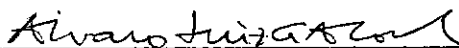


**UM ALGORITMO PARALELO PARA A SOLUÇÃO DIRETA DE  
SISTEMAS DE EQUAÇÕES DO MEF EM REDES DE TRANSPUTERS.**

**Walcyr Duarte Nascimento**

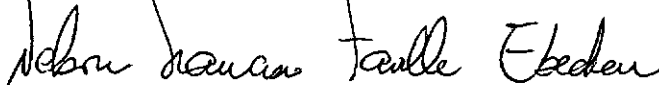
**TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS  
PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE  
FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS  
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM  
ENGENHARIA CIVIL.**

**Aprovada por:**

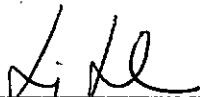


**Prof. Alvaro L. G. A. Coutinho, D.Sc.**

**( Presidente )**



**Prof. Néelson F. F. Ebecken, D.Sc.**



**Prof. Luiz Landau, D.Sc.**



**Prof. Elson Magalhães Toledo, D.Sc.**

**RIO DE JANEIRO, RJ - BRASIL**

**DEZEMBRO DE 1992**

Para Wanderley "in memorian",

Edir, Pedro e Alda.

## AGRADECIMENTOS

Ao professor Alvaro Luiz G. A. Coutinho pela irretocável orientação, pela dedicação e esforço.

Aos Professores do Programa de Engenharia Civil da COPPE, que colaboraram no Curso, através do ensino de disciplinas, por sua excepcional capacitação e motivação.

Aos professores Elson Magalhães Toledo e Júlio Portela e, ao amigo Eduardo Lúcio pela ajuda, incentivo e apoio.

Aos amigos José Ernesto e Renan pelo companheirismo e motivação durante os créditos e, ao colega Mota pelas sugestões e ajuda com o sistema Transputer.

A todos os colegas da Faculdade de Engenharia da UFJF, do LNCC, do Programa de Engenharia Civil da COPPE, e também aos funcionários do Programa, pela presteza e atendimento.

À CAPES pelo apoio financeiro sob a forma de Bolsa de Mestrado.

**NASCIMENTO, WALCYR DUARTE**

**Um algoritmo paralelo para a solução direta de sistemas de equações do MEF em Redes de Transputers**

**[Rio de Janeiro]**

**1992**

**XI, 110 p. 29,3 cm ( COPPE/UFRJ, M.Sc.,**

**Engenharia Civil, 1992 )**

**Tese - Universidade Federal do Rio de Janeiro, COPPE**

**1. Elementos Finitos**

**2. Computação Paralela**

**I. COPPE/UFRJ**

**II. Título (serie)**

Resumo da Tese apresentada a COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

UM ALGORITMO PARALELO PARA A SOLUÇÃO DIRETA DE  
SISTEMAS DE EQUAÇÕES DO MEF EM REDES DE TRANSPUTERS

Walcyr Duarte Nascimento

Dezembro de 1992

Orientador: Alvaro L. G. A. Coutinho.

Programa: Engenharia Civil

O objetivo deste trabalho é estudar a resolução pelo método direto de Gauss de sistemas de equações algébricas lineares oriundas do Método dos Elementos Finitos em computadores de arquitetura paralela. Utiliza-se um algoritmo baseado na decomposição Crout, com armazenamento tipo coluna ativa. Para a fase de fatoração é apresentado um algoritmo paralelo que trata da redução da coluna ativa. Este código foi implementado numa rede de *Transputers* e através de métodos de avaliação de desempenho analisou-se o comportamento do algoritmo para diversos problemas típicos, considerando diferentes configurações da rede.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirement for the degree of Master of Science (M.Sc.)

A PARALLEL ALGORITHM FOR THE DIRECT SOLUTION OF SYSTEM  
OF EQUATIONS OF THE FEM ON THE TRANSPUTERS NETWORKS

Walcyr Duarte Nascimento

December, 1992

Thesis Supervisor: Alvaro L. G. A. Coutinho.

Department: Civil Engineering.

In this work we study parallel direct solution techniques for finite element systems of equations in distributed memory machines. The standard Crout decomposition algorithm is employed and the coefficient matrix is stored in skyline format ( i.e., the active column reduction method ). The factorization phase is carried out in parallel and a special data structure was developed to handle the parallel decomposition. The resulting algorithm was implemented in a *Transputers* network, using the *PARALLEL FORTRAN* language. Several performance measurements were made in different benchmark problems considering various discretizations and network configurations. Good speed-up's were observed in all problems.

## ÍNDICE

<b><u>CAPITULO I</u></b>	<b><u>INTRODUÇÃO</u></b>	<b>1</b>
I.1	Razões do Processamento Paralelo	2
I.2	Desempenho dos Sistemas Paralelos	4
I.3	O Método dos Elementos Finitos e o Processamento Paralelo	6
I.4	Organização do Texto	11
<b><u>CAPITULO II</u></b>	<b><u>PROCESSAMENTO VETORIAL E PARALELO</u></b>	<b>14</b>
II.1	Computadores Vetoriais	14
II.2	Computadores Paralelos	17
II.2.1	Máquinas SIMD e MIMD	18
II.2.2	Memória Compartilhada e Distribuída	20
II.2.3	Esquemas de Ligações	22
<b><u>CAPITULO III</u></b>	<b><u>DESEMPENHO DOS SISTEMAS PARALELOS</u></b>	<b>34</b>
III.1	Velocidade Computacional	35
III.2	Medidas de Desempenho	37
III.3	Uma Implementação Hipotética	40
III.4	A Lei de Amdahl	42
III.5	A Importância da Granularidade	43
III.6	Perdas de Comunicação	46

<b><u>CAPITULO IV</u></b>	<b><u>AMBIENTE TRANSPUTER</u></b>	<b>49</b>
IV.1	Introdução	49
IV.2	O Transputer T800	50
IV.3	Programação do T800	51
<b><u>CAPITULO V</u></b>	<b><u>SOLUÇÃO DE GRANDES SISTEMAS DE EQUAÇÕES</u></b>	
	<b><u>DO MEF</u></b>	<b>54</b>
V.1	Generalidades	54
V.2	O Sistema Computacional Utilizado	55
V.3	Estudo para o Caso Sequencial	56
V.3.1	Preliminares	56
V.3.2	Métodos Diretos Baseados na Eliminação de Gauss	57
V.3.3	Formas de Armazenamento de K	61
V.3.4	Decomposição de Gauss no Método da Coluna Ativa	65
V.3.5	Redução do Vetor de Cargas	66
V.3.6	Retrosubstituição	66
V.3.7	Precisão na Eliminação de Gauss	67
V.3.8	Comentários Finais	68
V.4	Estudo para o Caso Paralelo	69
V.4.1	A Decomposição de Crout	69
V.4.2	Fase de Substituições	73
V.4.3	Estratégias de Paralelização	74
V.4.4	Implementação Computacional	75

<b><u>CAPITULO VI</u></b>	<b><u>APLICAÇÕES NUMÉRICAS</u></b>	<b>79</b>
VI.1	Generalidades	80
VI.1.1	Ligações entre os Processadores	80
VI.1.2	Cálculo do Ganho Absoluto (S) para as Diferentes Ligações	81
VI.1.3	Cálculo do Ganho Relativo (S') para as Diferentes Ligações	82
VI.1.4	Cálculo do Tempo de Comunicação e de Computação	82
VI.2	Resultados Numéricos	83
VI.2.1	Membrana	83
VI.2.2	Viga em Balanço	89
VI.2.3	Anel	92
VI.2.4	Conclusões	96
<b><u>CAPITULO VII</u></b>	<b><u>CONCLUSÕES FINAIS</u></b>	<b>97</b>
<b><u>APENDICE A - LISTAGEM DO PROGRAMA DE CONFIGURAÇÃO</u></b>		<b>100</b>
<b><u>APENDICE B - LISTAGEM DA ROTINA DE FATORAÇÃO</u></b>		<b>101</b>
<b><u>REFERENCIAS BIBLIOGRÁFICAS</u></b>		<b>104</b>

## C A P Í T U L O I

## INTRODUÇÃO

A pesquisa científica na Engenharia está tornando-se cada vez mais dependente do desenvolvimento e implementação de algoritmos paralelos eficientes em modernos computadores de alto desempenho [1,2 e 3]. Com isso, a análise de estruturas pelo MEF em computadores paralelos/vetoriais abre novas perspectivas para os engenheiros. Portanto, este trabalho consiste em estudar um algoritmo paralelo para a solução direta de sistemas de equações algébricas lineares oriundas do Método dos Elementos Finitos.

De modo a demonstrar a aplicabilidade e a eficiência das idéias apresentadas neste trabalho, um código paralelo foi implementado em uma rede com quatro processadores *Tansputers* T800 [4,5], tendo como hospedeiro um PC que realiza a interface da rede com o usuário

Este Capítulo está descrito em quatro Seções. A primeira, Seção I.1, aborda as principais razões que motivaram o surgimento do Processamento Paralelo. Uma discussão preliminar sobre desempenho de algoritmos paralelos, está apresentado na Seção I.2. A afinidade entre Processamento Paralelo e o Método dos Elementos Finitos é discutida na Seção I.3, e a Seção I.4 trata da organização dos demais Capítulos deste trabalho.

## I.1 RAZÕES DO PROCESSAMENTO PARALELO

A maioria dos computadores têm uma única unidade de processamento, ou processador, que efetua as operações de uma determinada tarefa. Levando-se em conta que, atualmente o auxílio do computador para a execução de cálculos é inevitável, o aumento do poder computacional destas máquinas e, conseqüentemente, o aumento de velocidade dos processadores, são de suma importância. Mas, devido à limitação da tecnologia atual para produzir processadores velozes, torna-se necessário o estudo sobre novas arquiteturas de computadores e de técnicas de processamento paralelo, como formas alternativas para solucionar estes problemas.

O uso do processamento paralelo para melhorar o desempenho de computadores digitais não é uma idéia nova, e de fato, tem sido considerada desde o projeto dos primeiros computadores nos anos 40. Tradicionalmente, a execução de tarefas complexas vem sendo realizada em computadores sequenciais, mas, já durante os anos 60 e 70, estes tiveram introduzidos dentro do projeto de seus processadores, técnicas de paralelismo, como o emprego de instruções vetoriais e de multiprocessadores, de modo a conquistar maior poder computacional. O paralelismo não se aplica apenas ao uso das máquinas de alto desempenho, ou supercomputadores, mas também às situações de baixa velocidade, uma vez que o uso dos supercomputadores envolve altos custos e afasta-se do conceito de um ambiente

computacional iterativo. Aliado a isto, o aumento do poder computacional dos microprocessadores de 32 bits no final dos anos 80, tornando possível a construção de sistemas de multiprocessadores com o mesmo desempenho de pico que os supercomputadores ( mas por uma fração do preço ), o desenvolvimento de linguagens para programação paralela, e as pesquisas em algoritmos paralelos, vem confirmando que o uso de processamento paralelo é uma das mais promissoras técnicas para se ampliar o poder computacional requerido pelas futuras aplicações.

De forma a se adaptar a esta nova realidade, os projetos que envolvem a próxima geração de supercomputadores, têm contado com o emprego de arquiteturas multiprocessadoras de modo a obter também, significativos avanços de desempenho, que agora são considerados como necessidade.

Os mais novos computadores de arquiteturas paralelas possuem, em geral, muitos processadores que trabalham simultaneamente na execução de uma tarefa. Um processador de extremo interesse é o *Transputer*, também chamado computador de um chip, projetado e desenvolvido pela INMOS [5]. Com o desenvolvimento destes, a implementação de algoritmos paralelos tem um ponto de partida acessível.

O conceito básico de processamento paralelo é que os cálculos envolvidos na análise são divididos e executados simultaneamente em diversos processadores. Desta forma, é de se esperar que o tempo de processamento seja inversamente proporcional ao número de processadores,

aumentando assim, o desempenho da máquina. Embora pareça simples, a exploração de algoritmos concorrentes e o eficiente uso do hardware pelos programadores não podem ser considerados como uma tarefa direta, muito pelo contrário. Visto que aplicações concorrentes são ambas assíncronas e não determinísticas, o programador para obter os benefícios do sistema disponível, deve ter um profundo conhecimento sobre seus hardware e software e isso ainda requer uma grande perícia.

Concluindo, entre as razões para o surgimento do Processamento Paralelo, custos e limitações técnicas, podem ser consideradas como as mais importantes.

## I.2 DESEMPENHO DOS SISTEMAS PARALELOS

Na Computação Paralela, ao contrário da Sequencial (onde todo o problema é solucionado em um único processador), a carga computacional é distribuída em vários processadores que operam em conjunto para solucionar um determinado problema. Ao tratarmos determinado problema de forma paralela, surgem várias questões que não ocorrem em um contexto sequencial. Se um determinado problema gasta um tempo  $t$  para ser feito em um único processador, em princípio,  $P$  processadores, realizariam a mesma tarefa em um tempo  $t/P$ . Esta situação ideal é muito difícil de se obter na prática, devido a questões que apresentaremos a seguir. Entretanto, todo o esforço deve ser efetuado para

se obter uma redução de tempo a mais próxima possível da situação ideal.

As principais questões que levam a uma perda de tempo em relação ao sistema ideal, isto é, que influenciam no desempenho de sistemas paralelos, serão abordadas a seguir. Uma primeira questão é a alocação de tarefas, isto é, a divisão e distribuição da carga computacional em tarefas menores a serem alocadas para diferentes processadores. A preocupação é que a carga seja distribuída da maneira mais equitativa possível, para assim se obter um melhor balanceamento de carga entre os diferentes processadores.

Uma segunda questão é a comunicação de resultados intermediários entre os processadores: as tarefas executadas pelos processadores não são, na grande maioria das vezes, totalmente independentes; alguns processadores necessitam trocar informações entre si para poderem prosseguir com seus cálculos de forma correta. Se o tempo requerido para estas trocas de informações for muito lento, o desempenho será comprometido.

Uma outra questão muito importante que surge no contexto da programação paralela é a sincronização entre os cálculos realizados pelos diferentes processadores. Em alguns métodos, chamados síncronos, os processadores devem aguardar, uns pelos outros, em determinados pontos, para poderem concluir seus cálculos ou receber novos dados; isto pode tornar necessário que os processadores tenham que ficar ociosos por alguns instantes durante a execução do algoritmo. Esse tempo que os processadores ficam parados

esperando uns pelos outros é denominado de tempo de sincronização. Em outros métodos, chamados assíncronos, não há necessidade desta espera; os cálculos e a comunicação de dados é realizada por um processador, de forma completamente independente do outro. Isto elimina o tempo de sincronismo, porém introduz novas questões que devem ser analisadas cuidadosamente quando da utilização do método. A questão relativa ao acentuado vínculo entre hardware e software em sistemas paralelos também pode comprometer o desempenho dos algoritmos paralelos. Um bom algoritmo para um sistema pode não ser tão bom quando usado em outra máquina.

A análise formal do desempenho comparativo entre algoritmos sequenciais e paralelos é feita usualmente utilizando-se conceitos de Ganho e Eficiência, que serão descritos no capítulo III.

Todas as questões citadas acima influenciam de forma decisiva nas medidas de desempenho de um algoritmo paralelo . Deve-se a elas o fato de ser praticamente impossível obter a situação ideal descrita anteriormente.

### I.3 O MÉTODO DOS ELEMENTOS FINITOS E

#### O PROCESSAMENTO PARALELO

Os problemas de Engenharia que têm seu comportamento físico regido por uma ou um conjunto de equações diferenciais parciais, possuem solução exata

somente para certos casos de geometria e condições de contorno simples. Recorre-se, então, a soluções aproximadas através de Métodos Numéricos, onde a integração das equações diferenciais parciais se transforma na resolução de um sistema de equações algébricas lineares, resultante da discretização dos campos a determinar.

Quanto ao tipo de forma e discretização, estes métodos podem ser classificados em dois grupos: métodos de domínio e métodos de contorno. Entre os de domínio podem ser destacados os Métodos das Diferenças Finitas e Volumes Finitos e o Método dos Elementos Finitos; entre os de contorno, o Método dos Elementos de Contorno, ou Método das Equações Integrais de Contorno.

No Método dos Elementos Finitos [46,47], o domínio é subdividido em uma série de subdomínios unidos entre si, denominados "elementos finitos". Sobre cada elemento, são definidas funções de forma, para representar o comportamento da solução. Estas funções são definidas unicamente em termos dos valores da solução em pontos internos ou no contorno de cada elemento. Este desenvolvimento pode ser obtido através de técnicas de resíduos ponderados, princípios variacionais ou um enfoque físico. O uso de computadores permite a utilização destes métodos, e tem viabilizado o seu uso para a resolução de grandes problemas.

Métodos Numéricos, em geral, e Métodos dos Elementos Finitos, em particular, têm seu desenvolvimento estritamente relacionado com os avanços na arquitetura dos computadores, já que sem estes, aplicações a problemas

industriais seriam impossíveis. Relacionados a isto, os usuários de Elementos Finitos têm sido beneficiados com o crescimento das pesquisas em torno das novas arquiteturas dos computadores, uma vez que a utilização deste método tem levado a consideração de grandes modelos, refinamento de malhas, elementos de ordem superior, entre outros.

Os procedimentos numéricos do Método dos Elementos Finitos exigem desta forma, na maioria das vezes, a resolução de grandes sistemas de equações algébricas lineares. Nestes casos, o custo computacional necessário à resolução destes sistemas corresponde à maior parte do custo total da análise. Devido a este fato, a exploração de técnicas de Processamento Paralelo em Programas de Elementos Finitos tem despertado grande interesse, requerendo entre outras coisas, a adaptação e modificação dos atuais algoritmos sequenciais para sua utilização com o máximo de eficiência em máquinas paralelas. Daí a importância e necessidade de implementar Programas de Elementos Finitos em Máquinas Paralelas.

Correntemente, a solução do sistema de equações lineares é obtida por uma maneira sequencial usando computadores de arquitetura convencional. Embora as máquinas vetoriais tenham promovido um ganho efetivo nestes códigos na década passada, o verdadeiro potencial para melhoramento da execução, está nos multicomputadores concorrentes.

Sabe-se já há longo tempo que algoritmos para solução de sistemas continham muitas tarefas e operações que poderiam ser calculadas em paralelo, entretanto,

arquiteturas apropriadas de computadores não tinham até então, sido desenvolvidas para tirar proveito deste fato. Então, alguns dos desafios da mecânica computacional passaram a ser: reformulação de algoritmos e rearranjo da computação de modo que os cálculos concorrentes em uma máquina multiprocessadora sejam utilizados de forma mais eficiente possível.

O objetivo deste trabalho é apresentar um algoritmo para solução de sistemas de equações lineares, que é uma etapa importante do Método dos Elementos Finitos, usando para a sua implementação, um sistema *Transputer* [5] e utilizando a linguagem FORTRAN Paralelo [7,8]. Buscou-se também, verificar se um sistema *Transputer* representa ou não um bom compromisso entre custo e desempenho.

De forma a prover uma visão geral do panorama atual, apresentamos as principais linhas de pesquisa envolvendo soluções diretas de sistemas de equações no QUADRO I.1. Neste quadro, os trabalhos estão agrupados quanto a forma e armazenamento da matriz de rigidez. Assim, temos basicamente três grupos: Método da Coluna Ativa, ou Skyline, Método Esperso Geral e Subestruturação (Decomposição de Domínios ). Relacionados ao primeiro grupo, temos: WILSON e DOWEY[9], que desenvolveram o código COLSOL vetorizado, para resolução das equações de equilíbrio de grandes sistemas estruturais, utilizando a decomposição de Crout. FARHAT e WILSON[11], semelhante a [9], utilizam o ENCORE, multiprocessador paralelo de memória comum, e também, o INTEL, multiprocessador paralelo de memória local. ALVES e OWEN[12], semelhantemente a [11],

utilizam o *Transputer*, multiprocessador paralelo com memória local. FARHAT[13], usa o multiprocessador vetorial/paralelo CRAY, para resolver sistemas semi-definidos e positivo definidos oriundos da análise termodinâmica de estruturas aeroespaciais, utilizando também a fatoração de Crout. Enquanto que STORAASLI[14], implementou um algoritmo similar baseado na fatoração de Cholesky. LOZUPONE[15], apresenta um código baseado nos núcleos computacionais BLAS3 [21], onde utiliza a fatoração de Cholesky, projetado para o IBM 3090, multiprocessador vetorial que possui uma estrutura de hierarquia de memória. O trabalho desenvolvido nesta Tese, foi baseado em FARHAT[10,11], e o código gerado, foi implementado e testado no *Transputer*.

No presente trabalho, que se inclui neste quadro de classificação, teve-se como metas a abordagem dos seguintes aspectos da computação paralela:

- Acessar e desenvolver o uso de técnicas de processamento paralelo, envolvendo conhecimentos em FORTRAN Paralelo e Sistemas *Transputers*.

- Obtenção do máximo proveito do poder computacional dos *Transputers* na resolução de sistemas de equações algébricas lineares para a análise estrutural.

SOLUCAO DIRETA DO SISTEMA DE EQUAÇÃO  $Ku=p$

SKYLINE	{	VETORIAL	{	<i>Wilson &amp; Dovey</i> , CRAY	[9]
		PARALELO	{	<i>Farhat &amp; Wilson</i> , HIPERCUBO	[11]
				<i>Alves &amp; Owen</i> , TRANSPUTER	[12]
MÉTODO ESPARSO	{	VET.+ PARAL.	{	<i>Farhat</i> , CRAY	[13]
				<i>Storaasli</i> , CRAY	[14]
				<i>Lozupone</i> , IBM 3090	[15]
SUBESTRUTURAÇÃO	{	PARALELO	{	<i>Lewis &amp; Simon</i> , CRAY	[16]
				<i>Boning &amp; Levy</i> , CRAY	[17]
		MULTIFRONTAL	{	<i>George</i> , HIPERCUBO	[18]
SUBESTRUTURAÇÃO	{	PARALLEL	{	<i>Duff</i> , CRAY	[19]
				<i>Liu</i> , HIPERCUBO	[20]
		VET. + PARAL.	{	<i>Farhat</i> , HIPERCUBO	[22]
				<i>Siqueira</i> , TRANSPUTER	[23]
				<i>Bergan</i> , CRAY	[24]

- QUADRO I.1 -

#### I.4 ORGANIZAÇÃO DO TEXTO

A organização dos demais capítulos deste trabalho é a seguinte:

O Capítulo II apresenta uma rápida abordagem dos computadores de funcionamento vetorial [25] e uma ampla abordagem dos de funcionamento paralelo [26,27]. Para os

deste último caso, são apresentadas as seguintes classificações: classificação quanto ao controle dos processadores, quanto ao tipo de acesso à memória pelos processadores e, quanto à forma de comunicação entre os processadores.

O Capítulo III é dedicado às discussões dos muitos fatores que influenciam o desempenho dos computadores paralelos [26] e, em particular dos sistemas Transputers [6]. A descrição do Sistema Transputer, sua forma de programação, sua classificação geral dentro dos sistemas multiprocessadores e algumas peculiaridades desta máquina estão apresentadas no Capítulo IV [4,5,6,7, e 28].

O Capítulo V é destinado à apresentação do sistema computacional disponível, e ao estudo dos algoritmos sequencial e paralelo, pelo método direto de Gauss, de sistemas de equações algébricas lineares positivas e simétricas. Os sistemas de equações considerados aqui, são oriundos da aplicação do Método dos Elementos Finitos. Descrevemos também neste capítulo a implementação no sistema *Transputer*, de um código em FORTRAN Paralelo baseado na decomposição de Crout para a fase de fatoração do sistema, onde emprega-se um algoritmo do tipo coluna ativa, ou Skyline [11]. Abordamos apenas a fase de decomposição, já que, em estudos anteriores [11,14 e 18], foi verificada a ineficiência da solução completa do sistema ( fase de fatoração, fase de retrosubstituição e fase de substituição progressiva ), para multiprocessadores de memória local. O código desenvolvido pode ser facilmente modificado para execução em qualquer classe de

multiprocessadores de memória local, com apenas pequenas modificações, devido às peculiaridades de cada máquina, caracterizando deste modo, a sua portabilidade. Entretanto, é preciso resaltar que o trabalho de programação aqui feito, o foi de forma a explorar eficientemente as características intrínsecas do sistema *Transputer*.

No capítulo VI analisamos os resultados, obtidos pelo programa proposto no capítulo anterior, a partir de uma série de exemplos que exploram diversas características topológicas que suas matrizes de rigidez podem apresentar.

No capítulo VII apresentamos as principais conclusões deste trabalho, onde são discutidas as dificuldades encontradas, bem como, possíveis aperfeiçoamentos. Finalizando, o Apêndice apresenta a listagem do código paralelo aqui desenvolvido.

## C A P Í T U L O   I I

### PROCESSAMENTO VETORIAL E PARALELO

O primeiro método amplamente disseminado para classificar arquiteturas de computadores foi proposto por FLYNN [27], em 1966. Esta classificação, entretanto, é bastante imprecisa para classificar as variedades de multiprocessadores e processadores vetoriais que surgiram ao longo dos últimos 10 anos [29] e, conseqüentemente, surgiram várias outras propostas com o objetivo de analisá-las [30,31,32 e 33]. Assim, neste capítulo, que está organizado em duas seções, procurou-se traçar um panorama atual dos computadores de processamento vetorial ( apresentados na Seção II.1 ) e, dos computadores paralelos( Seção II.2 ).

Na Seção II.1, apresenta-se o conceito de Segmentação (*Pipelining*) e a classificação dos computadores vetoriais quanto ao acesso à memória. Na Seção II.2, discutem-se as classificações: quanto ao controle dos processadores, quanto ao tipo de acesso à memória e, quanto à forma de comunicação.

#### II.1 COMPUTADORES VETORIAIS

Computadores de funcionamento vetorial são aqueles que utilizam o conceito de *Pipelining* [25], que é a segmentação explícita de uma unidade aritmética em

diferentes partes, isto é, divisão de todos os elementos da unidade na qual se alojam os circuitos que executam as operações aritméticas, em partes que podem ser operadas independentemente, onde cada qual desempenha uma subfunção sobre um par de operandos. Isto está exemplificado na figura II.1 para o caso de adição de ponto flutuante.

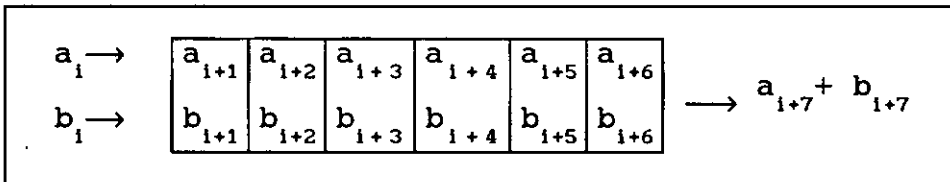


FIG. II.1 - Pipeline EM 6 SEÇÕES

Neste exemplo, um somador ( dispositivo cuja saída é a representação das quantidades de entrada ) de ponto flutuante é segmentado em seis seções, onde cada qual executa uma parte do somatório global. Cada segmento pode estar trabalhando em um par de operandos, de modo que seis pares de operandos podem estar no Pipeline ao mesmo tempo. A vantagem desta segmentação é que os resultados estão sendo computados a uma taxa que é seis vezes mais rápida ( ou, para um caso geral,  $k$  vezes, onde  $k$  é o número de segmentos ) que uma unidade aritmética onde são aceitos um par de operandos e computados os resultados antes de admitir o próximo par de operandos. Entretanto, de modo a utilizar esta capacidade, os dados devem atingir as unidades aritméticas de forma rápida o suficiente para manter o Pipeline cheio. Uma simples instrução de hardware controlará o carregamento dos operandos e o armazenamento dos resultados.

Os computadores vetoriais podem ser classificados

quanto o acesso à memória [26], como:

### a) Processadores de Memória a Memória

Aqui, as operações vetoriais requerem seus operandos diretamente da memória principal e armazena o resultado de volta na memória principal. Isto está exemplificado na figura II.2 para uma adição vetorial.

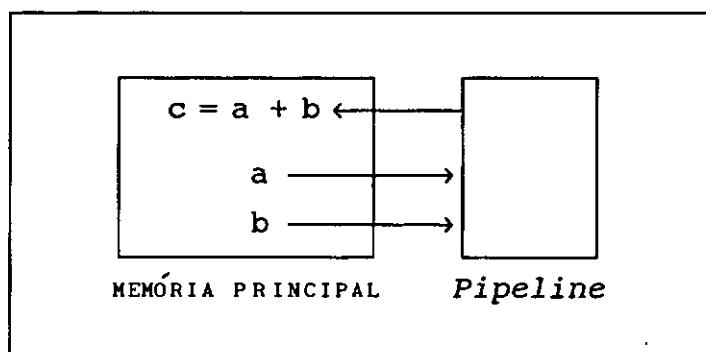


FIG.II.2 - OPERAÇÃO DE ADIÇÃO MEMÓRIA A MEMÓRIA

### b) Processadores Registrador a Registrador

Aqui, as operações vetoriais obtêm seus operandos de uma memória muito rápida, chamada *Registrador Vetorial* (*Vector Register*), e armazena os resultados de volta no registrador vetorial. Isto está exemplificado na figura II.3 para uma adição vetorial. Nela, cada registrador vetorial está dimensionado para conter um certo número de palavras. Os operandos para uma adição vetorial são obtidos de dois registradores vetoriais e o resultado é armazenado em outro registrador vetorial. Antes da adição vetorial, os registradores vetoriais devem ser carregados da memória principal e, em algum ponto, o vetor resultado é armazenado

de volta na memória principal.

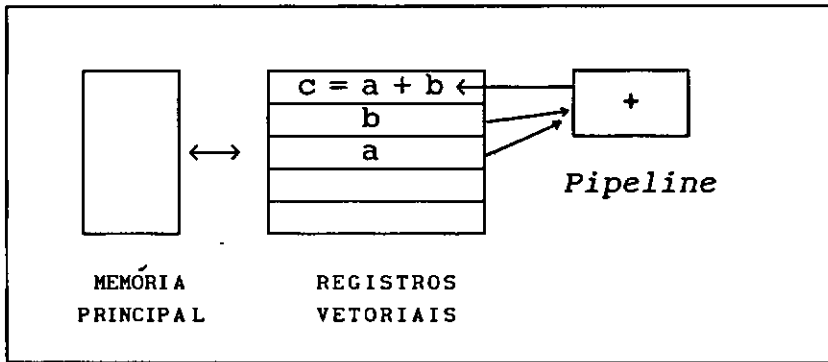


FIG. II.3 - ADIÇÃO REGISTRO A REGISTRO

### c) Hierarquias de Memória

Os mais recentes computadores de funcionamento vetorial utilizam uma complexa hierarquia de memória, isto é, uma divisão da memória principal que permite estabelecer diferenciações entre distintas zonas de memória do processador, para efeitos de endereçamento. O objetivo para o uso destes vários níveis de armazenamento é se permitir que se tenha os dados sempre disponíveis para as unidades aritméticas.

## II.2 COMPUTADORES PARALELOS

As inúmeras possibilidades de organização de um grande número de processadores têm produzido de forma correspondente, uma grande variedade de computadores paralelos. Estes têm como objetivo atingir um alto desempenho em classes de problemas tradicionalmente

dominados pelos supercomputadores, a um custo relativamente baixo. Com relação a esta grande variedade de computadores paralelos, apresentamos três critérios empregados para a sua classificação. O primeiro critério abordado no Ítem II.2.1, é quanto ao controle dos processadores. O segundo, descrito no Ítem II.2.2, é quanto o acesso à memória pelos processadores, e finalmente, o último critério, ( Ítem II.2.3 ), é quanto à forma de comunicação entre os processadores.

### II.2.1 MÁQUINAS SIMD E MIMD

De modo a conhecer como os processadores podem ser controlados, apresentamos duas classes principais de computadores: as máquinas SIMD e as MIMD [25,26].

#### a) SIMD (Single Instruction stream/Multiple Data stream)

Nesta classe, ilustrada na figura II.4, todos os processadores e módulos de memória paralela estão sob supervisão de uma única unidade de controle, e todos os processadores em particular fazem a mesma instrução (ou nenhuma) ao mesmo tempo. Assim, há um simples fluxo de instrução operando sobre múltiplos fluxos de dados, um para cada processador.

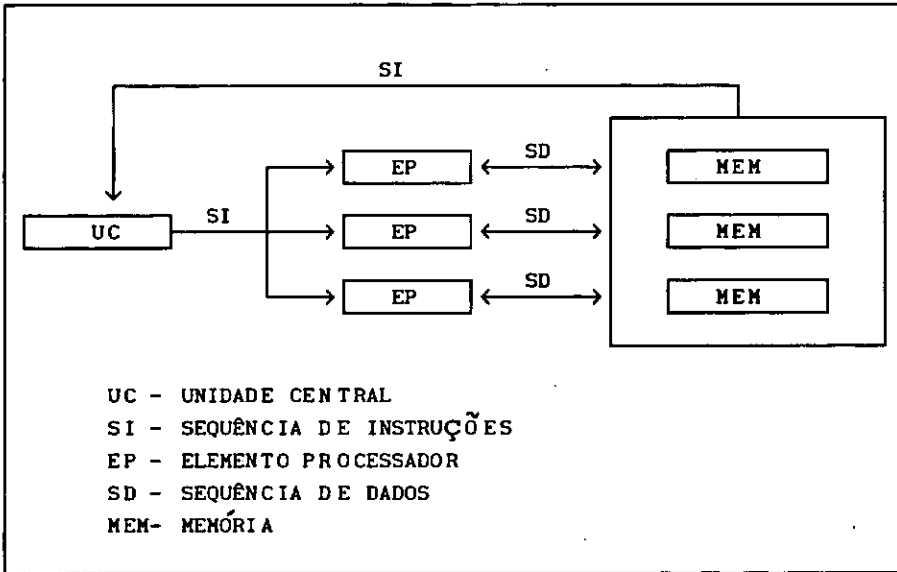


FIG. II.4 - COMPUTADOR SIMD

Os computadores Vetoriais também podem ser conceitualmente incluídos nesta classe tendo em vista que os elementos de um vetor estão sendo processados individualmente sob o controle de uma instrução física de hardware.

#### b) MIMD (Multiple Instruction stream/Multiple Data stream)

Em máquinas MIMD (frequentemente referidas como multiprocessadores), os processadores estão conectados aos módulos de memória por uma rede e executam suas tarefas sob seus próprios controles. Tal fato proporciona ao programador uma maior flexibilidade na designação das tarefas que os processadores executarão num mesmo instante de tempo. A grande maioria dos computadores paralelos estão

relacionados nesta classe, que está ilustrada na figura II.5.

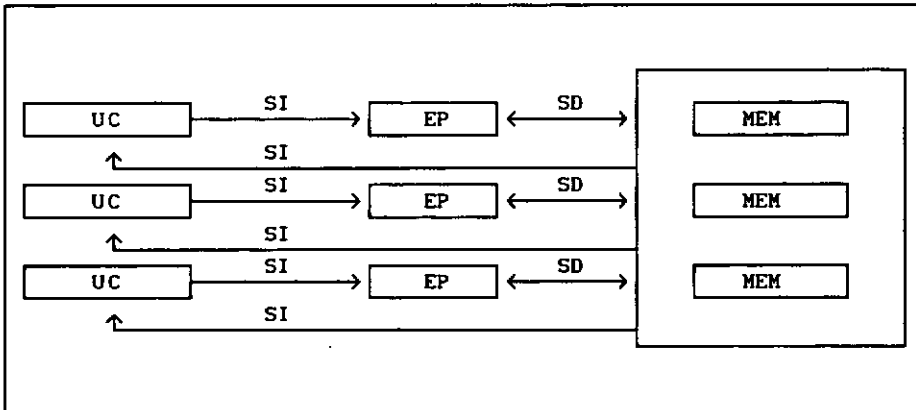


FIG. II.5 - COMPUTADOR MIMD

Em um sistema SIMD, a sincronização dos processadores individuais é realizada pela unidade de controle, mas em um sistema MIMD, outros mecanismos devem ser usados para assegurar que os processadores estejam fazendo seus trabalhos na ordem correta com o dado correto.

Cabe salientar que o sistema *Transputer* utilizado nos desenvolvimentos desta tese é do tipo MIMD.

## II.2.2 MEMÓRIA COMPARTILHADA E DISTRIBUÍDA

A característica mais crítica em uma máquina paralela é conhecer como os processadores estão conectados entre si e como eles acessam a memória [26].

Há dois casos extremos de classificação conhecidos como sistemas de memória compartilhada (*Shared Memory*) e sistemas de memória distribuída (*Local Memory*); existe ainda uma variedade de estruturas híbridas que utilizam ambas as características.

Computadores de memória compartilhada, ilustrado na figura II.6, normalmente consistem em um pequeno número de processadores, onde todos eles têm acesso a uma mesma memória central (comum), e cada um possui uma memória local que é usada para armazenar valores e informações desnecessárias a outros processadores.

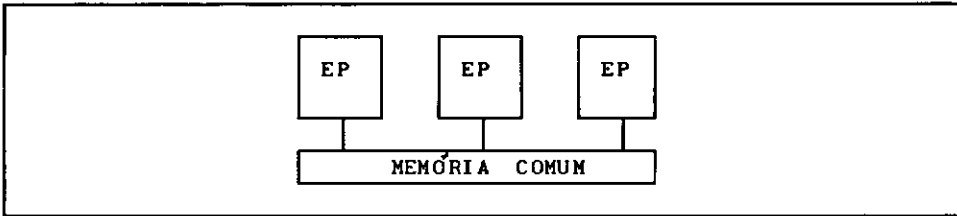


FIG.II.6 - MEMÓRIA COMPARTILHADA

A comunicação entre os processadores é feita via essa memória comum e ocorre da seguinte forma: um processador escreve na memória comum o dado a ser comunicado e qualquer outro processador que necessitar daquele dado faz uma leitura na mesma localização de memória.

Uma vantagem é que a comunicação entre processadores é muito rápida, ORTEGA [26] ; uma séria desvantagem deste tipo de sistema é o fato de que diferentes processadores podem necessitar utilizar a memória simultaneamente e, neste caso, apenas um poderá utilizá-la, enquanto que os outros deverão aguardar até que ela esteja livre, isto é, não há sincronismo na comunicação entre os processadores, o que acarreta em uma perda de tempo. O atraso gerado é chamado de tempo de contenção e ele cresce com o aumento do número de processadores.

Em computadores de memória distribuída (local), ilustrado na figura II.7, não existe memória comum, cada

processador tem sua própria memória local de uso exclusivo e, a comunicação entre os processadores é feita por envio e recebimento de mensagens ( "message-passing" ). Esta arquitetura admite que um grande número de processadores sejam conectados juntos. Nestes tipos de sistemas, o problema de tráfego de memória é de maior importância ( não há problemas de sincronização).

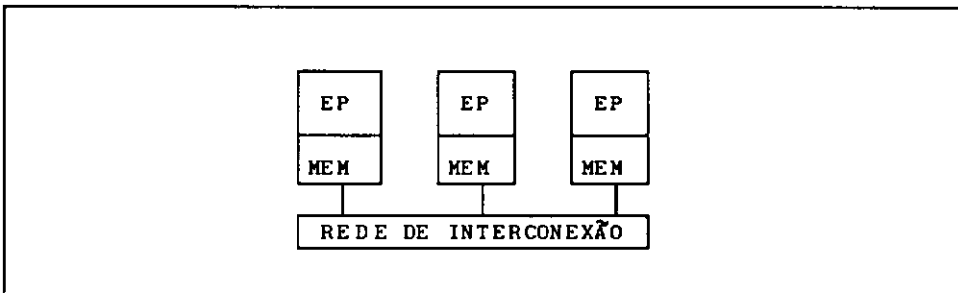


FIG.II.7 - MEMÓRIA DISTRIBUÍDA

Os *Transputers* são classificados como sistemas de memória distribuída.

### II.2.3 ESQUEMAS DE LIGAÇÕES

Provavelmente, o mais importante aspecto da computação paralela é conhecer como os processadores se comunicam uns com os outros. Isto é particularmente interessante para sistemas em que os processadores têm somente memória local, e também aos sistemas de memória compartilhada desde que a conexão para a memória compartilhada possa ser implementada por diferentes esquemas de ligações.

Os diferentes esquemas de ligações serão agora relacionados [26] :

### a) Conexão Completa

Em um sistema de Conexão Completa ( *Completely Connected* ), cada processador tem uma conexão direta para todos os outros processadores. Um sistema de conexão completa com  $P$  processadores requer  $P-1$  linhas emanadas para cada processador, o que é impraticável se  $P$  é muito grande. A figura II.8 exemplifica o modelo para  $P = 4$ .

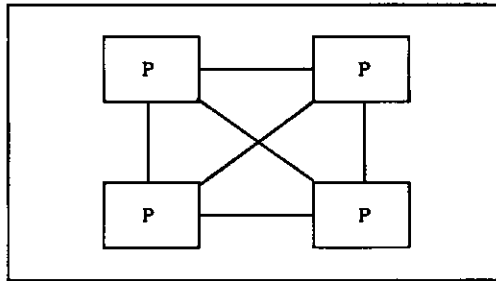


FIG. II.8 - CONEXÃO COMPLETA

### b) Barras Cruzadas

Outra aplicação para um sistema de conexão completa é por meio de uma barra de comutadores, como ilustrada na figura II.9. Aqui, cada processador pode ser conectado para cada memória por meio de comutadores. A vantagem é que qualquer processador acessa qualquer memória com um número pequeno de conexões de linhas. mas o número de comutadores para conectar  $P$  processadores e  $P$  memórias é  $P^2$ , o qual torna-se impraticável para altos valores de  $P$ .

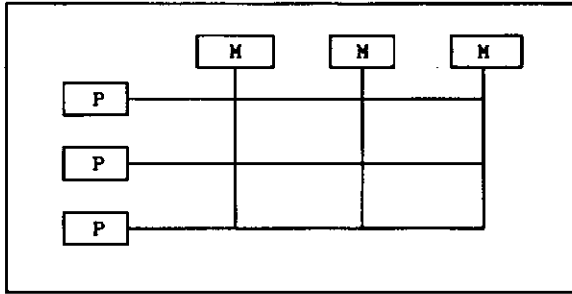


FIG. II. 9 - BARRAS CRUZADAS

### c) Linha e Anel

Uma Rede com barramentos (*Bus*), está ilustrado na figura II.10. Aqui, todos os processadores estão conectados por uma linha pela qual é possível transmitir informações em alta velocidade. Uma vantagem está no menor número de linhas em conexão, mas pode haver contenção no uso da linha por diferentes processadores. Isto pode tornar-se um problema severo quando do aumento do número de processadores.

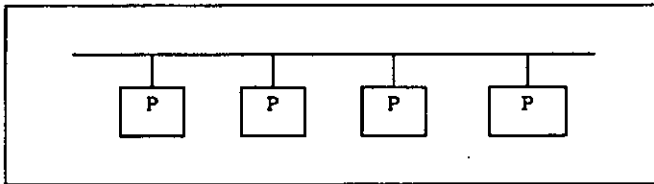


FIG. II. 10 - BARRAMENTO

Uma Rede em Anel (*Ring*), é uma rede de linha fechada como mostrada na figura II.11. Aqui os dados movem-se ao longo do anel e são enviados para cada processador. Alguns sistemas a usam para conectar processadores a uma memória global, outros, para sistemas de memória local.

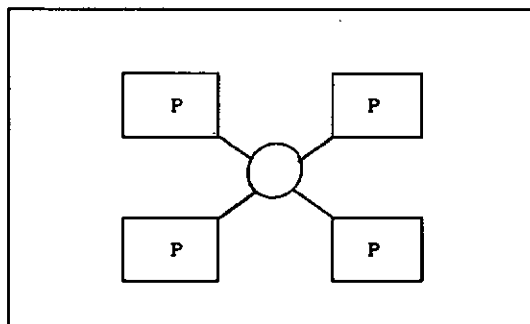


FIG. II.11 - ANEL

#### d) Conexão de Malha

Um dos esquemas de ligações mais populares é ter cada processador conectado a somente poucos processadores vizinhos. O exemplo mais simples disto é um Arranjo Linear ( *Linear Array* ), mostrado na figura II.12. Aqui, cada processador está ligado aos dois vizinhos mais próximos ( exceto nos extremos, os quais estão conectados para somente um ).

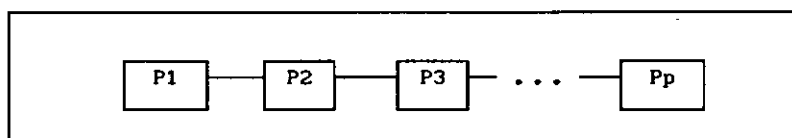


FIG. II.12 - ARRANJO LINEAR

Uma vantagem desta ligação está na sua simplicidade, uma vez que apenas duas linhas são emanadas de cada processador. A maior desvantagem é que os dados podem precisar passar através de muitos processadores para chegar ao seu destino final ( note que esta é diferente de uma conexão em linha, embora figuras II.12 e II.10 sejam parecidas ). Por exemplo, se um processador P1 precisar levar um dado para o processador Pp , o dado deve primeiro passar por P2, e então, transmitido a P3, e assim

sucessivamente. Então, P-1 transmissões devem ser feitas para o dado atingir o seu destino final.

O número máximo de transmissões que devem ser efetuadas para a comunicação entre dois processadores de um sistema é chamado de *Comprimento de Comunicação* ou *Diâmetro do sistema*.

O diâmetro de um arranjo linear é diminuído por um Arranjo em Anel ( *Ring Array* ), no qual agora, a distância máxima entre dois processadores é a metade em relação ao arranjo linear. A comunicação em um arranjo em anel pode também ser unidirecional ou bidirecional. A figura II.13 ilustra o arranjo em anel.

A maioria dos Arranjos de Malha Conectada ( *Mesh Connection* ) que atualmente tem sido construídos usam um modelo de conexão bidimensional. O esquema de conexão mais simples é mostrado na figura II.14, no qual os processadores estão dispostos em uma malha regular bidimensional e cada processador está conectado aos seus vizinhos de norte, sul leste e oeste. De modo que, nas bordas, os processadores estão conectados de maneira a envolver os demais.

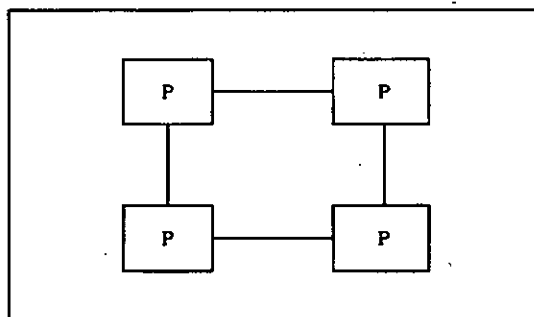


FIG. II.13 - RING ARRAY

Uma vantagem deste modelo está na simplicidade

das conexões. Ele também tem a mesma desvantagem de um arranjo linear em que a transmissão de dados entre dois processadores distantes deve passar através de uma sucessão de processadores intermediários.

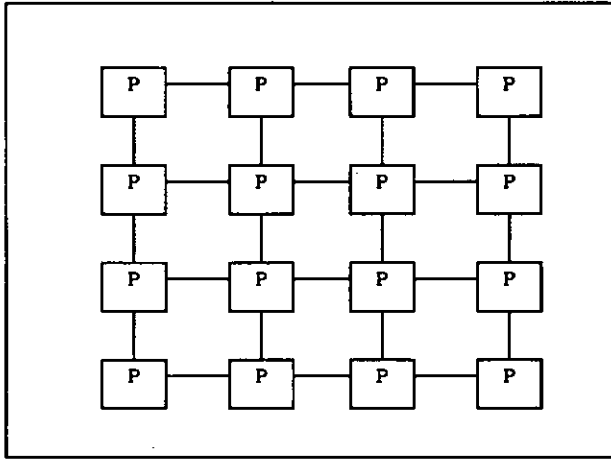


FIG. II.14 - ARRANJO MALHA CONECTADA

### e) Hipercubos

Uma interessante variação do princípio de conexão local está em considerar, conceitualmente, conexões locais de dimensões superiores. Considere primeiro a conexão padrão em tres dimensões ilustrada na figura II.15. Aqui os processadores são os vértices de um cubo no espaço 3 e as bordas do cubo são os links locais entre os processadores. Assim, pela figura II.15, cada processador está conectado a seus tres vizinhos mais próximos no sentido dos vértices mais próximos do cubo.

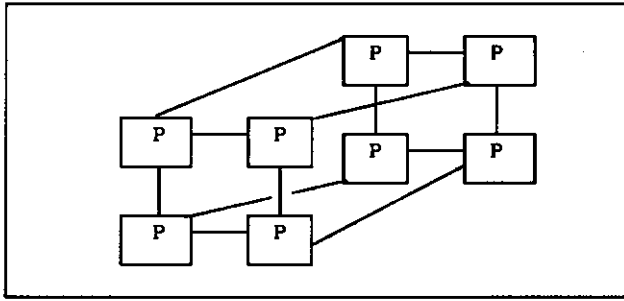


FIG.II.15 - HIPERCUBO

Agora, imagine um análogo esquema de conexão usando um cubo na dimensão  $k$ . Novamente, os processadores podem ser visualizados como os  $2^k$  vértices deste cubo  $k$ -dimensional. Cada processador está conectado para seus  $k$  vértice adjacentes, ao longo das bordas do cubo. Um modelo de conexão semelhante a este é chamado de Hipercubo ou Conexão Binária  $k$ -Cúbica. Para um cubo de 4 dimensões, (4-Cubo), há 16 processadores, com cada processador conectado a outros 4. A figura II.16 ilustra a maneira em três dimensões para se visualizar o modelo de conexão de um 4-Cubo e também mostra que o 4-Cubo pode ser obtido por correspondentes conexões dos vértices de dois 3-cubos. Em geral, pode-se construir um  $k$ -Cubo pela conexão de todos os processadores correspondentes de dois " $(k-1)$  - Cubos".

Em um esquema hipercubo de conexão, o número de conexões de cada processador aumenta com o acréscimo de processadores, e o comprimento de comunicação é apenas  $\log_2 P$ . Por exemplo, para 64 processadores, 6-Cubo, cada processador está conectado a outros seis processadores, e o comprimento de comunicação é 6, enquanto que para 1024 processadores, 10-Cubo, cada processador está conectado a outros dez processadores e o comprimento de comunicação é 10. Além disso, o hipercubo contém alguns outros modelos

de conexão. Por exemplo, anel ou arranjo de malha conectada podem ser incluídas ao se ignorar algumas das ligações locais. Por outro lado, a medida que a dimensão do cubo é aumentada, a complexidade e o número de linhas emanadas de cada processador também cresce e um limite prático para o tamanho do cubo será eventualmente atingido.

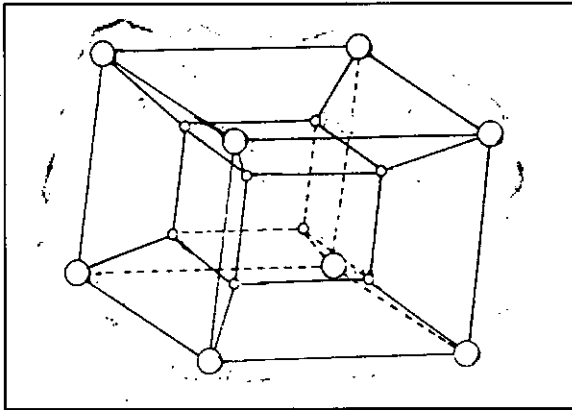


FIG. II.16 - HIPERCUBO DE ORDEM 4

#### f) Rede Comutada

Um caminho geral preferível para conexão de processadores, ou conexão de processadores à memória, é por meio de Rede Comutada ( *Switching Networks* ), que é o meio de ligação de um telefone a outro.

Uma rede comutada simples é descrita na figura II.17. A esquerda são mostrados oito processadores e a direita estão oito memórias. Cada caixa representa um comutador de duplo caminho e as linhas são *links* de transmissão.

Por meio destas redes de comutadores, cada um dos processadores pode endereçar algo para as memórias. Por exemplo, suponha P1 desejando endereçar M8. O comutador 1.1

é ajustado a aceitar o link de P1 e ajusta o caminho para o comutador 2.2, que ajusta o caminho para 3.4, que ajusta o caminho para M8.

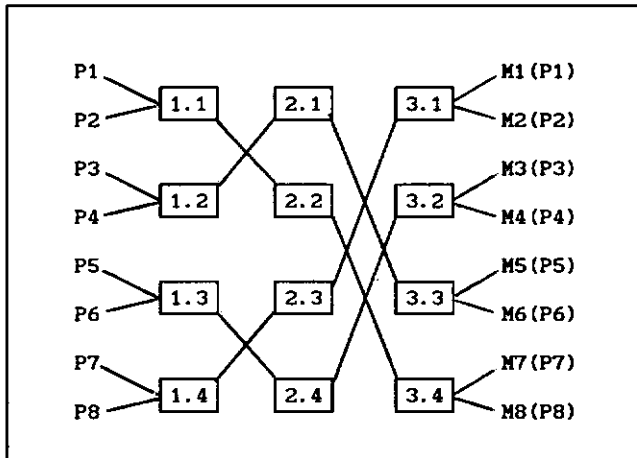


FIG.II.17 - REDE COMUTADA

A figura II.17 representa uma implementação de rede comutada para um sistema de memória compartilhada, uma vez que cada processador pode acessar cada memória. Alternativamente, as memórias da direita podem estar localizadas nos próprios processadores, como indicado pelo número do processador entre parenteses. Neste caso, a figura II.17 descreverá um sistema de memória local com o uso de " Message-Passing ".

Usando um interruptor de caminho duplo como na figura II.17, necessita-se de  $P/2$  interruptores para cada estágio de comutação e  $\log_2 P$  estágios para um total de  $(P/2) \log_2 P$  interruptores. Isto, comparado com  $P^2$  interruptores necessários ao esquema de Barras Cruzadas, torna-se mais favorável. Por exemplo, com  $P = 2^{10}$ , somente  $5 \times 2^{10}$  interruptores serão necessários, em comparação a  $2^{20}$ . Muitos sistemas paralelos utilizam a conexão por rede

Comutada, mas não necessariamente da forma mostrada na figura II.17.

### g) Esquemas Híbridos

Como foi apontado na discussão acima, cada esquema de conexões tem suas vantagens e desvantagens. Isto motiva a possibilidade de se combinar dois ou mais esquemas de modo a obter as maiores vantagens de cada e minimizar as desvantagens. Por exemplo, suponha que os processadores de um arranjo de malha conectada também estão conectados por um barramento. Então a comunicação entre os processadores próximos podem ser feitas pelos links locais, liberando o barramento desta quantidade de tráfego, enquanto que a comunicação entre os processadores mais distantes pode usar o barramento. Uma variedade de outros tipos de esquemas híbridos podem ser descritos.

### h) Grupos

Por definição, Grupos ( *Clusters* ) são agrupamentos de unidades ou dispositivos similares. Um esquema tipo grupo é ilustrado na figura II.18. Nesta, há  $n$  grupos, cada um consistindo de  $m$  processadores. Dentro de cada grupo, os processadores podem estar conectados por qualquer um dos esquemas apresentados anteriormente e os grupos estão ligados por um barramento. A comunicação dentro de cada grupo é chamada de *Local* enquanto que a comunicação entre os grupos é chamada de *Global*. O

esperado para tais esquemas é que haverá balanceamento para estes dois tipos de comunicação; isto é, a maior parte da comunicação será feita entre os processadores dentro de seu próprio grupo ( local ) e, com menor frequência, haverá comunicação entre os grupos.

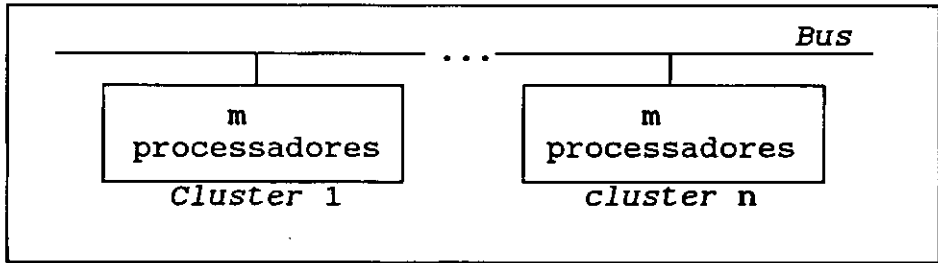


FIG. II.18 - CONEXÃO TIPO *Cluster*

Claramente, há um grande número de maneiras que um esquema tipo *Cluster* pode ser implementado. As ligações entre os grupos também podem usar outros tipos de esquemas. Além disso, o conceito de agrupamento pode ser usado repetidamente para se obter grupos de grupos de grupos e assim por diante.

#### i) Esquemas Reconfiguráveis

Outro modo para se tentar dominar as limitações de um esquema simples de ligação é quanto à capacidade de reconfigurar a conexão padrão. Isto pode ser feito estaticamente, no início do programa, ou dinamicamente no meio de um programa e sob o controle do programa.

## j) Sistemas Paralelo-Vetorial

Existe ainda os Sistemas Paralelos cujos processadores são vetoriais. São chamados de Sistemas Paralelo-Vetorial. A idéia é obter a maior vantagem possível utilizando-se de todas as ligações anteriormente discutidas. Estas máquinas serão as que provavelmente dominarão o mercado num futuro bem próximo.

## C A P Í T U L O    I I I

## DESEMPENHO DOS SISTEMAS PARALELOS

Como visto no Capítulo I, Seção I.2, a execução de um algoritmo concorrente em  $P$  processadores, deveria em princípio, ser  $P$  vezes mais rápida do que a mesma execução em um processador. Entretanto, foi visto também, que o desempenho é degradado devido a fontes de ineficiências que estão diretamente relacionadas com o hardware e/ou software, tais como: comunicação entre os processadores, sincronização de processos, distribuição desigual de cargas e disponibilidade de recursos computacionais, entre outras razões. De modo a entender e minimizar as perdas produzidas pelas fontes de ineficiências citadas acima, objetivando-se criar códigos paralelos eficientes em sistemas computacionais de multiprocessamento, este Capítulo introduz conceitos e métodos de avaliação de desempenho para a análise de algoritmos voltados à computadores paralelos e, em particular, aos sistemas *Transputers*.

Na Seção III.1, abordamos o conceito de velocidade computacional [6,25 e 26]. A Seção III.2, trata de alguns conceitos básicos e as principais medidas de paralelismo que acessam o desempenho de qualquer máquina paralela, tais como: grau de paralelismo, grau médio de paralelismo, granularidade, ganho e eficiência [6,26]. A Seção III.3, apresenta as expressões relacionadas aos vários fatores que afetam o desempenho de um programa

hipotético capaz de ser executado em um sistema de multiprocessadores [6]. Uma discussão detalhada sobre a importância destes fatores serão abordadas nas Seções III.4 em diante [2,6 e 26]. Devido ao profundo vínculo entre o hardware e o software, o uso da expressão sistema paralelo para designar o processamento paralelo, foi adotado por todos os demais capítulos deste trabalho.

### III.1 VELOCIDADE COMPUTACIONAL

A velocidade computacional de modernas arquiteturas está diretamente relacionada ao poder computacional destas máquinas e, atualmente, é uma importante medida para a avaliação do desempenho de um sistema paralelo. Em uma máquina sequencial, o desempenho nominal é usualmente dado pela velocidade de seu processador central, embora o desempenho real sofra grande influência da velocidade dos dispositivos de armazenamento, bem como das operações de ponto flutuante.

O uso de testes padronizados podem também dar uma indicação nítida da velocidade de um computador sequencial, apesar da dificuldade de se escolher um teste válido para um determinado problema em particular. Um modo mais prático [6] utilizado na comparação entre duas máquinas, é avaliar a velocidade da máquina que pode ser expressa tanto em termos de MFlops ( milhões de operações de ponto flutuante por segundo ), quanto em termos de MIPS ( milhões de instruções por segundo ). Para os supercomputadores atuais, o Gigaflop é usado para descrever a gigantesca velocidade

( 1GFlops = 1000 MFlops ) alcançada. Da definição de MFlops, admite-se que quando  $N$  operações de ponto flutuante são feitas em  $t$  microsegundos, a velocidade de computação é dada por,  $V = N / t$  MFlops, e vice-versa, quando  $N$  Flops são executados com uma velocidade computacional média de  $V$  MFlops, então o tempo de CPU é  $t = N / V$  microsegundos[25].

Similarmente, o desempenho nominal de um sistema paralelo é dado pela soma das velocidades de cada processador, embora ele não possa ser tomado como uma medida de desempenho para toda aplicação, já que, de um modo geral, programas paralelos são capazes de extrair somente uma fração do desempenho nominal de sistemas paralelos. Além disso, duas implementações de um mesmo algoritmo sobre o mesmo computador podem resultar em uma grande disparidade no desempenho, já que eles podem usar os recursos das máquinas de maneiras diferentes. A mesma disparidade pode ser encontrada na implementação de um mesmo algoritmo em duas máquinas diferentes, tornando com isso, mais controvertido o estabelecimento de testes padronizados para computadores paralelos, do que para máquinas sequenciais. Ainda, diferentes operações podem ser executadas em velocidades completamente diferentes, o que implica em que não se pode analisar o desempenho de um algoritmo paralelo pela simples contagem de número de Flops ( apesar deste número ser uma boa indicação em computadores clássicos ). Em vez disso, temos que levar em conta o fato de que partes específicas de um algoritmo paralelo exploram determinadas facilidades de uma dada arquitetura [25].

### III.2 MEDIDAS DE DESEMPENHO

Apresentamos agora alguns conceitos básicos de paralelismo, bem como as principais formas de avaliação do desempenho de um sistema paralelo composto de  $P$  processadores.

Um modo aprovado de avaliar o desempenho de uma aplicação paralela é através dos fatores de Ganho e Eficiência [6,26]. Antes de definirmos estes dois fatores, vejamos alguns conceitos básicos de paralelismo.

O primeiro conceito a ser definido, é o de Grau de Paralelismo de um algoritmo numérico, que representa o número de operações que podem ser feitas em paralelo [26]. É uma medida intrínseca do paralelismo do algoritmo pois não depende do número de processadores do sistema, embora isto afete o tempo necessário para completar um cálculo. Para ilustrar esta definição, considere o problema de adicionar dois vetores  $A$  e  $B$ , ambos de ordem  $n$ . As adições  $a_i + b_i$ ,  $i = 1, n$  são independentes e podem ser feitas em paralelo. Assim, seu Grau de Paralelismo será  $n$ . Note que, se  $n = 1000$  e  $P = 1000$ , então a adição será feita de uma só vez, mas se  $P = 10$ , serão necessários 100 intervalos de tempo.

Outro importante conceito a ser definido, é o Grau Médio de Paralelismo de um algoritmo, que representa o número total de operações em um algoritmo dividido pelo número de estágios do mesmo [26]. A importância deste novo conceito está relacionado com o fato de que cada estágio de solução de um algoritmo pode ter diferentes Graus de

## Paralelismo.

Relacionado ao conceito de Grau de Paralelismo, está a idéia de granularidade [26]. Granularidade de alta escala significa grandes tarefas executadas independentemente em paralelo, como por exemplo, a solução de seis diferentes grandes sistemas de equações lineares, cujas soluções serão combinadas no último estágio de cálculo. Granularidade em pequena escala significa pequenas tarefas que podem ser executadas em paralelo; um exemplo é a adição de dois vetores onde cada tarefa é a soma de dois simples escalares.

Agora, apresentaremos a medida mais comumente usada na avaliação de uma aplicação paralela, denominada Ganho. Este é um fator que compara o tempo total consumido por um algoritmo quando executado em uma máquina sequencial em relação ao tempo requerido pelo mesmo algoritmo quando executado em um sistema paralelo de P processadores. Assim, uma expressão para o Ganho pode ser definida como :

$$S = T_{seq}/T(P) \quad (III.1)$$

onde  $T_{seq}$  e  $T(P)$  representam o tempo consumido por uma máquina sequencial e por uma máquina paralela com P processadores, respectivamente.

O fator Ganho (S) dá uma medida bruta de como uma aplicação paralela é executada em comparação a um programa sequencial equivalente. Teoricamente, o ganho deverá ser sempre menor ou igual a P , embora dependendo de como o tempo sequencial é medido, o valor máximo de S poderá variar, uma vez que, o processador usado para medir  $T_{seq}$  na máquina sequencial pode ter uma velocidade muito diferente

da velocidade dos processadores do sistema paralelo, podendo mascarar a eficiência do algoritmo paralelo real. Mas se o tempo sequencial for medido por um processador exatamente igual ao processador da máquina paralela, este fator pode ser isolado.

A especificação do algoritmo sequencial a ser usado é também muito importante, já que diferentes algoritmos para um mesmo problema apresentam, geralmente, tempos de processamento diferentes, que conseqüentemente produziram ganhos também diferentes. Ainda, um algoritmo paralelo pode não ser o melhor algoritmo quando executado em um simples processador. Uma alternativa é definir o algoritmo sequencial ótimo para um problema particular e usá-lo para medir o tempo  $T_{seq}$  na expressão III.1. A definição de um algoritmo ótimo é entretanto muito vaga e duas alternativas são propostas a seguir [6]:

- (a) o tempo necessário pelo melhor algoritmo sequencial;
- (b) o tempo necessário por um simples processador, para executar um algoritmo paralelo particular.

A escolha de qualquer uma destas alternativas leva a diferentes valores de ganho que devem ser interpretados adequadamente. Para  $T_{seq}$  medido como em (b), seu valor coincide com  $T(1)$ . Esta escolha dá uma boa medida de como um algoritmo particular foi paralelizado, mas, não pode ser tomado como uma medida absoluta de como o algoritmo paralelo pode funcionar relativamente a um sequencial.

Definimos então tres diferentes tipos de ganho que podem ser calculados usando a expressão III.1 e que

diferem em como o tempo sequencial ( $T_{seq}$ ) é definido.

1. Ganho ou  $S$  é usado quando nenhuma restrição é feita para se medir o tempo sequencial. Este valor ( $S$ ) é empregado sobretudo quando uma comparação de máquinas diferentes é necessária, como por exemplo, para considerações de custo. Este valor ( $S$ ) não é influenciado pelo número total de processadores.

2. Um ganho relativo ou  $S_{rel}$  é calculado usando um bom algoritmo sequencial para medir  $T_{seq}$ , como proposto no item (a) acima. O valor máximo de  $S_{rel}$  deverá sempre ser menor ou igual ao número total de processadores  $P$ .

3. Um ganho absoluto ou  $S_{abs}$  usa  $T_{seq} = T(1)$  como proposto no item (b) acima.

O ganho pode ser usado para definir a eficiência de um sistema paralelo como:

$$E = S / P \quad (III.2)$$

A eficiência ( $E$ ) dá uma indicação da porcentagem do tempo total realmente dispendido na aplicação por cada um dos processadores. Valores de eficiência podem somente ser significativos se calculados usando  $S_{abs}$  ou  $S_{rel}$ .

O que pode-se concluir do que foi descrito, é que o uso de diferentes algoritmos, arquiteturas e softwares podem influenciar enormemente o desempenho dos sistemas paralelos.

### III.3 UMA IMPLEMENTAÇÃO HIPOTÉTICA

Para ilustrar a discussão a seguir, considere o seguinte exemplo, proposto em [6]. Seja um programa  $\rho$

rodando em uma máquina paralela com  $P$  processadores, executando dois processos distintos,  $\rho_p$  e  $\rho_s$ . O processo  $\rho_p$  pode ser dividido em  $M$  subprocessos independentes; cada um deles com um tempo  $t_p$  para executar sequencialmente e somente comunicando com outros processos em seu início para entrada e em seu final para saída. Seja também, o processo  $\rho_s$ , com tempo  $t_s$ , estritamente sequencial, isto é, não pode fazer uso de mais do que um processador. Os tempos totais de execução  $T_{seq}$  e  $T(P)$ , respectivamente para uma execução sequencial e paralela do programa  $\rho$ , podem ser calculados como:

$$T_{seq} = t_s + ( t_p * M ) \quad (III.3)$$

$$T(P) = \left\{ ( t_p + t_{oh} ) * \lfloor \beta \rfloor \right\} + t_s \quad (III.4)$$

onde o operador  $\lfloor \rfloor$  significa um número inteiro tal que  $\beta \geq \lfloor \beta \rfloor < (\beta+1)$ ,  $\beta = M / P$  é a medida de granularidade para o problema e  $t_{oh}$  é a perda total relativa para cada subprocesso de  $\rho_p$ .

Usando as expressões (III.3) e (III.4), as expressões (III.1) e (III.2) podem ser reescritas como:

$$SREL = \frac{P}{f_P + (1+\Gamma) (1-f) \lfloor \beta \rfloor / \beta} \quad (III.5)$$

$$E = \frac{1}{f_P + (1+\Gamma) (1-f) \lfloor \beta \rfloor / \beta} \quad (III.6)$$

onde:

$$f = t_s / T_{seq} \quad (\text{III.7})$$

$$\Gamma = t_{oh} / t_p \quad (\text{III.8})$$

Estas expressões incluem fatores que afetam o desempenho de um programa hipotético. Uma discussão mais detalhada sobre a importância de cada um destes fatores é elaborada nas seções seguintes.

#### III.4 A LEI DE AMDAHL

O efeito do fator  $f$ , na expressão (III.5) e (III.6), pode ser isolado se assumirmos que o sistema paralelo é ideal ( $\Gamma = 0$ ) e que  $[\beta] = \beta$ . Para este caso, o ganho e a eficiência do programa  $\rho$  podem ser reescritos como:

$$S_{REL} = \frac{1}{f + (1-f) / P} \approx \frac{1}{f} \quad (\text{III.9})$$

$$E = \frac{1}{1 + f (P-1)} \quad (\text{III.10})$$

As expressões (III.9) e (III.10) são amplamente conhecidas como Lei de Amdahl [6], a qual geralmente estabelece que a velocidade de um computador está sempre limitada pela velocidade de seu modo mais vagaroso de operação. Para uma máquina paralela, esta lei implica que, a menos que uma parte considerável do algoritmo esteja completamente paralelizada (modo rápido), o desempenho

final está limitada pela parte sequencial (modo lento-f).

Embora este modelo seja muito simplificado, é extremamente intuitivo: suponha que em um dado problema, metade das operações possam ser realizadas em paralelo e a outra metade não. Então,  $f = 1/2$  e (III.9) torna-se:

$$S = \frac{2}{1 + 1/P} < 2$$

Isto é, não importa quantos processadores existam, e ignorando todas as perdas geradas pela comunicação, sincronização e atrasos de contenção, o ganho é sempre menor que 2. Então, mais importante que o número de processadores  $P$ , é o número de operações do programa que podem ser realizadas em paralelo.

### III.5 A IMPORTÂNCIA DA GRANULARIDADE

Se o fator  $f$  é considerado nulo nas expressões (III.5) e (III.6), o ganho e a eficiência do processo  $\rho_p$  podem ser escritos como:

$$S_{REL} = \frac{\beta * P}{(1+\Gamma) \lfloor \beta \rfloor} \quad (III.11)$$

$$E = \frac{\beta}{(1+\Gamma) \lfloor \beta \rfloor} \quad (III.12)$$

Para o caso ideal, onde a perda de tempo é

insignificante ( $\Gamma=0$ ), observa-se pela Fig.III.1 [6], que o ganho não é uma função linear do número de processadores  $P$  [6]. Este fato é atribuído à distribuição da carga de processamento entre todos os processadores, que está representado matematicamente pela taxa  $\beta / \lfloor \beta \rfloor$  nas expressões (III.11) e (III.12). Uma implementação eficiente tem que assegurar que todos os processadores iniciem e terminem no mesmo tempo ou, em outras palavras, que todos os processadores estejam sempre ocupados. Sempre que a carga de processamento não está igualmente distribuída, a eficiência do sistema está comprometida visto que o processador menos carregado tem que esperar para o mais ocupado deles terminar.

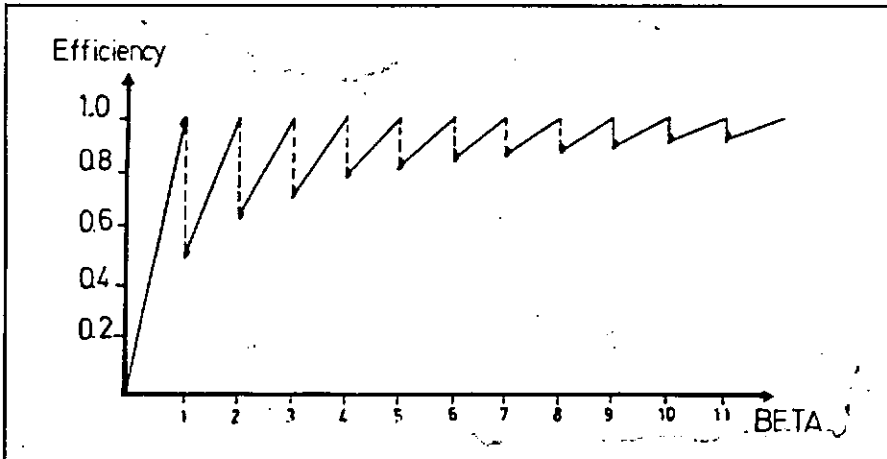


FIG.III.1 -  $E \times \beta$ , PARA O SISTEMA IDEAL

Para ilustrar esta discussão, considere o processo hipotético  $\rho_p$  para o qual a carga de processamento, representada pelo número  $M$ , será igualmente distribuída entre os  $P$  processadores, somente para o caso onde  $P$  é um fator de  $M$ . Quando isto por acaso acontecer ( $\beta = \lfloor \beta \rfloor$ ), todos processadores recebem o mesmo número de subprocessos

e um balanceamento perfeito de carga é alcançado. Quando esta condição não é satisfeita, entretanto, muitos processadores terão um subprocesso a mais e o balanceamento de carga será prejudicado.

Embora possa parecer óbvio, uma primeira regra quando do planejamento de programas paralelos é a impossibilidade de se beneficiar mais processadores do que a atual concorrência encontrada na aplicação. O processo  $\rho_p$ , por exemplo não pode tirar proveito de um número de processadores (P) maior do que o número de subprocessos disponíveis (M). Em outras palavras, se  $\beta \leq 1$  (ou  $M \leq P$ ), o ganho máximo é sempre igual a M.,

Para o caso onde  $\beta$  é maior do que 1, o ganho será igual a P somente quando for possível um balanceamento de carga perfeito. Caso contrário, o ganho será menor. Ainda, pela Fig.III.1, observa-se que quanto maior a granulometria, menor o efeito de balanceamento de carga sobre a eficiência. Tomar o número de processadores como um fator de M é o procedimento correto para se alcançar bons desempenhos, especialmente quando M é um número fixo.

O efeito da granularidade e do balanceamento de carga sobre a eficiência de programas paralelos mostram que, justamente para sistemas ideais, o desempenho final pode ser comprometido pela má implementação. A consideração deste fatores, junto com a Lei de Amdahl, é muito importante no projeto de qualquer aplicação, especialmente quando se considera que outras fontes de ineficiências também estarão presentes nos sistemas reais.

### III.6 PERDAS DE COMUNICAÇÃO

Os programas sequenciais têm geralmente a maioria dos dados alocados na memória central e, exceto para as tarefas de entrada e saída, o programador não necessita recorrer explicitamente para a alocação física de qualquer dado. A programação paralela requer muito mais atenção para o controle entre os dados e as várias unidades de processamento que operam sobre diferentes conjuntos de dados que têm de ser explicitamente distribuídos e monitorados pelo software. Estas operações extras obviamente representam perdas de tempo de processamento o qual em muitas máquinas são as principais fontes para o aparecimento de baixos desempenhos. As arquiteturas de memória compartilhada admitem todos os processadores acessarem os mesmos dados embora o acesso não possa, obviamente, ser feito ao mesmo tempo. A discussão desta seção está limitada, entretanto, à arquiteturas de memória distribuída e, seguimos o apresentado por [6].

As máquinas multi-processadores de mensagens passantes têm seus desempenhos inevitavelmente degenerado pela comunicação entre processadores. Quanto menor a necessidade de troca de informações entre processadores, tanto melhor o desempenho de todo o sistema. A quantidade de comunicação não é entretanto, a única variável do problema que depende ainda, tanto do software quanto da arquitetura do hardware. Em um sistema *Transputer*, por exemplo, existe uma perda de tempo do processador de cerca de 20 ciclos por processador ( ou aproximadamente  $1\mu s$  )

para a inicialização de cada comando de entrada e saída [6]. Devido à comunicação existente ser independente da CPU, esta perda depende pouco do tamanho da mensagem comunicada. Portanto, fazer uso de poucas, mas longas mensagens, será vantajoso e prudente. Todavia, a perda de eficiência devido à comunicação sempre estará presente e é representada pelo parâmetro  $\Gamma$  na expressão (III.12).

A sincronização é uma outra questão importante quando a perda de comunicação é considerada. Toda troca de dados envolve processos de entrada e de saída, os quais devem estar disponíveis para a realização da comunicação. Quando a sincronização destes processos não ocorrer, algum tempo é perdido e a eficiência do sistema é comprometida. Enquanto o hardware comumente assegura que o processo de entrada espera para o correspondente processo de saída tornar-se disponível ( e vice-versa ), o programador é capaz de evitar tal atraso, com a sincronização direta entre os vários processos. Uma fonte comum de falta de sincronização é a atribuição de diferentes cargas de processamento para dois processos de comunicação.

O hardware pode contribuir para o problema de sincronização: assumamos, por exemplo, que o tempo de funcionamento  $t_p$  para cada um dos subprocessos de  $\rho_p$  é menor que o tempo de entrada que a mensagem toma para ser recebida. Mesmo considerando que a entrada para o próximo subprocesso está sendo recebida enquanto a entrada do presente está sendo processada, o processador será lento para um período de tempo esperando pela entrada para terminar. Este fato contribui para o aumento da perda de

tempo e conseqüentemente diminuir o desempenho do sistema. Em sistemas *Transputers*, embora os *links* sejam sincronizados em hardware, dados de entrada podem por vezes, não estar disponíveis quando necessário, devido à baixa velocidade do *link*. Nestes casos, longas mensagens podem não ser a melhor solução e um compromisso tem que ser alcançado entre o número e o tamanho de mensagens [6].

## C A P Í T U L O   I V

### O AMBIENTE TRANSPUTER

#### IV.1 INTRODUÇÃO

O Transputer é um termo genérico utilizado para descrever uma família de dispositivos VLSI ( Very Large-Scale Integration ) que incluem, em um "superchip", controladores de disco, processadores de ponto flutuante, processador gráfico, dispositivo para processamento de sinal e processadores de 32 bits de uso geral [2,34]. A completa disposição dos componentes de um *Transputer* podem ser encontradas em [1,4]. Em termos de projeto, a implementação em hardware do *Transputer*, está baseada na filosofia do método " Communicating Sequential Process" (CSP) [35], onde o mecanismo de comunicação é baseado em trocas de mensagens diretamente pelos *links*. O processador *Transputer* foi especificado em conjunto com a linguagem paralela OCCAM [4,34], que também está baseada no método CSP. Redes de *Transputers* podem ser facilmente implantadas pela simples conexão dos processadores pelos seus *links*, sendo que a única limitação existente, é que cada processador não pode ser diretamente conectado a mais do que outros quatro, ou seja, as possíveis configurações estão restritas pelo número de *links*, e eventualmente, por limitações de sistemas particulares de hardware ( os limites de comutadores de *links*, por exemplo ).

O grande recurso dos *Transputers* realmente vem da flexibilidade e da facilidade de configuração do hardware. As dificuldades de implementação de programas em processadores *Transputers* deve-se sobretudo, a falta de suporte básico eficiente de software. Recentemente, um grande número de pacotes de software foi lançado no mercado, para suprir esta falha. Já estão disponíveis comercialmente, softwares tais como o compilador FORTRAN-Paralelo e o C-Paralelo. No entanto, a utilização de tais linguagens sequenciais adaptadas, normalmente reduz a eficiência do código, já que elas não possuem as mesmas características de programação paralela como as existentes na linguagem OCCAM [6].

Desta forma, os usuários interessados no uso destes processadores para a solução de um problema particular devem estar cientes dos desafios que deverão enfrentar: reescrever os algoritmos de modo paralelo, implementá-los de forma eficiente e depurá-los, além da necessidade de se escrever rotinas para distribuir e organizar dados, entre outros.

#### IV.2 O TRANSPUTER T800

O *Transputer* T800 [4] é um processador (lançado em meados de 1988) provido de unidade de ponto flutuante de 32 bits, suporte gráfico, um clock de 20 MHz, o que permite realizar operações de ponto flutuante a uma taxa de até 1,5 MFlops. Possui ainda 1 MBytes de memória RAM para processamento de alta velocidade e quatro links de

comunicação bidirecionais utilizados para transferir dados entre processadores a uma taxa de até 20 Mbytes/s que permitem a construção de redes de *Transputers* com simples interconexão entre os *links* [4].

#### IV.3 PROGRAMAÇÃO DO T800

O sistema computacional baseado no *Transputer* pode ser programado nas linguagens sequenciais de alto nível, tais como: FORTRAN, C, PASCAL e OCCAM. Independente da linguagem que será utilizada, a idéia fundamental na programação é definir processos, que são as unidades básicas do software. O sistema concorrente é então projetado a partir de um conjunto interconectado de processos, onde cada um deles pode ser considerado como uma unidade independente de projeto. A comunicação entre os processos é feita via canais de comunicação. Um canal pode somente transportar mensagens em uma direção. Caso a comunicação em ambas direções entre dois processos é requerida, dois canais devem ser usados.

Internamente, cada processo executa um certo número de ações. Uma ação pode, por sua vez, ser um conjunto de processos executados um após o outro, como numa linguagem de programação convencional; ou um conjunto de processos paralelos a serem executados todos ao mesmo tempo. O projeto do sistema é então hierarquicamente estruturado; a cada nível de projeto o projetista está preocupado apenas com um certo número de processos. A comunicação entre processos no *Transputer* é realizada de

forma síncrona, isto é, se um dado deve ser enviado de um processo para outro, a comunicação ocorre quando ambas estiverem disponíveis para se comunicar. O valor transmitido é então copiado do processo que envia para o processo que receba, através do canal de comunicação [1].

Com este tipo de comunicação, torna-se desnecessário o uso de qualquer mecanismo extra de sincronização entre os processos. Os conceitos de concorrência e comunicação podem ser implementados em um único ou em vários transputers. Como resultado, pode-se executar um programa paralelo em um *Transputer* individual ou numa rede de *Transputers*.

Quando o programa é executado num único *Transputer*, ele distribui seu tempo entre os processos concorrentes e o canal de comunicação é implementado pela movimentação de dados na memória (fig.IV.1).

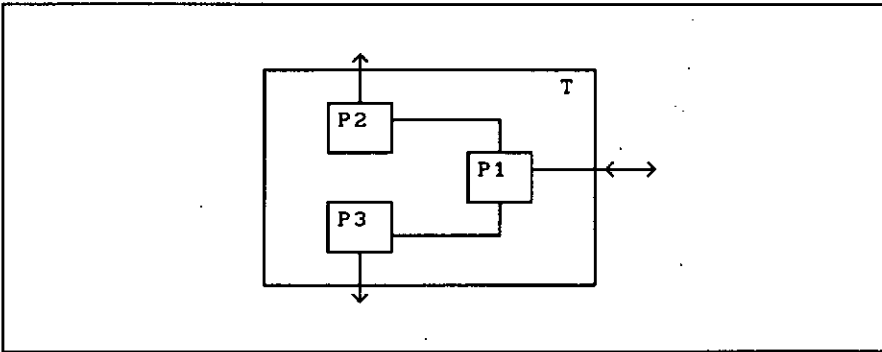


FIG.IV.1 - TRÊS PROCESSOS EM UM ÚNICO TRANSPUTER

Por outro lado, quando um programa é executado numa rede de *Transputers*, cada um deles executa o processo para o qual foi alocado. A comunicação entre diferentes *Transputers*, é implementada diretamente pelos canais de comunicação (fig.IV.2).

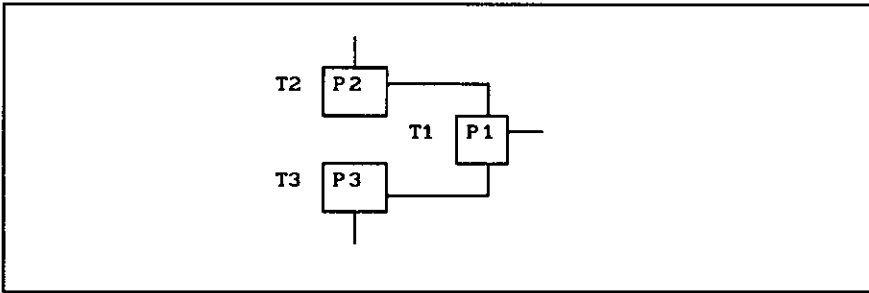


FIG. IV.2 - TRÊS PROCESSOS DISTRIBUÍDOS  
EM TRÊS TRANSPUTERS DISTINTOS

Desta forma concluímos que o mesmo programa pode ser implementado numa variedade de configurações dos *Transputers*, onde a configuração ótima é escolhida em função da necessidade do usuário: custo, desempenho, etc.

## C A P Í T U L O    V

## SOLUÇÃO DE GRANDES SISTEMAS DE EQUAÇÕES DO MEF

## V.1 - GENERALIDADES

O esforço computacional exigido para a resolução de sistemas de equações algébricas lineares do MEF representa em alguns casos até 80% do tempo total da análise. Assim, a sua implementação em multiprocessadores de alta velocidade, tem recebido considerável atenção dos engenheiros. A concorrência encontrada nos algoritmos varia em função de sua natureza e em geral não é uma tarefa fácil. Além disto, a estrutura de dados empregada para o algoritmo de solução, pode influenciar o desempenho global do sistema, devido à necessidade de transferência de dados entre os processadores. Assim, uma escolha cuidadosa do algoritmo de solução e de sua estrutura de dados é muito importante quando se procura maximizar o desempenho de programas paralelos.

Este capítulo está organizado em três seções. A primeira, apresenta o sistema computacional disponível para a realização deste trabalho. A segunda, é dedicada ao estudo da resolução sequencial de grandes sistemas de equações oriundas do MEF. A última seção será dedicada à discussão de um algoritmo paralelo específico para a fase de decomposição do esquema de Gauss para solução direta de sistemas de equações lineares.

Em estudos anteriores [11,14 e 18], foi verificado que

somente a fatoração deve ser efetuada em paralelo, principalmente em multiprocessadores de memória local, com comunicação do tipo mensagem-passante. O algoritmo aqui empregado baseia-se no modelo paralelo de solução simétrica proposto por FARAHT[10,11]. Inicialmente, o interesse foi para os sistemas de equações simétricos e esparsos que estão armazenados em uma estrutura de dados do tipo perfil [9]. Contudo, o esquema resultante aplica-se da mesma forma para sistemas não-simétricos e densos.

## V.2 - O SISTEMA COMPUTACIONAL UTILIZADO

A implementação computacional do algoritmo foi realizada em um sistema multiprocessador de memória distribuída. Este sistema é composto por quatro processadores montados em uma placa que é acoplada a um IBM-PC. Cada processador é baseado em um *chip* denominado *Transputer T800*, cujas principais características foram apresentadas no Capítulo IV. Um programa denominado servidor ("server") é quem permite que os *Transputers* sejam acessados a partir do PC. A interconexão física entre os processadores é mostrada na figura V.1. Nela, P1, P2, P3 e P4 são processadores do tipo *Transputer T800* e, as linhas de interconexão entre os blocos representam canais seriais bidirecionais de alta velocidade. Esta máquina, denominada "*Quadputer*", está instalada no Laboratório de Computação Paralela da COPPE. Atualmente, várias pesquisas aplicadas a soluções de problemas de grande porte, para processamento de alto desempenho, vêm sendo desenvolvidos tanto nos

Transputers, como no Hipercubo da Intel [28].

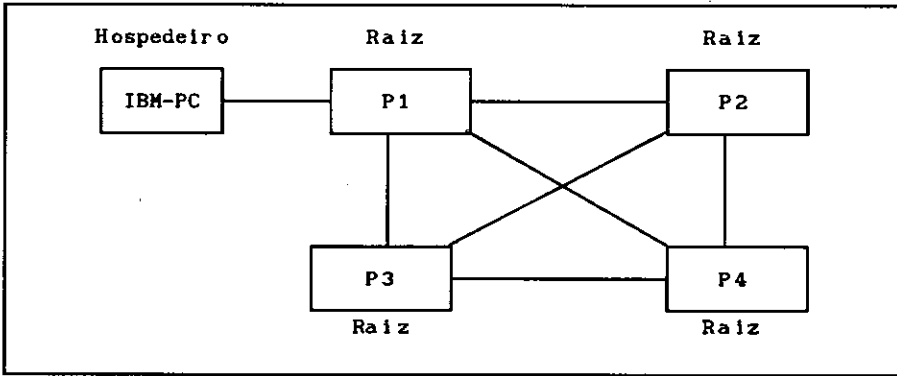


FIG.V.1 - INTERCONEXÃO ENTRE OS PROCESSADORES

### V.3 - ESTUDO PARA O CASO SEQUENCIAL

#### V.3.1 PRELIMINARES

Seja o sistema de equações lineares

$$K \cdot x = F \quad (V.1)$$

onde

$$K = \sum_{e=1}^{nel} K^e, \text{ é a matriz de rigidez "efetiva",}$$

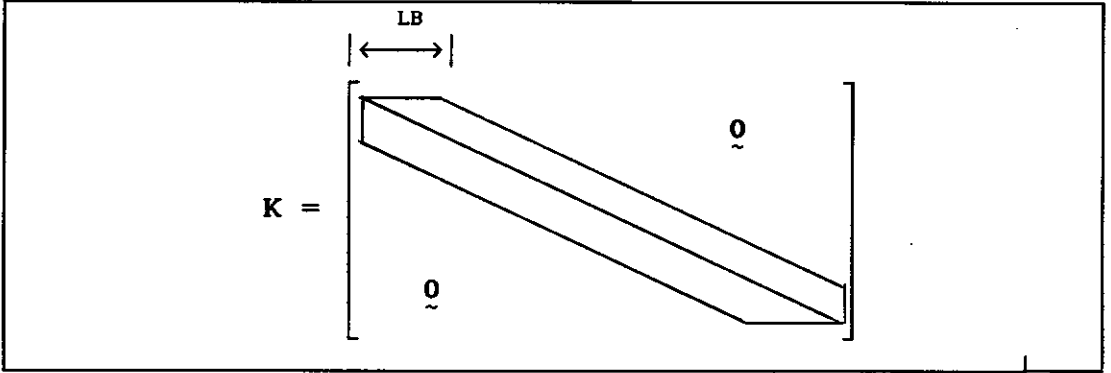
$$F = \sum_{e=1}^{nel} F^e, \text{ é o vetor de cargas "efetivo",}$$

e  $A$  é o operador de montagem;  $K^e$  e  $F^e$  são respectivamente as matrizes de rigidez e vetor de cargas para o  $e$ -ésimo elemento. Do ponto de vista computacional,  $K$  possui algumas propriedades que devem ser exploradas, visando um melhor desempenho computacional. Estas são:

a)  $K$  é Simétrica Positiva Definida (SPD),

$$[K_{ij}] = [K_{ji}];$$

- b)  $K$  é ESPARSA, já que muitos termos nas linhas (ou colunas) são nulos;
- c)  $K$  é em BANDA,  $LB$  = largura da banda;

FIG. V.2 - MATRIZ  $K$ 

Essencialmente, existem duas estratégias de solução: DIRETAS e ITERATIVAS. A maior diferença entre elas reside no fato que ao se adotar uma estratégia DIRETA o número de operações necessário para se obter a solução é conhecido a priori, e é função basicamente da ordem do sistema ( $n$ ), do  $LB$  e do grau de esparsidade do mesmo. Tal não ocorre com os métodos ITERATIVOS.

### V.3.2 MÉTODOS DIRETOS BASEADOS NA ELIMINAÇÃO DE GAUSS

O procedimento de Eliminação de Gauss consiste basicamente em subtrair no passo  $i$  uma sucessão de múltiplos da equação  $i$  das equações  $i+1, i+2, \dots, n$ , sendo  $i=1, 2, \dots, n-1$ . Desta forma, a matriz  $K$  será reduzida a uma forma triangular superior, a partir da qual é possível a determinação das incógnitas por RETROSUBSTITUIÇÃO. Todo o processo pode ser exemplificado pela solução do sistema apresentado no Quadro V.1[45].

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\left. \begin{array}{l} -(-4/5 \text{ EQ1}) + \text{EQ2} \\ -(1/5 \text{ EQ1}) + \text{EQ3} \end{array} \right\} \text{PASSO 1}$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & -16/5 & 29/5 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\left. \begin{array}{l} -(-16/14 \text{ EQ2}) + \text{EQ3} \\ -(5/14 \text{ EQ2}) + \text{EQ4} \end{array} \right\} \text{PASSO 2}$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & -20/7 & 65/14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 8/7 \\ -5/14 \end{bmatrix}$$

$$-(-20/15 \text{ EQ3}) + \text{EQ4} \quad \rightarrow \text{PASSO 3}$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 8/7 \\ 7/6 \end{bmatrix}$$

RETROSUBSTITUIÇÃO:

$$x_4 = 7/5$$

$$x_3 = \frac{8/7 - (-20/7) x_4}{15/7} = 12/5$$

$$x_2 = \frac{1 - (-16/5) x_3 - (1) x_4}{14/5} = 13/5$$

$$x_1 = \frac{0 - (-4) x_2 - (1) x_3 - (0) x_4}{5} = 8/5$$

Considerando as operações apresentadas no exemplo, a redução de  $K$  à forma triangular superior pode ser escrita como,

$$L_{n-1}^{-1} \dots L_2^{-1} L_1^{-1} K = S \quad (V.2)$$

onde  $S$  é a matriz triangular superior final e

$$L_1^{-1} = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & -l_{1+1,1} & 1 & & & \\ & -l_{1+2,1} & & 1 & & \\ & \vdots & & & \ddots & \\ & -l_{n1} & & & & 1 \end{bmatrix}; \quad -l_{1+i,1} = \frac{K_{1+j,1}^{(i)}}{K_{11}^{(i)}}$$

onde o superescrito (i) indica que um elemento da matriz  $L_{n-1}^{-1} \dots L_2^{-1} L_1^{-1} K$ , está sendo utilizado. Pelo exemplo a seguir, Quadro V.2, pode-se notar que a matriz inversa  $L_1$  pode ser obtida apenas trocando os sinais dos elementos fora da diagonal de  $L_1^{-1}$ .

$$L_1^{-1} = \begin{bmatrix} 1 & & \\ & 1 & \\ & -0.5 & 1 \end{bmatrix}; \quad L_1 = \begin{bmatrix} 1 & & \\ & 1 & \\ & 0.5 & 1 \end{bmatrix}$$

$$L_1^{-1} L_1 = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

QUADRO V.2

Portanto, pré-multiplicando-se a Equação (V.2) pelas respectivas inversas, tem-se:

$$K = L_1 L_2 \dots L_{n-1} S \quad (V.3)$$

onde,



$$1^a) \quad L \quad V = F \quad (V.9)$$

$$2^a) \quad D \quad L^T \quad d = V \quad (V.10)$$

Na prática, observa-se que  $L$  e  $V$  são formados modificando-se diretamente  $K$  e  $F$ .

### V.3.3 FORMAS DE ARMAZENAMENTO DE $K$

#### a) BANDA

É a forma de armazenamento mais antigo e utilizada a mais tempo, sendo também a mais simples. Armazena-se apenas o triângulo superior de  $K$ , conforme mostrado na Figura V.3.

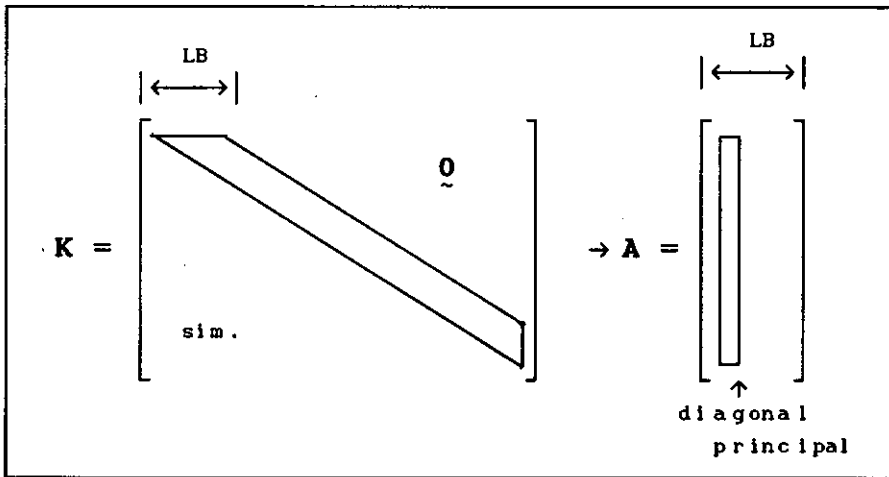


FIG. V.3 - ARMAZENAMENTO EM BANDA.

Pode-se mostrar que o esforço computacional para a resolução do sistema neste caso é proporcional ao produto  $n \cdot LB^2$ . Neste esquema de armazenamento, a minimização da largura de banda é fundamental (conforme o exemplo da FIG.V.4). Para tanto, podem ser utilizados algoritmos de reordenação nodal, como aquele apresentado por Cuthill-McKee[48]. Em algumas bibliotecas de "software" encontram-se rotinas extremamente otimizadas baseadas neste esquema de armazenamento. Por exemplo, podem-se citar os



assim, ao se utilizar métodos de armazenamento esparsos, é de importância fundamental uma reordenação das incógnitas, não para minimizar a banda, mas sim para minimizar o "fill-in". Estes métodos ainda não tem uma grande aceitação na comunidade de EF, apesar de existir um grande esforço de pesquisa sendo efetuado para adequar estes métodos às particularidades do MEF. Existem "softwares" disponíveis nas bibliotecas citadas anteriormente para este tipo de armazenamento. Entretanto, o esquema mais divulgado é aquele encontrado no SPARSEPACK, cuja documentação completa, incluída a teoria, é dada por GEORGE e LIU[51].

### c) MÉTODO DA COLUNA ATIVA

Seja a matriz de rigidez  $K$  apresentada na Figura V.6 e armazenada em um único arranjo  $A$  conforme a figura V.7. O arranjo  $MAXA = [1,2,4,6,10,12,16]$  é o ponteiro das diagonais. De forma que  $MAXA(I+1)-MAXA(I)$  fornece as alturas efetivas das colunas de  $K$ . Este método de armazenamento é o mais popular no MEF. Encontram-se na literatura ( BATHE[45], HUGHES[46] ) diversas implementações deste esquema.

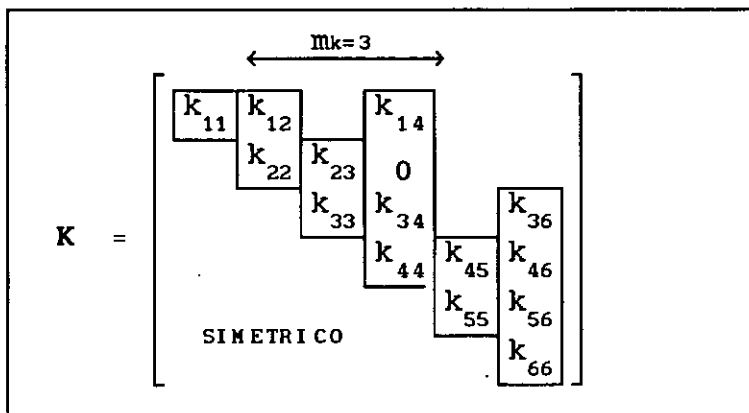


FIG. V. 6



### V.3.4 DECOMPOSIÇÃO DE GAUSS NO MÉTODO DA COLUNA ATIVA

Usando  $d_{11} = k_{11}$ , o algoritmo para o cálculo dos elementos  $l_{ij}$  e  $d_{jj}$  na  $j$ -ésima coluna é mostrado na Figura V.9.

```

para j = 2, ... , n
  para i = mj+1, ... , j-1
    gmj, j = kmj, j
    gij = kij - ∑r=mmi-1 lri grj
  fim i
  para i = mj, ... , j-1
    lij = gij / d11
  fim i
  djj = kjj - ∑r=mjj-1 lrj grj
fim j

```

FIG.V.9

Nesta figura  $m_j$ ,  $m_m$  e as alturas efetivas de colunas são dadas respectivamente por:

$m_j$  = número do 1.<sup>o</sup> elemento não-nulo da coluna  $j$

$m_m = \max(m_1, m_j)$

$(i - m_i)$  = alturas efetivas da coluna.

Com relação ao algoritmo é ainda importante salientar:

- Os somatórios não envolvem multiplicação por zeros fora das colunas ativas.
- Os elementos  $l_{ij}$  pertencem de fato a  $L^T$ , ao invés de  $L$ .
- Considerando área de armazenamento, os elementos  $l_{ij}$  são

calculados e armazenados sobre  $g_{ij}$ , isto é,  $l_{ij} \leftarrow g_{ij} / d_{ii}$ .  
Sendo assim,  $d_{jj}$  é armazenado sobre  $k_{jj}$ ,

$$( k_{jj} \leftarrow k_{jj} - \sum_{r=m_j}^{j-1} l_{rj} g_{rj} ).$$

Ou seja, ao fim do algoritmo os elementos da diagonal armazenam os  $d_{jj}$  e os  $l_{rj}$ .

### V.3.5 REDUÇÃO DO VETOR DE CARGAS

Após avaliados a decomposição, podemos reduzir o vetor de cargas conforme o algoritmo dado na Figura V.10.

$V_1 = F_1$ <p>para <math>i = 2, \dots, n</math></p> $V_i = F_i - \sum_{r=m_i}^{i-1} l_{ri} V_r$ <p>fim <math>i</math></p>
--

FIG. V. 10

Deve-se observar que a redução do vetor de cargas pode ser efetuada concomitantemente com a obtenção da decomposição.

### V.3.6 RETROSUBSTITUIÇÃO

A retrosubstituição segue o esquema da fig.V.11:

```

 $\bar{V} = D^{-1}V$ 

 $x_n = \bar{V}_n^{(n)}$ 
para i = n, ..., 2
  para r = m_1, ..., i-1
     $\bar{V}_r^{(i-1)} = \bar{V}_r^{(i)} - l_{r1} x_1$ 
  fim r
   $x_{i-1} = \bar{V}_{i-1}^{(i-1)}$ 
fim i

```

FIG. V.11

onde o sobrescrito (i-1) indica que este elemento é calculado na avaliação de  $x_{i-1}$ . Com respeito a área, deve-se notar que  $\bar{V}_k^{(j)}$ ,  $V_j$  é armazenado na mesma posição que  $V_k$ , isto é, a posição original de  $F_k$ .

### V.3.7 PRECISÃO NA ELIMINAÇÃO DE GAUSS

Pode-se mostrar que quanto maior for o número de condicionamento espectral da matriz de rigidez,  $\text{cond}(k) = \lambda_{\max} / \lambda_{\min}$ , onde  $\lambda_{\max}$  e  $\lambda_{\min}$  são respectivamente o maior e o menor autovalor de  $K$ , maiores serão os erros de arredondamento durante a decomposição, BATHE[45]. Uma maneira de estimar o número de dígitos perdidos durante a decomposição é através do procedimento da fig.V.12.

```

para   i = 1, ... ,n
       $\bar{d}_{11} = d_{11}$ 

      DECOMPOSIÇÃO

      DLOST =  $\log_{10} ( \bar{d}_{11} / d_{11} )$ 
      se   4 < DLOST ≤ 12   então
            ADVERTÊNCIA, i , DLOST
      fim se
      se   DLOST > 12   então
            ADVERTÊNCIA, i , DLOST
            PARE
      fim se
fim i

```

FIG. V.12

Este mecanismo interrompe a fatoração caso seja detectado um número de dígitos perdidos (DLOST) maior que 12 ( dependente da máquina ).

### V.3.8 COMENTÁRIOS FINAIS

Devido aos endereçamentos indiretos que ocorrem na implementação computacional, o único loop VETORIZÁVEL no processo de decomposição de Gauss é o produto escalar em

$$g_{ij} = k_{ij} - \sum_{r=m}^{i-1} l_{ri} \cdot g_{rj}$$

Entretanto, através da reorganização do código, é possível

vetorizar TODOS os loops do algoritmo, incluindo a redução do vetor de cargas e a retrosubstituição, aumentando consideravelmente a eficiência computacional. Tal fato foi explorado por LOZUPONE[15] que desenvolveu uma implementação voltada para os computadores IBM 3090. Uma versão vetorizada da rotina COLSOL ( BATHE[45] ), desenvolvido na COPPE segundo os comentários anteriores é empregada atualmente em alguns códigos de produção, tais como o ANFLEX[52].

#### V.4 - ESTUDO PARA O CASO PARALELO

##### V.4.1 - A DECOMPOSIÇÃO DE CROUT

O algoritmo de fatoração exposto anteriormente para computadores sequenciais pode ser sumarizado como a seguir:

$$A = LDL^T = LU \quad (V.11)$$

no qual a  $j$ -ésima coluna de  $U$  é calculada por:

$$U_{1j} = A_{1j} - \sum_{k=1}^{j-1} L_{1k} U_{kj} \quad j > i \quad (V.12)$$

e a  $j$ -ésima linha de  $L$  é dada por:

$$L_{j1} = U_{1j} / D_{11} \quad (V.13)$$

e os termos diagonais  $U_{11}$  e  $L_{11}$  são idênticos uma vez que  $L_{11}$  é normalizado como sendo igual à unidade.

De forma esquemática, em computadores sequenciais programados em FORTRAN, tem sido mais conveniente calcular  $U_{1j}$  modo-coluna, com  $i=1$  a  $j$ . Após cada coluna ser completada,  $L_{j1}$  é calculada modo-linha com  $i=1$  a  $j-1$  e

armazenada na forma transposta como  $L_{ij}^T$  onde  $U_{ij}$  estava previamente alocada. Os termos diagonais  $U_{ii}$  ou  $D_{ii}$  mantêm-se na mesma alocação que  $D_{ii}$ . Assim, pode-se observar que os componentes da matriz  $L$  são calculados sequencialmente, coluna por coluna, começando com a primeira coluna (fig.V.13).

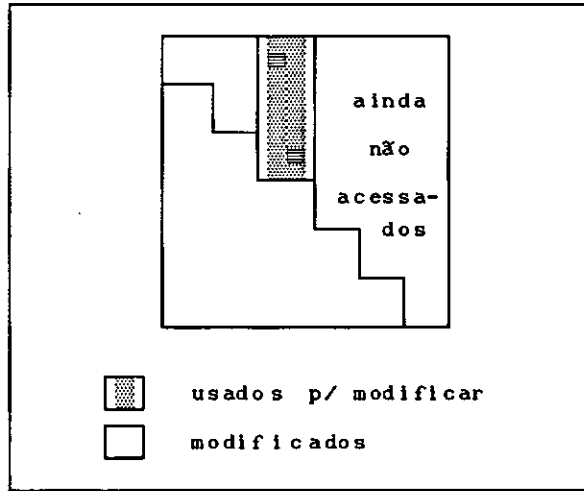


FIG. V.13

Este procedimento mostra que uma coluna só poderá ser calculada, após o cálculo prévio de todas as colunas anteriores. Em contraste com o sequencial, o mais conveniente aos computadores paralelos, será calcular  $U_{ij}$  modo-linha, dentro de uma estrutura de dados orientada por coluna (fig.V.7). Os elementos  $L_{ji} = U_{ij} / D_{ii}$  nunca estão explicitamente armazenados.

As figuras V.14 e V.15 mostram os coeficientes que influenciam a obtenção de um termo na diagonal principal e fora dela, respectivamente.



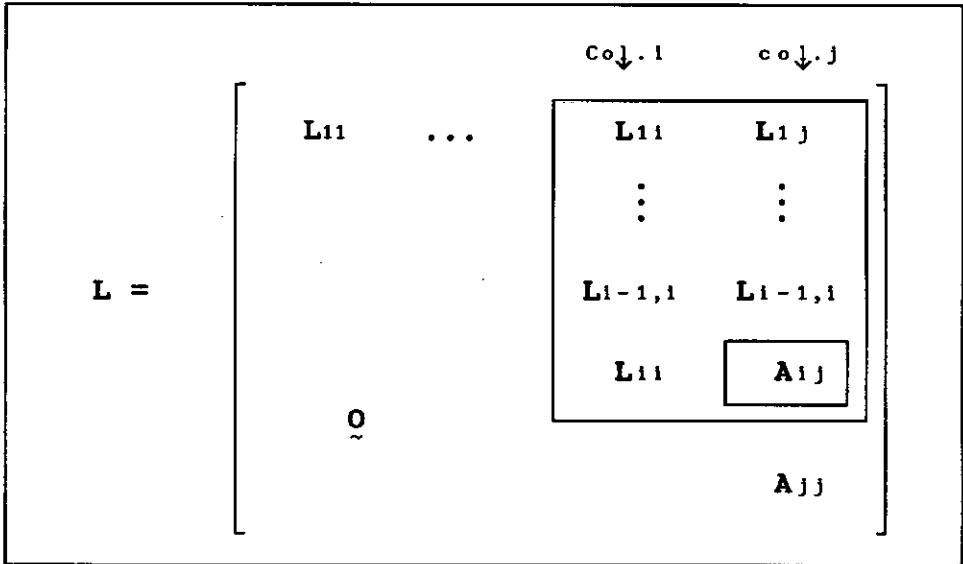


FIG. V. 15

Observando o grafo das dependências dos coeficientes da matriz de rigidez, mostrado na fig.V.16, o novo algoritmo pode ser descrito como tendo basicamente duas partes distintas dentro de um laço mestre, como descrito no Quadro V.3.

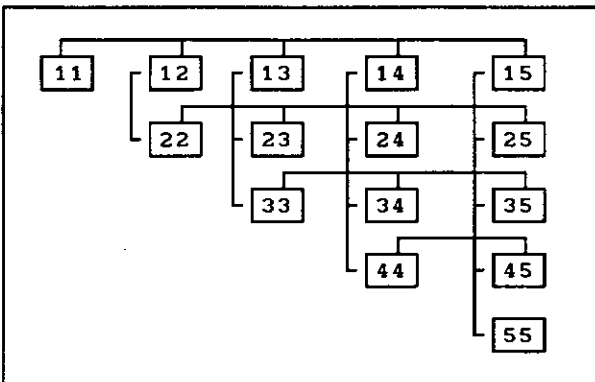


FIG. V. 16

DO  $k = 2, n$

- (1) armazena informação dentro de um vetor de trabalho;
- (2) calcula um conjunto de  $(n-K+1)$  vetores por produtos internos independentes;

ENDDO;

Isto permite um processamento vetorial dentro de cada processo concorrente. Portanto, ele exhibe dois níveis de paralelismo: concorrência no laço externo e pipelining nos laços mais internos.

Concluindo, para cada passo  $k$ , a parte superior da coluna  $k$ , bem como a parte à direita da linha  $k$ , serão completadas, como ilustrado na figura V.17. Esta descreve a designação de tarefas e dados a cada processador. Para cada passo  $k$ , a redução da linha  $k$  é completada por  $(n-k+1)$  tarefas independentes. Portanto, elas podem ser executadas concorrentemente por  $N_p$  processadores disponíveis.

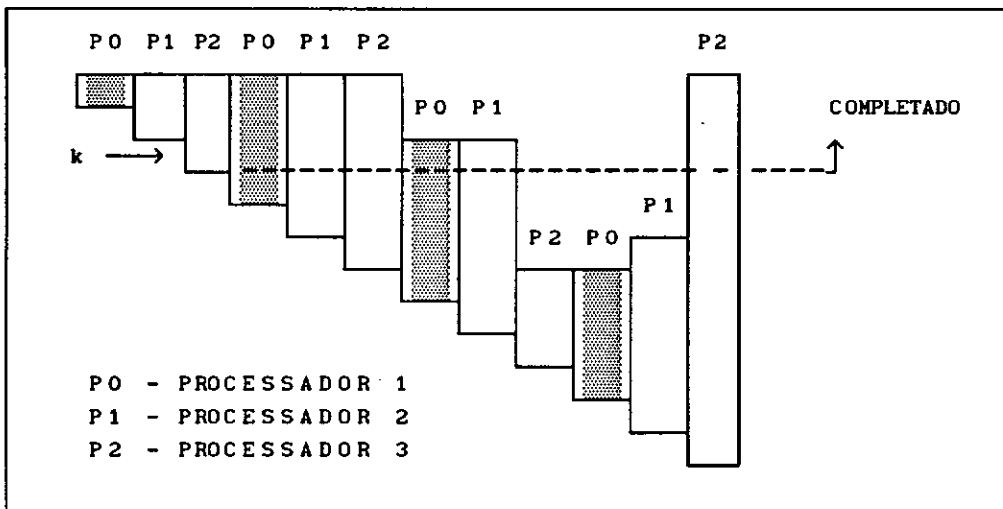


FIG. V.17 - DESIGNAÇÃO DAS COLUNAS A 3 PROCESSADORES

#### V.4.2 - FASE DE SUBSTITUIÇÕES

A substituição progressiva e a retrossubstituição são as soluções para, respectivamente, o sistema triangular inferior  $LV = F$ , e o sistema triangular superior  $DL^T d = V$ . Estas soluções possuem um paralelismo inerente que pode ser explorado por uma estratégia de varredura de coluna. Se

$T(t_{ij})$  denota a matriz triangular ( L ou  $DL^T$  ) do sistema, definimos as equações como a seguir (QUADRO V.4) :

```

DO k = 1, 2, . . . , n
  dk = Fk / tkk
  DO i = k+1, n
    Fi = Fi - dk tik
  ENDDO
ENDDO

```

QUADRO V.4

Desta forma, as iterações do laço interno são independentes. Assim, eles podem ser executados concorrentemente pelos  $N_p$  processadores alocados. Este esquema explora a exparsidade na solução d. Se  $F_k$  resultar zero no início do k-ésimo passo,  $d_k$  é zero e, todo o passo pode ser omitido.

### V.4.3 - ESTRATÉGIAS DE PARALELIZAÇÃO

De acordo com o que foi exposto acima, a distribuição de uma matriz A em uma implementação em paralelo, será feita por colunas, uma vez que, a comunicação entre os processadores pode ser minimizada. Usando tal estrutura de dados, os cálculos de todos os componentes de uma linha podem ser executados em paralelo, requerendo apenas a distribuição da coluna aos demais processadores. Ainda, devido a variação do comprimento de cada coluna, a cada processador será designado um número de colunas locais, cuja distribuição será realizada de acordo com a figura

V.17. Esta designação, distribui o armazenamento uniformemente entre os processadores. Além do mais, isto garante um balanceamento da carga computacional entre eles, durante todos os passos do esquema paralelo, principalmente para multiprocessadores com memória local [6,11].

#### V.4.4 - IMPLEMENTAÇÃO COMPUTACIONAL

Neste item, será abordada a implementação computacional da solução paralela, bem como a interação com a arquitetura concorrente. É apresentada uma solução paralela relativamente simples que pode ser integrada a códigos já existentes. Basicamente, somente a rotina de solução sequencial precisa ser extraída e substituída pelo algoritmo paralelo aqui proposto.

##### a) Distribuição das colunas:

A coluna  $j$  da matriz  $A$ , dita  $A_j$ , é armazenada no processador  $P_j = \text{mod}(j-1, N_p)$ . Isto garante a distribuição balanceada da carga computacional. Em cada processador  $P_j$ , três variáveis locais auxiliam este armazenamento. O vetor real COLVAL armazena sequencialmente os elementos das colunas de  $A$  que estão destinados para o processador. O vetor inteiro COLPOS de comprimento igual ao número de colunas designadas ao respectivo processador mais uma unidade (  $NCOLPN+1$  ), aloca a posição de cada coluna local de COLVAL. Sendo local a estrutura de dados designada a



## b) Decomposição do tipo LDL<sup>T</sup> :

A partir da equação V.10, podemos notar que a cada passo, os processadores necessitam trocar informações entre si para poderem prosseguir com seus cálculos de forma correta, significando portanto, que os processadores são obrigados a aguardarem uns pelos outros, durante a execução do algoritmo.

Para cada passo  $k$  da fase de fatoração, o processador root (processador que contém a coluna  $A_k$  de  $A$ ) atualiza a coluna ativa e a armazena em um vetor RBUFF que será transmitido aos demais processadores. Assim, para cada passo  $k$ , um processador diferente torna-se "root", e isto requer que os termos diagonais sejam residentes em cada processador. Portanto, um array diagonal PIVOT de comprimento NEQ é declarado em cada processador. Para cada passo  $k$ , ele é atualizado pelo processador root com o mais recente pivot  $D_{kk}$  computado. Para minimizar o número de comunicações entre os processadores, apenas uma mensagem é enviada a cada passo da fase de decomposição, para cada processador. Isto significa que  $P-1$  transmissões precisam ser executadas pelo processador root. Após o recebimento de RBUFF, cada processador  $P_j$  executa concorrentemente com todos os outros processadores  $(n-k+1)/P$  tarefas correspondentes às colunas  $A_j$  que estão destinadas a  $P_j$ .

O Quadro V.5 descreve a implementação paralela da fase de decomposição, que deverá ser designada a cada processador do sistema. Pode-se notar que este algoritmo consiste basicamente em atualizar o elemento diagonal e

transferir a coluna ativa para os demais processadores, a fim de que estes terminem os seus cálculos.

```

PROC #
PARA K = 1, NEQ  FAÇA
    ENCONTRA O PROCESSADOR ROOT
    NROOT = MOD(K-1,Np)
    SE É ROOT ( PROC # .EQ. NROOT )
        . ATUALIZA O ELEMENTO  DIAGONAL;
        . ENVIA A COLUNA ATIVA PARA TODOS
          OS OUTROS PROCESSADORES;
    SENÃO
        . RECEBE MENSAGEM DO  ROOT;
        . ATUALIZA TODOS OS SEUS ELEMENTOS
          LOCALIZADOS NA LINHA K;
FINAL

```

QUADRO V.5

### c) Agrupamento dos dados:

Após a fase de fatoração ser concluída, a estrutura de dados de  $A$  é reescrita com os valores obtidos para os  $DL^T$  e, a próxima fase, nominalmente denominada por substituição progressiva, será executada sequencialmente devido a sua pouca representatividade em relação ao ganho total.

## CAPÍTULO VI

### APLICAÇÕES NUMÉRICAS

Este capítulo apresenta os resultados obtidos pelo algoritmo proposto no capítulo anterior. Pressupõe-se que cada processador executa o mesmo código e que a matriz global já está distribuída entre eles, ou seja, cada um dos processadores tem armazenado em sua memória local, os vetores de trabalho (COLVAL, COLPOS e COLTIP). Como os processadores não compartilham memória, eles se comunicam através de trocas de mensagens, pelas ligações existentes entre si. Dessa forma, a Seção VI.1 apresenta as formas de ligações utilizadas no decorrer deste trabalho e também, apresenta como foi obtido os valores de Ganho e Eficiência, bem como as tomadas de tempo de comunicação e de CPU. A Seção VI.2 apresenta os resultados numéricos obtidos após submeter o algoritmo a situações diversas. Três estruturas distintas (membrana, viga e anel) foram escolhidas para este propósito. A razão da escolha destas, foi em função das características apresentadas pelas matrizes de rigidez oriundas da análise pelo MEF das mesmas, conforme mostrado na Fig.VI.1. Desta figura, observa-se claramente a distinção para cada classe de problema, pois, enquanto a matriz da figura VI.1.a possui uma largura de banda LB alta, para o caso VI.1.b, LB é muito menor e, para o caso VI.1.c, há o aparecimento de uma altura de coluna grande, devido ao acoplamento dos elementos finais com os iniciais.

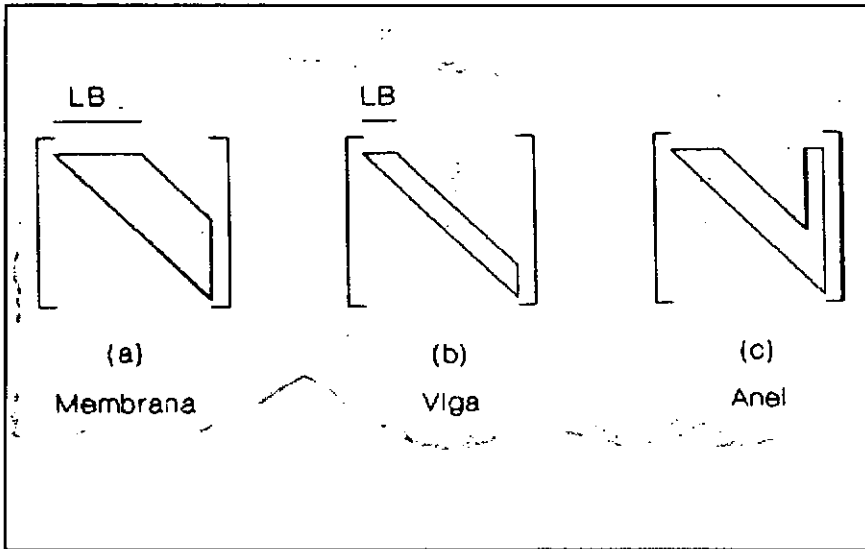


FIG. VI.1 - CARACTERÍSTICAS DAS MATRIZES

## VI.1 - GENERALIDADES

### VI.1.1 LIGAÇÕES ENTRE OS PROCESSADORES

O sistema computacional disponível para a implementação do código paralelo foi descrito em V.2. A partir do *Quadputer*, os processadores foram ligados de maneiras distintas, conforme a figura VI.2.

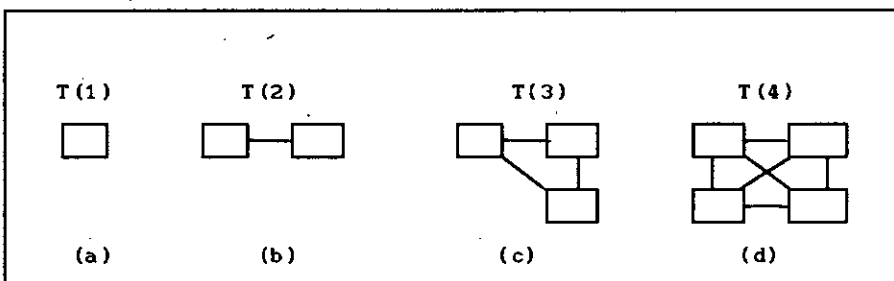


FIG. VI.2 - CONEXÃO ENTRE OS PROCESSADORES

Nesta figura,  $T(N)$  representa a tomada de tempo para  $N$

processadores que atuam concorrentemente na fatoração. Temos ainda, uma outra situação, mostrada na figura VI.3.

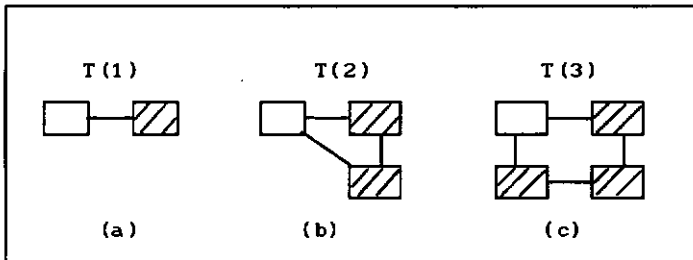


FIG. VI.3 - CONEXÃO ENTRE OS PROCESSADORES  
COM PROCESSADOR GERENTE

Nesta figura, apenas os processadores hachurados estão trabalhando na fase de fatoração; por exemplo, no caso (c), apenas 3 processadores executam concorrentemente a mesma tarefa. O processador ocioso (não hachurado) gerencia os demais, ficando com ele a responsabilidade de no início, distribuir os dados aos demais, e ao final da fatoração, receber os dados atualizados de cada processador.

### VI.1.2 CÁLCULO DO GANHO (S) PARA AS DIFERENTES LIGAÇÕES

Conforme visto anteriormente no ítem III.2,  $T_{seq}$  pode ser o tempo necessário para a execução do código paralelo em apenas um processador. Assim,  $S=T(1)/T(N)$ , dá uma boa medida de como um algoritmo particular foi paralelizado. Com o auxílio do pacote CHAN.TIMER disponível no compilador FORTRAN Paralelo,  $T(N)$  foi adotado como sendo o tempo de execução do processador mais lento, após a tomada de tempo dentre todos os processadores envolvidos na fatoração. Para todos os exemplos foram obtidos resultados satisfatórios de

S, demonstrando a paralelização eficiente do algoritmo.

### VI.1.3 CÁLCULO DO GANHO ( $S'$ ) PARA AS DIFERENTES LIGAÇÕES

De III.2, se tomarmos  $T_{seq}$  como o tempo necessário pelo melhor algoritmo serial existente,  $S' = T_{colsol}/T(N)$ , onde  $T_{colsol}$  refere-se ao código dado em [52]. Assim,  $S'$  dá uma medida absoluta de como um algoritmo paralelo pode funcionar relativamente a um sequencial. Para os diversos exemplos rodados, foi observado que nem sempre foi possível obter os mesmos valores de  $S$ , demonstrando que apesar da paralelização eficiente do código (obtidos em  $S$ ), bons resultados para  $S'$ , são alcançados apenas para determinadas classes de estruturas, conforme será mostrado em VI.2.

Após a execução de diversos exemplos, foi observado em todos, que o tempo decorrido pelo sistema em que há um processador gerente é mais lento do que quando todos os processadores estão trabalhando concorrentemente numa mesma tarefa. A situação em que ocorre ociosidade de um processador, mascara os resultados finais do desempenho do algoritmo, podendo chegar a resultados "quase" ideais de Ganho e Eficiência.

### VI.1.4 CÁLCULO DO TEMPO DE COMUNICAÇÃO E DE COMPUTAÇÃO

Conforme visto anteriormente no Ítem III.6, a comunicação entre os processadores é um fator importante ao avaliar o desempenho de um algoritmo paralelo. Assim sendo, o tempo de comunicação ( $T_{com}$ ), ou seja, o tempo necessário

para a realização das trocas de informações entre os processadores durante a fase de fatoração, e o tempo necessário para a realização dos cálculos ( $T_{cpu}$ ), foram obtidos da seguinte forma: para o caso em que apenas um processador trabalha,  $T_{cpu} = T(1)$ , pois não há trocas de informações; mas para  $N$  diferente de 1, o tempo de CPU será igual a  $T(N) - T_{com}$ , onde o  $T_{com}$  foi tomado como:  $T_{com} = NEQ * (N-1) * T_{LBM}$  e  $T_{LBM}$  é o tempo necessário para se transferir um vetor com a dimensão da largura de banda média. Melhor dizendo,  $T_{com}$  é o tempo que cada processador leva para enviar  $NEQ$  mensagens para os demais, onde  $NEQ$  é o número de equações do problema em questão.

## VI.2 - RESULTADOS NUMÉRICOS

Apresentaremos diversos exemplos, para várias discretizações, até a máxima capacidade de memória disponível em um *Transputer*, de forma a permitir em todos os casos, a medida de  $T(1)$  e de  $T_{colsol}$ . Cabe salientar aqui, que exemplos de maior dimensão poderiam ser analisados, utilizando toda a memória disponível nos diferentes processadores.

### VI.2.1 Membrana

O primeiro exemplo analisado é o de uma membrana quadrada, com 100mm de lado e espessura unitária, sob condições de estado plano de tensões, sujeita a quatro

cargas nos seus cantos, de  $25.0 \times 10^5 \text{ N}$ , aplicadas na direção das diagonais principais do domínio (fig.VI.4).

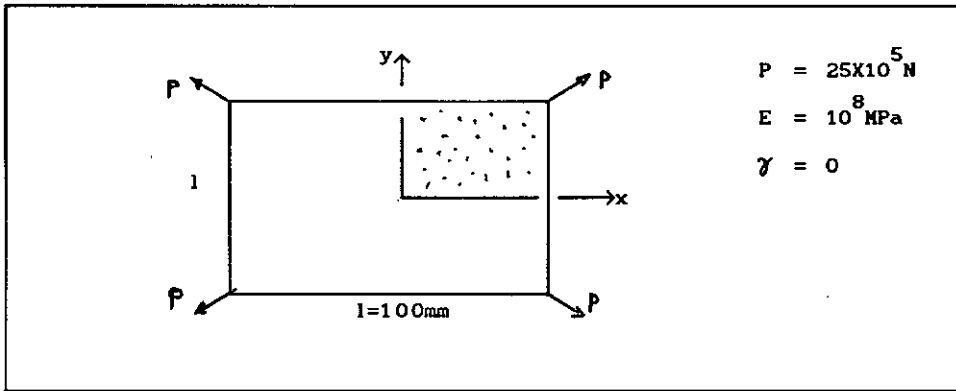


FIG. VI.4 - MEMBRANA

O material da membrana é linear e isotrópico com Módulo de Young igual a  $10^8 \text{ MPa}$  e coeficiente de Poisson nulo. Considerando a dupla simetria, um quarto de membrana foi discretizado por malhas uniformes de elementos quadriláteros isoparamétricos de 4 nós. Os experimentos numéricos foram analisados utilizando-se uma sequência de malhas com as características topológicas dadas pela tabela VI.1.

MALHA	NEL	NNOS	NEQ	NWK	LB	LBM
1	10x10	121	220	4978	25	22
2	17x17	324	612	22506	39	36
3	20x20	441	924	35958	45	42
4	27x27	784	1512	85966	59	56
5	32x32	1089	2112	141247	69	66
6	40x40	1681	3280	271918	85	82
7	44x44	2025	3960	360006	93	90

TAB. VI.1 - CARACTERÍSTICAS TOPOLÓGICAS

Nesta tabela o número de elementos da membrana é NEL, os correspondentes números de nós e o número de equações são denotados respectivamente por NNOS e NEQ. Também está incluída nesta tabela, o número de elementos da matriz de rigidez sob a coluna ativa(NWK), bem como a correspondente largura de banda LB e a largura de banda média (LBM).

As tomadas de tempo estão apresentadas na Tabela VI.2.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)	T(4)
1	220	0,28	0,44	0,42	0,33	0,27
2	612	1,89	2,91	2,54	1,92	1,61
3	924	3,44	5,32	4,52	3,36	2,82
4	1512	10,50	16,3	12,8	9,30	7,80
5	2112	20,10	30,9	24,4	17,2	14,2
6	3280	56,00	71,2	54,1	38,8	30,2
7	3960	80,90	102,5	77,7	55,4	42,9

TAB.VI.2 - TOMADAS DE TEMPO

Nesta tabela, ao confrontarmos o código serial mais rápido com o código paralelo em quatro processadores (  $T_{colsol} \times T(4)$  ), vemos que o código paralelo é mais rápido. Mas, para  $T_{colsol} \times T(3)$  e  $T_{colsol} \times T(2)$ , vemos que o código paralelo é o mais rápido apenas para determinados valores de NEQ. Por exemplo, para três processadores, o código paralelo só prevalece para números a partir de 924 equações e para dois processadores, este número sobe para 3280 equações. Ainda em relação a esta tabela, vemos que  $T_{colsol}$  é sempre menor do que  $T(1)$ , pois enquanto o código COLSOL trabalha por colunas, os

coeficientes da matriz, o código paralelo quando executado em apenas um processador, o faz por linhas, tornando-se mais lento.

Com relação à paralelização do código em relação ao serial mais rápido, constatamos pela tabela VI.3 que para a malha mais refinada, chegamos a valores de ganho ( $S'$ ) e eficiência ( $E'$ ), próximos de 2 e 50% respectivamente, com tempo de comunicação ( $T_{com}$ ), conforme mostrado na tabela VI.4, em torno de 20% do tempo total de processamento. Estes valores validam a utilização do código paralelo em problemas semelhantes a estes.

MALHA	S'2	S'3	S'4	E'2 %	E'3 %	E'4 %
1	-	-	1,04	-	-	26
2	-	-	1,17	-	-	29,3
3	-	1,02	1,23	-	34	30,8
4	-	1,13	1,35	-	37,6	33,8
5	-	1,17	1,42	-	39	35,4
6	1,04	1,44	1,86	52	48,1	46,4
7	1,04	1,46	1,89	52	48,6	47,2

TAB. VI.3 -  $S' = T_{colsol} / T(N)$  e  $E' = S' / N$

MALHA	2PROC		3PROC		4PROC	
	Tcom	Tcpu	Tcom	Tcpu	Tcom	Tcpu
1	9,5	90,5	27,3	72,7	48,1	51,9
2	7,9	92,9	21,4	78,6	38	62
3	7,2	92,8	21,1	78,9	37,2	62,9
4	6,0	94,0	16,7	83,3	29,7	70,3
5	4,4	95,6	12,6	87,4	23	77
6	4,3	95,7	12	88	23	77
7	3,6	96,4	10	90	19,4	81,6

TAB.VI.4 - Tcom(%) e Tcpu(%)

Quanto à paralelização do código em relação a ele mesmo, os resultados obtidos encontram-se na Tab.VI.5. Os valores de ganho (S) e eficiência (E), são próximos à 2,5 e 60% ,respectivamente, para o caso da malha mais refinada utilizando-se quatro processadores. Estes valores também demonstram a paralelização eficiente do código.

MALHA	S2	S3	S4	E2 %	E3 %	E4 %
1	1,05	1,33	1,63	52,5	44,3	40,8
2	1,15	1,52	1,81	57,5	50,7	45,3
3	1,18	1,75	2,09	58,8	58,2	52,2
4	1,27	1,75	2,09	63,4	58,3	52,3
5	1,27	1,80	2,20	63,5	60,0	54,4
6	1,32	1,84	2,36	66	61,3	59
7	1,32	1,85	2,39	66	61,7	59,8

TAB.VI.5 -  $S=T(1)/T(N)$  e  $E=S/N$

De forma a ilustrar o ganho e a eficiência obtidos a partir do esquema de ligação com processador gerente (fig.VI.3), a tabela VI.2.a, apresenta as tomadas de tempo para as malhas de número quatro e cinco.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)
4	1512	18,70	24,8	13,1	9,40
5	2112	34,30	46,9	24,5	17,3

TAB. VI.2.a - TEMPOS

Ao compararmos os valores desta tabela com os da tab.VI.2, notamos que os resultados de T(1) e T(2) estão bem próximos. A disparidade ocorre nos resultados obtidos para Tcolsol e T(1). Vemos que com o uso do processador gerente, eles estão bem maiores. Como o ganho e a eficiência são diretamente proporcionais a eles, encontramos resultados ( tab.VI.2.b ) bem próximos da situação ideal. Quanto aos tempos de comunicação, eles são idênticos aos da tabela VI.4.

MALHA	S2	S3	E2 %	E3 %
4	1,97	2,66	98,3	88,8
5	1,92	2,73	96	91

TAB. VI.2.b

### VI.2.2 Viga em Balanço

O segundo exemplo analisado é o de uma viga em balanço, sob condições de estado plano de tensões, com uma carga unitária concentrada aplicada na extremidade. Para efeito de consideração desta carga, aplicou-se uma distribuição parabólica de tensões no extremo da viga. A figura VI.5 ilustra as características físicas e geométricas do problema.

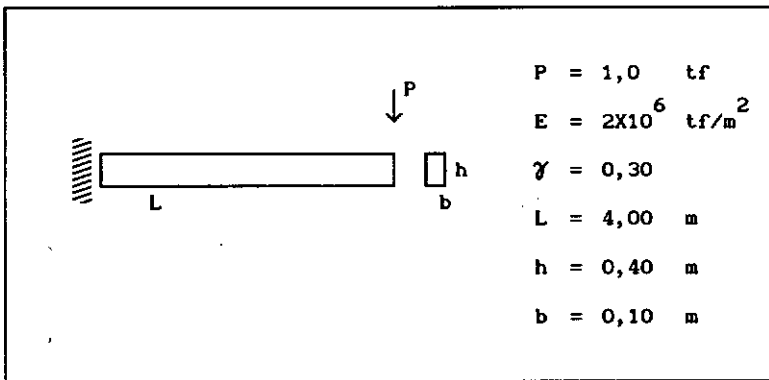


FIG. VI.5 - VIGA EM BALANÇO

O material da viga é linear e isotrópico. Ela foi discretizada por malhas uniformes de elementos quadriláteros isoparamétricos de 4 nós. As análises foram feitas utilizando-se uma sequência de malhas com as características topológicas dadas pela tabela VI.6. nesta tabela, encontram-se as mesmas informações dadas anteriormente na tabela VI.1.

MALHA	NEL	NNOS	NEQ	NWK	LB	LBM
1	4x121	610	1089	12987	13	11
2	4x196	985	1764	21087	13	11
3	4x256	1285	23040	27567	13	11
4	6x256	1799	3328	53591	17	16
5	6x356	2499	4628	74591	17	16
6	6x400	2807	5200	83831	17	16
7	8x400	3609	6800	137311	21	20
8	8x600	5409	10200	206111	21	20

TAB. VI. 6

As tomadas de tempo estão apresentadas na tabela VI.7.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)	T(4)
1	1089	0,44	2,09	2,08	1,57	1,31
2	1764	0,71	4,81	4,76	3,48	2,84
3	2304	0,93	7,81	7,66	5,54	4,46
4	3328	2,25	16,6	16,3	11,7	9,19
5	4628	3,14	30,3	29,7	21,1	16,4
6	5200	3,50	37,6	37,6	26,0	20,3
7	6800	6,90	65,2	64,2	45,0	34,6
8	10200	11,80	139,2	139,1	94,3	71,1

TAB. VI. 7

Por esta tabela, ao confrontarmos Tcolsol X T(N), vemos que o código paralelo é mais lento do que o serial e, a relação Tcolsol/T(N) diverge para valores de ganho (S') e eficiência (E'). Assim, para estruturas onde prevalecem

baixos valores de LB, o algoritmo de fatoração paralela proposto, parece não ser a melhor indicação. Apesar disto, o código foi paralelizado de modo eficiente, pois, pela tabela VI.8, obtivemos valores de ganho (S) e eficiência (E), próximos a 2 e 50%, respectivamente, para o caso mais refinado com quatro processadores. Logo, o tamanho do sistema não é um parâmetro suficiente para garantir bons resultados em relação ao melhor algoritmo serial. Nestes exemplos, a degradação ocorreu em função do pouco volume de dados tratados concorrentemente, ou seja, devido à baixa velocidade do link, pois, conforme evidenciado pelos resultados da tabela VI.9, o  $T_{com}$  é muito pequeno em relação ao  $T_{cpu}$ .

MALHA	S2	S3	S4	E2 %	E3 %	E4 %
1	1,00	1,33	1,59	50	44,3	40
2	1,01	1,38	1,69	50	46,1	42,3
3	1,02	1,41	1,75	50	47	43,8
4	1,02	1,42	1,80	50	47,3	45,1
5	1,02	1,44	1,85	50	48	46
6	1,02	1,45	1,86	50	48	46
7	1,02	1,45	1,89	51	48	47
8	1,02	1,48	1,95	51	49,2	48,9

TAB.VI.8 -  $S=T(1)/T(N)$  e  $E=S/N$

MALHA	2PROC		3PROC		4PROC	
	Tcom	Tcpu	Tcom	Tcpu	Tcom	Tcpu
1	6,7	93,3	17,2	82,8	32	68
2	4,6	95,4	12,9	87,1	23,6	76,4
3	3,4	96,6	10,6	89,4	19,8	80,2
4	3,3	96,7	10,4	89,6	18,8	81,2
5	3,1	96,9	9	91	17,8	82,2
6	2,8	97,2	8,1	91,9	15,1	84,9
7	2,1	97,9	6	94	12	88
8	1,5	98,5	4,4	95,6	8,7	91,3

TAB.VI.9 - Tcom(%) e Tcpu(%)

De forma a ilustrar o ganho e a eficiência obtidos a partir do esquema de ligação com processador gerente (fig.VI.3), a tabela VI.7.a, apresenta as tomadas de tempo para as malhas de número cinco, seis e sete.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)
5	4628	5,5	60,4	30,8	21,6
6	5200	6,2	75,3	38,3	26,6
7	6800	12,1	129,9	64,7	46,5

TAB.VI.7.a - TEMPOS

Conforme pode-se observar pela tabela VI.7.b, os resultados encontram-se bem próximos do caso ideal, com os tempos de comunicação, idênticos ao da tab. VI.9.

MALHA	S2	S3	E2 %	E3 %
5	1,96	2,79	98	93
6	1,96	2,79	98	93
7	2	2,79	100	93,1

TAB. VI.7.b

## VI.2.3 Anel

O terceiro exemplo analisado é o de um anel espesso sujeito a uma pressão interna  $P = 8 \text{ dN/mm}^2$ , sob condições de estado plano, conforme a figura VI.6. O material do anel é linear e isotrópico. Ele foi discretizado por malhas uniformes de elementos quadriláteros isoparamétricos de 4 nós. Os experimentos numéricos analisados apresentaram características topológicas segundo a tabela VI.10.

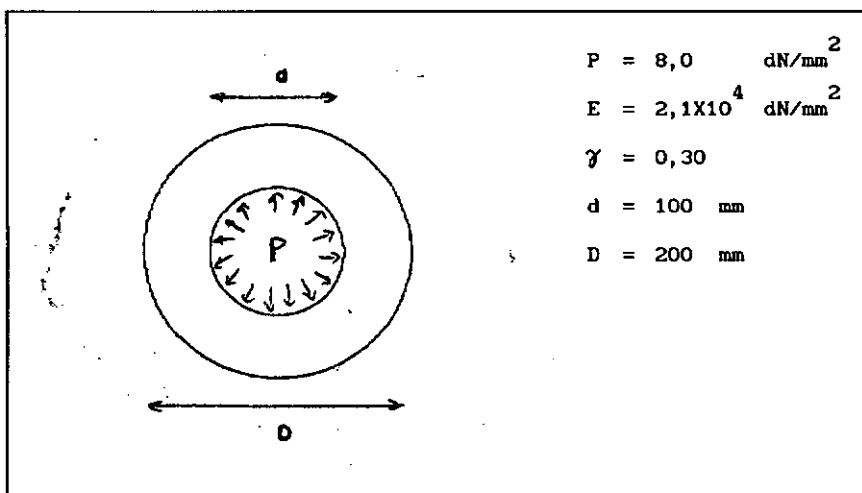


FIG.VI.6 - ANEL

MALHA	NEL	NNOS	NEQ	NWK	LB	LBM
1	8x160	1440	2880	112148	2866	38
2	8x180	1620	3240	126288	3226	38
3	8x200	1800	3600	140428	3586	39
4	10x200	2200	4400	206748	4382	46
5	10x386	4048	8096	381636	8078	47

TAB. VI.10

As tomadas de tempo estão apresentadas na Tab. VI.11.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)	T(4)
1	2880	9,78	22,8	20,3	14,4	11,8
2	3240	11,03	27,0	24,1	17,1	14,0
3	3600	12,26	31,6	28,3	20,0	16,3
4	4400	21,22	50,9	44,2	31,3	24,5
5	8096	63,16	135,6	117,7	81,9	62,5

TAB. VI.11

Nesta, pode-se observar que apenas a malha mais refinada quando executada em quatro processadores, prevaleceu em relação ao COLSOL. Ao calcularmos o ganho ( $S'$ ) e a eficiência ( $E'$ ) para este caso, obtivemos valores mínimos de 1,01 e 25,6% respectivamente. Isto significa que, apenas para sistemas maiores que 8096 equações, serão alcançados valores significativos de desempenho.

Agora, quanto à paralelização do código contra ele mesmo, conforme os resultados da tabela VI.12, chegamos a valores que demonstram a sua paralelização eficiente. Para o caso mais refinado rodado em quatro processadores, obtivemos valores de ganho ( $S$ ) e eficiência ( $E$ ), em torno de 2,2 e 55%, respectivamente e, o tempo de comunicação, dado pela tabela VI.13, ocupa 14% do tempo total de processamento.

MALHA	S2	S3	S4	E2 %	E3 %	E4 %
1	1,12	1,58	1,93	56	52,3	48
2	1,12	1,58	1,93	56	52,3	48
3	1,12	1,58	1,94	56	52,3	48
4	1,15	1,63	2,10	57,6	54,2	52,5
5	1,15	1,65	2,17	57,6	55,1	54,2

TAB.VI.12 -  $S=T(1)/T(N)$  e  $E=S/N$ 

MALHA	2PROC		3PROC		4PROC	
	Tcom	Tcpu	Tcom	Tcpu	Tcom	Tcpu
1	3,6	96,4	10,3	89,7	20,5	79,5
2	3,6	96,4	10,2	89,8	20,4	79,6
3	3,6	96,4	10,2	89,8	20,3	79,7
4	3,6	96,4	10,2	89,8	19,6	80,4
5	2,5	97,5	7,1	92,9	14	86

TAB.VI.13 - Tcom(%) e Tcpu(%)

Ainda, de forma a demonstrar o ganho e a eficiência obtidos com o esquema da ligação com processador gerente, a tab. VI.11.a, apresenta as tomadas de tempo para as malhas um, dois e três.

MALHA	NEQ	Tcolsol	T(1)	T(2)	T(3)
5	4628	5,5	60,4	30,8	21,6
6	5200	6,2	75,3	38,3	26,6
7	6800	12,1	129,9	64,7	46,5

TAB.VI.11.a - TEMPOS

Conforme pode-se observar pela tabela VI.11.b, os resultados encontram-se bem próximos do caso ideal, com os tempos de comunicação idênticos aos da tabela VI.13.

MALHA	S2	S3	E2 %	E3 %
5	1,96	2,79	98	93
6	1,96	2,79	98	93
7	2	2,79	100	93,1

TAB. VI.11. b

### VI.3 - CONCLUSÕES

Pelos resultados obtidos, concluimos que para todas as classes de estruturas analisadas, o código apresentou bons resultados, demonstrando assim, que a estratégia de paralelização adotada é eficiente.

Em relação ao melhor código serial (COLSOL), vimos que a degradação de desempenho ocorreu apenas quando houve pouco volume de dados tratados concorrentemente.

Quanto ao tempo de comunicação, vimos que, para Redes de Transputers, ele não é a principal fonte de degradação, ficando isto, a cargo do balanceamento de cargas, pois, quando temos um processador ocioso (fig.VI.3.a), há uma grande perda de tempo, que mascara os resultados de ganho e eficiência, resultando em fatores "quase" ideais.

## C A P Í T U L O   V I I

### CONCLUSÕES FINAIS

O objetivo foi apresentar um estudo teórico e a implementação em paralelo do método direto de Gauss para a solução de um sistema de equações oriundas do Método dos Elementos Finitos. Para tanto, utilizou-se de um algoritmo que se adaptasse ao máximo ao contexto paralelo, para obter desta forma, uma redução de tempo a mais próxima possível da situação ideal . Em termos práticos, a grande preocupação na escolha de um método é que ele nos possibilite resolver o sistema no menor tempo possível. Com relação a memória, a distribuição dos dados pelos diferentes processadores adotada, além de proporcionar uma economia de memória computacional, pois, esta fica reduzida a uma taxa que é inversamente proporcional ao número de processadores, garante que a carga de processamento seja distribuída de forma equitativa . A preocupação com a economia de memória torna-se muito importante quando tratamos com sistema de dimensão elevada, já que temos um grande volume de dados e, em geral, uma quantidade limitada de memória.

O trabalho foi desenvolvido basicamente em duas partes: na primeira, ( Capítulos I,II,III e IV ), foi estudado a teoria do processamento paralelo e em particular os sistemas *Transputers*. Na segunda parte, ( Capítulos V e VI ), foi estudado o modelo matemático para o método direto considerado, dando ênfase ao aspecto computacional, isto é, apresentação do algoritmo utilizado na implementação do

método em uma rede de *Transputers* e os resultados de medição de desempenho obtidos. Quanto às diferentes implementações computacionais fazemos as seguintes observações:

. A interconexão entre os processadores como mostrado na figura VI.2, funcionou muito bem para o Quadiputer;

. O critério de distribuição de cargas, foi de forma a distribuí-la equitativamente entre os processadores;

. Quando temos mais de uma tarefa sendo executada em paralelo em um único *Transputer*, ele distribui seu tempo entre elas. Este fato faz com que sua eficiência seja menor quanto maior o número de tarefas realizadas em paralelo pelo mesmo ( simulação );

. Um outro problema relaciona-se especificamente com a linguagem FORTRAN-PARALELO; como esta linguagem no *Transputer* não possui alocação dinâmica de memória, há um certo desperdício de memória computacional, ( a memória alocada não será mais utilizada);

A motivação deste trabalho foi mostrar a utilização do *Transputer* para diminuir o tempo de processamento. De uma forma geral, os resultados de desempenho foram satisfatórios, pois, apesar da máquina utilizada não ter uma capacidade de memória disponível razoável, obtivemos resultados significativos. Estes resultados avançam na direção de um melhor entendimento dos mecanismos síncronos, bem como de sua aplicação a problemas práticos da engenharia.

A seguir, apresenta-se alguns tópicos que servirão de sugestões para futuras pesquisas, no sentido de

aperfeiçoar os resultados obtidos por este trabalho:

. Otimização do programa, o que pode proporcionar melhores resultados em termos de perdas de tempo, como por exemplo, a resolução por blocos, como proposto em FARHAT[11];

. Implementar a parte de comunicação entre os processadores em OCCAM, já que esta linguagem é mais adequada para o Transputer que o FORTRAN.

. Implementar o programa no hipercubo da INTEL, onde a comunicação entre os processadores neste, é feita de maneira assíncrona, diminuindo a perda de tempo com a sincronização. A alicação destas técnicas introduzirão novas questões que deverão ser analisadas cuidadosamente, como por exemplo, os fatores relacionados com os efeitos da arquitetura do sistema [28].

# APENDICE A - LISTAGEM DO PROGRAMA DE CONFIGURAÇÃO

! MAIN.CFG - configuração p/ 4 processadores

!

processor host

processor root

processor P2

processor P3

processor P4

!

wire ? root[0] host[0]

wire ? root[1] P3[1]

wire ? root[2] P2[3]

wire ? root[3] P4[2]

wire ? P2[1] P4[1]

wire ? P2[2] P3[3]

wire ? P3[2] P4[3]

!

task MAIN ins=5 outs=5

task TAREFA1 ins=5 outs=5 data=500k

task TAREFA2 ins=5 outs=5 data=500k

task TAREFA3 ins=5 outs=5 data=500k

task filter ins=2 outs=2 data=50k

task afserver ins=1 outs=1

!

place afserver host

place MAIN root

place TAREFA1 P2

place TAREFA2 P3

place TAREFA3 P4

place filter root

!

connect ? filter[0] afserver[0]

connect ? afserver[0] filter[0]

connect ? filter[1] MAIN[1]

connect ? MAIN[1] filter[1]

connect ? MAIN[2] TAREFA1[2]

connect ? TAREFA1[2] MAIN[2]

connect ? MAIN[4] TAREFA2[4]

connect ? TAREFA2[4] MAIN[4]

connect ? MAIN[3] TAREFA3[3]

connect ? TAREFA3[3] MAIN[3]

connect ? TAREFA1[3] TAREFA2[3]

connect ? TAREFA2[3] TAREFA1[3]

connect ? TAREFA1[4] TAREFA3[4]

connect ? TAREFA3[4] TAREFA1[4]

connect ? TAREFA2[2] TAREFA3[2]

connect ? TAREFA3[2] TAREFA2[2]

APENDICE B - LISTAGEM DA ROTINA DE FATORAÇÃO

```

C-----
SUBROUTINE FACTOR
C-----
&( NEQ, NP, COLVAL, COLPOS, COLTIP, NCOLPN, L, MAXA )
INCLUDE 'CHAN.INC'
INTEGER CHOUT(3), CHIN(4), MAXA(NEQ+1), CANAL
INTEGER COLTIP(NCOLPN), COLPOS(NCOLPN+1), PROC
DIMENSION COLVAL(L), PIVOT(2113), RBUFF(2113)

C
CHIN(4) = F77_CHAN_IN_PORT (3)
CHOUT(1) = F77_CHAN_OUT_PORT(3)
CHIN(1) = F77_CHAN_IN_PORT (4)
CHOUT(2) = F77_CHAN_OUT_PORT(4)
CHIN(2) = F77_CHAN_IN_PORT (2)
CHOUT(3) = F77_CHAN_OUT_PORT(2)

NODE = 2

DO I = 1, NEQ
RBUFF(I) = 0
ENDDO

C
C .... LOOP ON THE NEQ STEPS.
C
DO 100 K=1,NEQ
C
C .... FIND ROOT NODE.
C
NROOT=MOD(K-1,NP)
C
C .... GET COLUMN K LOCAL NUMBER IN NODE NROOT.
C
LCNK = (K-NROOT-1)/NP + 1

IF(NODE.EQ.NROOT) THEN
C
C .... DEFINE ROOT PROCESS.
C
C 1 - BUFFER TEMPORARY ARRAY.
C
ISTART = COLPOS ( LCNK )
ISTOP = COLPOS ( LCNK + 1 ) - 2
KSTART = K - ISTOP + ISTART - 2
IB = 0

DO IR = ISTART, ISTOP
IB = IB + 1
RBUFF( IB ) = COLVAL( IR ) * PIVOT( KSTART + IB )
ENDDO

NWORDS = IB
C

```

```

C      2 - REDUCAO DO ELEMENTO DIAGONAL:
C
LENGTH = K - COLTIP(LCNK)
ID      = COLPOS(LCNK+1)-1
COLVAL(ID) = COLVAL(ID)
&      - DOTP( COLVAL(ISTART), RBUFF(1), LENGTH )
C
C      3 - TAIL COMPUTED PIVOT TO MESSAGE.
C
PIVOT(K)      = COLVAL(ID)
NWORDS       = NWORDS + 1
RBUFF(NWORDS) = PIVOT(K)
C
C      4 - TRANSMISSAO DE "BUFFER" A TODOS PROCESSADORES .
C
IF ( K.EQ.NEQ ) RETURN

NBYTES = 4 * NWORDS
DO CANAL = 1, 3
CALL F77_CHAN_OUT_MESSAGE(NBYTES,RBUFF,CHOUT(CANAL))
ENDDO
C
C      5 - SKIP RECVW.
C
GO TO 50

ENDIF
C
C .... DEFINE LEAF PROCESS.
C
C      1 - RECEBE MENSAGEM DO PROCESSO ROOT.
C
IF ( K.EQ.NEQ ) RETURN

CANAL = NROOT + 1
NWORDS = MAXA(K+1)-MAXA(K)
NBYTES = 4 * NWORDS
CALL F77_CHAN_IN_MESSAGE(NBYTES,RBUFF,CHIN( CANAL ))

PIVOT(K) = RBUFF(NWORDS)
C
C      2 - POINT TO LEADING ACTIVE COLUMN WITHIN NODE.
C
50 IF(NODE.LE.NROOT)
&THEN
    LACSTART = LCNK + 1
    LACSTOP  = NCOLPN
ELSE
    LACSTART = LCNK
    LACSTOP  = NCOLPN
ENDIF
C
C      3 - CHECK IF NODE IS ACTIVE.
C
IF(LACSTART.GT.LACSTOP) GO TO 100
C

```

```

C      4 - FOR EACH ACTIVE COLUMN, REDUCE ELEMENT
C      WITH GLOBAL ROW # = K (IF EXISTS)
C
DO 200 LAC = LACSTART, LACSTOP

KK = COLTIP(LAC)

IF(KK.GT.K) GO TO 200

ITOP   = COLPOS(LAC)
LENGTH = K - KK
ML     = MIN0(LENGTH,NWORDS-1)
I1     = ITOP + LENGTH - ML
I2     = NWORDS - ML
IK     = ITOP + LENGTH

COLVAL(IK) = ( COLVAL(IK)
&           - DOTP ( COLVAL(I1), RBUFF(I2), ML ) )
&           / RBUFF(NWORDS)

200    CONTINUE
100    CONTINUE

RETURN
END

```

```

C-----
C      FUNCTION  DOTP ( A, B, N )
C-----
C
DIMENSION A(N), B(N)
DOTP = 0

DO I = 1, N
DOTP = DOTP + A(I) * B(I)
ENDDO

RETURN
END

```

## REFERENCIAS BIBLIOGRAFICAS

- [ 1 ] TABAK, D. - " RISC Architecture " - Research Studies Pres, Letchworth, Herts., UK, (1987).
- [ 2 ] NOOR, A.K. - " New Computing Systems and Their Impact on Structural Analysis and Design " - Supercomputing in Engineering Structures, pp. 1-42, (1989).
- [ 3 ] WILSON, E.L. - " Finite Element Analysis on Computers With Multiple Processors " - Supercomputing in Engineering Structures, pp. 43-54, (1989).
- [ 4 ] Revista BYTE, pp. 287-296, November, (1988).
- [ 5 ] " The Transputer Data Limited ", Bristol, UK, (1989).
- [ 6 ] ALVES FILHO, J.S.R. - " The Use of Transputer Based Computers in Finite Element Calculations " - PhD Thesis, Department of Civil Engineering, University of Wales, Swansea, UK, (1989).
- [ 7 ] " Parallel FORTRAN / User Guide " - 3L Ltd and Edinburgh Portable Compilers Ltd., (1988).
- [ 8 ] BABB, R.G. - " Programming Parallel Processors " - Addison-Wesley Publishing Company, Inc., (1988).
- [ 9 ] WILSON, E., DOVEY, H. - " Solution or Reduction of Equilibrium Equations for Large Complex Structural Systems " - Advances in Engineering Software, 1(1), pp. 19-25, (1987).

- [10] FARHAT, C., - " *Multiprocessors in Computational Mechanics* " - PhD Thesis, Department of Civil Engineering, University of California, Berkeley, USA, (1987).
- [11] FARHAT, C., WILSON, E. - " A Parallel Active Column Equation Solver " - *Computers and Structures*, 28(2) : 289-304, (1988).
- [12] ALVES FILHO, J.S.R., OWEN, D.R.J.- "Using Transputers in Finite Element Computations " - *Benchmark*, October, pp. 39-41, (1989).
- [13] FARHAT, C. - " Redesigning the Skyline Solver for Parallel/Vector Supercomputers " - *Center for Space Structures and Controls*, Report CV-CSSC 89-19 (1989).
- [14] STORAASLI, O.O., NGYENT, D.T., AGARWAL, T.K. - " Parallel-Vector Solution of Large-Scale Structural Analysis Problems on Supercomputers " - *AIAA Journal*, V.28, N.7, pp. 1211-1216, July, (1990).
- [15] LOZUPONE, D.F., MAYES, P., RADICATI, G. - " Skyline Cholesky Factorization Using Level 3 BLAS " - *IBM - European Center for Scientific and Engineering Computing*, Research Report ICE-0037, (1990).
- [16] LEWIS, J., SIMON, H. - " The Impact of hardware Gather/Scatter on Sparse Gaussian Elimination", *SIAM J. Sci. Statist. Comput*, 9, pp.304-311, (1988)
- [17] BUNING, P., LEVY, J. - " Vectorization of Implicit Navier-Stokes Codes on the CRAY1 Computer " - *Tech. Report, Dep. of Aeronautics, Stanford University*, (1979).

- [18] GEORGE, A., HEATH, M., LIU, J.W.-H., NG, E.G.-Y. - " Sparse Cholesky Factorization on a Local-Memory Multiprocessor " - *SIAM J. Sci. Stat. Comput.*, 9, pp.327-340, (1988).
- [19] DUFF, I. - " Parallel Implementation of Multifrontal Schemes " - *Parallel Computing*, 3, pp.193-204, (1986).
- [20] LIU, J.W.-H. - " The Multifrontal Method and Paging in Sparse Cholesky Factorization " - *ACM Trans. Math. Software*, 15, pp. 310-325, (1989).
- [21] DONGARRA, J.J., DU CROZ, J.J., DUFF, I.S., HAMMARLING, S.J. - " A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs " - *Technical Memorandum No 122 MCS Division, Argonne National Laboratory*, (1988).
- [22] FARHAT, C., WILSON, E., POWELL, G. - " Solution of Finite Element System on Concurrent Processing Computers " - *Engng. Comput.*, 2, 157-165 (1987).
- [23] SIQUEIRA, M.Q. - " *Análise estrutural por Partição de Domínio em Ambientes de Processamento Paralelo* " - TESTE M.Sc., Programa de Engenharia Civil, COPPE/UFRJ, (1989).
- [24] STORAASLI, O.O., BERGAN, P. - " A Nonlinear Substructuring Method for Concurrent Processing-Computers " - *AIAA Journal*, v25, June, pp. 871-876, (1987).
- [25] DONGARRA, J.J., DUFF, I.S., SORENSEN, D.C., VAN DER VORST, H.A. - " *Solving Linear System on Vector and Shared Memory Computers* " - SIAM, (1991).

- [26] ORTEGA, J.M. - "Introduction to Parallel and Vector Solution of Linear Systems" - Plenum Press, New York, (1988).
- [27] FLYNN, M.J. - " Some Computer Organizations and their Effectiveness " - *I.E.E.E. Transactions on Computers*, v21(9):948-960, (1966).
- [28] MOTA, F.C. - " *Convergência de Métodos Numéricos Síncronos e Assíncronos para Equações Quase Lineares* " - TESE M.Sc., Programa de Engenharia Elétrica, COPPE/UFRJ, (1990).
- [29] AMORIM, C.L., BARBOSA, V.C., FERNANDES, E.S.T. - " *Uma Introdução a Computação Paralela e Distribuída* " - VI ESCOLA DE COMPUTAÇÃO, SBC, Campinas, SP, (1988).
- [30] HOCKNEY, W.R. - " MIMD Computing in USA - 1984 " - *Parallel Computing*, 2, pp. 119-136, (1985).
- [31] HÄNDLER, W. - " The Impact of Classification Schemes on Computer Architecture " - pp. 7-15, in *Proceedings of the 1987 International Conference on Parallel Processing*, (1977).
- [32] GAJSHI, D.J., PEIR, J.K. - " Essential Issues in Multiprocessor Systems " - *IEEE Computer*, pp. 9-27, June, (1985).
- [33] SIEGEL, H.J. - " *Interconnection Networks for Large-Scale parallel Processing* " - Lexington Books, Lexington, MA, (1984).
- [34] BURNS, A.- " *Programming in OCCAM2* " - University of Bradford, Addison-Wesley Publishing Company, (1988).

- [35] HOARE, C.A.R. - "Communication Sequential Processes"  
- Prentice Hall, Englewood Cliffs, NJ, (1985).
- [36] SCHENDEL, U., CONOLLY, B.W. - "Introduction to  
Numerical Methods for Parallel computers" -  
Ellis Horwood Ltd., Chischester, Great Britain,  
(1984).
- [37] BERTSEKAS, D.P., TSITSIKLIS, J.N. - "Parallel and  
Distributed Computation" - Prentice Hall, Inc.,  
Englewood Cliffs, NJ, (1989).
- [38] GALIVAN, K.A., HEAT, M.T., NG, E., ORTEGA, J.M.,  
PEYTON, B.W., ROMINE, C.H., SAMEH, A.H., VOIGT,  
R.G. - "Parallel Algorithms for Matrix  
Computations" - SIAM, (1990).
- [39] QUINN, M.J. - "Designing Efficient Algorithms for  
Parallel Computers" - McGraw-Hill, (1987).
- [40] SAGRILO, L.V.S. - "Confiabilidade de Estruturas  
Reticuladas em Ambientes de Processamento  
Paralelo" - TESE M.Sc., Programa de Engenharia  
Civil, COPPE/UFRJ, (1989).
- [41] NAVAUX, P.O.A. - "Introdução ao Processamento  
Paralelo" - Revista Brasileira de Computacao, Rio  
de Janeiro, V.5, N.2, pp.31-43, (1989).
- [42] JOHNSON, S. - "Communication Efficient Basic Linear  
Algebra Computations on Hypercube Architectures" -  
Research Report YALEU/DCS/RR-361, September,  
(1985).

- [43] CHEN, S., DONGARRA, J.J., HSIUNG, C. - " Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences With Small Granularity, *J. Parallel Distributed Comput.* 1, pp.22-31 (1984).
- [44] NOUR-OMID, B., PARK, K.C. - " Solving Structural Mechanics Problems on the Caltech Hypercube " - *Comput. Meth. Appl. Mech. Engng.*, 61, pp.161-176, (1987).
- [45] BATHE, K.J. - " *Finite Element Procedures in Engineering Analysis* " - Prentice Hall, Inc., Englewood Cliffs, N.J., (1982).
- [46] HUGHES, T.J.R. - " *The Finite Element - Linear Static and Dinamic Finite Element Analysis*" - Prentice-Hall International Editions, (1987).
- [47] ZIENKIWICZ, O.C., TAYLOR, R.L., - " *The Finite Element Method - Basic Formulations and Linear Problems*", Vol.1, MacGraw-Hill, (1989).
- [48] CUTHILL, E., MCKEE, J. - " Reducing the Bandwith of Sparse Symmetric Matrices", *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, ACM Publ. (1969).
- [49] DONGARRA, J.J., MOLER, C.B., BUNCH, J.R., STEWART, G.W. - "LINPACK User's Guide" - SIAM, Philadelphia, USA, ( 1979 ).
- [50] ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., SORENSEN, D. - "LAPACK User's Guide" - SIAM, Philadelphia, USA, (1992).

- [51] GEORGE, A., LIU, J.W.H. - "*Computer Solution of Large Sparse Positive Definite Systems*" - Prentice Hall, Inc., Englewood Cliffs, N.J., (1981).
- [52] ANFLEX - "Análise de Risers Flexíveis e Linhas de Amarração" , PETROBRAS/CENPES/SEDEM, (1991).