UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DA ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

IMPLEMENTAÇÃO PARALELA DO ALGORITMO LINHAS E SUPERFÍCIES ESCONDIDAS EM MÁQUINAS MIMD FRACAMENTE ACOPLADAS

Por : Diego A. Aracena Pizarro
Orientador: Prof. Dr. Clésio Luis Tozzi

TESE APRESENTADA À FACULDADE DE ENGENHARIA ELÉTRICA, DA UNIVERSIDADE ESTADUAL DE CAMPINAS, COMO PARTE DOS REQUISITOS EXIGIDOS PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA.

AGOSTO 1991

A

Patricia

Diego

Patricio

Alvaro

Sebastián e

meus pais

com muito carinho.

AGRADECIMENTOS

Toda minha gratidão ao Prof. Dr. Clésio Luis Tozzi, por sua dedicação e apoio durante todo o período de estudos, e principalmente nos momentos mais dificeis demostrando ser mais que um professor.

Agradeço à minha familia, que apesar de distante fez-me sentir seu carinho e confiança.

Meus reconhecimentos a todos os colegas e amigos que compartilharam conhecimentos e sabedoria, em especial ao grupo de trabalho de visão computacional e processamento paralelo, Antonio, Olga, Fernando e Eduardo pelas valiosas sugestões e aportes importantes na confecção deste trabalho.

Muito obrigado a Josué Viera, Sergio Haffner, Mario Zamorano, Marcelo Jara e a todos aqueles que me brindaram sua amizade e companheirismo.

Devo agradecer também às seguintes instituições:

Ao Instituto de Automação do Centro Tecnológico para Informática (C.T.I.), pela gentileza de permitir a utilização de seus recursos computacionais, de significativa importância no desenvolvimento do trabalho.

À UNICAMP pela oportunidade oferecida.

À Universidade de Tarapacá, pela licença de trabalho concedida.

Finalmente agradeço à minha esposa Patricia, por todo seu apoio, paciência e compreensão.

RESUMO

No presente trabalho realiza-se o estudo do comportamento da proposta para a implementação paralela do algoritmo "Linhas e Superfícies Escondidas", utilizando a técnica do pintor, podendo ser inserido no contexto de computação gráfica e processamento paralelo. O trabalho foi dividido em três etapas principais: Na primeira etapa, apresenta-se o estudo do algoritmo e a proposta para a implementação paralela, realizando a análise dos módulos e fluxo dos dados entre os módulos. Na segunda etapa apresenta-se o desenvolvimento do sistema que executa a aplicação em máquinas MIMD fracamente acopladas, utilizando o mapeamento de várias topologias na placa Inmos IMS B008, com um número reduzido de processadores. Na terceira etapa apresenta-se o desenvolvimento do simulador do sistema multiprocessador, que permitiu realizar o estudo do comportamento do algoritmo, em um maior número de processadores, considerando diferentes particionamentos, para os quais apresentam-se os resultados experimentais obtidos.

I N D I C E

CAPÍTULO 1
INTRODUÇÃO

CAPÍTULO 2
ANÁLISE DA APLICAÇÃO E PROPOSTA DE SOLUÇÃO

CAPÍTULO 3
CONSIDERAÇÕES GERAIS SOBRE PROCESSAMENTO PARALELO

CAPÍTULO 4

PROJETO E IMPLEMENTAÇÃO DO "SISTEMA ORIGINAL"

CAPÍTULO 5
PLANEJAMENTO E IMPLEMENTAÇÃO DO SIMULADOR

CAPÍTULO 6
ANÁLISE DE RESULTADOS

CAPÍTULO 7
CONCLUSÕES

CAPÍTULO 8
REFERÊNCIAS

ANEXO A
ALGORITMOS DE COMUNICAÇÃO

CAPÍTULO 1

INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS E OBJETIVOS.

Este trabalho tem como objetivo o estudo da utilização de processamento paralelo para a solução do algoritmo "Linhas e Superfícies Escondidas" (L.S.E.).

As seguintes condições devem ser observadas para a aplicação:

a) O algoritmo de L.S.E. adotado como solução é apresentado por Harrington [HARRI,1986] e implementado em forma sequencial por Regonessi [REGO,1990]. Além da implementação sequencial o autor propõe também a implementação paralela do algoritmo, realizando um estudo de particionamento de tarefas e extraindo medidas dos tempos de execução para cada um dos módulos que compõem esse algoritmo.

b) O estudo feito por Regonessi [REGO,1990] foi realizado em condições ideais, sem considerar os aspectos reais da execução de um algoritmo em paralelo, utilizando um sistema multiprocessador.

c) No estudo da aplicação foi observado que, na implementação do algoritmo selecionado para tratar o problema de "Linhas e Superfícies Escondidas", existem módulos do algoritmo cujo tempo de processamento é alto, dependendo da complexidade do objeto. Consequentemente, pelo uso do processamento paralelo é possível reduzir significativamente o tempo e, em alguns casos, obter respostas próximas ao tempo real, isto é, tempo de execução menor ou igual a 1/30 s para televisão ou 1/24 s para cinema.

Para poder realizar o estudo do algoritmo L.S.E., utilizando processamento paralelo, o problema foi dividido em duas partes :

1. Estudo do algoritmo L.S.E. proposto para a implementação paralela e apresentado por Regonessi [REGO,1990].

2. Implementar um "Software" que permita estudar o comportamento do algoritmo, citado no item 1, empregando um sistema multiprocessador real que considere os tempos de execução dos módulos do algoritmo L.S.E., simulando-os por retardos de tempo, e outros parâmetros tais como tempo de comunicação, contenção, interrupção, etc.

O "Software" mencionado no item 2, foi implementado num sistema multiprocessador do Centro Tecnológico para Informática (CTI), que conta com poucos processadores. Especificamente, foi utilizada uma placa Inmos IMS B008 com nove processadores "Transputers", conectada a um IBM PC. A construção do "Software" permitiu estabelecer os processos e tempos a serem considerados na construção de um simulador do sistema multiprocessador para um número maior de processadores.

Com base nas considerações apresentadas os objetivos para este trabalho podem ser resumido da seguinte maneira :

Objetivo geral : Realizar o estudo do comportamento do algoritmo L.S.E. utilizando máquinas MIMD fracamente acopladas (memória distribuída).

Objetivos específicos :

I) Criar um sistema que permita executar o algoritmo L.S.E. numa máquina fracamente acoplada, com um número limitado de processadores, simulando o tempo de execução das

tarefas por retardo de tempo equivalente.

Esse sistema, que é uma simulação do problema real, foi denominado, neste trabalho, de "sistema original", e foi desenvolvido para avaliar os processos envolvidos na execução do problema no sistema multiprocessador, tais como: escalonamento de tarefas, recepção de tarefas, rotina de comunicação, operação dos processos em forma concorrente e forma de operação do processador utilizado como módulo de processamento (transputer). Isso permitiu estabelecer quais os processos e quais os tempos deveriam ser considerados para a implementação do simulador;

II) Construir um simulador que permita alcançar o objetivo geral com um número maior de processadores. Como o "sistema original" permite estudar o comportamento com poucos processadores, foi necessário procurar uma estratégia que permitisse extrapolar a execução do "sistema original" para um número maior de processadores.

A estratégia selecionada consistiu em implementar um simulador de eventos, sincronizado por eventos. Os objetivos para a construção do simulador foram os seguintes :

a) Permitir o estudo do comportamento da aplicação mencionada, a fim de obter a melhor escolha para a execução dentre diferentes topologias;
b) Permitir diferentes níveis de particionamento de tarefas a serem executadas em forma paralela;
c) Empregar o Transputer para aproveitar as vantagens do Hardware e Software, assim como a linguagem de programação OCCAM2 e de ferramentas que facilitem seu desenvolvimento;
d) Permitir extrapolar a execução a um número maior de processadores, ligados mediante uma rede de interconexão, permitindo diferentes topologias.

1.2 ORGANIZAÇÃO DO TEXTO.

No segundo capítulo, aspectos do algoritmo L.S.E. são apresentados para permitir a compreensão do problema a ser resolvido. Nele se faz uma análise dos módulos que compõem o algoritmo L.S.E., tipo de gerenciamento necessário e quais particionamento que podem ser feitos.

No terceiro capítulo, são apresentados de forma suscinta os aspectos de processamento paralelo que têm relação direta com o desenvolvimento do trabalho. São mencionados os diferentes tipos de arquiteturas MIMD, as diferentes formas de interconexão, os parâmetros de desempenho mais comuns, etc. Ainda em relação a esses tópicos, no anexo A apresenta-se o desenvolvimento dos algoritmos de comunicação das topologias utilizadas.

No quarto capítulo, são descritos o projeto e a implementação do "sistema original" e, também, as ferramentas empregadas e as configurações implementadas com os recursos disponíveis.

No quinto capítulo, são descritos o projeto e a implementação do simulador de eventos. Apresenta-se, ainda, as análises da implementação realizada e da confiabilidade da execução em comparação com a operação do "sistema original".

No sexto capítulo, são apresentadas e analisadas as medidas realizadas com diferentes níveis de particionamentos da proposta para a implementação paralela do algoritmo L.S.E., considerando diferente número de processadores e topologias.

Finalmente, no sétimo capítulo, são apresentadas as conclusões resultantes do trabalho desenvolvido.

CAPÍTULO 2

ANALISE DA APLICAÇÃO E PROPOSTA DE SOLUÇÃO.

2.1 INTRODUÇÃO.

Neste capítulo será realizado um estudo da proposta para implementação paralela do algoritmo "Linhas e Superfícies Escondidas" (L.S.E). O algoritmo, considerado aqui foi proposto por Harrington [HARRI,1986], e sua implementação sequencial foi desenvolvida em forma prática por Regonessi [REGO,1990]. Nesse trabalho a autora propõe uma implementação paralela do algoritmo e um estudo de seu comportamento, quantificando parâmetros de desempenho ideais, a partir da medição do tempo de diferentes tarefas e sub-tarefas que existem no algoritmo, sem levar em conta os problemas reais, como custos de comunicação, contenção, alocação, distribuição, etc.

O nosso objetivo dentro deste trabalho foi reavaliar o paralelismo e otimizar a proposta feita por Regonessi [REGO,1990], simplificando os módulos e fazendo uma análise de cada um deles, a fim de caracterizar a sequência geral envolvida, estabelecer as relações entre os módulos e verificar o fluxo dos dados entre módulos e sub-módulos. Essa análise servirá como ponto de partida na proposta da estratégia de solução do problema, por meio da utilização do processamento paralelo.

2.2 MÓDULOS QUE COMPÕEM O ALGORITMO LINHAS E SUPERFÍCIES ESCONDIDAS.

A figura 2.1 apresenta o diagrama em blocos do algoritmo "Linha e Superfícies Escondidas" para implementação sequencial. Esse algoritmo compreende as seguintes fases:

- Remoção das faces visíveis;
- Decomposição de polígonos, utilizando o triângulo como primitiva básica para prosseguimento dos cálculos;

- Geração da lista definitiva de prioridades em função da distância do observador (classificação geométrica).

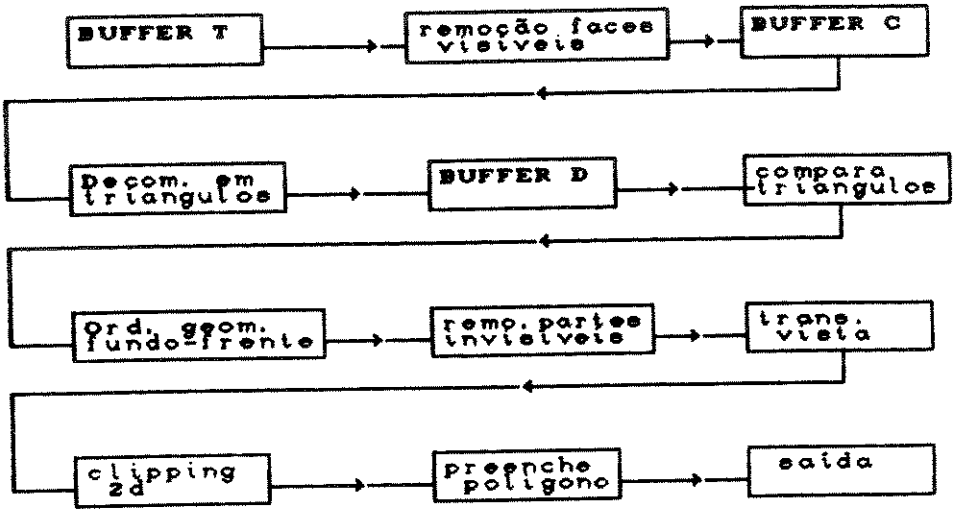


fig. 2.1 Diagrama em blocos do algoritmo "Linha e Superficies Escondidas"

A descrição inicial das faces do objeto está armazenada no buffer T. Essas faces são processadas pelo bloco de remoção de faces, descartando as faces não visíveis e armazenando no buffer C as faces visíveis. Cada face armazenada no buffer C é lida pelo processo de decomposição em triângulos, armazenando-os no buffer D. No processo de comparação entre triângulos, realiza-se uma análise para obter a posição relativa dos triângulos, estabelecendo-se a existência ou não de sobreposição entre eles, ou seja, se o triângulo está em sua frente ou atrás. O resultado do processo de comparação é registrado na estrutura de dados que por sua implementação fornece uma ordenação imediata dos triângulos. A etapa de remoção das partes invisíveis faz o tratamento dos triângulos que estão no fundo e deteta os segmentos que ficam ocultos pelos triângulos que estão em sua frente. Os segmentos visíveis são enviados ao bloco de visualização, representados na figura pelos blocos de transformação, clipping, preencher poligono e saída.

Uma explicação mais detalhada da implementação sequencial do algoritmo L.S.E. é apresentada nos trabalhos de Harrington [HARRI,1986] e Regonessi [REGO,1990].

2.3 ANÁLISE DA PROPOSTA DE IMPLEMENTAÇÃO PARALELA DO ALGORITMO.

Na presente implementação foi considerado que as faces do objeto são inicialmente entregues num arquivo de dados. Estas faces são lidas e posteriormente processadas em paralelo de acordo com a sequência dos módulos, que serão apresentados nas seções a seguir.

2.3.1 REMOÇÃO DAS FACES VISIVEIS.

A figura 2.2 mostra o fluxo paralelo do módulo de faces visíveis. Dado um objeto com n faces, se assumirmos um sistema com n processadores, de modo que cada processador processe uma face, em cada um deles existirão os seguintes sub-processos:

a) Detecção dos vértices na face;
b) Determinação do vértice mais à esquerda;
c) Determinação da direção do vetor de projeção, que consiste em realizar a diferença entre o vetor do centro da perpectiva de projeção e o vértice mais à esquerda;
d) Definição da diferença entre o vértice mais à esquerda e dos vértices que estão antes do vértice mais à esquerda;
e) Definição da diferença entre os vértices que existem após o vértice mais à esquerda e o vértice mais à esquerda;
f) Realização do produto vetorial entre os vetores resultantes nos itens (d) e (e);
g) Realização do produto escalar do vetor normal obtido no item (f) e o vetor de projeção obtido no item (c);
h) Avaliação do valor resultante obtido no item (g), que

indicará se a face é visível ou não.

A)

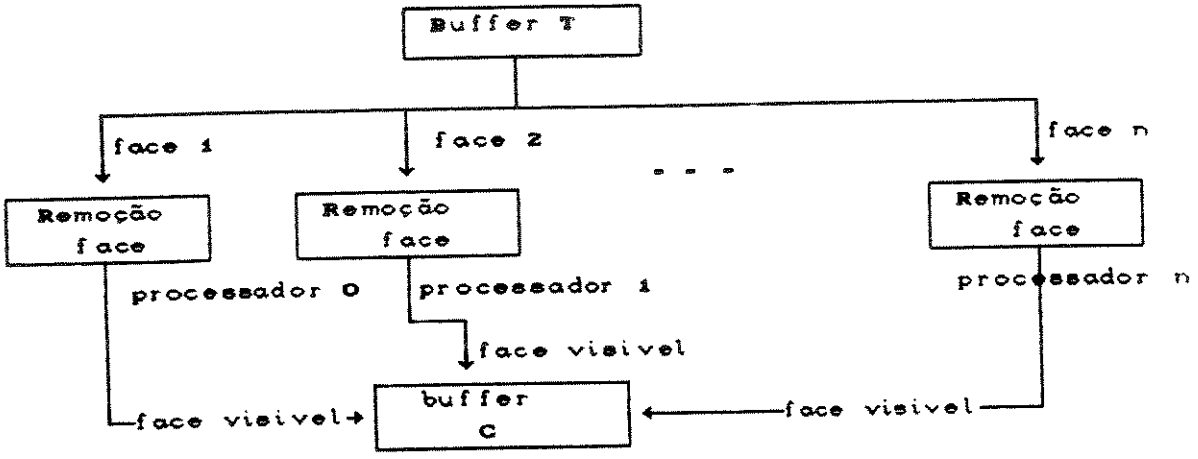


fig.2.2 Fluxo paralelo da remoção de faces.

2.3.2 DECOMPOSIÇÃO EM TRIÂNGULOS.

A figura 2.3 mostra o fluxo paralelo dos módulos de Decomposição das faces visíveis em triângulos. As operações desse processo são as seguintes :

- Cada face visível pronta no buffer C é entregue a um processador para sua decomposição em triângulos;
- Cada processador entrega ao buffer D os triângulos resultantes da face processada;
- Esse processo finaliza quando todas as faces são decompostas em triângulos. Portanto o buffer D deve armazenar o número total de triângulos a serem comparados no processo seguinte.

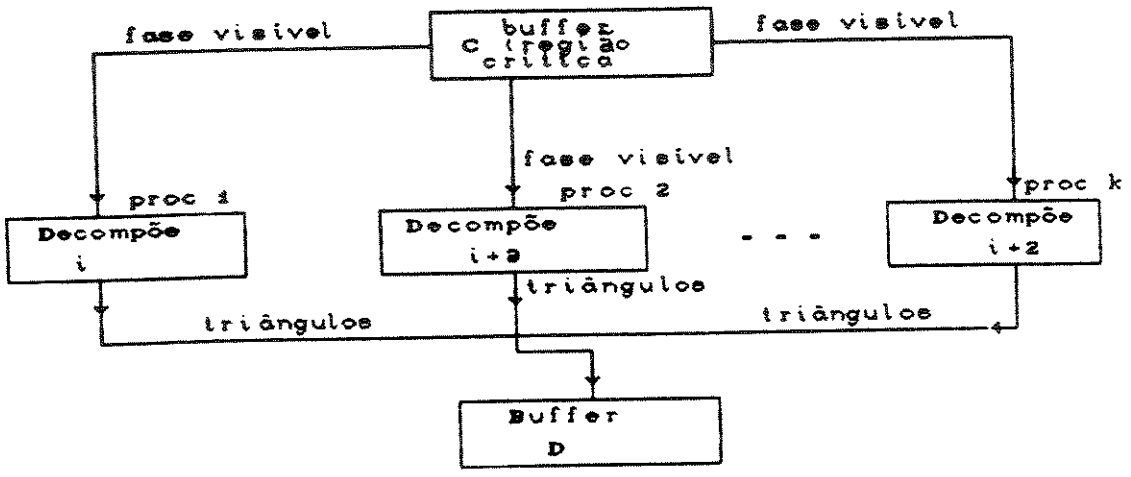


fig.2.3 processo paralelo p/decomposição de faces.

Na proposta feita por Regonessi [REGO,1990], o buffer C é apresentado como solução para armazenar as faces visíveis do objeto. No entanto se o processo de remoção e decomposição de faces fossem feitos no mesmo processador, o buffer C poderia ser eliminado.

2.3.3 COMPARAÇÃO ENTRE TRIÂNGULOS.

Nesse processo realiza-se a comparação entre dois triângulos para ver se existe sobreposição entre eles. Se existe sobreposição deve-se determinar a posição relativa dos mesmos.

No processo de comparação de dois triângulos existem três funções básicas para estabelecer o posicionamento :

1.- Minimax(i,j). Compara os triângulos em relação as coordenadas xy para verificar se existe sobreposição;
2.- Z-minimax(i,j). Verifica qual triângulo está na frente, em termos das coordenadas z;
3.- Tri-compare(i,j). Se as condições anteriores não forem suficientes para a determinação de profundidade, devem-se comparar os lados para determinar um ponto (x,y) comum aos dois triângulos, tal que a ordem de profundidade seja determinada

pelas coordenadas z.

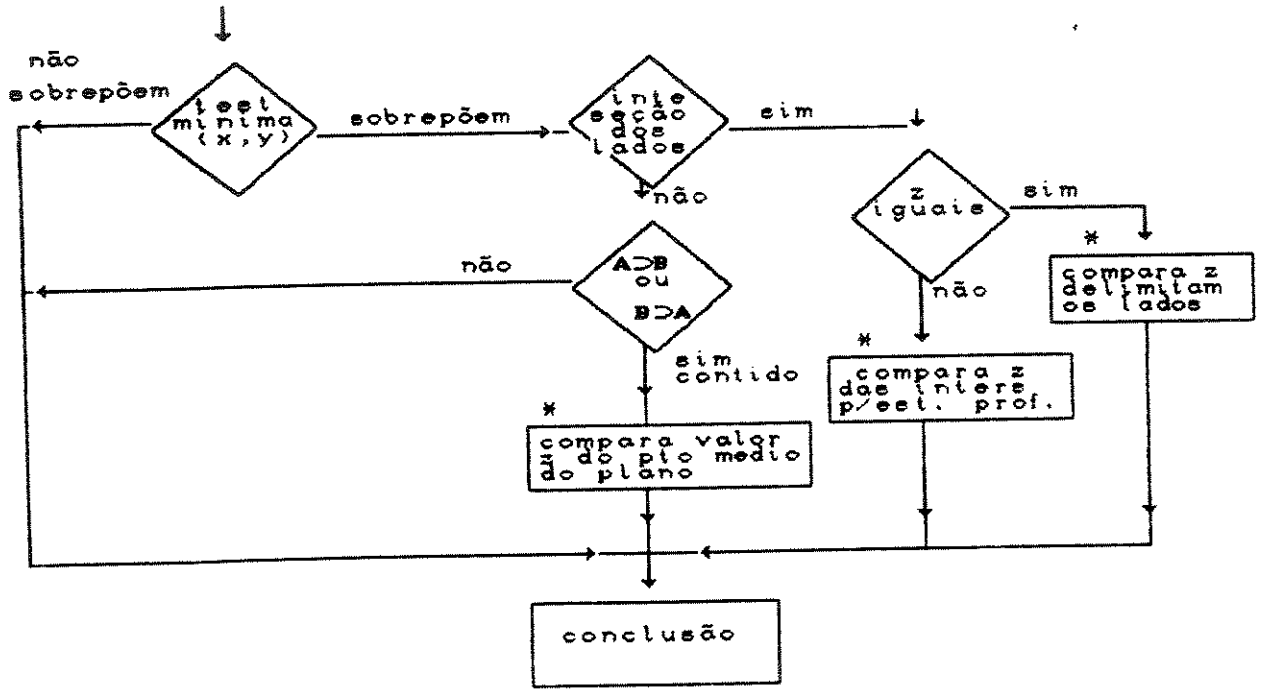


fig. 2.4. Comparação entre dois triangulos.
Onde A e B são triângulos,
⊃ significa incluido.

Na figura 2.4, mostra-se a sequência do processo de comparação de dois triângulos. Os blocos marcados com asterisco podem ser particionados em sub-tarefas paralelas.

No processo de comparação, considerando n triângulos, realiza n(n-1)/2 comparações. Esse processo pode ser feito na forma dinâmica ou estática.

Na forma estática, inicialmente todos os triângulos devem estar armazenados no buffer D, para que o processo de comparação seja iniciado.

Na forma dinâmica, o processo de comparação é disparado quando existe pelo menos um par de triângulos armazenados no buffer D, eliminando a necessidade de esperar pela realização inicial de todo o processo de decomposição.

2.3.4 ORDENAÇÃO GEOMÉTRICA.

O processo de ordenação geométrica é resultado da comparação dos triângulos. Dependendo da escolha da estrutura de dados, pode-se evitar o gasto de processamento da ordenação. Harrington [HARRI,1986], fornece uma idéia apropriada para essa fase do problema, criando duas estruturas de dados para indicar a posição dos triângulos. A primeira corresponde aos triângulos que estão no fundo e a segunda indica os triângulos que estão na frente.

A figura 2.5 mostra a estrutura de dados proposta por Harrington [HARRI,1986]. Nota-se que no arranjo inback, o elemento inback[i] é zero, para os triângulos que estão no fundo, e diferente de zero para os outros casos. Valor que reflete também quantos triângulos existem atrás do triângulo indicado.

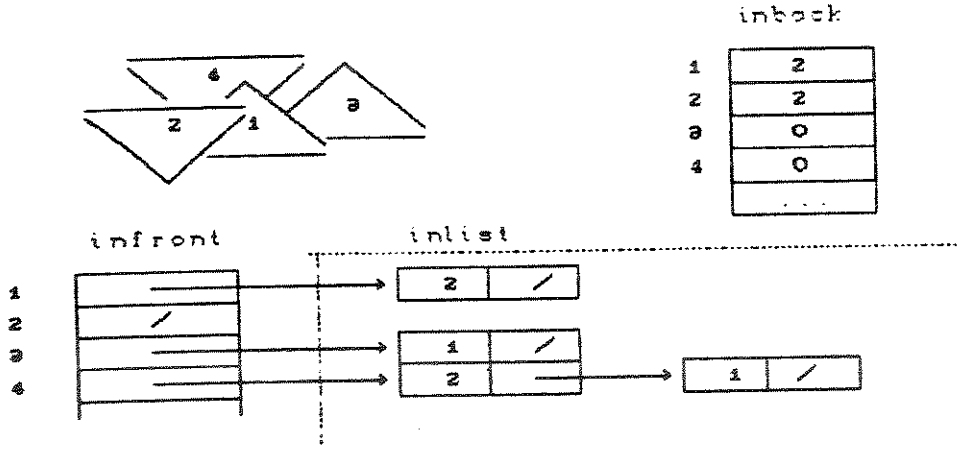


fig. 2.5. Lista encadeada para indicar que triângulos estão na frente.

2.3.5 REMOÇÃO DAS LINHAS ESCONDIDAS.

O algoritmo apresentado por Harrington [HARRI,1986], consiste em realizar o tratamento triângulo por triângulo, de trás

para frente, realizando a análise dos segmentos que formam os triângulos que se situam entre o triângulo de fundo e o triângulo mais a frente, eliminando os segmentos ocultos e armazenando os visíveis e parcialmente visíveis.

Do processo mencionado, observam-se duas grandes tarefas que podem ser paralelizadas. Um primeiro nível de paralelismo, denominado externo, é a distribuição dos triângulos para os quais o conteúdo de inback(i) é zero (triângulo de fundo). Um segundo paralelismo, denominado interno, é a eliminação dos segmentos não visíveis entre o triângulo de fundo tomado e os triângulos que estão em sua frente, e que se resume a uma comparação entre pares de triângulos.

Ao término de todos esses sub-processos de comparação eliminam-se todos os segmentos ocultos e armazenam-se apenas os segmentos visíveis no buffer C. Esse processo repete-se até que todos os triângulos sejam tratados.

Com relação ao paralelismo interno, pode-se ainda continuar dividindo e subdividindo, até um limite determinado pela comparação dos segmentos entre os dois triângulos envolvidos.

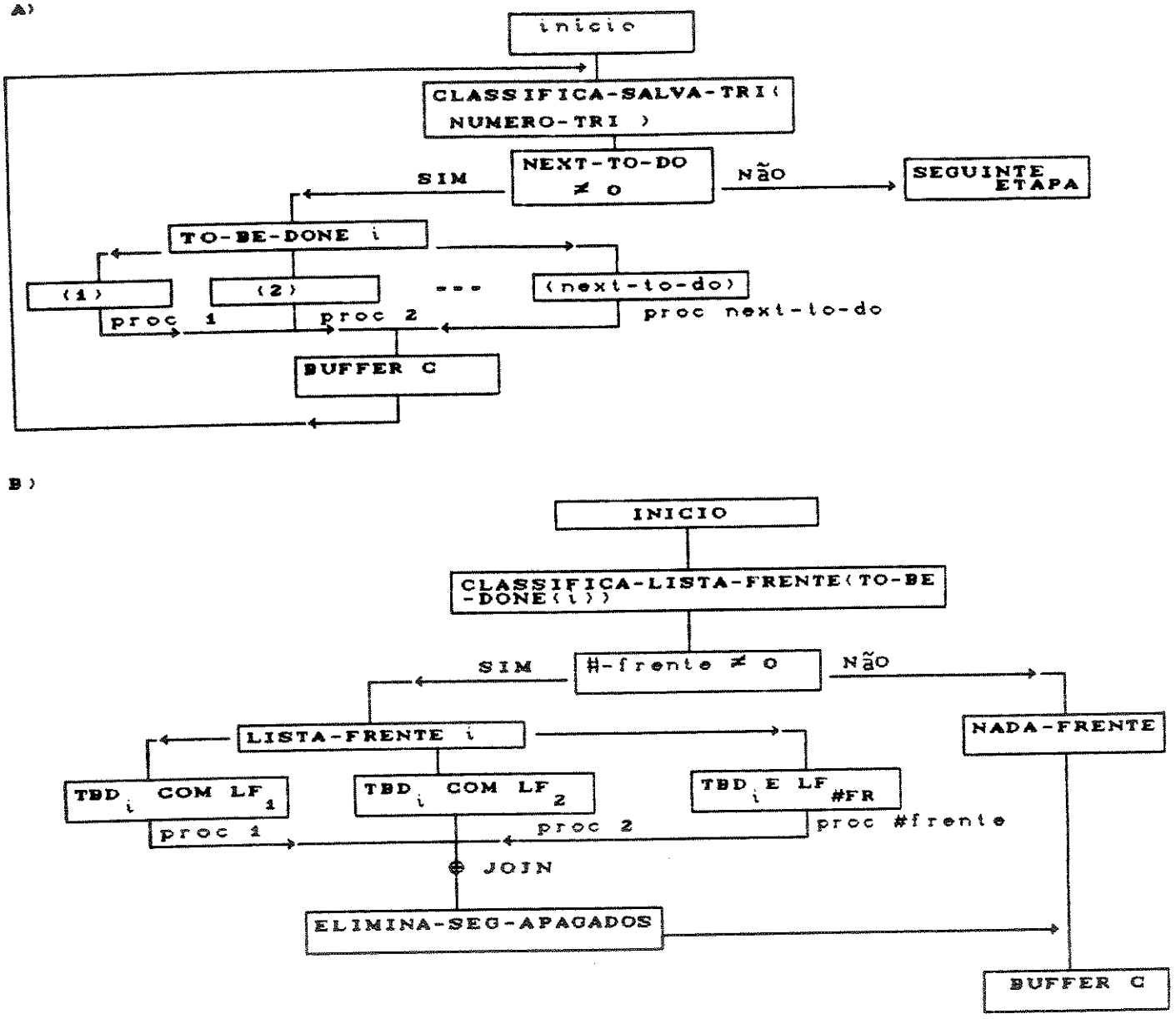


FIG. 2.6 Remoção das linhas escondidas.
a) "fork" para os triângulos do fundo.
b) "fork" para o triângulo do fundo e um triângulo em sua frente.

Na figura 2.6, considera-se que o procedimento CLASSIFICA-SALVA-TRI(NUMERO-TRI) carrega no arranjo TO-BE-DONE(i) todos os triângulos a serem testados, quando INBACK(i)=0 e NEXT-TO-DO ≠ 0. NEXT-TO-DO representa o número parcial de processos a serem distribuidos. Na parte b da figura 2.6, o procedimento CLASSIFICA-LISTA-FRENTE(TO-BE-DONE(i)) cria uma lista de todos os triângulos que estão na frente de

infront(to-be-done(i)). A frente indica o número de triângulos na frente e o número de comparações à realizar com o triângulo a ser testado.

2.4 COMPORTAMENTO DOS DADOS E PROPOSTA PARALELA.

A figura 2.7 mostra o diagrama em blocos para a solução paralela do algoritmo L.S.E.

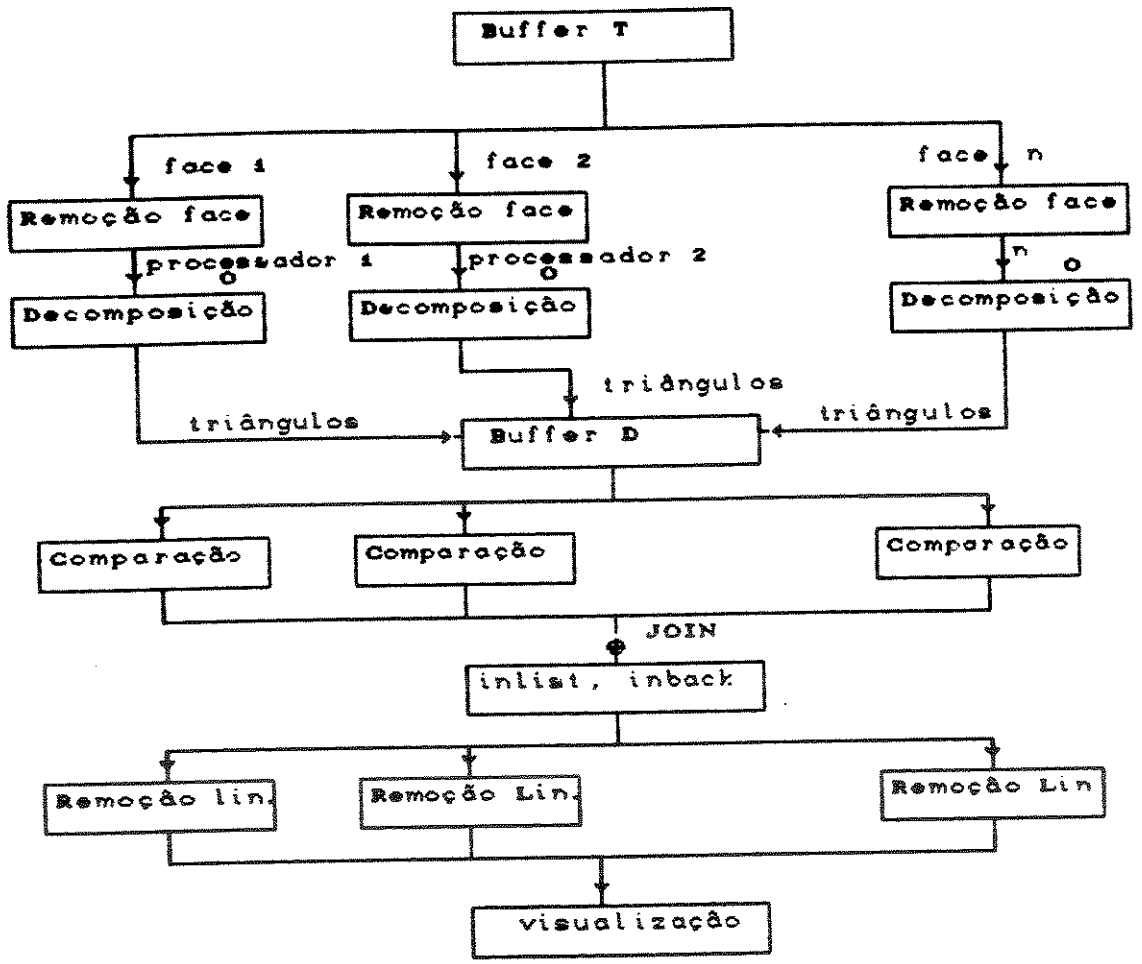


fig. 2.7, Proposta paralela do algoritmo "Linhas e Superficies Escondidas".

Pode-se observar da precedência de tarefas dos processos analisados na seção 2.3, que os módulos de remoção e decomposição possuem precedência de tarefas, onde uma tarefa de remoção de faces visíveis dispara uma tarefa de decomposição em triângulos dessa face visível. Como foi mencionado na seção 2.3.2, se considerarmos a execução de ambas as tarefas num processador,

obtem-se os seguintes ganhos :

a) Eliminação do espaço de armazenamento para as faces visíveis, resultantes do processo de remoção.
b) Minimização do esforço em distribuir ambas tarefas e diminuição do tempo gasto em comunicar os dados envolvidos.

Como resultado do processo de decomposição das faces visíveis em triângulos, no processo de comparação de triângulos, haverá uma quantidade de triângulos que deverão ser comparados, todos com todos, onde um triângulo participará em n-1 processos de comparação de triângulos de um total de n(n-1)/2 comparações. Este número pode ser menor se for garantida a não comparação entre triângulos da mesma face.

O nível de particionamento que poderá ser feito dependerá dos tempos envolvidos nos processos ou sub-processos. Se esses tempos são curtos em relação aos tempos de gerenciamento e comunicação, não devem ser considerados. No primeiro nível de particionamento, os dados não são de uso exclusivo de uma tarefa no processo de comparação por isso, devem estar armazenados no processador mestre para sua distribuição na forma mais eficiente, de tal maneira de garantir a distribuição de pares de triângulos a serem comparados nos processadores.

O processo de ordenação geométrica faz uso da estrutura de dados, mencionada na seção 2.3.5, para armazenar os resultados da comparação dos triângulos. O início da fase seguinte será possível quando todas as comparações entre triângulos estiverem terminadas e suas respostas armazenadas. Nesse ponto existe um "join" efetivo que deverá ser respeitado.

Em resumo, existem dois blocos principais a considerar no planejamento do software para a implementação da proposta paralela do algoritmo "Linhas e Superfícies Escondidas", esses blocos são:

Bloco inicial, composto pelos módulos remoção de faces, decomposição, comparação dos triângulos e ordenação geométrica;

Bloco final, composto pelo módulo remoção das linhas escondidas.

Do ponto de vista da implementação paralela, a solução proposta consiste em gerenciar um conjunto de processos que possuem um conjunto de tarefas independentes, que deverão ser executadas em forma paralela.

CAPÍTULO 3

CONSIDERAÇÕES GERAIS SOBRE PROCESSAMENTO PARALELO

3.1 INTRODUÇÃO.

Na implementação de um sistema com processamento paralelo, deve considerar-se o particionamento, o gerenciamento e a distribuição das tarefas, aproveitando as caraterísticas mais relevantes do Hardware e Software.

O **particionamento** é a fase na qual o problema é dividido em sub-problemas, as tarefas em sub-tarefas e assim por diante. Por definição, a meta da fase de particionamento é aumentar a utilização dos elementos processadores, maximizando o paralelismo e minimizando o custo de comunicação [GAUDI,1989]. Existe um limite para o particionamento, pode existir um particionamento de tarefas, que do ponto de vista teórico seja ótimo, ao permitir um máximo de paralelismo, mas na prática pode resultar numa execução de desempenho menor, porque não foi considerado o custo de comunicação envolvido.

O **gerenciamento** é o processo pelo qual o número de tarefas, os processadores disponíveis e o tipo de relação existente entre os módulos ou tarefas são aproveitados para obter um bom desempenho na execução do problema. A análise do comportamento dos módulos ou das tarefas que podem ser executadas em paralelo, a relação de precedência e a forma como fluem os dados, permitem determinar como será feita a alocação das tarefas nos processadores, a sincronização da execução e a distribuição dos dados.

Na **distribuição dos dados** e/ou sincronização da execução, é de vital importância o processo de comunicação (roteador dos dados). Esse processo deverá satisfazer as necessidades de comunicação de entrada e saída dos processadores,

e devem incluir as seguintes possibilidades:

a) De um processador a qualquer um dos processadores conectados (ponto a ponto);
b) De um processador a vários processadores (multicasting);
c) De um processador a todos os processadores (broadcasting);
d) Todos os processadores a todos os processadores (multibroadcasting).

3.2 DEFINIÇÕES DOS PARÂMETROS DE DESEMPENHO PARALELO.

A exploração do paralelismo como melhoramento do desempenho de um sistema computacional exige, em termos de hardware vários processadores ativos simultaneamente e, do ponto de vista do software, um programa estruturado como um conjunto de sub-tarefas independentes.

Em processamento sequencial, normalmente o desempenho de um sistema pode ser caracterizado em termos da taxa de instruções do processador e do tempo de execução requerido pelo software, num processador de taxa unitária. Em processamento paralelo isso é mais complexo. No domínio do hardware interessa não somente a taxa de instruções de um processador, mas também o número de processadores. No domínio do software, interessa não somente o serviço de demanda, mas também a estrutura do software [EAGER,1989].

Na avaliação de um sistema paralelo, as medidas de interesse são:

1) **Rapidez ou ganho**(speedup). é definido como a relação entre o tempo de execução do programa num único processador, e o tempo de execução do mesmo programa num computador paralelo com n processadores.

$$S(n) = \frac{T_s}{T_n} \qquad (3.1)$$

2) **Eficiência** (efficiency). é definida como a utilização média dos n processadores alocados. A relação entre "speedup" e eficiência é dada por:

$$E(n) = \frac{S(n)}{n} \qquad (3.2)$$

3) **Fração sequencial** (f). é definida como a relação entre o tempo de execução da parte da aplicação que não pode ser executado em paralelo, e o tempo total de execução.

$$f = \frac{\text{tempo sequencial}}{\text{tempo total da execução}} \qquad (3.3)$$

4) **Média de paralelismo** (A). é definido como a média de processadores ocupados durante a execução da aplicação, quando um número ilimitado de processadores está disponível. Pode também ser definida como a relação entre o tempo gasto para executar a aplicação em um único processador (sequencialmente) e o tempo gasto para processar a mesma aplicação em um número ilimitado de processadores.

5) **Máximo e mínimo grau de paralelismo** (M e m). O máximo grau de paralelismo (M) é definido como o número máximo de processadores que estão simultaneamente ocupados, quando um número ilimitado está disponível. De forma análoga, define-se o mínimo grau de paralelismo (m) como o número mínimo de processadores ocupados quando um número ilimitado está disponível.

No caso ideal, se o programa testado apresenta taxas elevadas de paralelismo e o "overhead" de comunicação entre processadores tende a zero, então o tempo de execução no computador paralelo tende a T_1/n. Isso faz com que o "speedup"

tenda a "n", que é o valor máximo. Na prática, sempre vai existir algum "overhead" de comunicação, fazendo com que o tempo de execução no computador paralelo, seja maior que no caso ideal e o "speedup" seja menor que o valor máximo [KIRNER,1991].

Existe uma série de parâmetros que medem o grau de perda do desempenho, denominadas fontes de retardo ou fontes de perdas de desempenho. Estes parâmetros são discutidos a seguir.

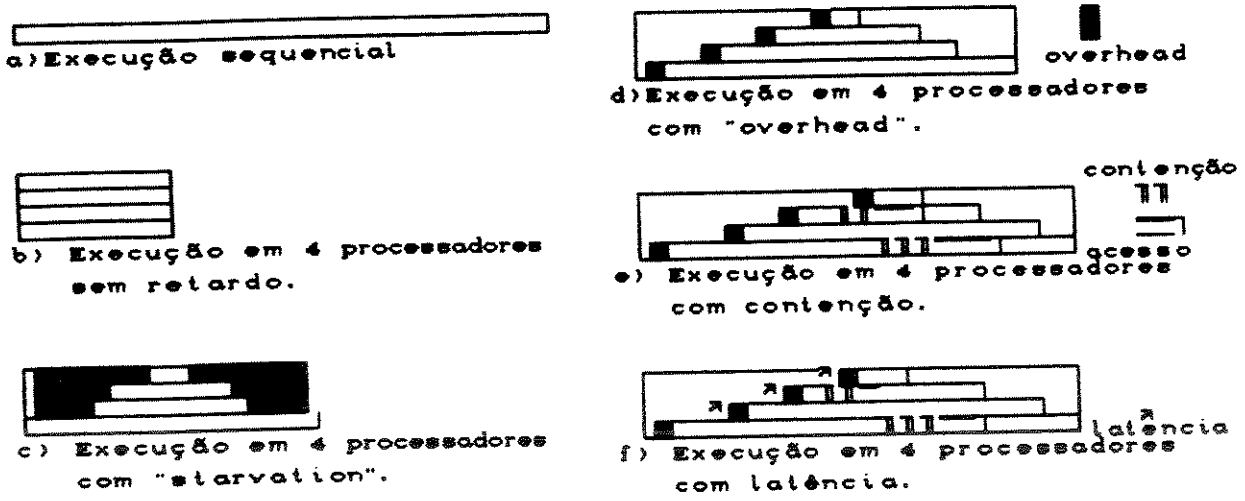


fig. 3.1. Diagramas de tempo e relações com parâmetros de desempenho.

1) **"Overhead"**. Tempo gasto na execução do código que é multiplicado e executado por todos os processos, exemplo: tempo de inicialização de um "loop". Está também relacionado ao incremento de tempo devido à transferência e ao início dos diversos processos da aplicação. Na figura 3.1 o "overhead" é visto pela comparação dos diagramas (c) e (d).

2) **"Starvation"**. Tempo em que os processadores alocados ficam inativos. As restrições da sequência e da dependência das partes da aplicação podem implicar em que nem todos os processadores fiquem em estado ativo simultaneamente. Essa perda de desempenho é visto na figura 3.1 pela comparação dos diagramas (b) e (c).

3) **Contenção**. Na execução do problema, onde se requer a participação de outros recursos que não estão disponíveis, existe um retardo por acesso. Na figura 3.1 a contenção é vista pela comparação dos diagramas (d) e (e).

4) **Latência**. A comunicação dos dispositivos de entrada e saída não é instantânea e leva a um retardo adicional. Esse retardo é visto pela comparação dos diagramas (e) e (f) na fig. 3.1.

As métricas mais empregadas para caracterizar o comportamento de uma aplicação paralela são a rapidez ("speedup") e a eficiência ("efficiency"). Dessas duas medidas pode extrair-se informação sobre a utilização efetiva da malha de processadores, a média de paralelismo e o tempo de execução. Essas duas medidas, não fornecem informação do grau de balançeamento na distribuição das tarefas, nem permitem dizer de forma certa qual é o número mais apropriado de processadores para uma aplicação em particular.

Karp e Flatt [KARP,1990], demonstraram experimentalmente que a fração sequencial (ou serial) é uma métrica que serve para medir o grau de distribuição das tarefas. A fração serial, junto às medidas de rapidez ("speedup") e eficiência ("efficiency"), fornecem melhor informação na escolha de um número apropriado de processadores para uma aplicação em particular.

Esse fato é melhor explicado com o exemplo a seguir : Se existem 12 tarefas iguais, há um perfeito balanço de carga para 2, 3, 4, 6 e 12 processadores, mas não existe para outros números de de processadores. Nesse caso, a fração serial será um parâmetro, que junto aos outros, permitirá dar mais informação na escolha de um sistema multiprocessador.

3.3 CUSTO DE COMUNICAÇÃO.

Às vezes a complexidade de comunicação é maior que a complexidade computacional, ou seja, que se pode gastar mais tempo em rotear os dados aos processadores que processá-los, pelo qual sempre deve-se considerar o custo de comunicação na determinação da complexidade de um algoritmo paralelo.

Sabendo que o custo de comunicação não é zero, os algoritmos devem ser cuidadosamente planejados para minimizar a quantidade de comunicação necessária. O desempenho de um algoritmo pode ser muito diferente em distintas arquiteturas. Por exemplo, no modelo "array processor", a sincronização é automática e os processadores trabalham em passos fixos. Em sistemas multiprocessadores e multicomputadores, a sincronização é alcançada através de programa, consumindo tempo. Quinn, M. [QUINN,1988] assegura que num computador MIMD, os algoritmos que executam relativamente poucas operações entre sincronizações, normalmente exibem baixa eficiência.

Do ponto de vista lógico, na atribuição dos processos aos processadores, em qualquer arquitetura, deve considerar-se dois aspectos importantes. Que são:

a) **Distribuição inicial**. Que consiste em comunicar da melhor maneira aos processadores os dados de operação e disparo dos processos nos processadores;
b) **Comunicação dos resultados (dados)**. De forma similar deve considerar-se a comunicação das saídas dos processos, para dar continuidade à execução da aplicação.

Do ponto de vista físico, o custo de comunicação, segundo El-Rewini e Lewis [EL-REW,1990], deve-se a dois tempos de retardos:

1.- Tempo de retardo resultante do meio de transmissão;

2.- Tempo de retardo da fila (contenção), devido às várias mensagens enviadas a um processador. Esse retardo é o tempo de espera gasto até que a mensagem seja processada.

Nos sistemas que funcionam com troca de mensagens, as contenções ocorrem quando duas ou mais tarefas, em diferentes processadores, enviam mensagens através de canais a um mesmo processador.

Em geral o parâmetro custo de comunicação é computado da seguinte maneira:

$$C(*) = \left[\frac{Data}{R} + I \right] * H + D \qquad (3.4)$$

onde :

H : Distância entre o processador emissor da mensagem e o processador destino;
I : Tempo de processamento da mensagem em cada processador;
D : Retardo na espera para ser atendido nos diferentes nós envolvidos na rota da mensagem;
R : Taxa de transmisão do meio.

3.4 CLASSIFICAÇÃO DAS ARQUITETURAS.

A classificação baseada nos fluxos de instruções e de dados, proposta por Flynn [FLYNN,1972], é uma das mais utilizadas. Essa classificação considera o processo computacional como a execução de uma sequência de instruções sobre um conjunto de dados.

A combinação de fluxos de instruções únicos e múltiplos

com fluxos de dados únicos e múltiplos, produz as quatro categorias de modelos computacionais. Que são:

a) **SISD** (fluxo único de instruções/fluxo único de dados). As instruções são executadas sequencialmente, mas as instruções podem explorar o esquema "pipeline" para elevar o desempenho [HWANG,1984]. A figura 3.2a mostra esse tipo de arquitetura.

b) **SIMD** (fluxo único de instruções/fluxo múltiplo de dados). Essa classe corresponde aos "array processor", onde existem múltiplos elementos processadores (PE) supervisionados pela unidade de controle. Todos os PE recebem a mesma instrução da unidade de controle, mas operam sobre diferentes conjuntos de dados. A figura 3.2b apresenta essa arquitetura.

c) **MISD** (fluxo múltiplo de instruções/fluxo único de dados). Nessa organização existem n unidades de processamento, cada uma executando distintas instruções e operando sobre o mesmo fluxo de dados. O resultado (saída) obtido de um processador será a entrada (operando) do processador seguinte que constitui a estrutura "macropipeline". Não existe nenhum computador com essa arquitetura. A figura 3.2c mostra essa organização.

d) **MIMD** (fluxo múltiplo de instruções/fluxo múltiplo de dados). Nessa categoria estão os computadores paralelos, cujos processadores executam instruções diferentes sobre dados diferentes e interagem entre si através de memória compartilhada ou de outro tipo de interconexão. A figura 3.2d mostra a configuração geral desse tipo de computador.

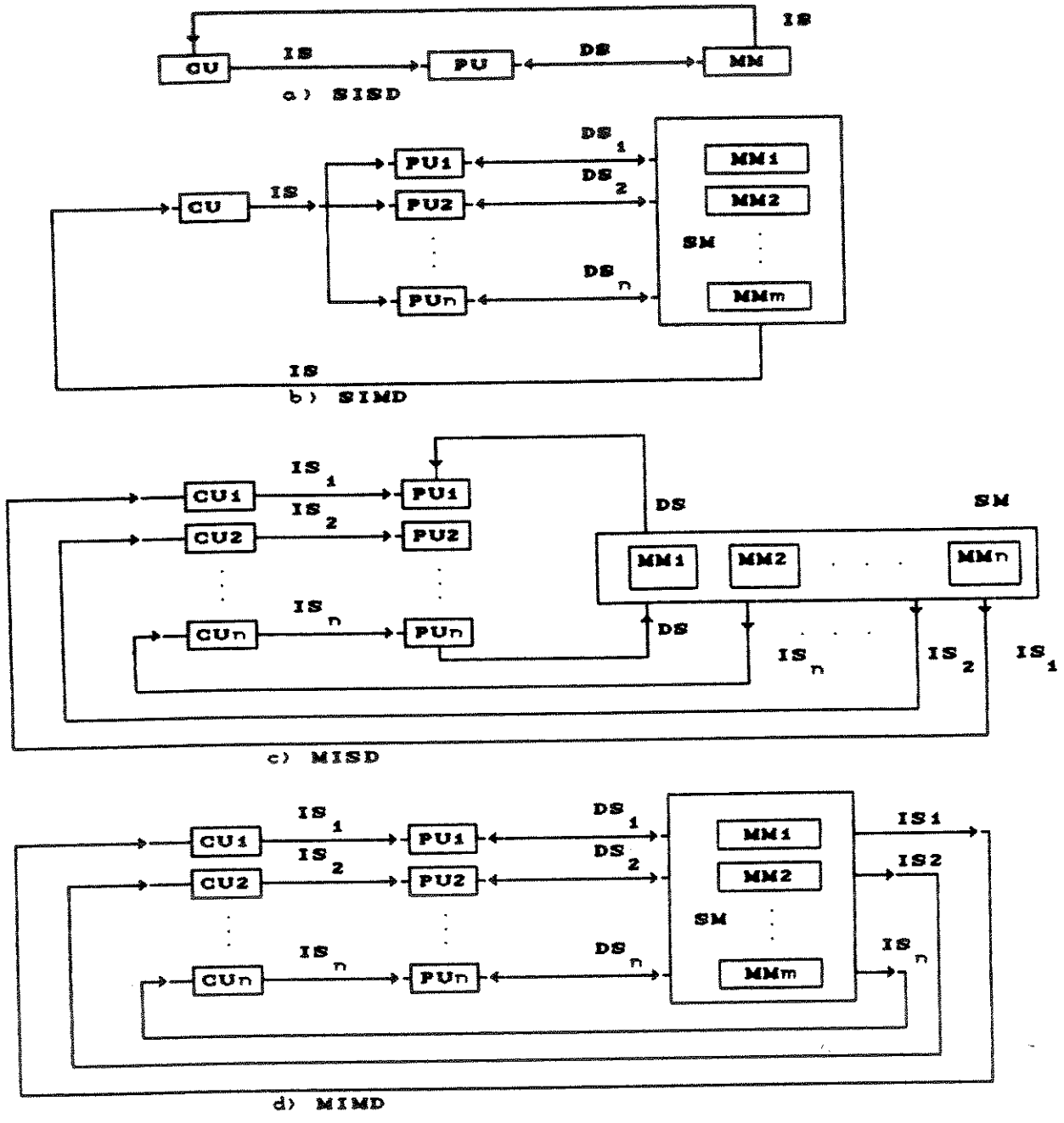


fig. 3.2, Taxonomia de Flynn.

Todas as categorias de Flynn baseiam-se em processamento dirigido pelas instruções, razão pela qual são denominados modelos convencionais, ou baseados em "Von Neuman".

Este trabalho concentra-se sobre as arquiteturas MIMD, que são caraterizadas pela existência de diversos elementos de processamento, que operam de modo assíncrono e comunicando-se entre si.

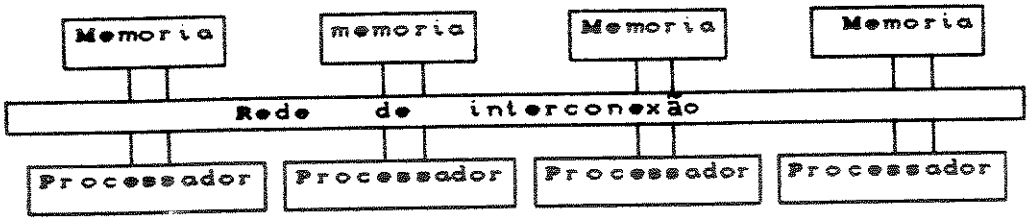
3.5 ARQUITETURAS MIMD.

As arquiteturas MIMD podem ser divididas em sistemas fortemente acoplados (ou com memória compartilhada) e sistemas fracamente acoplados (ou com memória distribuída).

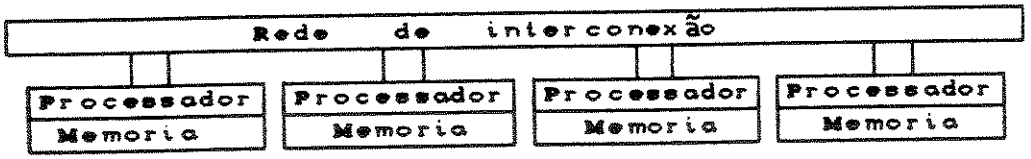
Nos sistemas fortemente acoplados, todos os processadores têm acesso a uma memória global. A comunicação e a sincronização dos processadores é feita através de dados colocados nessa memória.

Nos sistemas fracamente acoplados, cada processador possui uma certa quantidade de memória local, não acessível para os outros. A comunicação e sincronização dos processadores é feita através de uma rede de interconexão entre processadores. Como não existe memória compartilhada, os processadores devem trocar mensagens através da rede para compartilhar os dados.

A figura 3.3, apresenta os dois tipos de arquitetura MIMD, em forma generalizada.



a) Sistemas com memória compartilhada.



b) Sistema com memória distribuida.

fig. 3.3. Arquiteturas MIMD, a) fortemente acoplada b) fracamente acoplada.

Os computadores MIMD suportam um alto nível de paralelismo, organizados como tarefas independentes, que pode ser

explorado por algoritmos que suportam a metodologia "dividir e conquistar" [DUNCA,1990].

3.5.1 ARQUITETURAS DE MEMÓRIA COMPARTILHADA.

As arquiteturas de memória compartilhada efetuam a comunicação entre os processadores através do acesso de cada processador às memórias globais. Os computadores de memória compartilhada não apresentam os problemas das trocas de mensagens dos sistemas distribuídos, em contrapartida apresentam outros problemas, tais como: congestionamento no acesso à memória devido ao tipo de conexão utilizado e sincronização do acesso aos dados, com a necessidade de mecanismos de sincronização, para evitar que uma posição de memória seja acessada por um processo antes que outro termine de atualiza-lá (regiões críticas).

A figura 3.4 apresenta várias alternativas para conexão num sistema multiprocessador com memória compartilhada, as que são discutidas a seguir.

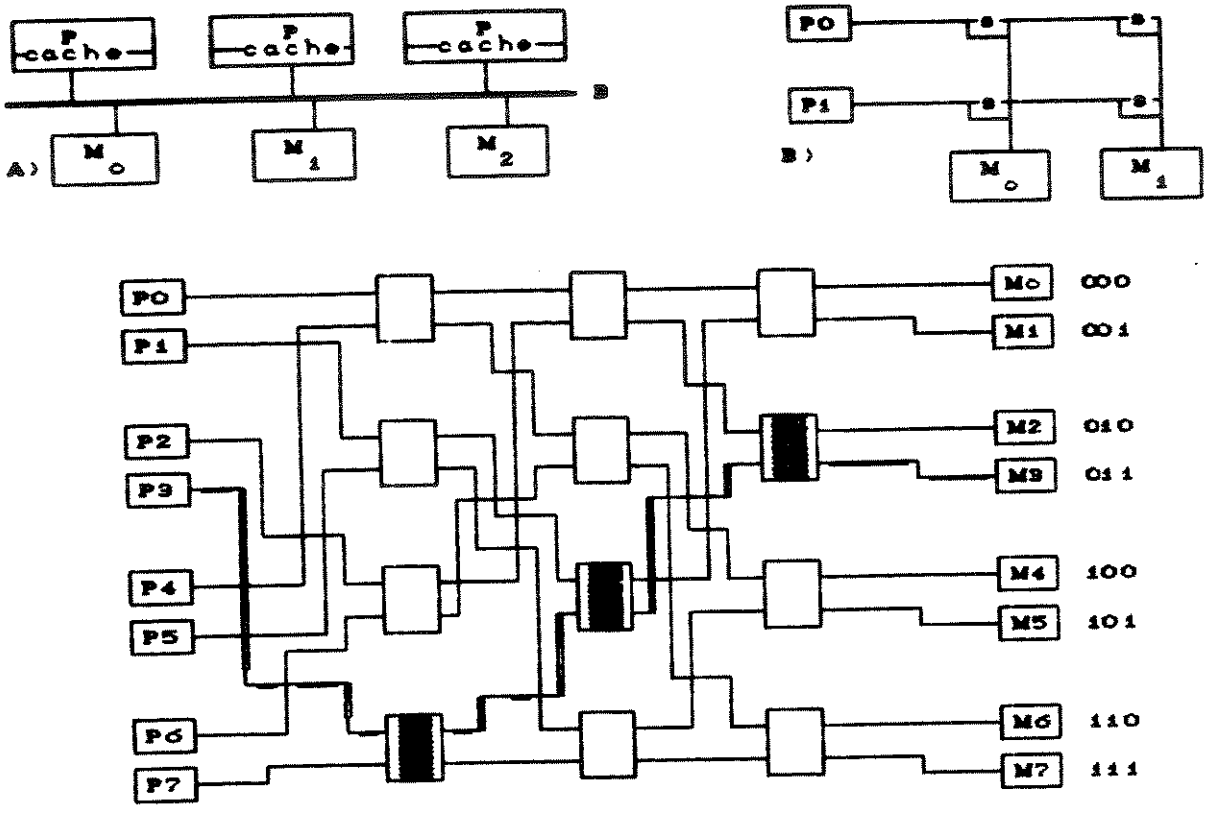


fig. 3.4. MIMD fortemente acoplados,
a) barramento
b) 2x2 crossbar
c) MIN 8x8 omega.

3.5.1.1 INTERCONEXÃO POR BARRAMENTO.

A figura 3.4a mostra um sistema multiprocessador utilizando barramento comum ou de tempo partilhado. Essa organização é a menos complexa e fácil de reconfigurar, normalmente é passiva, sem componentes ativos. As operações de transferência são controladas completamente pela interface do barramento, mecanismo que evita e resolve as contenções do envio e recepção de mensagens.

Apesar de que a organização num barramento único é confiável e relativamente barata, introduz um componente crítico no sistema, que pode causar a falha do sistema completo, como resultado do funcionamento errado da interface do barramento. Além disso a expansão do sistema, pelo acréscimo de processadores e memória, aumenta as contenções do barramento, degradando o número de operações por unidade de tempo do sistema ("throughput") e

aumentando a lógica de arbitragem do barramento.

Uma extensão da organização de um barramento comum é de dois barramentos unidirecionais (multi-barramento), isso alivia os problemas mencionados acima, mas, qualquer operação de transferência no sistema requer a utilização de ambos barramentos.

Existem vários fatores que afetam as caraterísticas de performance de um barramento, isso inclui o número de dispositivos ativos no barramento, algoritmo de arbitragem, controle centralizado, comprimento da mensagem, sincronização da transmissão e deteção de erros.

3.5.1.2 INTERCONEXÃO CROSSBAR.

A tecnologia de interconexão "crossbar" usa um comutador de n^2 pontos para conectar n processadores a n módulos de memória (figura 3.4b). A conexão "crossbar" evita as contenções fornecendo um caminho exclusivo entre cada possível par processador/memória.

Se comparado ao barramento, o crossbar permite uma maior taxa de troca de dados entre processadores e memória, em contrapartida apresenta um custo maior do que o apresentado pelo barramento.

3.5.1.3 MALHAS DE INTERCONEXÃO MULTIESTÁGIO.

As malhas de interconexão multiestágio (MIN) conseguem um melhor compromisso de preço/desempenho, em relação as caraterísticas apresentadas nas interconexões crossbar e barramentos. Um MIN NxN conecta N processadores a N módulos de memória, através de diversos estágios, ou bancos de chaves.

Se N é uma potência de 2, a MIN empregará $\log_2 N$ etapas de N/2 chaves de 2x2, para fornecer a conexão de NxN. Quando um processador faz um acesso à memória, especifica o destino pelo uso dos bits de controle de cada etapa.

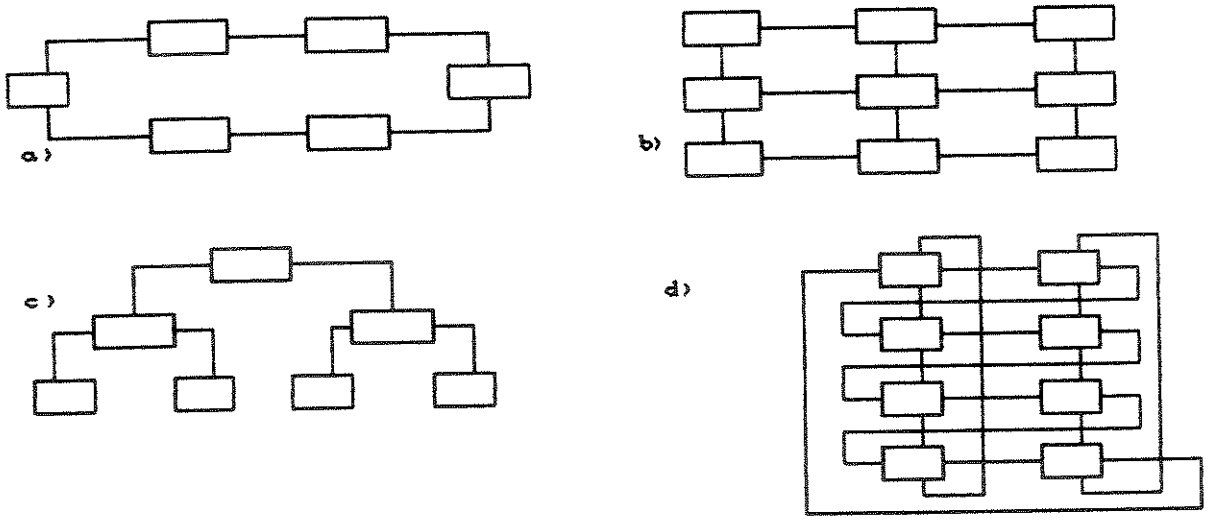
A figura 3.4c mostra uma malha ômega conectando 8 processadores e módulos de memória, onde os bits de controle "zero" indicam conexão direta e "um" em cruz [DUNCA,1990].

A capacidade de expansão é uma vantagem significativa das MINs, porque o diâmetro de comunicação é proporcional a $\log_2 N$. O BBN (Bolt, Beranek e Newman) Butterfly, por exemplo pode ser configurado com mais de 256 processadores [RETTB,1990].

3.5.2 ARQUITETURA DE MEMÓRIA DISTRIBUIDA.

Esse tipo de arquitetura une os nós processadores por uma malha de interconexão. Cada nó consta de um processador autônomo com memória local. Os nós compartilham dados pela troca explícita de mensagens através da malha.

A figura 3.5 mostra as topologias mais comuns desse tipo de arquitetura, as que são discutidas a seguir.



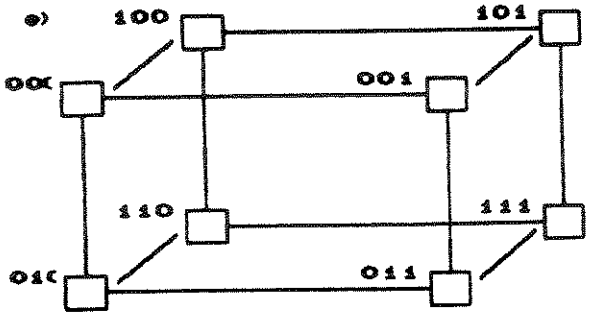


fig. 3.5 . MIMD fracamente acoplados; a) anel; b) malha c) árvore; d) malha "vrap-arround"; e) Cubo

3.5.2.1 ARQUITETURA COM TOPOLOGIA EM ANEL.

O diâmetro de comunicação (n/2) da arquitetura topológica em anel é reduzido pelas conexões tipo "Chordal". Usando esse tipo de conexão ou múltiplos anéis, pode ser aumentada a tolerância a falha de um sistema de comunicação. Essa topologia é mais apropriada para poucos processadores, executando algoritmos de baixa taxa de transferência de dados entre processadores (figura 3.5a).

Nessa topologia, uma mensagem passa de um processador a outro, geralmente, por nós intermediários que devem retransmiti-la.

3.5.2.2 ARQUITETURA COM TOPOLOGIA EM MALHA (MESH).

Uma malha bidimensional ou treliça (figura 3.5b), é uma topologia que tem n^2 nós, cada um conectado aos quatros vizinhos imediatos. A conexão "Wrap-arround" nas bordas fornece uma redução do diâmetro de comunicação de 2(n-1) a $2\times\lfloor n/2 \rfloor$.

As malhas 2D com "Wrap-arround" apresentam-se como uma ótima opção se for considerada a utilização de "transputer". Na faixa das centenas de processadores, esse tipo de malha tem um diâmetro menor do que hipercubo. Uma proposta interessante é a malha denominada "Midimew" (figura 3.5d). Herrada e outros [HERRA,1990] estabelecem todas as expressões analíticas para a

configuração da malha e critérios de operação.

3.5.2.3 ARQUITETURA COM TOPOLOGIA EM ÁRVORE (TREE).

Essa topologia é usada naturalmente para suportar algoritmos em cujo desenvolvimento utiliza-se o método "dividir e conquistar". Como por exemplo, busca e classificação de informação, algoritmos de processamento de imagem, "dataflow" e paradigmas de redução de programação.

O valor do diâmetro de comunicação de uma árvore binária, com n níveis e 2^n-1 processadores, é de 2(n-1) passos. A estratégia mais comum para reduzir o diâmetro é conectar todos os nós de um mesmo nível, conseguindo diminuir um passo no diâmetro.

A figura 3.5c mostra a topologia do tipo árvore.

3.5.2.4 ARQUITETURA COM TOPOLOGIA EM HIPERCUBO.

Uma topologia booleana n-cubo ou Hipercubo usa $N=2^n$ processadores arranjados num cubo de n-dimensões, onde cada nó tem $n=\log_2 N$ conexões bidirecionais aos nós adjacentes. Os nós individuais são identificados em forma única por valores numéricos de n-bits de 0 a N-1, e alocados de maneira que se assegure que os valores dos nós adjacentes diferem em um bit. O diâmetro de comunicação de um Hipercubo é $n=\log_2 N$. A figura 3.5e mostra um hipercubo para n=3.

Para desenvolver o algoritmo de comunicação para o hipercubo, utilizam-se as propriedades gráficas da topologia e sistema de numeração. Considere por exemplo, o problema de enviar dados do nó A ao nó B. Seja $A = a_1 a_2 \ldots a_n$; e $B = b_1 b_2 \ldots b_n$; A e B diferem em i bits, portanto um possível caminho é:

$$
\begin{aligned}
A = \text{nó } 0 &\longrightarrow a_1 a_2 a_3 \ldots a_i a_{i+1} \ldots a_n \ ; \\
1 &\longrightarrow b_1 a_2 a_3 \ldots a_i a_{i+1} \ldots a_n \ ; \\
2 &\longrightarrow b_1 b_2 a_3 \ldots a_i a_{i+1} \ldots a_n \ ;
\end{aligned}
$$

. . .

. . .

B = nó $i \longrightarrow b_1 b_2 b_3 \ldots b_i a_{i+1} \ldots a_n$;

A análise dos caminhos de A a B revela que ao iniciar a transferência não existe razão para começar pelo primeiro bit diferente. De modo geral, é possível começar pelo j-ésimo bit diferente (onde j <= i), até alcançar as i comutações para chegar a B, razão pela qual existem i caminhos diferentes, que são dependentes da escolha inicial.

A figura 3.6 mostra um algoritmo de roteamento que seleciona um caminho mínimo entre o nó fonte $S = s_{n-1} s_{n-2} \ldots s_0$, e o nó destino $D = d_{n-1} d_{n-2} \ldots d_0$. Esse algoritmo denominado ROUTE [HAYES,1989], consiste primeiro em calcular a operação OR-Exclusivo (XOR) bit a bit, entre os nós fonte e destino, ($x = S \oplus D$). Depois, se procura, da esquerda para a direita, um bit $x_i = 1$. O procedimento transmite a mensagem ao vizinho, fornecida pela operação OR-exclusivo do nó fonte e 2^i (i = posição relativa do bit). Considerando o nó vizinho como fonte, repete-se o procedimento até que os bits de x sejam zero, o que significa que a mensagem chegou ao destino.

```
Procedure ROUTE;
     BEGIN
          x(n-1) x(n-2) ... x(0) = ( s(n-1) ⊕ d(n-1) )
                                   ( s(n-2) ⊕ d(n-2) )
                                        ....
                                   ( s(0) ⊕ d(0) )
           FOR i=0 to n-1 DO
             IF x(i) =1 THEN
                { Envie a mensagem ao i-ésimo vizinho}
              END IF
     END.
```

fig. 3.6. Algoritmo Route para Hipercubo.

3.6 ESCALONAMENTO DE TAREFAS EM COMPUTADORES MIMD.

O processo de escalonamento é uma alocação dos processos aos processadores físicos. A meta do processo de escalonamento é minimizar o "makespan" ou tempo de término de um algoritmo paralelo.

Nesta seção serão mencionados os tipos de escalonamento existentes e formalismos necessários para seu estudo, assim como a análise de publicações sobre escalonadores, cujos conceitos foram aplicados no desenvolvimento deste trabalho.

3.6.1 MODELOS DE ESCALONADORES.

Existem basicamente dois modelos de escalonamento: o determinístico e o não-determinístico.

No **modelo não-determinístico** o tempo de execução de uma tarefa é representado por uma variável aleatória, condição que dificulta a solução do problema. Esse modelo não será abordado neste trabalho, visto que para a aplicação em mente os tempos podem ser considerados determinísticos.

Nos **modelos determinísticos** todas as informações para expressar as caraterísticas do problema são conhecidas. Essas caraterísticas são: tempo de execução de cada tarefa e relacionamento entre as tarefas no sistema. O objetivo do escalonador resultante é otimizar um ou mais critérios de avaliação. Por exemplo, num modelo determinístico, o tempo de execução de cada processo pode ser interpretado como o tempo máximo de processamento, ou como o tempo de processamento esperado das tarefas.

O modelo determinístico não é realista, porque não considera as variações no tempo de execução das tarefas produzidas pelas interrupções, contenções, etc., mas, pelo menos, fazem possível a alocação estática das tarefas aos processadores.

O escalonamento determinístico normalmente é ilustrado em carta Gantt, indicando o tempo de execução que cada tarefa ocupa no processador. Por exemplo, a figura 3.7, mostra o grafo de precedência e a carta Gantt correspondente.

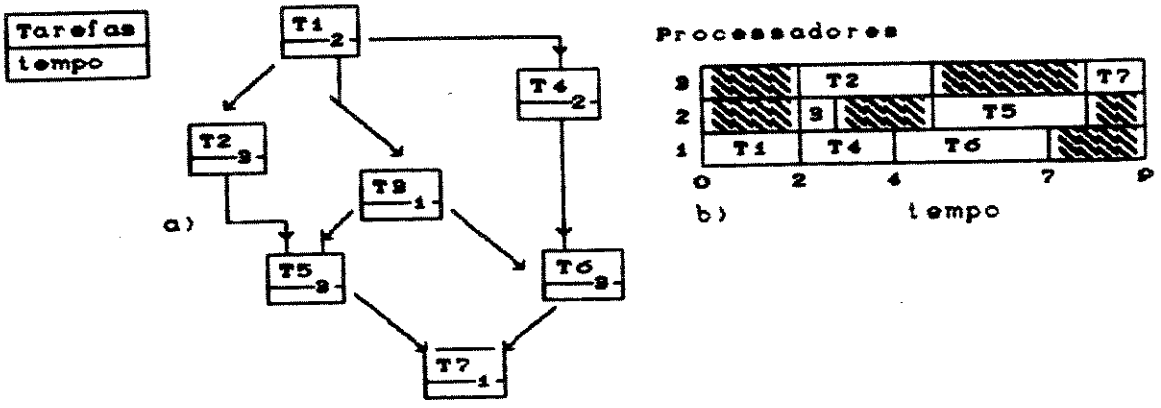


fig. 3.7, a) grafo de tarefas,
b) carta Gantt correspondente ao grafo de a).

Os seguintes formalismos são empregados na notação do grafo de tarefas [QUINN,1988].

Seja T_1, T_2, ..., T_n um conjunto de tarefas ordenadas parcialmente por precedência " < ".

a) A tarefa T_i e predecessora de T_j e T_j é sucessora de T_i, se $T_i < T_j$;
b) As tarefas sem nenhum predecessor são chamadas **tarefas iniciais**;
c) As tarefas de um conjunto são independentes, se para cada par de tarefas T_i e T_j no conjunto, nenhuma é predecessora da outra;
d) A largura de um grafo de tarefas é o tamanho do máximo conjunto de tarefas independentes;
e) Uma cadeia de tarefas é o grafo de tarefas totalmente ordenado;

f) O comprimento de uma cadeia é o número de tarefas na cadeia;

g) O nível de uma tarefa T num grafo de tarefas é o máximo comprimento da cadeia que pertence, a partir de uma tarefa inicial a T;

h) A profundidade de um grafo de tarefas G é o máximo nível de uma tarefa em G.

O conjunto de tarefas pode ser escalonado em forma preventiva ou não preventiva. No escalonamento **não preventivo**, a tarefa é executada integralmente num processador. Ao contrário, o escalonador **preventivo** permite que uma tarefa seja executada por partes, às vezes, em diferentes processadores.

3.6.2 ESCALONADOR DETERMINÍSTICO COM PRECEDÊNCIA DE TAREFAS.

Anger e outros [ANGER,1990], apresentam uma proposta que minimiza o tempo necessário para completar um conjunto dado de tarefas num sistema fracamente acoplado.

No caso clássico, se existem mais processadores que tarefas, o problema de minimizar o "makespan" chega a ser trivial. Isso não é satisfatório na presença de retardos de comunicação, quando existe precedência de tarefas. Considere por exemplo o grafo da figura 3.8a, onde cada tarefa tem um tempo de execução unitário. Sem "overhead" de comunicação, as tarefas podem ser escalonadas em dois processadores, como mostra a figura 3.8b, e executadas com um "makespan" de três. Se o tempo de comunicação suposto entre uma combinação de tarefas e processadores é de 0.5, então, o escalonamento da tarefa T_4 é feito segundo a carta Gantt apresentada na figura 3.8c. Se o retardo chega ser maior que 1, então o escalonamento é feito apenas em um processador (figura 3.8d).

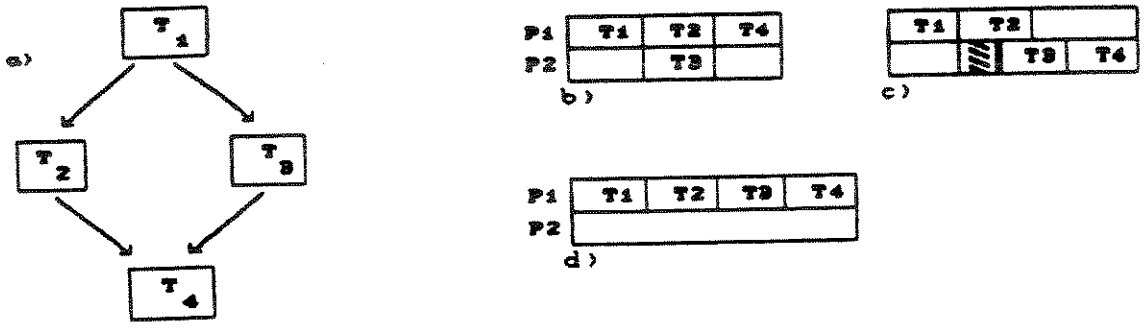


fig. 3.8. Do exemplo do algoritmo JLP
a) Grafo de precedência,
b) escalonamento sem retardo,
c) escalonamento com retardo 0.5,
d) escalonamento retardo > 1 .

A fim de mostrar a estratégia a ser seguida na confecção de um algoritmo que permita distribuir tarefas, será analisado um algoritmo escalonador, apresentado por Anger [ANGER,1990], que leva em conta a precedência de tarefas.

Nesse algoritmo considera-se uma relação de precedência tipo árvore e todas as tarefas de tempo unitário. O algoritmo escalonador produz um escalonamento factível dadas as seguintes condições: precedência de tarefas tipo árvore, suficientes processadores para as tarefas iniciais, conexões idênticas (link), malha de interconexão "fully conected" e curto retardo de comunicação.

Algoritmo JLP (Join Latest Predecessor).

Por simplicidade, as n tarefas são referidas como 1,2,...,n em vez de T_1, T_2,..., T_n.

Entrada: As tarefas 1,2,...,n com tempo de processamento μ_1, μ_2, ..., μ_n;
A relação de precedência $\longrightarrow$ tal que para qualquer i, $i \longrightarrow_I j$ (i precede imediamente a j);
Para um retardo de comunicação c(i,j) tal que para todo

$i \longrightarrow_{I} j$ $c(i,j) \leq \mu_k$ para todo i,j,k;

Também assuma que as tarefas são numeradas tal que $i \longrightarrow j$ implica que i<j.

Saída : Para cada i≤n; P(i) indica o processador no qual a tarefa i será escalonada;

S(i) e F(i) indicam tempo de começo e tempo de finalização,

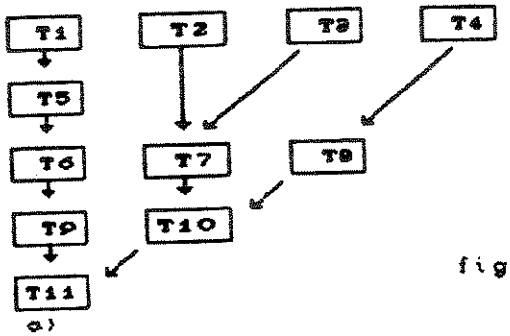
respectivamente, da tarefa i no processador P(i).

```
BEGIN
  FOR j = 1 TO n DO
    IF
      j = folha (não tem predecessor)
     THEN
       BEGIN
         P(j)=j
         S(j)=0
         F(j)=μ_j
        < j é escalonado >
       END
     ELSE
       BEGIN
         < defina um predecessor imediato k tal que F(k)+C(k,j)
           é máxima para todos os predecessores imediatos de j >
         P(j)=P(k)
         < Assegure que j não seja retardado por C(k,j) >
         S(j)=max< F(k), max< F(i)+C(i,j):i ->I j e i≠k >>
         < j começa quando k feche ou quando a ultima comunica-
           ção chegue do outro predecessor imediato >
         F(j)=S(j)+μ_j
         < j é escalonado >
       END
END.
```

fig. 3.9, Algoritmo JLP.

Na figura 3.10, apresenta-se um exemplo de escalonamento resultante do uso do algoritmo JLP (figura 3.9). Da análise dos resultados, o "makespan" é 20% maior do que seria se os retardos de comunicação não fossem considerados. Em geral, o retardo de comunicação pode causar um tempo maior, chegando até 100%.

A complexidade de tempo do algoritmo JLP é de ordem n "O(n)", isto é, o tempo requerido para produzir o escalonamento é linear em relação ao número de tarefas a serem escalonadas.



tempo	1	2	3	4	5	6
P1	T1	T5	T6	T9	...	T11
P2	T2	...	T7	T10	...	
P3	T3	...	...	...	...	
P4	T4	T8	...	...	...	

b)

fig. 3.1c. Exemplo de aplicação do algoritmo JLP.
a) grafo de precedência
b) Carta Gantt do escalonamento.

3.6.3 ESCALONADOR DETERMINÍSTICO DE TAREFAS INDEPENDENTES.

No problema de escalonar n tarefas, normalmente, é desejável minimizar o tempo total de execução, que corresponde ao tempo máximo de finalização do conjunto de tarefas escalonadas. É natural achar que a realização de um escalonamento preventivo, pode resultar numa boa solução do problema, mas, na presença de "overhead" de comunicação, pode significar um aumento do tempo de execução do algoritmo paralelo.

Uma maneira de medir a factibilidade do escalonamento preventivo é através da minimização do fluxo de tempo, que é definido como a soma dos tempos de finalização de cada tarefa. O que permite avaliar a incidência do "overhead" de comunicação, considerando a distribuição de tarefas incompletas aos processadores.

Leung e Young [LEUNG,1989] mostram um algoritmo que minimiza o tempo de execução do algoritmo paralelo, com a restrição de que o fluxo de tempo deve ser o mínimo. Esse algoritmo, chamado " algoritmo B", é apresentado a seguir:

Algoritmo B.

Esse algoritmo considera as seguintes declarações:

TS : Conjunto de tarefas $(T_1, T_2, \ldots, T_n)$.

F : Tempos das tarefas atribuídas ao processador i;

p : Tempo de cada tarefa;

OSL: Tempo ótimo do escalonamento;

m : Número de processadores disponíveis;

r : grupo de tarefas, quando m é menor do que o número de tarefas;

SPT: Escalonamento das (r-1)m tarefas em forma não preventiva.

As n-(n/m) tarefas finais são escalonadas em forma decrescente, segundo o tempo de cada tarefa.

Entrada : Um conjunto de n tarefas a ser escalonadas em $m \geq 1$ processadores idênticos.

Saída : Um escalonamento ótimo de tarefas em m processadores.

Método.

1. Construir o SPT para as tarefas $T_1, T_2, \ldots, T_{(r-1)m}$;

2. $F(k) \leftarrow \sum_{j=1}^{r-1} p_{(j-1)m+k}$ para cada $1 \leq k \leq m$;

3. $OSL \leftarrow \max_{k=1}^{m} \{ \sum_{l=1}^{k} (F(l)+p_{n-l+1})/k \}$;

4. Inicialize o conjunto de processadores disponíveis 1,2,..,m. Escalone as tarefas T_n, $T_{n-1}, \ldots, T_{n-m+1}$. Assuma que T_i será escalonado. Seja $k_1, k_2, \ldots, k_x$ os indices dos processadores disponíveis tal que $F(k_1) \leq F(k_2) \leq \ldots \leq F(k_{xi})$ < OSL. T_i será escalonado, segundo as seguintes três regras:

   1 Se $p_i \leq (OSL-F(k_{xi}))$, então T_i é escalonado no processador k_{xi}, $F(k_{xi}) \leftarrow F(k_{xi})+p_i$, $I \leftarrow I - (k_{xi})$;

   2 Se $p_i = OSL-F(k_l)$ para alguns $1 \leq l \leq xi$, T_i é escalonado completamente no processador k_l, $F(k_l) \leftarrow OSL$, $I \leftarrow I-(k_l)$;

   3 Se ambos casos anteriores não são aplicáveis, então faz que l seja tal que $OSL-F(k_{l+1}) < p_i < OSL-F(k_l)$. Escalone a parcela $OSL-F(k_{l+1})$ do T_i no processador k_{l+1} e o resto de T_i no processador k_l.
   $F(k_l) \leftarrow F(k_l)+(p_i-(OSL-F(k_{l+1})))$.
   $F(k_{l+1}) \leftarrow OSL$
   $I \leftarrow I - (k_{l+1})$.

fig. 3.11, Algoritmo preventivo de tarefas independente.

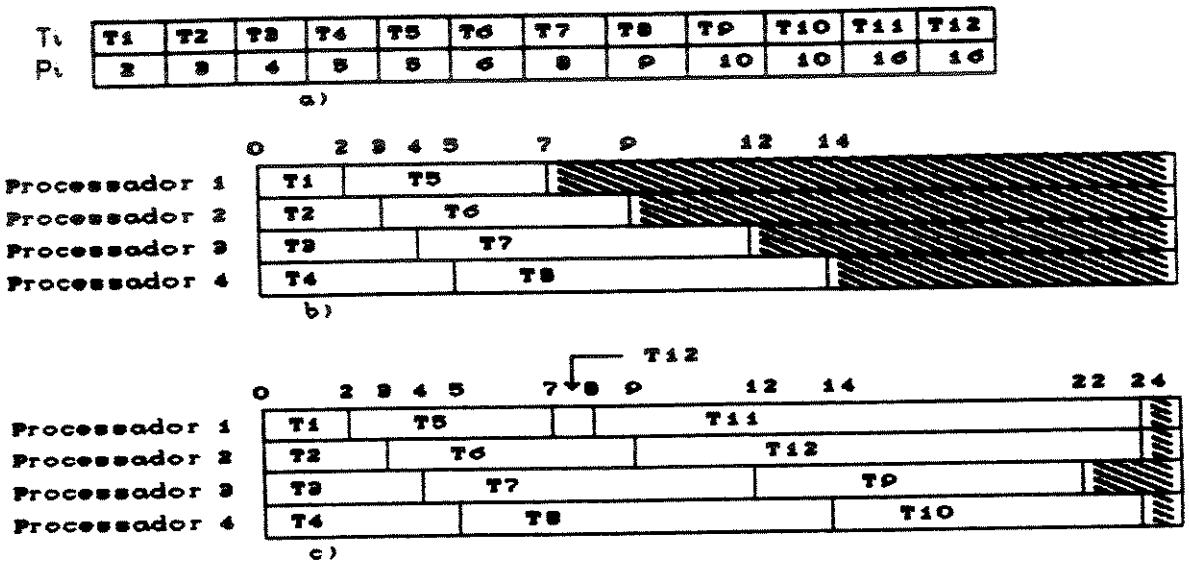


fig. 3.12, Exemplo do algoritmo B.

A figura 3.12 mostra um conjunto de 12 tarefas a serem escalonadas em quatro processadores. A figura 3.10b, mostra o efeito do escalonador SPT, construído no passo 1 do algoritmo. Os tempos de finalização dos processadores 1,2,3 e 4 são 7,9,12 e 14, respectivamente. No passo 3 é calculado OSL = 24. No passo 4, T_{12} é escalonado pela regra 3 no processador 1 e 2. Logo, o processador 2 é apagado do conjunto de processadores diponíveis. T_{11} é escalonado pela regra 2 no processador 1, eliminando-o do conjunto de processadores disponíveis. Finalmente T_{10} e T_9 são escalonadas pela regra 1, nos processadores 4 e 3, respectivamente.

O tempo gasto pelo algoritmo B é determinado como segue: Os passos 1 e 2 consomem tempo O(n). O passo 3 tem um tempo de ordem O(m). O passo 4 pode ser implementado para ser executado em tempo O(m log m). Em geral, o algoritmo consome um tempo de O(n + m log m).

Sobre esses dois algoritmos discutidos de forma resumida nesta seção, foram feitas modificações para a definição de um algoritmo escalonador não preventivo, necessário para a solução do problema.

3.6.4 PLANEJAMENTO DO ESCALONADOR PARA O PROBLEMA.

Da análise dos diferentes tipos de escalonadores, extrai-se apenas o aspecto filosófico e as primeiras aproximações, para chegar a montar um escalonador. Veja [GONZA,1977], [CHEN,1987], [DONGA,1987], [HWANG,1984], [JONES,1980], [SEVEIK,1989], e [ANGER,1990].

Em geral os métodos de escalonamento baseiam-se nas caraterísticas da aplicação, para estabelecerem as heurísticas de solução. Segundo a análise da aplicação, no capítulo 2, existe precedência de blocos e as tarefas que compõem os blocos num primeiro nível de particionamento são totalmente independentes. Por esse motivo, orientou-se o estudo para trabalhos escalonadores de tarefas independentes, em sistemas multiprocessadores, para serem implementados no "sistema original".

Leung e Young [LEUNG,1989], na explicação do algoritmo preventivo para escalonar n tarefas independentes (seção 3.6.3), com o objetivo de minimizar o tempo de execução do algoritmo paralelo, dá sugestões de como fazer um escalonador não preventivo, que foram já consideradas no desenvolvimento deste trabalho.

Foram consideradas as mesmas formulações formais, TS: conjunto de n tarefas independentes, com tempo de execução $[p_1, p_2, \ldots, p_n]$ a serem escalonadas em m)= 1 processadores idênticos; u(s): denota o tempo de finalização das tarefas de TS_i incluindo o retardo e contenção na comunicação.

Seja F(k) a somatória dos tempos gastos pelas tarefas atribuídas ao processador k, na execução de toda a aplicação. OCS (ótimo comprimento do escalonador) dependerá do número de processadores e será um valor que servirá para otimizar a distribuição das tarefas. I é o conjunto de processadores disponíveis.

ALGORITMO ESCALONADOR.

```
1. Inicialize TS, com as tarefa.
2. Distribuir aos m processadores as primeiras m tarefas
   e atualize F(k).
3. OCS ⟵ max ( p_i , ⌈Σp_i /m ⌉ ).
4. WHILE ∃ tarefas DO
     WHILE (I ≠ ∅ and TS ≠ ∅) DO
          (( extrair(TS_i ) ))
          pto.dif,k← 0; menor ← ∞; foi.escalonado ← TRUE
          WHILE (k < m) and foi.escalonado
               IF proc.disp(k)
                  IF p_i ≤ OCS-F(k)
                    « escalonar TS_i »
                    foi.escalonado ← FALSE
                  END-IF
                    dif ← p_i - (OCS - F(k))
                    IF dif ≤ menor
                         pto.dif←k;menor←dif
                    END-IF
               END-IF
               k ← k + 1
          IF   foi.escalonado
               « pegar ponteiro da diferença menor»
               « escalonar TS_i »
          END-IF
     IF ∃ tarefas e TS = ∅
          «carregar as tarefas na fila TS »
     END-IF
     IF ∃ processadores liberados
          « carregar os processadores na fila I »
     END-IF
     OCS ⟵ max ( F(k) + ⌈Σp_i /m ⌉ )
```

fig. 3.13, Algoritmo escalonador não preventivo de tarefas independentes.

O algoritmo escalonador de tareas independentes (figura 3.13), permite observar como será a distribuição das tarefas, que dependendo da operação e o efeito dos retardos de contenção e comunicação, pode não utilizar todos os processadores que estejam disponíveis, se as tarefas são de curta duração e as filas de processadores e tarefas são carregadas em forma concorrente.

3.7 AVALIAÇÃO DAS ARQUITETURAS.

Em geral, o comportamento das redes de interconexão para implementar um multiprocessador, pode ser prognosticado estudando as caraterísticas dos grafos que as modelam. Considerando o custo/benefício, as caraterísticas desejáveis das redes de interconexão [HERRA,1988] e [HERRA,1990] são:

a) Pequeno número de conexões por processador. Isso corresponde à minimização do grau do grafo;
b) Redução máxima dos retardos por comunicação. No grafo, corresponde à minimização do diâmetro (maior distância mínima entre qualquer par de processadores) e a distância média;
c) Alta tolerância a falhas (confiabilidade) para permitir o funcionamento degradado do sistema, em caso de falha dos processadores ou linhas de comunicação. Esse aspecto relaciona-se com a conectividade do grafo;
d) Existência de algoritmos de roteamento simples, considerando a minimização das comunicações.

Também, do ponto de vista da implementação do sistema, é fundamental a possibilidade de realizar uma topologia plana baseada na conexão de blocos básicos, em forma modular, e a facilidade de expansão da topologia.

A escolha entre uma arquitetura forte ou fracamente acoplada, depende da aplicação. Os sistemas com memória compartilhada são mais apropriados para aplicações que necessitam de muita comunicação entre os processadores. A existência da memória global torna mais fácil a troca de informações e o compartilhar de dados. Os sistemas de memória distribuída são mais expansíveis, permitindo o uso de um número maior de processadores.

O uso de muitos processadores num sistema fortemente acoplado, chama a atenção para os seguintes pontos:

a) Alta taxa de acesso à memória global, tornam o barramento o "gargalo" da arquitetura;
b) O uso de arquitetura tipo "crossbar" ou redes multiestágio, podem elevar demais a complexidade e o custo do sistema.

Os sistemas fracamente acoplados também apresentam dificuldades, como as seguintes :

a) Altas taxas de comunicação entre processadores podem levar a um fraco desempenho, principalmente se as mensagens recebidas por cada processador tiverem que ser retransmitidas;
b) A programação num ambiente fracamente acoplado é menos intuitiva do que num ambiente fortemente acoplado.

CAPÍTULO 4

PROJETO E IMPLEMENTAÇÃO DO "SISTEMA ORIGINAL"

4.1 INTRODUÇÃO.

Para o desenvolvimento do projeto e implementação do "software", foram consideradas as premissas mencionadas no capítulo 2, com respeito à aplicação e às partições que podem ser feitas. Desse estudo preliminar e da filosofia de operação das máquinas MIMD, foram extraídas as idéias iniciais para realizar a análise do problema e posterior implementação.

Neste trabalho será denominado "sistema original" a implementação paralela do algoritmo de Linhas e Superfícies Escondidas, sobre um número limitado de processadores, como foi proposto na seção 2.4 (figura 2.7). Como o objetivo está centrado no estudo da solução paralela, os algoritmos específicos do problema de L.S.E. foram substituídos por retardos equivalentes ao tempo de execução do trecho de programa específico.

O "sistema original" foi implementado numa placa Inmos IMS B008, conectada a um PC/XT ou AT. As tarefas resultantes da partição do algoritmo L.S.E. foram simuladas por um "loop" equivalente ao tempo de execução.

Da implementação do "sistema original", foram extraídos os tempos dos diferentes blocos da aplicação, para realizar a verificação e sintonização do simulador, que será apresentado no capitulo 5.

4.2 DETERMINAÇÃO DOS MÓDULOS FUNCIONAIS DO PROBLEMA.

Considerando a análise feita para o algoritmo "Linhas e Superfícies Escondidas", a forma de execução de cada bloco e a filosofia de operação das máquinas MIMD fracamente acopladas, foi

estabelecido o esquema, apresentado na figura 4.1 para a solução do problema, onde cada bloco tem a seguinte função.

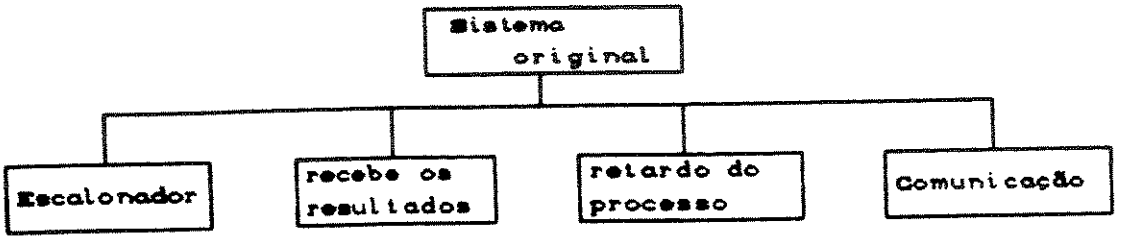


fig. 4.1, Blocos funcionais do esquema para a solução do problema L.S.E.

Escalonador. Esse módulo realiza a distribuição das tarefas aos processadores disponíveis. As tarefas são os particionamentos feitos na aplicação.

Recebe os resultados : Esse módulo realiza as seguintes funções, quando cada tarefa é concluida por um processador:

Atualiza a fila de processadores liberados;

Atualiza as estruturas de dados para o disparo das fases seguintes do processamento.

Retardo do processo : Esse módulo simula o tempo de execução de uma tarefa, através de funções de retardo.

Comunicação : Esse módulo sincroniza e transmite as mensagens na máquina.

Controle de fila : Esse módulo faz o gerenciamento das filas de tarefas (processos) TS e registro da utilização dos processadores disponíveis I.

4.3 ANÁLISE DOS MÓDULOS FUNCIONAIS.

4.3.1 ESCALONADOR.

Em princípio, para realizar a distribuição das tarefas, esse módulo verifica se existe ou não algum processador disponível

na fila. Em caso afirmativo, é atribuída uma tarefa a esse processador e acionado o bloco "comunicação" para a transmissão dos dados. Quando não existem processadores disponíveis na fila e/ou tarefas, é acionado o bloco "recebe resultados" para que realize a atualização dos processadores liberados. A figura 4.2 mostra um diagrama em blocos das atividades realizadas pelo módulo escalonador.

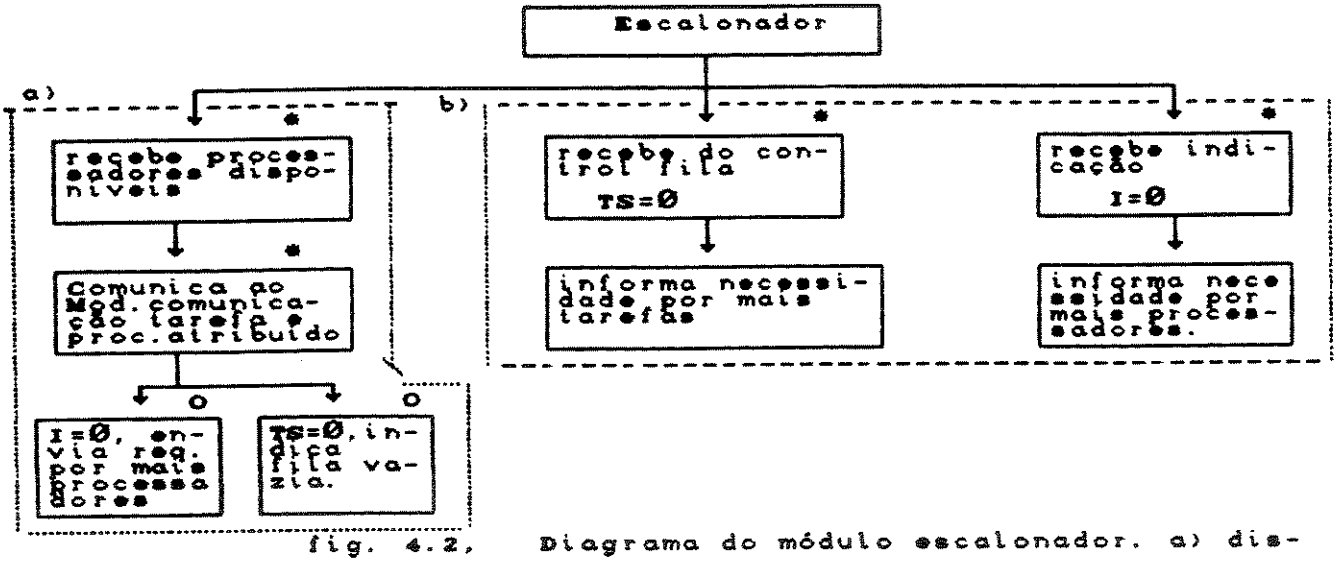


fig. 4.2, Diagrama do módulo escalonador. a) distribui tarefas, b). controlador da fila e comunicação com recebe resultados.

4.3.2 RECEBE RESULTADOS.

Esse módulo atua na recepção dos dados resultantes da execução de uma tarefa num processador, atualizando as estruturas de dados que permitem disparar os processos da fase seguinte. Também realiza a atualização da fila de processadores disponíveis. A figura 4.3 mostra o diagrama do módulo "recebe resultados".

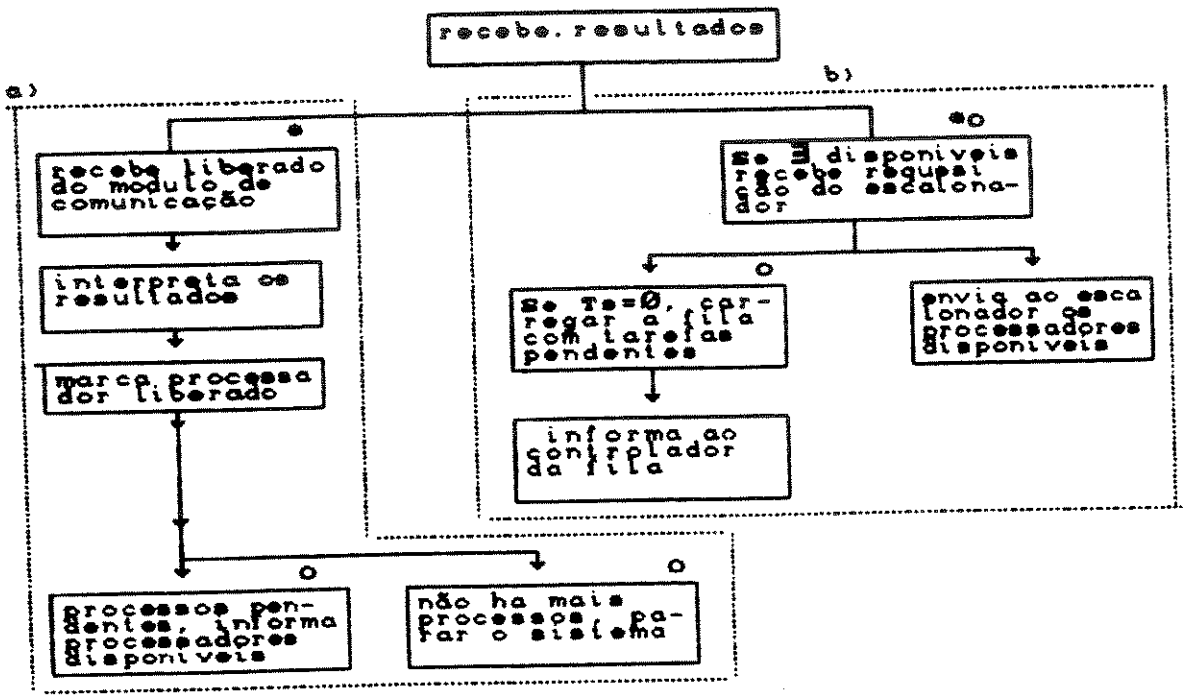


fig. 4.3,Diagrama do módulo "recebe resultados", a) recebe processadores livres e resultados, b) responde requisição ao escalonador.

4.3.3 RETARDO DO PROCESSO.

Esse bloco simula a execução de uma tarefa no processador atribuído, recebe a mensagem do bloco de comunicação e dispara o processo; quando finaliza envia uma mensagem de retorno ao processador remetente. A figura 4.4 apresenta, em detalhes, o diagrama das operações desse módulo.

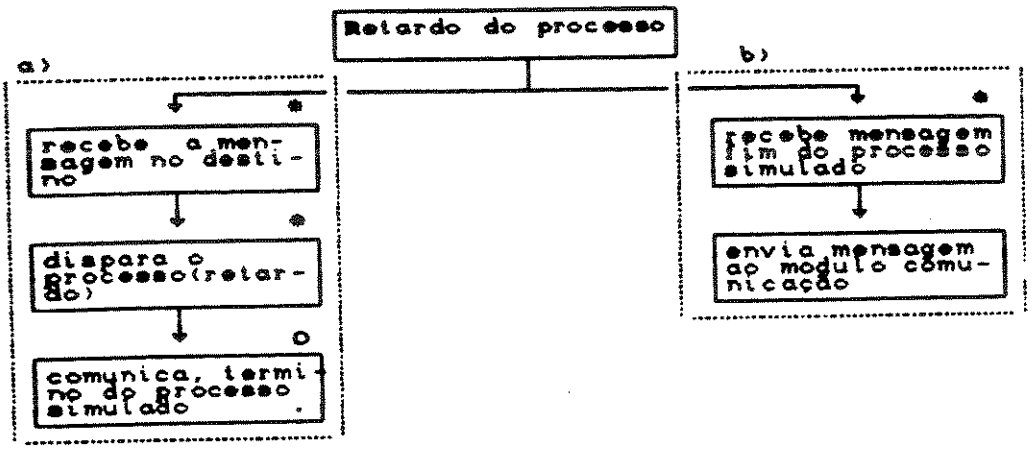


Fig. 4.4, Diagrama do retardo, a) disparo do processo simulado, b) comunica liberação do processador.

4.3.4 COMUNICAÇÃO.

Esse bloco sincroniza as comunicações internas e externas do processador e tem três funções fundamentais, que são:

a) Recebe as mensagens dos canais de entrada do processador; analisa se é o destinatário ou se deve enviá-la a um processador vizinho;
b) Recebe as mensagens, dos processos internos que devem ser enviadas a um processador externo ;
c) Analisa as mensagens e permite fazer o roteamento apropriado segundo a topologia e destino correspondente.

A figura 4.5 mostra o diagrama do bloco "comunicação".

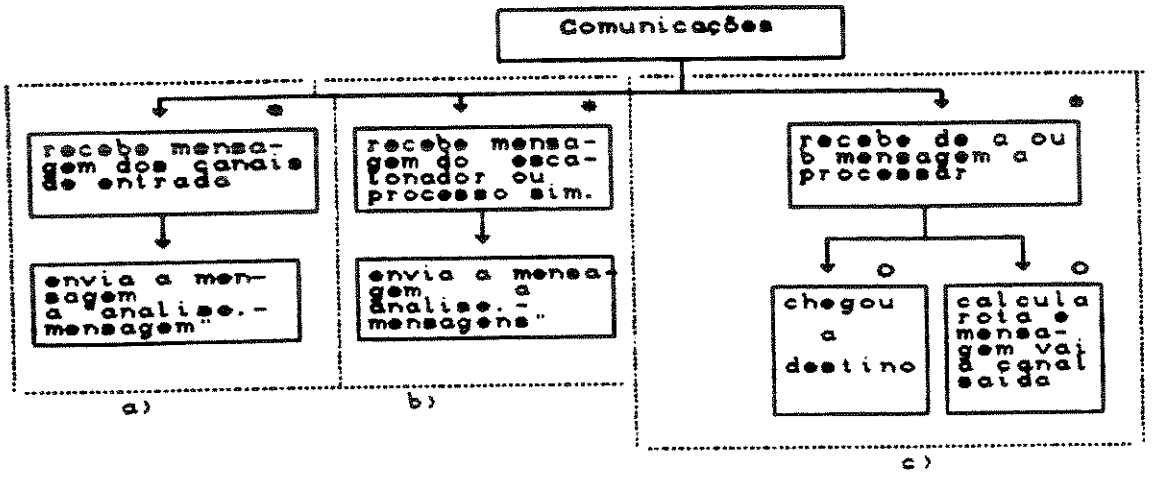


fig. 4.5, Diagrama do módulo Comunicações.
a) ingressa mensagem nos canais de entrada, b) ingressa mensagem do escalonador ou processo simulado, c) analisa destino das mensagens.

4.3.5 CONTROLE DA FILA DE PROCESSOS.

Esse bloco corresponde a uma fila simples, que permite fornecer tarefas ao bloco "escalonador", quando existe um processador disponível. O bloco deve permitir a realização de consultas sobre a existência de tarefas esperando por processamento. A figura 4.6 mostra o diagrama do módulo "controle da fila".

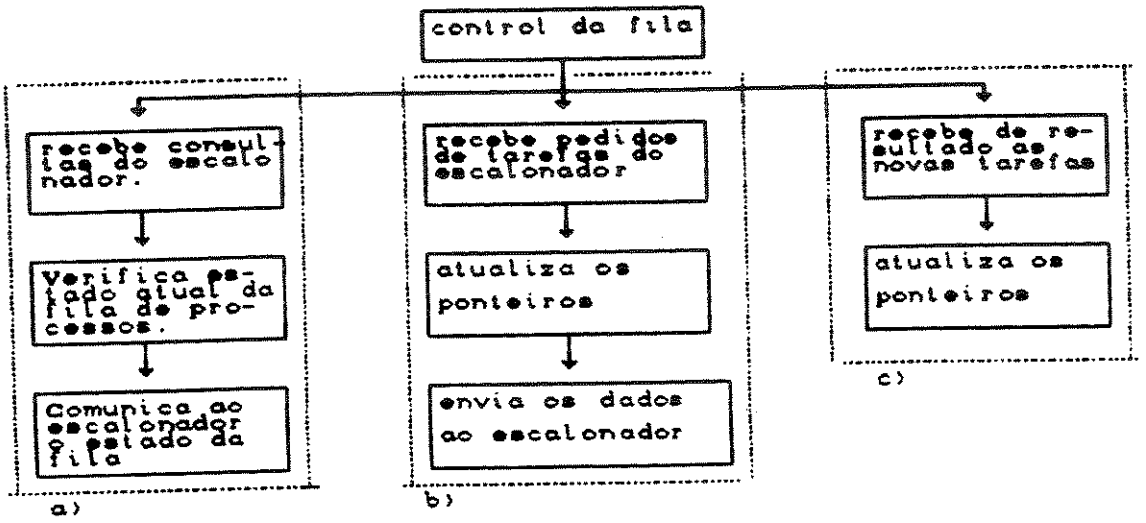


fig. 4.6, Diagrama do controle da fila de processos. a) consulta fila vazia, b) envia ao escalonador uma tarefa T_i c) carrega a fila de tarefas $Ts=Ts+T_i$

4.4 CONSIDERAÇÕES SOBRE A IMPLEMENTAÇÃO.

A fim de descrever a forma como os blocos funcionais interagem de modo concorrente, o problema foi modelado empregando Redes de Petri. Deve-se observar que o conjunto de tarefas a serem executadas é independente e, que para sua execução, será utilizado um conjunto de processadores; um deles atuando como mestre e o resto como escravos.

O processador mestre terá a função de gerenciar a execução das tarefas. O gerenciamento inclui os seguintes processos :

a) Escalonamento das tarefas ("escalonador");
b) Recepção de resultados ("recebe resultado");
c) Controle das filas de processadores disponíveis e tarefas ("controle das filas");
d) Sincronização das comunicações ("comunicação").

Os processadores escravos deverão executar os processos ou tarefas atribuídas e fazer a comunicação de dados e resultados. Os processos são os seguintes :

a) Simular o tempo de execução de uma tarefa ("retardo do processo");
b) Sincronização das comunicações ("comunicação").

Outro aspecto que deve ser mencionado é o tipo de arquitetura que foi empregada. Isto é a arquitetura MIMD fracamente acoplada ou de memória distribuída. Escolha que particulariza a sincronização dos processadores, como foi mencionado na seção 3.4. Além disso, foi usado o módulo Transputer (TRAM) como unidade básica de processamento, composto de memória e um microprocessador Transputer da Inmos [ITRM,1988] e [ITDM,1988]. Esse microprocessador possui "firmware", que permite gerenciar a execução dos processos concorrentes e interfaces de comunicação externa. As interfaces atuam independentemente do processador, utilizando DMA (Direct Memory Access). Mais detalhes sobre transputer podem ser obtidos em Burns [BURNS,1988], Jones [JONES,1988] e no manual Inmos [IORM,1988].

Na sequência se apresentam os diagramas e processos que correspondem ao transputer mestre e ao transputer escravo. Esses processadores estarão conectados por uma malha de interconexão, escolhida pelo usuário para a realização do teste.

4.4.1 DIAGRAMA DE PETRI DO PROCESSADOR MESTRE.

No processador mestre, além dos processos mencionados na seção 4.3, é necessário acrescentar os processos envolvidos nas interfaces de comunicação externa do transputer. Tendo em vista as caraterísticas do hardware do transputer e os processos alocados no processador mestre, devem ser considerados os seguintes blocos e transições para a Rede de Petri (figura 4.7).

TS : fila de processos simulados ou retardos;
I : fila de processadores disponíveis na malha;
Sd : escalonador; algoritmo de distribuição das tarefas aos processadores alocados;
Sa : o algoritmo de comunicação;
Sb : o algoritmo que recebe os resultados dos

processadores liberados, carregando-os na fila de processadores disponíveis (I);

Sce_i : interfaces de entrada dos links conectados ao processador;

Scs_i : interfaces de saídas dos links do processador.

As transições que estão representadas na Rede, que é uma extensão da Rede de Petri, chamada Rede de Petri "Numérica" [SYMON,1980], serão ativadas quando existam as seguintes condições:

1. As transições $tsce_i$ serão ativadas quando exista uma mensagem na entrada e o lugar Sce_i esteja liberado.
2. A transição td_o será ativada quando existam tarefas e processadores disponíveis nos lugares TS e I, e Sd estiver liberado.
3. A transição t_i será ativada quando exista um "token" no Sb (libera um processador e armazena resultados).
4. A transição td_1 será ativada quando exista um "token" em Sd e, Sa e Sb estiverem liberados. Na nomenclatura da extensão da Rede de Petri: quando Sa=0 e Sb=0.
5. As transições te_i serão ativadas quando Sce_i estiver ocupado (Sce_i=1) e os lugares Sa e Sb liberados (Sa=0 e Sb=0).
6. As transições tc_i serão ativadas quando exista uma mensagem para Scs_i, resultante das transições te_i e td.
7. A transição tb será ativada quando exista uma mensagem para Sb, resultante das transições te_i.

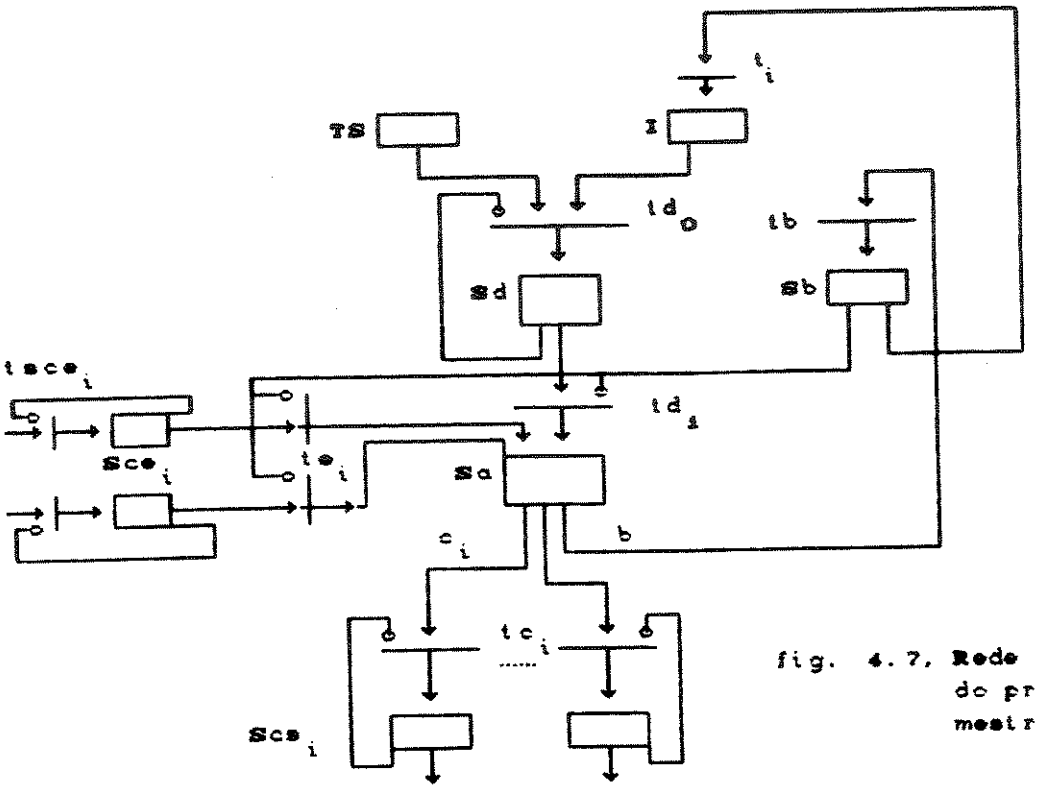


fig. 4.7, Rede de Petri do processador mestre.

O bloco de comunicação deve ter maior prioridade de operação, caso contrário o desempenho do sistema é baixo (Veja [BURNS,1988], [JONES,1988] e [ITDS,1988]). Dessa maneira, os blocos Sa, Sce_i, Scs_i ("eventos que atuam na comunicação") e Sb ("recebe resultados"), atuam em alta prioridade. Os processos alocados em baixa prioridade são Sd, TS e I. A figura 4.8 apresenta um diagrama esquemático das prioridades atribuídas ao processador mestre.

```
PRI PAR
  ...<<< Alta prioridade
     ... As comunicações
     ... Recebe resultados
     >>>
  ...<<< Baixa prioridade
     ... Controle das filas
     ... Escalonador
     >>>
```

fig. 4.8, Atribuição das prioridades no processador mestre.

Cada bloco apresentado na figura 4.7 é uma região

crítica. Isso significa que o recurso deve estar livre para que exista a transição. Quando existe mais de um bloco com atribuição de alta prioridade, ambos devem estar liberados para que um deles possa progredir. A tabela de estado mostrada na figura 4.9 caracteriza a operação do processador mestre.

Nessa tabela os estados das condições são representados por "0" e "1", sendo que "1" representa o estado ocupado e "0" o liberado. As ações atuam da seguinte maneira :

	Condições iniciais								Condições após a execução							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
TS	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
I	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Sa	0	0	1	0	1	x	x	0	x	1	0	0	0	x	x	1
Sb	x	0	0	1	0	x	x	0	x	0	1	0	0	x	x	0
Sce_i	x	x	x	x	x	x	0	1	x	x	x	x	x	x	1	0
Scs_i	x	x	x	x	0	1	x	x	x	x	x	x	1	0	x	x
Sd	0	1	x	x	x	x	x	x	1	0	x	x	x	x	x	x
início distribuição da tarefa	x								x							
distribui a tarefa		x								x						
chega um resultado			x								x					
registra e libera processadores				x								x				
vai ao canal de saída					x								x			
transfere a outro processador						x								x		
ingressa mensagem à entrada i							x								x	
processa mensagem da entrada i								x								x

fig. 4.9, Tabela estado do processador mestre.

a) Início distribuição da tarefa. Ocorre com o estado da condição Sd liberado e TS e I ocupados. A execução dessa ação deixa Sd em estado ocupado.
b) Distribui uma tarefa. Sa e Sb devem estar liberados. A execução dessa ação deixa Sd liberado e Sa ocupado;
c) A chegada de uma mensagem ao bloco Sa através das ações processa mensagem da entrada i ou distribui

uma tarefa, deixa Sa ocupado, executando-se a análise da mensagem. O resultado da ação poderá ativar vai ao canal de saída ou chega um resultado,

d) Se a ação chega um resultado for executada, Sb fica ocupado e é executada registra e libera processador, liberando Sa;

e) Se a ação vai ao canal de saída for executada, Scs fica ocupado, Sa é liberado e, posteriormente, tenta executar a ação pode transferir a outro processador;

f) A chegada de uma mensagem externa é recebida pelo canal de entrada, desde que Sce esteja liberado. Na sequência é executada a ação ingressa mensagem à entrada i.

g) Para processar a mensagem de um canal de entrada, Sa e Sb devem estar liberados.

4.4.2 DIAGRAMA DE PETRI DOS PROCESSADORES ESCRAVOS.

Aplicando os mesmos princípios utilizados para determinar o diagrama do processador mestre, os processadores escravos têm alocados dois processos importantes, a saber: o bloco de comunicação e o retardo do processo (processo simulado). Os blocos são os seguintes:

Sa : representa o algoritmo de análise do destino da mensagem;

Sb : representa o algoritmo de retardo do processo (processo simulado);

Sce_i : representa as interfaces dos links de entrada do processador escravo;

Scs_i : representa as interfaces dos links de saída do processador escravo.

As transições representadas no diagrama da figura 4.10, serão ativadas quando existam as seguintes condições:

1. As transições tse_i serão ativadas quando Sce_i estiver liberado (Sce_i=0).

2. As transições te_i e tb_2 serão ativadas quando Sa estiver liberado (Sa=0).
3. A transição tb será ativada quando, a partir de uma das transições te_i, existir uma mensagem para Sb.
4. As transições tc_i serão ativadas quando, a partir de uma das transições te_i, existir uma mensagem para Scs_i. Scs_i deverá estar liberado, ou seja, a transição é ativada se cs_i=1 (cs_i representa a variável setada resultante da transição te_i) e Scs_i=0, caso contrário fica esperando.

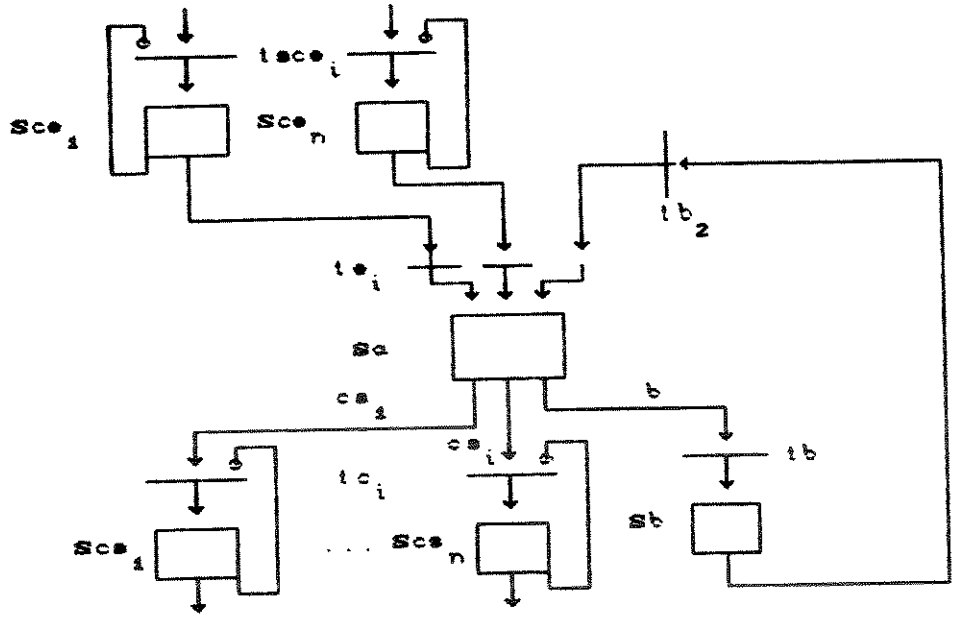


fig. 4.10, Diagrama de Petri do Processador escravo.

As prioridades atribuídas aos processos alocados nos processadores escravos são: bloco de comunicação em alta prioridade e processo de retardo (processo simulado) em baixa prioridade. Dessa forma, sempre que existir um processo de alta prioridade pronto e se o processo de retardo estiver ativo, será interrompida sua execução. A figura 4.11 mostra o esquema de prioridades do processador escravo.

```
PRI PAR
  ...As comunicações
  ... processo simulado
```

fig. 4.11, Prioridades dos processos.

A figura 4.12 mostra a tabela de estados dos processadores escravos. De forma similar à figura 4.9, os estados são representados por "0" e "1", indicando liberado e ocupado, respectivamente.

	Condições iniciais						Condições após a execução					
	1	2	3	4	5	6	1	2	3	4	5	6
Sa	x	0	1	1	x	x	x	1	0	0	x	1
Sb	x	x	0	x	x	1	x	x	1	x	x	0
Sce_i	0	1	x	x	x	x	1	0	x	x	x	x
Sce_i	x	x	x	0	1	x	x	x	x	1	0	x
ingressa mensagem na entrada i	x						x					
processa mensagem		x						x				
começa o processo simulado			x						x			
vai ao canal de saída i				x						x		
pode tranferir a outro processador					x						x	
fim do processo simulado						x						x

fig. 4.12, Tabela de estados dos processos dos processadores escravos.

4.4.3 ANÁLISE DOS DIAGRAMAS DE PETRI.

Em cada ação apresentada nas figuras 4.9 e 4.12, além dos tempos consumidos em cada bloco, devem ser considedos os tempos produzidos pelas transições e os retardos de contenção para que as transições ocorram.

O caso mais simples é aquele onde um canal de saída i do processador x, conectado a um canal de entrada i do processador y, precisa se comunicar. Assuma que, pelo efeito do processamento, o bloco Sa, que representa o algoritmo que analisa o destino da mensagem, está em estado de execução. O canal de saída será

forçado a esperar até que Sa mude de estado e, pela execução da instrução, seja habilitado o canal de entrada i do processador y, para iniciar a transferência. Pode ocorrer que nesse instante vários canais de entrada sejam habilitados e realizem a transferência de forma simultânea, ficando mais de uma entrada pronta para acessar o bloco Sa do processador y. A primeira entrada a ser atendida não acumulará retardo pela espera, mas as restantes deverão ter seus tempos aumentados até serem atendidas.

O tempo que representa o retardo de espera (ou contenção), será a diferença entre o tempo de chegada da mensagem à entrada do bloco e o tempo em que o bloco aceita a transição.

Para cada bloco apresentado no diagrama da figura 4.7, estabelecem-se os seguintes tempos:

Tempo de distribuição: é o tempo resultante da atribuição de uma tarefa a um processador disponível e do roteamento no bloco de comunicação (Sa);

Tempo de análise do destino: Tempo resultante da execução do algoritmo para roteamento da mensagem na topologia;

Tempo de transferência interna: Tempo de transferência das mensagens, quando se produz uma transição;

Tempo de transferência externa: Tempo de retardo resultante da passagem da mensagem através do meio de transmissão;

Tempo de liberação: Tempo que consome a rotina Sb, para armazenar os resultados e deixar o processador na fila de processadores disponíveis.

De maneira análoga, para cada bloco apresentado no diagrama da figura 4.10 estabelecem-se os seguintes tempos :

Tempo de transferência externa: Retardo de tempo resultante da passagem da mensagem através do meio de transmissão;

Tempo de transferência interna: Retardo de tempo

resultante da transferência de mensagem, entre blocos, no processador;

Tempo de análise do destino: Tempo de retardo resultante da execução do algoritmo para o roteamento da mensagem na topologia;

Tempo de execução das tarefas: Tempo atribuido para simular o processo, mais o tempo de interrupção resultante da atribuição de operação em baixa prioridade.

Todos estes tempos são utilizados na construção do Simulador (apresentado no capítulo seguinte), para registrar cada um dos processos que participam na sequência da execução nos processadores, na topologia selecionada.

Os canais de entrada e saída baseiam-se no princípio de operação mencionado no manual Inmos [ITRM,1988], por Shallow [SHALL,1990], Torngrem [TORNG,1990] e Burns [BURNS,1988], registrando seus tempos por links conectados em cada processador que participa na rede.

4.5 IMPLEMENTAÇÃO DO SISTEMA ORIGINAL EM MÁQUINAS MIMD (USANDO TRANSPUTERS).

Existem três aspectos importantes a serem considerados na implementação física do "sistema original", utilizando uma placa multi-transputers.

O primeiro aspecto é o Hardware utilizado. Nesse caso será empregada uma placa com capacidade de conectar dez transputers chamada IMS B008 [IB008,1988].

O segundo aspecto é a forma de configurar a placa. por meio da utilização do software MMS2 (Module Motherboard Software) que será comentado posteriormente, podem-se estabelecer as conexões através da programação do chip IMS C004. A forma de programar é diretamente relacionada com a configuração selecionada para o "hardware" quando a placa é montada. Esse módulo de

programação possui todas as alternativas necessárias para deixar a placa pronta para ser utilizada pelo TDS2 e o programa de aplicação. Maiores detalhes ver [IMMS,1989].

Por último, a exigência de programação imposta pela ferramenta (TDS2 e a linguagem OCCAM2), onde devem ser desenvolvidos módulos independentes e sem variáveis globais. Dessa maneira, os nós processadores escravos têm módulos compilados separadamente. Mais detalhes desse aspecto são fornecidos n manual Inmos [ITDS,1988].

4.5.1 PROJETO DA TOPOLOGIA HIPERCUBO NA PLACA IMS B008.

A placa B008 é altamente flexível, podendo ser programada por "software" e "hardware". Ela possui dez "slots" para módulos transputers(TRAMs), conectando fisicamente todos os TRAM (n) ao TRAM (n+1), mediante os links 2 e 1, respectivamente.

Além dos TRAMs já citados existe um módulo transputer IMS T212, que permite a configuração da malha "crossbar" implementada pelo componente IMS C004. O IMS C004 possui 32 links de entrada, 32 links de saída e o "link" de configuração. A placa B008 foi planejada de tal maneira que o TRAM0 "link" 3 e TRAM(1..9) links 0 e 3 estão diretamente ligados ao IMS C004. Existe também a possibilidade de conectar o TRAM0 "link" 0 ao IMS C004 através da área de conexões.

Consideradas essas condições, a placa foi configurada da seguinte maneira: TRAM0 "link" 0 conectado ao IBM PC através do IMS C012, "link" 1 conectado ao IMS T212 e os "links" 2 e 1 do TRAM3 e TRAM4 desconectados. Deste modo duas cadeias de TRAMs em linha são formadas pelos TRAMs(0..3) e TRAMs (4..9) (a figura 4.13 mostra a configuração da placa IMS B008).

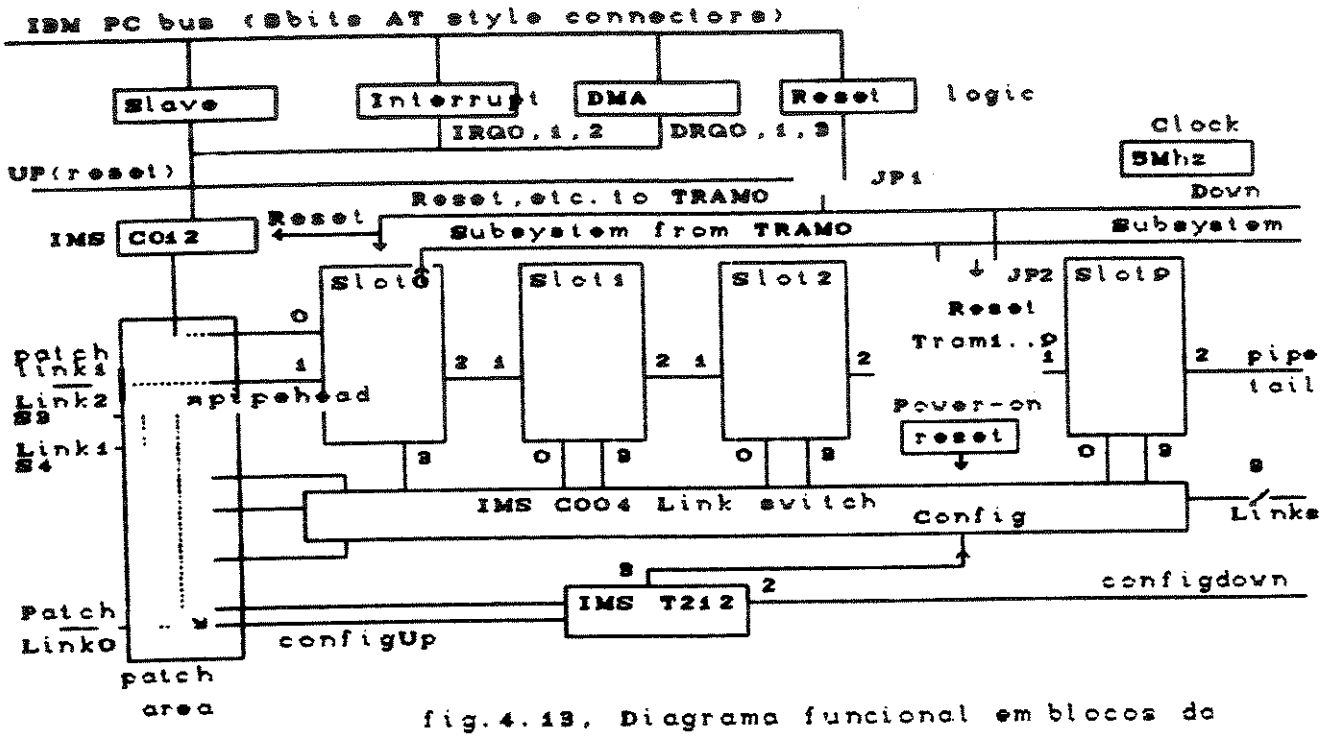


fig.4.13, Diagrama funcional em blocos da placa IMS B008

O software que permite programar a topologia desejada na placa IMS B008 é denominado MMS2. Basicamente, esse pacote trabalha com dois arquivos que descrevem a configuração existente do hardware (arquivo "hardwire") e um arquivo que descreve as mudanças físicas desejadas para configurar uma topologia (arquivo "softwire").

Para a utilização do software MMS2, é necessário o conhecimento da linguagem de especificação para a criação do arquivo "softwire" [IMMS,1989]. O software MMS2 fornece também um sistema de opções que permite fazer operações de inicialização do C004, teste dos arquivos fontes (hardwire e softwire), mudanças dos "links", "reset" do subsistema, visualização na tela do mapeamento da malha configurada e diagnóstico.

A figura 4.14 mostra a topologia mapeada (arquivo softwire). A tabela 1 apresenta o mapeamento fornecido pelo software MMS2, de acordo as especificações fornecidas pelo arquivo "softwire".

```
SOFTWIRE

  PIPE 0

     SLOT  1, LINK 0 TO SLOT  5, LINK 3
     SLOT  2, LINK 0 TO EDGE  0
     SLOT  3, LINK 0 TO EDGE  1
     SLOT  5, LINK 0 TO EDGE  2
     SLOT  6, LINK 0 TO SLOT  9, LINK 3
     SLOT  7, LINK 0 TO SLOT  3, LINK 3
     SLOT  8, LINK 0 TO SLOT  2, LINK 3
     SLOT  9, LINK 0 TO SLOT  1, LINK 3
     SLOT  6, LINK 3 TO EDGE  3
     SLOT  7, LINK 3 TO EDGE  4
     SLOT  8, LINK 3 TO EDGE  5

  END
```

fig. 4.14. Descrição do arquivo "softwire".

TABELA 1 Mapeamento da malha Hipercubica.

INMOS MODULE MOTHERBOARD
NETWORK MAPPER

THE TOTAL NUMBERS OF TRANSPUTERS FOUND IS 9

LINKS	0	1	2	3
id. tram type	id. link	id. link	id. link	id. link
MT * T800d	---	---	0 * 1	---
0 * T800d	3 * 3	MT * 2	1 * 1	7 * 0
1 * T800d	---	0 * 2	2 * 1	6 * 0
2 * T800d	---	1 * 2	3 * 1	5 * 0
3 * T800c	---	2 * 2	4 * 1	0 * 0
4 * T800d	7 * 3	8 * 2	5 * 1	---
5 * T800d	2 * 3	4 * 2	6 * 1	---
6 * T414b	1 * 3	5 * 2	7 * 1	---
7 * T414b	0 * 3	6 * 2	---	4 * 0

Para estabelecer os enlaces dos transputers de forma correta, devem-se associar os canais físicos aos canais lógicos, que representam as entradas e saídas de cada nó processador.

As atribuições feitas pela INMOS para os "links" físicos são de 0..3 para as saídas (link0out,..,link3out) e 4..7 para as entradas (link0in,..,link3in). Os arranjos "tos.out" e "tos.inp" foram carregados com esses valores, de acordo com a numeração fornecida pelo Mapa do MMS2, fazendo corresponder uma matriz de 8 filas e 3 colunas, num arranjo de dimensão um com 24 elementos.

Assim os arranjos ficam, como mostra o segmento do programa apresentado na figura 4.15.

```
--<<<  mapeamento dos links fisicos
VAL [24]INT los.out IS [2,0,3,1,2,3,2,1,3,1,3,2,2,0,1,1,2,0,2,1,
                        0,1,3,0]:

VAL [24]INT los.inp IS [6,4,7,5,6,7,6,5,7,5,7,6,6,4,5,5,6,4,6,5,
                        4,5,7,4]:

-->>>

          --<<<  declaracao INMOS dos link
               VAL link0out     IS  0:
               VAL link0in      IS  4:
               VAL link1out     IS  1:
               VAL link1in      IS  5:
               VAL link2out     IS  2:
               VAL link2in      IS  6:
               VAL link3out     IS  3:
               VAL link3in      IS  7:
          -->>>
```

fig. 4.15. Declaração dos links do sistema.

Para estabelecer o mapeamento dos endereços da topologia hipercúbica (numeração binária), foi preciso criar uma estrutura de dados que relacione os canais lógicos que entram e saem dos nós processadores. Os arranjos "sale" e "entra" foram carregados com esses números, de forma análoga aos anteriores, para ter uma atribuição automática na hora de localizar os processos e conexões necessárias para rodar o programa de aplicação. A declaração dos canais lógicos é apresentada na figura 4.16.

```
--<<<  mapeamento dos links lógicos
VAL [24]INT sale IS [1,10,20,0,11,21,2,9,23,3,8,22,7,12,18,6,13,
                     19,4,15,17,5,14,16]:

VAL [24]INT entra IS [0,8,16,1,9,17,3,11,19,2,10,18,6,14,22,7,15,
                      23,5,13,21,4,12,20]:

-->>>
```

fig. 4.16. Declaração dos canais lógicos.

Definidos a topologia e os mapeamentos necessários, bem como as estruturas de dados que permitem realizar as atribuições em forma apropriada, o passo a seguir é estabelecer quais processos vão se localizar em cada nó processador da malha criada.

4.5.2 ALOCAÇÃO DOS PROCESSOS NA TOPOLOGIA.

Pelas características impostas pela placa IMS B008, existe um TRAM "host" e um TRAM "root", denominados MT e 0, respectivamente, como mostra a figura 4.17. O MT possui uma capacidade de memória RAM de 2 Mbytes e está conectado ao IBM PC, os outros TRAMs possuem somente 32 Kbytes de RAM. Isso de alguma maneira, condiciona o número dos processos a serem executados em cada TRAM da malha.

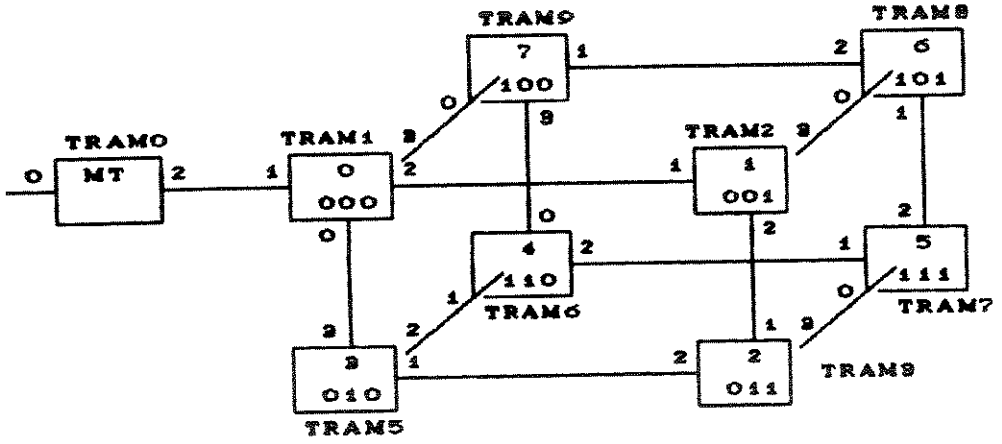


fig. 4.17. representação gráfica da tabela 1.

Cada TRAM escravo possui dois processos que são:

a) O **módulo de comunicação**, como foi mencionado na seção 4.3.2, atua em alta prioridade de operação;
b) O processo de retardo, que representa o tempo de execução do segmento de programa simulado. Esse processo opera em baixa prioridade.

O MT possui todos os processos que permitem monitorar o comportamento do programa, distribuir as tarefas (dados) a serem executadas em cada TRAM escravo, receber e atualizar a fila de processadores liberados, receber os resultados e atualizar a estrutura de dados que fornecerá os resultados finais.

O TRAM "root" será o processador de enlace das comunicações entre o TRAM "host" e a rede de processadores

escravos.

A figura 4.18 mostra o segmento de programa que estará residindo no TRAM "host". A figura 4.19 mostra o "folding" que estará residindo nos nós escravos.

```
<<< F monitor
... declarações globais
... PROC manejo da fila de tarefas ( problema mapeado)
... PROC escalonador das tarefas ( spt modificado )
... PROC recebe nós liberados, armazena resultados
... programa principal
>>>
<<< programa principal
... declarações de variáveis e constantes
SEQ
  ... tela inicial e captura dos dados gerais
  ... classificação dos processadores
  ... leitura do arquivo (problema mapeado)
  ... início dos processos concorrentes
>>>
<<< início dos processos concorrentes
... declaração dos links físicos
... atribuição dos canais do processador
PAR
  control.fila()
  recebe.resultados(total.nos, maestro, entrada.do.tram0)
  escalonador(total.nos, dados.do.prob.lse, total.tarefas,
              maestro, saida.ao.tram0)
>>>
```

fig. 4.18. Planejamento do programa no TRAM "host".

```
<<< F escravos de 1 .. max.transputers na   malha
#USE   "ncubolib.tsr"          --  lib. fornece os protocolos e
                                   constantes  da malha montada
PROC processos.nos.escravos (VAL INT fonte,
                             CHAN OF mensagem0 inp0,inp1,inp2,
                             CHAN OF mensagem0 out0,out1,out2)
  ... declaraçõe s locais
  SEQ
    ... atribuição dos dados fornecidos pela livraria
    PRI PAR
      ... as comunicaçõe s
      ... retardo do processo
>>>
```

fig. 4.19. Módulo dos processadores escravos.

4.5.3 PROGRAMA DE CONFIGURAÇÃO E CARGA DA MALHA N-CUBO.

Finalizando, na malha existirão três grandes processos denominados: monitor (TRAM MT), gerenciador (TRAM0) e escravos (TRAMs 1..7). A exigência para alocar programas nos processadores TRAMs 0..7 é ter todos os módulos compilados separadamente ("folding" SC). Os escravos devem estar dentro de um "folding" PROGRAM para poder carregar a malha, por meio do utilitário LOAD NETWORK do TDS2.

A figura 4.20a mostra a estrutura principal do "folding" PROGRAM. A parte b da figura 4.20 mostra a configuração feita e, por último, a figura 4.21 apresenta a definição da topologia e como estão associados os canais dos arranjos "entra" e "sale" do programa de configuração.

```
<<< F topologia
#USE "ncubolib.tsr"            -- LIB de protocolos e dados  da
                                  malha
... SC processos nós escravos
... SC maestro gerenciador
... Configuração dos TRAMs na IMS B008
>>>
```

fig. 4.20a. "folding" principal do PROGRAM.

```
<<< configuração dos TRAMs na IMS B008"

--<<<  declarações gerais
[24]CHAN OF mensagemO mensajero:
CHAN OF mensagemO entrada.mt, saida.mt :
-->>>

PLACED PAR
  PROCESSOR 0 T4

    --<<<  atribuição dos canais e link no processador 0
    -- entrada

    PLACE entrada.do.              AT link1in :
    PLACE mensajero[entra[0]] AT los.inp[0] :
    PLACE mensajero[entra[1]] AT los.inp[1] :
    PLACE mensajero[entra[2]] AT los.inp[2] :
    -- saida
    PLACE saida.ao.mt              AT link1out :
    PLACE mensajero[sale[0]]  AT los.out[0] :
    PLACE mensajero[sale[1]]  AT los.out[1] :
    PLACE mensajero[sale[2]]  AT los.out[2] :
    -->>>

    comunicação.dos.processos( entrada.do.mt,
                               mensajero[entra[0]],
                               mensajero[entra[1]],
                               mensajero[entra[2]],
                               saida.ao.mt,
                               mensajero[sale[0]],
                               mensajero[sale[1]],
                               mensajero[sale[2]])

  PLACED PAR l = 1 FOR (max.processor-1)

    PROCESSOR l T4
      --<<<  atribuição dos canais e link no processador l
      -- entrada
      PLACE mensajero[entra[3*l]]       AT los.inp[3*l] :
      PLACE mensajero[entra[(3*l)+1]]   AT los.inp[(3*l)+1] :
      PLACE mensajero[entra[(3*l)+2]]   AT los.inp[(3*l)+2] :
      -- saida
      PLACE mensajero[sale[3*l]]        AT los.out[3*l] :
      PLACE mensajero[sale[(3*l)+1]]    AT los.out[(3*l)+1] :
      PLACE mensajero[sale[(3*l)+2]]    AT los.out[(3*l)+2] :
      -->>>
      processos.nos.escravos( l,
                              mensajero[entra[3*l]],
                              mensajero[entra[(3*l)+1]],
                              mensajero[entra[(3*l)+2]],
                              mensajero[sale[3*l]],
                              mensajero[sale[(3*l)+1]],
                              mensajero[sale[(3*l)+2]])

>>>
```

fig. 4.20b. Programa da configuração e carga da malha.

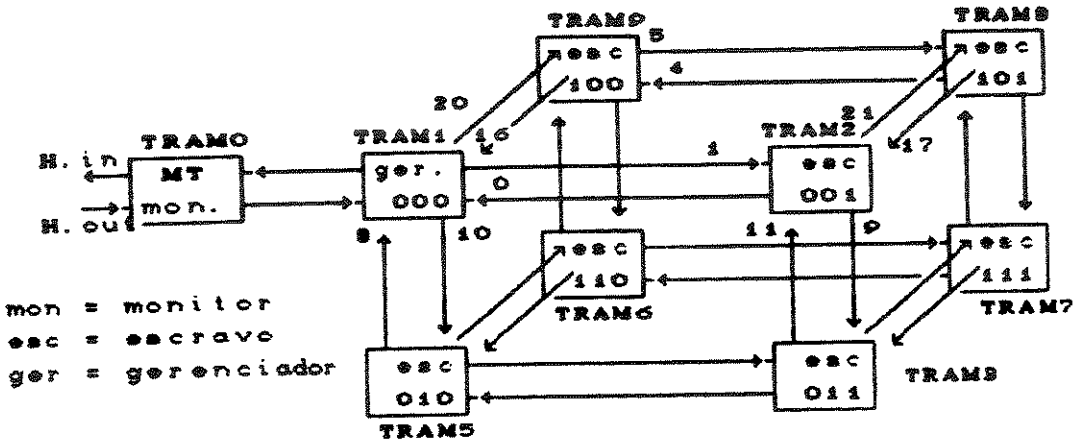


fig. 4.21, Mostra os processos localizados em cada TRAMs da placa e a atribuição dos canais para comunicar os nós de acordo com a numeração binária do Hipercubo.

Para executar o programa desenvolvido na configuração apresentada na figura 4.21, é necessário executar a seguinte sequência:

1.- Mediante o MMS2, setar a placa IMS B008 e fazer as conexões correspondentes;
2.- Compilar o "folding" PROGRAM;
3.- Executar o utilitário LOAD NETWORK;
4.- Carregar o executável do programa Monitor;
5.- Executar o programa monitor;
6.- Responder as opções da tela e esperar a finalização da execução.

4.5.4 AS IMPLEMENTAÇÕES EM OUTRAS TOPOLOGIAS.

Foram fisicamente implementadas mais quatro topologias na placa IMS B008, aproveitando o máximo possível as caraterísticas do hardware, como mostra a figura 4.22.

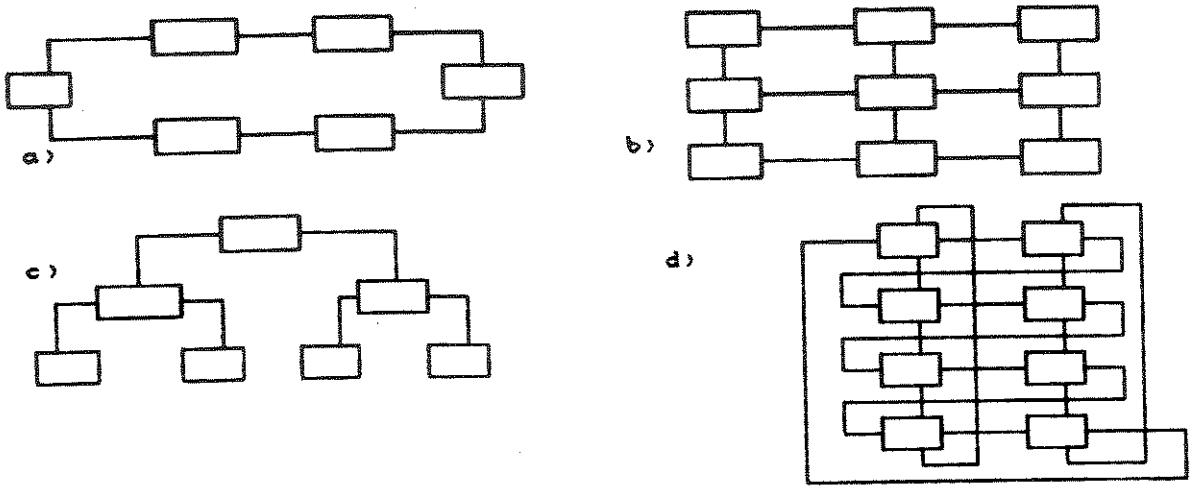


fig. 4.22. Topologias implementadas.
a) Anel; b) "Mesh";
c) Árvore; d) "Midimew".

Os programas necessários para configurar as demais topologias, empregando o mesmo tipo de alocação, descrito no item 4.5.1, substituem somente o "folding" PROGRAM, que contêm os processos de cada nó escravo e sua configuração na topologia. A figura 4.23 mostra o "folding" geral da implementação do "sistema original".

```
<<< Sistema original
... Livraria
... EXE programa monitor
... PROGRAM anel
... PROGRAM arvore
... PROGRAM hipercubo
... PROGRAM mesh
... PROGRAM midimew
>>>
```

fig. 4.23. "Folding" geral do simulador.

O objetivo dessas implementações foi obter tempos de execução, dos módulos e submódulos principais, para caracterizar as diferentes partes do simulador de topologias. Também, foi de interesse obter parâmetros de desempenho globais das diferentes topologias.

4.6 MEDIDAS DOS TEMPOS NO PROCESSADOR MESTRE.

O processador mestre, em qualquer topologia, terá os mesmos processos concorrentes, diferenciando-se apenas pelos

processos envolvidos nas comunicações.

Na seção 4.4.1, foram apresentados os processos que atuam no processador mestre. Na seção 4.4.3, foram definidos os tempos que devem ser considerados, resultantes das transições e execução dos processos alocados nesse processador.

Os tempos de interesse do processador mestre são os tempos de distribuição, liberação e comunicação. Fisicamente, esses tempos dependem da topologia do trabalho e do comprimento da mensagem (veja seção 4.5, fig.4.22).

A dependência do tamanho da mensagem deve-se a forma de operação das máquinas MIMD fracamente acopladas e do modo como são sincronizados os processos concorrentes no transputer.

Existe um caso especial onde os tempos de distribuição e liberação dependem da topologia, resultante da utilização da placa IMS B008, do mapeamento da topologia cúbica e da malha com "wrap-arround". O processador mestre está composto por dois transputers, um para o gerenciamento e o outro para atuar de interface com a topologia (veja figura 4.21 e 4.22). O que implica que os tempos de distribuição e liberação para essas topologias sejam compostos. A seguir, serão quantificados os tempos descritos na seção 4.4.3, através de testes. E, posteriormente, serão determinados os tempos compostos.

4.6.1 TEMPO DE DISTRIBUIÇÃO.

Esse tempo corresponde à atribuição de uma tarefa a um processador disponível e ao envio da mensagem à rotina de comunicações para o roteamento na topologia.

Para medir esse tempo de forma independente, foi montado o esquema apresentado na figura 4.24:

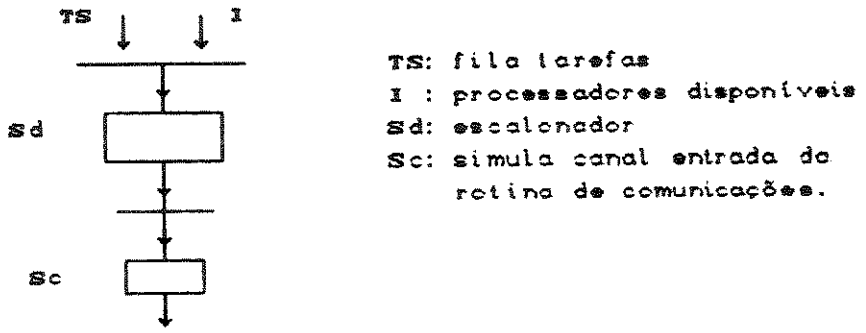


fig. 4.24. Esquema de medição do tempo de distribuição.

As filas TS e I foram carregadas para fornecer os dados à rotina de distribuição. Sd pega uma tarefa da fila de tarefas, atribui ela a um processador disponível da fila de processadores e envia a mensagem a Sc. Sc é um processo concorrente, que recebe a mensagem e registra o tempo do "timer" de medição. Os processos apresentados no esquema (fig. 4.24) são dispostos de maneira que seja aproveitado o "timer" em alta prioridade, que produz um tick a cada 1 μs. São registrados os tempos de um conjunto de tarefas enviadas, de tal maneira que, posteriormente, possa se obter um valor médio das medidas realizadas. Essa operação é repetida para vários comprimentos de mensagens (100 a 600 Bytes).

A tabela 4.2 reflete que o tempo de distribuição é linear e pode ser escrito na forma da equação 4.1.

$$t.dist = 0.31\ f + 57.0\ \mu s \qquad (4.1)$$

f	600	500	400	300	200	100
t	244.0	211.0	180.0	150.0	119.0	89.0

onde f : comprimento da mensagem em bytes.
t : tempo em micro segundos

Tabela 4.2. Tempos da rotina de distribuição.

4.6.2 TEMPO DE LIBERAÇÃO.

Esse tempo é gasto quando chega uma mensagem à rotina "recebe resultados", onde são armazenadas as respostas e realizada a liberação do processador, marcando o sinal disponível na fila de

processadores. Como na seção 4.6.1, foi montado um esquema (fig. 4.25), onde o gerador das mensagens, simulando o retorno ao processador mestre, envia as mensagens ao canal de entrada da rotina "recebe resultados" . Na última instrução, do módulo Sb, capta-se o valor do "timer", registrando a medida. Do conjunto de medidas extrai-se o valor médio gasto na operação.

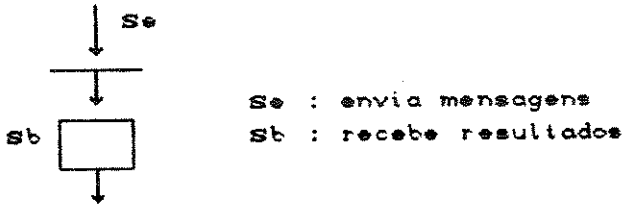


fig. 4.25, Esquema para medir o tempo de liberação.

A tabela 4.3, mostra os tempos registrados para a rotina de liberação ou "recebe resultados" e a equação 4.2 descreve o comportamento.

$$t.lib = 0.2752\ f + 23.36\ \mu s \qquad (4.2)$$

f	600	500	400	300	200	100
t	188.8	161.6	134.4	106.4	78.4	51.2

onde f : comprimento da mensagem em bytes.
t : tempo em micro-segundos.

Tabela 4.3, Tempos da rotina de liberacao.

4.6.3 TEMPO DE TRANSFERÊNCIA INTERNA.

É o tempo de comunicação entre dois processos concorrentes, através de um canal interno. Quando o primeiro processo executa a instrução de saída, a identificação da mensagem é armazenada no canal. O processador retira da execução o primeiro processo e escalona o segundo processo. Quando o segundo processo executa a instrução de entrada, a mensagem é copiada e o primeiro processo é agregado à lista do escalonador.

Para medir o tempo de transferência interna foi realizado um esquema com dois processos concorrentes, um emissor e

um receptor, obtendo-se um valor médio das medições. A tabela 4.4 mostra os dados obtidos e a equação 4.3 reflete o comportamento linear do tempo de transferência interna.

$$t.in = 0.3244\ f + 10.68\ \mu s \qquad (4.3)$$

f	600	500	400	300	200	100
t	179.7	148.1	118.1	88.2	54.2	16.9

onde f : comprimento da mensagem em bytes
t : tempo em micro-segundos.

Tabela 4.4, Tempo da transição entre processos.

Teoricamente, as instruções de entrada e saída na linguagem OCCAM2 consomem, cada uma, 2w+19 ciclos de processador, onde w é o tamanho da mensagem em palavras. Além disso, há ainda, o tempo gasto pelo processo emissor, para sair e voltar ao estado de execução. No lado receptor está o tempo gasto para entrar, sair e voltar ao estado de execução do processo receptor da mensagem. Burns [BURNS,1988] menciona que esse tempo adicional é de 19 a 58 ciclos de processador.

4.6.4 TEMPO DE TRANSFERÊNCIA EXTERNA.

Quando uma instrução de comunicação de saída externa é executada, são copiados o cabeçalho da mensagem e seu comprimento no registro da interface do link de saída. Posteriormente o processo que emite a mensagem deixa o estado de execução.

A interface do link de saída opera da seguinte maneira :

a) Da mensagem extrai-se palavra por palavra da memória, utilizando DMA;
b) A mensagem é transmitida byte por byte através do link;
c) Cada vez que um byte é enviado, a interface espera por um sinal de reconhecimento ACK;
d) Quando se recebe o ACK final, o processador muda o

estado do processo emissor, para que possa entrar em execução, e o canal é "resetado".

Quando se executa uma instrução de entrada no processador receptor, realiza-se a cópia do tamanho da mensagem na interface do link de entrada, para dar início à transferência. Depois, o processo receptor deixa o estado de execução.

A interface do link de entrada recebe a mensagem byte por byte, sendo cada byte é indicado por um sinal ACK. Na interface os bytes são colocados num buffer em palavras que são escritas na memória utilizando DMA.

Quando a comunicação termina, o processo receptor é alocado no final da lista de espera para a execução do escalonador. Se o processo é de alta prioridade, a execução de qualquer processo de baixa prioridade será interrompido.

O tempo de transferência foi medido de acordo com o esquema apresentado na figura 4.26. Foram utilizados dois transputers, um para emitir as mensagens e outro para recebê-las, registrando-se os tempos gastos na transferência. Os dados armazenados são enviados ao primeiro processador, que está conectado ao IBM PC XT, para mostrá-los na tela.

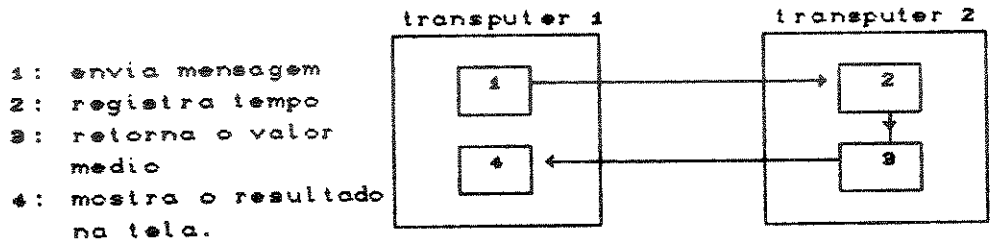


fig. 4.26, Esquema para medir o tempo de transferência externa.

Essa operação é repetida para mensagens de diferentes comprimentos.

Teoricamente, a velocidade de transferência é fixada por Hardware [ITRM,1988]. Nesse caso a velocidade de transferência dos

links foi fixada em 20 Mbits (apenas o link 0 do mestre está a 5Mbits, pois está conectado ao IBM PC XT ou AT).

Adicionalmente deve considerar-se o tempo necessário para:

a) Inicialização das interfaces de entrada e saída;

b) Retirar da execução o processo receptor;

c) Escalonar o processo receptor;

d) Iniciar a execução do processo emissor.

Os dados registrados são apresentados na tabela 4.5 e a equação 4.4 descreve o comportamento linear do tempo de transferência externa.

$$t.ext = 2.2498\ f - 4.56\ \mu s \qquad (4.4)$$

f	600	500	400	300	200	100	50
t	1354.4	1129.5	904.4	679.5	454.4	229.5	116.9

onde f : comprimento da mensagem em bytes.
t : tempo em micro-segundos.

Tabela 4.5, Tempo de tranferência dos canais entre processadores.

4.6.5 ANÁLISE DO TEMPO NA DETERMINAÇÃO DO DESTINO.

Esse tempo corresponde à rotina de comunicação da topologia. Neste trabalho, foram consideradas as seguintes topologias: anel, árvore, hipercúbica, malha e midimew (malha com "wrap-arround"). O número de canais de entrada e saída depende da topologia. A figura 4.27 mostra a estratégia para medir os tempos nos algoritmos de comunicação.

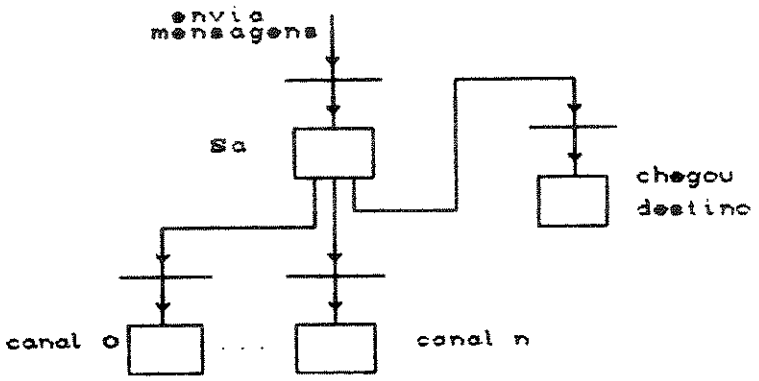


fig. 4.27. Esquema para medir o tempo no processo análise do destino.

O esquema montado serve para medir o tempo que consome a rotina de comunicação. Para isso, são enviadas dez mensagens para cada canal que participa na comunicação da topologia. Também observa-se na fig. 4.27, que os blocos que representam os canais de chegada ao destino, registram os tempos que a rotina gasta em determinar o destino. No final, mostra-se a média dos tempos registrados. Tais valores são apresentados na tabela 4.6. Cabe mencionar, que a rotina de medida realizada consome 9 μs, que devem ser subtraídos do tempo total.

a)

t \ f	600	500	400	300	200	100
c1	392.8	331.4	270.8	209.6	148.5	66.9
c2	392.4	331.5	270.2	209.2	148.0	85.5

$t.c_1 = 0.6518\ f + 9.14\ \mu s$ $t.c_2 = 0.6518\ f + 8.64\ \mu s$

b)

t \ f	600	500	400	300	200	100
c1	394.8	333.9	272.7	211.7	150.6	81.6
c2	606.4	514.8	428.2	331.8	239.9	126.2
c3	605.0	513.4	421.7	330.2	238.4	125.0

$t.c_1 = 0.6264\ f + 25.56\ \mu s$ $t.c_2 = 0.9604\ f + 48.7\ \mu s$
$t.c_3 = 0.96\ f + 42.5\ \mu s$

c)

t \ f	600	500	400	300	200	100
c1	401.8	340.9	279.8	218.7	154.5	85.7
c2	408.8	347.8	286.8	225.6	164.5	79.6
c3	414.6	353.6	292.4	231.4	170.2	88.5

$t.c = 0.6609\ f + 7.4(c-1) + 19.92\ \mu s$

d)

t \ f	600	500	400	300	200	100
c1	405.9	345.1	283.8	222.8	161.6	72.1
c2	406.5	345.7	284.4	223.3	162.2	82.8
c3	405.6	344.6	283.5	222.5	161.2	82.0
c4	405.5	344.5	283.3	222.3	161.0	81.8

$t.c_1 = 0.611\ f + 90.05\ \mu s$ $t.c_2 = 0.611\ f + 91.0\ \mu s$
$t.c_3 = 0.611\ f + 90.0\ \mu s$ $t.c_4 = 0.611\ f + 29.75\ \mu s$

e)

t \ f	600	500	400	300	200	100
c1	413.2	352.2	291.1	230.4	168.8	89.1
c2	417.6	356.4	295.4	234.3	179.1	97.5
c3	415.2	354.2	293.1	232.0	170.9	95.3
c4	413.5	352.5	291.4	230.5	169.1	93.5

$t.c_1 = 0.611\ f + 97.60\ \mu s$ $t.c_2 = 1.29025\ f + 41.9\ \mu s$
$t.c_3 = 1.29025\ f + 99.7\ \mu s$ $t.c_4 = 1.23\ f + 97.9\ \mu s$

onde f : comprimento das mensagens em bytes.
t : tempo em micro-segundos.
c : canal.

Tabela 4.6. tempos dos algoritmos de comunicacao, a) anel, b) arvore, c) hipercubica, d) malha, e) midimev.

4.7 MEDIDAS DOS TEMPOS NO PROCESSADOR ESCRAVO.

Na seção 4.4.3 foram também definidos os tempos de interesse com relação aos processos alocados nos processadores escravos. O processo de comunicação é o mesmo em toda a topologia, razão pela qual foram apresentados, na seção 4.6, os tempos de transferência interna e externa e o tempo na determinação do destino, faltando somente apresentar o tempo de retardo.

O tempo de retardo, como foi mencionado na seção 4.4.3, corresponde ao tempo de execução da tarefa atribuída ao processador escravo, simulada mediante um conjunto de instruções da linguagem utilizada na codificação do problema.

Foi criado um módulo básico com dois "loops" aninhados, de maneira que a execução do "loop" interno corresponda à execução de um retardo de 64 μs.

CAPÍTULO 5

PLANEJAMENTO E IMPLEMENTAÇÃO DO SIMULADOR

5.1 INTRODUÇÃO.

Existem dois aspectos a serem considerados em qualquer estratégia para construir um simulador. A primeira, é assegurar que a simulação seja uma representação do sistema real. A segunda, é produzir expressões estatísticas que permitam representar o comportamento do sistema [BAGCHI,1989].

Neste trabalho optou-se por realizar um simulador que seja fiel representação do "sistema original" apresentado no capítulo 4, de maneira que se possa realizar o estudo do comportamento da aplicação para um número maior de processadores, conectados em diferentes topologias.

A partir dos tempos obtidos da implementação do "sistema original", executando-se em vários processadores, foi modelado o simulador para que trabalhe em forma lógica, da mesma maneira que o "sistema original".

O simulador foi implementado numa placa IMS B004 com un transputer IMS T800, utilizando o ambiente de desenvolvimento para transputer TDS2 com a linguagem OCCAM2, para evitar a recodificação de módulos que foram utilizados em ambos sistemas.

5.2 MODELAMENTO DO SIMULADOR.

A construção do simulador implica em criar um sistema lógico, constituido por processo lógicos, que permita simular o "sistema original" (implementado no capítulo anterior).

O sistema lógico ou simulador deverá reproduzir a sequência exata das transmissões de mensagens entre os processos

concorrentes no processador e sua interação com outros processadores, que existam na malha de interconexão selecionada pelo usuário.

Para poder refletir a execução real, deverá existir uma estrutura de dados no simulador, que permita registrar o tempo e os dados que representam os processos que estão em execução. O tempo é controlado por um relógio local ou global de simulação. A estrutura de dados, denominada "lista de eventos", deverá conter os dados dos processos ativos, que permita o avanço da execução.

O algoritmo clássico de simulação por evento, opera do seguinte modo: da lista de eventos, extrai-se iterativamente os eventos de tempo menor, simulando-se seu efeito. Cada vez que é simulado um evento o relógio avança no tempo de processamento do evento. A figura 5.1 mostra o algoritmo principal de simulação de eventos, na forma sequencial [MISRA,1986].

```
Inicialize :
  clock ⟵ 0
  lista_eventos ⟵ <(t_i,m_i)/ a mensagem m_i será enviado em t_i>
Iterativo
  WHILE não termine DO
    BEGIN
        <remova processo de menor tempo (t,m) da lista_eventos>
        <Simule o efeito de transmitir m no tempo t>
        -- a simulação o evento pode criar novos eventos
        clock ⟵ t
    END
```

fig. 5.1. Algoritmo típico de simulação sequencial.

5.2.1 DEFINIÇÃO DOS PROCESSOS LÓGICOS.

Na seção 4.4, foi modelado o problema a ser simulado e foram estabelecidos os processos que existem em cada processador. Como forma de facilitar a utilização dos Diagramas de Petri, apresentados nas figuras 4.7 e 4.10, foram empregados identificadores para os blocos que realizam as mesmas operações nos processadores mestre e escravos.

Dessa maneira, podem ser descritos os processos lógicos, considerando a alocação dos processos feita na seção 4.4, para os processadores mestre e escravos. Isso é feito do modo seguinte:

a) Processador mestre :

- Sd : representa o algoritmo de distribuição das tarefas aos processadores alocados (escalonador);
- Sa : representa o algoritmo de comunicação;
- Sb : representa o algoritmo que recebe os resultados dos processadores liberados, carregando esses na fila de processadores disponíveis (I);
- Sce_i : representa as interfaces de entrada dos links conectados ao processador;
- Scs_i : representa as interfaces de saídas dos links do processador.

b) Processadores escravos :

- Sa : representa o algoritmo de comunicação;
- Sb : representa o algoritmo de retardo do processo (processo simulado);
- Sce_i : representa as interfaces dos links de entrada ao processador escravo;
- Scs_i : representa as interfaces dos links de saída do processador escravo.

Os processos lógicos Sa, Sb, Sce_i e Scs_i serão referenciados neste trabalho como Sa[i], Sb[i], Sce[i][j] e Scs[i][j] sendo i o número do processador e j o numero do canal de entrada ou saída correspondente. O processador mestre será referenciado quando i for igual a zero e os processadores escravos quando i for maior que zero.

5.2.2 ESTRUTURA DE DADOS DO SIMULADOR.

No simulador o avanço do processamento necessita de dados que permitam identificar o processo lógico que está ativo, a ação que deve ser realizada quando for simulada sua execução e, o

processo lógico que será disparado.

Foi definido o seguinte registro de dados para cada processo lógico dos processadores que participam na simulação.

Registro do evento (proc, vf, emissor, dest, viz, canal, t.proc, F)

Onde :

proc - identificação do processador ao qual pertence o processo lógico;

vf - identificação do processo lógico ativo.

emissor - identificação do processador emissor da mensagem;

dest - identificação do processador destino da mensagem;

viz - identificação do processador vizinho ao processador intermediário que processa a mensagem;

canal - identificação do canal utilizado na entrada ou saída da mensagem;

t.proc - tempo de finalização da execução do processo.

F - tempo de retardo.

Estes dados referentes ao registro do evento são transferidos ao novo processo lógico ativado, diferenciando-se do processo lógico antigo pelo tempo e pela identificação do processo.

Além da estrutura de dados, é necessário para cada processo lógico existente no simulador, associar uma variável de estado, que permita indicar se o mesmo está em execução ou suspenso, quando estão em operação as ações do sistema estabelecidas na seção 4.4.

5.3 PLANEJAMENTO DO SIMULADOR.

A estratégia do projeto do Simulador baseia-se nos "estados" envolvidos nos diferentes processos lógicos de cada processador. A execução é iniciada pela distribuição de tarefas num processador disponível. Para a tarefa chegar ao processador atribuído, a mensagem passa por processadores intermediários, que a transferem ao processador destino. Nos processadores intermediários, que participam no roteamento da mensagem, são gerados eventos, como se pode verificar na figura 5.2. Quando termina a execução da tarefa atribuida ao processador, ele envia uma mensagem de retorno ao processador mestre, para a sua liberação e armazenamento dos resultados no processador mestre.

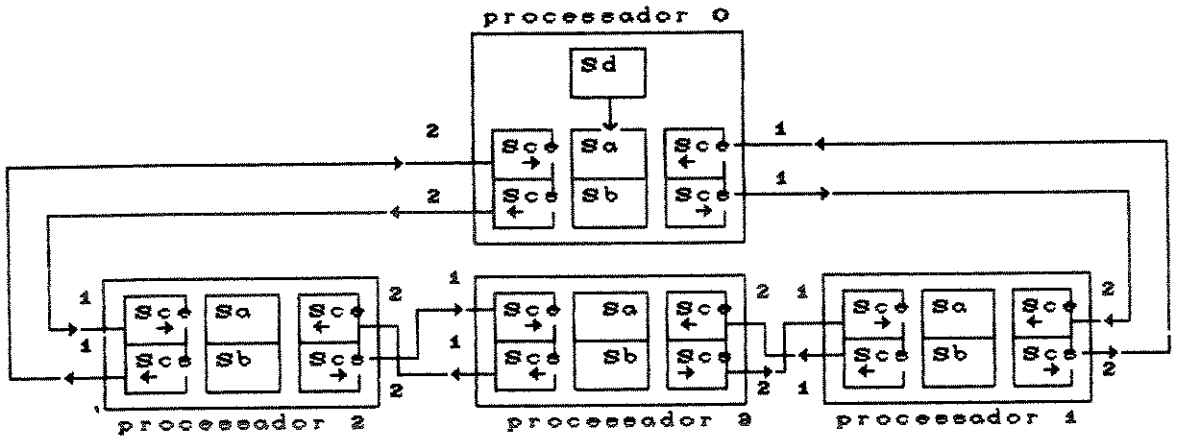


fig. 5.2. Diagrama exemplo.

No esquema da figura 5.2 (a nomenclatura utilizada foi definida na seção 5.2.1), os processadores estão conectados em forma cúbica, e existem três tarefas a processar, para serem distribuídas aos processadores 1, 2 e 3 respectivamente.

Quando o processo Sd do processador 0 distribui a primeira tarefa, é enviada uma mensagem ao processo Sa[0] para a determinação do destino. Como essa é a primeira tarefa, o processador selecionado é o processador 1. Assim, Sa[0] transfere a mensagem ao canal de saída conectado ao processador 1 (Scs[0][1]) e Sd pode distribuir a segunda tarefa.

Simultaneamente, os processos Sd e Scs[0][1] podem transferir as mensagens Sd a Sa[0] e Scs[0][1] ao canal de entrada do processador 1 (Sce[1][2]). O canal de entrada Sce[1][2] transfere a mensagem a Sa[1], para sua análise. Como a mensagem é para o processador 1, Sa[1] transfere a mensagem a Sb[1], iniciando a execução da tarefa.

Paralelamente, a mensagem da tarefa 2 é dirigida na direção do processador 2. Logo, a tarefa 3 fará o percurso através do processador 1, em direção ao processador 3. Se o processo Sb[1] estiver em execução, quando passa a mensagem da tarefa 3, produz-se uma interrupção da execução, até que Sa[1] fique em estado livre.

Ao terminar a execução das tarefas, esse fato é comunicado pelos processadores ao processador mestre, enviando uma mensagem com a seguinte sequência : Sb → Sa, Sa → Scs, Scs → Sce, Sce → Sa, ..., até chegar a destino, Sa[0] → Sb[0].

Dessa maneira, pode-se notar que cada sequência realizada pela passagem da mensagem são eventos que deverão ser tratados no tempo.

Como foi mencionado na seção 5.2, o simulador deve escolher dentre os processos lógicos ativos (eventos), aqueles que levam menor tempo para realizar a simulação. Considerando o aspecto de operação real, descrito no parágrafo anterior, foi implementado o simulador, cujo diagrama é apresentado na figura 5.3.

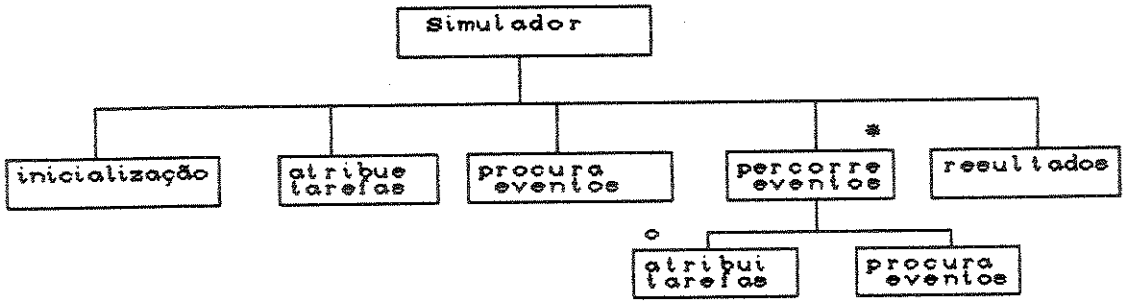


fig. 5.3. Diagrama geral do simulador.

Descrição dos módulos :

a) Bloco **"inicialização"** : Realiza o ingresso das tarefas e dos dados iniciais do Simulador, como: tipo de topologia, tempo de execução de cada tarefa, número de processadores diponíveis, etc;

b) Bloco **"atribui tarefas ao processador"**: Tem como função escalonar processos, dada a fila de tarefas e a fila de processadores disponíveis, atribuíndo uma tarefa a cada processador, computando o tempo de distribuição e marcando o processo Sd em estado ativo;

c) Bloco **"procura eventos a seguir"**: Realiza a busca de todos os processos lógicos "ativos" de cada processador (Sd, Sa, Sb, Sce e Scs), considerando apenas os de menor tempo. Deve-se assegurar a forma de selecionar o(s) evento(s) de tempo menor, do conjunto de processos lógicos em estado de execução, para que não exista a possibilidade de um evento de tempo maior ser selecionado.

d) Bloco **"percorre a fila eventos"**: Realiza simulação da execução dos processos ativos selecionados no bloco **procura eventos a seguir**.

e) Bloco **"resultados"**: forneçe, após a finalização da execução do simulador, os tempos e parâmetros de interesse, como : tempo de execução, tempo médio de comunicação, tempo médio de contenção, tempo médio de interrupção e os parâmetros de desempenho como "speedup" e eficiência.

A figura 5.4 mostra o texto estruturado de controle geral do simulador.

```
BEGIN
   << inicializar a fila de processadores disponiveis (I)>>>
   << inicializar a fila de processos a simular (TS) >>
   << atribui tarefa a um processador >>
   << procura tarefas a seguir >>
   WHILE ( ∃ processos ) do
     BEGIN
       << percorre a fila de eventos >>
       IF Sd <> 1 AND Sa[0] <> 1
           << atribue tarefa a um processador >>
       END-IF
       << procura tarefas a seguir >>
     END
   END-WHILE
   << mostrar resultados >>
END.
```

fig. 5.4. Texto estruturado geral do Simulador de topologias.

5.4 DESCRIÇÃO DOS MÓDULOS DO SIMULADOR.

5.4.1 ATRIBUI TAREFAS.

Esse módulo realiza o escalonamento de tarefas. Inicialmente, é procurado um processador disponível e uma tarefa na fila de processos. Se existe um processador disponível e uma tarefa, então é realizado o escalonamento. Isso é feito ativando-se o processo lógico de distribuição (Sd), no tempo do "clock" do processador mestre mais o tempo de distribuição (t.dist).

A figura 5.5 mostra o texto estruturado desse módulo.

```
BEGIN
  < procura um processador disponivel >
  IF ∃ processador disponivel THEN
        BEGIN
          < procura uma tarefa >
          IF ∃ tarefa THEN
                < escalona o processo >
             ELSE
                < libera processador disponivel >
          END-IF
        END
  END-IF
END.
```

fig. 5.5. Texto estruturado do módulo "distribui tarefas".

5.4.2 PROCURA EVENTOS A SEGUIR.

Esse bloco é a essência do simulador, porque deve selecionar os processos ativos de tempo menor, criando uma fila de eventos, resultantes da operação em paralelo, a serem simulados num dado instante.

A fila de eventos resultantes é ordenada segundo a prioridade de operação dos processos lógicos, descritos na seção 4.4. Para tal efeito, os processos lógicos foram codificados da seguinte maneira : $Sb_i=1$, $Sa_i=2$, $Sce_{ij}=3$, $Scs_{ij}=4$ e $Sd=5$.

A figura 5.6 mostra o texto estruturado do bloco "procura eventos a seguir".

```
<< procura tarefas a seguir >>
BEGIN
  < carrega Sd = i >
  WHILE (i<total.nos) DO
    BEGIN
      < carrega os Sa[i] = i >
      < carrega os Sb[i] = i >
      FOR j = 0 TO max.link DO
        BEGIN
          < carrega os Sce[i][j] = i >
          < carrega os Scs[i][j] = i >
        END
      i = i + 1
    END
  < seleciona os eventos de tempo menor >
END.
```

fig. 5.6. Texto estruturado do módulo "procura eventos a seguir".

No tratamento do processo Sa[i], que representa ao módulo de comunicação, realiza-se a atribuição do tempo, que é o tempo do processador i mais o "tempo na determinação do destino", resultante da execução do módulo "topologias". Esse módulo utiliza os dados de identificação do processador fonte e destino, entregando a identificação do processador vizinho , tempo e canal de saída a utilizar. A figura 5.7 mostra o texto estruturado do bloco topologias. Os detalhes dos algoritmos de cada topologia são apresentados no anexo A.

```
<< Topologias >>
BEGIN
  IF
    tipo = 1
      anel(fonte, destino, v1, i1, c1)
    tipo = 2
      árvore(fonte, destino, v1, i1, c1)
    tipo = 3
      n.cubo(fonte, destino, v1, i1, c1)
    tipo = 4
      malha.eqr(fonte, destino, v1, i1, c1)
    tipo = 5
      malha.midimev(fonte, destino, v1, i1, c1)
  END-IF
  vizinho ← v1
  consumo ← i1
  canal   ← c1
END.
```

fig. 5.7. Texto estruturado das topologias consideradas.

Na simulação do evento Sa, o processador vizinho é o processador que receberá a mensagem na sequência de execução.

5.4.3 PERCORRE A FILA DE EVENTOS.

É o módulo que realiza a simulação dos eventos fornecidos pelo módulo "procura eventos a seguir", podendo mudar os estados dos processos lógicos que foram considerados na fila de eventos.

A figura 5.8 mostra o texto estruturado do bloco "percorre fila eventos". Considere que "vf_i" representa a identifição do evento, "$fila_i$" o processador, e "topo.fila" o número máximo de eventos a analisar.

```
<< percorre a fila de eventos >>
BEGIN
  ti ⟵ topo.fila - 1
  WHILE ( ti > (-1) ) do
    Begin
      proc ⟵ fila(ti)
      analisa.eventos(proc, vf(ti))
      ti ⟵ ti -1
    END
  END-WHILE
END.

<< analisa.eventos >>
BEGIN
  IF
    vf(ti) = 1
      termino.retardo.ou.liberação(proc)
    vf(ti) = 2
      analisa.destino(proc)
    vf(ti) = 3
      canal.entrada(proc)
    vf(ti) = 4
      canal.saida(proc)
    vf(ti) = 5
      inicio.distribuição()
  END-IF
END.
```

fig. 5.8. Texto estruturado de "percorre a fila de eventos".

Cada bloco do algoritmo "análise _eventos", deve cumprir as regras de processos estabelecidas na seção 4.4. Caso contrário será acresido um tempo de contenção e postergada sua simulação.

A seguir é apresentada uma descrição dos módulos do bloco "percorre fila de eventos".

a) **Início_distribuição.** é o módulo que realiza a simulação do processo Sd, quando este está ativo. Se o processo lógico do processador mestre Sa[0] estiver ocupado, o processo (Sd) é retardado;

b) **Analisa_destino.** é o módulo que realiza a simulação do processo Sa. Dependendo dos dados fornecidos pela determinação do destino da mensagem (módulo "topologia"), dirige a mensagem a três possíveis processos a serem ativados:

   1) Se o dado "processador vizinho" difere da

identificação do processador do processo Sa, executa-se a ação "vai ao canal de saída".

2) Se o dado "processador vizinho" é igual à identificação do processador do processo Sa, mas difere da identificação do processador mestre, então executa-se a ação "inicio do processo simulado".

3) Se o dado "processador vizinho" é igual à identificação do processador do processo Sa e à identificação do processador mestre, significa que deve-se executar a ação "chegou um resultado".

Nesse módulo, deve-se considerar o tempo de retardo por contenção, caso o canal de saída esteja ocupado e o processo Sa necessite executar a ação "vai ao canal de saída".

Também, se o processo de "retardo" Sb[i] está em estado ativo, deve-se registrar o tempo de interrupção do processo simulado, até que o processo Sa[i] fique liberado;

c) **Término_retardo_ou_liberação.** é o módulo que realiza a simulação do processo Sb, executando as ações de "liberação do processador" e "término do processo simulado". Quando o processo lógico Sb[0] executa a ação "recebe resultados", a fila de processadores é atualizada, indicando a disponibilidade do processador liberado. Se o processo Sb executa a ação "término processo simulado", significa que o processo que simula a execução da tarefa num processador escravo, deve comunicar através de uma mensagem ao processador mestre, os dados resultantes e a liberação do processador. A mensagem é enviada ao processo Sa para roteá-la ao processador mestre. Se o processo Sa está ativo, então é registrado um tempo de contenção, é atualizado o tempo do processo Sb ao tempo maior dos processos lógicos envolvidos, e a simulação postergada.

d) **Canal_entrada.** é o módulo que realiza a simulação do

processo Sce, executando a ação "chegou uma mensagem na entrada i" e enviando a mensagem para análise ao processo Sa. Se o processo Sa estiver ativo, é registrado um tempo por contenção, e atualizado o tempo de Sce com o maior tempo dos processos lógicos envolvidos, e sua simulação é postergada;

e) **Canal_saída.** é o módulo que realiza a simulação do processo Scs, executando a ação "transfere a mensagem a outro processador", quando o processo Sce do processador destino está liberado. Caso contrário, é registrado um tempo por contenção, é atualizado o tempo do processo Scs com o maior tempo dos processos lógicos envolvidos e a simulação é postergada.

5.5 MANIPULAÇÃO DOS TEMPOS DO SIMULADOR.

Pelo princípio de operação do Simulador de Eventos, as mensagens são enviadas no instante em que é finalizada a execução do processo lógico e é iniciada a execução do seguinte. Dessa maneira, os cinco tipos de processos considerados, gerados pelos processadores, terão os tempos de simulação atribuídos da seguinte forma :

a) **Início distribuição.** Quando é considerada a simulação do processo Sd, o tempo do processo é o tempo do relógio do processador mestre mais o tempo de distribuição. Se o processo Sa está liberado, então Sa é ativado com um tempo igual ao tempo do processo Sd mais o **tempo na determinação do destino**.

b) **Análise destino.** Quando o processo Sa é simulado, podem ocorrer três possibilidades:

   1. Se a mensagem vai ao canal de saída e este está liberado, o tempo do processo Scs gerado será o tempo de Sa mais o **tempo de transferência externa**;
   2. Se a mensagem vai para o processo simulado e o processo Sb está liberado, o tempo do processo Sb gerado será o tempo de Sa mais o tempo da

tarefa atribuida (**tempo de retardo**);

3. Se a mensagem é a chegada de um resultado ao processador mestre, o tempo do processo Sb[0] será o tempo de Sa mais o **tempo de liberação**;

c) **Canal de entrada**. Quando o processo Sce é simulado e o processo Sa está liberado, o tempo do processo Sa será o tempo de Sce mais o **tempo na determinação do destino**.

Quando chega uma mensagem ao canal de entrada e o relógio local do processador é de menor tempo do que o tempo de chegada da mensagem, o relógio é atualizado com o tempo maior.

d) **Canal de saída**. Quando o processo Scs é simulado e o processo Sce do processador destino está liberado, o tempo do Sce do processador destino será o tempo de Scs mais o **tempo de transferência interna**.

O comportamento de todos os tempos mencionados nesta seção foram apresentados na implementação do "sistema original" na seção 4.6 e 4.7.

5.6 DADOS DE ENTRADA NO SIMULADOR.

O simulador recebe as tarefas a processar através de um arquivo. O tempo de execução de cada tarefa do arquivo é expressa em unidades de tempo. Esse arquivo é gerado por um programa que oferece a possibilidade de ingressar dado a dado, por bloco de tarefas còm tempos idênticos ou ambas.

Interativamente o simulador solicita os seguintes dados de entrada :

- Tamanho máximo da mensagem;
- Tempo do tick expresso em micro-segundos (μs);
- Tipo de topologia a considerar;
- Número de processadores disponíveis.

5.7 DADOS DE SAÍDA DO SIMULADOR.

Após da execução do simulador são fornecidos os seguintes resultados :

- Tempo de execução da aplicação simulada;
- Ganho ou "speedup" da aplicação;
- Eficiência;
- Valor médio do custo de comunicação;
- Valor médio do tempo de interrupção;
- Valor médio do tempo de retardo por contenção.

5.8 SINTONIZAÇÃO DO SIMULADOR.

Nesta seção mostra-se o resultado obtido da implementação do Simulador, comparado com os tempos medidos no "Sistema original", apresentado na seção 4.6 e 4.7. Pretende-se mostrar o grau de confiabilidade da operação do Simulador. Para realizar essa operação os retardos no "sistema original" foram ajustados a 30 ticks de 64 μs e o tamanho da mensagem a 200 bytes. O número de tarefas é igual ao número de processadores disponíveis na topologia.

Os valores obtidos na execução da implementação na placa Inmos IMS B008 são apresentados na tabela 5.1. A medida de maior interesse utilizada para os diferentes ajustes foi o tempo de execução. As medidas restantes são utilizadas para o estudo do comportamento da aplicação, discutidas no capítulo seguinte.

t.ex \ P	1	2	3	4	5	6	7	8
cubo	4.48	5.44	7.04	7.04	8.448	9.152	10.88	
malha	3.712	5.44	5.44	7.72	8.064	8.064	9.664	11.264
árvore	3.648	4.288	5.76	6.27	6.464	7.104	8.256	9.472
midimev	4.928	5.76	7.616	8.448	8.704	9.216	9.216	

Tabela 5.1 . Tempos de execução das topologias com 1..8 tarefas.
t.ex : em milisegundos (10^{-3}s.)

A tabela 5.2 apresenta os tempos de execução obtidos no Simulador. A tabela 5.3 mostra os erros resultantes da comparação dos tempos de execução das tabelas 5.1 e 5.2.

t.ex \ P	1	2	3	4	5	6	7	8
cubo	4.515	5.239	7.247	7.314	9.696	10.585	10.6	
malha	3.698	5.448	5.448	6.212	7.842	7.842	9.139	10.241
árvore	3.76	4.162	5.989	6.999	6.786	7.291	8.611	9.579
midimev	4.55	5.270	7.319	9.099	8.77	9.402	10.084	

Tabela 5.2. Tempos de execução do Simulador.
t.ex : em milisegundos (10^{-3} s.).

erro \ P	1	2	3	4	5	6	7	8
cubo	-0.78	-3.8	2.9	3.8	2.9	15.65	-2.6	
malha	0.5	0.055	0.055	-8.2	-2.8	-2.8	-5.6	-9.8
árvore	3.07	-3.027	3.09	1.7	4.9	1.8	-4.3	1.18
midimev	-8.3	-8.7	-4.05	-5.08	0.76	2.02	8.8	

Tabela 5.3. Erros resultantes das medições.

$$erro = \frac{t.ex1 - t.ex2}{min(t.ex1, t.ex2)} * 100$$

Existem diversos fatores que dificultam a sintonização do Simulador. Esses erros são causados por:

a) As medições feita na placa multi-transputer, às vezes, diferem em 100 ou mais μs, mesmo sem variar nenhuma condição. Esse fato é mais notório na medição da transferência da mensagem entre processadores;

b) Também deve ser considerado o erro de arrasto resultante da medida do tempo em alta e baixa prioridade. Os ticks de relógio em baixa prioridade de operação são incrementado a cada 64 μs; os ticks do relógio de alta prioridade cada 1 μs;

c) Para poder realizar as medidas dos blocos, a operação do módulo de interesse teve que ser alocado em alta prioridade. Esse fato produz um erro adicional na entrada e saída de execução, se o processo opera em baixa prioridade;

d) Por último, existem erros introduzidos pela utilização da instrução seletiva, no caso da linguagem OCCAM2. A instrução ALT não opera da forma FIFO (primeira mensagem a entrar, primeira a sair), mas por prioridade alocada na escrita da instrução de entrada. Esse fato é mencionado por diversos autores.

Veja Törngrem [TöRN, 1990], Shallow [SHALL,1990] e Jones [JONES,1989].

O Simulador é incapaz de considerar o fato mencionado no item (d), porque quebra a regra imposta para evitar o erro casual, que considera a execução dos eventos de menor tempo.

CAPÍTULO 6

ANÁLISE DE RESULTADOS

6.1 INTRODUÇÃO.

Neste capítulo são apresentados os resultados obtidos a partir das simulações realizadas, considerando os tempos de execução dos módulos do algoritmo L.S.E. definido em Regonessi [REGO,1990].

É realizada também uma comparação entre os resultados ideais do estudo teórico e os resultados reais considerando os tempos de comunicação obtidos através do "Simulador" desenvolvido neste trabalho.

As medidas de interesse apresentadas neste capítulo, são: "speedup", eficiência, tempo de execução, tempo gasto em comunicação, tempo gasto em contenção e tempo de interrupção. Dessas medidas, obtidas para diferentes número de processadores, podem-se extrair algumas conclusões, que dependerão das exigências impostas para a aplicação. Nesse caso, optou-se por mostrar o comportamento geral da aplicação, com ênfase no melhor "speedup", obtido para cada topologia.

6.2 FONTE DOS DADOS CONSIDERADOS.

Como foi mencionado anteriormente na seção 2.1, da implementação sequencial do algoritmo "Linhas e Superfícies Escondidas", foram determinados os tempos de execução dos módulos de remoção de faces visíveis, decomposição em triângulos e comparação de triângulos. O estudo foi baseado numa figura padrão, correpondente a uma tampa de chaleira, com 92 faces, considerando uma projeção paralela de topo, das quais 70 faces são visíveis.

Os dados obtidos nas medidas realizadas na execução sequencial, são apresentados nas seções seguintes, como parte do

estudo em condições ideais da execução paralela.

6.3 MEDIDAS NO MÓDULO DE REMOÇÃO DE FACES VISíVEIS.

6.3.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Do estudo feito sobre o comportamento do módulo de remoção de faces visíveis na implementação sequencial, foram extraídos os dados apresentados na tabela 6.1. Dados que foram utilizados para realizar a medida de rapidez ou ganho ideal para essa fase. Considerando um número ilimitado de processadores, o tempo mais longo corresponderá ao tempo de execução paralela (9 tick, onde cada tick a 1.715 ms) e, como tempo de execução sequencial, a soma dos tempos de todas as tarefas, obtem-se o seguinte "speedup" (com base na eq. 3.1):

$$S(n) = \frac{300}{9} = 33.3 \qquad (6.1)$$

Iterações	Ticks
78	3
12	4
2	9

Tabela 6.1. Distribuição dos dados do processo de Remoção de faces visíveis.

6.3.2 MEDIDAS NO SIMULADOR.

Foram consideradas as tarefas apresentadas na tabela 6.1, executadas nas diferentes topologias que o simulador oferece. Os resultados obtidos são apresentados na tabela 6.2.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nºprocessadores	60	30	31	48
T. execução (s)	0.0490024	0.049107	0.0088	0.04059
"Speedup"	10.49948	11.99599	7.4781608	12.09886
Eficiência	0.1544	0.397844	0.24129	0.2644556
T. comunica. (s)	1.058128	0.211844	0.12244	0.32217
T. interrup. (s)	0.19275	0.0360098	0.0109125	0.05566
T. contenção (s)	0.0178071	0.0042509	0.0066792	0.007457

Tabela 6.2. Distribuição dos dados ótimos, da execução do processo de remoção.

Do estudo do comportamento do processo de remoção de faces visíveis, considerando as topologias mais comuns pode-se mencionar o seguinte:

a) A topologia que apresentou o melhor tempo de execução foi a topologia malha, com um "speedup" correspondente ao 35% do "speedup" ideal.
b) A topologia árvore, em termos de eficiência, foi a que apresentou o melhor comportamento.
c) A topologia cúbica produziu os menores tempos de comunicação e interrupção dos processos, fornecendo um "speedup" razoável, considerando poucos processadores.

A figura 6.1 apresenta os gráficos do comportamento da rotina de remoção de faces visíveis. Nesses gráficos são apresentados o "speedup" e a eficiência para diferentes números de processadores.

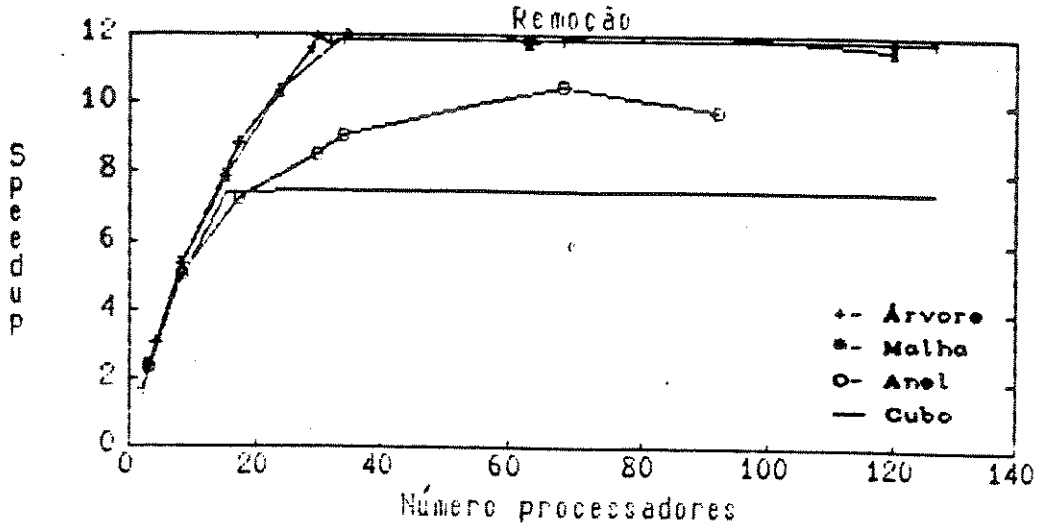


fig. 6.1a. "Speedup" do processo de remoção.

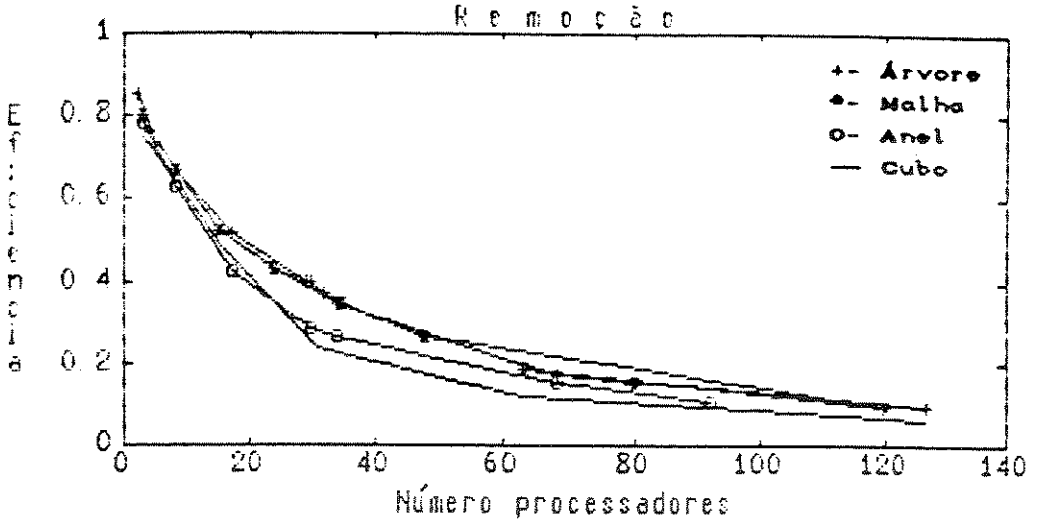


fig. 6.1b. Eficiência do processo de processo de remoção.

No gráfico da figura 6.1a, observa-se que o "Speedup" máximo é obtido com poucos processadores nas topologias malha e árvore. Observa-se que o "speedup" mantem-se constante, se o número de processadores aumenta. A topologia anel, por sua composição, oferece o melhor "speedup" para um número maior de processadores do que as outras topologias. A topologia cúbica mostra um comportamento mais regular, porque, uma vez alcançado o melhor "speedup", o aumento do número de processadores não modificará seu comportamento. Isso deve-se à estratégia de

distribuição das tarefas feitas pelo processo de escalonamento, em forma dinâmica.

A figura 6.1b mostra a utilização média dos processadores. Pode observar-se que a melhor curva de eficiência é apresentada pelas topologias árvore e malha, mas as curvas tendem a ser muito similares.

6.4 MEDIDAS DO MÓDULO DE DECOMPOSIÇÃO EM TRIÂNGULOS.

6.4.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Para a medição dos tempos relativos as tarefas no módulo de "decomposição das faces visíveis em triângulo", foram consideradas as faces visíveis resultantes do processo de remoção de faces. Os dados obtidos na decomposição dessas faces visíveis são apresentados na tabela 6.3.

Iterações	Ticks
25	5
49	6
2	8

Tabela 6.3. Distribuição dos dados do processo de Decomposição das faces visíveis.

O cálculo do "speedup", para esse módulo, é realizado da mesma forma que o cálculo do "speedup" ideal para a remoção de faces na seção 6.2. O valor obtido nesse caso é o seguinte :

$$S(n) = \frac{399}{8} = 49.87 \qquad (6.2)$$

6.4.2 MEDIDAS DO SIMULADOR.

De maneira similar à seção 6.3.2, foram consideradas as medidas dos tempos que são apresentadas na tabela 6.3, e as tarefas foram distribuídas aos processadores, nas diversas topologias permitidas pelo simulador. Os resultados da simulação

são mostrados na tabela 6.4.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nº processadores	69	48	31	69
T. execução (s)	0.04278	0.0352212	0.05198	0.0341194
"Speedup"	15.99516	19.4281722	13.169922	20.04097
Eficiência	0.2598	0.4047585	0.4246	0.31811
T. comunica. (s)	0.80778	0.1802187	0.10969	0.295192
T. interrup. (s)	0.087002	0.09492	0.009648	0.04988
T. contenção (s)	0.00267	0.002788	0.004499	0.0057948

Tabela 6.4. Distribuição dos valores ótimos da execução do processo de Decomposição das faces visíveis.

Observa-se na tabela 6.4, por um lado, que o melhor tempo de execução e, consequentemente, o melhor "speedup", é obtido com a topologia malha; porém com um número de processadores maior. Por outro lado, a topologia árvore oferece um "speedup" próximo ao melhor apresentado pela topologia malha. Outro aspecto interessante, é que as tarefas consideradas possuiam maior tempo de execução e o "speedup" obtido na simulação corresponde ao 40% do "speedup" apresentado no estudo teórico.

A figura 6.2 apresenta os gráficos das medidas obtidas para o estudo de comportamento desse módulo no simulador.

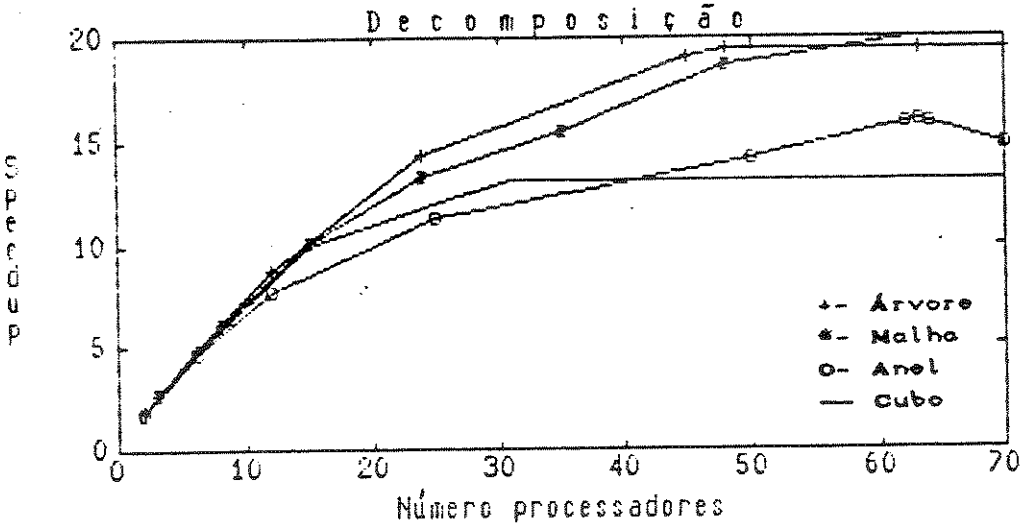


fig. 6.2a. "Speedup" do processo de Decomposição em triângulos

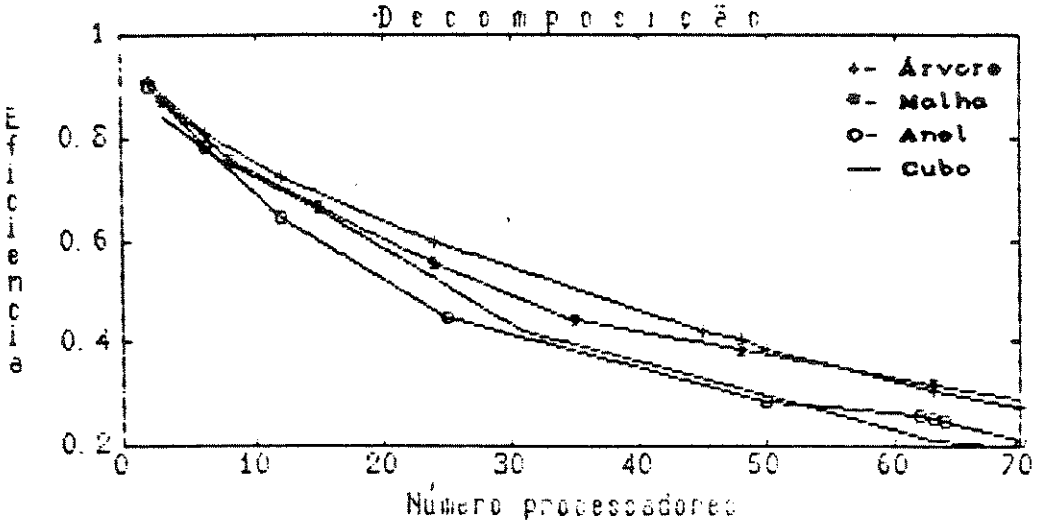


fig. 6.2b. Eficiência do processo de decomposição em triângulos.

No gráfico da figura 6.2a, novamente as topologias árvore e malha apresentam o melhor "speedup", com um aumento gradativo até chegar acima dos 48 processadores e, depois, mantendo um comportamento constante, no caso da árvore, e decrecente no caso da malha. O mesmo fato acontece com a topologia cúbica e anel pela forma de conexão.

Do gráfico da figura 6.2b, pode-se observar um aumento da eficiência, resultante da consideração de tarefas com tempo de execução maior do que na distribuição das tarefas de remoção de faces. Dos processo analisados, a topologia cúbica chega mais rapidamente ao melhor "speedup" e a eficiência tem uma queda mais rápida do que as outras topologias.

6.5 MEDIDAS DO MÓDULO DE COMPARAÇÃO DE TRIÂNGULOS.

6.5.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Considerando os dados coletados utilizando-se a tampa da chaleira, cada tick de relógio é de 1.715 ms. Apresenta-se na tabela 6.5 a distribuição dos dados do processo de comparação de triângulos.

Iterações	Ticks	Iterações	Ticks	Iterações	Ticks
4079	1	49	9	6	20
1554	2	19	11	13	21
17	4	10	12	20	22
34	5	11	13	12	23
13	6	34	14	4	24
59	7	11	16	4	26
37	8	4	17	6	33

Tabela 6.5. Distribuição dos dados do processo de Comparação de triângulos.

Pela análise dos dados verifica-se que o tempo total de execução sequencial corresponde a 11340 ticks e o tempo de maior caminho corresponde a 33 ticks. Portanto o "speedup" ideal dessa fase é:

$$S(n) = \frac{11340}{33} = 343.63636 \qquad (6.3)$$

Desse modo, 344 processadores executariam essa tarefa num tempo de 33 ticks (56.6ms).

6.5.2 MEDIDAS DO SIMULADOR.

Foi realizada a simulação da distribuição das tarefas do processo de comparação de triângulos, considerando-se o particionamento de tarefas a nível de comparação de pares de triângulos. A resposta da execução foi muito ruim, obtendo-se um "speedup" real de 8.0126 vezes, com uma malha de 15 processadores. A tabela 6.6 mostra o melhor comportamento da distribuição das tarefas desse módulo.

Topologias / Medidas	Anel	Malha	Hipercubo	Árvore
Nºprocessadores	14	15	31	15
T. execução (s)	2.9972	2.4209	9.550211	2.477
"Speedup"	6.4885	8.012672	5.47783	7.8482
Eficiência	0.46346	0.53417	0.176704	0.56059
T. comunica. (s)	16.7416	10.241	6.95964	10.29897
T. interrup. (s)	2.505187	1.59606	0.43656	1.27494
T. contenção (s)	0.559879	0.3133857	0.699148	0.415192

Tabela 6.6. Distribuição dos valores ótimos da execução do processo de comparação de triângulos.

Da análise dos dados, pode afirmar-se que nessa fase, o particionamento de tarefas não é apropriado, devido à quantidade de tarefas de curta duração (veja tabela 6.5). A figura 6.3 mostra os gráficos correspondentes à distribuição de 5995 tarefas, para diferentes números de processadores.

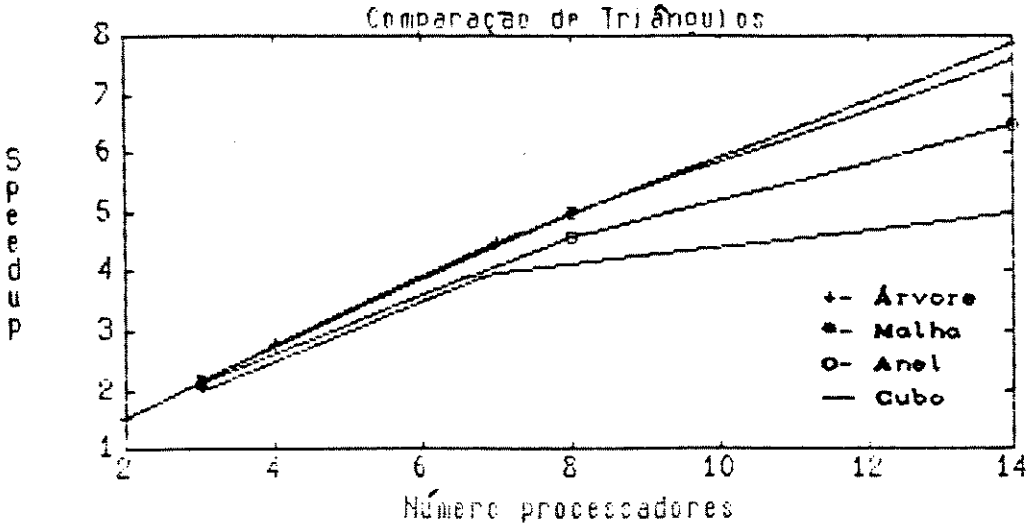


fig. 6.3a "Speedup" do processo de Comparação de triângulos.

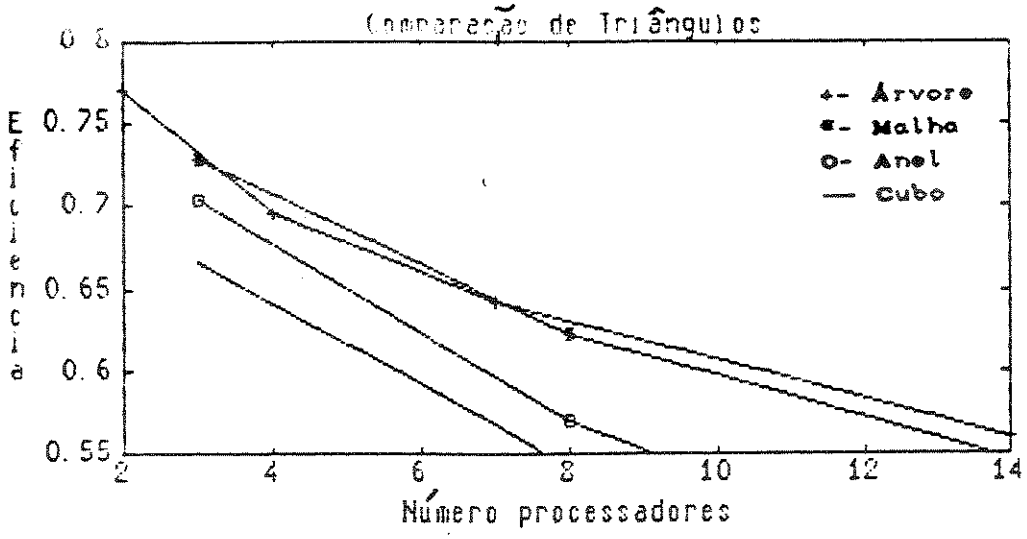


fig. 6.3b. Eficiência do processo de Comparação de triângulos.

Observa-se nos gráficos da figura 6.3, que a eficiência tem uma queda com apenas poucos processadores. Isso significa que o tempo das tarefas a serem distribuídas nos processadores deve

ser de maior duração, porque o tempo gasto em comunicação é crítico quando as tarefas são pequenas.

A fim de otimizar o tempo de execução do algoritmo, foram considerados particionamentos de granularidade mais grossa, agrupando de maneira aleatória as tarefas, num número fixo, a serem distribuídas nos processadores.

A tabela 6.7 mostra os melhores resultados obtidos na distribuição de blocos de tarefas (cada bloco é constituído de 10 processos de comparação de triângulos), executadas por um único processador (o teste foi feito para grupos de 5, 10 e 15 blocos tarefas).

Topologias / Medidas	Hipercubo	Árvore	Malha
Nºprocessadores	127	63	80
T. execução (s)	0.382785	0.380499	0.37686
"Speedup"	50.807	51.112548	51.606
Eficiência	0.1992	0.81121	0.645075
T. comunica. (s)	1.299	1.896756	2.54289
T. interrup. (s)	0.2148	0.491596	0.786621
T. contenção (s)	0.10828	0.098525	0.30406

Tabela 6.7. Distribuição dos valores ótimos do processo de comparação em lotes de 10 tarefas para cada processador.

Nota-se que a topologia em árvore possui um comportamento global melhor, com um número menor de processadores, porque apresenta uma eficiência melhor.

Da execução do conjunto de blocos de tarefas, onde cada bloco é constituído de 15 processos de comparação de triângulos por processador, observou-se, no entanto, que não se consegue um melhor "speedup". Esse fato indica que existe um número de tarefas que, agrupadas, apresentarão um melhor comportamento e que a consideração de um número maior de tarefas por bloco, empobrece a utilização do processamento para a aplicação.

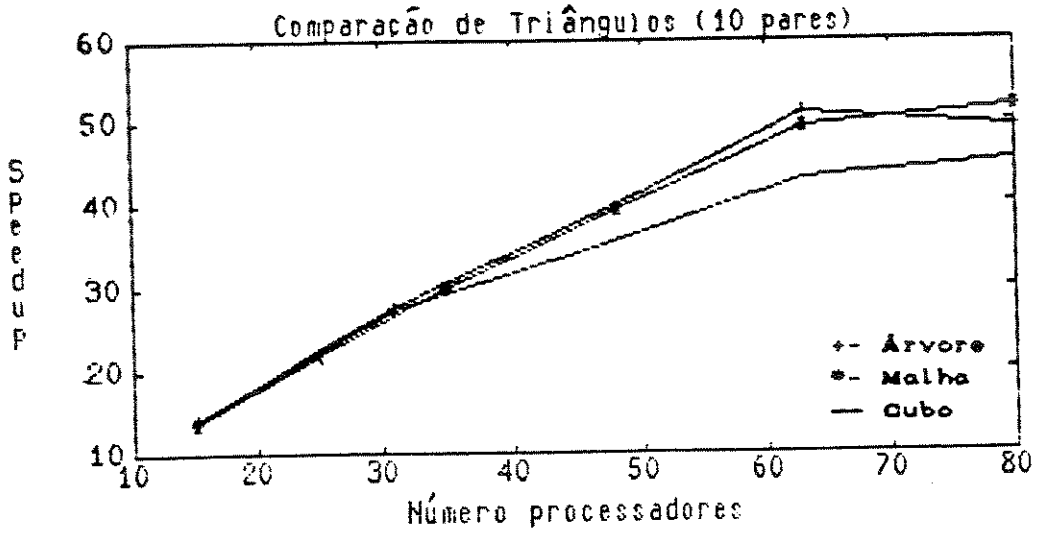


fig. 6.4a. "Speedup" do processo de comparação de triângulos em lotes de 10 tarefas.

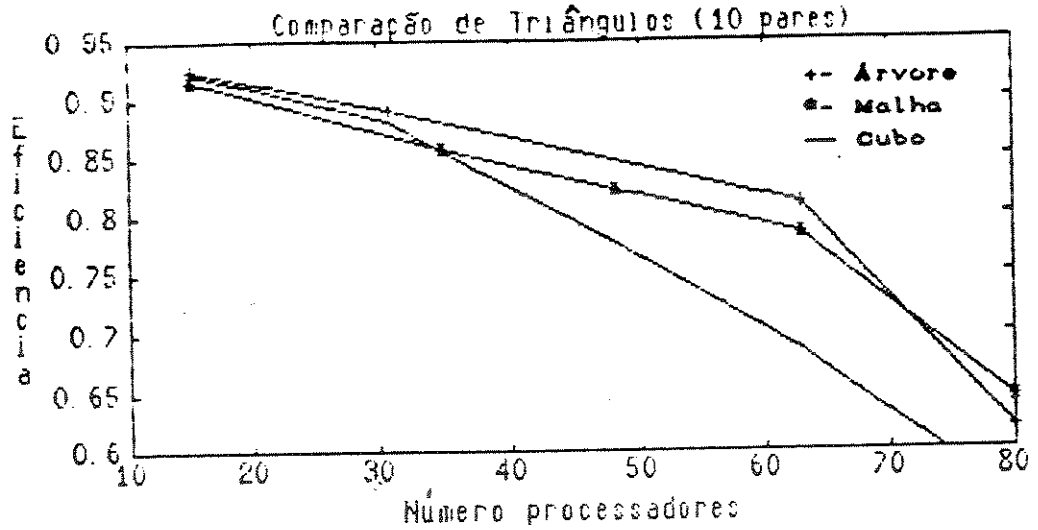


fig. 6.4b. Eficiência do processo de comparação de triângulos em lotes de 10 tarefas.

A figura 6.4 mostra os gráficos do "speedup" e a eficiência dos dados da simulação do processo de comparação de triangulos, considerando grupos de 10 tarefas. O melhor "speedup" é apresentado pela topologia malha, trabalhando com uma treliça de 9 filas e 9 colunas. A topologia árvore apresenta um comportamento aproximado com menos processadores e melhor eficiência, quando resulta o melhor "speedup". Se for aumentado o número de processadores, a eficiência tem uma queda significativa.

6.6 ESTUDO DE COMPORTAMENTO DA PROPOSTA PARA A IMPLEMENTAÇÃO PARALELA DO ALGORITMO "L.S.E".

Na seção 2.4 deste trabalho, foi apresentada uma nova proposta de implementação paralela. Essa proposta consiste em executar os processos de remoção e decomposição num mesmo processador, e iniciar o processo de comparação de triângulos apenas quando existir pelo menos um par de triângulos para serem comparados.

Nesta seção será apresentada a execução dos processos de remoção das faces visíveis e de decomposição em triângulos das faces visíveis e, posteriormente, o processo em forma integral, que foi denominado, na seção 2.4, "Bloco inicial" do algoritmo L.S.E.. Nesse caso o "bloco inicial" consistirá do processo de remoção-decomposição e do processo comparação de triângulos, em lotes de 10 tarefas a serem executadas por processador.

6.6.1 PROCESSO DE REMOÇÃO-DECOMPOSIÇÃO DAS FACES VISÍVEIS.

Considerando os mesmo dados extraídos da tampa da chaleira, para os processos de remoção e decomposição, respectivamente, foi realizada uma distribuição aleatória para combinar as tarefas de remoção e decomposição. Também, de forma aleatória, foi feita a eliminação das faces consideradas não visíveis.

A tabela 6.8 mostra os dados correspondentes à obtenção do melhor tempo de execução das topologias consideradas.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nº processadores	92	92	69	80
T. execução (s)	0.0620069	0.05687	0.08081	0.050995
"Speedup"	19.33305	21.0792	14.8331	21.055
Eficiência	0.2101419	0.2291227	0.2954	0.26919
T. comunica. (s)	1.66168	0.278821	0.15962	0.4875
T. interrup. (s)	0.199829	0.054084	0.01718	0.08149
T. contenção (s)	0.001485	0.0077209	0.047049	0.0104946

Tabela 6.8. Distribuição dos valores ótimos do processo de remoção-decomposição.

Analisando a tabela 6.8, verifica-se que o melhor comportamento é oferecido pela topologia malha ("mesh"), já que com um número menor de processadores, consegue um "speedup" próximo ao da topologia árvore.

A figura 6.5 mostra os gráficos obtidos nesse processo conjunto. Nota-se que as curvas mostram um comportamento muito similar ao processo de decomposição da figura 6.2. A diferença está no número de processadores necessários para obter maior "speedup", com uma eficiência menor.

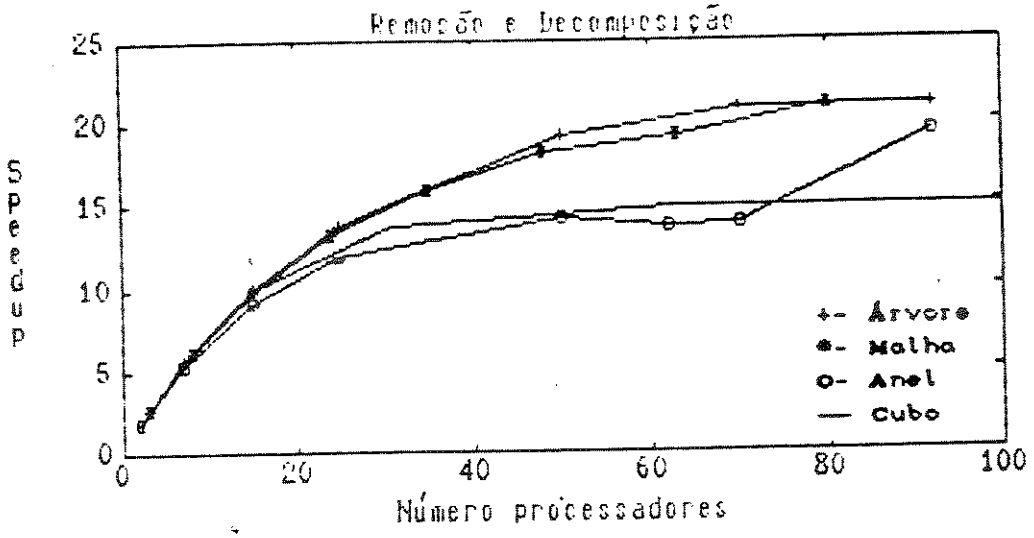


fig. 6.5a. "Speedup" do processo de remoção-decomposição.

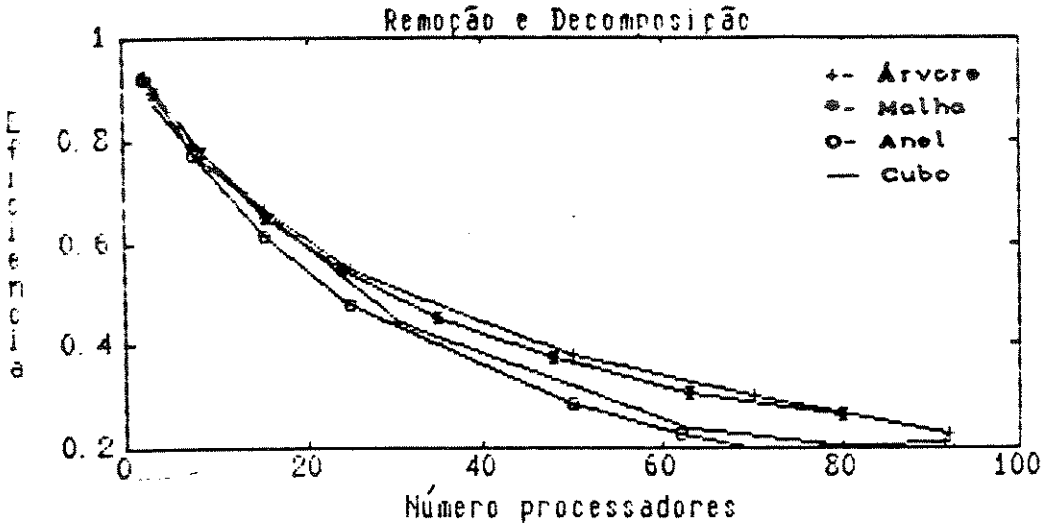


fig. 6.5b. Eficiência do processo de remoção-decomposição.

6.6.2 ESTUDO DO COMPORTAMENTO DO "BLOCO INICIAL".

Para realizar este estudo, foi criado um arquivo de processos que representa o particionamento das tarefas que compõem os módulos do "bloco inicial". Esse arquivo possui, inicialmente, as tarefas do processo de "remoção-decomposição", cujo comportamento foi apresentado na seção 6.5.1, e o particionamento do módulo de comparação de triângulos, onde são considerados grupos de 10 tarefas de comparação a serem distribuídas em cada processador disponível.

Neste estudo foram consideradas as topologias que apresentaram melhor desempenho, segundo o estudo feito nas seções anteriores deste capítulo.

Na obtenção dos dados, foi realizada uma varredura de comportamento nas topologias, considerando de 256 a 2 processadores interconectados. As topologias empregadas foram a Hipercúbica, Árvore e Malha ("mesh"). Os dados que mostram o melhor comportamento para cada topologia são apresentados na tabela 6.9.

Topologias / Medidas	Malha	Árvore	Hipercubo
Nº processadores	99	81	255
T. execução (s)	0.9769	0.977605	0.438618
"Speedup"	54.8685	54.07905	47.07966
Eficiência	0.554	0.683488	0.1846
T. comunica. (s)	4.06474	2.2605	1.422975
T. interrup. (s)	0.90568	0.51724	0.228019
T. contenção (s)	0.07446	0.065526	0.14552

Tabela 6.9. Distribuição dos valores ótimos do processo "bloco inicial".

A topologia árvore em conjunto possui melhor desempenho do que as outras topologias em termos de "speedup" e eficiência. Na execução dos módulos de forma separada, pode ser observado que a topologia em árvore também mostra bom desempenho com poucos processadores, conseguindo um "speedup" muito próximo do maior apresentado pela topologia malha.

Na figura 6.6 são mostrados os gráficos resultantes da simulação do bloco inicial, composto pelos módulos de "remoção decomposição" e "comparação em grupo de a 10".

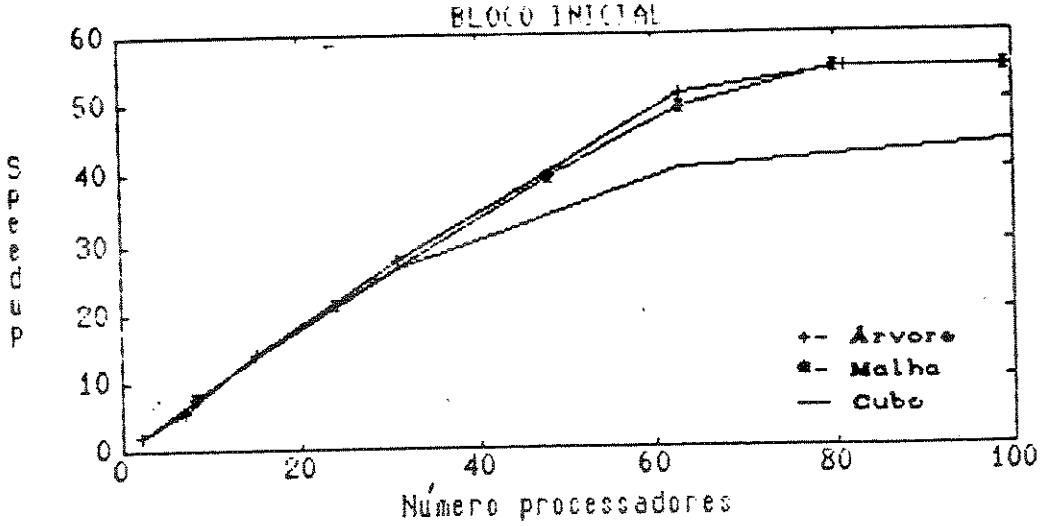


fig. 6.6a. "Speedup" do processo denominado "bloco inicial".

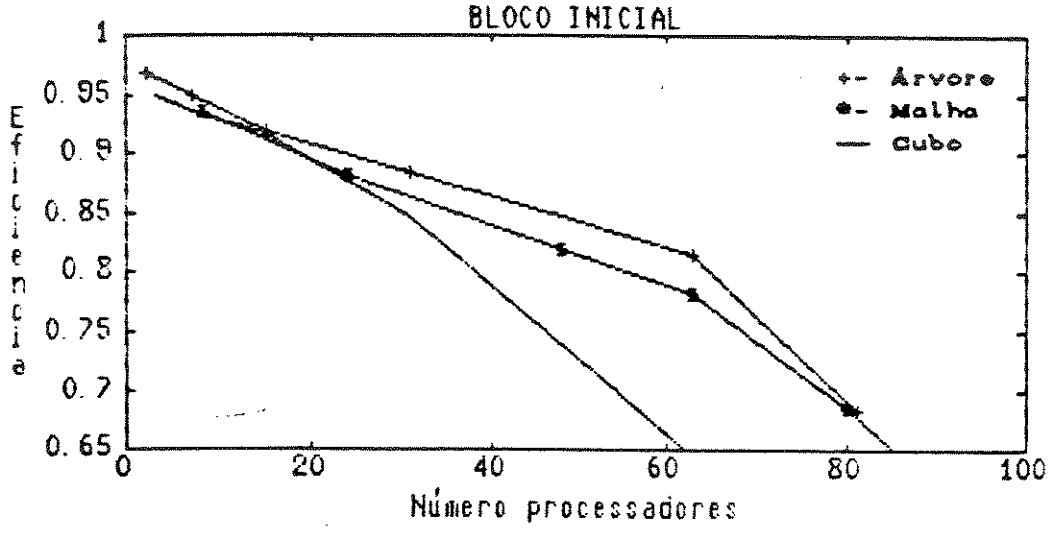


fig. 6.6b. Eficiência do processo "bloco inicial".

Nota-se a similaridade existente entre os gráficos apresentados nas figuras 6.4 e 6.6, podendo verificar-se que existe um pequeno aumento do "speedup" e menor eficiência, resultante do aumento das tarefas a distribuir. Também, observam-se os pontos máximos do "speedup" do gráfico da figura 6.6 que é próximo de 100 processadores. Para um número maior de processadores, a eficiência mostra uma queda, o que indica que não existirá um "speedup" significativamente melhor, considerando-se mais processadores disponíveis.

6.7 CONCLUSÕES DAS SIMULAÇÕES REALIZADAS.

Do estudo realizado, estima-se que o processo de comparação de triângulos, do algoritmo "L.S.E.", empregando a técnica do pintor, é a etapa que consome o maior tempo de processamento, seja sua execução sequencial ou paralela. Por tal razão, os melhoramentos e a estratégia de particionamento e distribuição desse módulo, são de vital importância no processamento do algoritmo, no menor tempo possível.

Introduzindo mudanças no particionamento dos módulos de remoção e decomposição, apenas se consegue uma redução de tempo, que é ínfimo se comparado com o processo de comparação de

triângulos.

Com o objetivo de melhorar o comportamento do processo de comparação, foram agrupados conjuntos de processos a serem executados num processador. Para isso, deve-se considerar que os dados de envio e retorno aumentam de forma linear, dependendo da quantidade de processos agrupados. Esse aspecto faz com que exista um agrupamento ótimo, como foi demonstrado na seção 6.5.2. A consideração de um número maior de tarefas por processador, resulta numa diminuição do desempenho.

No estudo do comportamento dos módulos iniciais do algoritmo L.S.E., observa-se que o comportamento ideal difere significativamente do comportamento real. O "speedup" real para este conjunto de dados é de 51 a 55 vezes, correspondendo às topologias árvore e malha ("mesh"), respectivamente. Esse valor corresponde a 15% do "speedup" ideal, indicado no trabalho de Regonessi [REGO,1990].

Neste estudo o "speedup" real máximo foi obtido para 100 processadores.

Com o objetivo de que o "bloco inicial" seja executado em tempo real, isto é, tempo de execução menor ou igual a 1/30 segundos para televisão ou 1/24 segundos para cinema, foram realizados os testes, onde cada tick foi de 1.715, 0.1715, 0.01715 e 0.001715 ms. para executar o processo de comparação de triângulos, considerando blocos de 10 tarefas de comparação por processador e mantendo constante o custo de comunicação.

A topologia malha apresentou o menor tempo de execução, porém, longe do tempo real (0.153ms), com uma treliça de 4x4 e um baixo "speedup" (0.122), para um tick de 0.001715ms.

Utilizando o mesmo principio de "dividir e conquistar", foram levantadas as seguintes hipoteses das 5995 tarefas de comparação de triângulos (110 triângulos):

a) Dividir as 5995 tarefas em dois grupos de 3000 e 2995 tarefas, e esses, na sua vez dividi-los em blocos de 20 tarefas de comparação a serem distribuídas por processador. Novamente a topologia malha apresenta o menor tempo de execução obtido de 0.04093 segundos, com um tick de relógio de 1.715 μs. Essa divisão seria apropriada para tempo real de cinema, mas não para televisão.

b) Dividir as 5995 tarefas em três grupos de 2000, e isses em blocos de 20 tarefas. Os dados obtidos são os da tabela 6.10.

tick = 17.15 μs

Topologias / Medidas	Malha	Árvore	Hipercubo
Nºprocessadores	15	31	63
T. execução(s)	0.0821664	0.08284	0.06257
"Speedup"	3.49649	3.4774	1.795

tick = 1.715 μs

Topologias / Medidas	Malha	Árvore	Hipercubo
Nºprocessadores	8	15	15
T. execução (s)	0.028404	0.031199	0.062
"Speedup"	0.3959	0.36055	0.18095

Tabela 6.10. Processo de comparação de triângulos considerando o processo em grupos de 2000 tarefas.

Pode-se observar na tabela 6.10, que a topologia malha com uma treliça de 3x3, apresenta o menor tempo de execução. Nos valores apresentados para a operação com um tick 1.715 μs e 17.15 μs, pode-se notar que as respostas de tempo de execução são próximas, no entanto, o "speedup" difere significativamente, isso permite ver claramente que o gasto de tempo de comunicação passa a ter importância na execução das tarefas de curta duração. Esse fato invalida o estudo, porque ao diminuir o tempo de relógio, a execução sequencial será menor, e a não redução do tempo de

comunicação, resultará em que a execução sequencial pode ser melhor do que o processamento paralelo.

Para que o processamento paralelo seja de interesse, sua utilização nesse tipo de aplicação, deverá apresentar taxas de transferências cada vez mais rápidas, otimizando a forma de comunicação das mensagens, minimizando as contenções e as interrupções dos processos, permitindo uma trajetória expedita da transmisão de mensagens entre processadores.

CAPÍTULO 7

CONCLUSÕES

7.1 DA CONSTRUÇÃO DO SISTEMA ORIGINAL.

Antes de fazer a implementação do "sistema original", utilizando a placa IMS B008, trabalhou-se com a placa IMS B004 com um transputer só, existente no labóratorio LCA. Essa mudança obrigou a considerar toda a lógica que se refere à alocação dos processos, mapeamento e configuração da topologia.

Em um transputer só, a implementação pode ter toda a lógica empregada na confeção de um programa em linguagem de alto nível (Pascal, C, Fortram, etc.) , como : uso de variáveis e constantes globais, emprego de procedimento com variáveis globais, pouco ou nada de manejo de canais, etc. Contudo, na hora de fazer a mesma implementação numa placa com vários transputers, surgiram problemas, porque os procedimentos a serem distribuidos na rede deviam ser: independentes, sem variáveis ou constantes globais, sem protocolos globais e sua comunicação externa, feita por canais de entradas e saídas.

Se essas afirmações fossem consideradas do início, o passo para implementar programas em vários transputers seria direto.

Como principais resultados iniciais, considerando os aspectos de implementação, observaram-se:

1) Com relação à implementação, o MMS2 não mapeia a conexão do link 3 do TRAM 0. Essa situação obrigou à realização de configurações onde existiam um tranputer "host" e o transputer "root" (figura 4.21), criando um engarrafamento e desperdiçando um transputer. Mediante o emprego dos programas testes fornecidos pelo TDS2 e utilitários "check ! mtest "

detetou-se a existência da conexão, refazendo o mapeamento (figura 4.22).

2) Detetou-se que as facilidades das sentenças de alto nível seletivas (ALT) não aproveitam as caraterísticas do processamento paralelo, oferecidas no transputer, porque a sentença atua quando existe uma entrada pronta, executando-se sem deixar progredir outra entrada até ficar liberada novamente. Dessa maneira as comunicações tornam-se quase sequenciais, além de atribuir prioridades arbitrárias. Shallow [SHALL,1990] e Torngrem [TORNG,1990] fazem um estudo e apresentam formas de minimizar esse efeito.

   A melhor maneira encontrada para não utilizar a sentença seletiva ALT, foi criar processos concorrentes independentes de entradas e saídas, porque se aproveitam na execução paralela das interfaces de comunicação.

3) Outro aspecto que foi comprovado foi a facilidade de uso do protocolo variante. Porém, sua utilização degrada o desempenho da aplicação, comparando-se ao desempenho de um protocolo de comunicação de tamanho fixo. Esse fato é resultado da operação dos canais de comunicação da linguagem OCCAM2.

7.2 DA CONSTRUÇÃO DO SIMULADOR.

Antes da versão final do simulador apresentado neste trabalho, tentou-se aproveitar as caraterísticas que oferece o transputers, para realizar simulação dos processos por meio do Timer (temporizadores), para representar a execução do processo simulado. Foi descartado pela dificuldade em registrar de forma real os tempos de contenção, interrupção e de predizer a sequência de processamento. Somente foi possível resgatar a lógica para

realizar o "sistema original" e quantificar os tempos gastos em comunicação, mas o processo simulado obrigava a esperar a finalização da hibernação do Timer.

A implementação do "sistema original" permitiu ir realizando otimizações na operação dos processos envolvidos resultando um simulador mais confiável, que representava a execução real do "sistema original", para um número maior de processadores.

No desenvolvimento do "sistema original" e aplicação de conhecimento de simulação de eventos discretos, foi possível obter uma ferramenta que permite quantizar o tempo gasto na comunicação dos dados, tempo de contenção e registrar o tempo de interrupção dos processos que operam com prioridade baixa. Isso faz com que a ferramenta desenvolvida represente um comportamento da aplicação mais próximo à realidade.

A escolha da realização de um simulador sincronizado por eventos, vem do fato que esta é a forma mais natural de execução dos processos mediante a utilização do transputer e a linguagem OCCAM2.

7.3 DO ESTUDO DA APLICAÇÃO.

Do estudo realizado da aplicação pode-se dizer que as topologias que apresentam melhor desempenho na execução dos diferentes módulos da aplicação, correspondem às topologias árvore e malha. A topologia árvore apresenta a vantagem de que com poucos processadores consegue um comportamento próximo ao apresentado pela topologia malha.

Infelizmente, ante a exigência do processamento em tempo real da aplicação, pode-se concluir que a utilização de máquinas MIMD fracamente acopladas, com o equipamento utilizado e forma de operação, não é possível sua execução em tempo real, pela alta taxa de transferência de dados do módulo de comparação de

triângulos. Este fato indica que o custo de comunicação toma importância ante a taxa de transferência de dados. O processo de comparação de triângulos consome 93% do tempo de processamento do "bloco inicial".

Da análise e do estudo do comportamento da aplicação pode-se afirmar o seguinte:

a) Deve ser feita a otimização da distribuição das tarefas no processo de comparação de triângulos, considerando a eliminação da comparação dos triângulos de uma mesma face.

b) É possível realizar o processo de comparação, mediante a distribuição em pacotes de triângulos de uma face a serem comparados com outra face.

c) Pode-se avaliar a possibilidade de utilizar máquinas MIMD fortemente acopladas, hibridas ou máquinas fracamente acopladas com mecanismos de comunicação otimizados.

7.4 CONCLUSÕES FINAIS.

Este trabalho contribui com um conjunto de ferramentas que permite representar o comportamento de uma aplicação na forma mais próxima à realidade, já que considera os custos reais na execução do algoritmo paralelo. Além disso pode servir para realizar a otimização dos processos envolvidos no processamento, como: otimização das rotinas de comunicação, eliminação do excesso de interrupções, minimização das contenções, etc.

Outro aspecto importante a destacar é o estudo e os programas desenvolvidos na utilização de uma rede de transputer, onde foi feito o mapeamento das topologias de maneira a permitir a ampliação do número de processadores a utilizar em forma física. Isso servirá para desenvolver trabalhos futuros utilizando transputers.

triângulos. Este fato indica que o custo de comunicação toma importância ante a taxa de transferência de dados. O processo de comparação de triângulos consome 93% do tempo de processamento do "bloco inicial".

Da análise e do estudo do comportamento da aplicação, pode-se afirmar o seguinte:

a) Deve ser feita a otimização da distribuição das tarefas no processo de comparação de triângulos, considerando a eliminação da comparação dos triângulos de uma mesma face.
b) É possível realizar o processo de comparação, mediante a distribuição em pacotes de triângulos de uma face a serem comparados com outra face.
c) Pode-se avaliar a possibilidade de utilizar máquinas MIMD fortemente acopladas, hibridas ou máquinas fracamente acopladas com mecanismos de comunicação otimizados.

7.4 CONCLUSÕES FINAIS.

Este trabalho contribui com um conjunto de ferramentas que permite representar o comportamento de uma aplicação na forma mais próxima à realidade, já que considera os custos reais na execução do algoritmo paralelo. Além disso pode servir para realizar a otimização dos processos envolvidos no processamento, como: otimização das rotinas de comunicação, eliminação do excesso de interrupções, minimização das contenções, etc.

Outro aspecto importante a destacar é o estudo e os programas desenvolvidos na utilização de uma rede de transputer, onde foi feito o mapeamento das topologias de maneira a permitir a ampliação do número de processadores a utilizar em forma física. Isso servirá para desenvolver trabalhos futuros utilizando transputers.

O estudo realizado em relação à aplicação, ao planejamento do "sistema original" e, finalmente, ao simulador, é perfeitamente aplicável para desenvolver um trabalho semelhante que considere as máquinas MIMD fortemente acoplada.

Este seria um aspecto de interesse, visto que o algoritmo L.S.E. apresenta uma alta taxa de transferência de dados nos processos de comparação de triângulos.

CAPÍTULO 8

REFERENCIAS

[ANGER,1990] Anger, F. Hwang, J. e Chow, Y. "Scheduling with sufficient lossely coupled processors", Journal of Parallel and Distributed Computing, 9, pp. 87-92.

[BAGCHI,1989] Bagchi, K. Olsen, O. Christensen, A. e Sorensen, L. "Simulation and Design of Message Routing Systems for Networks of Transputers", Annual Simulation Symmposium, March.

[BURNS,1988] Burns, A.""Programming in Occam2",Addison Wesley, England.

[DUNCA,1990] Duncan, R. "A survey of parallel computer architectures", IEEE Computer, February, pp. 5-16.

[EAGER,1989] Eager, D., Zahoran, J. and Lazowka, E. "Speedup Versus Efficiency in parallel systems", IEEE Transaction on Computers Vol 38, 3, pp. 408-423.

[ELREW,1990] El-Rewini, H and Lewis, T "Scheduling parallel program tasks onto arbitrary target machines", Journal of Parallel and Distributed Computing, 9, pp. 138-153.

[FLYNN,1972] Flynn, M. "Some computer organization and their efetiveness", IEEE Transactions on Computer, Vol. C-21, 9, pp. 948-960.

[FUJI,1991] Fujimoto, R. "Parallel Discrete event Simulation", Communications Of the ACM, Vol 33, 10, October.

[GAUDI,1989] Gaudiot, J. L. and Lee, H. T. "Occamflow: Amethodology for programming multiprocessor systems", Journal of Parallel and Distributed Computing, 7, pp. 96-124.

[GONZA,1977] Gonzalez, M. "Deterministic Processor Scheduling", ACM Computing Survey Vol. 9, 3, pp. 173-204.

[HARRI,1986] Harrington, S. "Computers Graphics: A programming approach", McGraw-Hill, Singapore, 2ª Edição.

[HAYES,1989] Hayes, J. and Mudge, T. "Hipercube Supercomputers". Proceeding of IEEE Vol 77, 12, pp. 1829-1841.

[HERRA,1988] Herrada, E., Beivide, R. y Navarro, J. "Sistemas Multiprocessadores de paso de mensajes", Mundo Electrónico , 181, pp. 47-56.

[HERRA,1990] Herrada, E. Arrabarrena, A. Balcazar, J. e Izen, O. "An optimal topology for multicomputers system based on a mesh of transputers", OCCAM user group newsletter, January, pp. 31-39.

[HWANG,1984] Hwang, K. and Briggs,F.A. "Computer arquitecture and parallel processing", Mc Graw-Hill, New York.

[IB008,1988] Inmos B008. "IMS B008 User guide and reference manual ", INMOS Ltda. England.

[IMMS,1989] Inmos MMS. " S708 User guide ", INMOS Ltda. England.

[IORM,1988] Inmos ORM. "Occam 2, reference manual", Prentice Hall, Cambridge.

[ITD,1988] Inmos TD. "Transputer Databook", Bath Press Ltd. Bath.

[ITDS,1988] Inmos TDS. "Transputers development System ", Prentice Hall, London.

[ITIS,1988] Inmos TIS. "Transputer instruction set; A compiler writers guide ", Prentice Hall, London.

[ITRM,1988] Inmos TRM. "Transputer reference manual", Prentice Hall, London.

[JONES,1988] Jones, G. e Goldsmith, M. "Programming in OCCAM2", Prentice Hall, London.

[KARP,1990] Karp, A. and Flatt, H. "Measuring parallel processor performance", Communications of the ACM, Vol. 33, 5, pp. 539-543.

[KIRNER,1991] Kirner, C. "Arquiteturas de Sistemas Avançados de Computação", Anais da EPUSP/IEEE em sistemas de computação de alto desempenho, Março.

[LAN,1990] Lan, Y., Esfahanian, A. and Ni, L. "Multicast in Hipercube Multiprocessor", Journal of Parallel and Distributed Computing, pp. 30-41.

[LEUNG,1989] Leung, J. and Young, G. "Minimizing schedule length subject to minimum flow time ", SIAM Journal Computing, Vol 18, 2, April, pp. 314-326.

[MISRA,1986] Misra, J. "Distributed discrete-event simulation", ACM Computing Surveys, Vol. 8, 1, pp. 39-65.

[PADMA,1990] Padmanabhan, K. "Cube structures for multiprocessors", Communications of the ACM Vol 33, 1, pp. 43-52.

[PETER,1981] Peterson, J. "Petri Net theory and modeling systems", Prentice Hall Inc. N. York.

[POUNT,1987] Pountain, D. and May, D. "A tutorial introduction to Occam programming", INMOS, London.

[QUINN,1988] Quinn, M. J. "Designing efficient algorithms for parallel computers", Mc Graw-Hill, Singapore.

[REGO,1990] Regonessi, M. "Relatorio de avanço de Tese".

[RETTB,1990] Rettberg, R., Crowther, W. Carvey, P. e Tomlinson, R. "The Monarch Parallel Processor Hardware Design", IEEE Computer, April, pp.

[SAAD,1988] Saad, Y. and Schultz, M. "Topological properties of Hypercubes", IEEE Transactions on Computers Vol 37, 7, pp. 867-872.

[SAAD,1989] Saad, Y. and Schultz, M. "Data communication in Hypercubes", Journal of Parallel and Distributed Computing,6, pp. 115-135.

[SHALL,1990] Shallow, P. "Really efficient multiple buffering in occam and efficient fair ALT", Occam user group, newsletter, january, pp. 42-49.

[SYMON,1980] Symons, F. Billington, J. e Kirton, P. "Representation of Data comunications processes", Australian Telecomunication Research, Vol. 14, 1.

[TORNG,1990] Törngren, M. "Transputer and occam based control system in electronic control of machines", Occam user group, newsletter, january, pp. 66-70.

ANEXO A

ALGORITMOS DE COMUNICAÇÃO.

A.1 INTRODUÇÃO.

Com a finalidade de realizar a implementação do "sistema original" e simulador, foram desenvolvidos e codificados algoritmos de comunicação que consideram o envio de mensagens de um processador a qualquer outro na topologia. Considerando como base os trabalhos analisados [HERRA,1990], [GAUDI,1989], [HWANG,1984], [HAYES,1989] e [SAAD,1989]. As topologias mais abordadas na proposição de algoritmos de comunicação são as topologias hipercúbicas e malha. Pela forma da análise e implementação dos algoritmos resulta quase direta a extensão para as topologias árvore, anel e midimew.

A.2 TOPOLOGIA ANEL.

O algoritmo de comunicação desenvolvido para essa topologia, considera que os processadores estão numerados em sentido horário e a comunicação será em essa direção para os processadores menores ou iguais a $\lceil n/2 \rceil$, o retorno será em sentido contrário. Em forma inversa para os processadores maiores que $\lceil n/2 \rceil$, a fim de realizar o mínimo de deslocamento das mensagens nos processadores intermediários que participam.

```
PROC anel ( VAL INT  an.fonte,
            [2] CHAN OF mensagemO inp,
            [2] CHAN OF mensagemO out)
  INT an.m, signo, media :
  SEQ
    media := total.nos / 2
    ALT i = 0 FOR 2
      inp[i] ? mensaje
        SEQ
          an.destino:=mensaje[0]
          an.m := an.destino - an.fonte
          IF
            an.m <> 0
              SEQ
                --<<<  vai para outro vizinho
                signo := (-1)
                IF
                  an.m > 0
                    signo := 1
                  TRUE
                    an.m := an.m * (-1)
                IF
                  an.m > media
                    signo := signo * (-1)
                  TRUE
                    SKIP
                --<<<  se envia
                IF
                  signo = 1
                    out [2] ! mensaje
                  signo = (-1)
                    out [1] ! mensaje
                  TRUE
                    SKIP
                -->>>
            an.m = 0
              out [0] ! mensaje
            TRUE
              SKIP
:
```

fig. A.1, algoritmo de comunicação da topologia anel.

A.3 TOPOLOGIA ÁRVORE.

Na confeção do algoritmo de comunicação, foram consideradas as definições básicas da estrutura de dados árvore binário, tais como: determinação do processador pai e dos processadores filhos e profundidade. A figura A.2 mostra o algoritmo de comunicação implementado em OCCAM2, utilizando nos sistemas simulador e "original".

```
PROC arvore (VAL INT a.fonte,[4]CHAN OF mensagem0 inp,
             [4]CHAN OF mensagem0 out)
  INT a.fonte, a.destino, na.destino, na.fonte, dif.ni, aux.ni,
      ni.a1, ni.a2, restim, m.a :
  BOOL ar.final :
  PROC loga.2(INT resultado, VAL INT x )
    REAL32 r32.aux :
    SEQ
      r32.aux:=LOG32((REAL32 ROUND (x+1)))
      resultado:=(INT TRUNC (r32.aux)) + 1
  :
  SEQ
    ALT i = 0 FOR 4
      inp[i] ? mensaje
        SEQ
          a.destino := mensaje[0]
          m.a:=a.destino - a.fonte
          IF
            m.a <> 0
              SEQ
                loga.2(na.destino,a.destino)
                loga.2(na.fonte, a.fonte)
                dif.ni:= na.destino - na.fonte
                IF
                  dif.ni <= 0
                    ar.final:=TRUE
                  dif.ni > 0
                    SEQ
                      aux.ni:=na.destino
                      WHILE na.fonte <> aux.ni
                        SEQ
                          ni.a1:=((a.destino+1)/2) - 1
                          aux.ni:=aux.ni -1
                          ni.a2:=a.destino
                          a.destino:=ni.a1
                      IF
                        ni.a1 <> ar.fonte
                          ar.final:=TRUE
                        ni.a1 = a.fonte
                          SEQ
                            restim:= ni.a2 REM 2
                            IF
                              restim = 0
                                out [3] ! mensaje
                              restim <> 0
                                out [2] ! mensaje
                IF
                  ar.final
                    out [1] ! mensaje
            m.a = 0
              out [0] ! mensaje
            TRUE
              SKIP
  :
-->>>
```

fig. A.2, algoritmo de comunicação da topologia árvore.

A.4 TOPOLOGIA HIPERCÚBICA.

A figura A.3 mostra o algoritmo de comunicação numa topologia Hipercúbica, que corresponde à codificação do texto estruturado apresentado em Hayes [HAYES,1987].

```
PROC  n.cubo (VAL INT n.fonte,
              [5] CHAN OF mensagemO inp,
              [5] CHAN OF mensagemO out)
  IN T kei, n.ruta, k.esimo, kdara, m.c :

SEQ
  AL T i = 0 FOR 5
     inp [i] ? mensaje
       SEQ
         n.destino:= mensaje[0]
         m.c:=n.destino-n.emisor
         IF
           m.c <> 0
             SEQ
               n.ruta:=n.emisor >< n.destino
               kei := 0
               k.esimo := 1
               flag.final := TRUE
               WHILE (kei < n.nos) AND flag.final
                 SEQ
                   kdara := n.ruta /\ k.esimo
                   IF
                     kdara = k.esimo
                       SEQ
                         flag.final := FALSE
                         out [kei + 1] ! mensaje
                 TRUE
                   SKIP
               kei:= kei + 1
               k.esimo := k.esimo * 2
         m.c = 0
           SEQ
             out [0] ! mensaje
         TRUE
           SKIP
:
-->> >
```

fig. A.3, Algoritmo de comunicação na topologia Hipercúbica.

Veja Törngrem [TöRN, 1990], Shallow [SHALL,1990] e Jones [JONES,1989].

O Simulador é incapaz de considerar o fato mencionado no item (d), porque quebra a regra imposta para evitar o erro casual, que considera a execução dos eventos de menor tempo.

CAPÍTULO 6

ANÁLISE DE RESULTADOS

6.1 INTRODUÇÃO.

Neste capítulo são apresentados os resultados obtidos a partir das simulações realizadas, considerando os tempos de execução dos módulos do algoritmo L.S.E. definido em Regonessi [REGO,1990].

É realizada também uma comparação entre os resultados ideais do estudo teórico e os resultados reais considerando os tempos de comunicação obtidos através do "Simulador" desenvolvido neste trabalho.

As medidas de interesse apresentadas neste capítulo, são: "speedup", eficiência, tempo de execução, tempo gasto em comunicação, tempo gasto em contenção e tempo de interrupção. Dessas medidas, obtidas para diferentes número de processadores, podem-se extrair algumas conclusões, que dependerão das exigências impostas para a aplicação. Nesse caso, optou-se por mostrar o comportamento geral da aplicação, com ênfase no melhor "speedup", obtido para cada topologia.

6.2 FONTE DOS DADOS CONSIDERADOS.

Como foi mencionado anteriormente na seção 2.1, da implementação sequencial do algoritmo "Linhas e Superfícies Escondidas", foram determinados os tempos de execução dos módulos de remoção de faces visíveis, decomposição em triângulos e comparação de triângulos. O estudo foi baseado numa figura padrão, correpondente a uma tampa de chaleira, com 92 faces, considerando uma projeção paralela de topo, das quais 70 faces são visíveis.

Os dados obtidos nas medidas realizadas na execução sequencial, são apresentados nas seções seguintes, como parte do

estudo em condições ideais da execução paralela.

6.3 MEDIDAS NO MÓDULO DE REMOÇÃO DE FACES VISÍVEIS.

6.3.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Do estudo feito sobre o comportamento do módulo de remoção de faces visíveis na implementação sequencial, foram extraídos os dados apresentados na tabela 6.1. Dados que foram utilizados para realizar a medida de rapidez ou ganho ideal para essa fase. Considerando um número ilimitado de processadores, o tempo mais longo corresponderá ao tempo de execução paralela (9 tick, onde cada tick a 1.715 ms) e, como tempo de execução sequencial, a soma dos tempos de todas as tarefas, obtem-se o seguinte "speedup" (com base na eq. 3.1):

$$S(n) = \frac{300}{9} = 33.3 \qquad (6.1)$$

Iterações	Ticks
78	3
12	4
2	9

Tabela 6.1. Distribuição dos dados do processo de Remoção de faces visíveis.

6.3.2 MEDIDAS NO SIMULADOR.

Foram consideradas as tarefas apresentadas na tabela 6.1, executadas nas diferentes topologias que o simulador oferece. Os resultados obtidos são apresentados na tabela 6.2.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nº processadores	60	30	31	48
T. execução (s)	0.0490024	0.049107	0.0088	0.04059
"Speedup"	10.49948	11.99593	7.4781608	12.09886
Eficiência	0.1544	0.397844	0.24129	0.2644556
T. comunica. (s)	1.058128	0.211844	0.12244	0.82217
T. interrup. (s)	0.18275	0.036000	0.0109125	0.06566
T. contenção (s)	0.0179071	0.0042500	0.0006792	0.007457

Tabela 6.2. Distribuição dos dados ótimos, da execução do processo de remoção.

Do estudo do comportamento do processo de remoção de faces visíveis, considerando as topologias mais comuns pode-se mencionar o seguinte:

a) A topologia que apresentou o melhor tempo de execução foi a topologia malha, com um "speedup" correspondente ao 35% do "speedup" ideal.
b) A topologia árvore, em termos de eficiência, foi a que apresentou o melhor comportamento.
c) A topologia cúbica produziu os menores tempos de comunicação e interrupção dos processos, fornecendo um "speedup" razoável, considerando poucos processadores.

A figura 6.1 apresenta os gráficos do comportamento da rotina de remoção de faces visíveis. Nesses gráficos são apresentados o "speedup" e a eficiência para diferentes números de processadores.

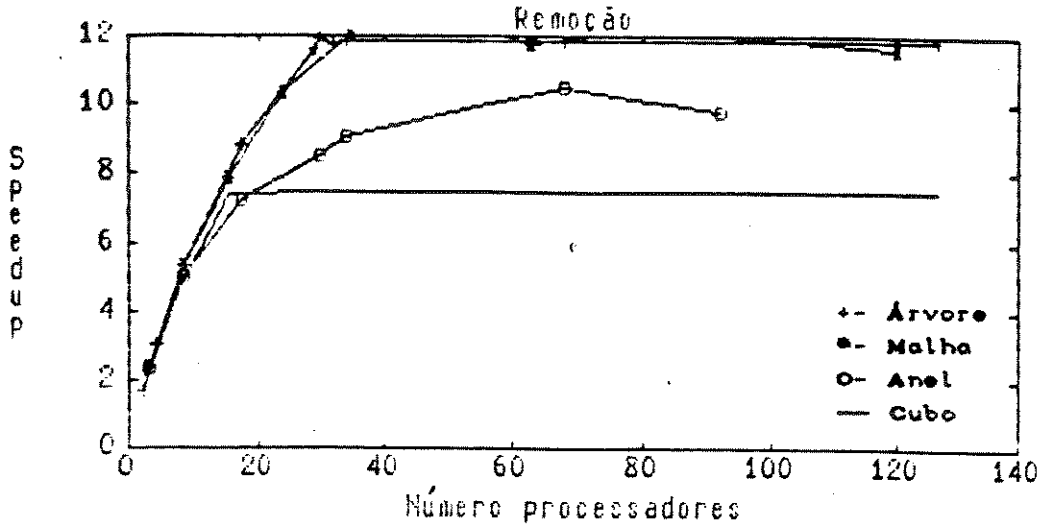


fig. 6.1a. "Speedup" do processo de remoção.

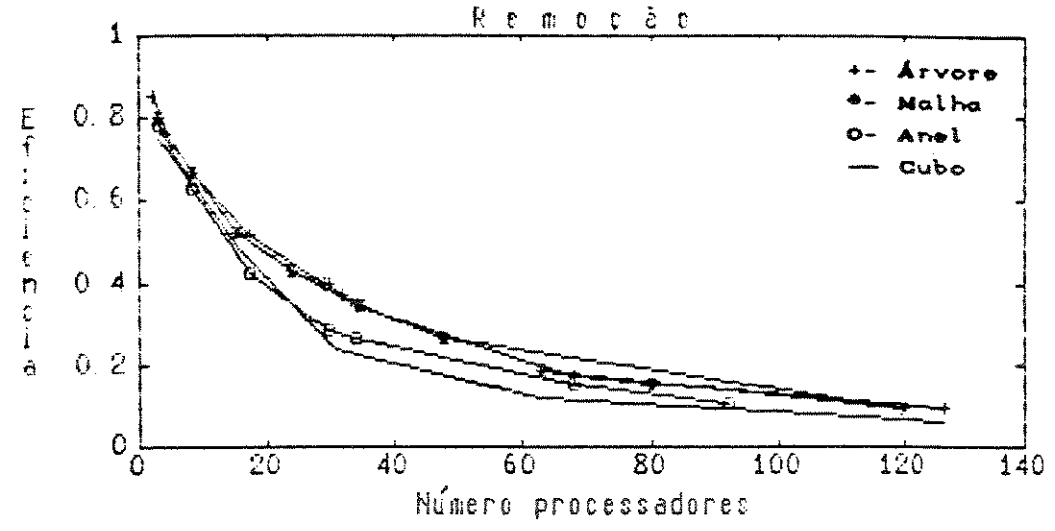


fig. 6.1b. Eficiência do processo de processo de remoção.

No gráfico da figura 6.1a, observa-se que o "Speedup" máximo é obtido com poucos processadores nas topologias malha e árvore. Observa-se que o "speedup" mantem-se constante, se o número de processadores aumenta. A topologia anel, por sua composição, oferece o melhor "speedup" para um número maior de processadores do que as outras topologias. A topologia cúbica mostra um comportamento mais regular, porque, uma vez alcançado o melhor "speedup", o aumento do número de processadores não modificará seu comportamento. Isso deve-se à estratégia de

distribuição das tarefas feitas pelo processo de escalonamento, em forma dinâmica.

A figura 6.1b mostra a utilização média dos processadores. Pode observar-se que a melhor curva de eficiência é apresentada pelas topologias árvore e malha, mas as curvas tendem a ser muito similares.

6.4 MEDIDAS DO MÓDULO DE DECOMPOSIÇÃO EM TRIÂNGULOS.

6.4.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Para a medição dos tempos relativos as tarefas no módulo de "decomposição das faces visíveis em triângulo", foram consideradas as faces visíveis resultantes do processo de remoção de faces. Os dados obtidos na decomposição dessas faces visíveis são apresentados na tabela 6.3.

Iterações	Ticks
25	5
49	6
2	8

Tabela 6.3. Distribuição dos dados do processo de Decomposição das faces visíveis.

O cálculo do "speedup", para esse módulo, é realizado da mesma forma que o cálculo do "speedup" ideal para a remoção de faces na seção 6.2. O valor obtido nesse caso é o seguinte :

$$S(n) = \frac{399}{8} = 49.87 \qquad (6.2)$$

6.4.2 MEDIDAS DO SIMULADOR.

De maneira similar à seção 6.3.2, foram consideradas as medidas dos tempos que são apresentadas na tabela 6.3, e as tarefas foram distribuídas aos processadores, nas diversas topologias permitidas pelo simulador. Os resultados da simulação

são mostrados na tabela 6.4.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nºprocessadores	63	48	31	63
T. execução (s)	0.04279	0.0352212	0.05198	0.034194
"Speedup"	15.99516	19.4281722	13.169922	20.04097
Eficiência	0.2538	0.4047585	0.4246	0.31811
T.comunica. (s)	0.80778	0.1802187	0.10963	0.295192
T.interrup. (s)	0.087002	0.09492	0.009648	0.04988
T.contenção (s)	0.00267	0.002788	0.004499	0.0057948

Tabela 6.4. Distribuição dos valores ótimos da execução do processo de Decomposição das faces visíveis.

Observa-se na tabela 6.4, por um lado, que o melhor tempo de execução e, consequentemente, o melhor "speedup", é obtido com a topologia malha; porém com um número de processadores maior. Por outro lado, a topologia árvore oferece um "speedup" próximo ao melhor apresentado pela topologia malha. Outro aspecto interessante, é que as tarefas consideradas possuiam maior tempo de execução e o "speedup" obtido na simulação corresponde ao 40% do "speedup" apresentado no estudo teórico.

A figura 6.2 apresenta os gráficos das medidas obtidas para o estudo de comportamento desse módulo no simulador.

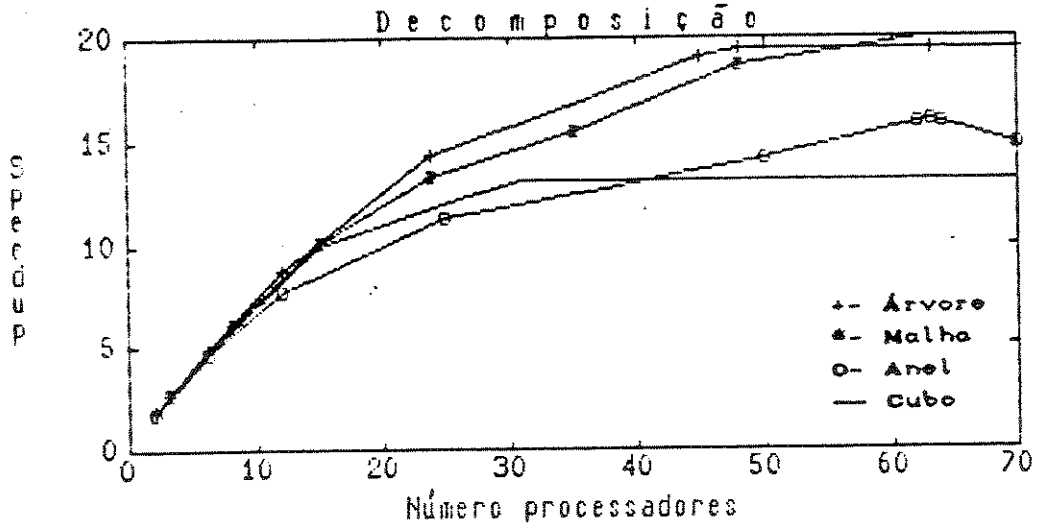


fig. 6.2a. "Speedup" do processo de Decomposição em triângulos

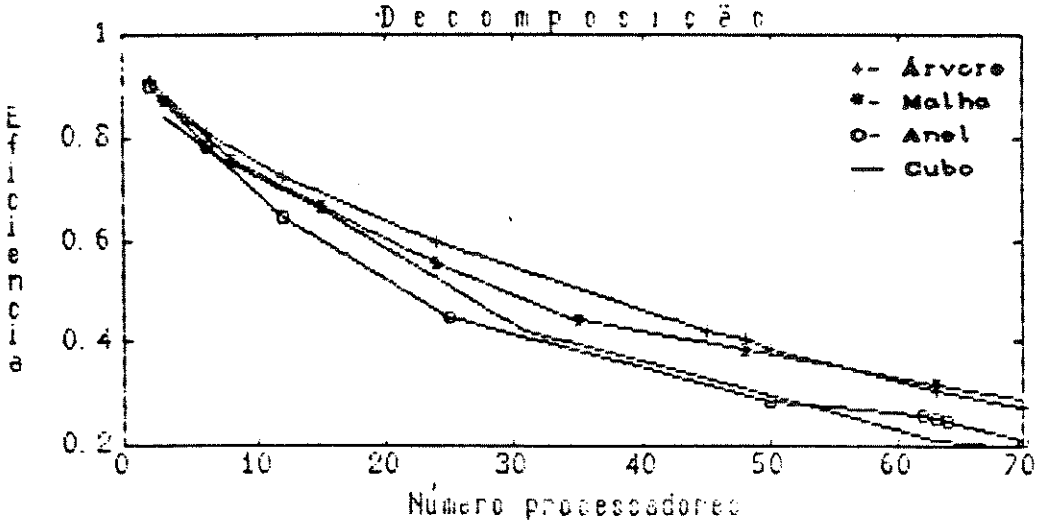


fig. 6.2b. Eficiência do processo de decomposição em triângulos.

No gráfico da figura 6.2a, novamente as topologias árvore e malha apresentam o melhor "speedup", com um aumento gradativo até chegar acima dos 48 processadores e, depois, mantendo um comportamento constante, no caso da árvore, e decrecente no caso da malha. O mesmo fato acontece com a topologia cúbica e anel pela forma de conexão.

Do gráfico da figura 6.2b, pode-se observar um aumento da eficiência, resultante da consideração de tarefas com tempo de execução maior do que na distribuição das tarefas de remoção de faces. Dos processo analisados, a topologia cúbica chega mais rapidamente ao melhor "speedup" e a eficiência tem uma queda mais rápida do que as outras topologias.

6.5 MEDIDAS DO MÓDULO DE COMPARAÇÃO DE TRIÂNGULOS.

6.5.1 MEDIDAS NA IMPLEMENTAÇÃO SEQUENCIAL.

Considerando os dados coletados utilizando-se a tampa da chaleira, cada tick de relógio é de 1.715 ms. Apresenta-se na tabela 6.5 a distribuição dos dados do processo de comparação de triângulos.

Iterações	Ticks	Iterações	Ticks	Iterações	Ticks
4079	1	49	9	6	20
1554	2	19	11	13	21
17	4	10	12	20	22
34	5	11	13	12	23
18	6	34	14	4	24
59	7	11	16	4	26
37	8	4	17	6	33

Tabela 6.5. Distribuição dos dados do processo de Comparação de triângulos.

Pela análise dos dados verifica-se que o tempo total de execução sequencial corresponde a 11340 ticks e o tempo de maior caminho corresponde a 33 ticks. Portanto o "speedup" ideal dessa fase é:

$$S(n) = \frac{11340}{33} = 343.63636 \qquad (6.3)$$

Desse modo, 344 processadores executariam essa tarefa num tempo de 33 ticks (56.6ms).

6.5.2 MEDIDAS DO SIMULADOR.

Foi realizada a simulação da distribuição das tarefas do processo de comparação de triângulos, considerando-se o particionamento de tarefas a nível de comparação de pares de triângulos. A resposta da execução foi muito ruim, obtendo-se um "speedup" real de 8.0126 vezes, com uma malha de 15 processadores. A tabela 6.6 mostra o melhor comportamento da distribuição das tarefas desse módulo.

Topologias / Medidas	Anel	Malha	Hipercubo	Árvore
Nºprocessadores	14	15	31	15
T. execução (s)	2.9972	2.4209	3.550211	2.477
"Speedup"	6.4885	8.012672	5.47783	7.8482
Eficiência	0.46346	0.53417	0.176704	0.56059
T. comunica. (s)	16.7416	10.241	6.95964	10.29897
T. interrup. (s)	2.505187	1.59606	0.49656	1.27494
T. contenção (s)	0.559879	0.3199857	0.698148	0.415192

Tabela 6.6. Distribuição dos valores ótimos da execução do processo de comparação de triângulos.

Da análise dos dados, pode afirmar-se que nessa fase, o particionamento de tarefas não é apropriado, devido à quantidade de tarefas de curta duração (veja tabela 6.5). A figura 6.3 mostra os gráficos correspondentes à distribuição de 5995 tarefas, para diferentes números de processadores.

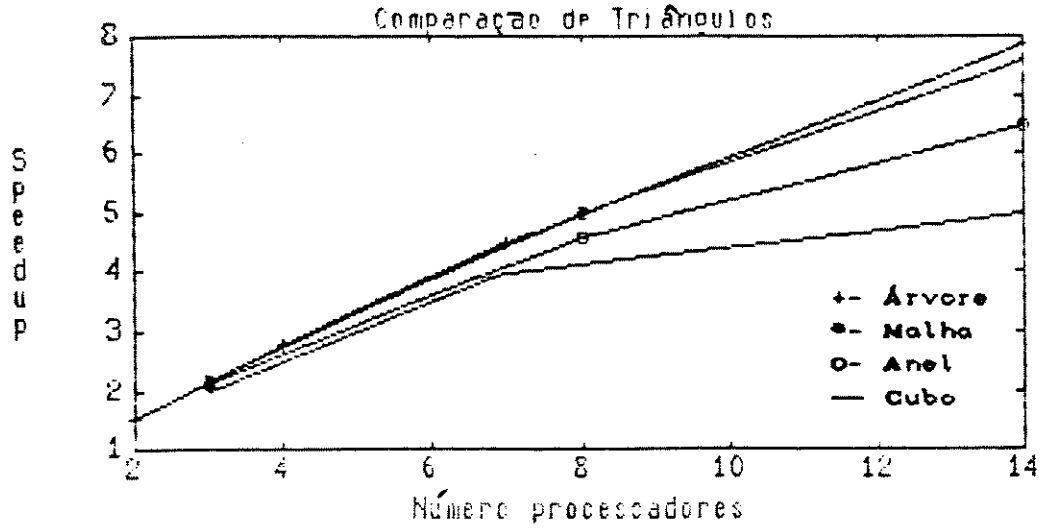


fig. 6.3a "Speedup" do processo de Comparação de triângulos.

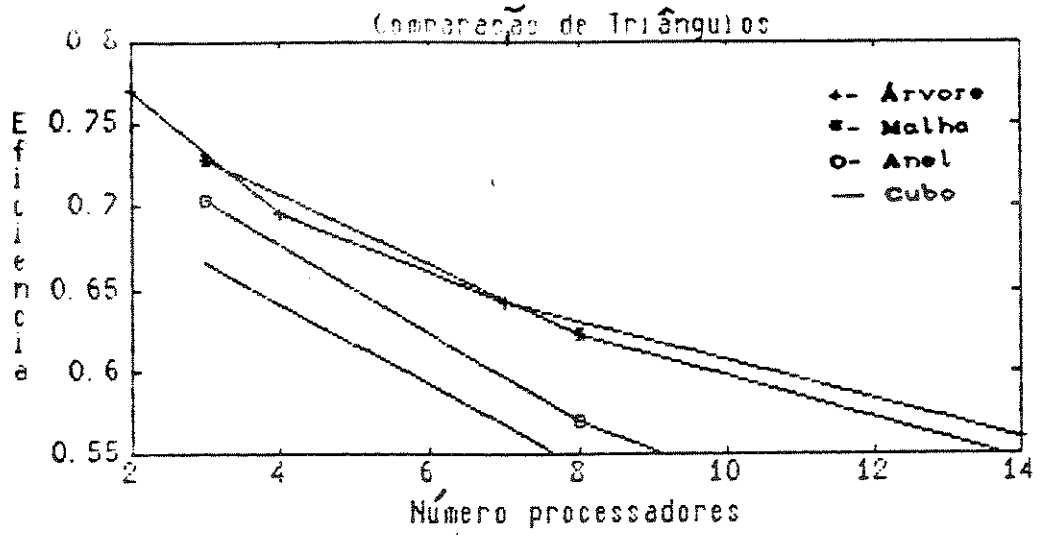


fig. 6.3b. Eficiência do processo de Comparação de triângulos.

Observa-se nos gráficos da figura 6.3, que a eficiência tem uma queda com apenas poucos processadores. Isso significa que o tempo das tarefas a serem distribuídas nos processadores deve

ser de maior duração, porque o tempo gasto em comunicação é crítico quando as tarefas são pequenas.

A fim de otimizar o tempo de execução do algoritmo, foram considerados particionamentos de granularidade mais grossa, agrupando de maneira aleatória as tarefas, num número fixo, a serem distribuídas nos processadores.

A tabela 6.7 mostra os melhores resultados obtidos na distribuição de blocos de tarefas (cada bloco é constituído de 10 processos de comparação de triângulos), executadas por um único processador (o teste foi feito para grupos de 5, 10 e 15 blocos tarefas).

Topologias / Medidas	Hipercubo	Árvore	Malha
Nºprocessadores	127	63	80
T. execução (s)	0.382785	0.380499	0.37686
"Speedup"	50.807	51.112548	51.606
Eficiência	0.1992	0.81181	0.645075
T. comunica. (s)	1.299	1.896756	3.54289
T. interrup. (s)	0.2148	0.491586	0.786621
T. contenção (s)	0.10828	0.098525	0.30406

Tabela 6.7. Distribuição dos valores ótimos do processo de comparação em lotes de 10 tarefas para cada processador.

Nota-se que a topologia em árvore possui um comportamento global melhor, com um número menor de processadores, porque apresenta uma eficiência melhor.

Da execução do conjunto de blocos de tarefas, onde cada bloco é constituído de 15 processos de comparação de triângulos por processador, observou-se, no entanto, que não se consegue um melhor "speedup". Esse fato indica que existe um número de tarefas que, agrupadas, apresentarão um melhor comportamento e que a consideração de um número maior de tarefas por bloco, empobrece a utilização do processamento para a aplicação.

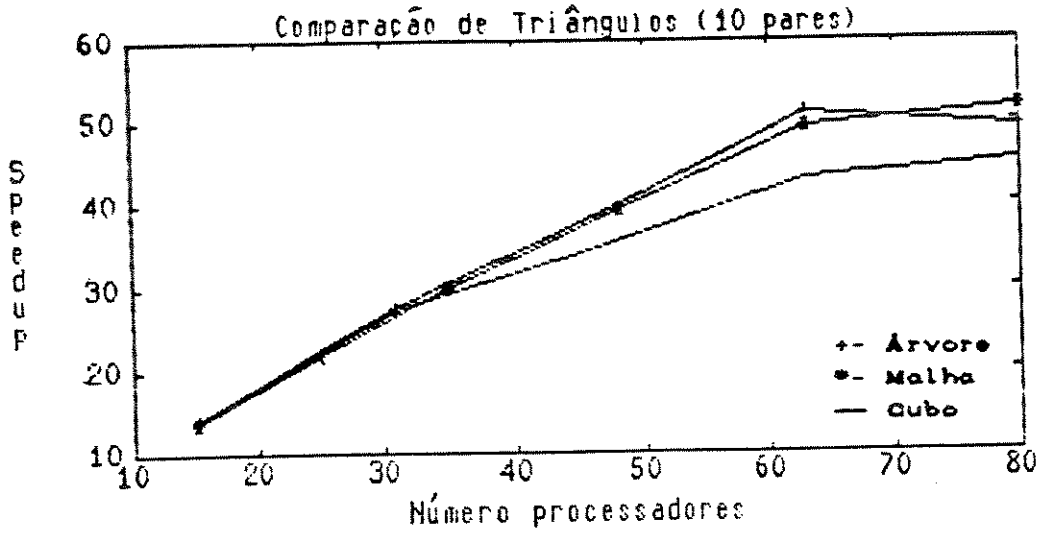


fig. 6.4a. "Speedup" do processo de comparação de triângulos em lotes de 10 tarefas.

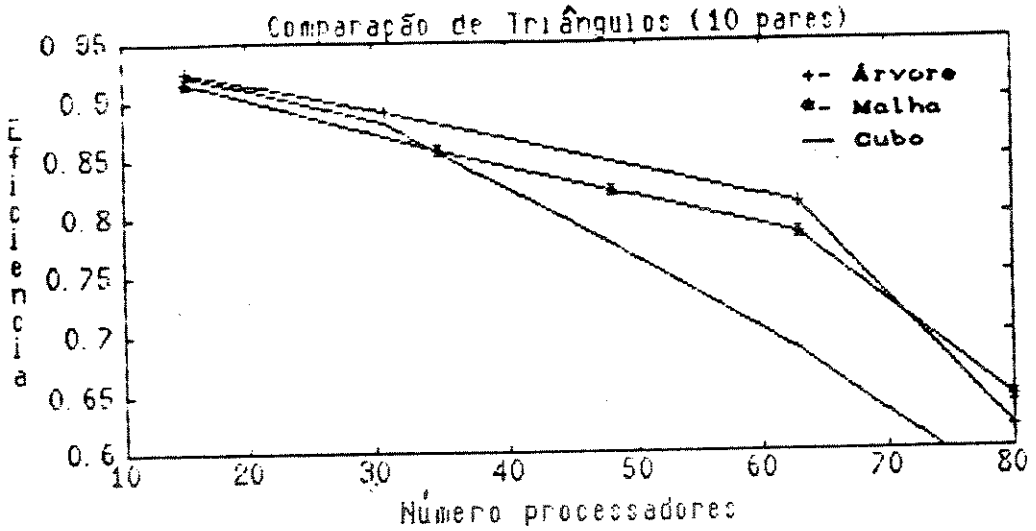


fig. 6.4b. Eficiência do processo de comparação de triângulos em lotes de 10 tarefas.

A figura 6.4 mostra os gráficos do "speedup" e a eficiência dos dados da simulação do processo de comparação de triangulos, considerando grupos de 10 tarefas. O melhor "speedup" é apresentado pela topologia malha, trabalhando com uma treliça de 9 filas e 9 colunas. A topologia árvore apresenta um comportamento aproximado com menos processadores e melhor eficiência, quando resulta o melhor "speedup". Se for aumentado o número de processadores, a eficiência tem uma queda significativa.

6.6 ESTUDO DE COMPORTAMENTO DA PROPOSTA PARA A IMPLEMENTAÇÃO PARALELA DO ALGORITMO "L.S.E".

Na seção 2.4 deste trabalho, foi apresentada uma nova proposta de implementação paralela. Essa proposta consiste em executar os processos de remoção e decomposição num mesmo processador, e iniciar o processo de comparação de triângulos apenas quando existir pelo menos um par de triângulos para serem comparados.

Nesta seção será apresentada a execução dos processos de remoção das faces visíveis e de decomposição em triângulos das faces visíveis e, posteriormente, o processo em forma integral, que foi denominado, na seção 2.4, "Bloco inicial" do algoritmo L.S.E.. Nesse caso o "bloco inicial" consistirá do processo de remoção-decomposição e do processo comparação de triângulos, em lotes de 10 tarefas a serem executadas por processador.

6.6.1 PROCESSO DE REMOÇÃO-DECOMPOSIÇÃO DAS FACES VISÍVEIS.

Considerando os mesmo dados extraídos da tampa da chaleira, para os processos de remoção e decomposição, respectivamente, foi realizada uma distribuição aleatória para combinar as tarefas de remoção e decomposição. Também, de forma aleatória, foi feita a eliminação das faces consideradas não visíveis.

A tabela 6.8 mostra os dados correspondentes à obtenção do melhor tempo de execução das topologias consideradas.

Topologias / Medidas	Anel	Árvore	Hipercubo	Malha
Nº processadores	92	92	63	80
T. execução (s)	0.0620069	0.05087	0.08081	0.050985
"Speedup"	19.88805	21.0792	14.8991	21.055
Eficiência	0.2101419	0.2291227	0.2954	0.26919
T. comunica. (s)	1.66168	0.278821	0.15962	0.4875
T. interrup. (s)	0.199229	0.054084	0.01718	0.08149
T. contenção (s)	0.001485	0.0077209	0.047042	0.0104946

Tabela 6.8. Distribuição dos valores ótimos do processo de remoção-decomposição.

Analisando a tabela 6.8, verifica-se que o melhor comportamento é oferecido pela topologia malha ("mesh"), já que com um número menor de processadores, consegue um "speedup" próximo ao da topologia árvore.

A figura 6.5 mostra os gráficos obtidos nesse processo conjunto. Nota-se que as curvas mostram um comportamento muito similar ao processo de decomposição da figura 6.2. A diferença está no número de processadores necessários para obter maior "speedup", com uma eficiência menor.

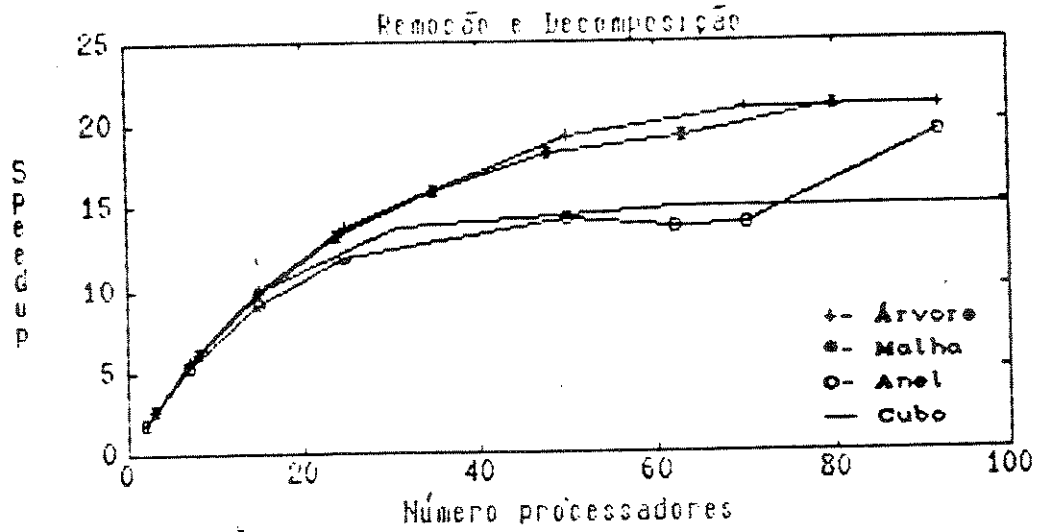


fig. 6.5a. "Speedup" do processo de remoção-decomposição.

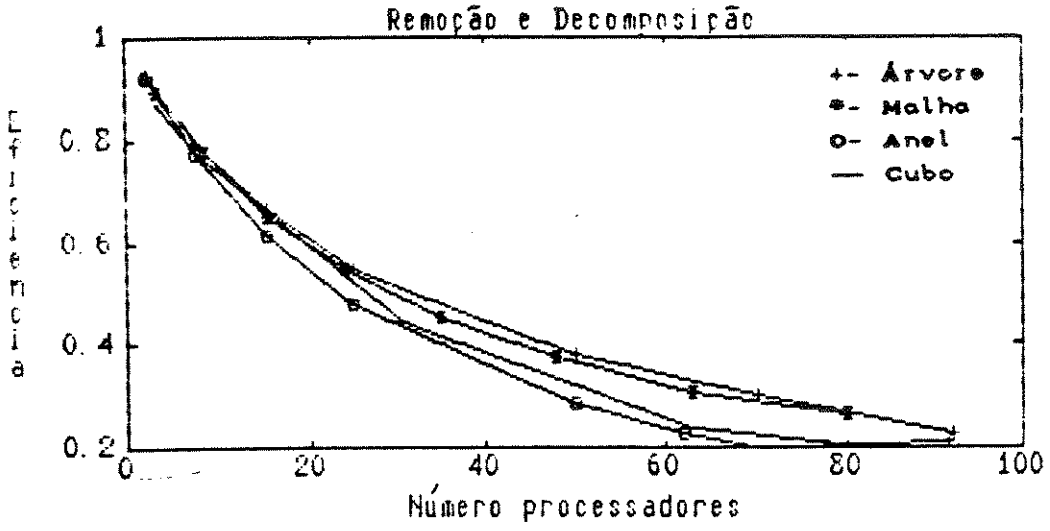


fig. 6.5b. Eficiência do processo de remoção-decomposição.

6.6.2 ESTUDO DO COMPORTAMENTO DO "BLOCO INICIAL".

Para realizar este estudo, foi criado um arquivo de processos que representa o particionamento das tarefas que compõem os módulos do "bloco inicial". Esse arquivo possui, inicialmente, as tarefas do processo de "remoção-decomposição", cujo comportamento foi apresentado na seção 6.5.1, e o particionamento do módulo de comparação de triângulos, onde são considerados grupos de 10 tarefas de comparação a serem distribuídas em cada processador disponível.

Neste estudo foram consideradas as topologias que apresentaram melhor desempenho, segundo o estudo feito nas seções anteriores deste capítulo.

Na obtenção dos dados, foi realizada uma varredura de comportamento nas topologias, considerando de 256 a 2 processadores interconectados. As topologias empregadas foram a Hipercúbica, Árvore e Malha ("mesh"). Os dados que mostram o melhor comportamento para cada topologia são apresentados na tabela 6.9.

Topologias / Medidas	Malha	Árvore	Hipercubo
Nº processadores	99	81	255
T. execução (s)	0.8769	0.977605	0.498618
"Speedup"	54.8685	54.67905	47.07966
Eficiência	0.554	0.689488	0.1846
T. comunica. (s)	4.06474	2.2605	1.422375
T. interrup. (s)	0.90569	0.51724	0.228019
T. contenção (s)	0.07446	0.065526	0.14552

Tabela 6.9. Distribuição dos valores ótimos do processo "bloco inicial".

A topologia árvore em conjunto possui melhor desempenho do que as outras topologias em termos de "speedup" e eficiência. Na execução dos módulos de forma separada, pode ser observado que a topologia em árvore também mostra bom desempenho com poucos processadores, conseguindo um "speedup" muito próximo do maior apresentado pela topologia malha.

Na figura 6.6 são mostrados os gráficos resultantes da simulação do bloco inicial, composto pelos módulos de "remoção decomposição" e "comparação em grupo de a 10".

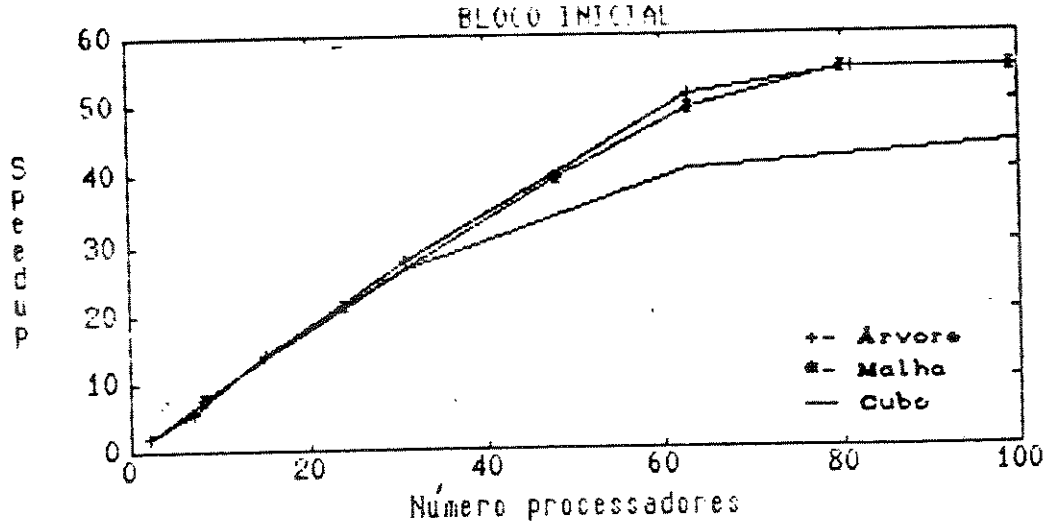


fig. 6.6a. "Speedup" do processo denominado "bloco inicial".

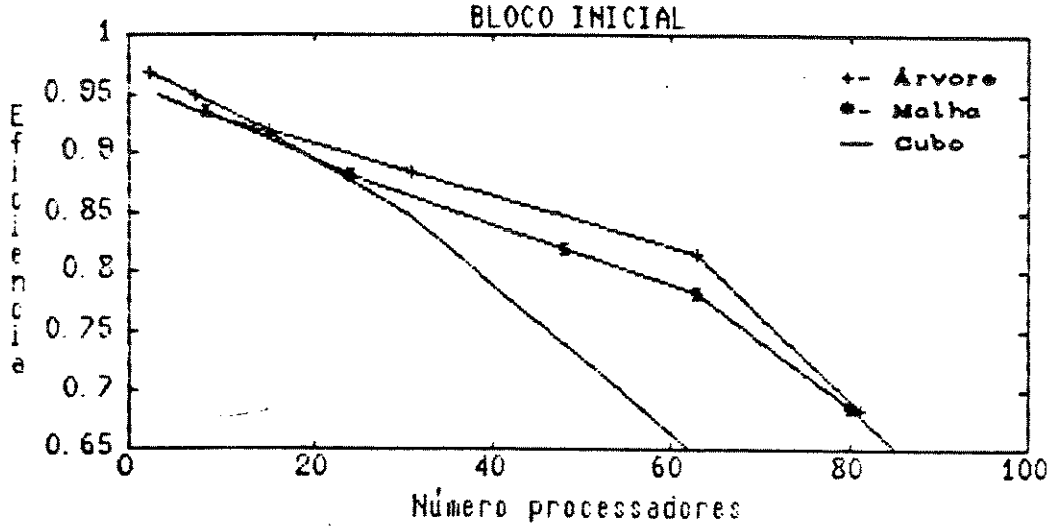


fig. 6.6b. Eficiência do processo "bloco inicial".

Nota-se a similaridade existente entre os gráficos apresentados nas figuras 6.4 e 6.6, podendo verificar-se que existe um pequeno aumento do "speedup" e menor eficiência, resultante do aumento das tarefas a distribuir. Também, observam-se os pontos máximos do "speedup" do gráfico da figura 6.6 que é próximo de 100 processadores. Para um número maior de processadores, a eficiência mostra uma queda, o que indica que não existirá um "speedup" significativamente melhor, considerando-se mais processadores disponíveis.

6.7 CONCLUSÕES DAS SIMULAÇÕES REALIZADAS.

Do estudo realizado, estima-se que o processo de comparação de triângulos, do algoritmo "L.S.E.", empregando a técnica do pintor, é a etapa que consome o maior tempo de processamento, seja sua execução sequencial ou paralela. Por tal razão, os melhoramentos e a estratégia de particionamento e distribuição desse módulo, são de vital importância no processamento do algoritmo, no menor tempo possível.

Introduzindo mudanças no particionamento dos módulos de remoção e decomposição, apenas se consegue uma redução de tempo, que é ínfimo se comparado com o processo de comparação de

triângulos.

Com o objetivo de melhorar o comportamento do processo de comparação, foram agrupados conjuntos de processos a serem executados num processador. Para isso, deve-se considerar que os dados de envio e retorno aumentam de forma linear, dependendo da quantidade de processos agrupados. Esse aspecto faz com que exista um agrupamento ótimo, como foi demonstrado na seção 6.5.2. A consideração de um número maior de tarefas por processador, resulta numa diminuição do desempenho.

No estudo do comportamento dos módulos iniciais do algoritmo L.S.E., observa-se que o comportamento ideal difere significativamente do comportamento real. O "speedup" real para este conjunto de dados é de 51 a 55 vezes, correspondendo às topologias árvore e malha ("mesh"), respectivamente. Esse valor corresponde a 15% do "speedup" ideal, indicado no trabalho de Regonessi [REGO,1990].

Neste estudo o "speedup" real máximo foi obtido para 100 processadores.

Com o objetivo de que o "bloco inicial" seja executado em tempo real, isto é, tempo de execução menor ou igual a 1/30 segundos para televisão ou 1/24 segundos para cinema, foram realizados os testes, onde cada tick foi de 1.715, 0.1715, 0.01715 e 0.001715 ms. para executar o processo de comparação de triângulos, considerando blocos de 10 tarefas de comparação por processador e mantendo constante o custo de comunicação.

A topologia malha apresentou o menor tempo de execução, porém, longe do tempo real (0.153ms), com uma treliça de 4x4 e um baixo "speedup" (0.122), para um tick de 0.001715ms.

Utilizando o mesmo principio de "dividir e conquistar", foram levantadas as seguintes hipoteses das 5995 tarefas de comparação de triângulos (110 triângulos):

a) Dividir as 5995 tarefas em dois grupos de 3000 e 2995 tarefas, e esses, na sua vez dividi-los em blocos de 20 tarefas de comparação a serem distribuídas por processador. Novamente a topologia malha apresenta o menor tempo de execução obtido de 0.04093 segundos, com um tick de relógio de 1.715 μs. Essa divisão seria apropriada para tempo real de cinema, mas não para televisão.

b) Dividir as 5995 tarefas em três grupos de 2000, e isses em blocos de 20 tarefas. Os dados obtidos são os da tabela 6.10.

tick = 17.15 μs

Topologias / Medidas	Malha	Árvore	Hipercubo
Nºprocessadores	15	31	63
T. execução(s)	0.0921664	0.08284	0.06257
"Speedup"	3.49649	3.4774	1.795

tick = 1.715 μs

Topologias / Medidas	Malha	Árvore	Hipercubo
Nºprocessadores	8	15	15
T. execução (s)	0.028404	0.091199	0.062
"Speedup"	0.3959	0.96055	0.18095

Tabela 6.10. Processo de comparação de triângulos considerando o processo em grupos de 2000 tarefas.

Pode-se observar na tabela 6.10, que a topologia malha com uma treliça de 3x3, apresenta o menor tempo de execução. Nos valores apresentados para a operação com um tick 1.715 μs e 17.15 μs, pode-se notar que as respostas de tempo de execução são próximas, no entanto, o "speedup" difere significativamente, isso permite ver claramente que o gasto de tempo de comunicação passa a ter importância na execução das tarefas de curta duração. Esse fato invalida o estudo, porque ao diminuir o tempo de relógio, a execução sequencial será menor, e a não redução do tempo de

comunicação, resultará em que a execução sequencial pode ser melhor do que o processamento paralelo.

Para que o processamento paralelo seja de interesse, sua utilização nesse tipo de aplicação, deverá apresentar taxas de transferências cada vez mais rápidas, otimizando a forma de comunicação das mensagens, minimizando as contenções e as interrupções dos processos, permitindo uma trajetória expedita da transmisão de mensagens entre processadores.

CAPÍTULO 7

CONCLUSÕES

7.1 DA CONSTRUÇÃO DO SISTEMA ORIGINAL.

Antes de fazer a implementação do "sistema original", utilizando a placa IMS B008, trabalhou-se com a placa IMS B004 com um transputer só, existente no labóratorio LCA. Essa mudança obrigou a considerar toda a lógica que se refere à alocação dos processos, mapeamento e configuração da topologia.

Em um transputer só, a implementação pode ter toda a lógica empregada na confeção de um programa em linguagem de alto nível (Pascal, C, Fortram, etc.) , como : uso de variáveis e constantes globais, emprego de procedimento com variáveis globais, pouco ou nada de manejo de canais, etc. Contudo, na hora de fazer a mesma implementação numa placa com vários transputers, surgiram problemas, porque os procedimentos a serem distribuidos na rede deviam ser: independentes, sem variáveis ou constantes globais, sem protocolos globais e sua comunicação externa, feita por canais de entradas e saídas.

Se essas afirmações fossem consideradas do início, o passo para implementar programas em vários transputers seria direto.

Como principais resultados iniciais, considerando os aspectos de implementação, observaram-se:

1) Com relação à implementação, o MMS2 não mapeia a conexão do link 3 do TRAM 0. Essa situação obrigou à realização de configurações onde existiam um tranputer "host" e o transputer "root" (figura 4.21), criando um engarrafamento e desperdiçando um transputer. Mediante o emprego dos programas testes fornecidos pelo TDS2 e utilitários "check ! mtest "

detetou-se a existência da conexão, refazendo o mapeamento (figura 4.22).

2) Detetou-se que as facilidades das sentenças de alto nível seletivas (ALT) não aproveitam as caraterísticas do processamento paralelo, oferecidas no transputer, porque a sentença atua quando existe uma entrada pronta, executando-se sem deixar progredir outra entrada até ficar liberada novamente. Dessa maneira as comunicações tornam-se quase sequenciais, além de atribuir prioridades arbitrárias. Shallow [SHALL,1990] e Torngrem [TORNG,1990] fazem um estudo e apresentam formas de minimizar esse efeito.

   A melhor maneira encontrada para não utilizar a sentença seletiva ALT, foi criar processos concorrentes independentes de entradas e saídas, porque se aproveitam na execução paralela das interfaces de comunicação.

3) Outro aspecto que foi comprovado foi a facilidade de uso do protocolo variante. Porém, sua utilização degrada o desempenho da aplicação, comparando-se ao desempenho de um protocolo de comunicação de tamanho fixo. Esse fato é resultado da operação dos canais de comunicação da linguagem OCCAM2.

7.2 DA CONSTRUÇÃO DO SIMULADOR.

Antes da versão final do simulador apresentado neste trabalho, tentou-se aproveitar as caraterísticas que oferece o transputers, para realizar simulação dos processos por meio do Timer (temporizadores), para representar a execução do processo simulado. Foi descartado pela dificuldade em registrar de forma real os tempos de contenção, interrupção e de predizer a sequência de processamento. Somente foi possível resgatar a lógica para

realizar o "sistema original" e quantificar os tempos gastos em comunicação, mas o processo simulado obrigava a esperar a finalização da hibernação do Timer.

A implementação do "sistema original" permitiu ir realizando otimizações na operação dos processos envolvidos resultando um simulador mais confiável, que representava a execução real do "sistema original", para um número maior de processadores.

No desenvolvimento do "sistema original" e aplicação de conhecimento de simulação de eventos discretos, foi possível obter uma ferramenta que permite quantizar o tempo gasto na comunicação dos dados, tempo de contenção e registrar o tempo de interrupção dos processos que operam com prioridade baixa. Isso faz com que a ferramenta desenvolvida represente um comportamento da aplicação mais próximo à realidade.

A escolha da realização de um simulador sincronizado por eventos, vem do fato que esta é a forma mais natural de execução dos processos mediante a utilização do transputer e a linguagem OCCAM2.

7.3 DO ESTUDO DA APLICAÇÃO.

Do estudo realizado da aplicação pode-se dizer que as topologias que apresentam melhor desempenho na execução dos diferentes módulos da aplicação, correspondem às topologias árvore e malha. A topologia árvore apresenta a vantagem de que com poucos processadores consegue um comportamento próximo ao apresentado pela topologia malha.

Infelizmente, ante a exigência do processamento em tempo real da aplicação, pode-se concluir que a utilização de máquinas MIMD fracamente acopladas, com o equipamento utilizado e forma de operação, não é possível sua execução em tempo real, pela alta taxa de transferência de dados do módulo de comparação de

triângulos. Este fato indica que o custo de comunicação toma importância ante a taxa de transferência de dados. O processo de comparação de triângulos consome 93% do tempo de processamento do "bloco inicial".

Da análise e do estudo do comportamento da aplicação, pode-se afirmar o seguinte:

a) Deve ser feita a otimização da distribuição das tarefas no processo de comparação de triângulos, considerando a eliminação da comparação dos triângulos de uma mesma face.
b) É possível realizar o processo de comparação, mediante a distribuição em pacotes de triângulos de uma face a serem comparados com outra face.
c) Pode-se avaliar a possibilidade de utilizar máquinas MIMD fortemente acopladas, hibridas ou máquinas fracamente acopladas com mecanismos de comunicação otimizados.

7.4 CONCLUSÕES FINAIS.

Este trabalho contribui com um conjunto de ferramentas que permite representar o comportamento de uma aplicação na forma mais próxima à realidade, já que considera os custos reais na execução do algoritmo paralelo. Além disso pode servir para realizar a otimização dos processos envolvidos no processamento, como: otimização das rotinas de comunicação, eliminação do excesso de interrupções, minimização das contenções, etc.

Outro aspecto importante a destacar é o estudo e os programas desenvolvidos na utilização de uma rede de transputer, onde foi feito o mapeamento das topologias de maneira a permitir a ampliação do número de processadores a utilizar em forma física. Isso servirá para desenvolver trabalhos futuros utilizando transputers.

triângulos. Este fato indica que o custo de comunicação toma importância ante a taxa de transferência de dados. O processo de comparação de triângulos consome 93% do tempo de processamento do "bloco inicial".

Da análise e do estudo do comportamento da aplicação, pode-se afirmar o seguinte:

a) Deve ser feita a otimização da distribuição das tarefas no processo de comparação de triângulos, considerando a eliminação da comparação dos triângulos de uma mesma face.
b) É possível realizar o processo de comparação, mediante a distribuição em pacotes de triângulos de uma face a serem comparados com outra face.
c) Pode-se avaliar a possibilidade de utilizar máquinas MIMD fortemente acopladas, hibridas ou máquinas fracamente acopladas com mecanismos de comunicação otimizados.

7.4 CONCLUSÕES FINAIS.

Este trabalho contribui com um conjunto de ferramentas que permite representar o comportamento de uma aplicação na forma mais próxima à realidade, já que considera os custos reais na execução do algoritmo paralelo. Além disso pode servir para realizar a otimização dos processos envolvidos no processamento, como: otimização das rotinas de comunicação, eliminação do excesso de interrupções, minimização das contenções, etc.

Outro aspecto importante a destacar é o estudo e os programas desenvolvidos na utilização de uma rede de transputer, onde foi feito o mapeamento das topologias de maneira a permitir a ampliação do número de processadores a utilizar em forma física. Isso servirá para desenvolver trabalhos futuros utilizando transputers.

O estudo realizado em relação à aplicação, ao planejamento do "sistema original" e, finalmente, ao simulador, é perfeitamente aplicável para desenvolver um trabalho semelhante que considere as máquinas MIMD fortemente acoplada.

Este seria um aspecto de interesse, visto que o algoritmo L.S.E. apresenta uma alta taxa de transferência de dados nos processos de comparação de triângulos.

CAPÍTULO 8

REFERENCIAS

[ANGER,1990] Anger, F. Hwang, J. e Chow, Y. "Scheduling with sufficient lossely coupled processors", Journal of Parallel and Distributed Computing, 9, pp. 87-92.

[BAGCHI,1989] Bagchi, K. Olsen, O. Christensen, A. e Sorensen, L. "Simulation and Design of Message Routing Systems for Networks of Transputers", Annual Simulation Symmposium, March.

[BURNS,1988] Burns, A.""Programming in Occam2",Addison Wesley, England.

[DUNCA,1990] Duncan, R. "A survey of parallel computer architectures", IEEE Computer, February, pp. 5-16.

[EAGER,1989] Eager, D., Zahoran, J. and Lazowka, E. "Speedup Versus Efficiency in parallel systems", IEEE Transaction on Computers Vol 38, 3, pp. 408-423.

[ELREW,1990] El-Rewini, H and Lewis, T "Scheduling parallel program tasks onto arbitrary target machines", Journal of Parallel and Distributed Computing, 9, pp. 138-153.

[FLYNN,1972] Flynn, M. "Some computer organization and their efetiveness", IEEE Transactions on Computer, Vol. C-21, 9, pp. 948-960.

[FUJI,1991] Fujimoto, R. "Parallel Discrete event Simulation", Communications Of the ACM, Vol 33, 10, October.

[GAUDI,1989] Gaudiot, J. L. and Lee, H. T. "Occamflow: Amethodology for programming multiprocessor systems", Journal of Parallel and Distributed Computing, 7, pp. 96-124.

[GONZA,1977] Gonzalez, M. "Deterministic Processor Scheduling", ACM Computing Survey Vol. 9, 3, pp. 173-204.

[HARRI,1986] Harrington, S. "Computers Graphics: A programming approach", McGraw-Hill, Singapore, 2ª Edição.

[HAYES,1989] Hayes, J. and Mudge, T. "Hipercube Supercomputers". Proceeding of IEEE Vol 77, 12, pp. 1829-1841.

[HERRA,1988] Herrada, E., Beivide, R. y Navarro, J. "Sistemas Multiprocessadores de paso de mensajes", Mundo Electrónico , 181, pp. 47-56.

[HERRA,1990] Herrada, E. Arrabarrena, A. Balcazar, J. e Izen, O. "An optimal topology for multicomputers system based on a mesh of transputers", OCCAM user group newsletter, January, pp. 31-39.

[HWANG,1984] Hwang, K. and Briggs,F.A. "Computer arquitecture and parallel processing", Mc Graw-Hill, New York.

[IB008,1988] Inmos B008. "IMS B008 User guide and reference manual ", INMOS Ltda. England.

[IMMS,1989] Inmos MMS. " S708 User guide ", INMOS Ltda. England.

[IORM,1988] Inmos ORM. "Occam 2, reference manual", Prentice Hall, Cambridge.

[ITD,1988] Inmos TD. "Transputer Databook", Bath Press Ltd. Bath.

[ITDS,1988] Inmos TDS. "Transputers development System ", Prentice Hall, London.

[ITIS,1988] Inmos TIS. "Transputer instruction set; A compiler writers guide ", Prentice Hall, London.

[ITRM,1988] Inmos TRM. "Transputer reference manual", Prentice Hall, London.

[JONES,1988] Jones, G. e Goldsmith, M. "Programming in OCCAM2", Prentice Hall, London.

[KARP,1990] Karp, A. and Flatt, H. "Measuring parallel processor performance", Communications of the ACM, Vol. 33, 5, pp. 539-543.

[KIRNER,1991] Kirner, C. "Arquiteturas de Sistemas Avançados de Computação", Anais da EPUSP/IEEE em sistemas de computação de alto desempenho, Março.

[LAN,1990] Lan, Y., Esfahanian, A. and Ni, L. "Multicast in Hipercube Multiprocessor", Journal of Parallel and Distributed Computing, pp. 30-41.

[LEUNG,1989] Leung, J. and Young, G. "Minimizing schedule length subject to minimum flow time ", SIAM Journal Computing, Vol 18, 2, April, pp. 314-326.

[MISRA,1986] Misra, J. "Distributed discrete-event simulation", ACM Computing Surveys, Vol. 8, 1, pp. 39-65.

[PADMA,1990] Padmanabhan, K. "Cube structures for multiprocessors", Communications of the ACM Vol 33, 1, pp. 43-52.

[PETER,1981] Peterson, J. "Petri Net theory and modeling systems", Prentice Hall Inc. N. York.

[POUNT,1987] Pountain, D. and May, D. "A tutorial introduction to Occam programming", INMOS, London.

[QUINN,1988] Quinn, M. J. "Designing efficient algorithms for parallel computers", Mc Graw-Hill, Singapore.

[REGO,1990] Regonessi, M. "Relatorio de avanço de Tese".

[RETTB,1990] Rettberg, R., Crowther, W. Carvey, P. e Tomlinson, R. "The Monarch Parallel Processor Hardware Design", IEEE Computer, April, pp.

[SAAD,1988] Saad, Y. and Schultz, M. "Topological properties of Hypercubes", IEEE Transactions on Computers Vol 37, 7, pp. 867-872.

[SAAD,1989] Saad, Y. and Schultz, M. "Data communication in Hypercubes", Journal of Parallel and Distributed Computing,6, pp. 115-135.

[SHALL,1990] Shallow, P. "Really efficient multiple buffering in occam and efficient fair ALT", Occam user group, newsletter, january, pp. 42-49.

[SYMON,1980] Symons, F. Billington, J. e Kirton, P. "Representation of Data comunications processes", Australian Telecomunication Research, Vol. 14, 1.

[TORNG,1990] Törngren, M. "Transputer and occam based control system in electronic control of machines", Occam user group, newsletter, january, pp. 66-70.

ANEXO A

ALGORITMOS DE COMUNICAÇÃO.

A.1 INTRODUÇÃO.

Com a finalidade de realizar a implementação do "sistema original" e simulador, foram desenvolvidos e codificados algoritmos de comunicação que consideram o envio de mensagens de um processador a qualquer outro na topologia. Considerando como base os trabalhos analisados [HERRA,1990], [GAUDI,1989], [HWANG,1984], [HAYES,1989] e [SAAD,1989]. As topologias mais abordadas na proposição de algoritmos de comunicação são as topologias hipercúbicas e malha. Pela forma da análise e implementação dos algoritmos resulta quase direta a extensão para as topologias árvore, anel e midimew.

A.2 TOPOLOGIA ANEL.

O algoritmo de comunicação desenvolvido para essa topologia, considera que os processadores estão numerados em sentido horário e a comunicação será em essa direção para os processadores menores ou iguais a $\lceil n/2 \rceil$, o retorno será em sentido contrário. Em forma inversa para os processadores maiores que $\lceil n/2 \rceil$, a fim de realizar o mínimo de deslocamento das mensagens nos processadores intermediários que participam.

```
PROC anel ( VAL INT  an.fonte,
            [2] CHAN OF mensagemO inp,
            [2] CHAN OF mensagemO out)
  INT an.m, signo, media :
  SEQ
    media := total.nos / 2
    ALT i = 0 FOR 2
      inp[i] ? mensaje
        SEQ
          an.destino:=mensaje[0]
          an.m := an.destino - an.fonte
          IF
            an.m <> 0
              SEQ
                --<<<  vai para outro vizinho
                signo := (-1)
                IF
                  an.m > 0
                    signo := 1
                  TRUE
                    an.m := an.m * (-1)
                IF
                  an.m > media
                    signo := signo * (-1)
                  TRUE
                    SKIP
                --<<<  se envia
                IF
                  signo = 1
                    out [2] ! mensaje
                  signo = (-1)
                    out [1] ! mensaje
                  TRUE
                    SKIP
                -->>>
            an.m = 0
              out [0] ! mensaje
            TRUE
              SKIP
:
```

fig. A.1, algoritmo de comunicação da topologia anel.

A.3 TOPOLOGIA ÁRVORE.

Na confeção do algoritmo de comunicação, foram consideradas as definições básicas da estrutura de dados árvore binário, tais como: determinação do processador pai e dos processadores filhos e profundidade. A figura A.2 mostra o algoritmo de comunicação implementado em OCCAM2, utilizando nos sistemas simulador e "original".

```
PROC arvore (VAL INT a.fonte,[4]CHAN OF mensagem0 inp,
             [4]CHAN OF mensagem0 out)
  INT a.fonte, a.destino, na.destino, na.fonte, dif.ni, aux.ni,
      ni.a1, ni.a2, restim, m.a :
  BOOL ar.final :
  PROC loga.2(INT resultado, VAL INT x )
    REAL32 r32.aux :
    SEQ
      r32.aux:=LOGB((REAL32 ROUND (x+1)))
      resultado:=(INT TRUNC (r32.aux)) + 1
  :
  SEQ
    ALT i = 0 FOR 4
      inp[i] ? mensaje
        SEQ
          a.destino := mensaje[0]
          m.a:=a.destino - a.fonte
          IF
            m.a <> 0
              SEQ
                loga.2(na.destino,a.destino)
                loga.2(na.fonte, a.fonte)
                dif.ni:= na.destino - na.fonte
                IF
                  dif.ni <= 0
                    ar.final:=TRUE
                  dif.ni > 0
                    SEQ
                      aux.ni:=na.destino
                      WHILE na.fonte <> aux.ni
                        SEQ
                          ni.a1:=((a.destino+1)/2) - 1
                          aux.ni:=aux.ni -1
                          ni.a2:=a.destino
                          a.destino:=ni.a1
                      IF
                        ni.a1 <> ar.fonte
                          ar.final:=TRUE
                        ni.a1 = a.fonte
                          SEQ
                            restim:= ni.a2 REM 2
                            IF
                              restim = 0
                                out [3] ! mensaje
                              restim <> 0
                                out [2] ! mensaje
                IF
                  ar.final
                    out [1] ! mensaje
            m.a = 0
              out [0] ! mensaje
            TRUE
              SKIP
  :
-->>>
```

fig. A.2, algoritmo de comunicação da topologia árvore.

A.4 TOPOLOGIA HIPERCÚBICA.

A figura A.3 mostra o algoritmo de comunicação numa topologia Hipercúbica, que corresponde à codificação do texto estruturado apresentado em Hayes [HAYES,1987].

```
PROC  n.cubo (VAL INT n.fonte,
              [5] CHAN OF mensagemO inp,
              [5] CHAN OF mensagemO out)
  IN T kei, n.ruta, k.esimo, kdara, m.c :

SEQ
  AL T i = 0 FOR 5
     inp [i] ? mensaje
       SEQ
          n.destino:= mensaje[0]
          m.c:=n.destino-n.emisor
          IF
            m.c <> 0
              SEQ
                n.ruta:=n.emisor >< n.destino
                kei := 0
                k.esimo := 1
                flag.final := TRUE
                WHILE (kei < n.nos) AND flag.final
                  SEQ
                    kdara := n.ruta /\ k.esimo
                    IF
                      kdara = k.esimo
                        SEQ
                          flag.final := FALSE
                          out [kei + 1] ! mensaje
                    TRUE
                      SKIP
                kei:= kei + 1
                k.esimo := k.esimo * 2
        m.c = 0
          SEQ
            out [0] ! mensaje
        TRUE
          SKIP
:
-->> >
```

fig. A.3, Algoritmo de comunicação na topologia Hipercúbica.

A.5 TOPOLOGIA MESH.

A figura A.4 mostra o algoritmo de comunicação da malha quadrática, desenvolvido a partir da descrição feita no Hwang e Brigg [HWANG,1984], que estabelece as regras para a numeração dos processadores.

1. As filas e colunas serão numeradas de 0..(r-1); onde $r=\sqrt{N}$.
2. Os nós processadores serão rotulados da seguinte forma:
   - (i-r) MOD N ; Norte
   - (i+1) MOD N ; Leste
   - (i-1) MOD N ; Oeste
   - (i+r) MOD N ; Sul.
3. As filas serão determinadas por $\lfloor i/r \rfloor$.
4. As colunas serão determinadas por (i MOD r).
5. Quando a coluna seja maior do que $\lfloor n/2 \rfloor$, a comunicação será através das colunas.

A.6 TOPOLOGIA MESH COM WRAP-ARROUND.

A figura A.5 mostra o algoritmo implementado na utilização da malha 2D com "Wrap-arround". Esse algoritmo foi extraído do trabalho de Herrada [HERRA,1990], que considera as seguintes regras para a numeração dos processadores e critérios de operação.

O rotulado é feito da seguinte maneira :

1. Calcular os parâmetros $b = \lceil \sqrt{(n/2)} \rceil$, $r = \lceil n/b \rceil b - N$ $h = b + r$, $v = \lceil n/b \rceil - r$.
2. Planejar a malha retangular de largo h e alto v.
3. Se $r \neq 0$ e $v \neq b-1$, então, fazer uma nova malha de largura r e alto v-b+1.
4. Estabeleça os links "wrap-arround", conecte cada nó (i,0) ao topo da coluna (i+r)MODh, e conecte cada nó (h-1,j) à fila (j+b-1)MODv.
5. Cada nó é rotulado da seguinte forma na malha :

(i+b)	MOD	N	Norte
(i-b)	MOD	N	Sul
(i+b-1)	MOD	N	Leste
(i-b+1)	MOD	N	Oeste

```
PROC malha.midimev ( VAL INT mm.fonte,
                     [5] CHAN OF mensagem0 inp,
                     [5] CHAN OF mensagem0 out)
  INT mm, remainder, cond, signo, bm :
  SEQ
    ALT i = 0 FOR 5
      inp [i] ? mensaje
        SEQ
          ms.destino:=mensaje[0]
          mm := mm.destino - mm.fonte
          IF
            mm <> 0
              SEQ
                aux.rm := (REAL32 ROUND (total.nos/2))
                r.bm := SQRT(aux.rm) + 0.998 (REAL32)
                bm := (INT TRUNC (r.bm))
                signo := (-1)
                IF
                  mm > (total.nos/2)
                    mm := total.nos - mm
                  mm > 0
                    signo := 1
                  mm > ((total.nos/2 ) * (-1))
                    mm := mm * (-1)
                  mm > (total.nos * (-1))
                    SEQ
                      signo := 1
                      mm := total.nos + mm
                remainder := mm REM bm
                cond := (mm/bm) - (2*remainder)
                IF
                  remainder > 0
                    IF
                      ((bm - cond)*signo) > 0
                        out[2] ! mensaje  --(mm.fonte + bm) - 1
                      ((bm - cond) * signo) <= 0
                        out[4] ! mensaje  --(mm.fonte - bm) + 1
                  remainder = 0
                    IF
                      signo = 1
                        out[1] ! mensaje  --mm.fonte + bm
                      signo = (-1)
                        out[3] ! mensaje  --mm.fonte - bm
            mm = 0
              out [0] ! mensaje  -- chegou
            -->>>
  :
```

fig. A.5, midimev -- malha com wrap-arround --