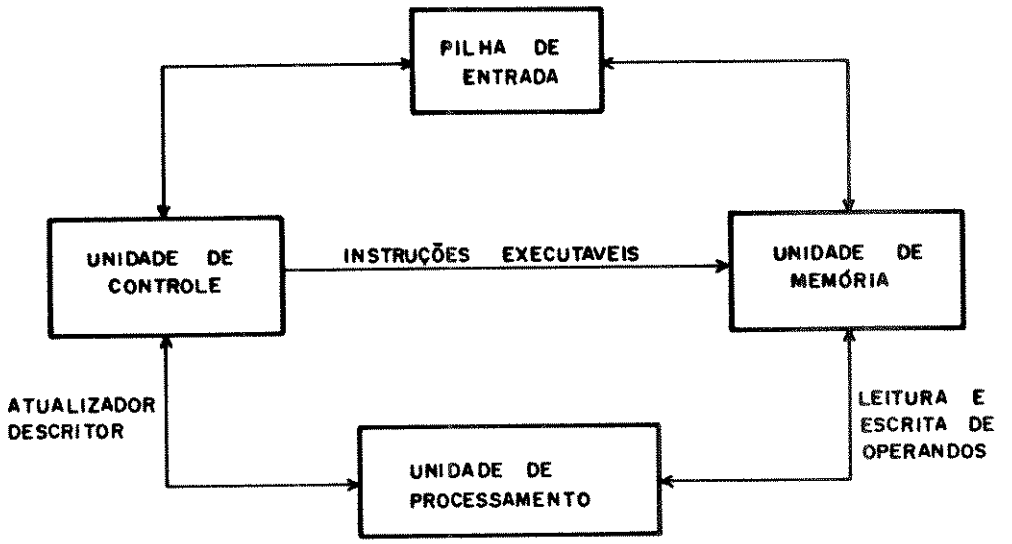


FIGURA 2.13 - SISTEMA LAU.

O sistema contém:

a) Unidade de Memória - que providencia a armazenagem de instruções e dados.

b) Unidade de Controle - sustentando a memória de controle.

c) Unidade de Processamento - consistindo de 32 elementos de processamento idênticos.

Cada elemento é um processador micro-programado de 16

bits. Talvez a parte mais interessante é a unidade de controle, onde o contador de programa (von Neumann) é substituído por duas memórias: a memória de controle de instrução e a memória de controle de dados.

2.3.2.1.7 - Computador a Fluxo de Dados de "Newcastle"

Projetado na Universidade de "Newcastle", o computador JUMBO foi construído para estudar a integração de computação a fluxo de dados e fluxo por controle.

A organização da máquina JUMBO é mostrada na Fig.2.14.

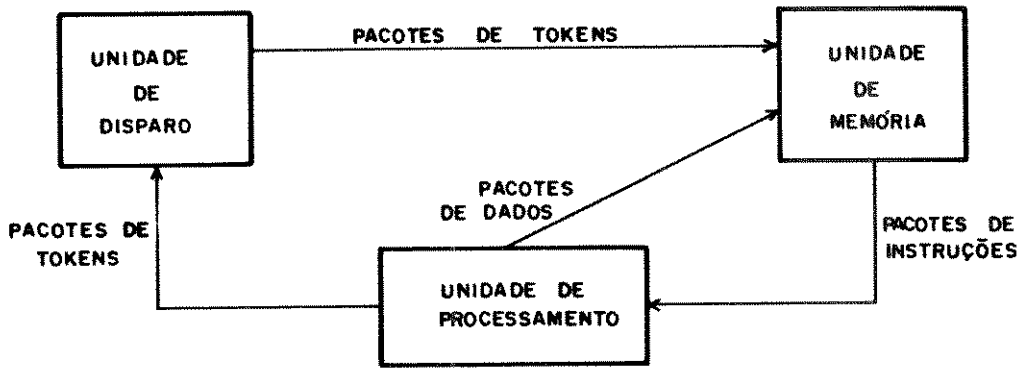


FIGURA 2.14 - COMPUTADOR A FLUXO DE DADOS DE "NEWCASTLE".

Ela consiste de três unidades principais interconectadas por "buffers" "FIFO" :

a) Unidade de Disparo - controla a habilitação de instruções igualando um conjunto de pacotes de dados, que são enviados para a Unidade de memória quando completadas.

b) Unidade de Memória - providencia armazenagem para dados e instruções.

c) Unidade de Processamento - suporta execução de instruções e a distribuição dos resultados.

2.4 - COMENTÁRIOS

Apresentamos neste capítulo as características básicas para sistemas de processamento paralelo cujo processamento é dirigido por controle. Em seguida caracterizamos os sistemas cujo processamento é dirigido por dados.

Múltiplos "JOBs", multiprogramação, "time-sharing", multiprocessamento, paralelismo a partir da divisão de programas em processos e tarefas, paralelismo entre múltiplas instruções de um programa, e a concorrência dentro de cada instrução, em máquinas com as estruturas propostas por Flynn (SISD, SIMD, MISD, MIMD), foram analisados para descrever paralelismo a partir dos sistemas cujo processamento é dirigido por controle.

Caracterizamos software nestas máquinas através da centralização do controle (Sistema Operacional Centralizado), ou da distribuição do controle (Sistema Operacional Distribuído). Neste último levantamos características dos sistemas distribuídos de uma forma geral: preço/desempenho; crescimento incremental; disponibilidade e confiabilidade; a questão da granularidade fina, média, e grossa; dificuldade em manipular "overhead"; e a dificuldade em obter um estado global do sistema.

Na caracterização para os sistemas cujo processamento é

dirigido por dados, descrevemos a linguagem original para programas a fluxo de dados, e a máquina que foi proposta para execução da linguagem.

Características como Máquina a Fluxo de Dados Estática, e Máquinas a Fluxo de Dados Dinâmica, foram consideradas.

Foram apresentadas também as estruturas de algumas máquina a fluxo de dados ("MIT", "TI-DDP", "UTAH", "IRVINE", "MANCHESTER", "LAU", e "NEWCASTLE"), onde verificamos que algumas arquiteturas são parecidas à proposta da máquina original (proposta feita por [DENNIS,1974]), e outras mais na linha de multiprocessamento.

Processamento Paralelo portanto, vem evoluindo ao longo dos anos. Teve seu início a partir da divisão do tempo de uso da UCP, passa pelas arquiteturas "array processor", "pipeline", multiprocessamento, e fluxo de dados; e evolui para sistemas massivamente paralelos implementados inclusive para operar através de processamento dirigido por dados.

C A P Í T U L O 3

SOFTWARE BÁSICO DE MÁQUINAS A FLUXO DE DADOS

3.1 - INTRODUÇÃO

No capítulo anterior levantamos algumas características fundamentais das arquiteturas a fluxo de dados tanto com relação a programas a fluxo de dados como também com relação à organização de máquinas tipo fluxo de dados. Passamos agora a descrever mais detalhadamente os aspectos de software para computadores com arquiteturas a fluxo de dados.

3.2 - SOFTWARE NAS ARQUITETURAS TIPO von NEUMANN

Em função dos princípios de von Neumann, podemos observar nas máquinas convencionais cinco componentes básicos: a Unidade de Entrada (E), a Unidade de Saída (S), a Unidade de Memória (MP), a Unidade Lógica e Aritmética (ULA), e a Unidade de Controle (UC). A realização de uma computação (execução de um programa) é feita através da comutação entre ciclos de busca e ciclos de execução de instruções na seguinte forma: através de um Registrador Contador de Programa (CP) é acessada uma posição de

memória onde se encontra a instrução a ser executada, que é removida da memória e posicionada em um Registrador de Instrução (RI) na UC, correspondendo esta à fase de busca da instrução; em seguida esta instrução é decodificada pela UC para ser executada, se necessário ocorre aqui a busca de dados, sendo concluída com a execução efetiva da instrução através da ULA e atualização do CP com o endereço da próxima instrução.

O que se pode observar nesta estrutura é um controle que se baseia em uma sequência de eventos bem definidos principalmente pelo fato de que a próxima instrução é acessada somente após uma instrução ter sido executada, característica essa fundamental nas máquinas convencionais.

Com a estrutura acima pode-se verificar que todo software, a partir do nível convencional da máquina, possue uma característica puramente sequencial, sendo esse o caso de programas escritos em linguagem "assembly" e consequentemente programas escritos em linguagens de alto nível.

Isto não significa que é impossível introduzir-se alguma forma de execução paralela no modelo de computação von Neumann [ARVIND e IANNUCCI, 1983]. Linguagens de programação concorrentes permitem obter um certo paralelismo em uma máquina implementada conforme os princípios de von Neumann, porém o paralelismo aqui é obtido de forma temporal e não espacial [BEN-ARI, 1982], [KIRNER e MENDES, 1988]. Se quisermos obter um paralelismo espacial será necessário incluir à organização da máquina, modificações que vão desde estruturas tipo "pipeline" até estruturas com vários processadores executando computações simultaneamente.

Apesar das diferentes características nestas estruturas, a execução dos programas a nível convencional, continua sendo executado de forma sequencial, é o caso de multiprocessadores, onde programas são executados simultaneamente, mas em cada processador individual de forma sequencial.

3.3 - SOFTWARE NAS ARQUITETURAS TIPO FLUXO DE DADOS

Com a proposta de um máximo de paralelismo, o software nas arquiteturas a fluxo de dados, como visto nos capítulos anteriores, implementados através de uma linguagem gráfica, caracterizam uma estrutura totalmente diferente para a execução dos programas, agora não através de um controle pré-estabelecido para a execução das instruções, mas sim em função da disponibilidade dos dados. Consequentemente, a organização para a máquina que executa este tipo de programa, não pode depender, por exemplo, de um contador de programa, mas de uma estrutura que habilite uma instrução quando seus dados estão disponíveis, e como vimos, isto vem sendo implementado através de estruturas contendo várias unidades funcionais (Processamento, Memória, Rede de Distribuição, Rede de Arbitração, Rede de Controle), que se comunicam através de pacotes de informação, como instruções para execução, dados ou controle.

Poder-se-ia definir aqui a existência de um software básico (a partir do nível de microprograma se necessário), rodando sobre um hardware com as características descritas na

proposta feita por Dennis, responsável pelo funcionamento da máquina, que: detecta uma instrução habilitada, encaminha-a para execução, a execução propriamente dita, e o encaminhamento dos resultados [BUGGATTI, 1986], que como dissemos está diretamente ligado aos gabaritos de memória para cada operação a fluxo de dados.

Conforme descrito no capítulo anterior pudemos verificar que algumas máquinas foram implementadas utilizando vários processadores, o que nos leva a uma categoria de máquinas executando programas a fluxo de dados, que são os sistemas multiprocessadores, que apesar de serem sistemas tipicamente von Neumann, podem executar programas a fluxo de dados através de um conjunto de macro operações. Lógicamente o tempo de processamento e comunicação nestes sistemas são fundamentais.

O software por sua vez deverá ser capaz de gerenciar todo o processo de execução e comunicação entre os elementos do sistema de tal forma a permitir a execução de programas a fluxo de dados. Este software deverá ser distribuído entre todos os elementos para que executem os programas a fluxo de dados corretamente.

3.4 - COMENTÁRIOS

Apresentamos neste capítulo a característica fundamental para as arquiteturas que utilizam os princípios de von Neumann (Unidade de Entrada/Saída, Unidade lógica e aritmética, Memória Principal, Unidade de Controle, Contador de Programa, e Registrador de Instrução, Ciclo de Busca, Ciclo de

Execução).

A partir desta análise, passamos a caracterizar o que seria a implementação da Linguagem Original para programação a fluxo de dados, em uma máquina própria para a execução de programas escritos nesta linguagem, e detectamos, por exemplo, a não existência de um contador de programa nesta máquina.

Consideramos também a utilização de máquinas com arquitetura multiprocessamento executando programação a fluxo de dados de forma MACRO (operações a fluxo de dados implementadas através de um conjunto de instruções de máquina do multiprocessador), apontando a necessidade do sistema ter alta velocidade de processamento como também alta taxa de comunicação entre os elementos do sistema.

O Software Básico das máquinas a fluxo de dados dependerá então da arquitetura da máquina. Para uma arquitetura igual a proposta inicial (Rede de arbitração, Rede de distribuição, Seção memória, Seção de processamento), não deverá existir contador de programa, existirá sim um software de controle distribuído entre os elementos do sistema de tal forma a fluirem os dados pela máquina. Para um sistema multiprocessamento, deverá existir um software de controle instalado em cada microprocessador (MACRO), que receba, executa, e transmita dados dos programas a fluxo de dados entre os elementos do sistema multiprocessador.

C A P Í T U L O 4

A UTILIZAÇÃO DO COMPUTADOR PARALELO ESTRUTURADO RECURSIVAMENTE (CPER) COMO UMA MÁQUINA A FLUXO DE DADOS DINÂMICA (MFDD).

4.1 - INTRODUÇÃO

Como visto, as máquinas com arquiteturas a fluxo de dados surgiram como alternativa para processamento paralelo, visando ultrapassar os limites de desempenho de máquinas von Neumann e minimizar a tarefa de identificação do paralelismo a nível de software.

Vimos também que existem diversos tipos de máquinas a fluxo de dados estruturadas de forma estática ou dinâmica. As máquinas com estruturação estática manipulam reentrância pelo método "lock" ou "acknowledge", enquanto as máquinas com estruturação dinâmica usam "code copying" ou "tagged tokens" [VEEN, 1986]. A organização interna dessas máquinas pode usar vários tipos de topologias tais como: anel, barramentos hierárquicos, etc.

Nesses ambientes, um dos problemas críticos do processamento de programas a fluxo de dados está na enorme demanda por comunicação de dados (processamento de grão fino).

Assim, a exploração eficiente do paralelismo exige arquiteturas paralelas com alto desempenho de comunicação. Uma das abordagens, utilizadas para obtenção de altas taxas de comunicação, está associada ao uso de múltiplos barramentos organizados hierarquicamente [SHIMADA et al, 1986], [ITO et al, 1986], [SATO, 1992].

4.2. O COMPUTADOR PARALELO ESTRUTURADO RECURSIVAMENTE (CPER)

4.2.1. Arquitetura do CPER

O CPER usa como referência uma estrutura básica formada por um computador Hospedeiro e um barramento paralelo interligando N processadores de execução como mostrado na Fig.4.1.

Estes processadores também são interligados através de uma rede em anel com conexão ponto-a-ponto. Tanto o barramento, quanto a rede em anel, possuem "gateways" para interligação com o mesmo tipo de meio de comunicação externo [KIRNER e MARQUES, 1986], [KIRNER, 1989], [GONÇALVES, 1991].

Para a montagem do sistema, constrói-se inicialmente uma estrutura básica com N processadores de execução simples (nível 1, que chamamos de "CLUSTER"). A visualização dessa estrutura básica como um processador de execução mais potente (nível 2) permite a montagem de um novo barramento paralelo e de um novo anel ponto-a-ponto interligando N desses novos processadores. O uso recursivo da mesma regra de agrupamento permite o crescimento do sistema até o nível desejado.

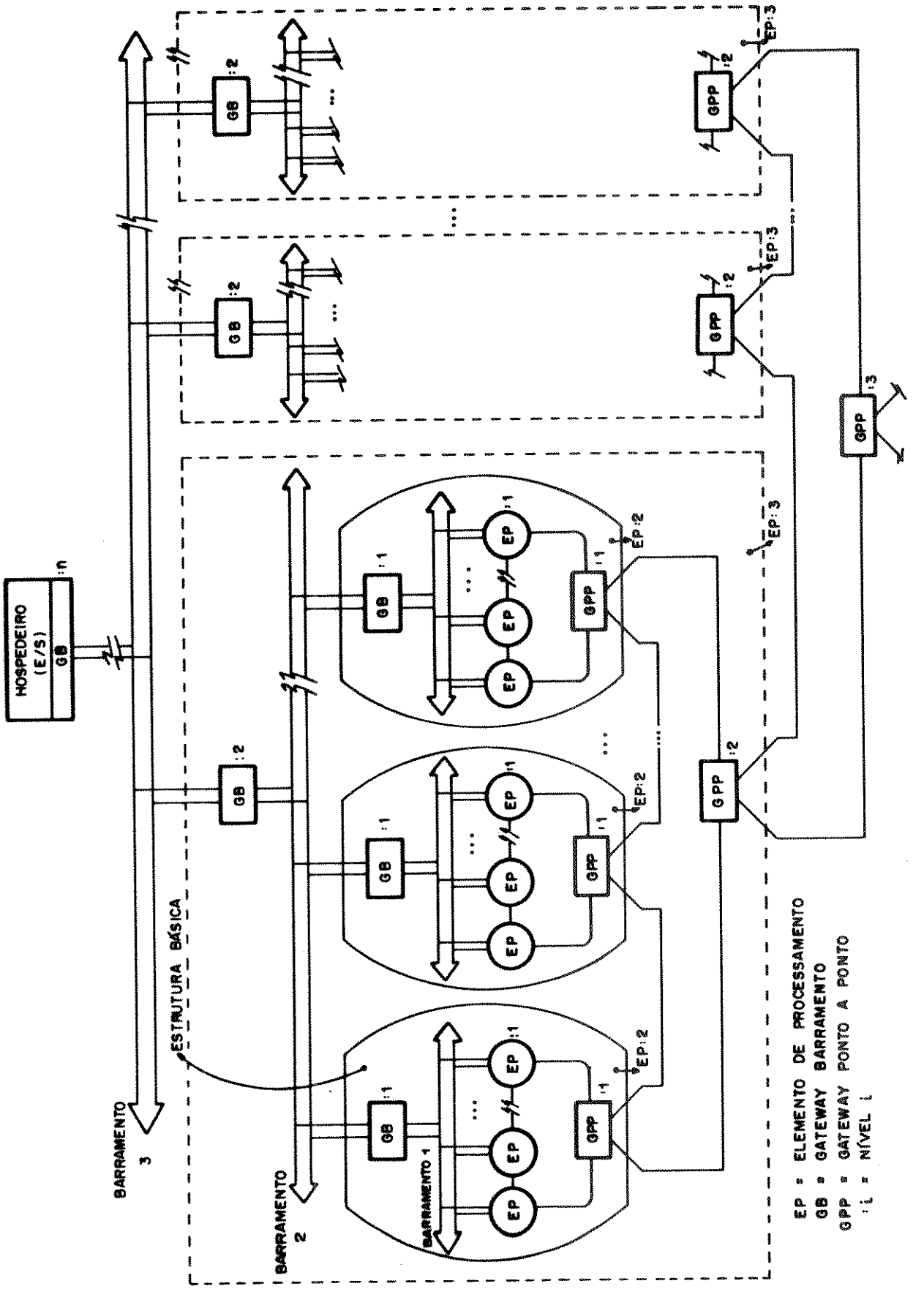


FIGURA 4.1 - ARQUITETURA DO CPER.

4.2.2. Elementos de Processamento (EP)

O Elemento de Processamento simples (nível 1) contém 3 módulos: interface com o barramento, unidade de processamento, e interface ponto-a-ponto (Fig.4.2).

A interface com o barramento é encarregada de receber e de enviar informações através do barramento paralelo. Para isto, ela deve contar com uma fila de "buffers" de entrada, um "buffer" de saída, um circuito de reconhecimento de endereço, um circuito de acesso ao barramento, e uma lógica de Tolerância a Falhas. A unidade de processamento é um sistema processador-memória de alto desempenho, responsável pela movimentação e processamento dos dados. A interface ponto-a-ponto tem a função de receber e de enviar informações da rede em anel.

O projeto está estruturado com a utilização de circuitos "MSI" e "LSI" na interface com o barramento, e com a inserção de um processador "transputer" (32 bits/20 MIPS) e 32 KBytes de "higher performance memory" na unidade de processamento e na interface ponto-a-ponto.

4.2.3 - Barramento Paralelo

O barramento paralelo (TABELA I) contém 65 linhas, sendo 32 para transferência de dados (podendo ser ampliado), 8 para cabeçalho de pacote, 8 para endereço fonte, 8 para endereço destino, 4 para reserva de dados, e 5 para controle. Existem

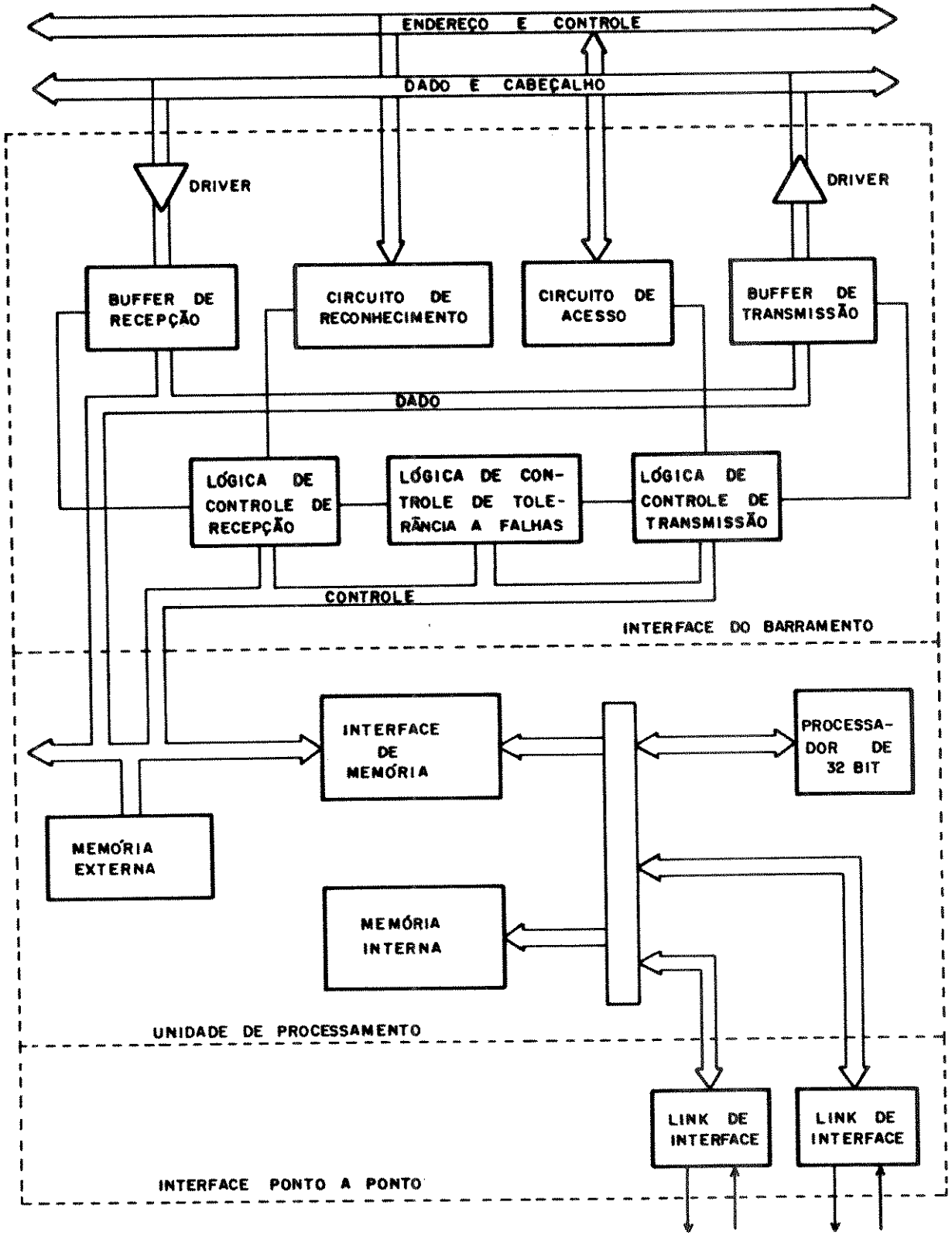


FIGURA 4.2 - DIAGRAMA DE BLOCO DO ELEMENTO DE PROCESSAMENTO.

também 8 linhas sobressalentes para o caso de falhas nas linhas. É possível interligar até 64 processadores de execução em um barramento, em função das restrições elétricas dos circuitos usados na interface com o barramento [GONÇALVES, 1991].

O mecanismo de acesso ao barramento é tal que: um sinal de prioridade circula entre todos os dispositivos num esquema

TABELA I - LINHAS DOS BARRAMENTOS

No. de LINHAS	NOME	DESCRIÇÃO
32	dados	conteúdo da mensagem
8	cabeçalho	cabeçalho da mensagem
8	end. emissor	endereço do dispositivo que envia mensagem
8	end. destino	endereço do dispositivo que receberá a mensagem
1	ocupado	indica se o barramento está ocupado; foi implementado com lógica de fiação "wired OR"
1	hab_end	habilita os dispositivos a reconhecerem o endereço destino presente no barramento
1	resetbus	coloca todos os dispositivos no estado inicial
1	hab_saida_RCP	indica que a saída do Registrador Circulante de Prioridade pode ser consultada
1	chegou	indica que a mensagem chegou no dispositivo destino
1	dado_lido	indica que a mensagem foi lida pelo dispositivo destino
1	estouro	indica a existência de algum "buffer" de recepção cheio; foi implementado com lógica de fiação "wired OR"
1	hp	acesso ao barramento reservado para dispositivos de alta prioridade
1	sincronismo	linha de difusão de relógio
65 linhas de Comunicação		

"round-robin" (Fig.4.3), através de um registrador circulante de prioridade (centralizado em algum EP, mas acionado por todos os EPs do barramento), que estando habilitado circula essa prioridade e suas saídas se tornam as linhas de prioridades de cada um dos EPs. Apenas um EP terá o sinal habilitado.

Para garantir acesso ao barramento, o EP deverá manter-se com a prioridade, bloqueando temporariamente a sua circulação (inibindo o registrador circulante de prioridade através do sinal HAB_SAIDA_RCP), e após ter reservado o barramento, só então libera a circulação. Com esse procedimento a arbitragem para o próximo acesso ao barramento terá se encerrado (o pretendente mais próximo do EP emissor é que estará com o sinal) antes mesmo de encerrar-se a transmissão.

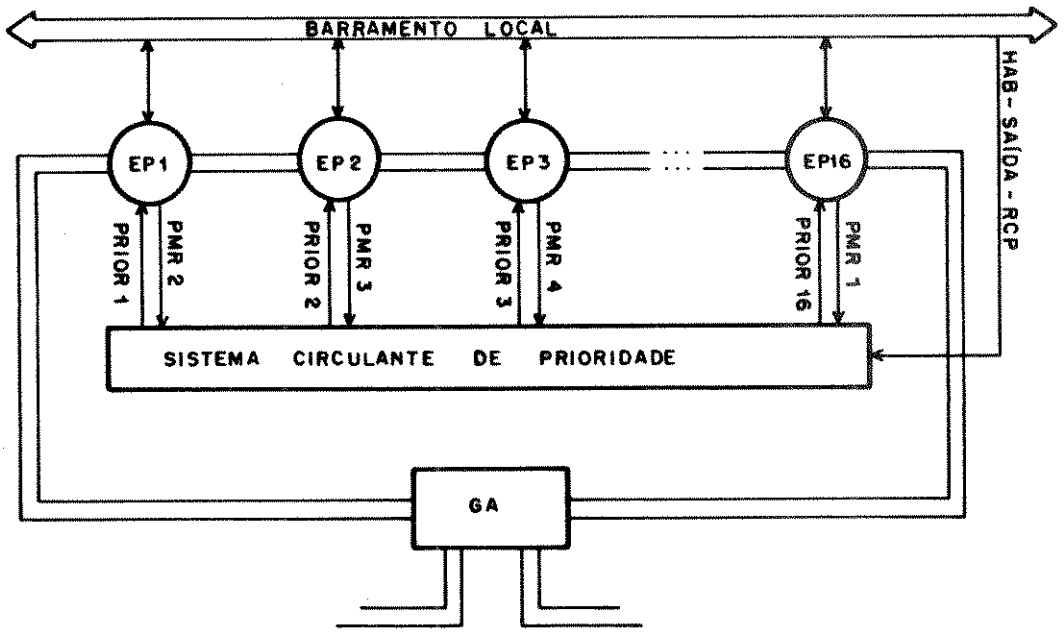


FIGURA 4.3 - CONEXÕES ENTRE O SISTEMA CIRCULANTE DE PRIORIDADE E OS ELEMENTOS DE PROCESSAMENTO.

Esta sobreposição dos ciclos de arbitragem aos ciclos de transmissão, diminui consideravelmente o "overhead" do sistema. O processo de decisão sobre qual EP terá a prioridade é realizado dentro do registrador, que possui uma linha de entrada vinda de cada EP (utilizada para especificar a posição de cada EP no esquema "round-robin"), e uma linha de saída para cada EP que define o sinal de prioridade. O sinal de Inibe Registrador é comum no barrameto pois só é utilizado por aquele EP que ganhou a prioridade.

Os "gateways" são semelhantes aos Elementos de Processamento (proposto com "transputer"), e possuem duas interfaces uma para cada barramento, com o mesmo conjunto de linhas e o mesmo mecanismo de arbitragem.

4.2.4 - Tolerância a Falhas

O sistema apresenta alta confiabilidade, proveniente do uso de técnicas de Tolerância a Falhas como: uso de linhas redundantes no barramento paralelo, uso da rede em anel ponto-a-ponto bidirecional, e uso de procedimentos de teste, isolamento, e reintegração de Elementos de Processamento.

No caso de existir falha no barramento ou na interface com o barramento, a rede em anel ponto-a-ponto bidirecional será utilizada na comunicação entre os elementos daquele "cluster".

Esta alternativa de curto prazo permite que o sistema continue operando, ao mesmo tempo em que viabiliza

o teste das unidades danificadas [MARQUES et al, 1988], [GONÇALVES, 1991].

O potencial do processamento é mantido através do diagnóstico periódico de todos os EPs. O diagnóstico é realizado em cada "cluster" por dois EPs dedicados à monitoração do sistema. Ocorrendo a falha em um deles, o outro notificará o sistema e passará a substituí-lo. O resultado do diagnóstico é passado ao Hospedeiro. Os EPs monitores também controlam o esquema de passagem de prioridade.

4.3 - SOFTWARE NO CPER

O software de base do computador paralelo deverá ser constituído por um núcleo , parte de um sistema operacional distribuído. Esse núcleo executará as funções básicas do sistema como: comunicação e sincronização, gerenciamento de memória, tratamento de falhas, etc. O sistema operacional distribuído deverá ser responsável pelas funções de carregamento, manipulação de Entrada/Saída, reconfiguração do sistema, etc. As camadas de apoio podem ser ajustadas ou complementadas para adequar-se ao tipo de aplicação que for executada no computador [SILVA e KIRNER, 1989].

4.3.1 - Executando Programas a Fluxo de Dados no CPER

Com um software básico apropriado, o CPER funciona como uma máquina a fluxo de dados dinâmica usando "tagged token". Assim, as operações do programa a fluxo de dados são disparadas

por dados que tenham o mesmo "tag" associado [VEEN, 1986], [SILVA e KIRNER, 1989].

Para cada execução de um programa, função, ou procedimento, é gerado um "tag", representando uma ativação. No caso de existirem "loops", implementados com operações iterativas (" WHILE, REPEAT, FOR "), a entrada em cada uma dessas operações gerará um novo "tag" que será associado aos parâmetros de entrada com os "tags" antigos. Para cada ciclo do "loop", o novo "tag" deverá ser ajustado para indicar a iteração. Como as operações iterativas podem ser aninhadas, é útil que o "tag" também contenha o nível de aninhamento (Fig.4.4). Ao término de cada operação iterativa, "function", "procedure" ou programa, a parcela de "tag" correspondente é destruída.

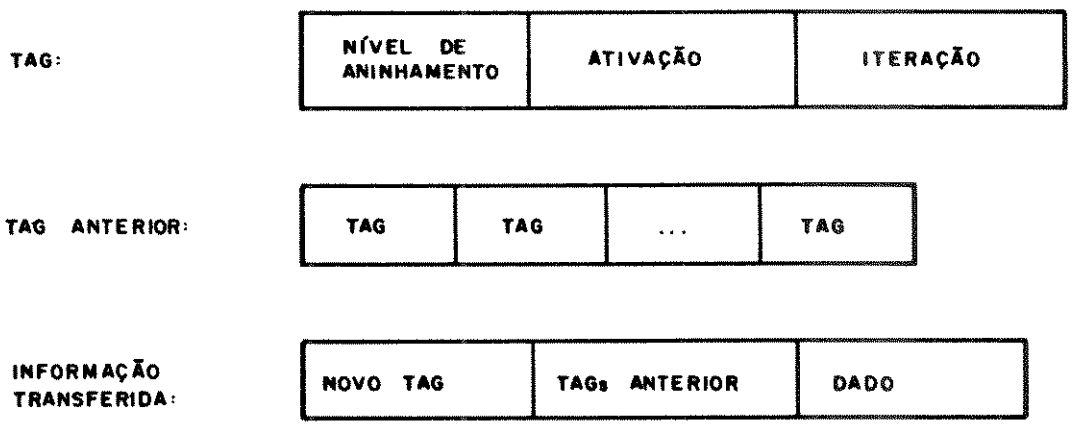


FIGURA 4.4 - FORMATO DO DADO.

Antes da execução de um programa a fluxo de dados na máquina paralela, é necessário que suas várias operações sejam alocadas nos Elementos de Processamento, por um carregador [D'AREZZO e KIRNER, 1989].

Na fase de execução, cada operador, ao receber todos os dados de entrada necessários com o mesmo "tag" associado, executará sua operação e remeterá o resultado para outro operador. Esta troca de informações ocorrerá até que o resultado final de cada ativação do programa seja produzido.

4.3.2 - Estrutura e funcionamento dos Operadores

A maioria dos operadores mais comuns usados em grafos a fluxo de dados (Fig.4.5) podem ser mapeados em um operador genérico com 3 entradas e 3 saídas (Fig.4.6A). Isto facilita o tratamento e manipulação dos operadores tanto no carregador quanto nos Elementos de Processamento.

Para operadores com número de campos maior do que o previsto no operador genérico, o sistema usará um gabarito ampliado.

O gabarito simplificado de memória de um operador genérico deve conter pelo menos a especificação da operação, 3 campos de dados de entrada, 3 campos de endereço de saída, um campo indicando os campos de dados e de endereços válidos para a operação (máscara), e um campo indicando a presença ou ausência dos dados ou endereços válidos (status). Como o sistema permite o uso de "tagged token", poderão existir várias ativações e várias iterações de cada operação. Neste caso, o gabarito de memória assume o formato de uma lista encadeada (Fig.4.6B), contendo uma cabeça principal que aponta a lista de ativações sucessivas. Cada ativação, por sua vez, serve como cabeça secundária para apontar

a lista de iterações sucessivas.

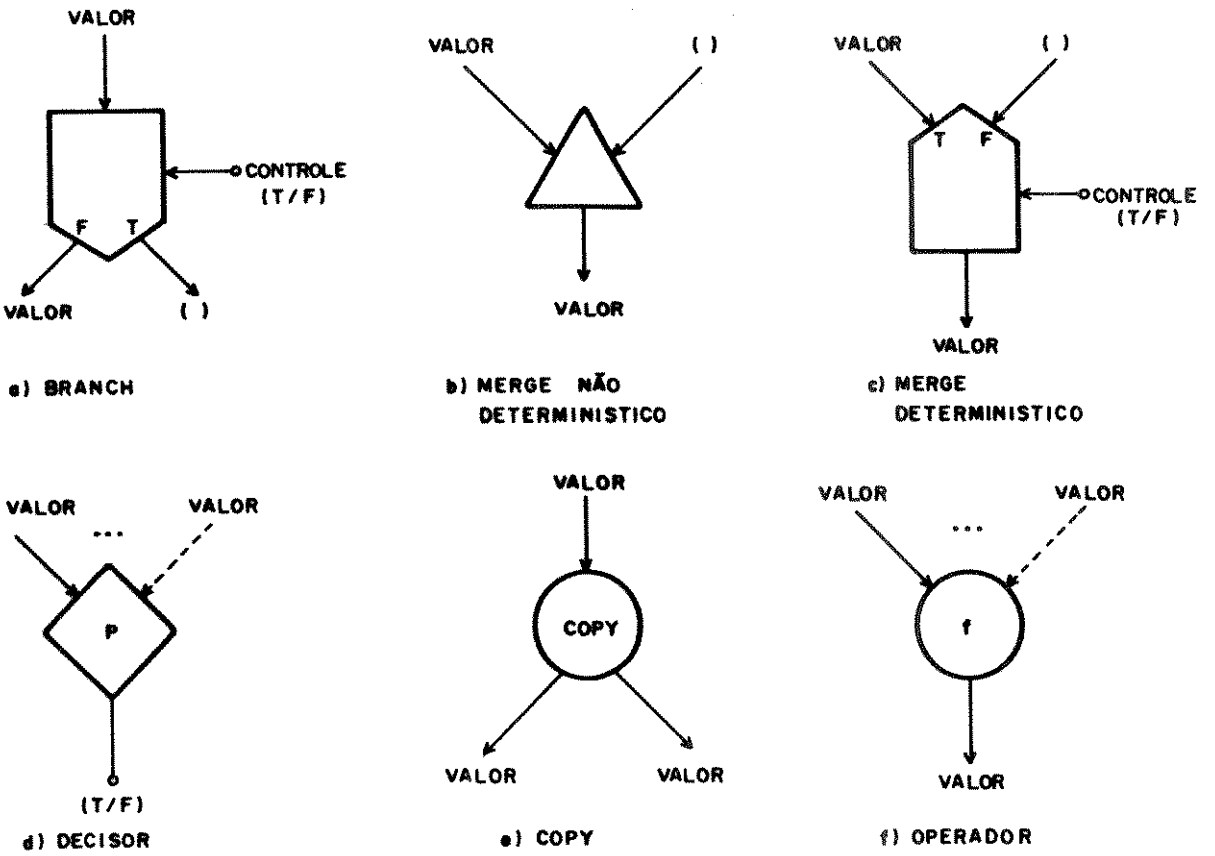


FIGURA 4.5 - OS OPERADORES PARA PROGRAMAS A FLUXO DE DADOS.

A cabeça principal contém informações fixas como a operação, endereços destinos, e máscara de dados e endereços. A cabeça secundária contém a parcela do "tag" recente relativa a ativação e os outros "tags" anteriores. Cada iteração contém a parcela do "tag" relativa a iteração, o estado dos dados e os campos de dados.

Uma operação simples, quando recebe um dado de uma única ativação, gera uma cabeça secundária e a iteração inicial,

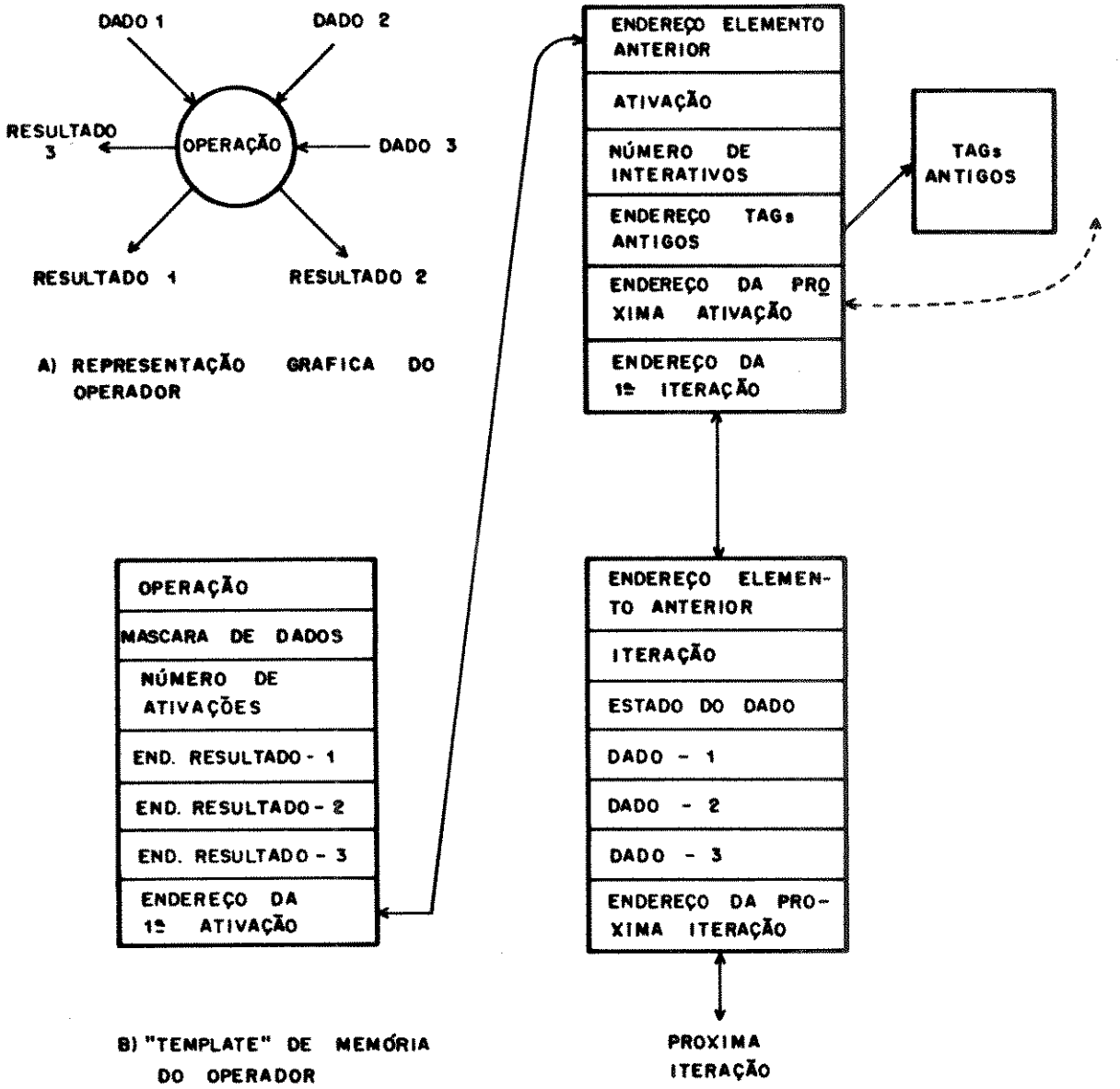


FIGURA 4.6 - ESTRUTURA GENÉRICA DE UM OPERADOR.

ficando com o gabarito de memória igual ao da Fig.4.6. Uma operação dentro de um "loop" iterativo vai adicionando novos elementos na lista encadeada apontada pela cabeça secundária conforme vão chegando dados com novos "tags" alterados pelo campo de iteração.

A chegada de todos os dados de uma operação provoca a execução da operação e a destruição do elemento da lista encadeada; se não restar nenhum elemento ligado em uma cabeça secundária, ela será destruída.

A localização de um dos vários conjuntos de dados (elemento de lista encadeada) é realizada pela pesquisa nas ativações seguida pela pesquisa nas iterações. Quando o conjunto de dados não é encontrado ele é criado e o dado é colocado. A obtenção de um bom desempenho na execução de operações depende da existência de um algoritmo de busca eficiente.

Cada Elemento de Processamento tem um gerente de memória que cuida das alocações e devoluções de blocos de memória utilizados, quando os elementos de lista encadeada são criados ou destruidos.

4.3.3 - Construtores Iterativos

As construções iterativas (" WHILE, REPEAT, FOR "), são estruturas que, apesar de serem muito importantes em programação a fluxo de dados necessitam de certos cuidados para sua utilização.

Para garantir o funcionamento correto destas

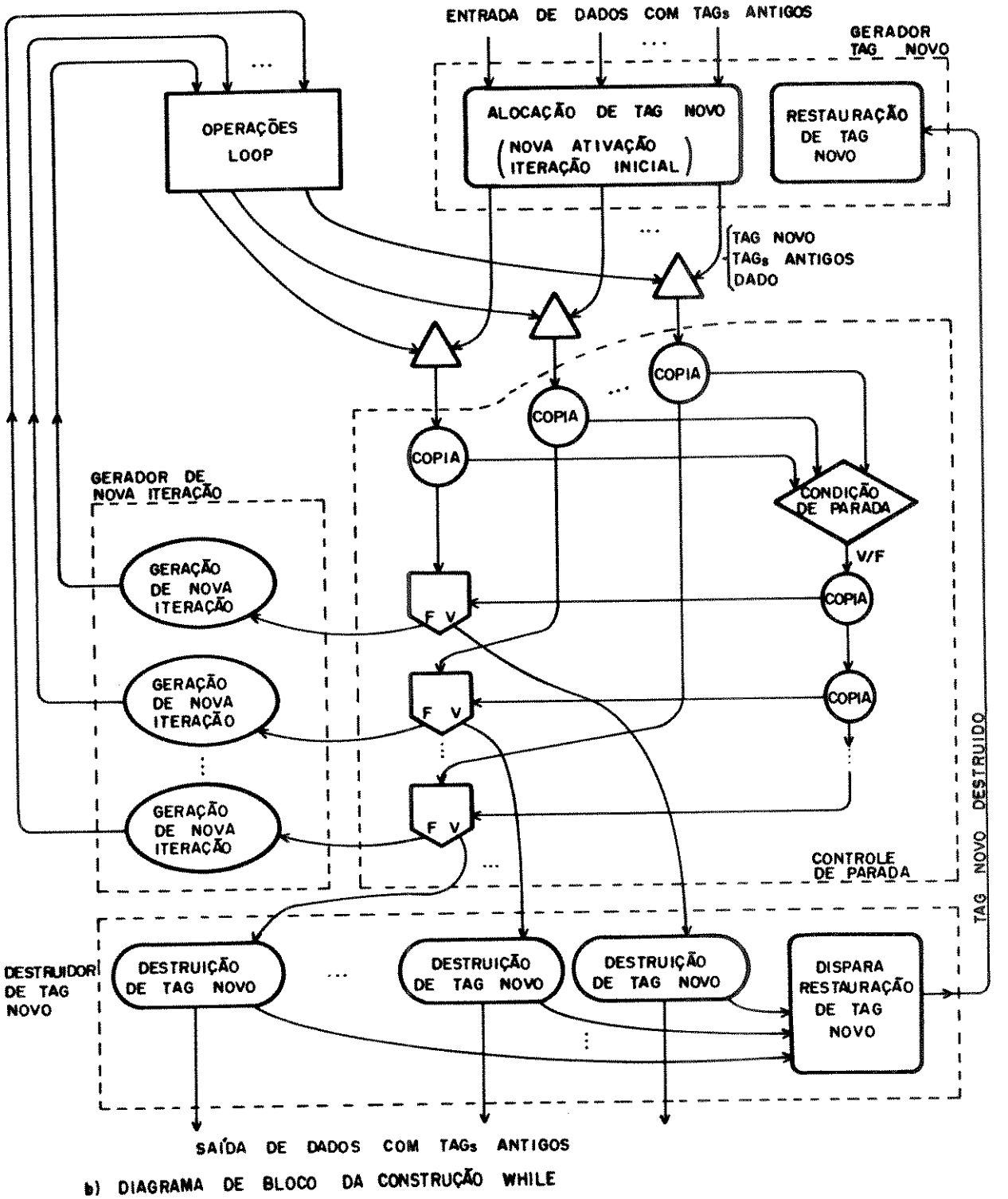
estruturas, o seu início acontece somente quando todos os parâmetros de entrada ficam disponíveis. Isto porque é preciso gerar um novo "tag" para todos os dados que entram na estrutura e que possuem um mesmo "tag" anterior. Se isso não for sincronizado na entrada, fica muito difícil gerar esse novo "tag" para os dados que são parceiros (possuem um mesmo "tag" anterior).

A partir daí, as variações de velocidade de execução das operações de cada "loop" podem fazer com que os resultados finais sejam emitidos em tempos diferentes para serem usados por outras operações externas. A construção iterativa "WHILE" é apresentada aqui na forma de diagrama de blocos, que são operações mais complexas ou que contém subgrafos. Esses blocos são: "new tag manager", "stop control", " new iteration generators", "new tag destructors", e "loop operations". O bloco "new tag manager" é responsável pela alocação de um novo "tag" (ativação, iteração inicial) a um conjunto de dados e pela devolução desse "tag" ao conjunto de "tags" disponíveis. O bloco "stop control" verifica a condição de parada da iteração. O bloco "new iteration" modifica o "tag", gerando um novo valor para o campo de iteração. O bloco "new tag destructors" elimina dos dados de saída o "tag" recebido na entrada da construção iterativa, e o devolve ao bloco "new tag manager". O bloco "loop operations" contém as operações internas a construção iterativa.

O diagrama de blocos da construção "WHILE" encontra-se na Fig.4.7. Esta estrutura genérica deve ser ajustada para cada caso principalmente nos módulos: "stop control" e "loop operations". A operação DECISOR deverá receber somente os dados que fazem parte da condição de parada. As operações do

WHILE CONDIÇÃO VERDADEIRA
DO OPERAÇÃO LOOP

a) CONSTRUÇÃO DE LINGUAGEM DA CLAUSULA WHILE



b) DIAGRAMA DE BLOCO DA CONSTRUÇÃO WHILE

FIGURA 4.7 - O CONSTRUTOR ITERATIVO "WHILE".

"loop" poderão envolver um ou mais dados recebidos pelo "loop" assim como valores intermediários ali gerados. A execução da estrutura não modifica os "tags" velhos dos dados de entrada, pois ela adiciona um novo "tag" para o seu funcionamento e o retira no final.

4.3.4 - O Carregador

O carregador é um processo localizado no Hospedeiro encarregado de mapear convenientemente as operações de um grafo a fluxo de dados aos Elementos de Processamento do sistema [D'AREZZO e KIRNER, 1989].

Essa distribuição dependerá do número de operações do programa e do número de Elementos de Processamento do sistema. Se houver um número maior de Elementos de Processamento, a colocação de uma operação em cada Elemento de Processamento é mais apropriada para a exploração de múltiplas ativações. Isto pela facilidade de gerenciamento das múltiplas filas de ativações e iterações, conforme proposta anterior para o gabarito de uma operação, se restringindo apenas a uma operação. O que não acontece no caso de existir um número maior de operações, algumas delas devendo ficar juntas em um mesmo Elemento de Processamento.

A escolha das operações que deverão ficar sozinhas ou juntas nos Elementos de Processamento terá implicações no desempenho do sistema. É recomendável que operações de estruturas iterativas de níveis mais internos tenham prioridade para ficarem sozinhas pois apresentam maior taxa de execução. A escolha de grupos de operações de níveis mais externos para ficarem juntas

pode ser sistematizada no sentido de permitir a comunicação interna, que é muito rápida, com o mínimo de prejuízo de ativações sucessivas.

O algoritmo 1 e a Fig.4.8 apresentam um exemplo de um programa convertido para um grafo a fluxo de dados. Neste grafo, as operações da construção iterativa localizada à esquerda entre os pontos P4 e P5, seriam escolhidas para ficarem sozinhas nos Elementos de Processamento de um mesmo "cluster". As operações iniciais e, eventualmente aquelas da primeira construção iterativa, seriam agrupadas em pequenos conjuntos e alocadas a outros Elementos de Processamento.

(A L G O R I T M O 1)

```
Program Exemplo;
Var x,xfat,count: integer;
Var fat,fats: real;
Begin
  fats:=0;
  count:=0;
  Readln(x);
  if x>=0 then
    while x<10 do
      Begin
       fat:=1;
       xfat:=x;
       while xfat>0 do
        Begin
          fat:=fat*xfat;
          xfat:=xfat-1;
        End;
       fats:=fats+fat;
       x:=x+1;
       count:=count+1;
      End
End.
```

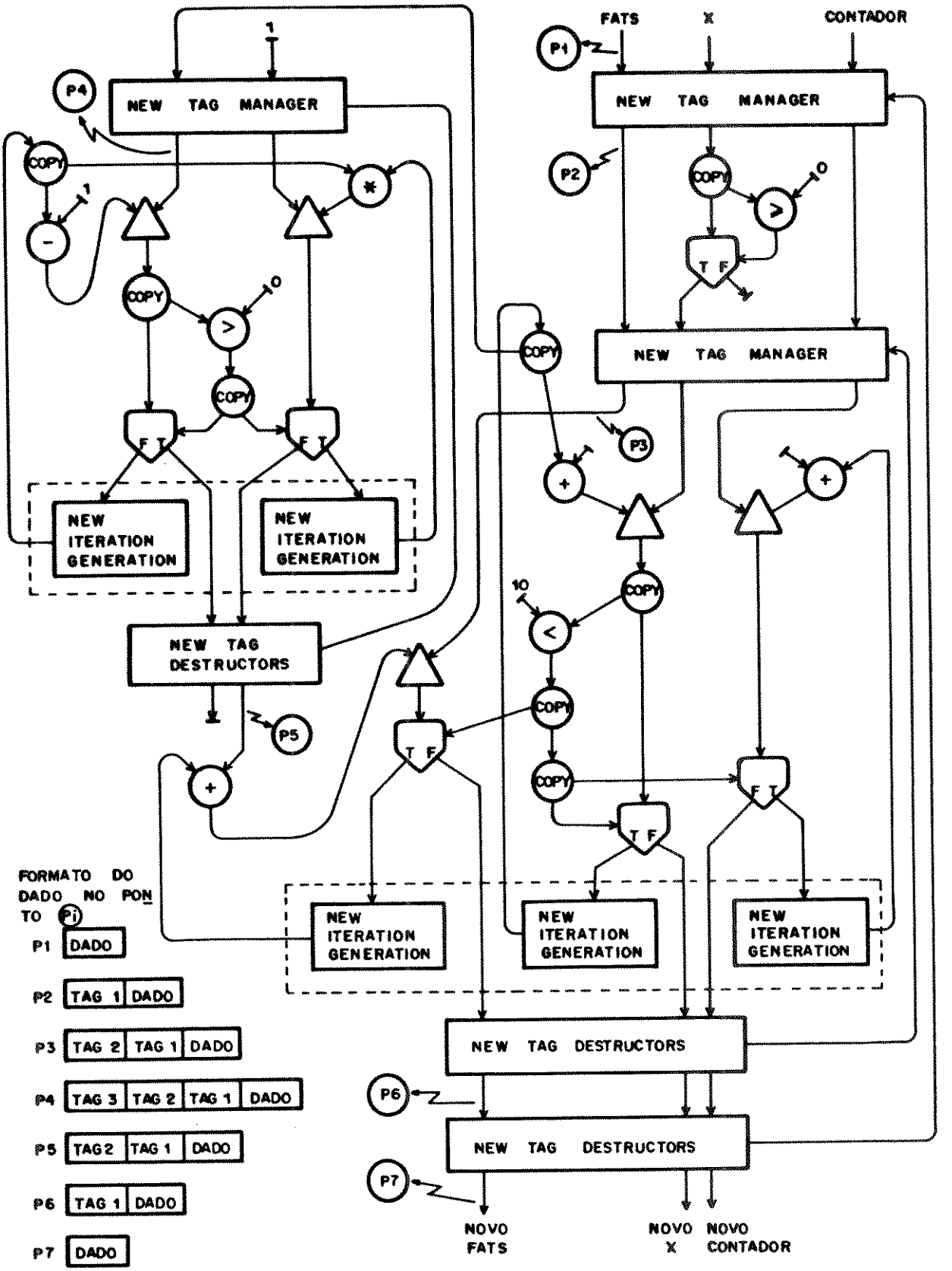


FIGURA 4.8 - EXEMPLO DE UM PROGRAMA A FLUXO DE DADOS SENDO EXECUTADO NO CPER.

4.4 - CONCLUSÃO

Neste capítulo vimos que em geral as máquinas a fluxo de dados podem utilizar estruturas tipo anel, barramento hierárquico, etc.

Vimos que um dos problemas críticos do processamento de programas a fluxo de dados está na enorme demanda por comunicação de dados (processamento de grão fino); e que a exploração eficiente de paralelismo exige arquiteturas paralelas com alto desempenho de comunicação. É o caso dos sistemas com múltiplos barramentos organizados hierarquicamente [SHIMADA et al, 1986], [ITO et al, 1986], [SATO, 1992].

O projeto CPER teve então como objetivo satisfazer todos esses requisitos básicos, através de:

a) Utilização de circuitos "MSI" e "LSI" na interface com o barramento.
b) Inserção de um processador "transputer" (32 bits/20 MIPS) e 32 KBytes de "higher performance memory" na unidade de processamento.
c) Interface ponto-a-ponto.
d) Sobreposição dos ciclos de arbitragem aos ciclos de transmissão no barramento.
e) Alta confiabilidade, proveniente do uso de técnicas de Tolerância a Falhas como: uso de linhas redundantes no barramento paralelo, uso da rede em anel ponto-a-ponto bidirecional, e uso de procedimentos de teste,

isolamento, e reintegração de elementos de processamento.

Já o software do CPER (parte de um Núcleo em um Sistema Operacional Distribuído) teve como objetivo configurar o CPER de tal forma que ele opere como uma Máquina a Fluxo de Dados Dinâmica (MFDD), através do gerenciamento de todo o processo de comunicação e sincronização no CPER; gerenciamento de memória do sistema; tratamento de falhas; funções de carregamento; manipulação de Entrada/Saída; e reconfiguração do sistema.

Na programação a fluxo de dados, foram propostas várias estruturas, ("new tag manager", "new iteration generators", "new tag destructors") para permitir fluxo de dados dinâmico.

Com esta estrutura de hardware e de software podemos concluir que o CPER se mostra uma Máquina a Fluxo de Dados Dinâmica (MFDD) que explora ao máximo o paralelismo natural da programação a fluxo de dados, principalmente se fizermos um mapeamento (operação/Elemento de Processamento) adequado explorando ao máximo as características de granularidade fina, média e grossa.

C A P Í T U L O 5

TOLERÂNCIA A FALHAS EM PROGRAMAS A FLUXO DE DADOS NA MFDD

5.1 - INTRODUÇÃO

Mostramos no capítulo anterior o modelo de programação a fluxo de dados sobre o CPER que permite a execução concorrente de operações sobre múltiplos processadores, reduzindo significativamente o tempo de execução de programas que normalmente são executados de maneira sequencial mas que possuem um alto grau de paralelismo interno. [SRINI, 1985].

Essa vantagem e os resultados da computação podem ser perdidos se um processador falhar. Por isso, um aspecto importante em sistemas concorrentes é a Tolerância a Falhas, que permite uma reconfiguração do sistema, em decorrência da falha, causando uma perda de potencialidade mas evitando a queda total do sistema.

Vem sendo propostas várias arquiteturas a fluxo de dados [GURD et al, 1985], [ITO, 1986], [SHIMADA, 1986], [SATO, 1992], mas poucas com características de Tolerância a Falhas [SRINI,1985], o que passaremos a discutir em seguida.

5.2 - TOLERÂNCIA A FALHAS EM ARQUITETURAS A FLUXO DE DADOS IMPLEMENTADOS ATRAVÉS DE SISTEMAS MULTIPROCESSADORES

Para que um sistema multiprocessador suporte o modelo a fluxo de dados com reatribuição de processador em decorrência de falhas [RANDEL, 1983], [SRINI, 1985], algumas questões devem ser levantadas:

1) As operações e dados em um processador que falha têm que ser reatribuídas para outro processador. Isto requer uma área de armazenagem auxiliar para essas informações. Onde se localiza essa área e qual deverá ser a sua capacidade?

2) O que acontece quando uma operação dispara e ocorre uma falha de hardware no processador?

3) O que acontece com a área de armazenagem auxiliar, utilizada para reatribuição, quando ocorre uma falha de hardware nestas localidades?

4) Como são notificados os outros processadores para pararem de enviar dados para o processador que falhou?

Na tentativa de solucionar essas questões, pode-se classificar Tolerância a Falhas através de ações de curto, médio

ou longo prazo [MARQUES et al, 1988]. Ações de curto prazo são ações tomadas instantaneamente a nível do hardware como, por exemplo, a mudança no tamanho da palavra de dados a ser transmitida, ou ainda a substituição de um banco de memória por outro já ativo e previamente definido. Já as ações de médio prazo são aquelas que necessitam de alguma intervenção do software a nível do Núcleo de um Sistema Operacional Distribuído, que demanda uma série de ajustes menores para alguma reconfiguração do sistema. Finalmente, ações de longo prazo são ações que demandam providências a nível do sistema operacional para uma reconfiguração mais profunda. Assim, quanto maior for o tempo gasto para superar a falha, melhor será esta superação. As atividades de curto prazo, no entanto, são extremamente necessárias, uma vez que são elas que garantem o funcionamento do sistema, na presença de falhas, enquanto as ações de médio e longo prazo são tomadas.

5.2.1 - Verificação da Computação e Diagnose de Falhas

A diagnose de falhas em modelos de computação assíncrona, como é o caso do modelo a fluxo de dados, é crítica, pela grande quantidade de espaço e tempo necessários para fazer todo a verificação da computação.

Em um único processador, pode-se preservar o estado do sistema, bastando para isso armazenar contador de programa, registradores chaves, e registradores de uso geral, como elementos de verificação da computação. Em um sistema paralelo, contendo algumas centenas de processadores, salvar o estado de

todos os processadores é praticamente impossível, e se a computação não puder ser verificada, toda vez que um ou um conjunto de processadores falhar, a computação terá de ser reiniciada.

No caso específico da verificação da computação, para o modelo a fluxo de dados, uma forma conveniente é verificar cada operação que dispara em cada processador.

A área de armazenagem das informações para reatribuição, em decorrência de uma falha, pode estar nos elementos vizinhos àquele que falhou, em um determinado caminho de dependência num grafo a fluxo de dados. Desta maneira, não há necessidade de salvar o conteúdo dos registradores de um determinado processador. Se ocorrer uma falha, os dados daquele processador são reatribuidos para outro processador, uma vez que esses dados, estarão armazenados nos vizinhos, implicando em uma degradação elegante para o sistema.

5.3 - A TOLERÂNCIA A FALHAS NA MFDD

O tratamento de falhas na MFDD é realizado através de ações específicas de curto, médio, e longo prazo, pela verificação a nível de hardware (barramento e processador), bem como a nível de software (núcleo do sistema) na execução das operações a fluxo de dados.

5.3.1 - Tolerância a Falhas no Hardware do CPER

A Tolerância a Falhas no hardware do CPER, como já

vimos, é obtida de algumas maneiras: pelo uso de linhas redundantes no barramento paralelo; pelo uso de um anel duplo bidirecional; por procedimentos de testes através dos EPs supervisores, e pela substituição de um processador por outros [MARQUES et al, 1988].

5.3.2 - Tolerância a Falhas na Programação a Fluxo de Dados da MFDD

A execução de programas ocorrerá conforme descrito na seção 4.3.1, através de um conjunto de operações armazenadas em cada EP na MFDD. À medida que essas operações vão sendo executadas, pacotes são enviados e recebidos pelos Elementos de Processamento que contém as operações, até que não haja mais operações para serem executadas [SILVA e KIRNER, 1990].

A área de armazenagem das operações e dos dados de um determinado EP para eventual reatribuição, está dividida entre o Hospedeiro e os EPs vizinhos àquele em um caminho de dependência num grafo a fluxo de dados, respectivamente.

A verificação de falhas na programação a fluxo de dados é feita através de um protocolo de comunicação entre os EPs [SILVA e KIRNER, 1990], [GRECO e SILVA, 1990a].

O não envio de dados de um EP para outro é tratado a nível de um núcleo do sistema, detectado através do protocolo, pelo estouro de um "timeout" específico, que ocorre quando um EP, recebendo um dado, ativa uma operação, e fica aguardando outros dados, que deveriam disparar aquela operação mas que não chegam, sugerindo a falha de um EP antecessor, ou da via de comunicação

entre esses EPs.

Quando falamos em comunicação entre as operações a fluxo de dados, estamos falando de comunicação em um nível acima do hardware do CPER, que possui barramento e "gateway".

Dependendo de como foram carregadas as operações em cada EP no CPER, poderemos ter comunicação entre operações a fluxo de dados dentro de um mesmo EP (quando existem mais que uma operação alocada em um mesmo EP - granularidade média ou grossa), entre EPs em um mesmo "CLUSTER" (operações alocadas em EPs de um mesmo "CLUSTER" - granularidade fina, média, ou grossa), e entre EPs em "CLUSTERS" diferentes (operações que foram alocadas em EP/"CLUSTER" diferentes - granularidade fina, média ou grossa). Por isso a comunicação está um nível acima do hardware. Ela é efetuada entre as operações a fluxo de dados. No entanto no nível de hardware, o barramento deverá ser utilizado, e como já existe uma Tolerância a Falhas nesse nível e até mesmo através de EPs que supervisionam outros EPs, o protocolo para a programação a fluxo de dados foi proposto assumindo essa Tolerância a Falhas do hardware.

5.3.2.1 - O Protocolo Tolerância a Falhas

No CPER, em cada EP existe um "buffer" de entrada e um "buffer" de saída.

Se assumirmos uma operação a fluxo de dados por EP (granularidade fina), os "buffers" vão ser diretamente ligados às linhas de entrada e saída das operações. Caso haja mais que uma operação por EP, esses "buffers" vão ser distribuídos entre as

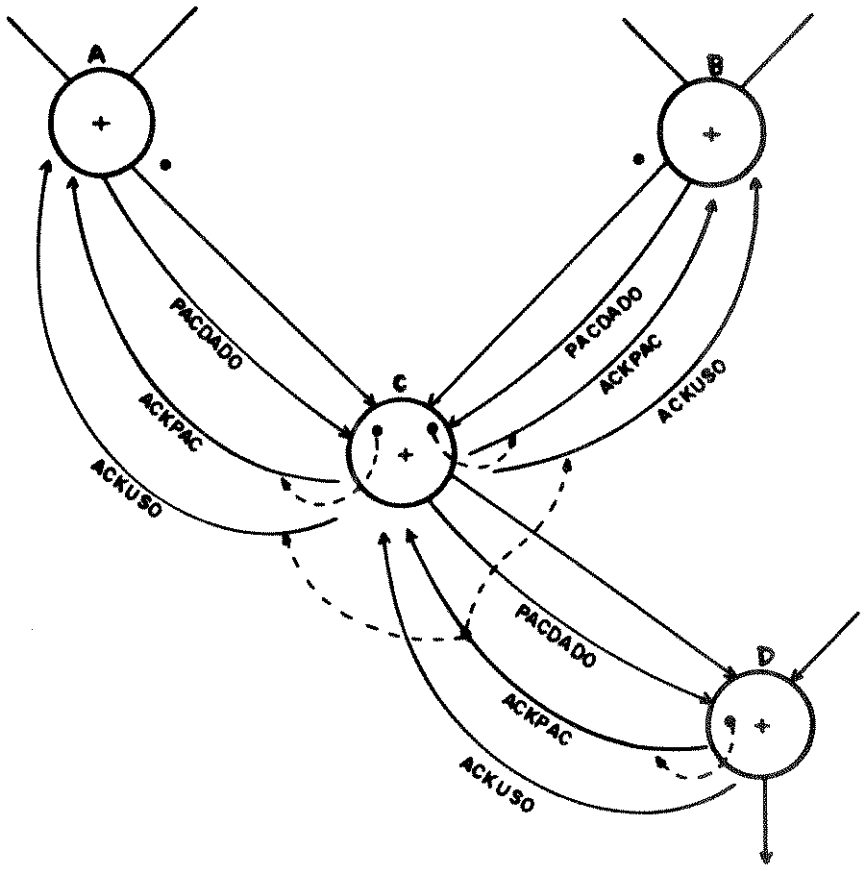


FIGURA 5.1 - PROTOCOLO PARA TOLERÂNCIA A FALHAS NA PROGRAMAÇÃO A FLUXO DE DADOS NO CPER.

operações, mas de alguma forma, a nível lógico, poderemos definir um "buffer" para cada linha de entrada e saída.

O Protocolo de Tolerância a Falhas é descrito na Fig.5.1.

Quando um EP-A manda um pacote de dados para um EP-C, em EP-A é disparado um "timeout" de reconhecimento de pacote, e o pacote enviado permanece armazenado em EP-A.

Chegando um reconhecimento de pacote pelo EP-C, o EP-A dispara um novo "timeout" para reconhecimento do uso daquele primeiro pacote enviado. Enquanto isso o pacote continua

armazenado em EP-A.

Quando o pacote enviado ao EP-C é utilizado, no caso o que gerou o pacote para um EP-D, em decorrência da execução de uma operação, o EP-C devolve um pacote de reconhecimento do uso, um para cada linha de onde ele recebeu pacotes.

Quando o pacote de reconhecimento do uso chega à fonte EP-A, então o pacote que permanecia armazenado no EP-A é descartado.

A não chegada de um reconhecimento de pacote ou de um reconhecimento do uso do pacote, faz com que o EP-A solicite de um EP supervisor naquele "CLUSTER", se aquele EP que não respondeu durante o "timeout" está com algum problema. Se o problema é detectado, então é acionado o Hospedeiro, que guarda um mapa de todos os EPs/operações no CPER, de tal forma que ele faça uma reconfiguração para que todos os EPs ligados àquele que falhou, se liguem a outro EP substituto, e os dados que vinham sendo mantidos em "buffers" sejam agora transferidos para esse novo EP. Caso contrário um novo "timeout" é disparado em EP-A, aguardando nova espera por reconhecimento. Isto assegura o assincronísmo na programação a fluxo de dados, e também permite verificar eventual falha em algum EP ou barramento no CPER.

Um problema ocorre quando duas operações estão interligados em um mesmo EP. Significa que se ocorrer uma falha naquele EP, todos os dados da linha de comunicação entre as duas operações, internas àquele EP, serão perdidos. O mesmo ocorre quando existe um "loop" inteiro dentro de um mesmo EP. Destacamos então três casos distintos para a elaboração do processo Tolerância a Falhas, conforme descritos nas Fig.5.2, Fig.5.3, e

Fig. 5.4, respectivamente.

Analisando os três casos, vemos que no primeiro caso, a aplicação do protocolo é imediata, em se tratando da alocação de apenas uma operação para cada EP.

No segundo e terceiro caso, vemos a alocação de mais que uma operação para cada EP, e particularmente no terceiro caso, existe um "loop" fechado dentro de um EP. Para esses casos, manter os dados internos intactos, e definir os "timeouts" para cada operação, apontam soluções complexas.

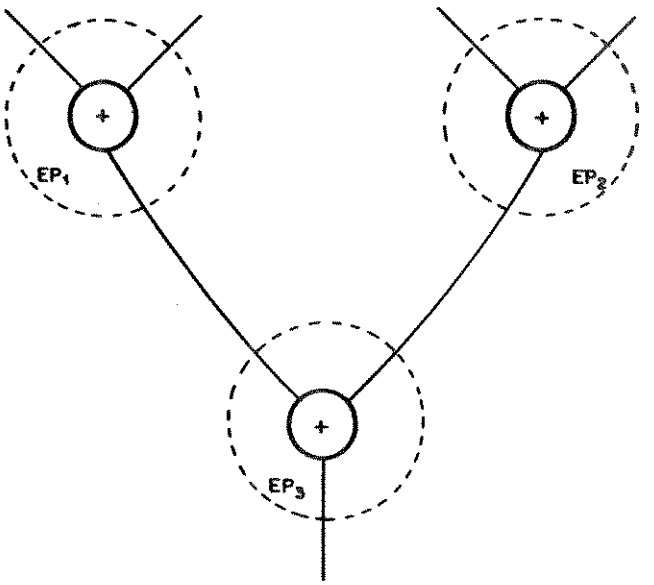


FIGURA 5.2 - UM PRIMEIRO CASO PARA PROCESSO TOLERÂNCIA A FALHAS.

Considerando a Tolerância a Falhas no hardware do CPER, optamos por:

a) Definir um "timeout" padrão (ex, o maior tempo de execução de uma operação a fluxo de dados e transmissão em "cluster" diferentes)

b) À cada estouro de "timeout" (conforme o protocolo T.F.) o EP que detectou o estouro, contesta Hospedeiro ou EP supervisor ligado ao EP destino para questionar se o EP destino está vivo (se estiver, novo "timeout", senão acionar reconfiguração), esquema conhecido como "Watchdog".

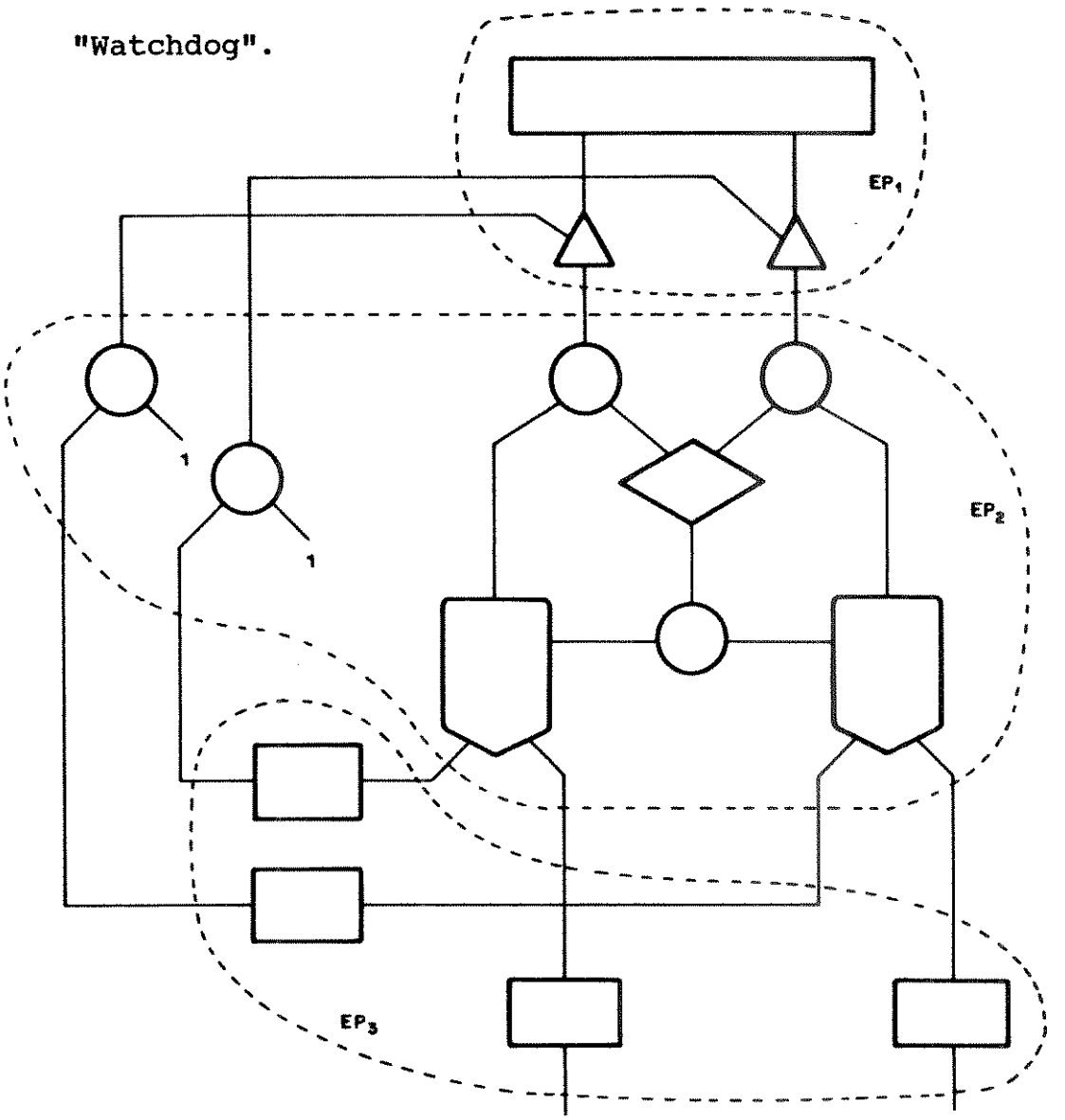


FIGURA 5.3 - UM SEGUNDO CASO PARA PROCESSO TOLERÂNCIA A FALHAS.

c) O EP de verificação deverá também ter condições de avaliar sobre "loops" infinitos, com o uso de variáveis para essa verificação, podendo definir se o EP sendo questionado está ou não com problemas.

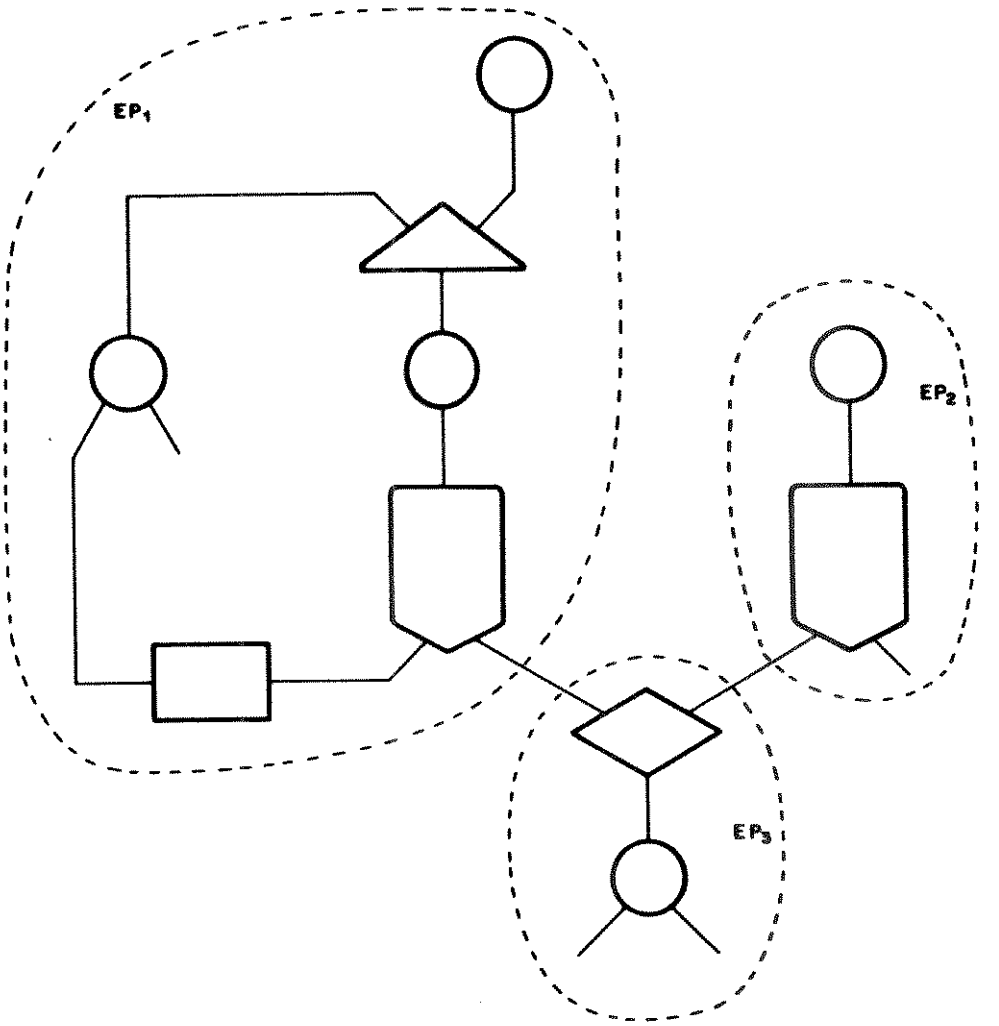


FIGURA 5.4 - UM TERCEIRO CASO PARA PROCESSO TOLERÂNCIA A FALHAS.

d) A questão dos pacotes orfãos, como descrito na Fig.5.5, EP-2 manda pacote para EP-3 que devolve reconhecimento do pacote. EP-2 devolve reconhecimento do uso para EP-1 mas que não chega,portanto EP-3 tem um pacote orfão. EP-1 vai notar que EP-2 morreu e portanto quando isso for notificado é perguntado ao EP-3 se ele recebeu algo, se recebeu há apenas uma simples reconfiguração dos dados, e das operações do EP-2 para outro senão, há uma reconfiguração dos dados, das operações e de um novo processamento para gerar dados para o EP-3.

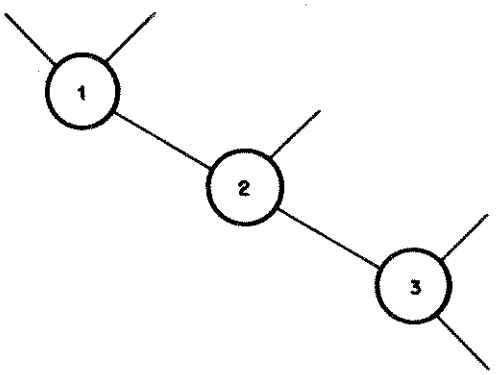


FIGURA 5.5 - O CASO DOS PACOTES ORFÃOS.

Soluções para problemas como pacotes orfãos, e várias operações por EP, serão estudados futuramente, nos restringiremos a implementar o processo Tolerância a Falhas na relação uma

operação por EP (granularidade fina), através do uso de um "timeout" padrão para o reconhecimento de pacotes.

Na Fig.5.6 mostramos um diagrama de estados do processo de recepção de dados em um EP tendo como base a execução do protocolo de Tolerância a Falhas. Na Fig.5.7 mostramos o diagrama de estado de um EP emissor.

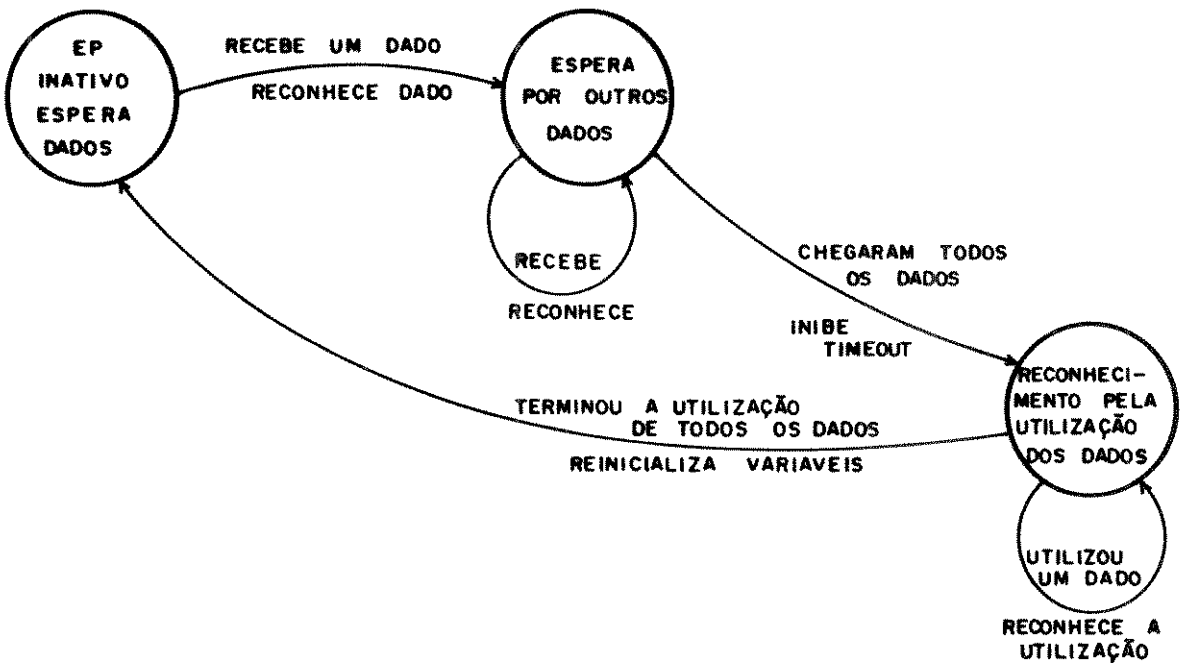


FIGURA 5.6 - DIAGRAMA DE ESTADOS DE UM EP RECEPTOR TOLERANTE A FALHAS.

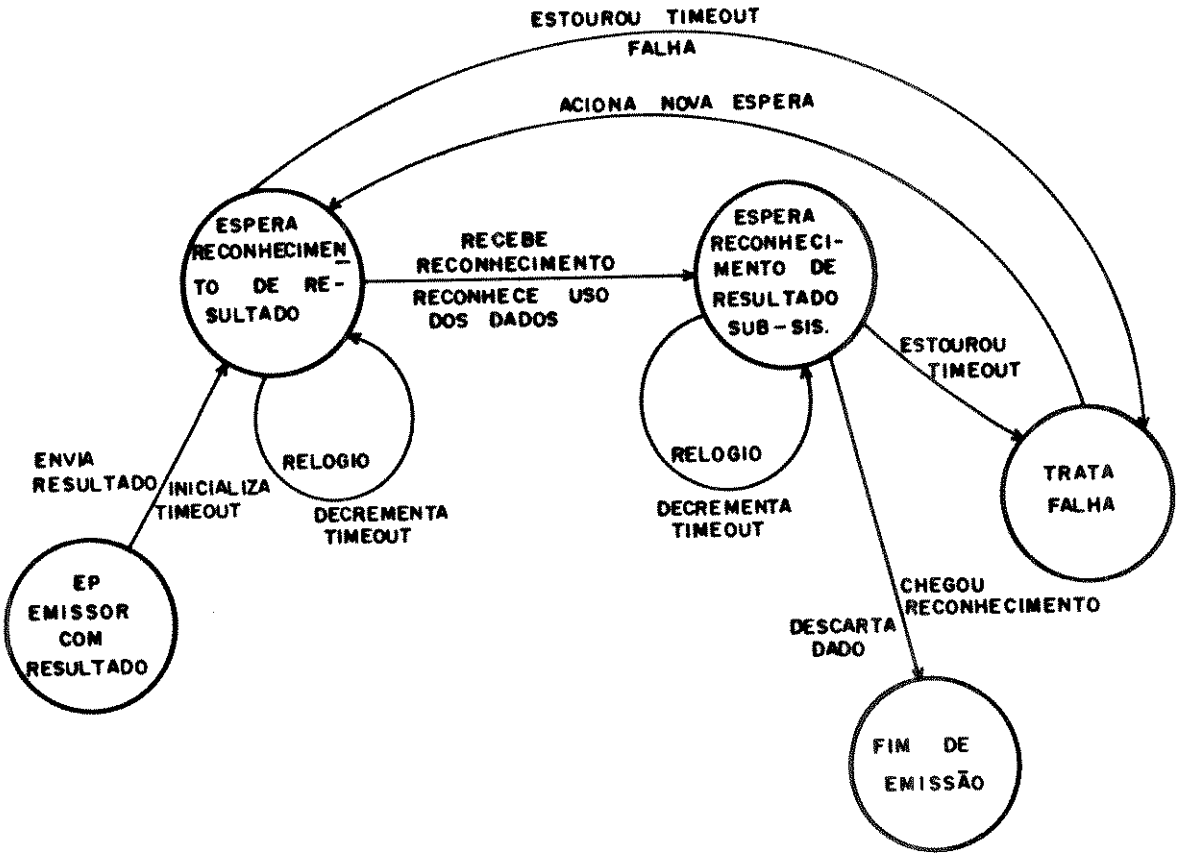


FIGURA 5.7 - DIAGRAMA DE ESTADO DE UM EP EMISSOR TOLERANTE A FALHAS.

5.3.2.2 - O "Gateway" Tolerante a Falhas

Na atribuição das operações a fluxo de dados no CPER pode ocorrer de serem alocadas operações em EPs localizados em um único "cluster" ou em "clusters" diferentes.

Quando os EPs estão localizados em um único "cluster" o protocolo para Tolerância a Falhas é o mesmo descrito na seção anterior. Entretanto, quando as operações são alocadas em diferentes "clusters", é necessário utilizar o "gateway". (GB na Fig.4.1).

O "gateway" possui uma estrutura similar ao Elemento de Processamento (EP) mas sua função básica é somente retransmitir informações de um "cluster" para o outro.

Em geral os "gateways" gerenciam filas de transmissão e recepção entre "clusters". Quando o "gateway" falha, todas as informações são perdidas, incluindo os pacotes do protocolo de Tolerância a Falhas. Isto pode causar o "timeout" de todos os elementos esperando por reconhecimento. Nestas condições, é impossível recuperar as informações perdidas. A solução tradicional para este caso é manter um "gateway" redundante. Mas esta solução pode resultar em um custo proibitivo.

Foi proposto então uma solução de software para resolver este problema.

A idéia básica é considerar o "gateway", que está posicionado no barramento principal do CPER, interligando os vários "clusters", como um Elemento de Processamento executando operações a fluxo de dados como todos os outros Elementos de Processamentos.

O "gateway" tolerante a falhas (FTG) pode ser definido então como um Elemento de Processamento executando operações (FTG) no programa a fluxo de dados, a ser inseridas entre as operações que estão localizadas em "clusters" diferentes.

Na Fig.5.8 mostramos um exemplo de programa que utiliza as operações (FTG) para representar o "gateway" tolerante a falhas.

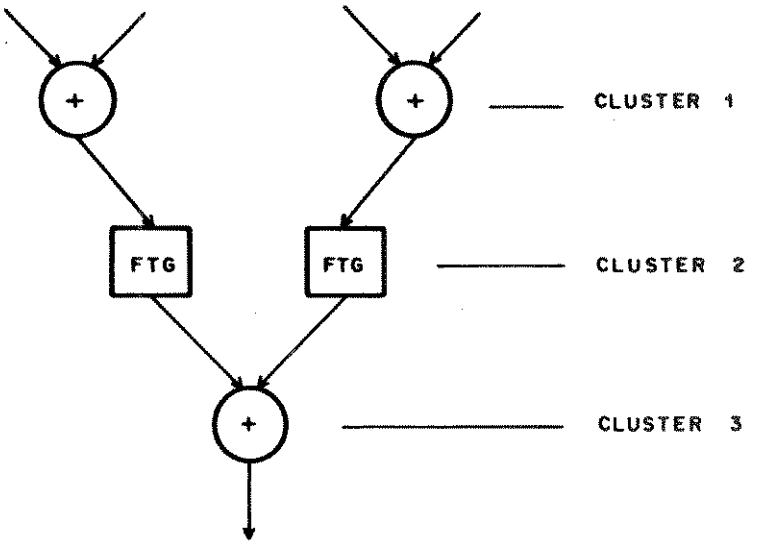


FIGURA 5.8 - EXEMPLO DE UM PROGRAMA QUE UTILIZA OPERAÇÕES FTG.

O mesmo protocolo para Tolerância a Falhas usado para outros Elementos de Processamento, é aplicado ao "gateway". No caso de algum falhar, o protocolo é capaz de providenciar a reconfiguração, isolando aquele que falhou.

Desta forma não é dado nenhum tratamento especial ao

"gateway", apenas que na programação a fluxo de dados deverão ser incluidos no programa as instruções (FTG) quando uma operação está ligada à outra em EP/CLUS diferentes.

Essas operações são apenas copiadoras de dados, de tal forma que o que for recebido será reenviado com a mesma estrutura de "tags" e dados.

5.4 - CONCLUSÃO

Mostramos neste capítulo que um aspecto importante em sistemas concorrentes é a Tolerância a Falhas, que permite uma reconfiguração do sistema, em decorrência da falha, causando uma perda de potencialidade mas evitando a queda total do sistema.

No caso do Fluxo de Dados em uma máquina multiprocessador, vimos que as operações e dados em um processador que falha têm de ser reatribuídas para outro processador, e classificamos Tolerância a Falhas através de ações de curto, médio ou longo prazo.

Mostramos também que a diagnose de falhas em modelos de computação assíncrona, como é o caso do modelo a fluxo de dados, é crítica, pela grande quantidade de espaço e tempo necessários para fazer toda a verificação da computação, e que no caso específico da verificação da computação, uma forma conveniente é verificar cada operação que dispara em cada processador.

Foi proposto então a verificação de falhas na programação a Fluxo de Dados no CPER, através de um protocolo de comunicação entre os EPs. Definimos um "timeout" padrão a ser utilizado no protocolo (ex, o maior tempo de execução de uma

operação a fluxo de dados e transmissão em "cluster" diferentes) de tal forma que à cada estouro de "timeout" (conforme o protocolo T.F.) o EP que detectou o estouro, contesta Hospedeiro ou EP supervisor ligado ao EP destino para questionar se o EP destino está vivo (se estiver novo "timeout" senão acionar reconfiguração), esquema conhecido como "Watchdog".

Foi proposto também o "gateway" tolerante a falhas (FTG), que pode ser definido como um Elemento de Processamento executando operações (FTG) no programa a fluxo de dados, a ser inseridas entre as operações que estão localizadas em "clusters" diferentes. Essa proposta de operações "gateways" é uma solução de software para o "gateways" tradicional com redundância.

Podemos concluir que a Tolerância a Falhas na programação a Fluxo de Dados da MFDD , através do protocolo de comunicação e do "gateways" implementado através de operações (FTG), torna o CPER como uma máquina de vanguarda no que se refere a programação a Fluxo de Dados Tolerante a Falhas.

C A P Í T U L O 6

O SIMULADOR DE PROGRAMAS A FLUXO DE DADOS PARA A MFDD (FDsim) Fluxo de Dados simulador.

6.1 - INTRODUÇÃO

O FDsim teve como objetivo, validar a estrutura dos programas a fluxo de dados tolerante a falhas na MFDD, e que deverá ser utilizado como um laboratório para o desenvolvimento do software de base do CPER para que este opere como a MFDD [SILVA e KIRNER, 1989].

Além de validar as estruturas dos programas a fluxo de dados, o FDsim gera resultados de desempenho da MFDD para alguns programas que foram executados através do simulador.

Em seguida faremos uma descrição da estrutura da MFDD que foi simulada, para em seguida descrever o FDsim propriamente dito.

6.2 - A MÁQUINA A FLUXO DE DADOS DINÂMICA (MFDD)

Como vimos a MFDD executa programas a fluxo de dados sobre o hardware do CPER.

O CPER vem sendo proposto com: "transputer", a serem utilizados como Elementos de Processamento; um barramento de 32 ou 64 linhas de dados, além das linhas do protocolo a nível de hardware; um sistema de recepção de dados nos Elementos de Processamento baseado em "hashing" de memória, isto para a execução a fluxo de dados; e um Hospedeiro tipo IBM/PC 386.

6.2.1 - Parâmetros do CPER que foram considerados no SIMULADOR FDsim.

Em se tratando de um Elemento de Processamento baseado no "transputer" [INMOS, 1985], que tem um desempenho na ordem de 20 MIPS, podemos considerar uma taxa média de 50 ns para executar uma instrução.

Assumindo uma média de 10 instruções de máquina para a execução de uma operação a fluxo de dados, teremos um tempo médio de processamento, em cada elemento executando uma operação a fluxo de dados, na ordem de 500 ns.

A comunicação através do barramento no CPER é tal que: o elemento que deseja transmitir, solicita o barramento. Caso o barramento esteja desocupado, é efetuado a transferência de informação, caso contrário, o elemento espera até que lhe seja liberado o barramento.

A liberação do barramento ocorre dentro de um esquema de prioridade por posição (em uma estrutura em forma de anel), da direita para a esquerda no barramento, de tal forma que quando alguém solicita e o barramento está ocupado, ele fica aguardando

a liberação. Se alguém mais a direita deste elemento também solicita o barramento ele é que ganhará a permissão para a transmissão [GONÇALVES, 1991].

Quando um elemento ganha o barramento, ele pode transmitir suas informações, só que enquanto isso, a permissão já é passada para o próximo elemento mais à esquerda desejando transmitir, e isso é garantido por um hardware específico, independente da transmissão de dados; é portanto transparente a disputa.

Em estudos foram comprovados a taxa de 61 ns o tempo para transmitir um pacote de dados de 32 ou 64 bits, o que nos leva a uma taxa de 132 Mbytes/seg no barramento [GONÇALVES, 1991].

Os pacotes a fluxo de dados a serem transmitidos pelo barramento irão conter 600 bits (distribuídos entre "tags", dados, e endereços) o que nos leva a um esquema de transmissão de 10 pacotes pelo barramento, para se transmitir um pacote fluxo de dados.

Já a recepção de dados pelos Elementos de Processamento, mais especificamente os pacotes a fluxo de dados, tem uma complicação adicional que é encontrar os parceiros dos dados em uma operação onde existem mais que um arco de entrada. É o caso de uma soma por exemplo. Isso foi resolvido por um esquema de "hashing" de memória garantindo a recepção em 70 ns [MARQUES e KIRNER, 1992a], [MARQUES e KIRNER, 1992b] para cada dado.

Entre recepção e processamento de uma operação tipo soma, por exemplo, perderemos, 70ns de "hashing", um para cada dado de entrada e mais 500 ns de processamento, que foram

levados em conta na simulação, além dos outros parâmetros citados acima.

6.3 - O SIMULADOR FDsim

O FDsim está implementado em Pascal versão 6.0 e roda sobre uma máquina da linha IBM/PC.

O simulador foi implementado através de "UNITs" do Pascal, cada uma responsável pela implementação de um módulo no FDsim, são eles: O módulo EDITOR, o módulo ANALISA, o módulo RELÓGIO, o módulo RECEPÇÃO, o módulo TOLERÂNCIA A FALHAS, o módulo BARRAMENTO, o módulo EXECUÇÃO, e o módulo PRINCIPAL.

No Apêndice E relacionamos todas as "Units", seus tamanhos, o tamanho do programa principal e o tamanho do programa FDsim executável.

6.3.1 - Características gerais do FDsim

O FDsim, através de seus módulos, permite que o usuário escreva um programa a fluxo de dados, segundo as operações propostas para a MFDD, edite este programa segundo uma análise prévia, e solicite a execução do programa.

Na execução, o simulador solicita do usuário os dados de entrada, que poderão ser fornecidos na forma "pipeline" de dados (uma rajada de dados de entrada), e que vão sendo alocados segundo a proposta para a MFDD (geração das filas de ativações e iterações), para em seguida dar início à execução propriamente dita, quando é ativado o módulo relógio que praticamente comanda

todas as ações a partir deste ponto.

A partir do acionamento do módulo relógio, os dados gerados pelas execuções das operações a fluxo de dados, vão fluindo pela MFDD, segundo características do hardware do CPER: recepção de dados por um Elemento de Processamento e consequente "hashing" deste dado em uma determinada operação; processamento destes dados quando todos estão presentes em uma operação; envio dos resultados desta operação através do barramento de dados; execução do processo de Tolerância a Falhas durante a execução

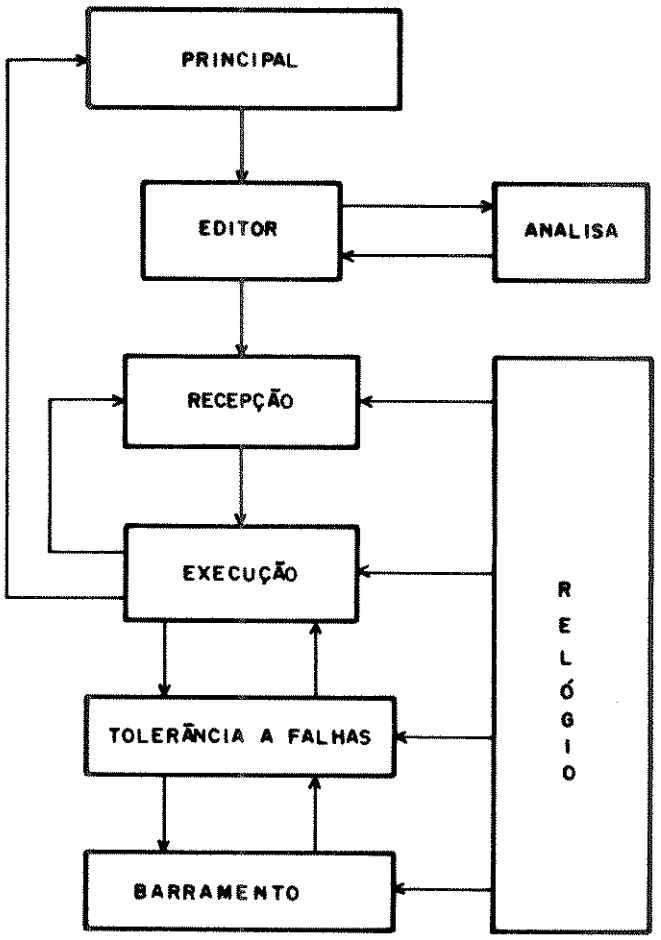


FIGURA 6.1 - DISPOSIÇÃO DOS MÓDULOS DO FDsim.

do programa a fluxo de dados; computação do tempo total de execução do programa em função do relógio central; e finalmente apresentação do resultado.

Na Fig. 6.1 está descrito a disposição dos módulos no FDsim e como eles se relacionam, e na Fig. 6.2 um diagrama de fluxo de como opera o FDsim.

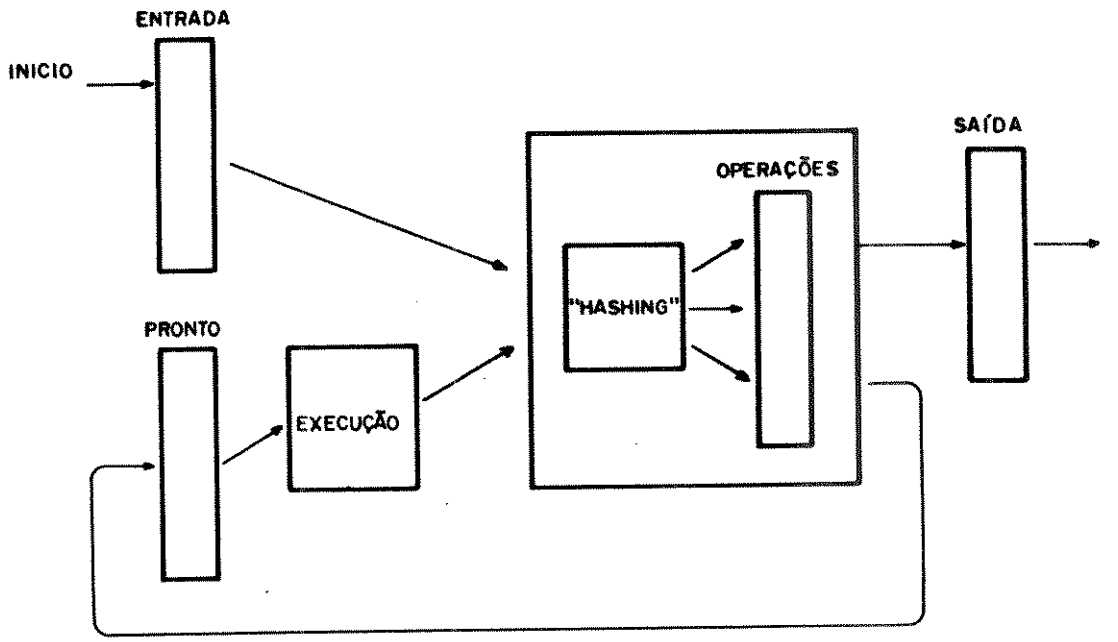


FIGURA 6.2 - DIAGRAMA DE FLUXO DA EXECUÇÃO DE PROGRAMAS NO FDsim.

Faremos em seguida a descrição de cada módulo do simulador, suas estruturas, parâmetros, e funcionamento.

6.3.2 - O Módulo EDITOR

O módulo Editor é responsável pela edição de programas a fluxo de dados que permite que um programa seja originalmente

escrito na forma gráfica e transferido ao computador através de comandos de recepção das operações e suas ligações com outras.

Toda a edição, execução e visualização de resultados dos programas são efetuados através de telas, todas implementadas dentro do módulo EDITOR.

Na descrição do programa a fluxo de dados, as operações são numeradas automaticamente, e para cada operação é mostrado a figura correspondente àquela operação. Dentro desta figura o usuário fornece informações como: arcos de entrada, arcos de saída, processador que esta operação deverá ser alocada, e "cluster" onde se encontra este processador.

Na edição é criado um "record" para cada operação descrita, que contém basicamente campos com toda a especificação da operação como: tipo da operação, especificação das linhas de entrada daquela operação, especificação das linhas de saída, alguns campos adicionais e alguns ponteiros. O conjunto desses "records" irá compor um arquivo que corresponderá ao programa a fluxo de dados editado, que ficará armazenado em disco.

6.3.3 - O Módulo ANALISA

Uma vez montado o arquivo do programa a fluxo de dados, é necessário fazer uma ligação entre todos os arcos das operações do programa e, em seguida, uma análise dessas ligações para ver se nenhum arco ficou sem ligação.

A análise e ligação dos arcos são feitas dentro do módulo ANALISA.

Essas tarefas são feitas a partir do arquivo do

programa a fluxo de dados, verificando-se qual arco de saída se ligará com qual arco de entrada, entre as operações do programa. Em cada coincidência, os "records" das duas operações cujos arcos coincidiram, são completados com as seguintes informações: quais operações estão se ligando; e em quais processadores/"clusters" está ligado aquele arco. Desta forma é possível, a partir de algum arco, saber quem está ligado com quem, tanto visto pelo arco de saída como pelo arco de entrada de uma operação.

A partir desta ligação e análise, é gerado um relatório que mostra todos os arcos que ficaram sem ligação. Caso existam edita-se o programa para corrigir essas falhas. Caso contrário, se todas as ligações foram efetuadas, o programa a fluxo de dados está pronto para ser executado.

6.3.4 - O Módulo RELÓGIO

Basicamente é neste módulo onde todas as ações são disparadas, é onde toda a estrutura de dados do FDsim é manipulada. Passamos então a descrever esta estrutura para em seguida descrever o módulo relógio propriamente dito.

6.3.4.1 - Estrutura de Dados do FDsim para implementar o CPER na MFDD.

Foi criado uma Matriz de ponteiros que é preenchida a partir de um arquivo de programa a fluxo de dados, em função dos "records" das operações. Essa matriz, denominada MCE, descreve o CPER, conforme descrito na Fig. 6.3.

Cada elemento da MCE é um ponteiro, que aponta o primeiro elemento de uma lista encadeada de operações naquele Elemento de Processamento (EP).

Podemos olhar a estrutura global da matriz MCE conforme descrito na Fig. 6.4.

Conforme Fig.6.3 e Fig. 6.4 vemos a necessidade de trabalhar com várias entidades simultaneamente, uma vez que se trata de vários elementos de hardware independentemente (enquanto máquina real).

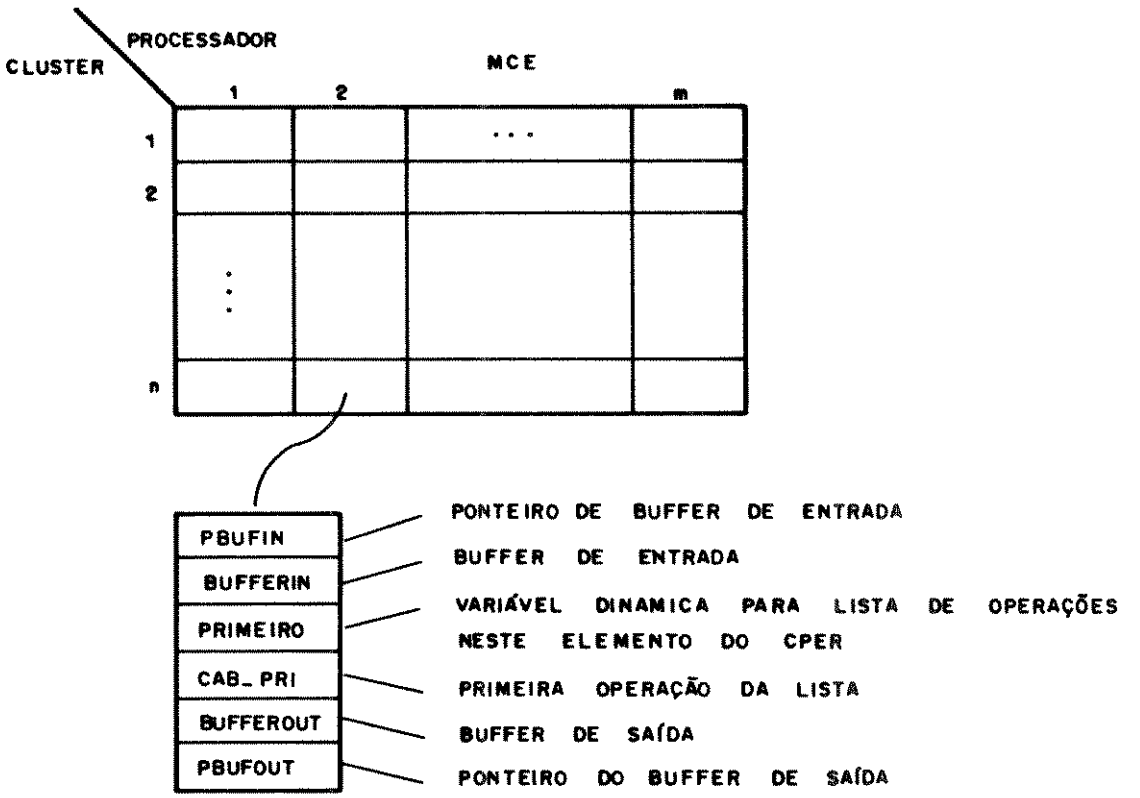
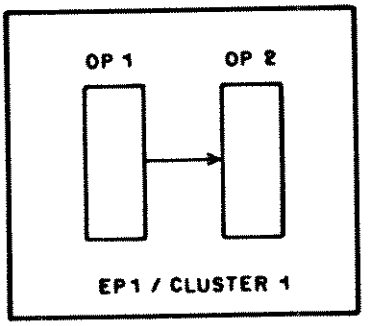
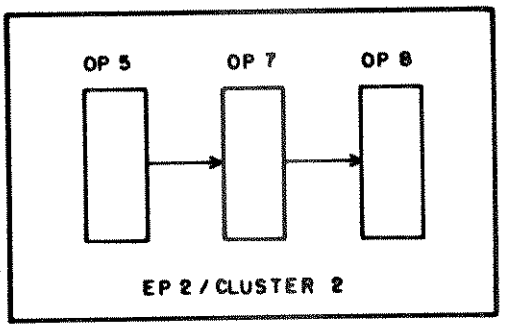


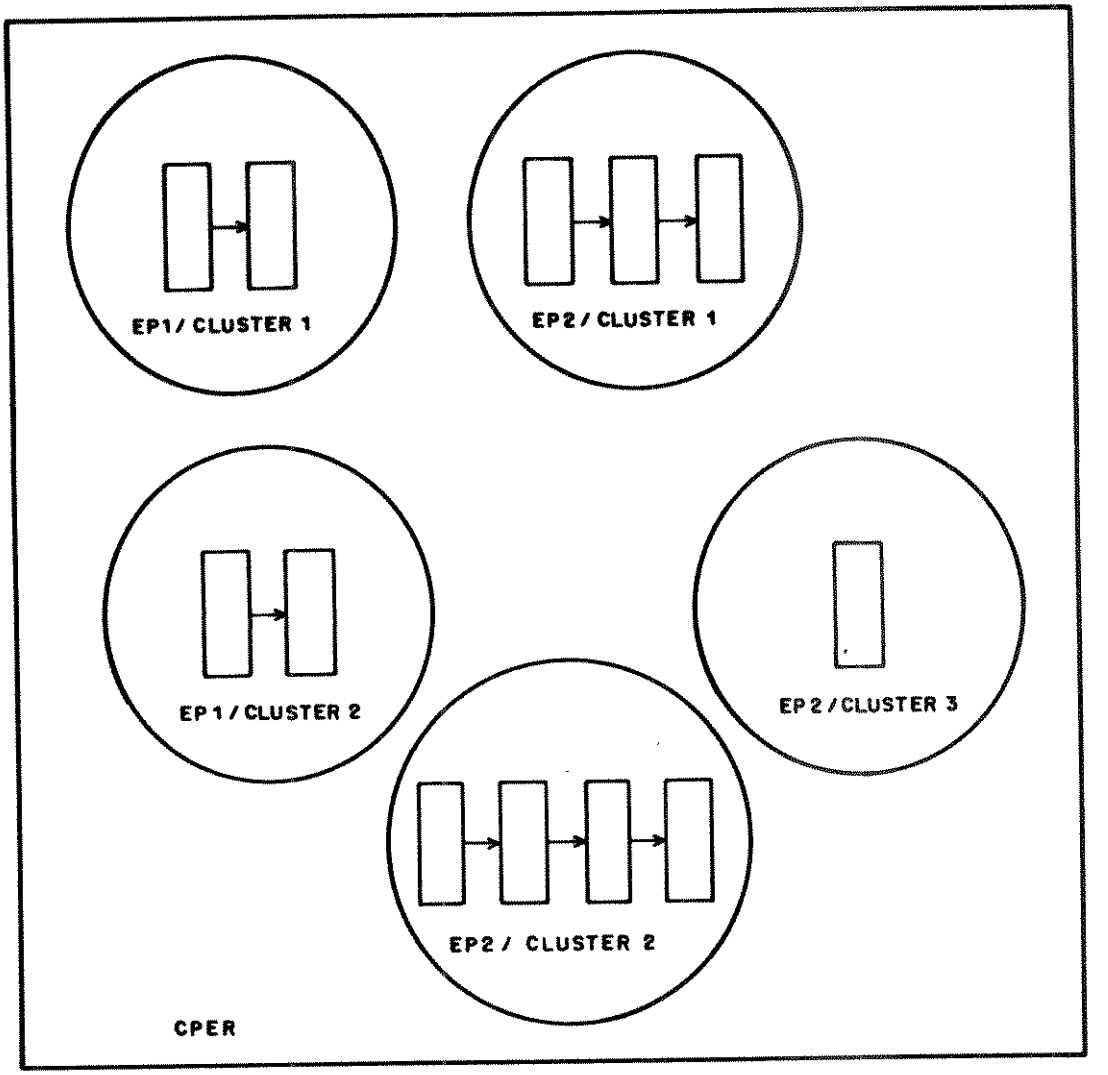
FIGURA 6.3 - ESTRUTURA DA MATRIZ CPER (MCE).



a) EP1/CLUSTER 1 COM 2 OPERAÇÕES



b) EP2/CLUSTER 2 COM 3 OPERAÇÕES



c) O CPER COM OPERAÇÕES DISTRIBUIDAS ENTRE OS EP's/CLUSTER CONFIGURADOS NA MATRIZ CPER (MCE)

FIGURA 6.4 - EXEMPLO DE DISTRIBUIÇÃO DAS OPERAÇÕES DISTRIBUÍDAS NA MATRIZ MCE.

Na execução, para os "records" das operações, filas de ativação, filas de iteração, e para algumas variáveis no FDsim, foram utilizadas variáveis com alocação dinâmica que nos possibilitou uma ocupação de memória altamente eficiente, e também em função da indefinição de espaço de memória necessário quando da execução de um programa a fluxo de dados com entradas na forma "pipeline".

6.3.4.2 - A Estrutura do RELÓGIO

Utilizamos o relógio por eventos, ou seja, todo evento pronto (recepção de dados, execução de operação, transmissão, e "timeout"), é assinalado na matriz de eventos (MEV) (Fig.6.5), para cada EP, com um tempo pré-estabelecido de ocorrência. Essa ordem dos eventos é proposital, pois como o EP é monoprocessador, só poderá executar uma coisa de cada vez, e vai se respeitar essa ordem, exceto "timeout" que é tratado por interrupção.

Na MEV, portanto, para cada EP é sinalizado qual evento está ocorrendo e qual será o tempo gasto para se executar esse evento.

O relógio faz uma varredura na matriz MEV e verifica qual evento possui o menor tempo entre todos os eventos que estão ocorrendo. Identificado qual ou quais eventos possuem o menor tempo, a execução é efetivada e o tempo transcorrido é descontado de todos os eventos que não possuiam o menor tempo, o que significa a ocorrência simultânea desses eventos.

Toda vez que termina um evento em um EP, se existir outro, ele é ativado e o tempo deste evento começa a ser descontado.

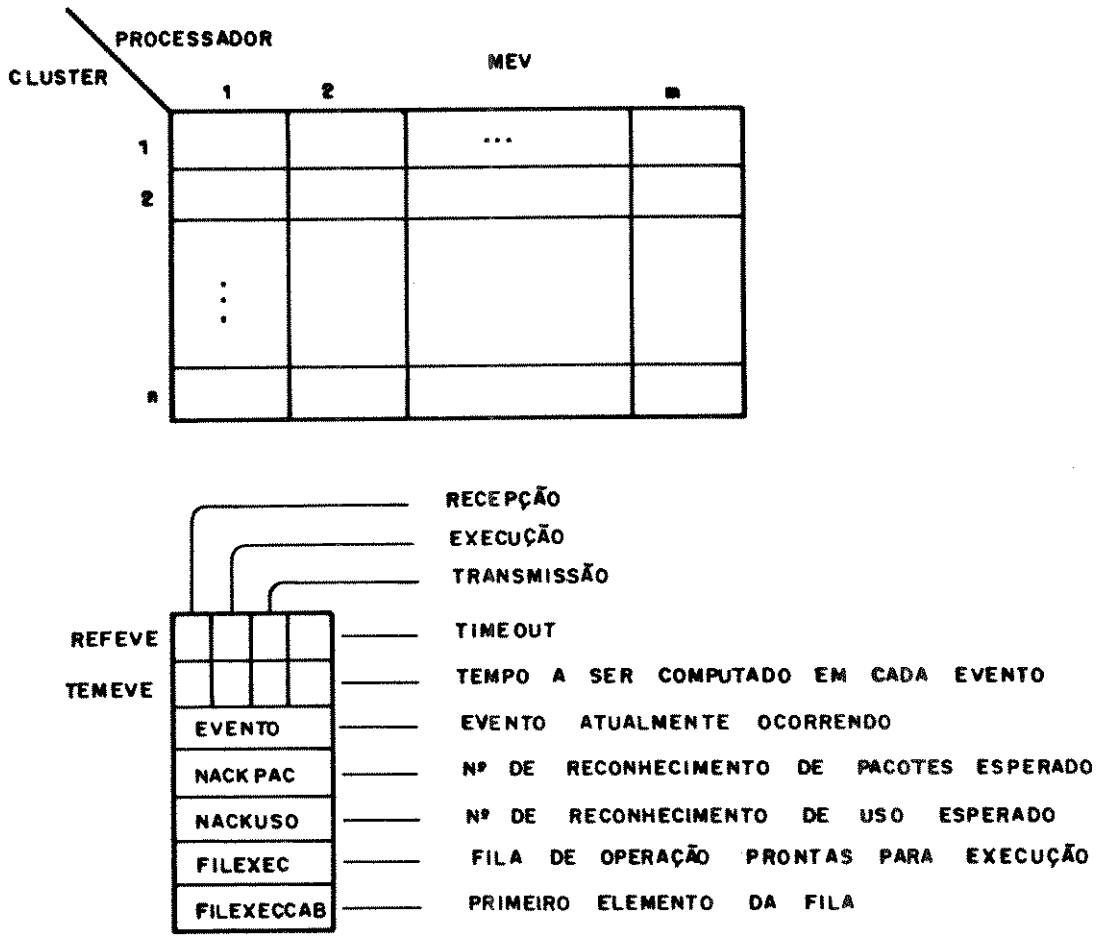


FIGURA 6.5 - ESTRUTURA DA MATRIZ DE EVENTOS NO CPER (MEV).

6.3.5 - O Módulo RECEPÇÃO

Partindo da MCE, que é gerada a partir do arquivo de Programas a Fluxo de Dados, a recepção de dados pelo processo EP se dará através da especificação na MEV, de que é Recepção naquele EP, e que portanto ele deve receber esse dado no momento oportuno.

Quando um EP/CLUS envia dados para outro EP/CLUS, esse dado é convertido em um pacote fluxo de dados e vai ficar armazenado em um "buffer" de recepção no EP/CLUS correspondente. Neste pacote deverá constar informações do tipo: o dado a ser utilizado, em qual operação esse dado deverá ser atribuido, qual ativação e em qual iteração ele está inserido.

No CPER, mais especificamente em cada EP, foi definido um "buffer" de recepção, capaz de receber um conjunto desses pacotes em paralelo com o processamento do EP.

A colocação de um pacote de informação no "buffer" de um EP/CLUS sinaliza na MEV o evento de recepção, caso ainda não tenha sido sinalizado, pois o "buffer" poderá já ter algum pacote.

O EP/CLUS com evento de recepção, deverá pegar todos os pacotes do "buffer", na ordem em que foram recebidas, criar as listas correspondentes (ativação e iteração), caso não existam, ou encontrar o lugar do dado nestas listas, e verificar quais instâncias da operação estão prontas para serem executadas, o que deverá ser sinalizado na MEV, especificando que aquele EP/CLUS tem uma instância da operação para ser executada.

Cabe observar que como o dado vem com todas as suas especificações, sabemos exatamente sobre qual operação estamos falando, qual sua ativação, qual seu nível de aninhamento e que iteração corresponde aquele dado. Em se tratando de uma recepção todo este processo é supervisionado pelo processo TF através do protocolo de Tolerância a Falhas.

Partindo do princípio que um dado chega sempre após o outro e nunca simultaneamente, por causa do acesso ao barramento, não haverá portanto duas instâncias de uma mesma operação prontas para a execução simultaneamente, mas sim uma sequência bem definida de quais instâncias da operação estão ficando prontas, sequência esta armazenada em uma lista de instâncias prontas na própria matriz MEV, através de ponteiros das variáveis dinâmicas das listas de ativação e iteração que determinam a instância para a execução, eliminando assim qualquer dificuldade sobre qual instância deverá ser executada em qual momento.

6.3.6 - O Módulo EXECUÇÃO

Para a execução de uma operação em um EP/CLUS, basta pegar a primeira referência de ponteiros na MEV, da instância a ser executada, e efetuar a execução. Os resultados deverão ser encaminhados para transmissão através da montagem de uma mensagem, que é colocada no "buffer" de saída daquele EP/CLUS, e em seguida, sinalizado na MEV que aquele EP/CLUS deverá transmitir dados.

6.3.7 - O Módulo BARRAMENTO

O processo barramento simula o modo de operação do barramento no CPER, que opera através de troca de mensagem, onde a prioridade circula entre os dispositivos num esquema "round-robin". O direito de ocupação do barramento é concedido através de um sinal de prioridade que circula simetricamente entre eles. A implementação deste esquema é feita usando-se registrador de deslocamento que, estando habilitado, circula a prioridade [GONÇALVES, 1991].

No CPER, para garantir acesso ao barramento, o dispositivo deverá manter-se com a prioridade bloqueando temporariamente a sua circulação. Após ter reservado o barramento para seu uso libera a circulação da prioridade.

Com este procedimento, de modo geral, a arbitragem para o próximo acesso ao barramento terá se encerrado (o pretendente mais próximo do emissor estará com o sinal de prioridade), antes mesmo de encerrar-se a transmissão anterior. Esta sobreposição dos ciclos de arbitragem aos ciclos de transmissão, diminui consideravelmente o "overhead" do sistema.

O processo barramento portanto executa esse mecanismo de acesso entre os Elemento de Processamento e o barramento. Esse processo é executado apenas pelo EP que ganhou o barramento.

Um EP ganha o barramento quando: ele solicitou transmissão, essa transmissão ficou pendente na Matriz de Eventos MEV, e na ocorrência do evento transmissão naquele EP/CLUS, é verificado se o barramento está disponível. Se disponível ele é

alocado e a transmissão é efetuada. Caso contrário, enquanto o barramento não está disponível, ele deverá ficar aguardando permissão, sinalizando que o deseja. Isto é feito através de um sinal em uma Matriz Barramento (MB), cuja estrutura é descrita na Fig. 6.6, tal que cada linha e coluna da matriz é um processador/"cluster" do CPER respectivamente, especificando se "QUERO" transmitir, e se o "TOKEN" está presente ou não, como elementos da matriz.

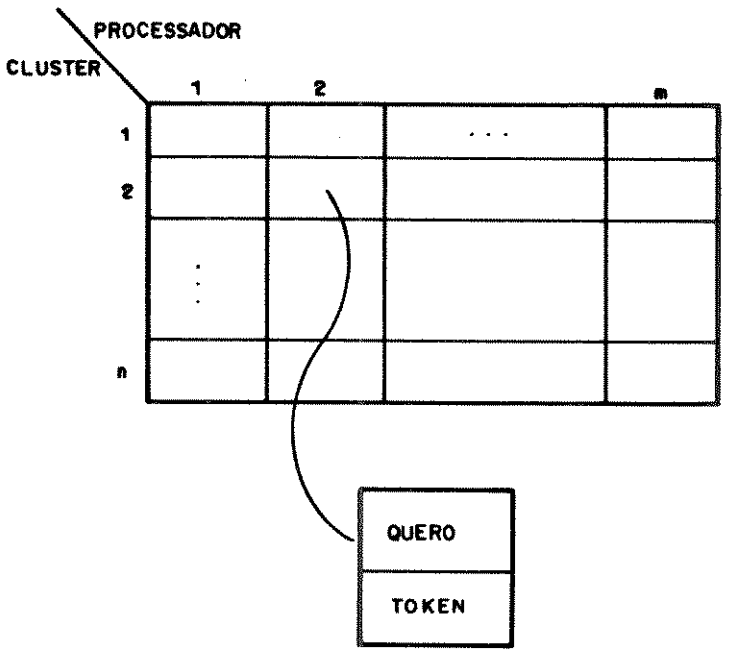


FIGURA 6.6 - ESTRUTURA DA MATRIZ BARRAMENTO NO CPER (MB).

O EP que deseja transmitir, a partir do momento em que MEV liberou transmissão naquele EP/CLUS, sinaliza QUERO em MB.

Se a permissão não está com ninguém, ele ganha e passa a executar o processo barramento, que faz a transmissão, além de passar a permissão para outro EP que deseja transmitir, naquele

"cluster".

Se o barramento está ocupado, o EP que deseja transmitir, fica preso no processo de transmissão na MEV, aguardando a liberação do barramento. Isto ocorre simplesmente sinalizando transmissão na MEV e permitindo que o tempo corra.

A liberação do barramento vai ocorrer quando um EP, que tem o barramento (e portanto está executando o processo barramento), utiliza o barramento e verifica se existe alguém querendo a permissão quando então é sinalizado "TOKEN" naquele EP daquele "cluster" específico.

Esse "TOKEN" fará com que aquele EP, quando da ocorrência de uma transmissão, ganhará o barramento, e portanto ele irá executar o processo barramento.

Na matriz evento, então, vão existir EPs esperando "token", e EPs transmitindo. Todos os EPs que querem transmitir, quando for possível transmitir, chamam o processo barramento que verifica na matriz barramento se ele pode pegar o barramento, senão sinaliza na MEV continuidade da transmissão e fica implicitamente à espera do "token".

6.3.8 - O Processo Tolerância a Falhas

O Processo Tolerância a Falhas tem como objetivo gerenciar toda a comunicação a nível de operações a fluxo de dados, através de um protocolo de comunicação, de forma a permitir Tolerância a Falhas durante a execução de um programa na MFDD.

6.3.8.1 - OS Procedimentos "SEND" E "RECEIVE"

Os procedimentos "SEND" e "RECEIVE" são os responsáveis por todo o processo de comunicação dentro da programação a fluxo de dados, inclusos no processo Tolerância a Falhas.

Quando um EP/CLUS vai transmitir um pacote de dados, ele aciona a rotina "Send" e passa para a rotina o pacote de dados a ser transmitido. A rotina "Send" pega o pacote de dados, atualiza "timeout" de reconhecimento de pacote (esse "timeout" vai correr em paralelo com o processamento do EP), solicita barramento (segundo descrição de operação do processo barramento), transmite e sinaliza "status" no EP de espera de reconhecimento.

Se ocorrer "timeout", a rotina "Send" é novamente acionada e faz consulta para verificar se o EP destino está operando normalmente. Se "sim", reatualiza "timeout", senão solicita reconfiguração.

Se chegar um "ackpac" (detectado na recepção de pacotes no "buffer" de entrada de um EP durante um processo de recepção, destacando aqui a existência de dois tipos de pacotes: o de controle e o de dados), a rotina "send" atualiza "timeout" de reconhecimento do uso do pacote ("ackuso"), e sinaliza "status" no EP de espera por ("ackuso").

Se ocorrer "timeout", agora para o uso do pacote, a rotina "Send" é novamente acionada e faz teste também para verificar se o EP destino está operando. Se "sim", reatualiza "timeout" senão, solicita reconfiguração.

Na recepção, (não aquela que ocorre em paralelo ao processamento do EP mas, aquela em que os dados do "buffer" vão ser alocados em filas ou servirão para controle), o EP faz a chamada da rotina "Receive" que recebe um pacote do "buffer", verifica se é pacote de controle (delibera sobre o controle) ou dados (atribui o dado segundo as listas de ativações e iterações, monta pacote de "ackpac", solicita barramento, transmite o pacote de "ackpac", e sinaliza estado do EP que deve devolver "ackuso").

Tão logo o dado tenha sido usado (é quando dados dispararam uma operação em um EP, a execução desta operação gera um resultado, que é transmitido para outros EPs na forma de um pacote de dados. O EP que recebe esse dado devolve um "ackpac", significando que aquele dado foi recebido e portanto, é possível responder aos EPs que enviaram dados para a operação que foi executada, que seus dados foram usados e já se encontram seguros no destino), a rotina "Receive" é portanto novamente acionada por aquele EP que executou a operação, que monta um pacote de "ackuso", solicita barramento, transmite, e atualiza status de EP e a rotina "Receive" é abandonada.

6.4 - CONCLUSÃO

Como vimos, o FDsim teve como objetivo, validar a estrutura dos programas a fluxo de dados tolerante a falhas na MFDD, e que deverá ser utilizado como um laboratório para o desenvolvimento do software de base do CPER para que este opere como uma MFDD, além de gerar resultados de desempenho.

Tendo como base o "Transputer" 20 MIPS, pudemos

considerar uma taxa média de 500 ns para executar uma operação a fluxo de dados.

Mostramos também que a comunicação através do barramento no CPER ocorrerá dentro de um esquema de prioridade por posição (em uma estrutura em forma de anel), da direita para a esquerda no barramento, de tal forma que quando alguém solicita, e o barramento está ocupado, ele fica aguardando a liberação. Se alguém mais a direita deste elemento também solicita o barramento, ele é quem ganhará a permissão para a transmissão.

Vimos que em estudos foram comprovados a taxa de 61 ns na transmissão de um pacote de dados de 32 ou 64 bits, o que nos leva a uma taxa de 132 Mbytes/seg no barramento, e que para mensagens a fluxo de dados com 600 bits (distribuídos entre "tags", dados e endereços), precisaremos de 10 pacotes a serem transmitidos pelo barramento.

Descrevemos também um esquema de "hashing" de memória para a recepção de dados, garantindo a recepção em 70ns para cada dado, o que numa soma por exemplo, perderemos 70 ns para receber cada dado e mais 500 ns para executá-la, que foram levados em conta na simulação, além dos outros parâmetros.

Como vimos o FDsim está implementado em Pascal versão 6.0 e roda através de uma máquina da linha IBM/PC. O simulador foi implementado através de "UNITs" do Pascal, cada uma responsável pela implementação de um módulo no FDsim: o módulo EDITOR, o módulo ANALISA, o módulo RELÓGIO, o módulo RECEPÇÃO, o módulo TOLERÂNCIA A FALHAS, o módulo BARRAMENTO, o módulo EXECUÇÃO e o módulo PRINCIPAL. É através destes módulos que o

usuário pode escrever um programa a fluxo de dados, segundo as operações propostas para a MFDD, editar este programa, e solicitar a sua execução.

O FDsim portanto, através de seus módulos, cada um simulando atividades do hardware e do software básico como:

a) mecanismos de acesso ao barramento;
b) parâmetros como:
    b.1) velocidade do barramento;
    b.2) tempo de "hashing" de memória;
    b.3) tempo de execução;
    b.4) "timeout";
c) "hashing" de memória;
d) execuções das operações a fluxo de dados;
e) escalonamento das atividades (recepção, execução, transmissão e "timeout") em um EP;
f) protocolo de comunicação;
g) "gateway";
h) manipulação de "buffers" de entrada e saída;

se mostra um simulador que se aproxima bastante do que será a realidade da máquina CPER, configurada como uma MFDD.

Acreditamos portanto que seus resultados geram uma referência consistente com os resultados a serem obtidos através da MFDD real.

Vemos principalmente que o FDsim se configura como um laboratório de software para o desenvolvimento da máquina.

No caso do hardware, o esquema de "hashing" de memória,

deverá utilizar toda a estrutura de dados proposta no simulador, para a identificação entre os dados parceiros em uma instrução.

No caso do software, todo o desenvolvimento do protocolo de comunicação para Tolerância a Falhas, o próprio processo de execução das operações, a solicitação do "hashing" na memória e o "gateway", deverão ser implementados a partir da experiência do desenvolvimento do simulador.

C A P Í T U L O 7

AVALIAÇÃO DA MFDD

7.1 - INTRODUÇÃO

Como vimos o FDsim (Fluxo de Dados simulador) é um simulador da MFDD que permite editar, analisar, e executar programas a fluxo de dados, tendo como base o hardware do CPER descrito inicialmente, e teve como objetivo, validar as estruturas propostas para os programas a fluxo de dados tolerante a falhas, e também ser utilizado como um laboratório para o desenvolvimento do software de base da MFDD, além de gerar resultados de desempenho.

Vimos também que o FDsim está implementado em Pascal através de alguns módulos: o Módulo Editor; o Módulo Analisa; o Módulo Relógio; o Módulo Recepção; o Módulo de Execução; o Módulo Barramento; e o Processo Tolerância a Falhas, através das rotinas "SEND" e "RECEIVE", que implementam o protocolo de Tolerância a Falhas na MFDD.

Descrevemos também as estruturas básicas no FDsim, como:

a) Matriz Hardware da MFDD - cujas linhas e colunas

especificam "cluster" e EPs respectivamente, onde cada elemento dessa matriz descreve a estrutura de um EP

b) Matriz Barramento - cujas linhas e colunas descrevem também"clusters"e EPs respectivamente, onde cada elemento da matriz descreve permissão de passagem e qual EP quer transmitir, de tal forma que apenas uma permissão de passagem circula na matriz, alcançando sempre aquele EP que quer transmitir,simulando o esquema "round-robin", um para cada "cluster".

c) Matriz de Eventos cujas linhas e colunas também descrevem os"clusters" e EPs respectivamente, onde cada elemento da matriz, através de um conjunto de variaveis, reflete o estado atual de cada EP, de tal forma que o EP circula entre 3 estados distintos: recepção, execução, e transmissão; interrompido apenas por "timeout" no protocolo de comunicação.

d) Listas Encadeadas - a partir da matriz hardware da MFDD, utilizadas para gerar as ativações e iterações correspondente a cada operação a fluxo de dados.

A partir da estrutura do FDsim e dos parâmetros a ele atribuidos, foi possível executar alguns programas a fluxo de dados e obter alguns resultados, de tal forma que se pudesse verificar algumas características da MFDD no que se refere ao desempenho.

Com o objetivo de tornar consistente os resultados obtidos, foram feitas algumas análises estocásticas, para os

programas que foram executados na MFDD através do FDsim. Para isso foi necessário incluir no simulador, um processo de entrada de dados aleatório de tal forma que os resultados obtidos a partir do simulador pudessem ser comparados aos obtidos através da análise estocástica [GORDON, 1978].

7.2 - ENTRADA DE DADOS NO FDsim SEGUNDO CHEGADA POISSON

Na entrada aleatória de dados, em se tratando de uma análise estocástica a partir da análise de filas, assumimos entrada de dados com chegada Poisson, com as seguintes características:

a) Geração de valores com Distribuição Exponencial a partir de:

$$z = -\ln(1-y) \qquad (1)$$

onde y é um número aleatória no intervalo $0 \leq y \leq 1$.

No APÊNDICE A está descrito como chegamos à expressão (1) para distribuição exponencial.

b) Assumindo um intervalo entre chegadas obtido a partir de dados do CPER:

c = 132 Mbytes/seg; capacidade máxima no barramento

k = 600 bits; tamanho das mensagens

m = c/k = 132 Mbytes/600 bit = 1.76 Mmensagens

u = 1/m = 570 ns;tempo máximo de transmissão (2)

c) Cálculo do intervalo entre as chegadas, obtidos a partir de (1) e (2):

t = z * u; intervalo entre as chegadas

Podemos obter então a Tabela II para o intervalo entre as chegadas segundo parâmetros acima:

T A B E L A II

Números Aleatórios Uniformemente Distribuídos y	Números Aleatórios Exponencialmente Distribuídos z	Intervalo de tempo entre chegadas t (ns)	Tempo entre chegadas acumulado (ns)
0.100	0.104	60	60
0.375	0.470	267	327
0.084	0.087	50	377
0.990	4.600	2622	2990
0.128	0.137	78	3077
0.660	1.086	619	3696

A partir da Tabela II pudemos obter o tempo entre as chegadas que foram utilizados no FDsim para gerar a entrada de dados aleatórias para a execução do programa a fluxo de dados, segundo chegada Poisson, e computado no tempo total de execução do programa a fluxo de dados. A escolha de qual operação receberia o dado, foi obtida também de forma aleatória. Vale

observar que o processo vale inclusive para as entradas de dados na forma "pipeline", onde uma primeira operação a fluxo de dados pode receber todas as entradas na forma "pipeline" (várias ativações), só então uma segunda operação receberia seus dados.

7.3 - RESULTADO DA EXECUÇÃO DE PROGRAMAS NO FDsim

A TABELA III contém os parâmetros e os resultados obtidos a partir da execução dos programas a fluxo de dados no FDsim.

TABELA III

PARÂMETROS:

A) TEMPO DE TRANSMISSÃO PELO BARRAMENTO
DE UM PACOTE 64 BITS: 61ns

B) NÚMERO DE BITS EM UMA MENSAGEM FLUXO DE DADOS: 600bits

C) NÚMERO DE PACOTES A SEREM TRANSMITIDOS: 10

D) TEMPO DE "HASHING" DE MEMÓRIA PARA ALOCAR UM DADO: 70ns

E) TEMPO DE PROCESSAMENTO DE UMA OPERAÇÃO
A FLUXO DE DADOS : 500ns

PRG	NOP	EP	GAT	CLU	NIVEIS 1	2	3	4	5	PIPELINE/RESULTADO (ns) pipe1	pipe2	pipe3
1	5	5	0	1	3	1	1	0	0	3746.76	6237.45	8829.46
2	3	3	0	1	2	1	0	0	0	2307.39	3952.08	6018.18
3	5	3	2	3	2	2	1	0	0	4848.89	7192.72	9796.38
4	9	7	2	3	4	2	2	1	0	6689.90	9684.14	-------
5	9	7	2	3	3	1	2	2	1	6557.89	-------	-------
6	5	5	0	1	2	2	1	0	0	4953.03	7641.31	10888.78
7	4	4	0	1	1	1	1	1	0	5192.23	8288.17	10958.60

PRG - referência dos programas conforme descrito nas figuras:

Fig.7.1, Fig.7.3, Fig.7.5, e Apêndice F.
NOP - Número de operações do programa
EP - Quantidade de EPs utilizados no programa
GAT - Quantidade de "Gateways" utilizados no programa
CLU - Número de "Cluster" utilizados no programa
NÍVEIS (1,2,3,4,5) - Níveis de paralelismo das operações.

Nível 1 possue operações de entrada de dados no programa a fluxo de dados.

Níveis 2,3,4 e 5 possuem as operações sub-sequentes no programa. Cada nível contém um número de operações que serão executadas em paralelo.

Ex: PRG 1 possue 5 operações distribuídas em 5 EPs distintos, em um único "cluster" (veja Fig.7.1), dispensando o "gateway", onde 3 delas serão executadas em paralelo no nível 1, 1 no nível 2, e 1 no nível 3. PRG 5 (veja Apêndice F2),possue 9 operações, 7 distribuídas em 7 EPs distintos, e 2 em dois "gateways" distintos, sendo necessário portanto 3 "clusters", onde 3 delas serão executadas em paralelo no nível 1, 1 no nível 2, 2 no nível 3, 2 no nível 4, e 1 no nível 5.

PIPELINE/RESULTADO (ns) - Resultado da execução dos programas quando se atribui entrada de dados na forma "pipeline". (pipe1) significa entrada de apenas um dado para cada operação de entrada de dados do usuário; (pipe2) duas entradas simultânea de dados para cada operação de entrada; e (pipe3) três entradas simultânea de dados para cada operação de entrada.

O número limitado de operações, "gateways", e alguns resultados não obtidos, como: programa 4 pipe3, programa 5 pipe2 e pipe3; deveram-se basicamente às limitações do PASCAL 6.0, que não gerencia bem grandes estruturas de dados, e não manipula bem memória estendida, recursos esses bastante utilizados no simulador, que possui 9283 linhas de comandos Pascal distribuídos em 12 "UNITS" e mais o programa principal, num total de 118880.bytes de programa executável (APÊNDICE E). Sendo assim só foi possível gerar resultados para programas pequenos.

7.4 - ANÁLISE DE FILAS PARA PROGRAMAS A FLUXO DE DADOS EXECUTADOS NO FDsim

Na análise de filas, consideramos três modelos, que pudessem ser utilizados como base para a análise dos programas executados no FDsim e que foram analisados estocasticamente.

Todos os modelos tiveram uma característica fundamental que é modelar a execução a fluxo de dados, mas principalmente modelar a operação do barramento do CPER, que é um elemento de gargalo do sistema.

A entrada de dados para as Redes de Filas de todos os programas analisados, foi considerada a partir de uma taxa que representa 30% da capacidade máxima do barramento, dados esses que virão do Hospedeiro. E em função do número de filas que recebem dados externos, foi feita uma divisão proporcional (Pi) da taxa para cada fila.

O primeiro modelo analisado foi o modelo de Rede de Filas com Realimentação (realimentação gerada pelo barramento do CPER por estar ocupado).

O PROGRAMA 1, relacionado na Tabela III, está descrito na Fig.7.1, e o seu modelo de Rede de Filas com Realimentação está descrito na Fig.7.2.

Conforme Fig.7.2, cada fila representa uma operação do Programa 1. A disposição das filas obedeceu a mesma disposição, por níveis das operações no programa.

A Realimentação em algumas filas, se deve ao fato de que em um mesmo nível deverá ocorrer uma disputa no barramento do

CPER para transmitir dados, o que significa que quando algum elemento tentar transmitir, o barramento poderá estar ocupado, e portanto, aquele elemento deverá fazer uma nova tentativa de transmissão. A probabilidade Li é decorrente deste mecanismo de acesso ao barramento onde qualquer elemento (aleatoriamente) pode tentar transmitir. A probabilidade Li portanto foi obtida levando em consideração a probabilidade do barramento estar ocupado. Nesta análise, supondo n elementos concorrendo ao barramento de maneira aleatória (sem controle), teremos:

$$\text{Prob } \{\text{não haver colisão}\} = np(1-p)^{n-1} \qquad (3)$$

Onde:

p é a probabilidade de um "buffer" enviar um pacote que é equivalente a ter pelo menos um pacote no "buffer";

n é o número de processadores.

Portanto:

$p = 1 - \text{Prob } \{k=0\}$ o que nos leva a:

$$p = 1 - \exp(-\lambda t) \qquad (4)$$

No APÊNDICE B descrevemos como chegamos à expressão (3) para probabilidade de não haver colisão.

Para uma taxa de 132 Mbytes/seg em 61ns para transmitir um pacote de 64 bits; e para um pacote total de 600 bits teremos aproximadamente 600ns para transmitir um pacote. Podemos então assumir t = 600ns, com uma taxa λ aproximadamente igual a 0.528 Mmsg/seg.

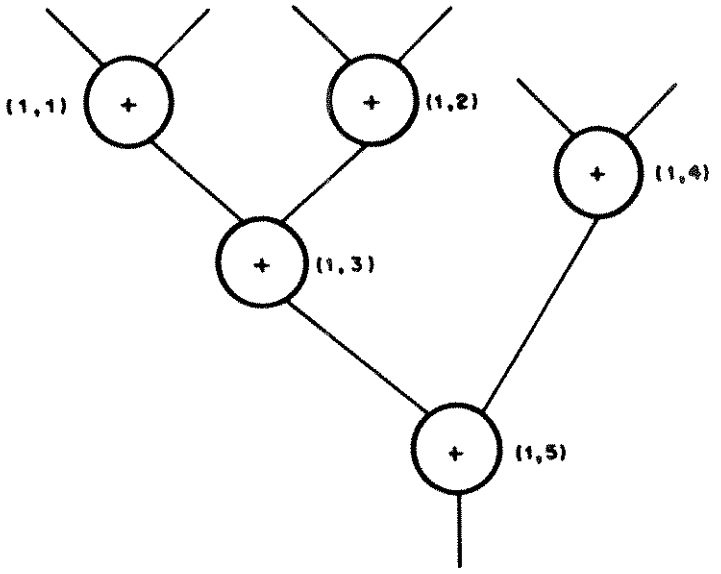


FIGURA 7.1 - PROGRAMA 1.

Em função de (3), e (4) podemos calcular Li para cada ponto do programa dependendo do número de elementos que concorrem ao barramento.

No modelo, as setas indicando saídas isoladas, a partir de uma probabilidade Di, foram considerada pelo fato de que algumas operações recebem dois dados de entrada (dois clientes) e geram apenas um resultado na saída (um cliente). Portanto, compensamos essa diferença, através de Di, com perda de clientes na entrada de uma operação, de tal forma que a junção de duas entradas gerem apenas um cliente, enquanto que o outro é perdido, metade em cada entrada.

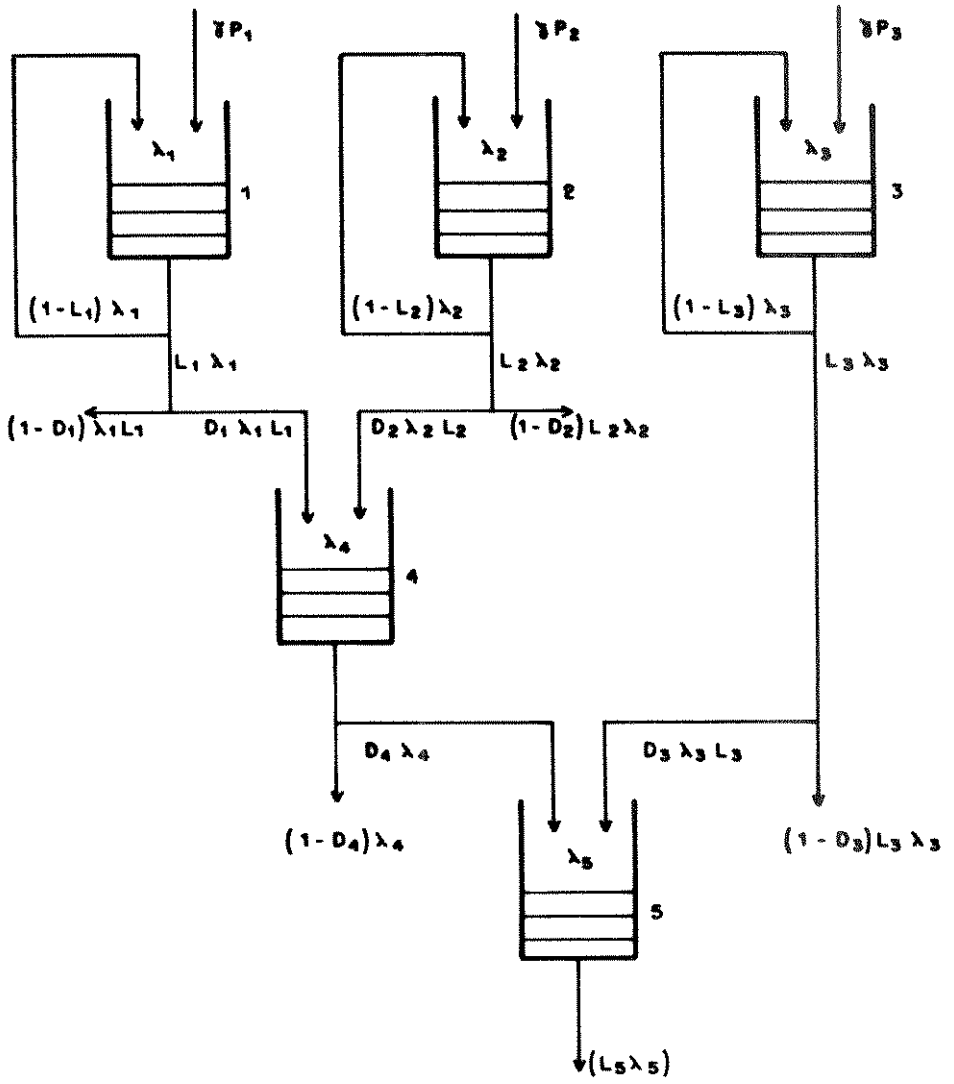


FIGURA 7.2 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO PROGRAMA 1.

Na Fig. 7.3 descrevemos o Programa 3, relacionado na Tabela III, que é um programa que utiliza operações "gateways". Seu modelo de Rede de Filas com Realimentação está descrito na Fig.7.4.

No modelo da Fig.7.4, cabe observar que não houve nenhuma tentativa de enquadrar operações "gateway" em algum esquema especial, pois se trata de uma operação a fluxo de dados como as outras disputando barramento exatamente da mesma forma.

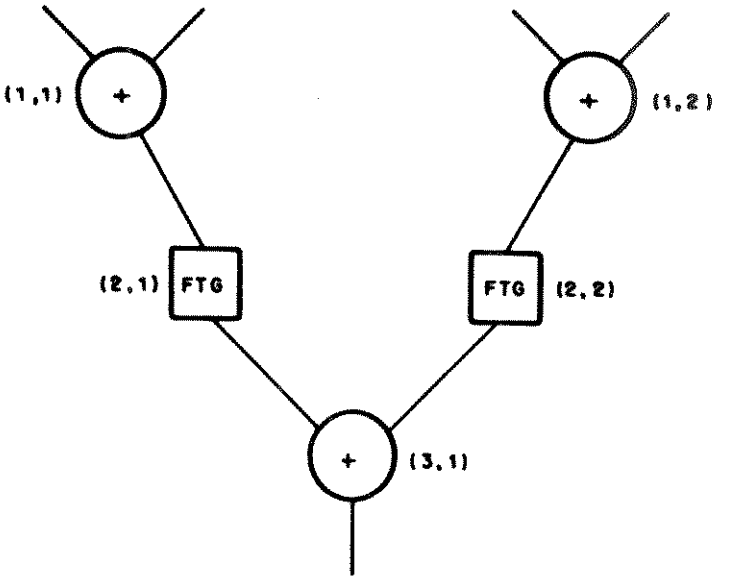


FIGURA 7.3 - PROGRAMA 3.

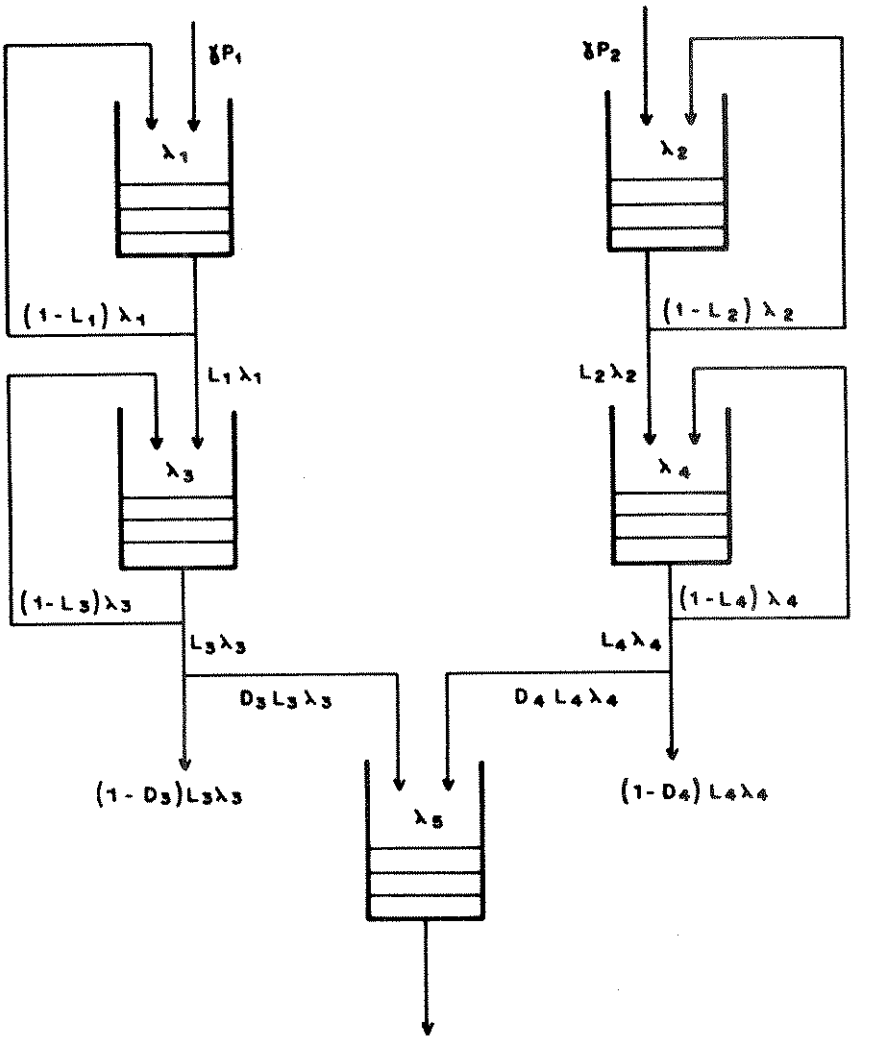


FIGURA 7.4 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO PROGRAMA 3.

Na Fig. 7.5 descrevemos o Programa 4, também relacionado na Tabela III, que possui um número maior de operações e também utiliza operações "gateway". Seu modelo de Rede de Filas com Realimentação está descrito na Fig. 7.6.

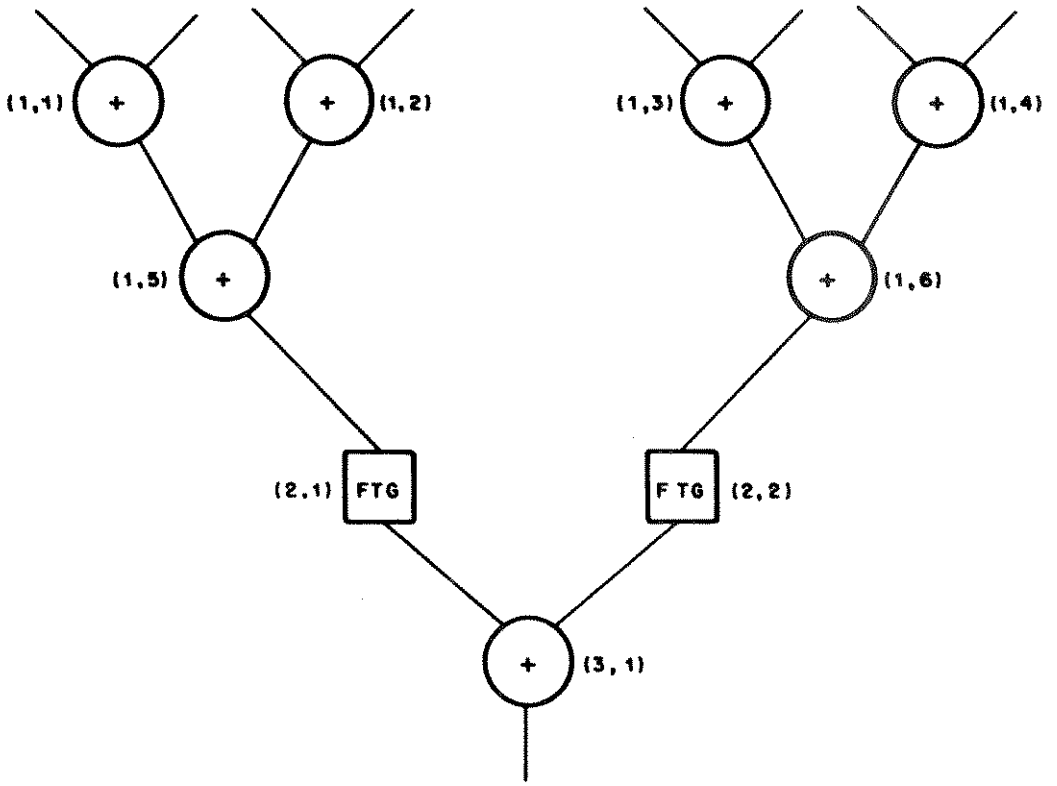


FIGURA 7.5 - PROGRAMA 4.

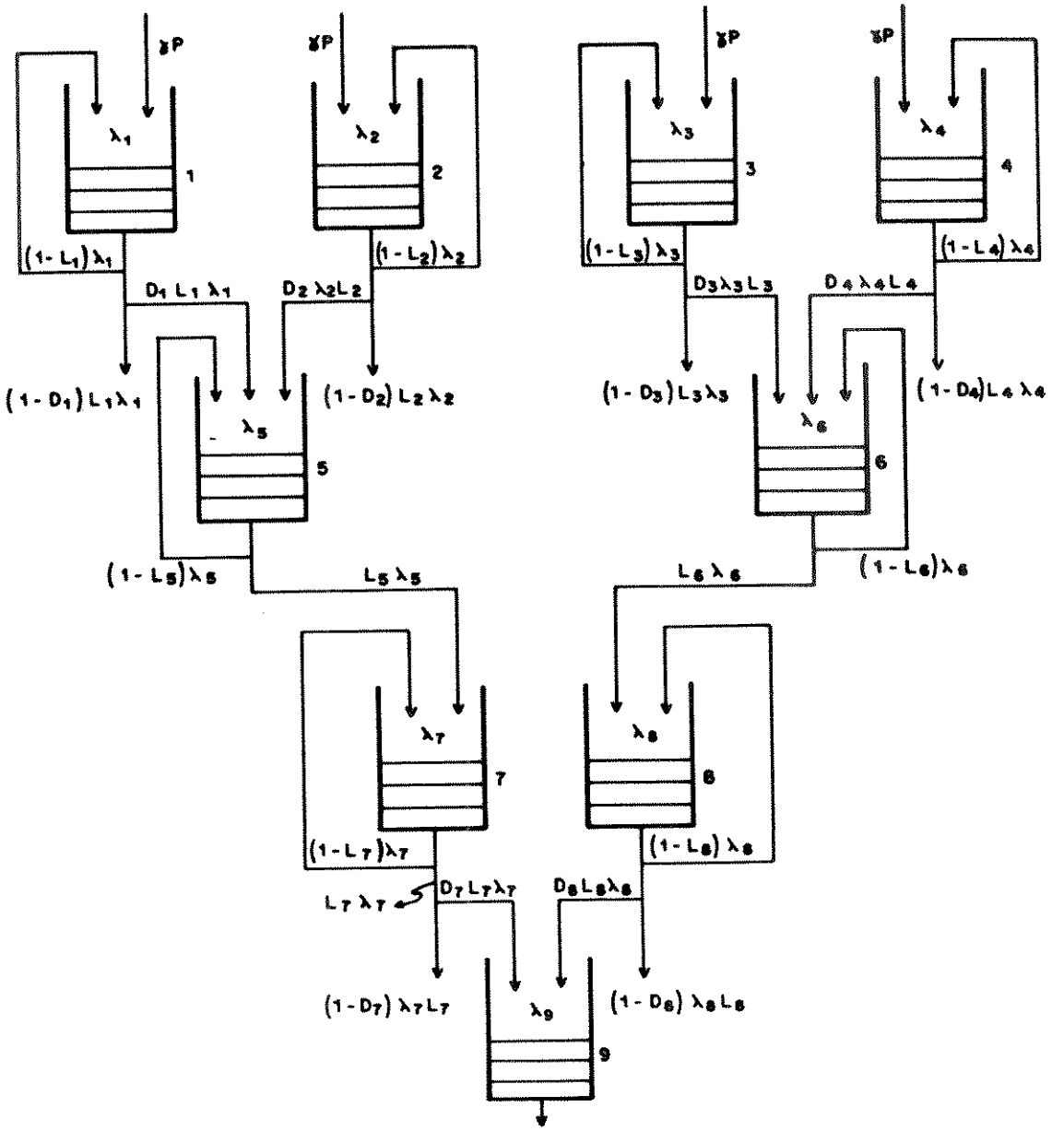


FIGURA 7.6 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO

PROGRAMA 4.

O segundo modelo analisado, é o que elabora um mecanismo de acesso em forma de anel no barramento [HAMMOND e O'REILY, 1986]. Neste modelo é calculado um tempo médio para escoar toda uma mensagem através do barramento (serviço exaustivo), segundo o mecanismo de acesso em anel. A expressão utilizada para a obtenção do tempo médio é a descrita a seguir:

$$W = \frac{M\ w\ (1 - S/M)}{2\ (1 - S)} + \frac{S\ \bar{x}}{R\ (1 - S)} \qquad (5)$$

Onde:

$S = M\ \lambda\ \bar{x}/R$

$w = 61ns$; tempo para transmissão no barramento

$R = 132$ Mbytes/seg ; capacidade do barramento

$\bar{x} = 75$ bytes ; bytes por mensagem

No APÊNDICE C descrevemos como chegamos à expressão (5) para análise do modelo de Redes em Anel.

Na Fig. 7.7 descrevemos o modelo de Rede de Filas em Anel, correspondente ao Programa 1 descrito na Fig.7.1.

No modelo da Fig 7.7, podemos notar que cada fila também representa uma operação do Programa 1. A disposição das filas também obedeceu a mesma disposição, por níveis, das operações no programa. A diferença deste modelo está na forma de acesso ao barramento, representada na figura pelo círculo tracejado, pois aqui se utiliza um mecanismo de acesso em forma de anel de tal forma que quando um elemento quer transmitir ele deve aguardar permissão para transmitir, e quando isso acontece ele transmite todas as informações que ele tem em uma única vez. Neste modelo é mantido também a probabilidade Di do modelo anterior, como a probabilidade de compensação do número de entrada de dados e do número de saída de dados.

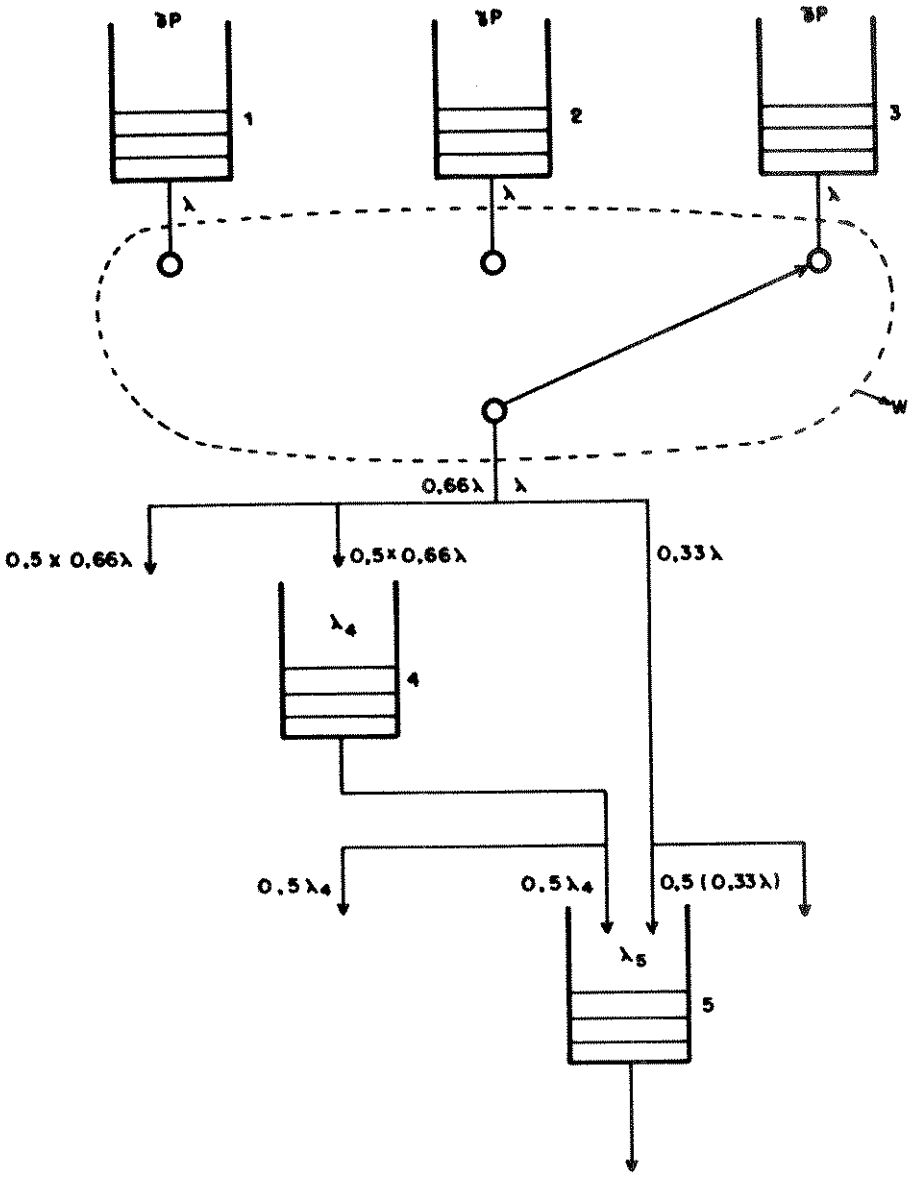


FIGURA 7.7 - REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 1.

Na Fig. 7.8, e Fig. 7.9 descrevemos os modelos de Rede de Filas em Anel, correspondentes aos Programa 3 (Fig. 7.3), e Programa 4 (Fig. 7.5), respectivamente. Aqui também não houve necessidade de tratamento especial para operações "gateway".

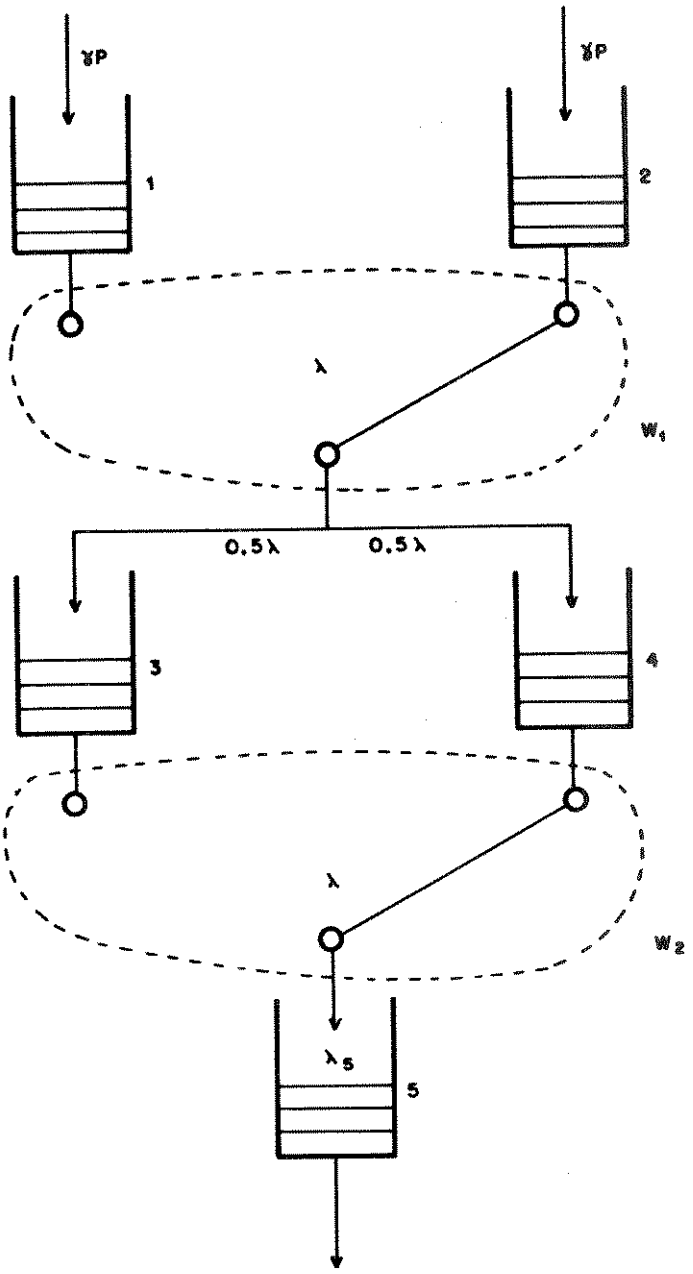


FIGURA 7.8 - REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 3.

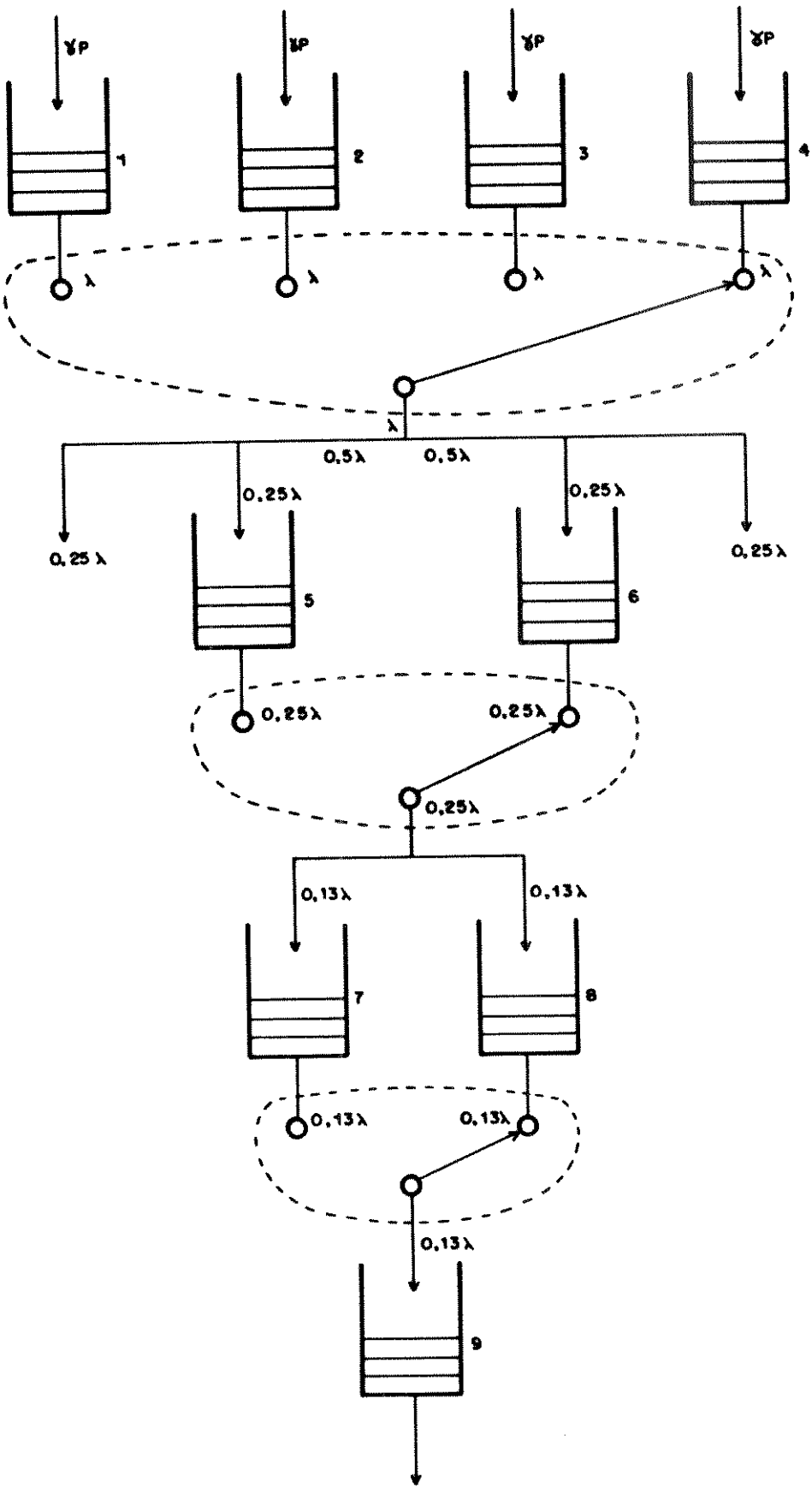


FIGURA 7.9 - REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 4.

O terceiro modelo analisado, é o que elabora um mecanismo de acesso em forma de anel com "slots" no barramento [MOTOYAMA e ARANTES, 1992], [TAKAGI, 1986]. Neste modelo, a mensagem é dividida em vários pacotes e a transmissão é feita por pacotes utilizando o mecanismo de acesso em anel por "slots". A expressão utilizada para a obtenção do tempo total de transmissão de todos os "slots" é a descrita a seguir:

$$Wp = \left\{ \frac{\lambda\ G\ Tu\ (N+1) + 2\ G\ (N+1) - 1}{2\ [\ 1 - \lambda\ G\ Tu\ (N+1)\ ]} \right\} Tu \quad (6)$$

Onde:

N o número de processadores;

G o número médio de pacotes por mensagem (10 no CPER)

Tu o tempo de transmissão do último pacote (61 ns)

No APÊNDICE D descrevemos como chegamos à expressão (6) para análise do modelo de Rede em Anel com "Slots".

A figura que descreve o modelo de Rede de Filas com mecanismo de acesso em anel com "slots" é também a Fig. 7.7, Fig. 7.8, e Fig. 7.9, para o Programa 1, Programa 3, e Programa 4, respectivamente; exceto que a transmissão agora é por "slots".

Também neste modelo é mantido a probabilidade Di dos modelos anteriores, e também não há nenhum tratamento especial para operações "gateway".

Os Programas 2, 5, 6, e 7, e seus respectivos modelos

para Rede de Filas, segundo os três modelos analisados, estão descritos no Apêndice F.

A TABELA IV contém os resultados obtidos a partir da análise de Rede de Filas feita para os programas referenciados na Tabela III (resultados do FDsim), segundo os modelos descritos neste capítulo.

T A B E L A IV

PRG	NOP	EP	GAT	CLU	M O D E L O S Realim.	Anel	Anel/slots
1	5	5	0	1	3906.00	4084.00	8078.00
2	3	3	0	1	2840.00	2818.00	5770.00
3	5	3	2	3	4004.00	4125.00	9570.00
4	9	7	2	3	5231.00	5329.00	13544.00
5	9	7	2	3	5044.00	5285.00	12785.00
6	5	5	0	1	5184.00	4125.00	9570.00
7	4	4	0	1	5442.00	-------	--------

PRG - Referência do programa
NOP - Número de operações do programa
EP - Quantidade de EPs utilizados no programa
GAT - Quantidade de "Gateways" utilizados no programa
CLU - Número de "Cluster" utilizados no programa

Obs: No programa 7 não existe resultados para o modelo em Anel, e Anel com "slots", pois não existe a disputa pelo acesso ao barramento.

Na Tabela V, agrupamos o resultados obtidos através do simulador FDsim (Tabela III), com os resultados obtidos através dos três modelos propostos para análise de filas (Tabela IV).

T A B E L A V

	FILAS			FDsim		
	MODELOS			PIPELINE/RESULTADOS(ns)		
PRG	Realim.	Anel	Anel/slots	pipe1	pipe2	pipe3
1	3906.00	4084.00	8078.00	3746.76	6237.45	8829.46
2	2840.00	2818.00	5770.00	2307.39	3952.08	6018.18
3	4004.00	4125.00	9570.00	4848.89	7192.72	9796.38
4	5231.00	5329.00	13544.00	6689.90	9684.14	-------
5	5044.00	5285.00	12785.00	6557.89	-------	-------
6	5184.00	4125.00	9570.00	4953.03	7641.31	10888.78
7	5442.00	-------	-------	5192.23	8288.17	10958.60

Analisando a Tabela V, podemos verificar que os resultados obtidos através do simulador e os obtidos através da análise de filas estão consistentes, apesar da pequena quantidade de valores que foi possível levantar.

Para fazer uma comparação entre os programas executados na MFDD, através do FDsim, e o que seria fazer esses programas de forma totalmente sequencial, montamos a Tabela VI.

Conforme a Tabela VI, podemos notar que os programas, se rodados de forma sequencial em um sistema monoprocessador tipo "transputer", acabariam gerando resultados mais rápidos que quando executados na MFDD, principalmente aqueles programas que utilizam "gateways", e particularmente em função do protocolo de Tolerância a Falhas.

T A B E L A VI

PRG	PIPELINE/RESULTADO (ns) pipe1	pipe2	pipe3	SEQUENCIAL pipe1	pipe2	pipe3	
1	3746.76	6237.45	8829.46	2500.00	5000.00	7500.00	
2	2307.39	3952.08	6018.18	1500.00	3000.00	4500.00	
3	4848.89	7192.72	9796.38	1500.00	3000.00	4500.00	
4	6689.90	9684.14	-------	3500.00	7000.00	-------	
5	6557.89	-------	-------	3000.00	-------	-------	
6	4953.03	7641.31	10888.78	2500.00	5000.00	7500.00	
7	5192.23	8288.17	10958.60	2000.00	4000.00	6000.00	
	D I F E R E N Ç A FDsim / Sequencial			O V E R H E A D FDsim (%)			Média
1	1246.76	1237.45	1329.46	+150	+107	+96	117
2	807.39	952.08	1518.18	+130	+97	+100	109
3	3348.89	4192.72	5296.38	+348	+259	+226	277
4	3189.90	2684.14	-------	+345	+222	-----	283
5	3557.89	-------	-------	+337	-----	-----	337
6	2453.03	2641.31	3308.78	+147	+91	+81	106
7	3192.23	4288.17	4958.60	+159	+107	+82	116

Em função das dificuldades em obter resultados mais expressivos através do simulador (seção 7.3), fizemos uma projeção para a execução de outros programas, cuja análise inicia a partir da diferença de resultados gerados entre os programas (FDsim/Sequencial - Tabela VI).

Computando nos programas executados no FDsim, apenas os tempos de execuções para cada nível do programa (Tabela III), independente de quantas operações existam em um determinado nível (paralelismo propriamente dito), podemos calcular uma porcentagem de acréscimo de "overhead" sofrido por cada programa executado através do FDsim, em função do barramento, do protocolo tolerância a falhas, e do próprio

"gateway", em geral com um aumento de 112% em média de "overhead" nos programas que não utilizam o "gateway", e 299% em média quando é utilizado o "gateway".

Desta forma podemos considerar que:

1) A porcentagem de "overhead" vai diminuindo à medida que vai havendo "pipeline" de dados nas operações.

2) Fazendo uma projeção para outros programas, tal que fossem executados através do FDsim, computando apenas os níveis de paralelismo acrescidos das porcentagens descritas acima, e tivessem sua execução também computada de forma sequencial, pudemos gerar a TabelaVIIa e Tabela VIIb.

T A B E L A VIIa

PRG	NOP	EP	GAT	CLU	NIVEIS 1	2	3	4	5	6	SEQUENCIAL(ns) pipe1	pipe2	pipe3
1	11	11	0	1	6	3	1	1	0	0	5500.00	11000.00	16500.00
2	15	15	0	1	6	4	2	1	0	0	7500.00	15000.00	22500.00
3	20	19	1	3	10	5	2	1	1	1	9500.00	19000.00	28500.00
4	24	23	1	3	12	6	3	1	1	1	11500.00	23000.00	34500.00
5	28	27	1	3	14	7	4	1	1	1	13500.00	27000.00	40500.00

PRG	NOP	EP	GAT	CLU	NIVEIS 1	2	3	4	5	6	FDsim/PIPELINE(ns) pipe1	pipe2	pipe3
1	11	11	0	1	6	3	1	1	0	0	3180.00	6360.00	9540.00
2	15	15	0	1	6	4	2	1	0	0	4240.00	8480.00	12720.00
3	20	19	1	3	10	5	2	1	1	1	9870.00	19740.00	29610.00
4	24	23	1	3	12	6	3	1	1	1	9870.00	19740.00	29610.00
5	28	27	1	3	14	7	4	1	1	1	11515.00	23030.00	34545.00

T A B E L A VIIb

PRG	PIPELINE/RESULTADO (ns) pipe1	pipe2	pipe3	SEQUENCIAL pipe1	pipe2	pipe3
1	3180.00	6360.00	9540.00	5500.00	11000.00	16500.00
2	4240.00	8480.00	12720.00	7500.00	15000.00	22500.00
3	9870.00	19740.00	29610.00	9500.00	19000.00	28500.00
4	9870.00	19740.00	29610.00	11500.00	23000.00	34500.00
5	11515.00	23030.00	34545.00	13500.00	27000.00	40500.00

S P E E D U P

Programa - 1 0.15
Programa - 2 0.11
Programa - 3 0.04
Programa - 4 0.04
Programa - 5 0.04

"Speed up" obtido através de:

Aceleração = Tseq/Tparal = Ac

Eficiência = Acerelação / no. de processadores.

Exemplo Programa 1:

Ac = 16500 / 9540 = 1.72

Eficiência = 1.72 / 11 = 0.15

Conforme projeções e "speed up" acima podemos verificar que a Eficiência da MFDD dependerá de:

a) Número de operações do programa a fluxo de dados a ser executado;
b) Maneira como que essas operações estarão distribuídas na

MFDD;

c) Número de dados de entrada ("pipeline") do programa.

Podemos concluir então que a MFDD projeta uma grande eficiência sobre uma máquina monoprocessador, à medida que:

1) Se tenha programas altamente paralelos a serem convertidos em programas a fluxo de dados, para execução na MFDD;
2) Exista um mapeamento das operações a fluxo de dados a serem atribuidas aos EPs da MFDD [D'AREZZO, 1991], de tal forma a se obter o máximo de paralelismo na execução dos programas a fluxo de dados;
3) Se faça uso intenso da entrada de dados na forma "pipeline";

Ainda como característica fundamental da MFDD, está a Tolerância a Falhas na programação a fluxo de dados, que torna a MFDD tolerante a falhas. E em se tratando de máquina a fluxo de dados, tal característica coloca a MFDD como um sistema de vanguarda no que diz respeito à programação a fluxo de dados tolerante a falhas.

7.5 - CONCLUSÃO

Na avaliação da MFDD buscamos tornar consistente os resultados obtidos através do FDsim, comparando-os aos resultados obtidos através de algumas análises estocásticas para

alguns programas que foram executados na MFDD através do FDsim.

Para isso foi necessário incluir no simulador um processo de entrada de dados com chegada Poisson, que geraram os resultados descritos na TABELA II, segundo os parâmetros atribuidos ao FDsim, e que pudessem então ser comparados aos resultados obtidos através dos modelos estocásticos analisados.

O primeiro modelo analisado foi o modelo de Rede de Filas com Realimentação (realimentação gerada pelo barramento do CPER por estar ocupado). O segundo modelo analisado, é o que elabora um mecanismo de acesso em forma de anel no barramento. E o terceiro modelo analisado, é o que elabora um mecanismo de acesso em forma de anel com "slots" no barramento.

Feita a análise dos modelos, pudemos então gerar a TABELA IV, que contém os resultados obtidos para a análise de cada modelo em cada programa rodado no FDsim.

Na Tabela V, agrupamos o resultados obtidos através do simulador FDsim (Tabela III), com os resultados obtidos através dos três modelos propostos para análise de filas (Tabela IV), e pudemos verificar que os resultados obtidos através do simulador e os obtidos através da análise de filas estão consistentes, apesar da pequena quantidade de valores que foi possível levantar.

A TABELA VI foi montada para fazer uma comparação entre os programas executados na MFDD, através do FDsim, e o que seria executar esses programas de forma totalmente sequencial, onde verificamos que os programas sequenciais acabariam sendo executados mais rapidamente.

Em função das dificuldades em obter resultados mais

expressivos através do simulador fizemos uma projeção para outros programas, computando apenas os tempos por níveis de paralelismo, acrescidos das porcentagens descritas na TABELA VI, que comparados ao modelo sequencial (Tabela VIIa, Tabela VIIb), pudemos verificar que a Eficiência da MFDD dependerá do número de operações do programa a fluxo de dados a ser executado; da maneira como que essas operações estarão distribuídas na MFDD; e do número de dados de entrada ("pipeline") do programa.

Concluimos então que a MFDD projeta uma grande eficiência sobre uma máquina monoprocessador, à medida que: se tenha programas altamente paralelos a serem convertidos em programas a fluxo de dados, para execução na MFDD; exista um mapeamento das operações a fluxo de dados a serem atribuidas aos EPs da MFDD, de tal forma a se obter o máximo de paralelismo na execução dos programas a fluxo de dados; e se faça uso intenso da entrada de dados na forma "pipeline".

Finalmente como característica fundamental da MFDD, está a Tolerância a Falhas na programação a fluxo de dados, que torna a MFDD tolerante a falhas. E como dissemos , em se tratando de uma máquina a fluxo de dados, tal característica coloca a MFDD como um sistema de vanguarda no que diz respeito à programação a fluxo de dados tolerante a falhas.

C A P Í T U L O 8

CONCLUSÕES GERAIS

8.1 - COMENTÁRIOS FINAIS E CONTRIBUIÇÕES

Para o descrição deste trabalho, iniciamos por apresentar as características básicas dos sistemas de processamento paralelo cujo processamento é dirigido por controle, e em seguida caracterizamos os sistemas cujo processamento é dirigido por dados.

Caracterizamos software nestas máquinas através da centralização do controle (Sistema Operacional Centralizado), e da distribuição do controle (Sistema Operacional Distribuído).

Nos sistemas cujo processamento é dirigido por dados, descrevemos a linguagem original para programas a fluxo de dados, a máquina que foi proposta para execução da linguagem, consideramos características como Máquina a Fluxo de Dados Estática, e Máquinas a Fluxo de Dados Dinâmica, foram apresentadas também as estruturas de algumas máquinas a fluxo de dados, e concluímos por fazer um estudo do software para máquinas a fluxo de dados.

Apresentamos então a proposta do projeto CPER que teve

como objetivo básico possuir alta velocidade de comunicação no barramento e alta velocidade de processamento, servindo às necessidades básicas para um sistema multiprocessamento que opera como uma máquina a fluxo de dados.

Apresentamos também o software do CPER (parte de um Núcleo em um Sistema Operacional Distribuído) que teve como objetivo configurar o CPER de tal forma que ele opere como uma Máquina a Fluxo de Dados Dinâmica (MFDD), com características de Tolerância a Falhas na programação a fluxo de dados.

Com esta estrutura de hardware e de software pudemos concluir que o CPER se mostra uma Máquina a Fluxo de Dados Dinâmica (MFDD) que explora ao máximo o paralelismo natural da programação a fluxo de dados, principalmente se fizermos um mapeamento (operação/Elemento de Processamento) adequado explorando ao máximo as características de granularidade fina, média e grossa.

Apresentamos um simulador, o FDsim, que teve como objetivo, validar a estrutura dos programas a fluxo de dados tolerante a falhas na MFDD, e que deverá ser utilizado como um laboratório para o desenvolvimento do software de base do CPER para que este opere como a MFDD, além de gerar resultados de desempenho.

Na avaliação da MFDD pudemos concluir que: à medida que se tenha programas altamente paralelos a serem convertidos em programas a fluxo de dados, para execução na MFDD; exista um mapeamento das operações a fluxo de dados a serem atribuidas aos EPs da MFDD, de tal forma a se obter o máximo de paralelismo na execução dos programas a fluxo de dados;

e se faça uso intenso da entrada de dados na forma "pipeline", deveremos obter grande eficiência sobre uma máquina monoprocessadora.

Destacamos novamente como característica fundamental da MFDD, a Tolerância a Falhas na programação a fluxo de dados, que torna a MFDD tolerante a falhas. E como dissemos, em se tratando de uma máquina a fluxo de dados, tal característica coloca a MFDD como um sistema de vanguarda no que diz respeito à programação a fluxo de dados tolerante a falhas.

Um protótipo reduzido da estrutura hierárquica do CPER, com 3 Elementos de Processamento ligados por um barramento, foi montado para testar o comportamento e desempenho da interface no barramento [KIRNER, 1989], [MARQUES et al, 1988].

O Simulador está implementado em PASCAL, com um total de 9283 linhas num total de 118880 Kbytes de programa executável.

8.2 - SUGESTÕES PARA CONTINUAÇÃO DA PESQUISA

Como continuidade do projeto, está a transposição do programa FDsim para uma máquina mais poderosa (SUN, CYBER, IBM-RISC, MULTI-TRANSPUTER, outras), com o objetivo de obter resultados mais expressivos da simulação, e oferecer ao usuário uma interface gráfica mais amigável, com edição, acompanhamento gráfico do processo de simulação, e visualisação gráfica dos resultados.

Em paralelo estaremos fazendo um estudo da viabilidade de desenvolver o CPER a Fluxo de Dados utilizando microcontroladores, que são dispositivos de baixo custo, de fácil

utilização, e de grande aceitação no parque industrial brasileiro, com o objetivo básico de oferecer um sistema de grande capacidade de processamento, tolerante a falhas, baixo custo, e que possa ser utilizado em aplicações tipicamente industriais.

A P E N D I C E A

C H E G A D A P O I S S O N

Vamos supor A(t) a probabilidade de que o tempo entre chegadas seja maior que t.

Desde que F(t) é a probabilidade de que o tempo entre chegadas seja menor que t, podemos dizer que a distribuição de chegada é dada por:

$$A(t) = 1 - F(t) \qquad (1)$$

Supondo λ o número de chegadas por unidade de tempo, e ?▲t um tempo pré-definido: dizemos que λ▲t é a probabilidade de uma chegada em ▲t.

Com esta afirmação é possível mostrar que a distribuição do tempo entre chegadas é exponencial. A função densidade de probabilidade do intervalo de tempo entre chegadas é dado por:

$$f(t) = \lambda\, e^{(-\lambda t)}$$

A distribuição de chegadas é:

$$A(t) = e^{(-\lambda t)}$$

Pode-se mostrar também que a probabilidade de n chegadas ocorrendo no período de comprimento t é dado por:

$$p(n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

conhecida como distribuição de Poisson.

Como vimos em (1), a função cumulativa da distribuição exponencial é dada por:

$$y = 1 - e^{-\lambda t}$$

que pode ser invertida obtendo:

$$\lambda t = -\ln(1-y)$$

onde y é um valor entre 0 e 1.

A P E N D I C E B

P R O B A B I L I D A D E D E N Ã O C O L I S Ã O N O B A R R A M E N T O

Supondo "p" a probabilidade de haver uma mensagem em um EP para transmissão, e (1-p) a probabilidade de não haver a mensagem:

$$\text{Prob \{não colisão\}} = p(1-p)^{n-1} + p(1-p)^{n-1} + \ldots\ldots$$

$$\text{Prob \{não colisão\}} = np(1-p)^{n-1}$$

Onde n é o número de EPs ligado ao barramento.

A P E N D I C E C

R E D E S E M A N E L

Supondo:

w - tempo médio para fazer poll para outra estação.

Nm - número médio de mensagens no poll.

$\overline{x}$ - comprimento médio das mensagens.

R - capacidade do canal de transmissão.

λ - número de mensagens por unidade de tempo.

Podemos ter:

$\frac{Nm\ \overline{x}}{R}$ - tempo requerido para esvaziar o sistema.

$Tc = M\ [Nm\ \overline{x}\ /\ R + w]$ - tempo de ciclo médio da rede.

onde M o número de elementos ligados ao poll.

Tomando $Nm = \lambda Tc$ podemos ter:

$$Tc = \frac{M\ w}{1 - M\ \lambda\ x\ /\ R} \quad \text{segundos}$$

$S = M\ \lambda\ x/R$ a largura de faixa do sistema

$$Tc = \frac{M\ w}{1 - S}$$ segundos, que é o tempo de ciclo médio do sistema de comunicação

ANÁLISE DO ATRASO NO SISTEMA DE COMUNICAÇÃO

W - tempo de espera para a mensagem alcançar o topo da fila para transmissão em um elemento ligado ao sistema de comunicação.

W1 - espera no buffer enquanto o sistema de comunicação serve outro elemento.

W2 - espera enquanto este elemento está sendo servido.

Portanto W = W1+W2

Assim:

$Nm = \lambda\ Tc$ n. médio de pacotes sendo servido.

$\lambda\ Tc\ \bar{x} / R$ - tempo de serviço médio.

$p = \lambda\ \bar{x} / R$

p Tc - tempo de serviço médio por estação.

Tc (1-p) - comprimento médio do tempo que um elemento i espera para ser servido.

Portanto:

$$W1 = \frac{(1 - P)\ Tc}{2} \text{ segundos}$$

ou

$$W1 = \frac{M\ w\ (1 - p)}{2\ (1 - Mp)}$$

$$W2 = \frac{(M\ \lambda)\ \bar{x}^2\ /\ R^2}{2\ (1 - Mp)}$$

Se $S = M\ \lambda\ \bar{x}\ /\ R$

$$W2 = \frac{S\ \bar{x}^2}{2\ \bar{x}\ R\ (1 - S)}$$

Portanto:

$$W = \frac{M\ w\ (1 - S/M)}{2\ (1 - S)} + \frac{S\ \bar{x}}{R\ (1-S)}$$

A P E N D I C E D

R E D E E M A N E L P O R S L O T S

O sistema de análise proposto por Takagi [Takagi,1987], é um sistema multi filas e único servidor cíclico do tipo anel em "slots".

Para estimar o tempo de atraso das mensagens temos:

TMi - atraso médio para transferência de mensagem. Que é o tempo médio de duração desde a chegada da mensagem ao nó Ei até a transmissão para o no destino.

WMi - tempo médio que a mensagem espera na fila até que ela seja servida (tempo da mensagem chegar ao nó Ei até o inicio do serviço do primeiro pacote da mensagem).

WPi - tempo médio de espera do pacote na fila até que ele seja servido.

λi - taxa de chegada de pacotes ao nó Ei por Poisson.

$1/\mu i$ - distribuição exponencial para o tempo de serviço.

Gi -número de pacotes por mensagem.

Com as suposições acima, o tempo médio de espera de mensagem é dado por:

$$WMi = WPi - (Gi - 1).\ ti, \quad i=1,2,....N$$

onde ti é o tempo esperado entre o inicio do serviço de dois pacotes consecutivos em Ei.

$$TMi = WMi + (Gi - 1).\ ti + Tu + Tp$$

onde :

[(Gi - 1). ti] , tempo médio de serviço de todos exceto o último pacote de mensagem;

Tu - tempo de transmissão do último pacote;

Tp - tempo médio de propagação do último pacote de Ei para o destino;

$$TMi = WPi + Tu + Tp, \quad i = 1, 2,... N.$$

Considerando que a carga é idêntica para todos os nós, e que todos os nós estão posicionados à mesma distância no anel, podemos utilizar o resultado do modelo de servidor ciclico limitado do Takagi, assim o tempo médio WPi é dado por:

$$WP = \{ \frac{\lambda\ G\ Tu\ (N + 1) + 2\ G\ (N + 1) - 1}{2\ [1 - \lambda\ G\ Tu\ (N + 1)]} \} Tu$$

e i foi suprimido dada a condição simétrica da rede.

A P E N D I C E E

T A M A N H O D O S M Ó D U L O S E D O F D s i m

MÓDULO		TAMANHO (linhas)
GLOBAL	-	780
TELAS	-	1266
ALTERACA	-	763
EDITOR	-	893
ANALISA	-	525
GLOBAL2	-	84
PROCTF	-	678
PROCEPIN	-	305
PROCREL	-	424
PROCEX	-	1048
PROCBAR	-	84
FDSIMPRI	-	1091
FDSIM	-	1342
TOTAL	-	9283

FDSIM.EXE - 118880 bytes

A P Ê N D I C E F

DESENHOS DOS PROGRAMAS 2, 5, 6, e 7

MODELOS DE REDE DE FILAS DOS PROGRAMAS 2, 5, 6 e 7

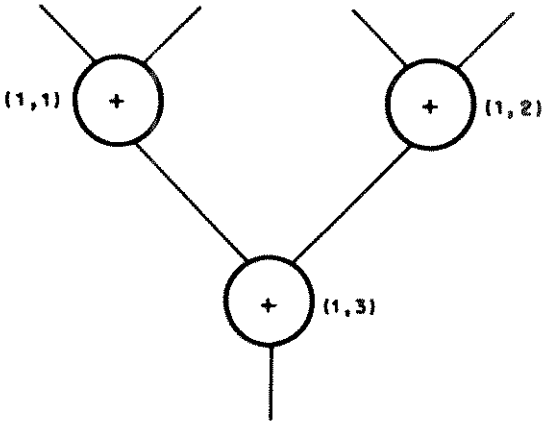


FIGURA F1 - PROGRAMA 2.

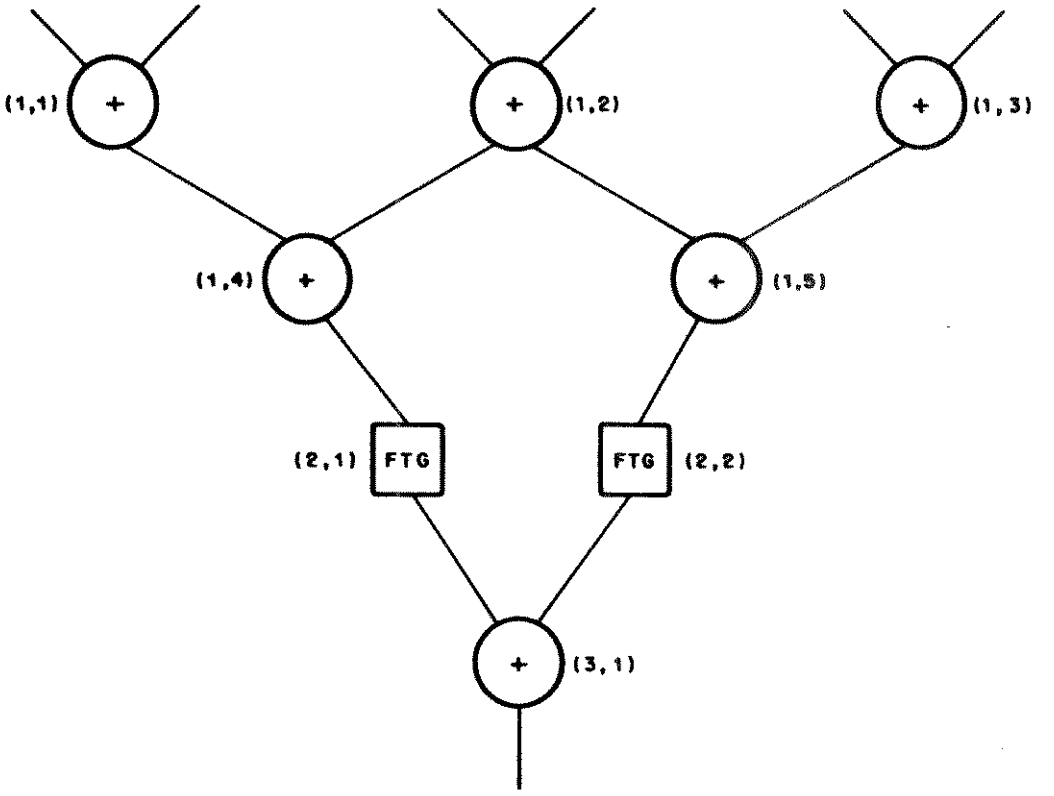


FIGURA F2 - PROGRAMA 5.

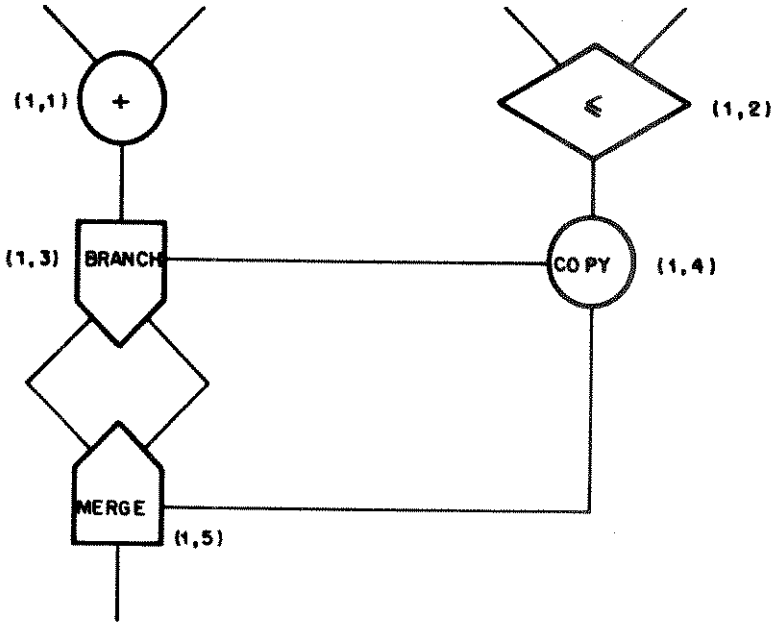


FIGURA F3 - PROGRAMA 6.

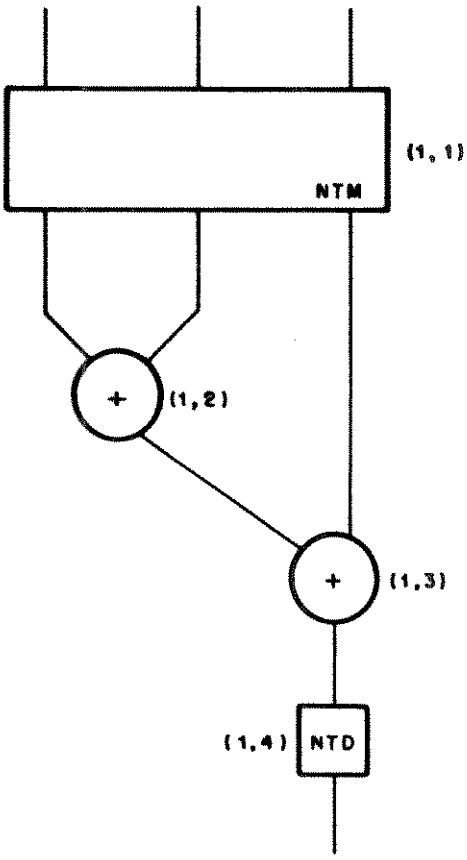


FIGURA F4 - PROGRAMA 7.

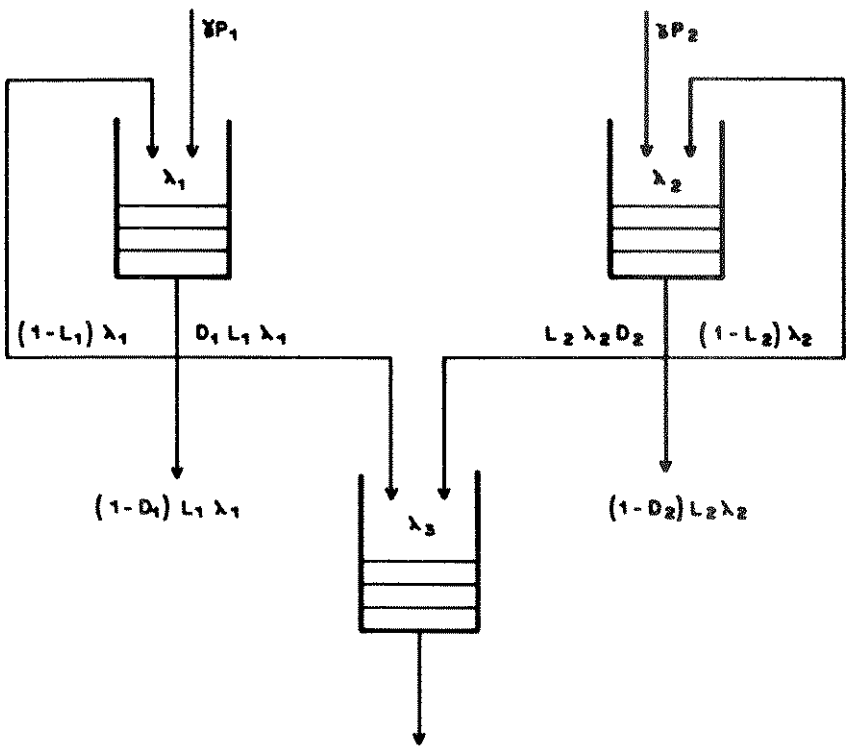


FIGURA F5 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO

PROGRAMA 2.

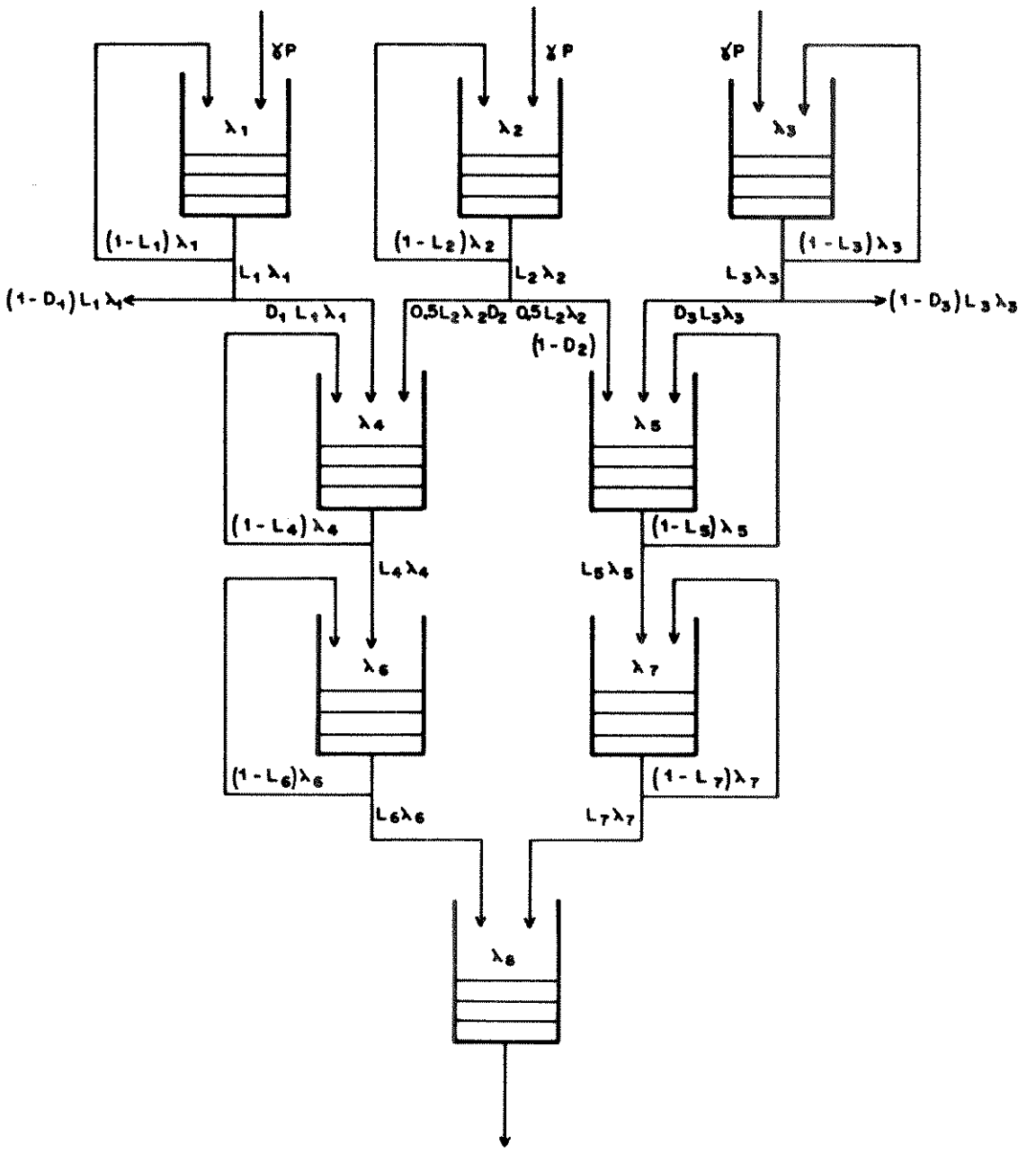


FIGURA F6 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO

PROGRAMA 5.

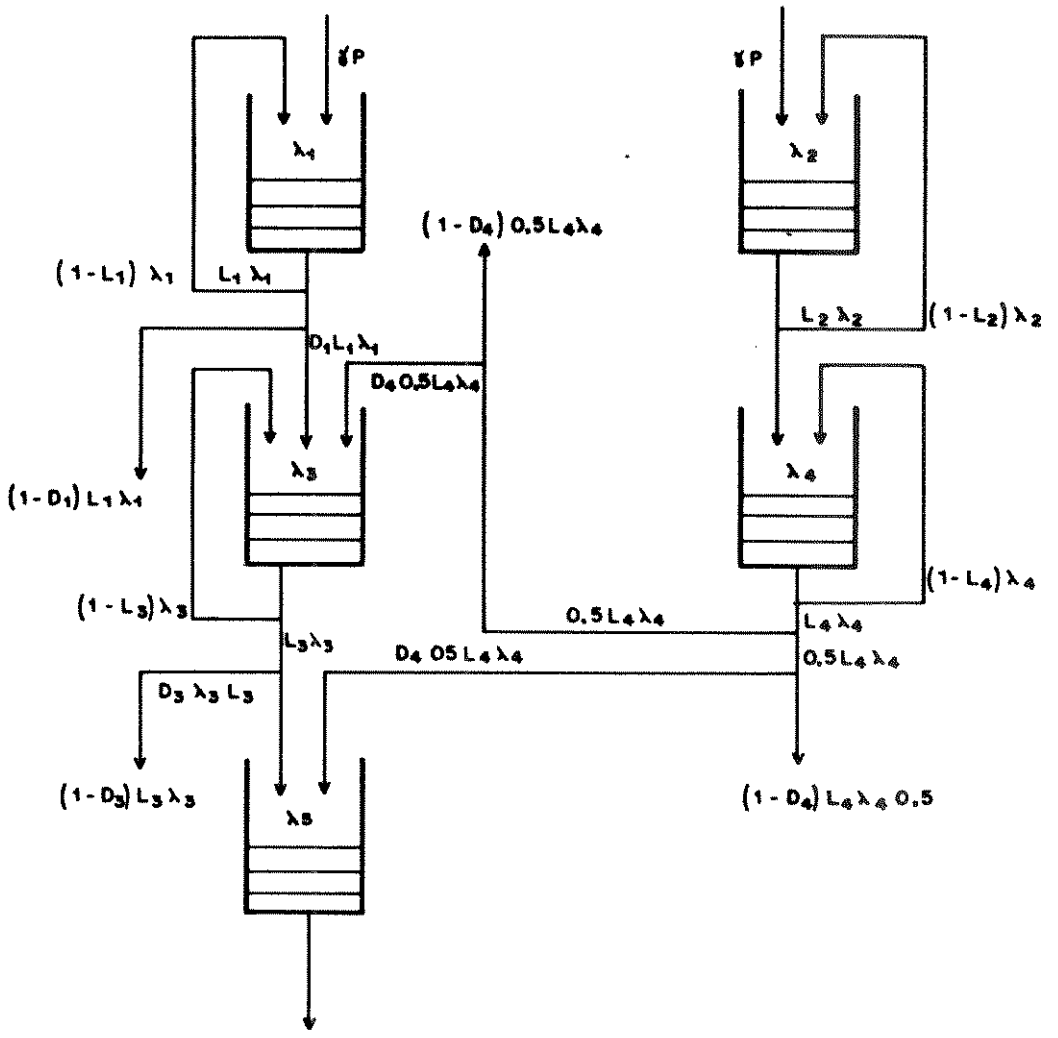


FIGURA F7 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO

PROGRAMA 6.

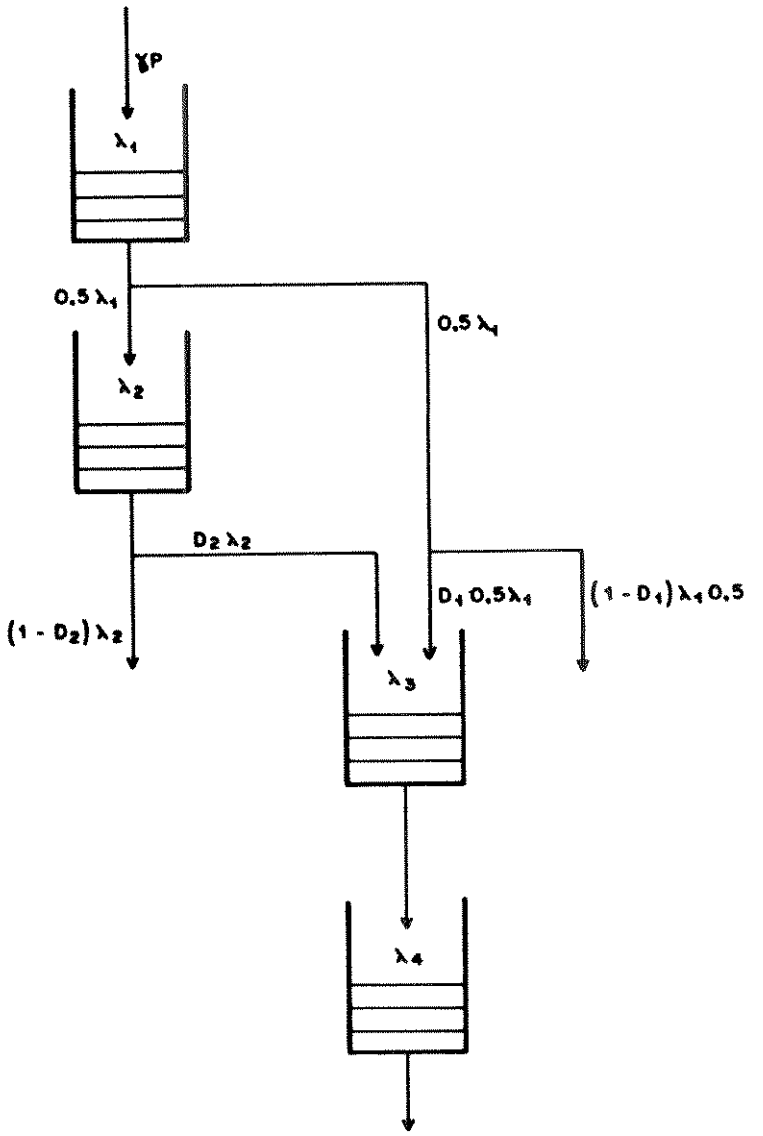


FIGURA F8 - MODELO DE REDE DE FILAS COM REALIMENTAÇÃO

PROGRAMA 7.

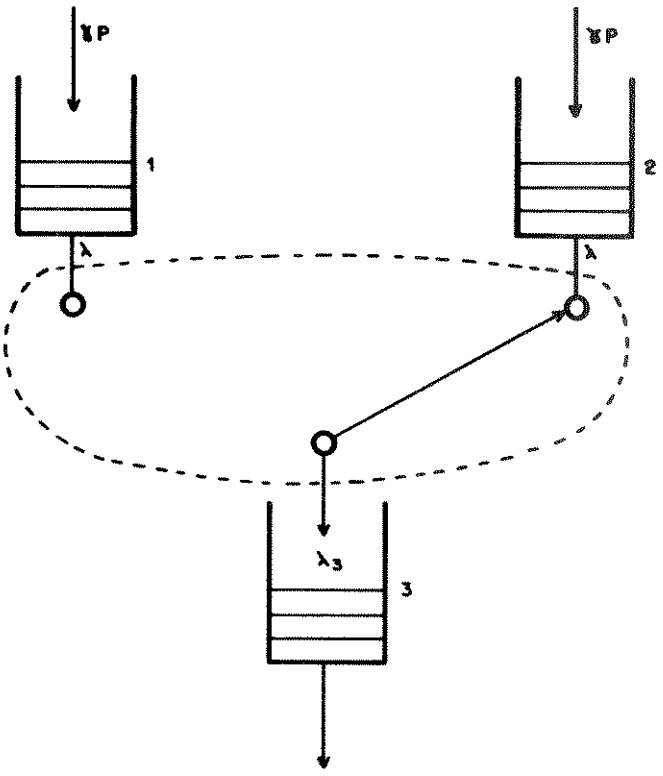


FIGURA F9 - MODELO DE REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 2.

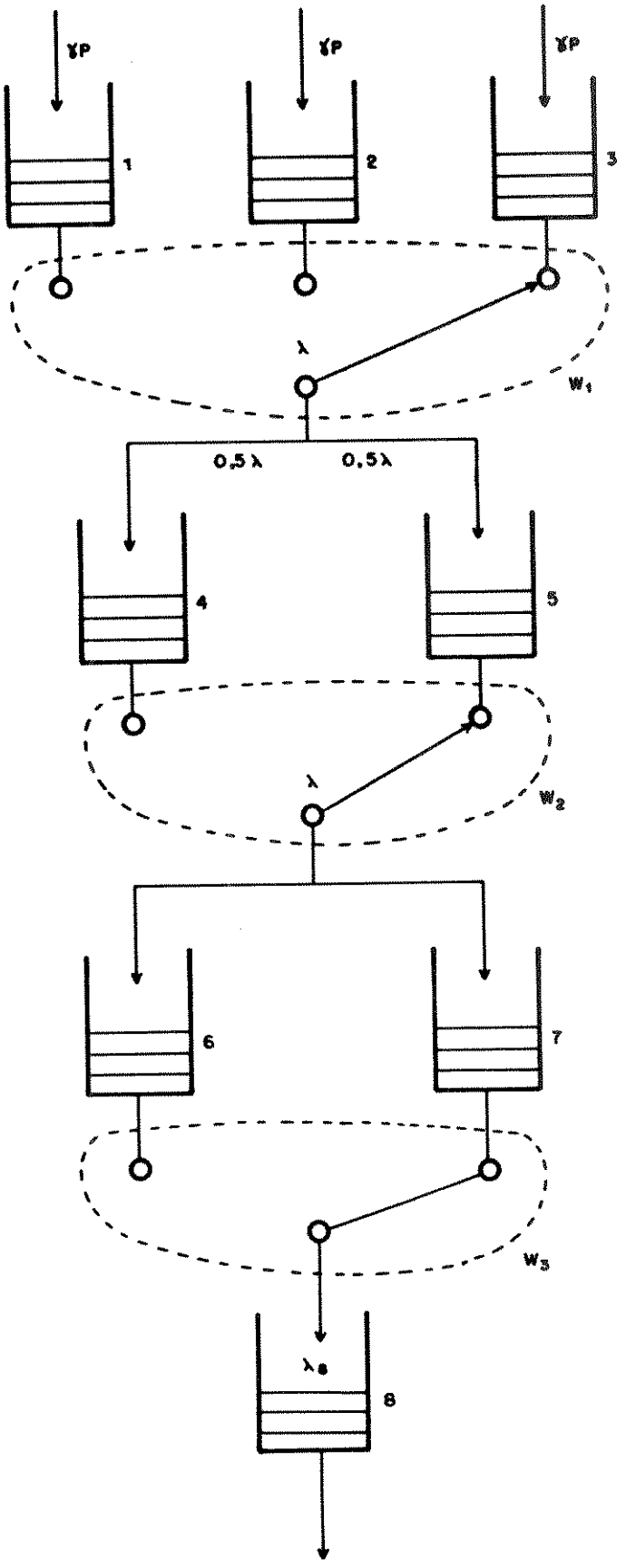


FIGURA F10 - MODELO DE REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 5.

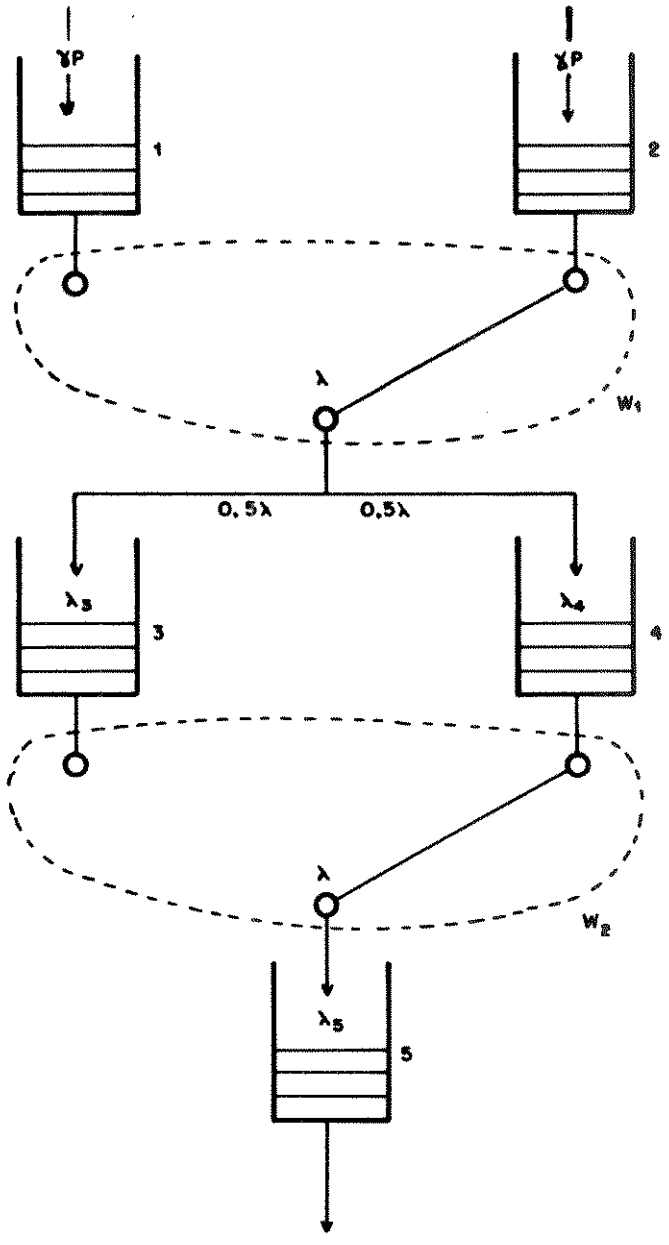


FIGURA F11 - MODELO DE REDE DE FILAS COM MECANISMO DE ACESSO EM ANEL - PROGRAMA 6.

REFERÊNCIAS BIBLIOGRÁFICAS:

ACAMPORA, A.S, et al. A Centralized-bus Architecture for local Area Networks. In: Proceeding of the IEEE International Conference on Communication, 1983, p. 932-938.

ACAMPORA, A.S, HLUCHYJ, M.G. A new local area networks architectures using a centralized bus. IEEE Communications Magazine, v. 22, n. 8, p. 12-21, 1984.

ACKERMAN, W. Data Flow Languages. IEEE Computer, v.18, p.15-25, Feb. 1982.

AGERWALA, T., ARVIND. Data Flow Systems. IEEE Computer, v.18, p.10-13, 1982.

ARVIND, IANNUCCI, R.A. A Critique of Multiprocessing von Neumann Style. In: Proc. of the 10th Annual International Symposium on Computer Architecture, 1983, Stockholm, Sweden, 1983, 10p., p. 426-436.

ARVIND, KATHAIL, V. A Multiple Processor Dataflow Machine That Supports Generalized Procedures. In: Proc. of 8th Annual International Symposium on Computer Architecture, 1981, Minneapolis, USA, May 1981, 11p., pp.291-302.

BEN-ARI, M. Principles of Concurrent Programming. Prentice/Hall, 1982. 172p.

BUEHRER, R., EKANADHAM, K. Incorporating Dataflow Ideas into von Neumann Processors for Parallel Execution. IEEE Transactions on Computers. v. C-36, n. 12, p.1515-1522, Dec. 1988.

BUGATTI, I.D. Computadores a Fluxo de Dados: Aplicações em Central de Comutação Telefônica. CAMPINAS : UNICAMP, 1986. Dissertação (Mestrado em Engenharia Eletrônica) - Faculdade de Engenharia Elétrica, UNICAMP, 1986.

BUGATTI, I.D. Linguagens de Programação para Computadores a Fluxo de Dados. Campinas : UNICAMP, 1986. Relatório Técnico.

BURR, W.E. An Overview of the Proposed American National Standards for Local Distributed Data Interfaces. Communications of the ACM, v.26, p. 554-561, Aug. 1983.

CATTO, A. J. Nondeterministic Programming in a Dataflow Environment. Ingland : University of Manchester, 1981, 187p. Thesis (Doctor of Philosophy), 1981.

CHAMBERS, F.B. et al. Distributed Computing. Flórida : Academic Press, 1984. 324p.

CORSO, D., VERRUA, L. Contention delay in distributed priority networks. Euromicro Journal, n.13, p. 12-29, 1984.

D'AREZZO, V.M., KIRNER, C. Mapeamento de operações em uma máquina a fluxo de dados. In: XXII Congresso Nacional de Informática, 1989, São Paulo, Set. 1989, 8p., p. 186-194.

D'AREZZO, V. M. Mapeamento de Programas a Fluxo de Dados em Ambiente Distribuído. São Carlos: Universidade Federal, 1991, 142p., Dissertação (Mestrado em Ciências da Computação), DC, 1991.

DAVIS, A.L., KELLER, R. M. Data Flow Program Graphs. IEEE Computers. v.18, p.A26-41, Feb. 1982.

DENNIS, J.B. First Version of a Data Flow Produce Language Lecture Notes in Computer Science, v.19, n. 362- 372, 1974.

DENNIS, J.B., MISUNAS, D.P. A Preliminary Architecture for a Basic Data-Flow Processor. In: Annual Symp. Computer Architecture, 2, 1975, Proceedings, 1975. 6p. p.126-132.

DENNIS, J.B., MISUNAS, D.P. The Varieties of Data Flow Computers. In: Proc. of the First International Conference on Distributed Computing Systems, October 1979, 10p., 430-439.

DENNIS, J.B. Data Flow Supercomputer. IEEE Computer, v.13, n. 11, p. 48-56, Nov. 1980.

FINLEY, M. R. Jr. Optical Fibers in Local Area Networks. IEEE Communications Magazine. v.2, n.8, pg 22-35, Aug. 1984.

FLYNN, M. J. Some Computer Organization and Their Effectiveness. IEEE Trans. on Comp. v. C-21, n.9, p. 948-960. Sept. 1972

GAUDIOT, J.L. Methods for Handling Structures in Dataflow Systems. In: Proc. of the 12th International Symposium on Computer Architecture, 1985, Boston, USA, Jun. 1985, 6p., p.352-358.

GHOSAL, D., BRUYAN, L. N. Performance Evaluation of a Dataflow Architecture. IEEE Transactions on Computers, v. 39, n.5, p.615-627, 1990.

GONÇALVES, R.C.M.G. CPER-1: Um computador paralelo distribuído de alto desempenho. São Carlos : UFSCar, 1991, 114p. Dissertação (Mestrado em ciências da computação), UFSCar, 1991.

GORDON, G. System Simulation. New Jersey : Prentice-Hall, 1978. 324p.

GURD, J., WATSON, I. Structuring software for Parallel Execution. I.In: Data-Driven Systems for High Speed Parallel Computing. Computer Design, v. 19, n. 6, p. 91-100, Jun. 1980.

GURD, J., WATSON,I. Hardware Design. II. In: Data-Driven System for High Parallel Computing. Computer Design, v.19, n. 7, p. 97-106, Jul. 1980.

GURD, J.R., et al. The Manchester Dataflow Computer. Communication of the ACM, v. 28, n. 1, p. 34-52, Jan. 1985.

GURD, J.R. et al. Fine-Grain Parallel Computing: The Dataflow Approach In Lecture Notes in Computer Science, 272 (Future Parallel Computers). Spring-Verlag, Berlin, p. 82-152, 1987.

HAMMOND, J. L., O'REILLY, P. J. P. Performance analysis of Local Computer Networks. Massachusetts : Addison-Wesley Publishing Company, 1986. 411p.

HWANG, K., BRIGGS, F.A. Computer Architecture and Parallel Processing. New York : McGraw-Hill Book Company, 1984. 846p.

IANNUCCI, R.A. Toward a Dataflow / von Neumann Hybrid Architecture. In: Proc. of the 15th Annual International Symposium on Computer Architecture, 1988, Honolulu, Hawaii, May 1988, 9p., pp.131-140.

INMOS Transputer Development System - User Manual - Copyright 1985, INMOS limited.

ITO, N. et al. The Architecture and Preliminary Evaluation Results of the Experimental Parallel Inference Machine PIM-D. In: Proc.of the 13th Annual International Symposium on Computer Architecture, July 1986, Tokyo, Japan, 1986, 7p., p. 149-156.

JÉZÉQUEL, J.M., JARD, C. L'expérimentation d'algorithmes distribués sur machines paralleles avec ECHIDNA. France: Institut National de Recherche en Informatique et en Automatique, 1991, 64p. Rapports Technique Nº603.

JOSHI, S., LYER, V. New standards for Local Networks push upper limits for ligthware data. Data Communications, p.127-138, July 1984.

KAR, R. P. Data-Flow Multitasking. Dr. Dobb's Journal, p. 16-24, Nov. 1989.

KIRNER, C. Suporte para Desenvolvimento de Sistemas Distribuídos : uma implementação. In: Anais do IV Congresso da Sociedade Brasileira de Computação - XI SEMISH, 1984, Viçosa, MG, Jul. 1984, 12p., p.109-121.

KIRNER, C., MARQUES, E. Suporte para Desenvolvimento de Sistemas Distribuídos Baseado num Barramento Paralelo Centralizado. In: Anais do V Congresso da Sociedade Brasileira de Computação XI CLAI - XII SEMISH, 1985, Porto Alegre , RS, jul. 1985. 12p., p. 423-435.

KIRNER, C. Desenvolvimento de suporte basico para sistemas operacionais distribuidos. Rio de Janeiro : Universidade Federal, 1986, 232p. Dissertação (Doutorado em Computação) COPPE, 1986.

KIRNER, C., MARQUES, E. Design of Distributed System Suport Based in a Centralized Parallel Bus. Computer Architecture News, v. 14, n.4, p. 15-26, Sep. 1986.

KIRNER, C., BAEMA, W.C. Design of a Fault-Tolerant Environment for Implementation of Distributed Operating Systems. In: Proc. of 21th National Conference on Informatics, 1988, Rio de Janeiro, Brazil, Aug. 1988, 9p., pp.686-695.

KIRNER, C., MENDES, S.B.T. Sistemas Operacionais Distribuidos. Campus, 1988, 184p.

KIRNER, C. Design of a Recursively Structured Parallel Computer. In: Proc.of the 17th Annual Computer Science Conference - ACM, Feb. 1989, Louisville, USA.

KLEINROCK, L. Queueing Systems. Volume I: Theory. New York : John Wiley & Sons, 1975. 417p.

KRUSE, R.L. Data Structures & Program Design. Prentice Hall, 1984, 486p.

KRVATRACHUE, B., LEWIS, T. Grain Size determination for parallel processing. IEEE Software, v.5, n.1, p. 23-32, 1988.

LINDSAY, D.C. Local area networks: bus and ring vs. coincident star. Computer communication review, v. 12, n.3-4, p. 83-91. 1982.

MARQUES, E. Projeto de um Barramento Compacto para Sistemas Distribuídos. São Carlos : UFSCar, 1984. Relatório Técnico.

MARQUES, E., KIRNER, C. Projeto de um Subsistema de comunicação para aplicações em tempo real. In: XX Congresso Nacional de Informática, VII Feira Internacional de informática, 1987, São Paulo, SP, 1987, 7p., p. 777-784.

MARQUES, E., KIRNER, C., OBAC RODA, V. Mecanismos de tolerancia a falhas num subsistema de comunicacao baseado em barramento paralelo centralizado. In:Simposio em sistemas de computadores tolerantes a falhas, 2., 1987, Campinas, SP, 1987. 18p., p. 131-149.

MARQUES, E. Projeto de uma Rede Local de Computadores de Alta Velocidade. São Carlos: Universidade de São Paulo, 1988, 111p. Dissertação (Mestrado em Computação) ICMSC, 1988.

MARQUES, E., KIRNER, C. Projeto de um Elemento de Processamento de um Computador Maciçamente Paralelo para Execução a Fluxo de Dados. São Paulo : USP,1992a. Exame de Qualificação (Doutorando em Engenharia de Computação e Sistemas Digitais) - POLI, USP, 1992a. 90p.

MARQUES, E., KIRNER, C. Projeto de uma Unidade de Mathing Store para execução de Progamas a Fluxo de Dados num Computador Massivamente Paralelo. In: XII Congresso da Sociedade Brasileira de Computação,1992b, Rio de Janeiro,1992b, p. 30-44.

MENDELSON, B., SILBERMAN, G.M. Mapping Dataflow Programs on a VLSI Array of Processors. In: Proc. of the 14th Annual International Symposium on Computer Architecture, 1987, Pittsburgh, USA, Jun. 1987, 8p., pp.72-80.

MOTOYAMA, S., ARNATES, M. P. C. A Proposal for a Metropolitan Area Network with priority for the Synchronous Traffic. Campinas : UNICAMP , 1992, 10p. Relatório Técnico.

O'BRIEN, S. Turbo Pascal 6. New York : McGraw-Hill Book Company, 1991. 690p.

PRESSMAN, R. S. Software Engineering. New York : McGraw-Hill Book Company, 1987. 567p.

RAKOTOARISOA, H., MIUSSI, P. PARSEVAL: PARallélisation sur réseaus de Tranputers de Simulations pour L'EVALuation de performances. France: Institut National de Recherche en Informatique et en Automatique, 1991, 39p. Rapports Technique Nº131.

RANDEL, B. Fault Tolerance & System Structuring. University of Newcastle upon Tyne/Computing Laboratory : England, Dec. 1983. Technical Report n.189.

RUMBAUGH, J. A Data Flow Multiprocessor. IEEE Transactions on Computers. p. 138-146, Feb. 1977.

SATO, M. et. al. - Thread-Based Programming for the EM-4 Hybrid Dataflow Machine. Annual International Symposium on Computer Architecture, Austrália, 1992. p. 146-155.

SHIMADA, T. et. al. Evaluation of a Prototype Dataflow Processor of the Sigma-1 for Scientific Computations. In: Proc.of the 13th Annual International Symposium on Computer Architecture, Jun. 1986, Tokyo, Japan, 1986, 8p., p. 226-234.

SHIPPEN, G.B., ARCHIBALD, J.K. A Tagged Token Dataflow Machine for Computing Small, Iterative Algoritms. Computer Achitecture News, ACM, v. 15, n. 6, p. 9-18, Dec. 1987.

SIEWIORECK, D. P. The theory and practice of reliable system design. Massachusetts, Digital Press, 1982. 753p.

SRINI, V.P. A Fault-Tolerant Dataflow System. IEEE Computer, v. 18, n. 11, p. 54-68, Mar. 1985.

SRINI, V.P. An Architectural Comparison of Dataflow Systems. IEEE Computer. v. 18, p.68-88, Mar. 1986.

STRECKER, W. D. CI: A high speed Interconnect for Computer and Mass Storage Controllers. in: Proc. of the Eighth International Fiber Optics Communications and Local Area Networks Exposition, pg. 246-250, 1984.

TAKAGI, H. Analysis of Polling Systems. London : MIT Press, 1985. 197p.

TANENBAUM, A.S. Computer Networks. Englewood Cliffs, N. J. Prentice-Hall, Inc., 1981. 517p.

TAUB, D.M. Improved control acquisition scheme for the IEEE 896 futurebus. IEEE Micro, v. 7, n. 3, p. 52-62, 1987.

TEIXEIRA, C.A.C., TREVELIN, L.C. Protocolo de Acesso e Superação de Falhas em Redes com Topologia Estrela Coincidente. In: Anais do II Simpósio sobre Redes de Computadores. 1984, Campina Grande, Paraíba, UFPb, abr. 1984. 14p., p.16.1-16.15.

THORNTON, J. E. Back-End Network Approaches. IEEE Computer, p.10-17, Feb. 1980.

TISCHER, M. Turbo Pascal 6.0 - simples e rápido. São Paulo : Makron Books, 1992. 160p.

TRELEVEAN, P.C., BROWNBRIDGE, D.R., HOPKINS, R.P. Data-Driven and Demand-Driven Computer Architecture. ACM Computer Surveys, v. 14, n. 1, p. 93-143. Mar. 1982.

VEEN, A. H. Dataflow Machine Architecture. Computing Surveys, ACM, v. 18, n. 4, p. 365-396, Dec. 1986.

WATSON, I., GURD, J. A Pratical Data Flow Computer. IEEE Computer, v.18, p. 51-57, February 1982.

TRABALHOS PUBLICADOS PELO AUTOR E

PRÊMIO OBTIDO

SILVA, J.L. Análise e Projeto de um Subsistema de Comunicação para uma Rede de Computadores. São Carlos : USP, 1986. Dissertação (Mestrado em ciências da computação) - Intituto de Ciências Matemática de São Carlos, USP, 1986. 110p.

KIRNER, C., SILVA, J.L. Protocolo de comunicacao em um sistema distribuido baseado num barramento paralelo centralizado. In: V Simposio brasileiro de redes de computadores, 1987, Sao Paulo, SP, 1987. 20p., p. 250-270.

SILVA, J., KIRNER, C. Desenvolvimento do software basico de um sistema distribuido para aplicacoes em tempo real. In: XX Congresso nacional de informatica. 1987, Sao Paulo, SP, 1987, 7p., p.683-690.

SILVA, J.L., NORDE, C. Sistema Operacional Distribuído. São Carlos: UFSCar, 1987, 75p. Relatório Técnico.

SILVA, J.L. Sistemas de Processamento Convencionais de Alta Velocidade. São Carlos: UFSCar, 1988, 68p. Relatório Técnico.

SILVA, J.L. Computadores de Arquitetua "Dataflow". São carlos: UFSCar, 1988, 60p. Realtório Técnico.

MARQUES, E., SILVA, J.L., KIRNER, C. An Experience in Developing High-Speed Local-Area Networks. In: Proc.of the 8th SCCC International Conference on Computer Science, Jul. 1988, Santiago, Chile, 1988, 11p., p. 87-98.

SILVA, J.L., KIRNER, C. Development of the Basic Software of a Tagged-Token Data-Flow Machine. In: IX international Conference of the Chilean Computer Science Society, 1989a, Santiago, Chile, 1989a, 9p., p. 245-254.

SILVA, J.L., KIRNER, C. Fault-Tolerant Data Flow Processing in a Parallel Computer. In: X international Conference of the Chilean Computer Science Society, 1990, Santiago, Chile, 1990, 9p., p. 245-254.

GRECO, J.A., SILVA, J.L. Protocolo de Comunicação para a programação a fluxo de dados em um Computador Paralelo. São Carlos : UFSCar, 1990a. 10 p. Relatório técnico.

GRECO, J.A., SILVA, J.L. Simulador de um Supervisor para programas a fluxo de dados na Máquina a fluxo de dados dinâmica (MFDD). São Carlos : UFSCar, 1990b. 71 p. Relatório técnico

SILVA, J.L., KIRNER, C., MOTOYAMA, S. Fault Tolerant Data Flow Processing in a Hierarchical Parallel Buses Computer. Submetido ao: The Twenty Third Annual International Symposium on Fault-Tolerant Computing, a realizar-se de 22 a 24 de Julho de 1993 em Toulouse na França.

PRÊMIO OBTIDO

IV PRÊMIO NACIONAL DE INFORMÁTICA

Secretaria Especial de Informática

Moddata S.A.

Fundação Roberto Marinho

1º Lugar - Categoria Software

com o trabalho:

SILVA, J.L., KIRNER, C. Suporte para Programação "Dataflow" em um computador paralelo. In: XXII Congresso Nacional de Informática, IX Feira Internacional de Informática, 1989b, São Paulo, SP, 1989b, 8p., p. 131-143.