PROCESSAMENTO PARALELO EM ANÁLISE ESTRUTURAL

MARCELO NOVAES DE REZENDE

Tese apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Doutor em Engenharia de Estruturas.

ORIENTADOR: Prof. Dr. João Batista de Paiva

São Carlos

1995

FOLHA DE APROVAÇÃO

Tese defendida e aprovada em 27/3/1995
perante a Comissão Julgadora:

Prof. Associado JOÃO BATISTA DE PAIVA - Orientador
(Escola de Engenharia de São Carlos - USP)

Prof. Doutor PHILIPPE R. B. DEVLOO
(Universidade Estadual de Campinas - UNICAMP)

Prof. Doutor CAETANO TRAINA JUNIOR
(Instituto de Ciências Matemáticas de São Carlos - USP)

Prof. Doutor HUMBERTO BREVES CODA
(Escola de Engenharia de São Carlos - USP)

Prof. Doutor MARCIO ANTONIO RAMALHO
(Escola de Engenharia de São Carlos - USP)

Presidente da Comissão de Pós-Graduação
Prof. Dr. EDUARDO CLETO PIRES

Coordenador da área - Engenharia de Estruturas
Prof. Dr. MOUNIR KHALIL EL DEBS

para minha esposa Adriana e
meus filhos Rafael e Beatriz

AGRADECIMENTOS

Ao Prof. Dr. João Batista de Paiva, pela valiosa orientação dada ao longo deste trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela bolsa de estudo concedida.

À Universidade Federal de São Carlos, pela autorização de uso de seus computadores.

SUMÁRIO

RESUMO

REZENDE, M.N. *Processamento paralelo em análise estrutural.* São Carlos, 1995. 112p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.

Este trabalho aborda a utilização de computadores paralelos no processamento de problemas de análise estrutural. Inicialmente apresentam-se peculiaridades dos computadores de arquitetura paralela e do desenvolvimento de programas voltados a tais máquinas. Discute-se então a paralelização de duas etapas típicas da aplicação do método dos elementos finitos em análise estrutural : a montagem da matriz de rigidez da estrutura e a resolução do sistema de equações lineares. São propostos algoritmos alternativos para ambas as etapas. Finalmente é abordada a implementação de um programa de análise não linear de treliças tridimensionais com várias etapas paralelas.

Palavras-chave: Processamento paralelo (computadores eletrônicos) - análise estrutural; Método dos elementos finitos

ABSTRACT

REZENDE, M.N. *Parallel processing in structural analysis.* São Carlos, 1995. 112p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.

This work is about the use of parallel computers on solving structural analysis problems . Initially, the particular features of parallel computers and the related software development are presented. The parallelization of two typical steps of finite method element are discussed : the assembly of global stiffness matrix and the solution of the linear system of equations. Alternative algorithms are proposed for both steps. Finally, the implementation of a program for nonlinear space truss analysis with several parallel steps is presented.

Keywords: Parallel processing (Eletronic computers) - structural analysis; Finite element method

1 - INTRODUÇÃO

O quase inextricável tratamento matemático dos problemas oriundos da análise estrutural impôs por muito tempo aos engenheiros, a representação dos mesmos por aproximações dos escassos problemas teóricos de solução até então matematicamente viável. Obviamente, as claras limitações de tais procedimentos levaram à busca de intrumentos de análise operacionalmente factíveis que conduzissem a respostas mais precisas aos problemas práticos. Com o surgimento dos computadores digitais, por volta dos anos 50, viabilizou-se a solução automática de problemas com um número finito de variáveis e, consequentemente, a solução de problemas envolvendo o meio contínuo representados por um número finito de parâmetros, ou seja, problemas contínuos discretizados.

O procedimento geral de discretização de problemas contínuos delineados por expressões definidas matematicamente pode ser a pura definição do método dos elementos finitos (assim nomeado no início da década de 60). O processo de discretização do método é feito, como pode ser visto em ZIENKIEWICZ(1967), de acordo com o seguinte princípio : o meio contínuo é dividido em um número finito de partes (elementos), cujo comportamento se especifica em função de um número finito de parâmetros.

Aumentando-se a discretização do meio contínuo, o problema real normalmente torna-se melhor representado, produzindo respostas cada vez mais precisas. A discretização mais refinada no entanto, pelo fato de abordar o problema com um número maior de parâmetros, implica no tratamento computacional de matrizes e vetores de grandes dimensões, requerendo computadores com grande área de memória, além de velocidade de processamento tal que viabilize a aplicação do

método em problemas práticos. Além dos requisitos relativos à máquina e à formulação utilizada na discretização , deve-se destacar também a importância da qualidade dos programas utilizados, sem a qual a potencialidade de processamento da máquina não é confiavelmente usufruída. Pode-se portanto afirmar que a eficácia da aplicação de métodos que fazem uso da discretização na análise estrutural depende do trinômio máquina, formulação e programa.

Particularmente no aspecto máquina, é interessante notar-se a ascendência comum da grande maioria dos computadores atuais no esquema criado por von NEUMANN(1945), a chamada "máquina de von Neumann". Nesse tipo de arquitetura, idealizada nos anos 40, o processador busca sequencialmente cada instrução na memória, executando-a em seguida. Refletindo o esquema arquitetural, as linguagens de programação voltadas a tais máquinas (ditas sequenciais) obrigam os programadores a desenvolver programas em que a solução de problemas é realizada no cumprimento sequencial de tarefas. Desta forma, mesmo os problemas que admitem solução na forma de múltiplas tarefas executadas concorrentemente, quando processados em máquinas sequenciais, necessariamente realizam uma a uma tais tarefas, multiplicando o tempo de processamento.

Do início dos anos 50 até os anos 80 o desempenho das máquinas sequenciais aumentou à assombrosa taxa de dez vezes a cada cinco anos . Porém, ao longo desse período, os empecilhos cada vez maiores encontrados na incessante busca de performance desses computadores tornaram evidente a limitação de tal esquema arquitetural e levaram à introdução de paralelismo . A introdução do paralelismo nas arquiteturas significou o rompimento com a necessidade de se executar instruções uma a uma e, como seria de se esperar, tornou as arquiteturas e linguagens sequenciais um caso particular das paralelas. Mesmo na breve história da utilização dos computadores no meio científico, a disponibilidade de computadores paralelos é extremamente recente e, por outro lado, inegavelmente promissora, pois ao romper com os limites (até físicos, como será abordado no segundo capítulo deste trabalho) das arquiteturas sequenciais, abre as portas para a solução automática de um vasto conjunto de problemas de áreas diversas, os quais são de processamento inviável nos moldes de desempenho dos computadores sequenciais. Tipicamente tais

problemas exigem respostas rápidas para tarefas extremamente complexas, como é o caso da utilização de inteligência artificial no controle de aviões, robôs e etc. Há, além da esfera do processamento em tempo real, casos de problemas de alta complexidade em termos de número de operações, que nos limites do processamento sequencial a resposta só seria dada em tempo não viável. Um exemplo desse tipo de problema é a análise estrutural envolvendo muitos graus de liberdade e ou tratamento não linear, ou ainda dinâmico.

O objetivo deste trabalho é introduzir os conceitos da computação paralela, visando a utilização de sua potencialidade de processamento em problemas de análise estrutural. Abordam-se inicialmente generalidades sobre arquiteturas paralelas e desenvolvimento de programas voltados às mesmas. Posteriormente são tratados aspectos da paralelização de etapas típicas do método dos elementos finitos em análise estrutural, particularmente para o caso de arquiteturas MIMD. Finalmente apresentam-se detalhes da implementação de um programa para análise não linear de treliças tridimensionais com várias etapas paralelas.

No segundo capítulo são descritas resumidamente as principais arquiteturas paralelas existentes na atualidade, sendo que uma detalhada apresentação do assunto pode ser encontrada em DeCEGAMA(1989). No referido capítulo, inicialmente o esquema e as limitações da "máquina de von Neumann" são tratados. A classificação corrente das arquiteturas paralelas é então abordada e, finalmente, as características básicas dos principais esquemas arquiteturais paralelos são discutidas.

O terceiro capítulo contém peculiaridades do desenvolvimento de software voltado aos computadores paralelos. Sendo o preeminente objetivo da utilização de computadores paralelos a redução do tempo de processamento, sob essa ótica abordam-se nesse capítulo os princípios básicos da elaboração de programas eficientes e confiáveis para tais máquinas. Um breve resumo das ferramentas disponíveis para elaboração de software paralelo é também incluído nesse capítulo. Um aprofundamento teórico sobre o desenvolvimento de algoritmos paralelos eficientes aparece em QUINN(1987).

O quarto capítulo aborda a paralelização de uma típica etapa da aplicação do método dos elementos finitos em análise estrutural : a montagem da matriz de

rigidez da estrutura. São apresentadas as paralelizações conhecidas, as quais são normalmente oriundas de adaptações do algoritmo sequencial usual, cabendo destaque para os trabalhos de CHIEN & SUN(1989) e ADELI & KAMAL(1992). Um algoritmo paralelo alternativo para a montagem da matriz de rigidez da estrutura é também proposto nesse capítulo.

A solução paralela de sistemas de equações lineares é tratada no quinto capítulo. Essa etapa, comum a vários métodos de análise estrutural, dentre eles método dos elementos finitos e método dos elementos de contorno, é a que geralmente requer maior esforço computacional, sendo portanto extremamente frutífera a sua paralelização e justificável a existência de numerosos trabalhos nessa área. FREEMAN(1992) apresenta a paralelização de métodos diretos e iterativos de solução de sistemas de equações lineares e aborda também a aplicabilidade dos mesmos em arquiteturas MIMD de memória compartilhada e distribuída. O estudo neste trabalho restringe-se à paralelização normalmente feita do método de eliminação de Gauss para arquitetuta MIMD de memória compartilhada. É também proposto um esquema alternativo para a etapa de triangularização do método de eliminação de Gauss, o qual procura mitigar o custo computacional das diversas criações de processos concorrentes do esquema normalmente utilizado. A paralelização do método de eliminação de Gauss também é tratada em LAKSHMIVARAHAN(1990), DEKKER(1994) e QUINN(1987).

No capítulo seis, os algoritmos paralelos propostos nos capítulos quatro e cinco são utilizados na implementação de um programa paralelo de análise não linear de treliças tridimensionais em computador de arquitetura MIMD de memória compartilhada. Além da paralelização da montagem da matriz de rigidez da estrutura e da solução do sistema de equações lineares o programa efetua também o cálculo paralelo dos esforços normais nas barras e do vetor de resíduos com sua correspondente norma Euclidiana. são feitas avaliações de *Speed-up* do programa e individualmente de suas etapas paralelizadas.

Finalmente, no capítulo sete apresentam-se as principais conclusões do trabalho e são feitas também sugestões para prosseguimento de pesquisa na área.

2- PARALELISMO NA ARQUITETURA DE COMPUTADORES

2.1 INTRODUÇÃO

Os computadores sequenciais, apesar das diferenças de desempenho que os possam distinguir, apresentam semelhanças tais em suas arquiteturas, que permitem aos programadores desenvolver programas em linguagens de alto nível sem maiores preocupações relativas a peculiaridades da arquitetura da máquina. Já no desenvolvimento de software voltado a computadores paralelos há uma clara e necessária vinculação entre o programa e a arquitetura da máquina a qual o mesmo se destina. Isto se deve à diversidade de arquiteturas paralelas existente e implica na necessidade de noções básicas sobre arquiteturas de computadores a quem pretende desenvolver esse tipo de software. Esses conceitos básicos são apresentados neste capítulo.

2.2 A MÁQUINA DE VON NEUMANN

Pode-se dizer que a arquitetura da grande maioria dos computadores sequenciais e paralelos existentes na atualidade é, de alguma forma, oriunda do sistema de computação idealizado por von NEUMANN(1945), conhecido como "máquina de von Neumann". Com esse sistema começou o conceito de "programa armazenado", caracterizado pelo fato do programa e os dados compartilharem a memória. Deve-se ressaltar que nas arquiteturas anteriores a memória era reservada apenas aos dados e resultados intermediários, sendo que a sequência de instruções era estabelecida pelo posicionamento de controles (chaves) externos à máquina .

Conforme apresentado em ALMEIDA (1991), o modelo de von Neumann

(fig. 2.1) é, na sua forma clássica, composto por cinco unidades básicas :

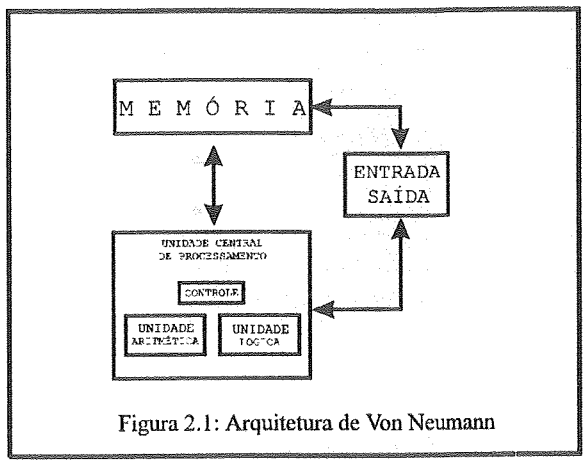


Figura 2.1: Arquitetura de Von Neumann

> Entrada : transmite dados e instruções do ambiente exterior para a memória;

> Memória : armazena instruções, dados e resultados intermediários;

> Unidade lógico-aritmética (ALU) : executa as operações lógicas e aritméticas;

> Controle : interpreta as instruções e providencia a execução;

> saída : transmite os resultados finais para o ambiente exterior.

O funcionamento da máquina de von Neumann se dá pela execução sequencial de instruções. A execução de cada uma das instruções é feita em um ciclo composto por seis etapas (fig.2.2) :

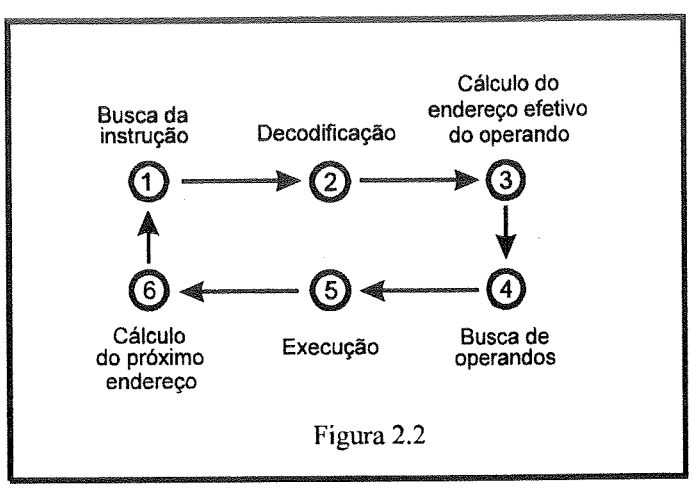


Figura 2.2

1- Busca da instrução : a CPU (unidade central de processamento, engloba controle e unidades lógica e aritmética) busca na memória a próxima instrução a ser executada;

2- Decodificação (na CPU a instrução é decodificada);

3- Cálculo do endereço efetivo dos operandos envolvidos na instrução;

4- Busca dos operandos na memória;

5- Execução da operação;

6- Cálculo do endereço efetivo da próxima instrução a ser executada.

O fato da máquina de von Neumann requerer vários acessos à memória (a qual é notoriamente mais lenta que a CPU) durante o ciclo de instrução é hoje a principal fonte das críticas feitas a esse modelo de arquitetura. Em DeCEGAMA(1989) são descritas algumas arquiteturas alternativas ainda pouco utilizadas, como é o caso da *Data flow* e da *Reduction*. No conceito de *Data flow* as instruções são cumpridas tão logo estejam prontos os operandos e não seguindo a sequência imposta do esquema da "máquina de von Neumann", já no conceito de *Reduction*, as instruções são realizadas quando necessárias à execução de outras.

2.3 OS CAMINHOS DA EVOLUÇÃO DA MÁQUINA DE VON NEUMANN

Dos primeiros computadores digitais construídos ainda na década de 40 (ENIAC, EDSAC, EDVAC e UNIVAC), apenas o ENIAC (o primeiro deles) não pode ser considerado plenamente uma máquina de von Neumann, pois não fazia ainda uso do conceito de "programa armazenado". Essas quatro máquinas pioneiras faziam acesso serial à memória. A passagem para acesso paralelo à memória se deu com a construção do IBM 701 em 1953, e este pode ser considerado o marco inicial das arquiteturas paralelas.

Segundo HOCKNEY(1988), do aparecimento dos primeiros computadores de aplicação comercial na década de 50 até os anos 80, verifica-se um aumento de dez vezes no desempenho dos computadores a cada cinco anos, sendo esse desempenho medido pela quantidade de instruções que a máquina pode executar em determinado intervalo de tempo (taxa de execução). Esse notável aumento de

desempenho se deveu em parte aos avanços na tecnologia de semicondutores e, em parte à evolução do modelo original da máquina de von Neumann.

A melhoria de desempenho pelo avanço da tecnologia de semicondutores se dá pelo aumento da velocidade dos componentes (memória, CPU e etc), fazendo com que o tempo dispendido em um ciclo de instrução seja minimizado e, consequentemente, a quantidade de instruções efetuadas por intervalo de tempo (taxa de execução) aumente. Os custos agregados à elaboração de uma nova tecnologia de semicondutores no entanto, são elevadíssimos e, além disso, conforme mostrado em DeCEGAMA(1989), há um limite físico para a velocidade de transmissão de sinais determinado pela velocidade de propagação da luz no vácuo ($3x10^8$m/s) . Com esse limite físico, os computadores sequenciais estão condenados a não ultrapassar 10^9FLOPS (operações em ponto flutuante por segundo). Os atuais supercomputadores sequenciais estão aproximadamente dez vezes mais lentos que esse limite.

A idéia básica de se aumentar o desempenho pela otimização da arquitetura é simples : embora o tempo de ciclo de instrução seja fixo, pode-se, pela introdução de paralelismo, fazer que em cada ciclo de máquina seja executada mais do que uma instrução, aumentando-se a taxa de execução e obviamente o desempenho do computador. O termo "paralelismo" deve ser entendido como processamento simultâneo de tarefas, programas, rotinas, processos, laços de instrução e instruções. Em ALMEIDA(1991) encontra-se um exemplo ilustrativo do ganho de desempenho oriundo da otimização da arquitetura : O ciclo de máquina do computador Cray X-MP é 21 vezes mais rápido que o do VAX 11/780, e o desempenho do primeiro é aproximadamente 100 vezes superior ao do segundo, ou seja o computador Cray X-MP além do tempo de ciclo menor, consegue um enorme excedente de desempenho em relação ao VAX 11/780 em função de sua arquitetura permitir maior nível de paralelismo.

A busca de alto desempenho nos computadores tem portanto na atualidade duas correntes: a primeira, restrita aos grandes fabricantes mundiais (IBM, NEC, Cray e etc), concentra seus esforços na evolução da tecnologia de componentes, com vistas à diminuição do tempo de ciclo; a segunda procura ganhos de desempenho pela paralelização das arquiteturas, combinando vários processadores já comuns no

mercado. A segunda corrente tem permitido a novos fabricantes desenvolver rapidamente e com custos reduzidos, máquinas de alto desempenho.

2.4 CLASSIFICAÇÃO DO PARALELISMO ENVOLVIDO NAS ARQUITETURAS (TAXONOMIA DE FLYNN)

Embora a máquina de von Neumann seja ainda o esquema básico das arquiteturas atuais, sua concepção inicial é um modelo fundamentalmente sequencial e, portanto, limitado em termos de desempenho. A otimização desse modelo foi quase sempre caracterizada por tentativas de se proporcionar algum tipo de paralelismo ao longo do ciclo de instrução.

Para a classificação do tipo de paralelismo envolvido nos sistemas computacionais é comum utilizar-se a taxonomia proposta por M. FLYNN(1972), a qual leva o seu nome :

1- SISD (Single Instruction Single Data) : Refere-se ao processamento de uma única série de instruções sobre uma única sequência de dados, ou seja, é o modelo clássico da máquina de von Neumann;

2- SIMD (Single Instruction Multiple Data) : Trata-se do processamento simultâneo da mesma sequência de instruções sobre diferentes dados. Por efetuarem paralelamente a mesma instrução sobre diferentes elementos de um vetor, alguns tipos de processadores vetoriais, como será visto, pertencem a esta categoria;

3 - MISD (Multiple Intruction Single Data) : As máquinas dessa categoria efetuam paralelamente diferentes instruções sobre a mesma sequência de dados. FREEMAN(1992) relata que ainda não foram desenvolvidas máquinas MISD;

4- MIMD (Multiple Instruction Multiple Data) : Engloba uma grande variedade de arquiteturas que possibilitam o processamento paralelo de instruções diferentes sobre dados diferentes. As máquinas MIMD podem ser entendidas como um conjunto de processadores trabalhando independentemente sob o controle de um único sistema operacional. A maneira como os processadores e memórias estão conectados, como será visto adiante, permite subdivisões da categoria MIMD. Os computadores pertencentes à categoria MIMD são também conhecidos pelo nome de Multiprocessadores.

Há também a possibilidade de existência de uma arquitetura híbrida dentro da taxonomia de Flynn, ou seja, pode-se construir por exemplo uma máquina MIMD que, para a otimização do processamento vetorial utilize um conjunto de processadores vetoriais (SIMD). Pode-se ainda utilizar uma máquina MIMD para processamento vetorial, ou seja, simular-se uma máquina SIMD em uma MIMD.

2.5 ARQUITETURAS USUAIS NOS COMPUTADORES DE ALTO DESEMPENHO

2.5.1 PROCESSADORES VETORIAIS

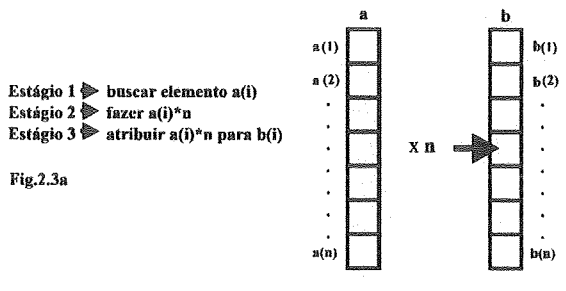
Os problemas envolvendo complexos modelos matemáticos abrangem hoje variadas atividades. Da exploração de petróleo, passando por análise estrutural e aviação até o processamento de imagens, são inúmeros os problemas representados por complexos modelos matemáticos e que, na maioria das vezes, requerem alto desempenho computacional para a sua solução. O ponto comum dos programas que tratam tais problemas é a numerosa ocorrência de operações sobre estruturas de dados do tipo matriz ou vetor. Desta forma, havendo uma arquitetura especializada no tratamento de operações vetoriais, aumenta-se grandemente o desempenho dos computadores que processam problemas numéricos.

As arquiteturas especializadas no tratamento de operações vetoriais são denominadas processadores vetoriais e os computadores que as possuem denominam-se computadores vetoriais. Há duas formas mais usuais de se construirem processadores vetoriais : com a utilização do conceito de "pipelining" (forma mais usual) e utilizando-se máquinas SIMD "puras".

Segundo STONE(1987) a técnica de "pipelining" (processamento em duto) tem sido a técnica mais utilizada para aumento de desempenho de computadores. Essa técnica, extremamente semelhante à idéia da linha de montagem industrial, consiste na divisão de uma tarefa em subtarefas representadas por estágios sucessivos e na existência de executores independentes para cada um deles . Supondo-se uma divisão de tarefa em quatro estágios, quando uma tarefa está no quarto estágio já há, concorrentemente, três outras tarefas respectivamente no terceiro, segundo e primeiro estágio. Desta forma, embora cada tarefa seja realizada em quatro estágios, a cada

mudança de estágio há a conclusão de uma nova tarefa.

A utilização da técnica de pipelining nas operações sobre vetores é bastante eficiente, pois tais operações são, na maioria das vezes, repetições da mesma tarefa sobre os elementos do vetor. Como exemplo de uma operação vetorial em pipeline seja a operação de multiplicar um escalar n por um vetor a, resultando no vetor b (figura 2.3.a). A tarefa de multiplicar cada elemento do vetor a pelo escalar n e armazenar o resultado em b pode ser hipoteticamente dividida em três subtarefas, cada uma realizada em um ciclo : buscar na memória o elemento de a (estágio 1), multiplicar o elemento de a por n (estágio 2) e armazenar o resultado no elemento correspondente de b (estágio 3). Observando o resultado após cada ciclo (figura 2.3.b), verifica-se que depois do terceiro ciclo, produz-se um novo elemento de b a cada novo ciclo, representando uma triplicação da velocidade em relação a um esquema sem pipeline.



Estágio 1 ▶ buscar elemento a(i)
Estágio 2 ▶ fazer a(i)*n
Estágio 3 ▶ atribuir a(i)*n para b(i)

Fig.2.3a

Ciclo	Estágio 1	Estágio 2	Estágio 3
1	busca a(1)	—	—
2	busca a(2)	a(1) * n	—
3	busca a(3)	a(2) * n	b(1)←a(1) * n
4	busca a(4)	a(3) * n	b(2)←a(2) * n
5	busca a(5)	a(4) * n	b(3)←a(3) * n
6	busca a(6)	a(5) * n	b(4)←a(4) * n
7	busca a(7)	a(6) * n	b(5)←a(5) * n

Fig. 2.3 b

Fig. 2.3: Técnica de "Pipelining"

Na elaboração de uma máquina SIMD "pura" para processamento vetorial são associados diversos processadores de funcionamento sincronizado sob o comando de uma unidade de controle (figura 2.4). Nesse tipo de processador vetorial todos os processadores efetuam (paralelamente) a mesma instrução sobre diferentes elementos do vetor (ou matriz), por esse motivo são denominados máquinas SIMD "puras", ao

contrário do esquema em "pipeline" onde são realizados diferentes estágios de uma tarefa paralelamente .

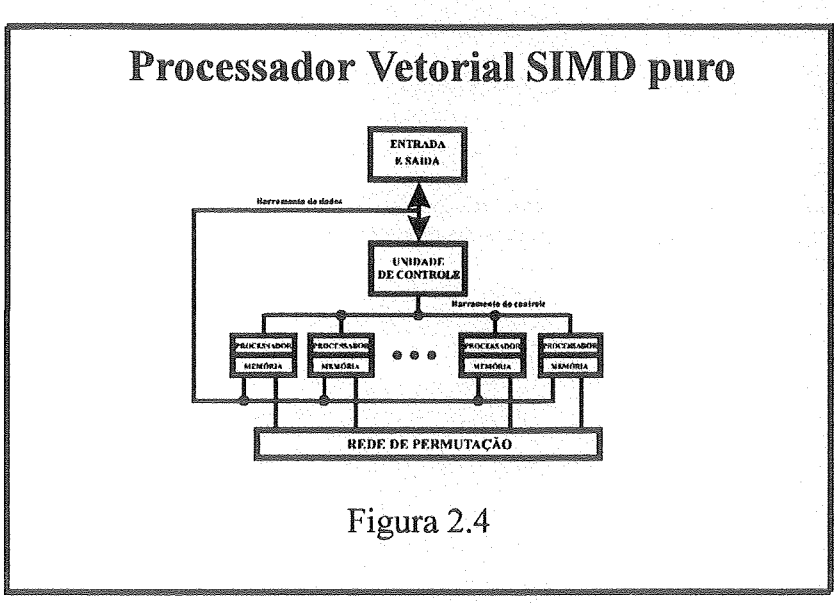


Figura 2.4

O esquema "pipeline" de processadores vetoriais tem sido o preferido na construção de computadores vetoriais. Essa preferência pode ser explicada por dois motivos : o esquema "pipeline" produz altos ganhos de desempenho com custos reduzidos e, além disso, adapta-se melhor ao tratamento de vetores de diversas dimensões. O Processador vetorial no esquema SIMD "puro" apresenta melhor desempenho quando a dimensão do vetor envolvido é um número múltiplo do de processadores, pois nesse caso, a operação sobre os elementos do vetor se dará em etapas com plena utilização dos processadores (sem ociosidade de um ou mais processadores).

2.5.2 **MULTIPROCESSADORES**

O termo multiprocessador refere-se às arquiteturas compostas por um conjunto de processadores independentes e de algum modo conectados entre si e ou com módulos de memória. É grande a variedade de arquiteturas englobadas nessa classificação, porém três aspectos básicos nos permitem tratá-las em grupos

peculiares: a organização da memória, a granularidade e a forma de interconexão dos elementos da arquitetura.

2.5.2.1 MEMÓRIA GLOBAL E MEMÓRIA LOCAL EM ARQUITETURAS PARA PROCESSAMENTO PARALELO

Quando a memória é distribuída entre os processadores na forma de memórias locais, caracteriza-se um sistema levemente acoplado. Por outro lado, quando há uma memória global acessível a todos os processadores diz-se que o sistema é fortemente acoplado. Alguns autores preferem denominar os sistemas levemente acoplados "multicomputadores", mantendo a denominação "multiprocessadores" apenas para os fortemente acoplados.

Os sistemas de memória global, segundo DeCEGAMA(1989), estão sujeitos a dois tipos de conflito de acesso à memória : conflito de software e conflito de hardware. O conflito de software ocorre quando um processador altera um valor da memória global em uso simultâneo por outro processador, possivelmente levando a resultados errôneos; o conflito de hardware se dá pela tentativa de acesso simultâneo de mais de um processador à memória, levando à necessidade de sincronização. O problema de conflito de software no acesso à memória nos sistemas de memória global é um ponto básico a ser considerado no desenvolvimento de software voltado a essa arquitetura e será abordado em detalhes no terceiro capítulo deste trabalho. Outro problema associado às arquiteturas de memória global é o custo elevado da montagem básica e da ampliação de tais sistemas, se comparado ao custo correspondente das arquiteturas de memória local. Além disso, o aumento do número de processadores nas arquiteturas de memória global é frequentemente bastante limitado.

Um aspecto negativo dos sistemas baseados em memória local é o baixo desempenho quando do processamento de programas altamente requerentes de comunicação entre os processadores já que, dependendo da rede de interconexão entre processadores, a troca de dados entre eles pode ser muito lenta, principalmente quando os dados passam por vários processadores até atingir o processador destino. Outro ponto fraco desses sistemas é a pouca disponibilidade de compiladores que

paralelizam automaticamente programas sequenciais. Essa pequena disponibilidade pode ser explicada pela complexidade desses compiladores, os quais necessitam particionar não somente as instruções, mas também os dados entre os processadores.

2.5.2.2 GRANULARIDADE

O conceito de granularidade em computadores de arquitetura paralela refere-se à dimensão e complexidade das tarefas realizadas pelos processadores. No paralelismo de granularidade grossa há poucos (tipicamente menos de dez) processadores executando tarefas complexas, sendo portanto processadores mais complexos. No paralelismo de granularidade fina, por outro lado, os processadores desempenham apenas tarefas mais simples, sendo necessário portanto um grande número deles (até milhares de processadores). Na granularidade fina a tarefa executada por um processador pode ser até mesmo apenas uma operação lógica ou aritmética. Há ainda, entre esses extremos, os casos chamados de granularidade média. As arquiteturas de média granularidade são hoje, segundo DeCEGAMA(1989), forte tendência, pela disponibilidade de processadores de bom desempenho e custo reduzido, viabilizando máquinas com mais de cem processadores.

2.5.2.3 INTERCONEXÃO ENTRE ELEMENTOS DAS ARQUITETURAS PARA PROCESSAMENTO PARALELO

A forma de conexão entre as unidades funcionais (processadores, unidades de memória e dispositivos de entrada e saída) interfere profundamente no desempenho e custo de implementação das arquiteturas para processamento paralelo. Há, segundo ALMEIDA(1991), cinco principais formas de interconexão : barramento, barramento cruzado, rede matricial, hipercubos e redes de conexão multiestágio.

A mais simples e comum forma de se promover a interconexão dos elementos das arquiteturas paralelas é a utilização de um barramento (*bus*) comum a todas as unidades funcionais (figura 2.5). A grande limitação desse sistema é o fato de apenas um acesso (pedido de leitura ou gravação) ser possível a cada instante no barramento, tornando a comunicação sequencial. Esse caráter sequencial da comunicação no sistema de barramento é um limitante de desempenho que se

manifesta de forma crescente com o aumento do número de processadores, implicando na forte limitação do número máximo de processadores conectados nesse sistema.

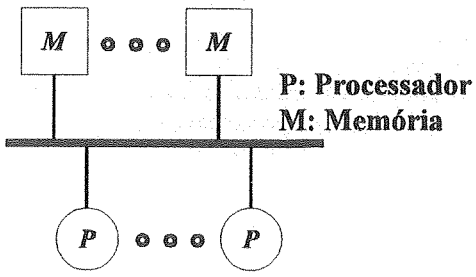


Figura 2.5 Interconexão por barramento (bus)

No sistema de barramento cruzado ou *crossbar* (figura 2.6), a memória é dividida em unidades e é possível aos processadores o acesso simultâneo a diferentes unidades de memória. Embora seja um sistema evidentemente mais eficiente do que o de barramento, paga-se o custo devido ao elevado número de chaveamentos necessários e complexidade de implementação. Na figura 2.6 estão representados os processadores (P_i) e os módulos de memória (M_i).

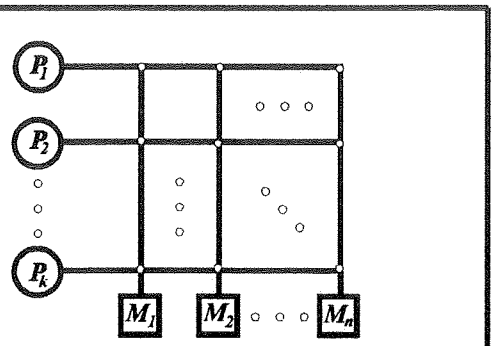


Figura 2.6 : Sistema de barramentos cruzados

Na rede matricial (figura 2.7) cada processador possui quatro conexões, ligando-o a quatro processadores vizinhos. Para esse sistema é típica a utilização de *transputers*. O transputer (*trans*istor for multicom*puter*) é uma unidade especialmente desenvolvida para arquiteturas paralelas composta de : unidade central de processamento, unidade de memória, unidade externa de memória de interface e quatro controladores para conexão (*link controllers*). Geralmente os transputers são agrupados em placas e trabalham conectados a um computador (normalmente IBM PC, aumentando o poder de tais máquinas).

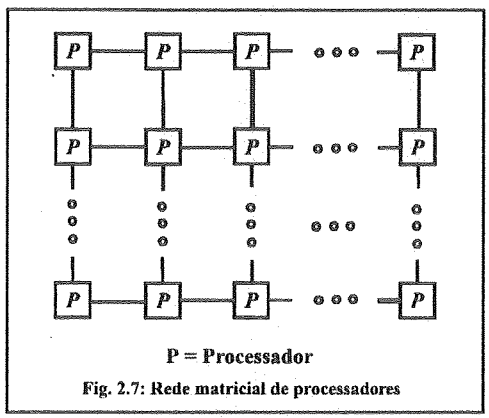


Fig. 2.7: Rede matricial de processadores

No arranjo em hipercubo de dimensão "n", existem 2n processadores, cada um deles ligado a "n" processadores (figura 2.8, hipercubo com n=3). Segundo DeCEGAMA(1989), numerosas aplicações científicas que utilizam "malhas" ou "árvores"são candidatas a eficientemente implementáveis no esquema de hipercubo.

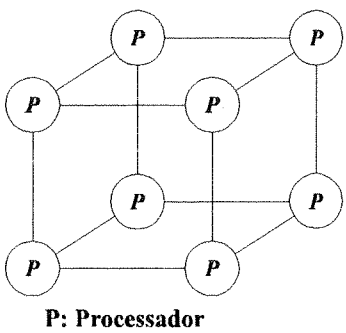


Fig. 2.8: Interconexão por hipercubo

Outra possibilidade de interconexão é a utilização de rede de interconexão múltiestágio (figura 2.9). A idéia desse esquema é promover a conexão de qualquer processador a qualquer unidade de memória, porém a um custo inferior ao do barramento cruzado. O custo inferior decorre do fato de que a comunicação entre um processador e uma unidade de memória se dá em vários estágios (mais lentamente portanto), e não diretamente como no caso do esquema de barramento cruzado. Por exemplo, para se efetuar a conexão de "n" processadores a "n" unidades de memória são necessárias n^2 chaves no esquema barramento cruzado e da ordem de nlog(n) chaves no esquema multiestágio.

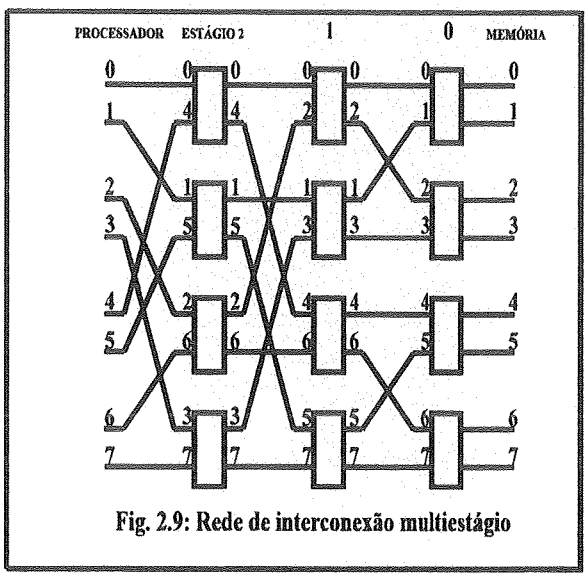


Fig. 2.9: Rede de interconexão multiestágio

2.6 MEDIDAS DE DESEMPENHO PARA COMPUTADORES DE ARQUITETURA PARALELA

A maioria das aplicações às quais se destinam os computadores de arquitetura paralela trabalha intensamente com números reais e, desta forma, a métrica para comparação de desempenho entre esses computadores tem sido frequentemente a quantidade de milhões de operações por segundo efetuadas com números reais (Mflops). Os fabricantes geralmente apresentam o chamado "desempenho de pico" de suas máquinas em temos de Mflops. Esse desempenho, como o próprio nome já o diz, é o número máximo de operações com números reais atingível pela máquina em um segundo e, obviamente não representa o desempenho

que a máquina terá em aplicações específicas. A tabela 2.1, extraída de ALMEIDA(1991) , mostra o desempenho de pico de alguns computadores.

Tabela 2.1 - Desempenho de pico

Modelo	Desempenho de pico (Mflops)
Cray Y-MP 832	2667
Hitachi S-820/80	3000
NEC SX-2	1300
IBM 3090/600J-VF	828
Fujitsu VP-200	533
Convex C-240	200

Para estabelecer-se uma comparação mais consistente do desempenho de computadores de variadas arquiteturas (incluindo arquiteturas paralelas) é usual a utilização do ***benchmark***. Esta técnica avalia o computador através do desempenho do mesmo na execução de um conjunto bem definido de programas. Normalmente a avaliação é feita hierárquicamente desde o processamento de operações básicas até a execução de aplicações completas. Alguns ***benchmarks*** são hoje padrões internacionais. Dentre eles destacam-se o ***Livermore Loops***, o ***Linpack***, os ***Los Alamos Benchmark Codes*** e o ***Perfect Club***.

Ao submeter vários modelos de computadores a ***benchmarks*** diversos, nem sempre o desempenho relativo entre eles se manterá, ou seja, um computador pode apresentar o melhor desempenho em determinado benchmark e não repetir o fato em outro. Assim, fica clara a quase impossibilidade de se obter apenas um único número para definir efetivamente o desempenho de um computador.

3 - DESENVOLVIMENTO DE SOFTWARE PARALELO

3.1 INTRODUÇÃO

O desenvolvimento de software sequencial é hoje tarefa assistida por numerosas técnicas oriundas da engenharia de software, guiando desde a análise de requisitos até a etapa de testes e manutenção. Várias dessas técnicas, no entanto, não se aplicam ao desenvolvimento de software paralelo, já que pode ser considerada válida a afirmação de que a programação sequencial é um caso particular da programação paralela.

O trabalho de quem desenvolve software paralelo é conceber a solução do problema na forma de execução concorrente de processos e garantir a correta e confiável execução paralela do programa correspondente. A análise sob uma ótica paralelizante confiável só se dá, no entanto, quando o programador já possui consolidados os conceitos básicos da programação paralela, os quais são apresentados neste capítulo.

3.2 PECULIARIDADES DO DESENVOLVIMENTO DE ALGORITMOS PARALELOS

3.2.1 AVALIAÇÃO DE DESEMPENHO

A medida usual de desempenho no processamento de programas ou trechos de programas paralelos, é o *Speed-up* para "n" processadores , definido por:

$$Speed\text{-}up = \frac{Tempo\ de\ processamento\ com\ 1\ processador}{Tempo\ de\ processamento\ com\ n\ processadores}$$

Aliada à medida de Speed-up aparece a medida de **Eficiência**, sendo definida por :

$$Eficiência = \frac{100 . Speed\text{-}up}{número\ de\ processadores} (\%)$$

O limite superior de *Speed-up* obtível na paralelização de um programa é dado pela chamada "lei de AMDAHL(1967)" , a qual estabelece" :

$$Speed\text{-}up \leq \frac{1}{s + \frac{r}{n}}$$

onde "r"é a fração (em termos de tempo) paralelizável do programa, "s" é a fração sequencial restante (s = 1- r) e "n" o número de processadores. Pela "lei de Amdahl", se um programa é 50% paralelizável em termos de tempo (s = r = 0.5), o máximo *Speed-up* atingível é 2, mesmo para elevado número de processadores (basta verificar que no limite para "n" tendendo a infinito, a expressão da "lei de Amdahl" tende a 2).

São dois os principais fatores que, na prática, tornam o limite de *Speed-up* imposto pela "lei de Amdahl" inatingível : o primeiro é o tempo gasto pelo sistema operacional na preparação para a execução em paralelo dos processos ; o segundo é a degradação de desempenho oriunda da necessidade de sincronização entre os processos, fazendo que um ou mais processadores fique em "estado de espera". Exemplificando, nos sistemas de memória global, há necessidade de sincronização no acesso a recursos compartilhados (tipicamente acesso à memória global) e, no caso de sistemas de memória local, a comunicação entre os processos é um caso de atividade sincronizada. Com o aumento do número de processadores há o consequente incremento na necessidade de sincronização, implicando portanto na diminuição da eficiência.

3.2.2 DEPENDÊNCIA DE DADOS

Quando para a correta execução paralela de processos é necessária sincronização para evitar conflitos no acesso a variáveis compartilhadas, ou mesmo

quando há operações que dependem da ordem em que os processos são executados, diz-se haver dependência de dados entre esses processos. Os problemas oriundos da dependência de dados ficam muitas vezes ocultos na fase de testes do software paralelo, pois em vários processamentos do mesmo código sobre os mesmos dados o resultado poderá ser correto, manifestando-se a incorreção apenas em um ou outro processamento isolado, já que vários fatores (interrupções, por exemplo) podem alterar a marcha de execução de um ou mais processos concorrentes. Esse caráter não determinístico da ocorrência de erros provocados pela dependência de dados implica em cuidadosa análise do problema no projeto do software paralelo, sob pena da não validação do mesmo por deficiência de confiabilidade.

O exemplo clássico de dependência de dados é o caso de dois processos concorrentes que podem atualizar a mesma variável. Por exemplo, seja um programa com um trecho sequencial onde é atribuído o valor zero à variável global "carga". Após o trecho sequencial há um trecho paralelo com dois processos ("P1"e "P2") que alteram a variável carga, conforme o esquema a seguir :

TRECHO SEQUENCIAL

carga ← 0.0

TRECHO PARALELO

PROCESSO P1	**PROCESSO P2**
carga ← carga + 10	**carga ← carga + 20**

Após o término dos dois processos concorrentes, a variável carga poderá conter o valor dez, vinte ou trinta (valor correto), dependendo da sequência em que ocorrem as leituras, somas e atribuições à variável carga. Para que após a execução de P1 e P2 o valor de carga seja dez, uma sequência possível é :

P1 : lê o valor de carga (zero)

P2 : lê o valor de carga (zero)

P2 : Soma vinte ao valor de carga lido (zero) e atribui valor vinte para carga

P1 : Soma dez ao valor de carga lido (zero) e atribui valor dez para carga.

O conceito de dependência de dados é fundamental para a correta utilização de *loops* paralelos. Quando um *loop* é processado paralelamente, divide-se o número de iterações do mesmo entre os processadores, para que essas iterações possam ser efetuadas concorrentemente. Desta forma, para a correta execução paralela de loops, é essencial a não existência de dependência de dados entre as iterações. Para não haver dependência de dados no caso de *loop*, nenhuma iteração do mesmo pode escrever em uma posição de memória que é lida por outra iteração. Quando as iterações são isentas de dependência de dados, a sequência em que as mesmas são executadas não interfere no resultado, implicando em poderem ser executadas concorrentemente. Para exemplificar a ocorrência ou não de dependência de dados em *loops*, considerem-se os dois exemplos apresentados a seguir :

Exemplo 1> *loop* sem dependência de dados :

```
PARA i=1 até N FAÇA
   a[ i ] = b[ i ] + 1
FIM_FAÇA
```

Exemplo 2> *loop* com dependência de dados :

```
PARA i=2 até N FAÇA
   a[ i ] = a[ i - 1 ]
FIM_FAÇA
```

No primeiro exemplo, a variável que é escrita (a[i]) não é lida em nenhuma outra iteração, caracterizando a não ocorrência de dependência de dados. No segundo exemplo, por outro lado, a variável escrita (a[i]) é sempre lida pela iteração i+1 e, assim, a i-ésima iteração não poderá ser executada sem que tenha sido concluída a iteração anterior, caracterizando claramente a dependência de dados no *loop*.

3.2.3 SINCRONIZAÇÃO

Quando ocorre a dependência de dados, é necessária a sincronização dos processos para obterem-se resultados corretos oriundos de processos concorrentes. Sincronizar processos significa fazer com que executem suas tarefas em momentos apropriados, evitando-se por exemplo que dois ou mais processos tentem simultaneamente atualizar uma variável compartilhada. Tipicamente, os instrumentos

de sincronização disponíveis nas linguagens paralelas de programação interrompem e liberam trechos de processos que possuem dependência de dados. Os mais usuais instrumentos de sincronização são as barreiras (*barriers*), os bloqueios (*locks*) e os semáforos (*semaphores*).

3.2.3.1 BARREIRAS (*barriers*)

O objetivo da utilização de uma barreira é fazer com que o processamento não passe de determinada instrução (a barreira) até que todos os processos cheguem a essa instrução. Normalmente a barreira é necessária quando os processos concorrentes anteriores a ela produzem um resultado sem o qual o processamento não pode prosseguir, tornando-se assim imprescindível a conclusão de todos esses processos. Por exemplo, seja um trecho de programa que calcula paralelamente (com 4 processadores) a norma Euclidiana de um vetor a de dimensão 100, dado a seguir :

```
INÍCIO_DOS_PROCESSOS_PARALELOS
    i = número do processo
    soma[ i ] = 0.0
    PARA j = ( i - 1 )*25+1  até i*25 FAÇA
        soma[ i ] = soma[ i ]  +  a( j ) * a( j )
    FIM_FAÇA
FIM_PROCESSOS_PARALELOS
BARREIRA
soma_total = soma[ 1 ] + soma[ 2 ] + soma[ 3 ] + soma [ 4 ]
norma = raiz_quadrada (soma_total )
```

Nesse exemplo, enquanto os processos paralelos não estiverem concluídos não estarão disponíveis soma[1], soma[2], soma[3] e soma[4] e, consequentemente não poderá ser efetuada a soma que resulta em "soma_total", justificando a inclusão do comando **BARREIRA** antes dessa soma.

3.2.3.2 BLOQUEIOS (*locks*)

Os bloqueios são instrumentos de sincronização que permitem a exclusão mútua de processos concorrentes em trechos onde ocorre a dependência de dados. Para o entendimento de sua aplicação, apresentar-se-á uma nova versão do exemplo do ítem

3.2.2, onde o problema da dependência de dados é tratado com o uso de bloqueios :

TRECHO SEQUENCIAL

carga ← 0.0

TRECHO PARALELO

PROCESSO P1	PROCESSO P2
BLOQUEIO	**BLOQUEIO**
carga ← carga + 10	carga ← carga + 20
DESBLOQUEIO	**DESBLOQUEIO**

Imaginado-se, sem perda de generalidade, que o processo P1 chega antes de P2 ao comando bloqueio, a partir desse momento, quando P2 atingir esse comando ficará bloqueado até que o processo P1 passe pelo comando "DESBLOQUEIO", garantindo a exclusão mútua no trecho em que é feito o incremento da variável carga e, por conseguinte, proporcionando a correta execução do programa.

3.2.3.3 SEMÁFOROS (*semaphores*)

A idéia da criação de semáforos para a sincronização de processos é devida a DIJKSTRA(1968). O semáforo é um valor inteiro "S" (inicialmente 1) que pode ser obtido através de duas operações atômicas "P(S)" e "V(S)" ou seja, operações indivisíveis que são completamente realizadas sem interferências de outros processos. Quando um processo vai entrar em "área crítica", aplica a operação P(S), que inicialmente indica o valor de S. Permite-se ou bloqueia-se o acesso à "área crítica" respectivamente no caso de S=1 e S=0. Quando S=1 e o processo entra na "área crítica", o valor de S passa para 0 e assim, nenhum outro processo subsequente poderá entrar concorrentemente na área. Ao sair da área crítica o processo aplica a operação V(S) que novamente atribui 1 para S, liberando o acesso à área crítica.

O exemplo de sincronização apresentado no ítem 3.2.3.2 (bloqueios) pode também ser facilmente implementado com a utilização de semáforos :

TRECHO SEQUENCIAL

inicialização_do_semáforo(s=1)

carga ← 0.0

TRECHO PARALELO

PROCESSO P1	PROCESSO P2
P(s)	**P(s)**
carga ← carga + 10	carga ← carga + 20
V(s)	**V(s)**

No trecho sequencial ocorre a inicialização do semáforo (s=1). O primeiro processo que chega na área crítica aplica a operação P(s), verifica que s=1 e entra na área crítica atribuindo valor 0 para s. O outro processo fica então impedido de entrar na área crítica até o primeiro executar a operação V(s) que novamente atribui valor 1 para s. A diferença entre esse esquema e o do bloqueio (*lock*) é que ao encontrar s=0 (impedido) o processo, no caso do semáforo, não precisa, necessariamente, permanecer aguardando a liberação, podendo executar outras instruções.

3.2.4 BALANCEAMENTO DE TRABALHO

O desenvolvimento de algoritmos paralelos é, basicamente, uma distribuição entre processadores de tarefas para processamento concorrente. Porém, na busca de desempenho (*Speed-up*), mais do que simplesmente distribuir tarefas passíveis de processamento concorrente, deve-se fazer com que essas tarefas representem esforços computacionais semelhantes para os processadores, ou seja, deve-se prover um balanceamento de trabalho. Não havendo uma distribuição igualitária de tarefas, os processadores submetidos a esforços computacionais menores concluem

antecipadamente suas respectivas tarefas, mantendo-se ociosos até a finalização de todas as tarefas atribuídas aos demais processadores, levando consequentemente a uma perda de desempenho.

- É quase sempre possível alguma interferência para que o algoritmo tenha melhor balanceamento de trabalho. Por exemplo, seja um trecho de algoritmo sequencial que soma os elementos a partir da diagonal principal de cada linha de uma matriz quadrada (matriz **A**, de dimensão nxn, sendo n número par), inserindo a soma em um elemento de vetor correspondente à referida linha (vetor **b**, de dimensão n). Em "pseudo código" esse algoritmo pode ser escrito como :

```
PARA  i = 1 até n FAÇA
   soma = 0.0
   PARA j = i até n FAÇA
       soma = soma + a( i , j )
   FIM_FAÇA
  b( i ) = soma
FIM_FAÇA
```

Esse algoritmo é composto por um aninhamento de dois *loops*. Cada iteração do *loop* mais externo (variável "i ") altera elemento diferente do vetor **b** e, portanto, desde que os índices dos *loops* e a variável "soma" sejam armazenados em áreas locais de memória, cada iteração do *loop* mais externo é independente das outras, podendo o *loop* externo (e assim a totalidade do algoritmo) ser processado em paralelo. Imaginando que o algoritmo será aplicado sobre uma matriz de dimensões 100x100 e que serão utilizados 5 processadores em paralelo, caberá a cada um deles somar 20 linhas. A soma sobre cada linha, porém só é realizada a partir da diagonal principal e, desta forma, a primeira linha requer 100 operações de soma, a segunda 99, a vigésima 81, implicando em diferente esforço computacional no processamento de cada linha .

Normalmente os loops processados em paralelo dividem o total das iterações entre os processadores em sequência, ou seja, o primeiro processador no exemplo ficaria com as vinte primeiras iterações; o segundo com a iteração vinte até quarenta e ao quinto (último), caberia processar as vinte últimas iterações. Com essa divisão das iterações o primeiro processador seria submetido a esforço computacional muitíssimo

superior ao do último, e este ficaria ocioso rapidamente no processamento do loop (o primeiro realizaria 1810 adições e o último apenas 190). Pode-se alterar o algoritmo dividindo o *loop* que vai de 1 até n, para um novo loop de 1 até n/2, processando 2 linhas a cada vez (linha "i" e linha "n-i+1"), como apresentado a seguir :

```
PARA i = 1 até n/2 FAÇA
    soma1 = 0.0
    PARA  j = i até n FAÇA
        soma1 = soma1 + a (i , j )
    FIM_FAÇA
    b( i ) = soma1
    soma 2 = 0.0
    iaux = n - i + 1
    PARA k = iaux até n FAÇA
       soma2 = soma2 + a( k , j )
   FIM_FAÇA
   b( iaux ) = soma2
FIM_FAÇA
```

onde i , j , soma1, soma2 e iaux são variáveis locais aos processadores.

Nesse algoritmo, a cada iteração é processada a linha "i" que requer "n-i+1" adições e a linha "n-i+1", a qual requer "i-1" adições. Desta forma o número total de adições para o processamento de qualquer das iterações é de "n" adições e assim, para qualquer distribuição de iterações entre os processadores, obter-se-á um bom balanceamento de trabalho.

Alguns compiladores permitem ao programador estabelecer diferentes tipos de escalonamento de trabalho (*schedules*) para os processadores na execução paralela de *loops*, entre eles o escalonamento intercalado e o escalonamento dinâmico. Utilizando *schedule* intercalado (*interleave*), o processamento do loop com 100 iterações e 5 processadores seria feito da seguinte maneira : o processador 1 executaria as iterações 1,6,11,16...; o processador 2, as iterações 2,7,12,17..., e assim por diante. Com o escalonamento dinâmico, em tempo de processamento, o sistema aloca na maneira mais otimizada as iterações para os processadores (evitando ociosidade), entretanto esse escalonamento requer considerável esforço computacional para sua realização, necessitando prévia avaliação de viabilidade de uso.

3.3 FERRAMENTAS PARA DESENVOLVIMENTO DE SOFTWARE PARALELO

3.3.1 AUTOPARALELIZADORES

Desde a sua criação nos anos 60 até o presente, a linguagem FORTRAN tem sido a preferida no meio científico e obviamente há uma grande quantidade de programas sequenciais de grande interesse codificados nessa linguagem. Muitos desses programas envolvem vultoso processamento numérico, implicando em alto tempo de processamento, e desta forma, tais programas certamente tornar-se-iam mais utilizáveis se partes de seu processamento pudessem ser efetuadas em paralelo, reduzindo o tempo total de execução.

Atualmente, embora hoje seja razoável a disponibilidade de computadores de arquitetura paralela, a nova codificação dos antigos programas sequenciais em FORTRAN, voltada a processamento paralelo, é sem dúvida fato raro por dois motivos: o primeiro é a escassa documentação e a complexidade de tais programas, onde pequenas modificações no código fonte podem ter efeitos colaterais imprevisíveis; o segundo é a natural resistência feita pelos programadores à necessidade de aquisição de novos conhecimentos em informática para promover a otimização de um programa que já funciona sequencialmente. Para possibilitar, sem necessidade de nova codificação, a paralelização de trechos desses programas, foram criados os autoparalelizadores (*parallelising compilers*).

Os autoparalelizadores são compiladores (normalmentes voltados a sistemas de memória global) que, automaticamente, localizam no código sequencial, trechos (usualmente *loops* do FORTRAN) paralelizáveis, ou seja *loops* em que as iterações são independentes (uma iteração não influi na outra) , gerando um código executável com paralelização dos trechos viáveis. Alguns autoparalelizadores inclusive fornecem uma listagem mostrando os *loops* paralelizados e os motivos da não paralelização dos outros *loops*. Contrapondo-se à sua praticidade, o desempenho do código gerado pelos autoparalelisadores atuais é de certa forma limitado, pois só são reconhecidas as mais evidentes paralelizações possíveis .

3.3.2 LINGUAGENS PARALELAS DE PROGRAMAÇÃO

3.3.2.1 PARADIGMAS

Conforme apresentado em FREEMAN(1992), uma linguagem paralela de programação é aquela projetada para aproveitar as facilidades oferecidas pelas arquiteturas paralelas. A organização da memória (compartilhada ou local) da arquitetura destino do código é um ponto de diferenciação entre linguagens paralelas, havendo paradigmas de linguagens voltadas às duas formas básicas de organização.

Para as linguagens paralelas voltadas aos sistemas de memória compartilhada, é desejável a possibilidade de :

- Criação e destruição de processos (dinâmica e estaticamente),
- Definr variáveis locais e globais aos processos e
-Permitir sincronização entre os processos com a utilização de travamentos, barreiras, semáforos e etc.

Os sistemas baseados em memória local requerem linguagens paralelas que ofereçam facilidades para :

- Envio e recebimento de mensagens entre processos,
- Envio de mensagens de um processo para todos os outros,
- Comunicação síncrona (o processo que envia aguarda a preparação do que recebe a mensagem),
- Comunicação assíncrona (o processo que envia não aguarda a preparação do que recebe a mensagem, podendo a mensagem ficar armazenada em *buffer*) e
- Associar um processo a um processador específico.

3.3.2.2 LINGUAGENS PARALELAS CRIADAS COMO EXTENSÃO DE LINGUAGENS SEQUENCIAIS

O uso intenso da linguagem FORTRAN no meio científico tem estimulado o aparecimento de versões extendidas da mesma com vistas à utilização de computadores de arquitetura paralela. As versões extendidas do FORTRAN (e de outras linguagens como Pascal e C, por exemplo) são, basicamente, a mesma linguagem sequencial com a inclusão de comandos especiais que permitem o processamento paralelo de trechos de programa e fornecem meios de sincronização dos processos. Há versões extendidas

de FORTRAN voltadas para os dois tipos básicos de organização de memória (global e local), embora haja clara prevalência da arquitetura de memória global nos computadores de alto desempenho voltados ao processamento de problemas numéricos.

Dentre as extensões de FORTRAN destinadas a sistemas de memória global podemos citar a *Encore parallel* FORTRAN e a extensão de FORTRAN 77 utilizada neste trabalho, a qual destina-se a estações *Silicon Graphics* do tipo *power 4S-440D* e é apresentada com detalhes no anexo . Tipicamente, essas linguagens possuem comandos para delimitar trechos a serem processados em paralelo do tipo :

PARALLEL /* início do trecho paralelo (são criados os processos) */
Declarações /* geralmente especificações de variáveis locais ou globais */
Comandos
END PARALLEL /* final do trecho paralelo */

e comandos específicos para processamento paralelo de *loops*, por exemplo :

DO ALL i = 1, 10
Declarações /* geralmente especificações de variáveis locais ou globais */
Comandos
END DO ALL.

Além disso, são disponíveis comandos para sincronização de processos do tipo barreiras, bloqueios e semáforos. Por exemplo, no trecho apresentado a seguir, o processamento só passará do comando **BARRIER** (barreira) quando todos os processos o atingirem (o processo que chegar antes aguarda os mais lentos) :

Comando
Comando
BARRIER.

No caso de extensões FORTRAN para computadores paralelos dotados de processador vetorial, há instruções específicas para tratamento de vetores. Essas instruções são reconhecidas pelos processadores e enviadas ao processador vetorial disponível.

Extensões do FORTRAN voltadas a sistemas paralelos de memória local são também disponíveis e dentre elas aparece o *intel iPSC/2*, destinado a arquitetura hipercubo e o *3L FORTRAN*, destinado a sistemas compostos por *transputers*.

A utilização de extensões de FORTRAN para arquiteturas paralelas tem a óbvia vantagem de poupar os programadores do aprendizado de uma linguagem inteiramente nova, cabendo aos mesmos apenas a concentração de esforços no aprendizado das técnicas de programação paralela. A desvantagem é a pequena portabilidade dessas linguagens.

3.3.2.3 LINGUAGENS DE PROGRAMAÇÃO ORIGINALMENTE PARALELAS

As linguagens originalmente paralelas são aquelas cujo projeto inicial já prevê a utilização da mesma em arquiteturas paralelas. Dentre essas linguagens, há um claro destaque de ADA. Essa poderosa linguagem de alto nível foi desenvolvida ainda na década de 70 para o departamento de defesa dos Estados Unidos , visando a elaboração de sistemas de processamento em tempo real em aplicações militares. Embora hoje ADA seja um padrão nas forças armadas dos Estados Unidos, a sua efetiva utilização foi se dando lentamente, principalmente pela pequena disponibilidade inicial de bons compiladores.

O projeto da linguagem ADA não foi direcionado a um tipo particular de arquitetura, sendo que a mesma dispõe de instrumentos para comunicação entre processos (comuns nos sistemas de memória local) e mecanismos de sincronização no acesso `a memória global . A construção que permite o processamento concorrente em ADA é a *task* (tarefa). O trecho de código contido na *task* pode ser executado sequencial ou concorrentemente, sendo que há comando especial (*entry*) para promover a comunicação entre as tarefas concorrentes.

4 - PARALELIZAÇÃO DE ETAPAS TÍPICAS DO MÉTODO DOS ELEMENTOS FINITOS EM ANÁLISE ESTRUTURAL: A MONTAGEM DA MATRIZ DE RIGIDEZ GLOBAL

4.1- INTRODUÇÃO :

Nos últimos anos tem sido vultoso o desenvolvimento dos computadores com arquitetura para processamento paralelo. A área de análise numérica de estruturas foi uma das que mais se beneficiou com a chegada de tais máquinas, já que vários algoritmos complexos outrora sequenciais apenas pela indisponibilidade dos computadores paralelos, hoje podem ter uma versão paralela, implicando em sensível redução do tempo de processamento.

Quando da utilização do método dos elementos finitos, destaca-se a possibilidade de se efetuar, através de algoritmo paralelo, algumas de suas etapas típicas de maior esforço computacional, como por exemplo a montagem da matriz de rigidez da estrutura e a solução do sistema de equações lineares . Os algoritmos paralelos desenvolvidos para tais etapas, no entanto, tem apresentado empecilhos que por certo mostram a necessidade de reanálise do método dos elementos finitos sob uma ótica paralelizante e não simplesmente o desenvolvimento de algoritmos paralelos por pequenas alterações nos algoritmos sequenciais existentes.

Para a etapa da montagem da matriz de rigidez da estrutura em computadores de arquitetura MIMD e memória global especificamente, vários algoritmos paralelos têm sido desenvolvidos a partir de pequenas variações de algoritmos sequenciais usuais e pode-se dizer que o que os diferencia entre si é o tratamento dado ao problema da possibilidade de mais de um processador, simultaneamente, atualizar a mesma porção

da matriz de rigidez da estrutura, levando a resultados errôneos. Esse problema é denominado *racing condition*.

Neste capítulo são apresentados os algoritmos já existentes para a montagem paralela da matriz de rigidez da estrutura, os quais se caracterizam por adaptações do algoritmo usual sequencial e, em seguida, é proposto um algoritmo alternativo no qual o problema da atualização simultânea de elementos da matriz de rigidez da estrutura é eliminado sem a necessidade de sincronização, implicando em ganho de desempenho. É introduzido também o conceito de abordagem nodal plena, mais adequado à geração de matrizes de rigidez globais de malhas irregulares ou oriundas de análise não linear. Propõe-se ainda a inclusão da etapa de imposição de condições de contorno à etapa de montagem da matriz de rigidez global. Finalmente é avaliado o desempenho (*speed-up)* dos algoritmos propostos.

4.2 ALGORITMOS PARALELOS PARA MONTAGEM DA MATRIZ DE RIGIDEZ DA ESTRUTURA ORIUNDOS DO ALGORITMO SEQUENCIAL USUAL

4.2.1 O ALGORITMO SEQUENCIAL USUAL

O algoritmo sequencial usual de montagem de matriz de rigidez da estrutura (aqui denominado usual por ser o algoritmo apresentado nas referências básicas do método dos elementos finitos, em particular em ZIENKIEWICZ(1967) e SORIANO(1981)) reflete a própria idéia básica do método dos elementos finitos : a matriz de rigidez da estrutura é montada pela somatória das contribuições dos elementos que formam a malha. Assim, o algoritmo sequencial usual é resumidamente, um *loop* que para cada elemento da malha vai somando as equações da matriz de rigidez do mesmo às equações correspondentes da matriz de rigidez global da estrutura.

Exemplificando o processo usual de montagem da matriz de rigidez da estrutura, considere-se a figura 4.1 onde aparece uma placa discretizada em elementos finitos triangulares de três pontos nodais, localizados nos vértices dos triângulos. Considere-se também que o elemento contém três parâmetros por ponto nodal. Na figura 4.1 está representada a inserção na matriz de rigidez global da placa das equações da matriz de rigidez do elemento triangular de número 3. Na aplicação do algoritmo usual, o

processo apresentado para o elemento 3 seria aplicado sequencialmente a todos os elementos da malha. O algoritmo 4.1 traz em "pseudo código" a montagem sequencial usual da matriz de rigidez da estrutura .

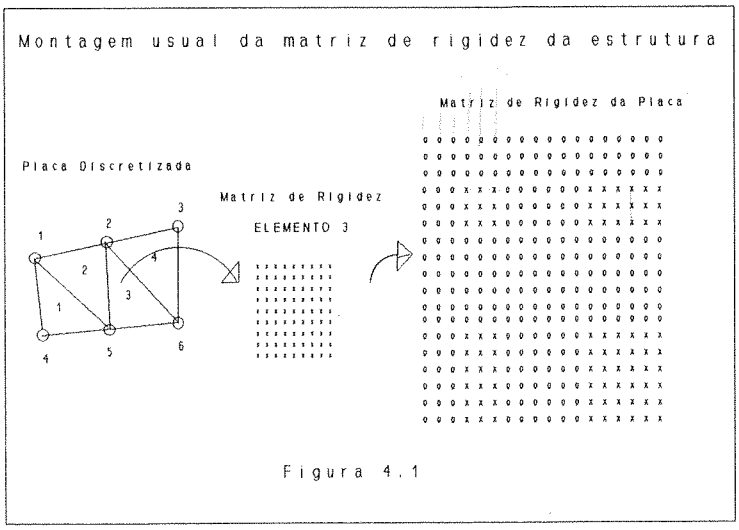


Figura 4.1

ALGORITMO 4.1 - MONTAGEM SEQUENCIAL USUAL DA MATRIZ DE RIGIDEZ GLOBAL DA ESTRUTURA :

NPON=NÚMERO DE PONTOS DO ELEMENTO

NPAR=NÚMERO DE PARÂMETROS NODAIS DO ELEMENTO

NP[i,k]=número do k-ésimo ponto nodal do elemento i

MATG[i,j]= matriz de rigidez global (da estrutura)

MATL[i,j]= matriz de rigidez do elemento i (disponível na memória).

(continuação algoritmo 4.1)

```
1   INÍCIO
2   PARA i= 1 ATÉ NÚMERO_DE_ELEMENTOS_DA_MALHA FAÇA
3     PARA j=1 ATÉ NPON FAÇA
4        ponto=NP[i,j]
5        linha_base = NPAR*ponto - ( NPAR- 1 )
6        PARA k=1 ATÉ NPON FAÇA
7           ponto_aux=NP[i,k]
8           coluna_base = NPAR*ponto_aux - ( NPAR-1 )
9           PARA m=1 ATÉ NPAR FAÇA
10            LL  = NPAR*( j-1 ) + m
11            LG = ( linha_base - 1 ) + m
12            PARA n=1 ATÉ NPAR FAÇA
13               CL  = NPAR*( j-1 ) + n
14               CG = ( coluna_base - 1 ) + n
15               c_band = CG - LG + 1
16               SE c_band > 0 ENTAO
17                    MATG[LG,c_band] = MATG[LG,c_band] + MATL[LL,CL]
18               FIM_SE
19            FIM_FAÇA
20          FIM_FAÇA
21       FIM_FAÇA
22    FIM_FAÇA
23  FIM_FAÇA
24  FIM
```

4.2.2- O PROBLEMA DA ATUALIZAÇÃO SIMULTÂNEA DA MATRIZ DE RIGIDEZ GLOBAL NA PARALELIZAÇÃO DO ALGORITMO USUAL

Conforme foi apresentado, algoritmo usual é basicamente um *loop* que vai inserindo uma a uma a matriz de rigidez dos elementos finitos da malha na matriz global. Assim, a forma aparentemente mais fácil de se paralelizar tal algoritmo é distribuir entre os processadores porções de elementos finitos a serem inseridos na matriz global. A única a mudança a ser feita no algoritmo usual (Algoritmo 4.1) anteriormente apresentado seria a inclusão da expressão " EM PARALELO " ao *loop* mais externo (algoritmo 4.2).

ALGORITMO 4.2 : PARALELIZAÇÃO DO ALGORITMO USUAL DE MONTAGEM DA MATRIZ DE RIGIDEZ GLOBAL : SUJEITO A *RACING CONDITION*

(continuação algoritmo 4.2)

INÍCIO

PARA i= 1 ATÉ NÚMERO_DE_ELEMENTOS_DA_MALHA **FAÇA_EM_PARALELO**

Linha 3 até linha 22 do algoritmo 4.1

FIM_FAÇA_EM_PARALELO

FIM

Entretanto, essa forma de paralelização pode, potencialmente, levar ao problema de atualização simultânea da matriz de rigidez global quando processadores distintos estão paralelamente inserindo matrizes de rigidez de elementos com pelo menos um ponto nodal em comum. Nesse caso porções das matrizes de rigidez dos elementos serão inseridas em posições comuns da matriz global e, como esse procedimento estará se realizando concorrentemente, há a possibilidade de atualização simultânea e com ela a possibilidade de erros.

Um exemplo de possibilidade de atualização simultânea aparece na figura 4.2. Nela está representada a montagem paralela da matriz de rigidez global de uma placa utilizando-se dois processos concorrentes (processo 1 e processo 2). No exemplo os processos 1 e 2 estão concorrentemente inserindo respectivamente as contribuições dos elementos finitos 1 e 3. As posições da matriz de rigidez global em que serão inseridos termos da matriz de rigidez do elemento 1 são representadas por "O"; já as posições correspondentes à matriz de rigidez do elemento 3 são representadas por "X". Deve ser observado que há posições na matriz de rigidez global que são comuns aos elementos 1 e 3. Essas posições comuns (representadas por $\otimes$), encontram-se nas equações referentes ao ponto nodal 5 que é um ponto de ligação entre os elementos 1 e 3. Desta forma, estando os processos 1 e 2 concorrentemente inserindo as equações dos elementos 1 e 3 e havendo posições comuns de inserção na matriz global é possível a ocorrência do problema da atualização simultânea.

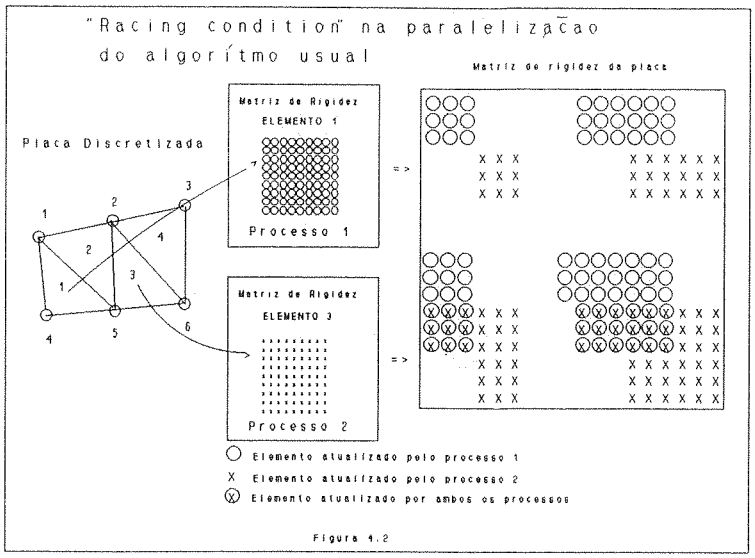


Figura 4.2

4.2.3 PROPOSTAS DE PARALELIZAÇÃO DO ALGORITMO USUAL

Para a correta paralelização do algoritmo usual de montagem da matriz de rigidez global é essencial uma estratégia de sincronização que evite a *racing condition*. O tratamento de tal problema tem gerado soluções diversas que vão da simples utilização de instrumentos de sincronização (semáforos, bloqueios e etc.) até a especial numeração dos elementos . Destacam-se as propostas de CHIEN & SUN(1989) e ADELI & KAMAL(1992), as quais são resumidamente apresentadas a seguir.

CHIEN & SUN(1989) propuseram o tratamento do problema da atualização simultânea da matriz por uma numeração especial dos elementos e na divisão da estrutura em subestruturas relacionadas a cada um dos processos concorrentes. Fazendo que em cada subestrutura os primeiros elementos e os últimos sejam os que têm pontos nodais em comum com os elementos das outras subestrututras e, fazendo com que os diversos processadores efetuem a montagem nessa ordem, os mesmos chegarão aos pontos comuns das subestruturas em tempos distintos. Essa solução, embora em princípio dispense o uso de sincronização, é problemática para o caso de muitos processadores montando uma estrutura formada por poucos elementos. Além disso há um excedente de trabalho para dividir-se a malha da maneira requerida pelo algoritmo. Em outra proposta, os mesmos autores sugerem a utilização de bloqueios (*locks*) nos

trechos de código onde ocorre a atualização de termos da matriz de rigidez global. Com a utilização dos bloqueios, quando um processo entra na região de código de atualização (área crítica), todos os outros processos ficam impedidos de entrar concorrentemente em tal região, evitando a *racing condition*. Obviamente, com a utilização de vários processadores, tal estratégia implica em ociosidade de processos e, consequentemente, diminuição de *speed-up*.

ADELI & KAMAL(1992) apresentam duas soluções para o problema da atualização simultânea da matriz. A primeira solução envolve a divisão da malha em subestruturas associadas aos processos paralelos. Para a sincronização de tais processos, os autores propõem o uso de semáforos.. A segunda solução, além da divisão da malha em subestruturas, utiliza áreas adicionais de memória (uma para cada processo envolvido) para as regiões da matriz acessáveis por mais de um processador. Desta forma os processadores nunca trabalham em áreas comuns da matriz global, tendo para si, áreas exclusivas. No final do algoritmo são somados os termos das áreas adicionais de memória utilizadas.

4.3 ALGORITMO PARALELO PARA MONTAGEM DA MATRIZ DE RIGIDEZ DA ESTRUTURA ORIUNDO DE ALGORITMO SEQUENCIAL ALTERNATIVO

4.3.1 O ALGORITMO SEQUENCIAL ALTERNATIVO

O processo alternativo de montagem da matriz de rigidez da estrutura, proposto por REZENDE(1990) trouxe o conceito de "abordagem nodal" na análise estrutural pelo método dos elementos finitos. Diferentemente do algoritmo usual em que a matriz de rigidez global é montada pela somatória de contribuições dos elementos, nessa abordagem alternativa a etapa de montagem de matriz de rigidez da estrutura se dá pela somatória de contribuições dos pontos nodais. Resumidamente o algoritmo é um *loop* que percorre todos os pontos nodais inserindo na matriz de rigidez global a porção da matriz de rigidez (porção referente ao ponto nodal) de cada elemento ligado ao referido ponto (figura 4.3). Desta forma, a cada iteração do *loop* são formadas as linhas completas da matriz de rigidez global correspondentes ao ponto nodal.

O algoritmo sequencial alternativo de montagem da matriz de rigidez da estrutura

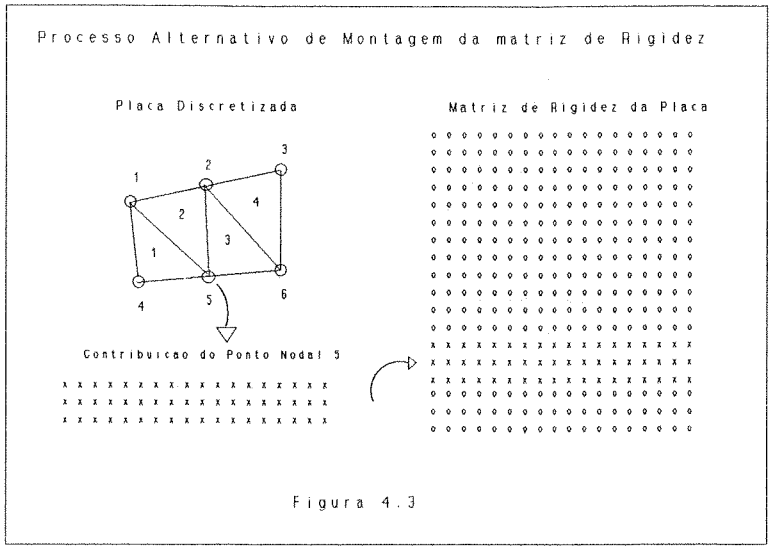


Figura 4.3

requer a existência de uma estrutura de dados auxiliar que para cada ponto contenha as seguintes informações :

✓ Número de elementos ligados ao ponto e

✓ Para cada elemento ligado ao ponto :

- número do elemento
- ponto correspondente (numeração local)

Como exemplo, para o ponto nodal 5 da malha da figura 4.3, a estrutura de dados auxiliar poderia ser a apresentada a seguir.:

PONTO NODAL : 5
NÚMERO DE ELEMENTOS CONCORRENTES : 3

ELEMENTO	PONTO NODAL CORRESPONDENTE
1	2
2	1
3	3

O algoritmo 4.3 mostra em "pseudo código" como a estrutura de dados auxiliar pode ser facilmente montada durante a leitura dos dados dos elementos finitos.

ALGORITMO 4.3 - CRIAÇÃO DA ESTRUTURA DE DADOS AUXILIAR DO ALGORITMO ALTERNATIVO DE MONTAGEM DA MATRIZ DE RIGIDEZ GLOBAL DURANTE A LEITURA DOS DADOS DOS ELEMENTOS FINITOS

NPON=número de pontos nodais do elemento
NP[i,k]=número do k-ésimo ponto nodal do elemento i
NCONC[i]=número de elementos ligados ao ponto i
ELEM_CONC[k,i]=número do i-ésimo elemento ligado ao ponto k
NO_CORRESP[k,i]=numeração local do ponto k para o i-ésimo elemento a ele ligado

```
INÍCIO
PARA i=1 ATÉ Número_de_pontos FAÇA
  NCONC[i]=0
FIM_FAÇA
PARA i=1 ATÉ Número_de_elementos FAÇA
 PARA j=1 ATÉ NPON FAÇA
   LEIA NP[i,j]
   P=NP[i,j]
   NCONC[P]=NCONC[P]+1
   ELEM_CONC[P,NCONC[P]]=i
   NO_CORRESP[P,NCONC[P]]=j
 FIM_FAÇA
FIM_FAÇA
FIM
```

Na implementação do algoritmo 4.3 em linguagens de programação com alocação estática de memória (FORTRAN 77) é necessário estabelecer-se o número máximo de elementos que podem se ligar a determinado ponto nodal, pois as áreas de memória que receberão os *arrays* ELEM_CONC[k,i] e NO_CORRESP[k,i] não podem ser alteradas durante o processamento. Esse fato leva a um aproveitamento ineficiente em termos de ocupação da memória, pois cada ponto terá disponível (mesmo no caso de não vir a utilizá-la) capacidade para armazenar o número máximo de elementos que podem se ligar a um ponto. Nas linguagens com alocação dinâmica de memória (C, Pascal e versões recentes de FORTRAN, por exemplo) a implementação pode tornar-se mais eficiente se para cada elemento ligado a um ponto for alocada (em tempo de processamento) a correspondente área de memória.

ALGORITMO 4.4 - MONTAGEM SEQUENCIAL ALTERNATIVA DA MATRIZ DE RIGIDEZ DA ESTRUTURA

NPAR = Número de parâmetros nodais do elemento
NPON = Número de pontos do elemento
NP[i,k]=número do k-ésimo ponto nodal do elemento i
MATG[i,j]= matriz de rigidez global
MATL(elem)[i,j]= matriz de rigidez do elemento "elem"

```
1  INÍCIO
2  PARA i=1 ATÉ Número_de_pontos_nodais FAÇA
3     PARA j=1 ATÉ NCONC[i]  FAÇA
4       ELEM = ELEM_CONC[i,j]
5       PON = NO_CORRESP[i,j]
6       LINHA_BASE =  NPAR*i - (NPAR - 1)
7       PARA k=1 ATÉ NPON FAÇA
8        PAUX= NP[ELEM,k]
9        COLUNA = NPAR*PAUX - (NPAR-1)
10       PARA m=1 ATÉ NPAR FAÇA
11         LINHA_LOCAL = NPAR*(PON-1) + m
12         LINHA_GLOBAL = ( LINHA_BASE - 1 ) + m
13         PARA n=1 ATÉ NPAR FAÇA
14            COLUNA_GLOBAL = NPAR*(k-1) + n
15            COLUNA_GLOBAL = (COLUNA -1 ) + m
16           DIF = COLUNA_GLOBAL-LINHA_GLOBAL+1
17            SE DIF > 0 ENTÃO
18            MATG[LINHA_GLOBAL,DIF] = MATG[LINHA_GLOBAL,DIF]
                +   MATL(elem)[LINHA_LOCAL,COLUNA_LOCAL]
19            FIM_SE
20         FIM_FAÇA
21       FIM_FAÇA
22      FIM_FAÇA
23    FIM_FAÇA
24 FIM_FAÇA
25 FIM
```

O algoritmo 4.4 traz em "pseudo código", a montagem alternativa da matriz de rigidez da estrutura (arranjo em banda), na hipótese da estrutura auxiliar já se apresentar previamente definida e ainda para o caso das matrizes de rigidez dos elementos estarem disponíveis na memória.

4.3.2 PARALELIZAÇÃO DO ALGORITMO ALTERNATIVO

Conforme foi apresentado em sua descrição, cada iteração do loop mais externo do algoritmo alternativo gera as linhas da matriz de rigidez global correspondentes ao ponto nodal da iteração. Assim, diferentes iterações trabalham em linhas diferentes da matriz de rigidez global, não havendo portanto possibilidade de ocorrência da atualização simultânea da matriz de rigidez global. Em outras palavras, o loop mais externo não apresenta dependência de dados e consequentemente pode ser paralelizado sem qualquer tipo de sincronização. Em termos de algoritmo, a paralelização é realizada acrescentando-se a expressão "em paralelo" ao loop mais externo (algoritmo 4.5).

ALGORITMO 4.5 - PARALELIZAÇÃO DO ALGORITMO ALTERNATIVO DE MONTAGEM DA MATRIZ DE RIGIDEZ GLOBAL

INÍCIO
PARA i=1 ATÉ Número_de_pontos_nodais **FAÇA_EM_PARALELO**

Linha 3 até linha 23do algoritmo 4.4

FIM_FAÇA_EM_PARALELO
FIM

4.3.3 ABORDAGEM NODAL PLENA NA MONTAGEM DA MATRIZ DE RIGIDEZ

Na análise linear via método dos elementos finitos é comum a utilização de malhas regulares, ou seja, malhas em que vários elementos apresentam a mesma matriz de rigidez (elementos típicos), a qual é montada apenas uma vez e armazenada para a posterior utilização por parte do algoritmo de montagem da matriz de rigidez global. Esse procedimento, obviamente não se aplica às análises lineares com malhas irregulares e também à análise não linear em geral via métodos incrementais iterativos, onde ao longo do processo de carga as variações da geometria e de esforços nos elementos alteram as matrizes de rigidez dos mesmos.

Particularmente no caso da montagem paralela da matriz de rigidez global com abordagem nodal , a matriz de rigidez de um elemento é requerida tantas vezes qual seja

o número de pontos nodais aos quais o elemento é ligado, já que para cada um deles o elemento cederá uma parcela de sua matriz. Desta forma, para prover a disponibilidade da matriz do elemento quando a mesma é solicitada aparecem, a princípio, duas soluções: a primeira seria a montagem da matriz do elemento repetidamente, a cada solicitação (solução econômica em termos de utilização de memória e dispendiosa em termos de tempo de processamento); a segunda solução seria o armazenamento da matriz de rigidez de cada elemento após a sua primeira utilização (solução dispendiosa em termos de utilização de memória e econômica em termos de tempo de processamento). Como ambas as soluções têm desempenho computacional pobre em algum aspecto, desenvolveu-se uma solução alternativa que combina economia de uso de área de memória e rapidez de processamento : a montagem da matriz de rigidez global com abordagem nodal plena.

Na abordagem nodal plena não há algoritmos para a montagem da matriz de rigidez de elementos, e sim algoritmos para a montagem de sub-matrizes de rigidez dos elementos que correspondem à contribuição do mesmo à cada um de seus pontos nodais. Desta forma, se o elemento finito possui quatro pontos nodais, deverá possuir quatro algoritmos, cada um produzindo as linhas da matriz de rigidez correspondentes a cada um dos pontos nodais. Quando, na abordagem nodal plena, estão sendo montadas as equações globais referentes a um ponto nodal, para cada elemento a ele ligado será montada apenas a sub-matriz de rigidez correspondente ao referido ponto.

Na figura 4.4 está representada a montagem das equações globais referentes a um ponto nodal (5) de uma treliça plana. O elemento de treliça possui dois pontos nodais e assim, na abordagem nodal plena, requer dois algoritmos de montagem das sub-matrizes de rigidez correspondentes aos dois pontos nodais. Deve ser observado na figura 4.4 que conforme o elemento está ligado ao ponto nodal 5 através de seu ponto nodal "i" ou "j", ele fornecerá as respectivas equações ao ponto nodal. O algoritmo 4.6 traz a montagem paralela alternativa da matriz de rigidez global com abordagem nodal plena . Esse algoritmo requer a prévia montagem da estrutura de dados auxiliar da abordagem nodal (algoritmo 4.3).

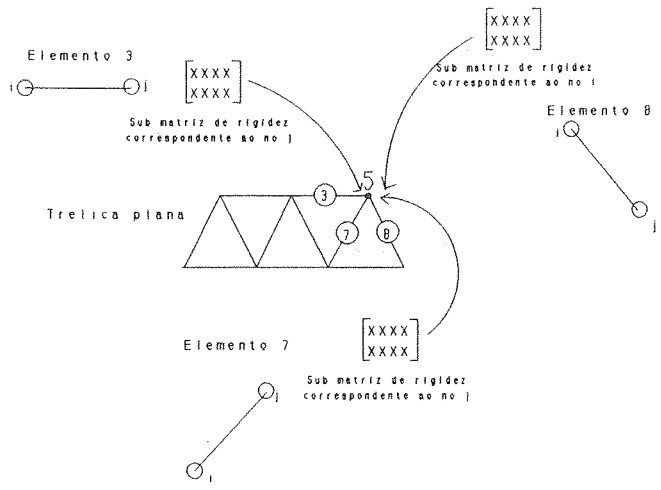


Figura 4.4 Abordagem nodal plena

ALGORITMO 4.6 - MONTAGEM PARALELA ALTERNATIVA DA MATRIZ DE RIGIDEZ GLOBAL COM ABORDAGEM NODAL PLENA

NPAR = Número de parâmetros nodais do elemento

NPON = Número de pontos do elemento

NP[i,k]=número do k-ésimo ponto nodal do elemento i

NCONC[i]=número de elementos ligados ao ponto i

ELEM_CONC[k,i]=número do i-ésimo elemento ligado ao ponto k

NO_CORRESP[k,i]=numeração local do ponto k para o i-ésimo elemento a ele ligado

MATG[i,j]= matriz de rigidez global

MATL[p,i,j]= sub-matriz de rigidez local que contém as "NPAR" linhas correspondentes ao ponto nodal "p" (numeração local).

(algoritmo 4.6 - continuação)

```
INÍCIO
PARA i=1 ATÉ Número_de_pontos_nodais FAÇA_EM_PARALELO
  PARA j=1 ATÉ NCONC[i]  FAÇA
    ELEM = ELEM_CONC[i,j]
    PON = NO_CORRESP[i,j]
    MONTE_A_SUBMATRIZ_DO ELEMENTO_ELEM_PONTO_PON
    LINHA_BASE =  NPAR*i - (NPAR - 1)
    PARA k=1 ATÉ NPON FAÇA
      PAUX= NP[ELEM,k]
      COLUNA = NPAR*PAUX - (NPAR-1)
      PARA m=1 ATÉ NPAR FAÇA
         LINHA_LOCAL =  m
         LINHA_GLOBAL = ( LINHA_BASE - 1 ) + m
         PARA n=1 ATÉ NPAR FAÇA
           COLUNA_LOCAL = NPAR*(k-1) + n
           COLUNA_GLOBAL = (COLUNA -1 ) + m
           DIF = COLUNA_GLOBAL-LINHA_GLOBAL+1
           SE DIF>0 ENTÃO
           MATG[LINHA_GLOBAL,DIF] = MATG[LINHA_GLOBAL,DIF]  +
              MATL[PON,LINHA_LOCAL,COLUNA_LOCAL]
           FIM_SE
         FIM_FAÇA
      FIM_FAÇA
    FIM_FAÇA
  FIM_FAÇA
FIM_FAÇA_EM_PARALELO
FIM
```

4.3.4 IMPOSIÇÃO DE CONDIÇÕES DE CONTORNO ALIADA À MONTAGEM PARALELA DA MATRIZ DE RIGIDEZ GLOBAL DA ESTRUTURA

A criação de processos paralelos sempre implica em esforço computacional adicional. Desta forma, é sempre interessante que os processos criados envolvam montante de trabalho suficiente para tornar os benefícios da paralelização maiores do que o esforço computacional inerente à criação dos processos concorrentes. Particularmente na utilização do método dos elementos finitos em análise estrutural, há etapas de relevante esforço computacional como a montagem da matriz de rigidez global e etapas menos expressivas em termos de operações, como é o caso da etapa de imposição de condições de contorno. Se a paralelização de etapas de pequeno esforço computacional pode ser prejudicial, a inclusão das mesmas em etapas

paralelizadas mais significantes é procedimento benéfico ao desempenho desde que tal inclusão não implique em degradação do balanceamento do trabalho do algoritmo original de maior esforço computacional.

Em REZENDE(1990) é proposta a montagem sequencial da matriz de rigidez global aliada à imposição de condições de contorno. No processo proposto, a imposição de parâmetros nodais nulos é feita eliminando-se a linha e a coluna referentes ao parâmetro nodal. Além disso, é também eliminada a linha correspondente do vetor de cargas nodais. Como exemplo, considere-se a matriz de dimensões 8x8 da figura 4.5. Dessa matriz serão eliminadas a linha e a coluna 6, alterando as dimensões da matriz para 7x7. Deve ser observado que a eliminação da linha e da coluna altera os índices de alguns elementos. Os índices que os elementos terão após a eliminação da linha e da coluna são as chamadas "posições efetivas". Sabendo-se, *a priori* as posições efetivas dos elementos, pode-se montar diretamente a matriz de rigidez global já com as linhas e colunas eliminadas. No processo proposto por REZENDE(1990), antes da etapa de montagem da matriz de rigidez global, há um pequeno algoritmo (4.7) que gera um *array* com as posições efetivas dos índices da matriz (POSEFET(i)) . Para a execução do algoritmo 4.7 é necessário um *array* (PARAM(i)) que indique se determinado parâmetro nodal é nulo. Por exemplo, se o parâmetro nodal número 10 for nulo então PARAM(10) será 1, caso contrário será nulo.

Após a execução do algoritmo 4.7, para a inserção da imposição de condições de contorno no algoritmo paralelo alternativo de montagem da matriz de rigidez global (com abordagem nodal ou nodal plena), basta impor que os índices dos elementos sejam os oriundos das posições efetivas dos mesmos. O algoritmo 4.8 traz a montagem paralela da matriz de rigidez global com abordagem nodal plena, já com a imposição de condições de contorno. A transformação do algoritmo com abordagem nodal é totalmente análoga.

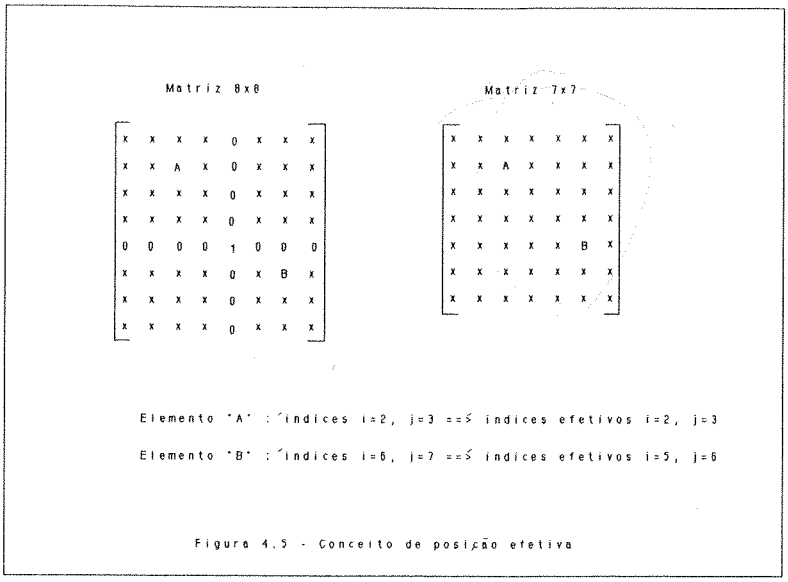


Figura 4.5 - Conceito de posição efetiva

ALGORITMO 4.7 - CÁLCULO DE POSIÇÕES EFETIVAS DE ÍNDICES

NPAR = número de parâmetros nodais da estrutura

INÍCIO
desconto = 0
PARA i=1 até NPAR **FAÇA**
  SE PARAM(i) = 0 **ENTÃO**
    POSEFET(i) = 0
    desconto = desconto +1
  CASO_CONTRÁRIO
    POSEFET(i) = i - desconto
  FIM_SE
FIM|_FAÇA
FIM

ALGORITMO 4.8 - MONTAGEM PARALELA DA MATRIZ DE RIGIDEZ GLOBAL COM ABORDAGEM NODAL PLENA E IMPOSIÇÃO DE CONDIÇÕES DE CONTORNO

OBS : Previamente devem ser utilizados os algoritmos 4.3 e 4.7
NPAR = Número de parâmetros nodais do elemento
NPON = Número de pontos do elemento
NP[i,k]=número do k-ésimo ponto nodal do elemento i

(continuação algoritmo 4.8)

NCONC[i]=número de elementos ligados ao ponto i
ELEM_CONC[k,i]=número do i-ésimo elemento ligado ao ponto k
NO_CORRESP[k,i]=numeração local do ponto k para o i-ésimo elemento a ele ligado
MATG[i,j]= matriz de rigidez global
MATL[p,i,j]= sub-matriz de rigidez local que contém as "NPAR" linhas correspondentes ao ponto nodal "p" (numeração local).
POSEFET[i]=Posição efetiva do índice "i" após a imposição das condições de contorno

```
INÍCIO
PARA i=1 ATÉ Número_de_pontos_nodais FAÇA_EM_PARALELO
  PARA j=1 ATÉ NCONC[i]  FAÇA
    ELEM = ELEM_CONC[i,j]
    PON = NO_CORRESP[i,j]
    MONTE_A_SUBMATRIZ_DO ELEMENTO_ELEM_PONTO_PON
    LINHA_BASE =  NPAR*i - (NPAR - 1)
    PARA k=1 ATÉ NPON FAÇA
      PAUX= NP[ELEM,k]
      COLUNA = NPAR*PAUX - (NPAR-1)
      PARA m=1 ATÉ NPAR FAÇA
        LINHA_LOCAL =  m
        LINHA_GLOBAL = ( LINHA_BASE - 1 ) + m
        PARA n=1 ATÉ NPAR FAÇA
          COLUNA_LOCAL = NPAR*(k-1) + n
          COLUNA_GLOBAL = (COLUNA -1 ) + m
/*  Se a posição efetiva da coluna ou da linha é zero o elemento não é inserido, caso contrário é
inserido na posição efetiva correspondente */
          SE POSEFET(LINHA_GLOBAL) ≠ 0 E POSEFET(COLUNA_GLOBAL) ≠ 0 ENTÃO
            LINHA_GLOBAL=POSEFET(LINHA_GLOBAL)
            COLUNA_GLOBAL=POSEFET(COLUNA_GLOBAL)
            DIF = COLUNA_GLOBAL-LINHA_GLOBAL+1
            SE DIF > 0 ENTÃO
              MATG[LINHA_GLOBAL,DIF] = MATG[LINHA_GLOBAL,DIF]   +
                        MATL[PON,LINHA_LOCAL,COLUNA_LOCAL]
            FIM_SE
          FIM_SE
        FIM_FAÇA
      FIM_FAÇA
    FIM_FAÇA
  FIM_FAÇA
FIM_FAÇA_EM_PARALELO
FIM
```

4.3.5 MEDIDAS DE DESEMPENHO DOS ALGORITMOS PROPOSTOS

O desempenho (*speed-up*) dos algoritmos paralelos alternativos de montagem da mtariz de rigidez global com abordagem nodal e com abordagem nodal plena foi avaliado em implementações efetuadas em computador Silicon Graphics modelo 4S-440D com arquitetura paralela[1] . Os exemplos tratados procuram mostrar a sensível influência do número de elementos da malha e da complexidade da montagem da matriz de rigidez dos mesmos no *speed-up*.

As medidas de *speed-up* partiram de um programa de análise linear de treliças tridimensionais. Foram analisadas três treliças tridimensionais : treliça "G" (3258 barras, figura 4.6), treliça "M" (2048 barras, figura 4.7) e treliça "P" (160 barras, figura 4.8). são características comuns às três avaliações efetuadas :

✓ Os algoritmos utilizados sempre efetuavam conjuntamente a imposição de condições de contorno (o tempo de montagem do vetor de posições efetivas (algoritmo 4.7) foi desprezado). A inclusão ou não dessa etapa na montagem da matriz de rigidez traz variações desprezíveis de tempo de processamento .

✓ O esforço computacional da montagem da estrutura de dados auxiliar (algoritmo 4.3) não foi incluído nas medidas de *speed-up*, pois não foi realizada a paralelização da mesma (essa parcela sequencial comprometeria as medidas de speed-up com vários processadores). Em medições feitas em processamento sequencial, o tempo de montagem da estrutura auxiliar representou menos de 2,5% do tempo total de montagem da matriz de rigidez para o caso de análise linear de treliças tridimensionais com abordagem nodal plena, devendo essa parcela ser bem menor para o caso de matrizes de rigidez de elementos mais complexos.

✓ Para simular a montagem de matrizes de rigidez de elementos mais complexos foram inseridas operações de ponto flutuante nas rotinas de montagem de matriz de

[1]Detalhes da linguagem de programação e do computador utilizados neste trabalho, além do processo para determinação do *speed-up*, estão no ANEXO .

rigidez dos elementos com abordagem nodal plena. Assim, a referência ao exemplo "G1000" significa que foram inseridas mil operações do tipo ponto flutuante nas rotinas de cada ponto nodal do elemento (no caso, a barra de treliça possui dois pontos nodais).

✓ Foram também feitas medições para o caso de abordagem nodal (não plena), a qual é útil na montagem de matrizes de rigidez de estruturas discretizadas com malhas regulares em análise linear. Nessas medições não foram incluídos os tempos de montagem das matrizes de rigidez dos elementos típicos, as quais já estavam disponíveis na memória. Assim, referência ao exemplo "GN", significa exemplo "G" com abordagem nodal.

A tabela 4.1 traz as duplas "tempo obtido (segundos) / *Speed-up*" para cada um dos exemplos. Como no computador utilizado a montagem da matriz global é extremamente rápida, as medições foram feitas em repetições (*loops*) da etapa, ou seja, o tempos obtidos são a divisão dos tempos totais pelo número de repetições. Em seguida, apresentam-se as figuras relativas aos exemplos e os gráficos de *Speed-up* e eficiência.

TABELA 4.1	NÚMERO DE PROCESSADORES				
EXEMPLO	1	2	3	4	Repetições
G	0,354 / 1,00	0,188 / 1,88	0,126 / 2,81	0,102 / 3,47	500
G1000	2,84 / 1.00	1,44 / 1,97	0,98 / 2,90	0,75 / 3,79	100
GN	0,282 / 1,00	0,158 / 1,78	0,108 / 2,61	0,086 / 3,28	500
M	0,200 / 1,00	0,108 / 1,85	0,074 / 2,70	0,058 / 3,45	500
M1000	1,63 / 1,00	0,84 / 1,94	0,57 / 2,86	0,43 / 3,79	100
MN	0,160 / 1,00	0,090 / 1,78	0,064 / 2,50	0,052 / 3,08	500
P	0,0146 / 1,00	0,0078 / 1,87	0,0056 / 2,60	0,0046 / 3,17	5000
P1000	0,127 / 1,00	0,065 / 1,95	0,048 / 2,65	0,040 / 3,18	1000

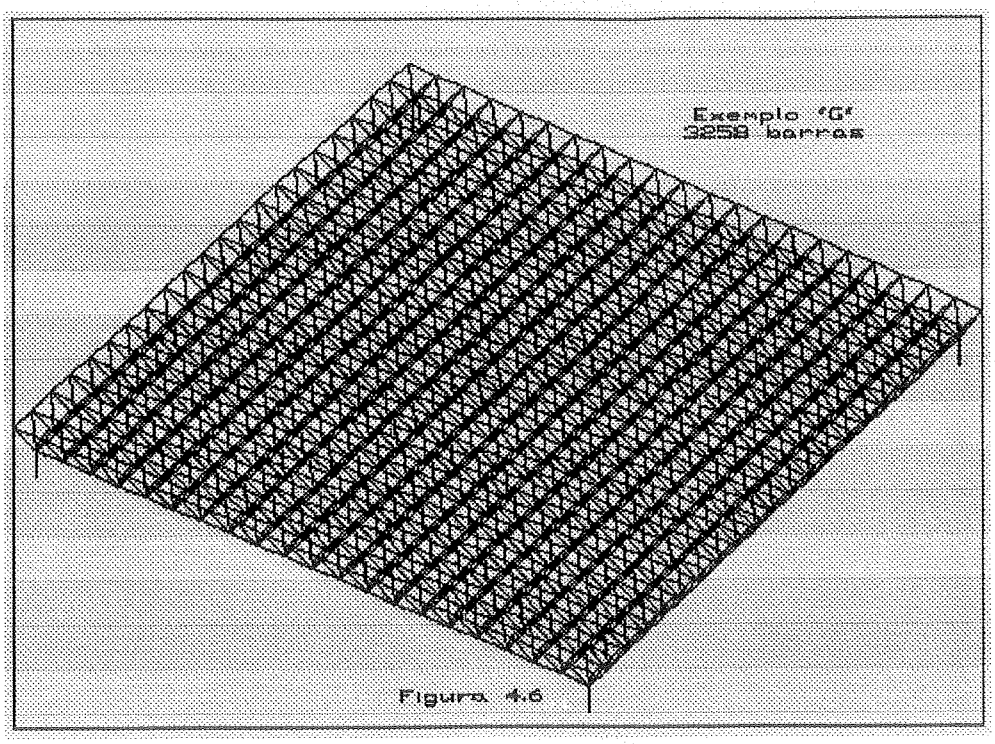
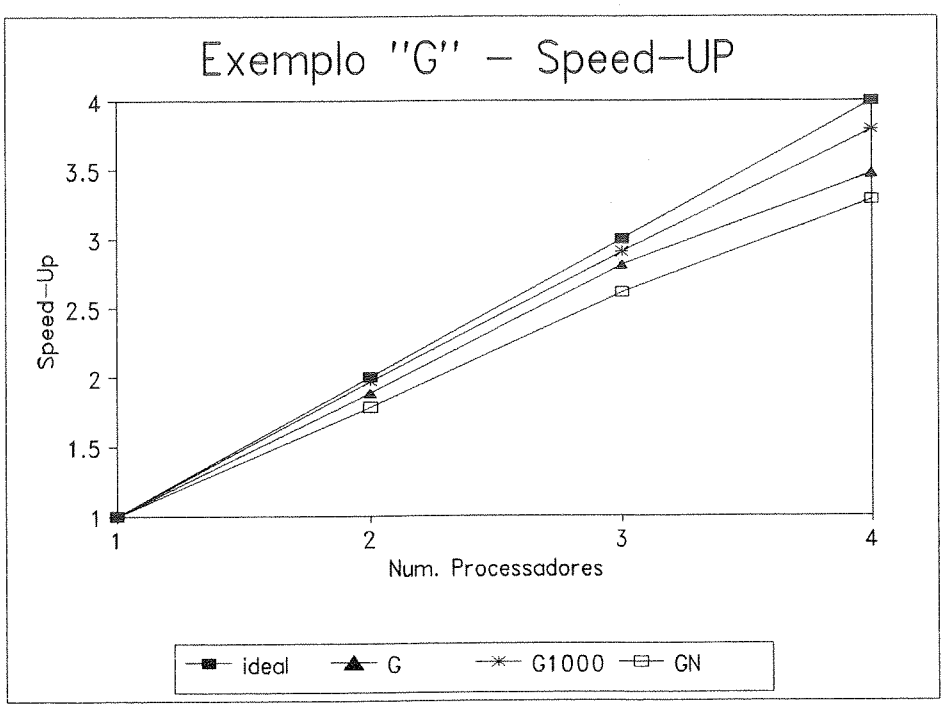
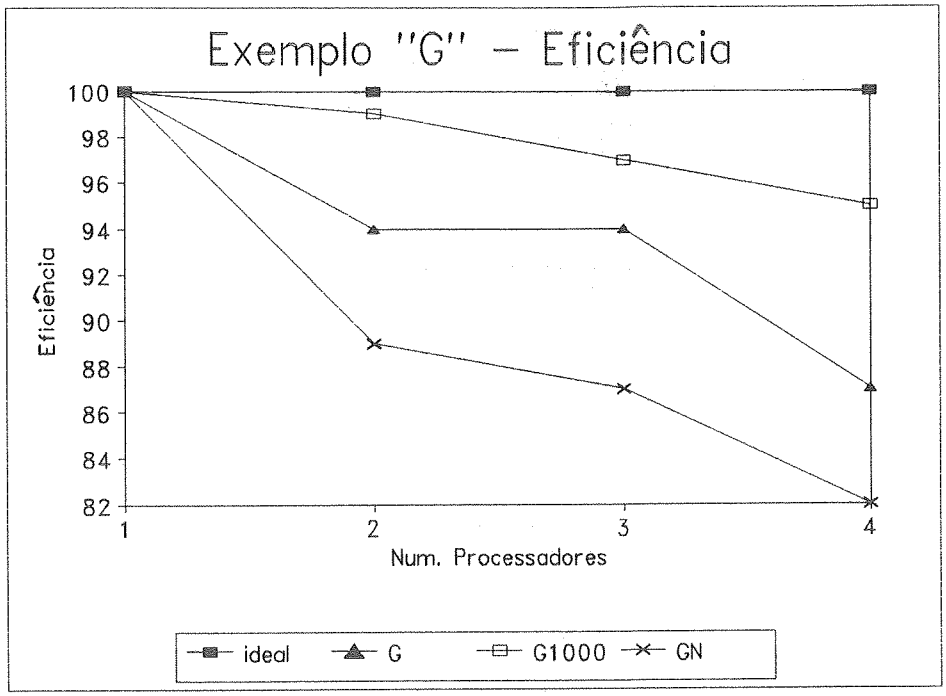
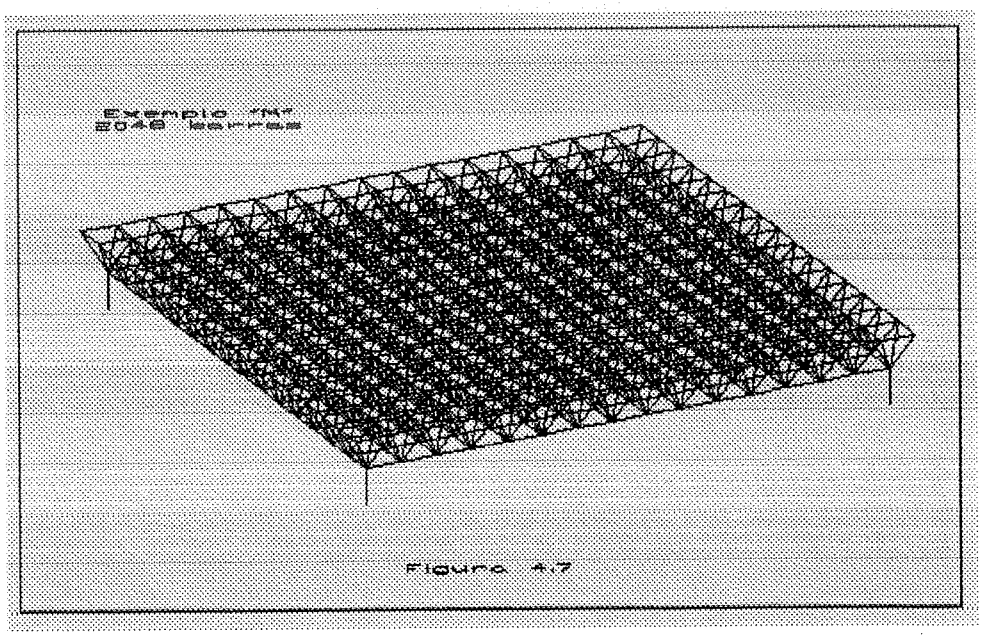
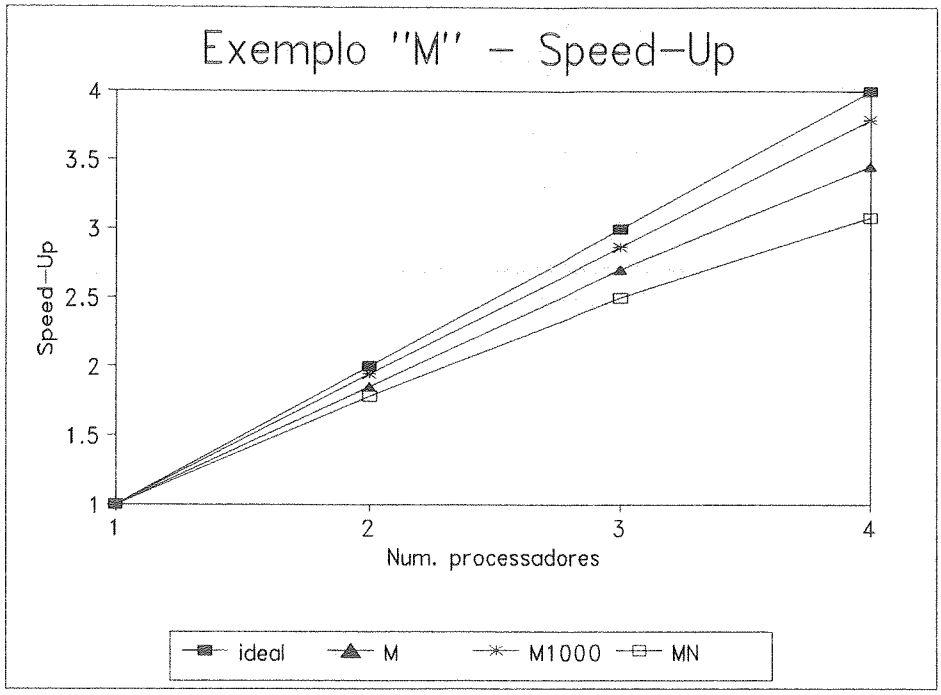

Exemplo "G"
3258 barras

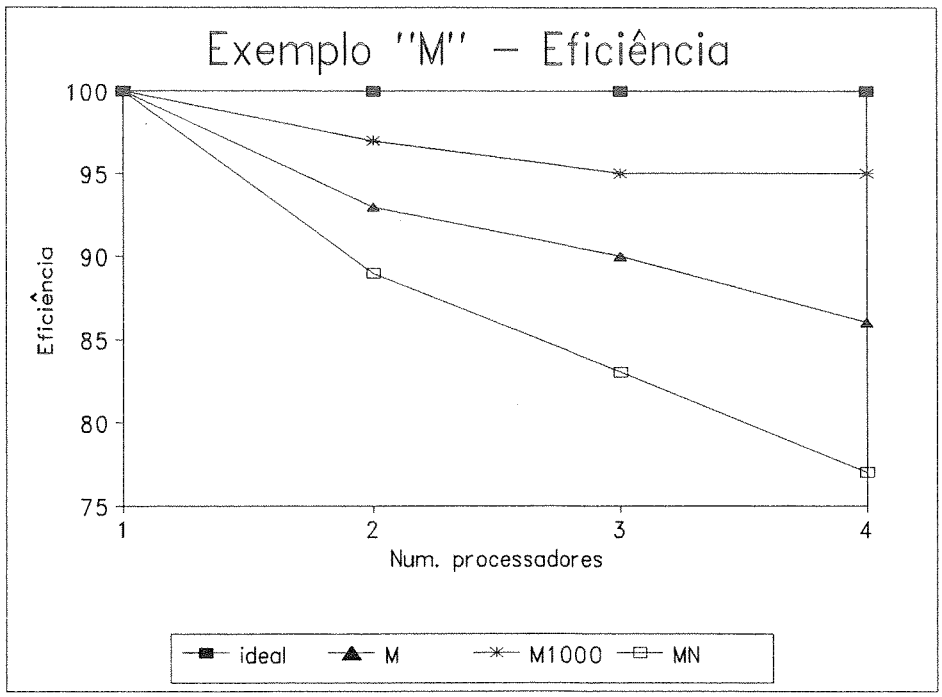
Figura 4.6

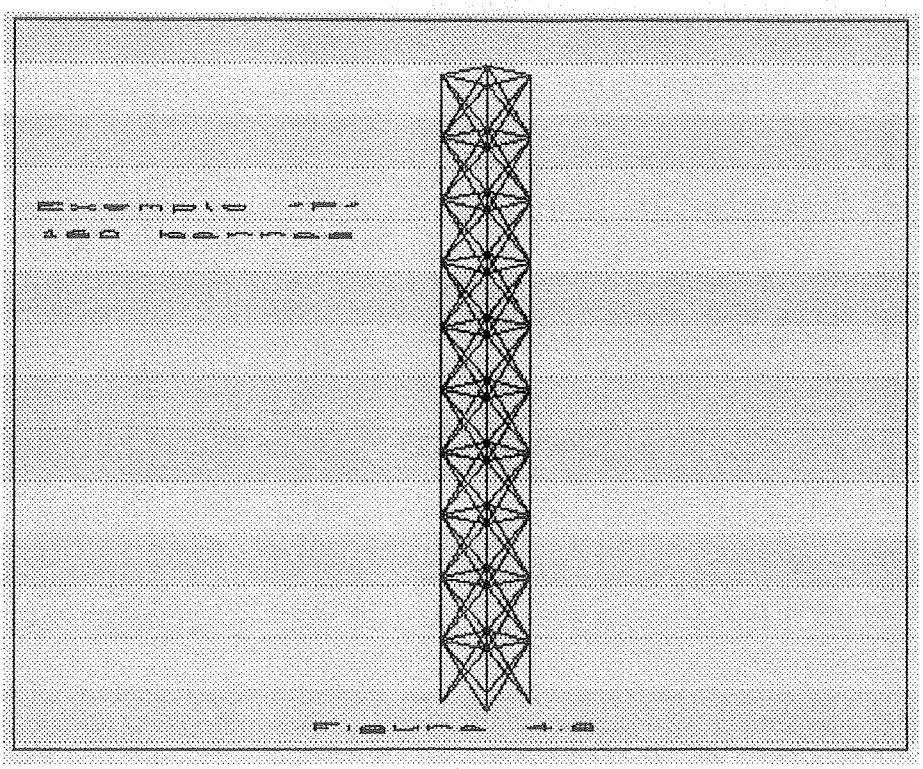

Exemplo "G" – Speed-UP
Speed-Up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. Processadores
ideal
G
G1000
GN

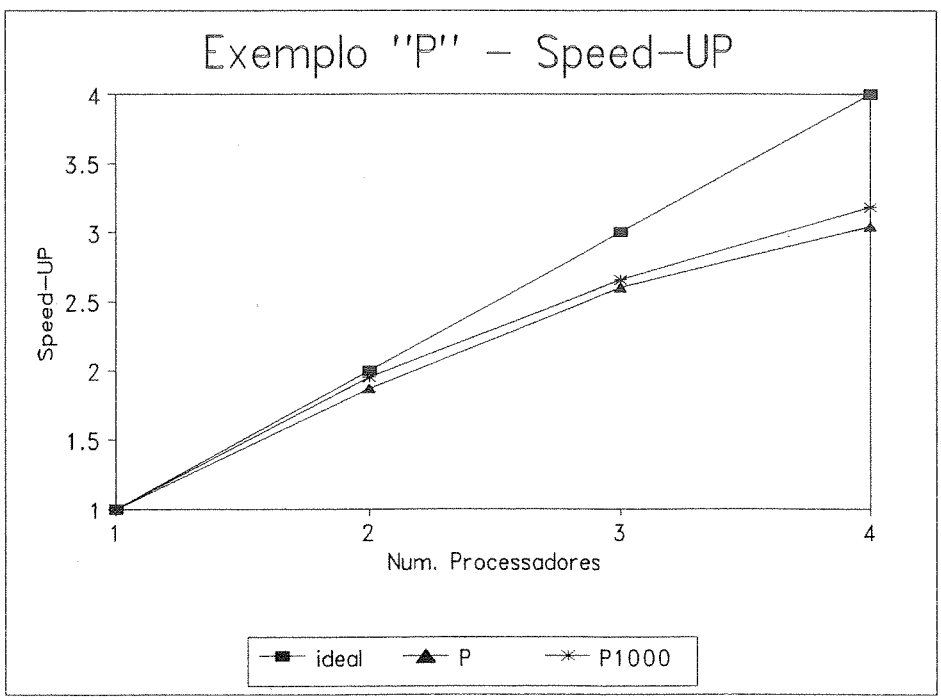

Exemplo "G" – Eficiência
100
98
96
94
92
90
88
86
84
82
Eficiência
1
2
3
4
Num. Processadores
ideal
G
G1000
GN

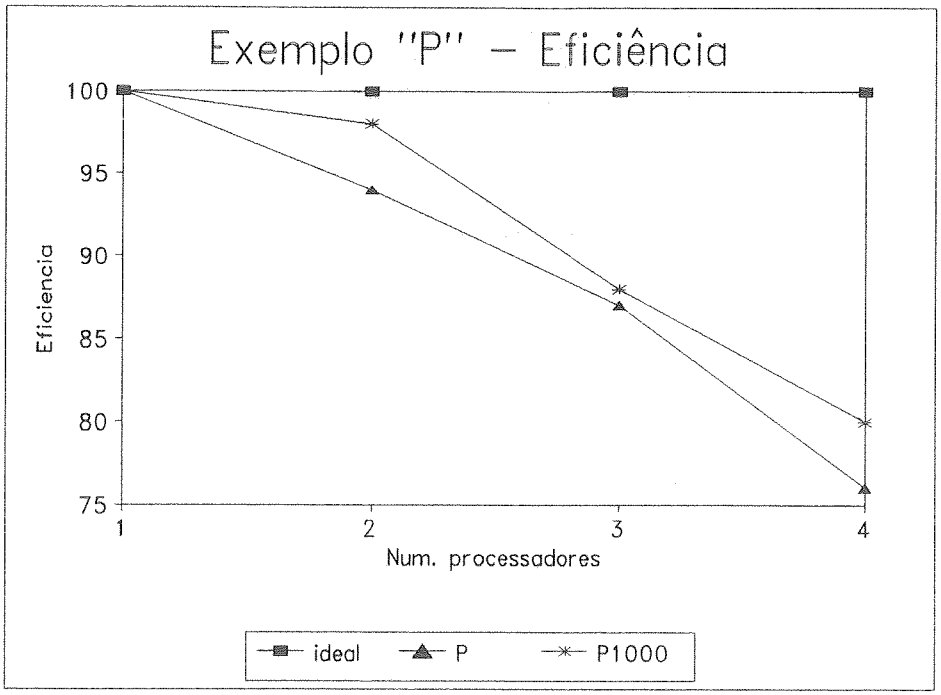



Exemplo "M" – Speed-Up
Speed-Up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
M
M1000
MN


Exemplo "M" – Eficiência
Eficiência
100
95
90
85
80
75
1
2
3
4
Num. processadores
ideal
M
M1000
MN




Exemplo "P" – Speed-UP
Speed-UP
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. Processadores
ideal
P
P1000



4.3.6 INTERPRETAÇÃO DOS RESULTADOS OBTIDOS

4.3.6.1 FATORES DE ALTERAÇÃO DE EFICIÊNCIA

Com os resultados obtidos, nota-se que a eficiência dos algoritmos propostos aumenta com a elevação do número de elementos na malha, ou mesmo com o incremento de complexidade da geração das matrizes de rigidez dos elementos. Este fato pode ser explicado da seguinte forma : tanto o aumento do número de elementos da malha, quanto o incremento de complexidade da matriz de rigidez dos mesmos, aumentam o total de trabalho que será executado concorrentemente; como há um esforço inicial de paralelização (aproximadamente constante), esse esforço tornar-se-á cada vez menos relevante se comparado com o total de trabalho realizado paralelamente, implicando em aumento de eficiência.

A não necessidade de sincronização dos algoritmos propostos, leva a supor

que diminuições de eficiência poderiam ter como origem (além de fatores inerentes à máquina, como é o caso do esforço computacional inicial de paralelização) apenas em um não balanceamento de trabalho na execução paralela do algoritmo. Como na paralelização os pontos nodais são divididos entre os processadores para a montagem da matriz global, um não balanceamento seria verificado apenas no processamento de malhas em que alguns poucos pontos nodais recebessem um número muito elevado de elementos se comparado aos outros pontos da malha e ainda esses pontos nodais fossem vinculados a apenas apenas um processador, concentrando trabalho no mesmo, fato incomum nas malhas usualmente utilizadas em analise estrutural.

4.3.6.2 COMPARAÇÃO COM RESULTADOS CONHECIDOS

Embora fosse extremamente profícua uma comparação direta entre os resultados obtidos com os algoritmos propostos neste trabalho e os apresentados por ADELI & KAMAL(1992) e CHIEN & SUN(1989), tal comparação é infrutífera entre outros pelos seguintes motivos :

✓ os resultados foram obtidos em máquinas diferentes, implicando em diferentes tempos de criação de processos paralelos, diferentes tempos de acesso à memória global, diferentes esquemas de sincronização e etc;

✓ nos algoritmos que exigem um processamento prévio (divisão da malha em subestruturas), os autores citados não incluem o tempo de tal processamento no tempo de montagem total da matriz de rigidez, nem o estimam. Além disso não se sabe como tais algoritmos prévios se comportam em malhas genéricas;

✓ nas análises lineares efetuadas por ADELI & KAMAL(1992), não fica claro se a matriz de rigidez de cada elemento é montada, ou se pela regularidade das malhas utilizaram-se "elementos típicos".

As tabelas 4.2 e 4.3 trazem resultados (*speed-up* para quatro processadores) de dois exemplos retirados respectivamente de CHIEN & SUN(1989) e ADELI & KAMAL(1992).

TABELA 4.2	CHIEN & SUN(1989)

Tipo de análise	Número de elementos	Sincronização	*Speed-up* (4 proc.)	Eficiência
NÃO LINEAR	200 (treliça plana)	SIM	3,25	81,3%
NÃO LINEAR	200 (treliça plana)	NÃO	3,59	89,8%

TABELA 4.3	ADELI & KAMAL (1992)

Tipo de análise	Número de elementos	Sincronização	*Speed-up* (4 proc.)	Eficiência
LINEAR	266 (pórtico plano)	SIM	3,0	75%
LINEAR	266 (pórtico plano)	NÃO	2,8	70%

As seguintes análises dos resultados dessas tabelas podem ser feitas :

✓O valor de *speed-up* da tabela 4.2 para algoritmo não sincronizado (3,59) embora represente ótima eficiência (89,8%) não deve ser considerado como parâmetro de comparação pois é oriundo de estratégia de numeração de generalidade duvidosa, conforme já observado por ADELI & KAMAL (1992).

✓Apenas como referência, uma comparação grosseira poderia ser feita entre o exemplo não sincronizado (tab. 4.3) e o exemplo P deste trabalho, já que

os problemas são de dimensões tais que os esforços computacionais são comparáveis, embora haja todo um conjunto, já discutido, de fatores que desabonem tal comparação. Nesse tipo de comparação, nota-se que o exemplo "P", para quatro processadores, apresenta *speed-up* 3,17, o qual é superior ao exemplo não sincronizado da tabela 4.3 (2,8).

✓ É interessante notar que segundo ADELI & KAMAL(1992) embora os resultados da tabela 4.3 de *speed-up* para algoritmo sincronizado seja superior ao de algoritmo não sincronizado, essa vantagem é muito pequena e pode se reverter para o caso em que haja várias possibilidades de *racing condition* e a consequente necessidade de sincronização crescente.

Embora a comparação direta de *speed-up* entre os algoritmos propostos neste trabalho e os apresentados por ADELI & KAMAL(1992) e CHIEN & SUN(1989) seja impraticável, pode-se, conceitualmente verificar a superioridade dos algoritmos deste trabalho em relação aos algoritmos sincronizados e não sincronizados propostos pelos autores anteriormente citados. A superioridade em relação aos algoritmos sincronizados se dá pela queda de desempenho que estes têm quando da presença de muitos processadores, ou quando a partição da malha entre os processadores leva à ocorrência de muitas regiões sujeitas à *racing condition*. A superioridade em relação aos algoritmos não sincronizados é oriunda da simplicidade do processamento prévio requerido pelos algoritmos deste trabalho, o qual é de pequena monta e de aplicabilidade genérica, duas características não encontradas simultaneamente nos algoritmos propostos por ADELI & KAMAL (1992) e CHIEN & SUN(1989).

5- PARALELIZAÇÃO DE ETAPAS TÍPICAS DA APLICAÇÃO DO MÉTODO DOS ELEMENTOS FINITOS EM ANÁLISE ESTRUTURAL: RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINEARES

5.1 INTRODUÇÃO :

A aplicação do método dos elementos finitos na análise linear de estruturas implica sempre em uma etapa de resolução de sistema de equações lineares. Além disso, mesmo no tratamento de problemas estruturais não lineares, a referida etapa ocupa grande destaque, já que nos dias atuais é expressiva a tendência de utilização de processos incrementais iterativos, os quais tratam o problema não linear como uma sequência de aproximações lineares.

Seja na análise linear ou na não linear via método dos elementos finitos, a etapa de resolução de sistema de equações lineares é a que geralmente requer maior esforço computacional, normalmente pelo fato de estarem envolvidas operações em matrizes de grandes dimensões. A importância de tal etapa justifica, portanto, a existência dos numerosos trabalhos científicos nessa área abrangendo algoritmos sequenciais, e mais recentemente, algoritmos paralelos.

Particularmente no caso dos algoritmos paralelos, conforme CODENOTTI(1991) tem sido três as principais linhas de atuação de pesquisa :

1- Desenvolvimento de algoritmos paralelos partindo da adaptação de algoritmos sequenciais de desempenho consagrado.

2- Desenvolvimento de algoritmos paralelos partindo da adaptação de algoritmos sequenciais de menor desempenho, os quais, não necessariamente apresentam desempenho inferior nas versões adaptadas ao processamento paralelo.

3- Desenvolvimento de algoritmos novos, especialmente voltados para as arquiteturas de processamento paralelo.

Neste capítulo, seguindo a primeira das tendências citadas, apresenta-se a forma usual de paralelização (para arquiteturas MIMD de memória global) do consagrado algoritmo "método de eliminação de Gauss". É também proposto um algoritmo alternativo paralelo da etapa de triangularização desse método. Foi realizada a implementação paralela usual e alternativa do método para o caso de matriz simétrica com arranjo em banda em computador Silicon Graphics modelo 4S-440D com arquitetura paralela[1]. Finalmente apresentam-se vários resultados de desempenho dos algoritmos (*speed-up*).

5.2 O MÉTODO DE ELIMINAÇÃO DE GAUSS

Seja o problema da solução do seguinte sistema de equações lineares

$$\mathbf{A}x=\mathbf{b} \qquad 5.1$$

onde **A** é uma matriz quadrada (matriz dos coeficientes de dimensões nxn), x é o vetor incógnito de dimensão n a ser determinado e finalmente **b** é o vetor de termos independentes (dimensão n). No Método de Eliminação de Gauss a idéia básica é conduzir inicialmente o sistema de equacões 5.1 à forma

$$\mathbf{U}x=\mathbf{b}' \qquad 5.2$$

onde **U** é uma matriz triangular superior, sendo essa transformação inicial denominada triangularização. A segunda etapa do método consiste na solução do sistema triangularizado e é denominada retrosubstituição. As etapas de triangularização e retrosubstuição são efetuadas a partir de apenas uma operação elementar sobre o sistema : somar a uma equação um múltiplo de outra . Os algoritmos comentados sobre as duas etapas são apresentados a seguir .

[1]No ANEXO estão as características do computador utilizado e da linguagem de programação, além do processo para obtenção do *Speed-up*.

5.2.1 ETAPA DE TRIANGULARIZAÇÃO

Seja o sistema de equações lineares 5.1 (apresentado agora em sua forma explícita), o qual se pretende levar à forma triangularizada

$$
\begin{array}{llllll}
a_{11}x_1 & +a_{12}x_2 & +a_{13}x_3 & +\ldots\ldots & +a_{1n}x_n & =b_1 \\
a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & +\ldots\ldots & +a_{2n}x_n & =b_2 \\
a_{31}x_1 & +a_{32}x_2 & +a_{33}x_3 & +\ldots\ldots & +a_{3n}x_n & =b_3 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
a_{n1x_1} & +a_{n2x_2} & +a_{n3x_3} & +\ldots\ldots & +a_{nn}x_n & =b_n
\end{array}
\qquad \ldots\ldots (5.1)
$$

Sendo a_{11} diferente de zero pode-se eliminar x_1 da segunda equação somando-se à mesma a primeira equação multiplicada por

$-a_{21}/a_{11}$. Analogamente pode-se eliminar x_1 das outras n-2 equações restantes, resultando nas equações 5.2.

$$
\begin{array}{llllll}
a_{11}x_1 & +a_{12},x_2 & +a_{13}x_3 & +\ldots\ldots & +a_{1n}x_n & =b_1 \\
0 & +a'_{22}x_2 & +a'_{23}x_3 & +\ldots\ldots & +a'_{2n}x_n & =b'_2 \\
0 & +a'_{32}x_2 & +a'_{33}x_3 & +\ldots\ldots & +a'_{3n}x_n & =b'_3 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
0 & +a'_{n2}x_2 & +a'_{n3}x_3 & +\ldots\ldots & +a'_{nn}x_n & =b'_n
\end{array}
\qquad \ldots (5.2)
$$

O mesmo processo utilizado para se eliminar x_1 da equação 2 até n pode ser utilizado para se eliminar x_2 da equação 3 até n e, de uma forma genérica, eliminar-se

xi da equação i+1 até n (supondo-se i<n), levando-se finalmente o sistema à forma triangular superior (5.3). O algoritmo 5.1 traz em "pseudo-código" a etapa de triangularização.

$$
\begin{array}{cccccc}
u_{11}x_1 & +u_{12}x_2 & +u_{13}x_3 & +\ldots\ldots & u_{1n}x_n & =b'_1 \\
0 & +u_{22}x_2 & +u_{23}x_3 & +\ldots\ldots & u_{2n}x_n & =b'_2 \\
0 & +0 & +u_{33}x_3 & +\ldots\ldots & u_{3n}x_n & =b_3, \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
0 & 0 & 0 & 0 & +u_{nn}x_n & =b'_n
\end{array}
\qquad \ldots (5.3)
$$

ALGORITMO 5.1 - ETAPA DE TRIANGULARIZAÇÃO DO MÉTODO DE ELIMINAÇÃO DE GAUSS

N é a dimensão do problema
a(i , j) é o elemento de índices "i" e "j" da matriz de coeficientes
b(i) é o elemento de índice "i" do vetor de termos independentes

```
PARA i=1 até N-1 FAÇA
   PARA j=i+1 até N FAÇA
      sc = a( j , i )/a( i , i )
     PARA k = i+1 até N FAÇA
            a( j , k) = a( j , k )-sc*a( i , k )
     FIM_FAÇA
     b( j )=b( j )-a( j , i )*b( i )
   FIM_FAÇA
FIM_FAÇA
```

5.2.2 ETAPA DE RETROSUBSTITUIÇÃO

Estando o sistema de equações na forma triangular superior (5.2), a incógnita x_n pode ser imediatamente obtida como

$$x_n = b'_n/u_{nn}$$

e substituindo-se x_n na equação n-1, obtém-se a incógnita x_{n-1} dada por

$$x_{n-1} = (b'_{n-1} - u_{n-1} x_n)/u_{n-1,n-1}.$$

Observe-se que inicialmente obteve-se x_n e depois, utilizando-se x_n obteve-se x_{n-1}. Genericamente, tendo-se obtido $x_n, x_{n-1}, x_{n-2}, \ldots, x_{n-k}$, pode-se obter x_{n-k-1} através de

$$x_{n-k-1} = b'_{n-k-1} - \sum_{i=n-k}^{N} u_{n-k-1,\, i}\, x_i$$

Com esse processo determina-se todo o vetor incógnito x. O fato de as incógnitas x_k que vão sendo obtidas serem substituídas nas equações k-1 até 1 justifica o nome de retrosubstituição atribuído à etapa. O algoritmo 5.2 traz em "pseudo-código" a etapa de retrosubstituição do método de eliminação de Gauss.

ALGORITMO 5.2 - ETAPA DE RETROSUBSTITUIÇÃO DO MÉTODO DE ELIMINAÇÃO DE GAUSS

N é a dimensão do problema
u(i , j) é o elemento de índices "i" e "j" da matriz de coeficientes já na forma triangular superior.
b(i) é o elemento de índice "i" do vetor de termos independentes que juntamente com a matriz dos coeficientes foi submetido à etapa de triangularização
x(i) é o vetor incógnito.

```
x( n )=b( n )/u( n , n )  { obtém a incógnita xn }
PARA i=n-1 até 1 (incremento -1) FAÇA
   x( i ) = b( i )
   PARA j = i+1 até N FAÇA
      x( i ) = x( i ) -u( i , j)*x( j )
   FIM_FAÇA
   x( i ) = x( i )/u( i, i ) { obtém a incógnita xi }
FIM_FAÇA
```

5.3 A PARALELIZAÇÃO USUAL DO ALGORITMO " ELIMINAÇÃO DE GAUSS "

O método de eliminação de Gauss é um método direto, ou seja, sabe-se *a priori* quantas operações serão necessárias para se chegar à solução. Conforme apresentado em FREEMAN(1992) as etapas de triangularização e retrosubstituição são processos respectivamente da ordem de N^3 e N^2 operações do tipo ponto flutuante (sendo N a dimensão do problema). Portanto, principalmente a etapa de triangularização requer grande esforço computacional no caso de sistemas de grandes dimensões (usuais na utilização do método dos elementos finitos) e assim, sua paralelização é sem dúvida extremamente benéfica em termos de tempo de processamento. A possibilidade mais evidente de paralelização das etapas de triangularização e retrosubstituição do método de eliminação de Gauss é apresentada em exemplos básicos de publicações sobre paralelização de métodos numéricos (dentre essas publicações, aparece em FREEMAN(1992)) e por esse motivo é referenciada neste trabalho como forma "usual" de paralelização.

O algoritmo sequencial correspondente à etapa de triangularização é, basicamente, um aninhamento de três loops. O loop mais externo tem por índice "i" e esse índice corresponde à equação cuja incógnita correspondente (x_i) será eliminada nas equações i+1 até n (sendo "n" o número total de equações do sistema). Internamente ao loop "i" aparece o loop de índice "j" o qual indexa a equação na qual está sendo eliminada a incógnita x_i O loop mais interno (índice "k"), por sua vez, indexa qual elemento da equação "j" está sendo modificado.

A paralelização direta (sem necessidade de sincronização) do loop mais externo ("i"), embora seja a mais desejável por envolver a totalidade da etapa, não é possível pela clara alteração nas mesmas variáveis efetuadas por diferentes iterações do loop. Por exemplo, quando "i" é igual a 3 são alteradas as equações 4,5,6,7...n; essa iteração não pode ser realizada paralelamente à correspondente a "i" igual a 5 pois esta altera as equações 6,7...n como o faz a outra iteração (i=3). Já a paralelização do loop "j" (interno ao loop "i") pode ser feita diretamente já que as iterações alteram variáveis diferentes (equações diferentes).

A etapa sequencial de retrosubstituição é um alinhamento de dois loops. O loop mais externo tem por índice "i" e indexa a i-ésima incógnita do sistema que está

sendo obtida. O loop e índice "j" (interno ao loop "i") indexa as incógnitas de índice "j" já obtidas, que nesse loop são multiplicadas por coeficientes da equação "i".

A paralelização direta do loop mais externo é claramente não factível pois por exemplo, para se obter a incógnita x_{n-4} é necessário que tenham sido obtidas as incógnitas x_{n-3}, x_{n-2}, x_{n-1} e x_n, mostrando que a k-ésima iteração do loop depende da anterior, ou seja, as iterações do loop não podem ser efetuadas paralelamente.

O loop mais interno (índice "j"), na forma como foi apresentado (forma usual do algoritmo), não pode ser paralelizado diretamente pois diferentes iterações do índice "j" alteram a mesma incógnita x(i) impedindo a execução em paralelo do loop "j". Em FREEMAN(1992), encontra-se uma variante da etapa de retrosubstituição que permite a paralelização direta do loop mais interno. O "pseudo-código" da variante da retrosubstituição é apresentado no algoritmo 5.3.

ALGORITMO 5.3 - VARIANTE DA ETAPA DE RETROSUBSTITUIÇÃO

N é a dimensão do problema
u(i , j) é o elemento de índices "i" e "j" da matriz de coeficientes já na forma triangular superior.
b(i) é o elemento de índice "i" do vetor de termos independentes que juntamente com a matriz dos coeficientes foi submetido à etapa de triangularização.
x(i) é o vetor incógnito

```
PARA i=1 até n FAÇA
    x( i )=b(i )  { inicialização do vetor incógnito com o vetor
FIM_FAÇA                          de termos independentes  }
PARA i=n até 1 ( incremento -1 ) FAÇA
   x( i ) = x( i )/u( i , i ) { obtenção de x_i}
   PARA j = 1 até i-1 FAÇA
      x( j ) = x( j ) - u( j , i )*x( i ) { utilização de x_i nas outras
   FIM_FAÇA                                  incógnitas     }
FIM_FAÇA
```

No algoritmo variante de retrosubstituição (5.3), o loop mais externo do aninhamento ("i") continua não diretamente paralelizável já que a obtenção da incónita x_{i-1} depende da obtenção da incógnita x_i. O loop mais interno, no entanto, na forma variante pode ser diretamente paralelizado pois as diferentes iterações alteram variáveis também diferentes. Deve-se notar que no algoritmo variante de retrosubstituição quando uma incógnita é obtida, todas as futuras operações envolvendo a mesma são imediatamente realizadas no loop "j". Outra particularidade

é o fato do vetor incógnito ser inicializado igualando-o ao vetor de termos independentes resultante da etapa de triangularização.

O algoritmo paralelizado usual utilizado nesse trabalho é basicamente o algoritmo do método de eliminação de Gauss com as paralelizações já apresentadas para as etapas de triangularização e para a variante da etapa de retrosubstituição. Além disso, o algoritmo é adaptado à resolução dos sistemas de equações usuais resultantes da aplicação do método dos elementos finitos em análise estrutural, ou seja, matriz dos coeficientes simétrica, com característica de banda e armazenada em arranjo retangular.

A simetria permite que a matriz original (matriz quadrada) seja armazenada em um arranjo retangular onde o elemento genérico de índices "i" e "j" da matriz quadrada terá, nesse novo arranjo, índices "i" e "j-i+1". Desta forma, os elementos pertencentes à diagonal principal na forma quadrada, ocuparão a primeira coluna na forma retangular. Além da simetria, as matrizes de coeficientes oriundas do método dos elementos finitos na análise estrutural apresentam a peculiaridade da característica de banda : os elementos não nulos estão agrupados proximamente à diagonal principal, e desta forma são necessárias apenas algumas colunas no arranjo retangular para conter todos os elementos não nulos da matriz. Ao número de colunas do arranjo retangular que contém todos os elementos não nulos dá-se o nome de largura de semi banda. O algoritmo 5.4 traz a versão paralela usual do método de eliminação de Gauss.

ALGORITMO 5.4 - VERSÃO PARALELA USUAL DO MÉTODO DE ELIMINAÇÃO DE GAUSS PARA MATRIZ SIMÉTRICA COM ARRANJO EM BANDA

N é a dimensão do problema

MBAND é a largura de semi banda

a(i,j) é o elemento de índices "i" e "j" da matriz de coeficientes já triangularizada em arranjo retangular (N linhas e MBAND colunas).

b(i) é o elemento de índice "i" do vetor de termos independentes que juntamente com a matriz dos coeficientes é submetido à etapa de triangularização. No final do processamento b(i) torna-se o vetor incógnito procurado.

(continuação do algoritmo 5.4)

precis = precisão adotada no trabalho = 1.0e-15

```
/* TRIANGULARIZAÇÃO */

1  INÍCIO
2  PARA m = 1 até N-1 FAÇA
3     PARA L=2 até MBAND FAÇA_EM_PARALELO
4    { variáveis locais :L,sc,j,i e k }*
5          SE (ABS(a( m , L )) >= precis) FAÇA
6            sc = a( m , L )/a(m , 1)
7            j = 0
8            i = m - 1 + L
9            PARA k =2 até MBAND FAÇA
10              j = j + 1
11              a(i,j) = a(i,j) - sc * a(m,k)
12           FIM_FAÇA
13             b( i ) = b( i ) -sc*b(m)
14         FIM_SE
15   FIM_FAÇA_EM_PARALELO
16 FIM_FAÇA

/* RETROSUBSTITUIÇÃO */

17 PARA i = 1 até N FAÇA
18    m = N - i + 1
19    b( m ) = b( m )/a(m , 1)
20    lim = m - MBAND
21    SE lim < 1 ENTÃO lim = 1
22    PARA j = lim até m-1 FAÇA_EM_PARALELO**
23    { variável local : j }*
24       b(j) = b(j) - a(j,m-j+1) * b(m)
25    FIM_FAÇA_EM_PARALELO
26 FIM_FAÇA
27 FIM
```

* Essa declaração de variável local deve ser entendida como **local ao processador**. Geralmente as declarações de variável local ao processador são aplicadas às variáveis que dependem do índice que indexa o loop paralelo.

** Esse loop embora paralelizável, é etapa de menor esforço computacional no algoritmo e, dependendo da máquina utilizada, o custo computacional da preparação da paralelização poderá ser maior que os benefícios em termos de tempo de processamento obtidos. Nesses casos, quando é disponível o recurso da paralelização condicional (o loop só é paralelizado quando se cumpre uma condição, por exemplo, a dimensão de determinado vetor ser maior que um certo valor estabelecido), o algoritmo permite generalidade de utilização com bons resultados.

5.4 PARALELIZAÇÃO ALTERNATIVA DA ETAPA DE TRIANGULARIZAÇÃO DO MÉTODO DE ELIMINAÇÃO DE GAUSS

Conforme apresentado no ítem 5.3, na paralelização usual da etapa de triangularização do algoritmo método de eliminação de Gauss, o *loop* mais externo não é paralelizado, mas sim o imediatamente interno a este. Desta forma, a cada iteração do *loop* mais externo são criados processos paralelos para a execução concorrente do *loop* interno. Como a criação de processos paralelos implica em incremento de esforço computacional, essa criação de processos paralelos a cada iteração do *loop* mais externo (uma para cada linha da matriz dos coeficientes) pode representar um acréscimo não desprezível de tempo de processamento à etapa de triangularização. Esse problema foi o alvo principal do desenvolvimento do algoritmo alternativo.

O algoritmo alternativo de paralelização da etapa de triangularização elimina a montagem repetida dos processos paralelos transformando o *loop* mais externo e o imediatamente interno da paralelização usual em um *loop* paralelo único, requerendo portanto apenas uma criação de processos paralelos. Se, por exemplo, o *loop* mais exteno possui "n" iterações e o imediatamente inferior "m", o *loop* único possuirá "n*m" iterações e os índices antigos dos dois *loops* iniciais escrever-se-ão em função do índice do *loop* único. O algoritmo 5.5.a mostra em "pseudo código" um aninhamento de dois *loops* e o 5.5.b traz o algoritmo 5.5.a transformado em um *loop* único, onde são disponíveis os índices antigos dos dois loops transformados.

ALGORITMO 5.5.A - ANINHAMENTO DE *LOOPS*

```
PARA i=1 até FAÇA
    PARA J=1 até  m FAÇA
        instruções
    FIM_FAÇA
FIM_FAÇA
```

ALGORITMO 5.5.B - TRANSFORMAÇÃO DO ALGORITMO 5.5.A EM LOOP ÚNICO

```
div significa divisão inteira
PARA k=1,n*m FAÇA
    i = ( ( k-1 ) div m )+1
    j = k - ( i - 1 )*m
    instruções
FIM_FAÇA
```

Obviamente, no caso da etapa de triangularização do algoritmo do método de eliminação de Gauss, a simples transformação dos dois *loops* aninhados em um *loop* único não elimina o problema da "sequencialidade intrínseca" do *loop* original mais externo. Por exemplo, se no algoritmo paralelizado usual a iteração "i"' do *loop* mais externo não podia ser realizada antes da iteração "j", correpondentemente haverá no algoritmo alternativo um conjunto de iterações do *loop* único que não poderá ser executado antes de um outro conjunto de iterações. No esquema usual de paralelização da triangularização, como o *loop* mais externo é executado sequencialmente não há necessidade de sincronização. No algoritmo alternativo, no entanto, existe apenas um *loop* abrangendo todas as correspondentes iterações do *loop* mais externo da paralelização usual, implicando portanto na necessidade de sincronização do mesmo para a sua correta paralelização..

A sincronização do *loop* único do algoritmo alternativo de triangularização é feita em duas etapas : no início e no final da iteração. No início da iteração, há instruções para se verficar se a iteração pode ou não ser executada naquele momento; caso possa, a iteração é realizada e, caso contrário permanece aguardando liberação em um esquema semelhante ao bloqueio (*lock*). No final da iteração há instruções para serem liberadas as iterações que dependem da execução da iteração corrente.

Para se entender o processo de liberação de iterações, basta verificar que cada iteração do *loop* único do algoritmo alternativo corresponde à operação de uma linha (aqui chamada "linha base") sobre outra posterior (aqui chamada "linha alvo") para tornar nulo o elemento da linha alvo abaixo da diagonal principal da linha base. Assim, para que uma iteração do *loop* único possa ser realizada são necessárias duas condições : todas as linhas anteriores à linha base devem ter concluído suas operações sobre a mesma e sobre a linha alvo.

O controle de liberação de uma iteração do *loop* único do algoritmo alternativo é feito com a utilização de dois *arrays :* um para controlar a liberação da linha base (LINBASE[i]) e outro para controlar a liberação da linha alvo (LINALVO[i]). Assim, para uma iteração "i "ser liberada é necessário que haja a liberação de sua linha base "n " (LINBASE[n] = 1) e que haja também a liberação de sua linha alvo correpondente à iteração "i" (LINALVO[i] = 1).

Ao ser concluída a iteração, as iterações dependentes da iteração corrente são

liberadas, ou seja, é atribuído valor 1 para algum elemento de LINBASE[i] e ou de LINALVO[i]. Desta forma, é necessário saber-se qual iteração é liberada pela iteração corrente. Essa informação está contida no *array* QUAL_A [i] , que indica qual iteração é liberada pela iteração "i ". O *array* QUAL_A[i] é gerado anteriormente ao procesamento do algoritmo. Quando uma linha base conclui as operações sobre a primeira linha alvo abaixo dela, esta linha alvo está liberada para ser linha base, ou seja, se esta linha alvo é "k", LINBASE[k] será 1. A tabela 5.1 traz o exemplo da triangularização pelo algoritmo alternativo mostrando para cada iteração do *loop* único qual é a linha base, qual é a linha alvo e quais iterações ou linhas bases são liberadas. O exemplo refere-se à triangularização de uma matriz simétrica com arranjo em banda tendo 30 linhas e 4 colunas. São apresentadas as primeiras quatorze iterações.

TABELA 5.1

iteração	**linha base**	**linha alvo**	**linha base liberada**	***iteração liberada**
1	1	2	2	6
2	1	3	-	4
3	1	4	-	5
4	2	3	3	9
5	2	4	-	7
6	2	5	-	8
7	3	4	4	12
8	3	5	-	10
9	3	6	-	11
10	4	5	5	15
11	4	6	-	13
12	4	7	-	14
13	5	6	6	18
14	5	7	-	16

* iteração correspondente à linha alvo liberada

É interessante observar na tabela 5.1 que a iteração 1 do *loop* único, além de liberar a linha base 2, libera também a iteração 6 que tem como linha base 2 e linha alvo 5, linha esta não trabalhada pela linha base 1. Essa liberação da linha alvo 5 para a linha base 2, sem da operação da linha base 1 sobre a linha alvo 5 decorre dá caracteristica de banda e de simetria da matriz . Pela característica de banda, sabemos que o elemento (1,5) da matriz é nulo e por simetria o elemento (5,1) também o é e, desta forma dispensa o trabalho da linha base 1 sobre a linha alvo 5, já que o elemento da linha alvo 5 abaixo da diagonal principal da linha base 1 já é nulo. O mesmo fato aparece nas iterações 4,7,10 e 13 da tabela 5.1 . O algoritmo 5.6 traz as operações prévias necessárias à execução do algoritmo paralelo alternativo de triangularização.

ALGORITMO 5.6 - INICIALIZAÇÃO PARA O ALGORITMO PARALELO ALTERNATIVO DE TRIANGULARIZAÇÃO

ndes = número de linhas da matriz dos coeficientes

mband = número de colunas da matriz dos coeficientes (largura de semi banda)

```
INÍCIO
PARA  i=1 até ndes*(mband-1) FAÇA
  LINALVO[ i ] = 0
  QUAL_A[ i ] = 0
FIM_FAÇA
LINBASE[ 1 ] =1
PARA i=1 até mband FAÇA
   LINALVO[ i ] =1
FIM_FAÇA
PARA i = 2 até ndes FAÇA
   LINBASE[ i ] = 0
FIM_FAÇA
PARA i=1 até ndes FAÇA
  index = ( i - 1 )*(mband - 1)+1
  QUAL_A[ index ] = index + 2*(mband - 1) -1
  PARA kk = 1 até mband - 2 FAÇA
      QUAL_A[ index +  kk ] =  index + kk + mband -2
  FIM_FAÇA
FIM_FAÇA
FIM
```

Para se conseguir o máximo desempenho do algoritmo alternativo de paralelização da etapa de triangularização é necessária uma adequada estratégia de escalonamento (*schedule*) na partição das iterações do *loop* único entre os processadores para a execução concorrente. O objetivo desse escalonamento é fazer com que os processadores quase sempre executem iterações liberadas, não permanecendo em ociosidade. Por exemplo, numa partição em que as iterações são simplesmente divididas (escalonamento simples) entre os processadores, ou seja, o primeiro processador fica com as primeiras "n" iterações,..., até o último que fica com as últimas "n" iterações, sendo "n" o resultado da divisão do número de iterações pelo número de processadores , os últimos processadores ficarão com as iterações correspondentes às últimas linhas base e, desta forma, ficarão ociosos até a liberação das mesmas.

A estratégia ideal para a partição das iterações é o escalonamento intercalado, ou seja, o primeiro processador fica com a iteração 1, m+1, 2*m+1,...e etc, sendo "m" o número de processadores; o segundo processador fica com a iteração 2, m+2, 2*m+2,...e etc. Com essa partição os processadores vão em conjunto das primeiras às últimas linhas base, evitando a ociosidade. O escalonamento intercalado pode também ser feito em "pedaços" (*chunk*), onde o primeiro processador recebe as iterações 1 até k, m*k+1 até m*k + k,... e etc. Essa estratégia de escalonamento em "pedaços" pode beneficiar o desempenho pois, ao pegar um conjunto contíguo de iterações, as operações em memória poderão ser feitas em posições próximas o bastante para propiciar uma eficiente utilização de eventual memória *cache* dos processadores. O algoritmo 5.7 traz em "pseudo código" a paralelização alternativa da etapa de triangularização do método de eliminação de Gauss.

ALGORITMO 5.7 - PARALELIZAÇÃO ALTERNATIVA DA ETAPA DE TRIANGULARIZAÇÃO DO MÉTODO DE ELIMINAÇÃO DE GAUSS PARA MATRIZ SIMÉTRICA COM ARRANJO EM BANDA

* O algoritmo 5.6 deve ser previamente executado

ndes = número de linhas da matriz dos coeficientes

mband = número de colunas da matriz dos coeficientes

(continuação do algoritmo 5.7)

div = divisão inteira

abs = valor absoluto

a(i,j) é o elemento de índices "i" e "j" da matriz de coeficientes já triangularizada em arranjo retangular (N linhas e MBAND colunas).

b(i) é o elemento de índice "i" do vetor de termos independentes que juntamente com a matriz dos coeficientes é submetido à etapa de triangularização.

precis = precisão adotada no trabalho = 1.0 e -15

```
1  INÍCIO
2  PARA im=1 até ndes*(mband -1) FAÇA_EM_PARALELO
3     n = ( im + mband - 2) div (mband - 1 )
4      L = im - ( n - 1 )*(mband - 1 ) + 1
5     ENQUANTO NÃO (( LINBASE[ n ] =1) E (LINALVO [ im ]=1)) FAÇA
6      FIM_FAÇA
7     SE ( abs ( a( n , L ) ) ≥  precis ) ENTÃO
8        sc = a( n , L )/ a( n , 1 )
9         j = 0
10       i = n + 1 - L
11       PARA k = L até mband FAÇA
12           j = j + 1
13          a( i , j ) = a( i , j ) - sc*a( n , k )
14       FIM_FAÇA
15       b( i ) = b ( i ) - sc*b( n )
16    FIM_SE
17     LINALVO[QUAL_A[ im ] = 1
18    SE ( LINALVO[ QUAL_A[im]+1] ≻ LINALVO[QUAL_A[im+1]]) ENTÃO
19          LINBASE[n +1]=1 1
20     FIM_SE
21 FIM_FAÇA_EM_PARALELO
22 FIM
```

5.4 AVALIAÇÃO DE DESEMPENHO DOS ALGORITMOS PARALELOS USUAL E ALTERNATIVO DO MÉTODO DE ELIMINAÇÃO DE GAUSS

O desempenho (*speed-up*) dos algoritmos paralelos (usual e com triangularização alternativa) do método de eliminação de Gauss para matriz dos coeficientes simétrica com arranjo em banda foi avaliado em implementações efetuadas em computador Silicon Graphics modelo 4S-440D com arquitetura paralela. As medidas partiram de um programa de análise linear de treliças tridimensionais. Foram resolvidos os sistemas de equações lineares correspondentes às análise lineares de quatro treliças tridimensionais, com detalhes apresentados na tabela 5.2.

TABELA 5.2			SIST. DE EQUAÇÕES LINEARES		
EXEMPLO	BARRAS	PONTOS NODAIS	LINHAS	COLUNAS	FIGURA
GBG	1568	451	1251	675	5.1
GBP	1632	420	1224	72	5.2
PBG	648	181	531	300	5.3
PBP	672	176	504	48	5.4

Tipicamente as matrizes de rigidez de treliças são esparsas e, desta forma, para serem obtidos resultados de desempenho dos algoritmos para matrizes sem esta característica, foi processado o exemplo "PBG" não se considerando os zeros da matriz. Para se fazer isso basta a alteração do teste de precisão da etapa de triangularização (linha 7 dos algoritmos 5.4 e 5.7) , fazendo que a condição do mesmo seja sempre verdadeira (por exemplo fazendo **SE** 1=1 **ENTÃO**...). O exemplo "PBG" analisado como se não fosse representado por matriz esparsa é referenciado como exemplo "PBGNE".

Conforme foi apresentado, o escalonamento intercalado é fundamental para o bom desempenho do algoritmo alternativo de triangularização. No caso do algoritmo usual na utilização de matrizes esparsas, esse esquema torna-se também grandemente necessário ao desempenho, sob pena de degradação do balanceamento de trabalho e consequentemente do desempenho. Isto se deve ao fato de que sendo a matriz esparsa, em muitas iterações não se cumpre a condição da linha 7 do algoritmo 5.4, tornando mitigado o esforço computacional da iteração. Se várias iterações de menor esforço computacional (iterações menores) ficam concentradas em poucos processadores, haverá prejuízo do balanceamento de trabalho. Como as iterações menores não estão regularmente distribuídas entre as iterações do loop paralelizado da triangularização , o escalonamento intercalado as distribui mais equilibradamente entre os processadores, promovendo a melhora do balanceamento de trabalho. Todos os exemplos da tabela 5.2 e o exemplo "PGNE" foram executados com escalonamento intercalado de "pedaço" 8. Para mostrar a queda de desempenho com a utilização de escalonamento simples, o exemplo "PBG" foi também processado com tal escalonamento, sendo esse exemplo referenciado por "PBGES".

Embora o algoritmo alternativo de paralelização atue apenas na etapa de triangularização, a comparação com a paralelizacão usual se deu no tempo de processamento total do algoritmo de eliminação de Gauss, visto que a etapa de retrosubstituição, nos exemplos testados, sempre representou menos de um por cento do tempo total de processamento. Na tabela 5.3 aparecem as duplas "**tempo obtido (segundos) / Speed-up**" no processamento do sistema de equações lineares dos diversos exemplos. Como no computador utilizado a solução é extremamente rápida para pequenos problemas , as medições foram feitas em repetições (*loops*) da etapa, ou seja, o tempos obtidos são a divisão dos tempos totais pelo número de repetições. Em seguida, apresentam-se as figuras correspondentes aos exemplos processados e os gráficos de *speed-up*.

TABELA 5.3		NÚMERO DE PROCESSADORES				
Exemplo	Algoritmo	* 1	2	3	4	repetições
GBG	usual	348/ 1,00	208/ 1,67	141/ 2,47	113/ 3,08	1
GBG	alternativo	348/ 1,00	207/ 1,68	136/ 2,56	112/ 3,11	1
GBP	usual	4,6/ 1,00	3,1/ 1,48	2,3/ 2,00	2,0/ 2,30	10
GBP	alternativo	4,6/ 1,00	3,3/ 1,39	2,4/ 1,92	2,1/ 2,19	10
PBG	usual	22,5/ 1,00	14,5/ 1,55	10,5/ 2,14	9,0/ 2,50	4
PBG	alternativo	22,5/ 1,00	14,5/ 1,55	10,5/ 2,14	9,0/ 2,50	4
PBP	usual	0,70/ 1,00	0,51/ 1,37	0,38/ 1,84	0,35/ 2,00	100
PBP	alternativo	0,70/1,00	0,53/ 1,32	0,41/ 1,71	0,41/ 1,71	100
PBGNE	alternativo	62/ 1,00	39/ 1,59	27/ 2,30	21/ 2,95	1
PBGES	usual	22,5/1,00	21,8/ 1,03	17,8/ 1,26	15,3/ 1,47	4

* Para 1 processador utilizou-se o menor dos tempos entre os dois algoritmos (usual e alternativo), permitindo a comparação de *speed-up* com base no mesmo tempo sequencial. O algoritmo alternativo mostrou-se ligeiramente mais lento para processamento sequencial em virtude da sincronização (que não seria necessária nesse caso).

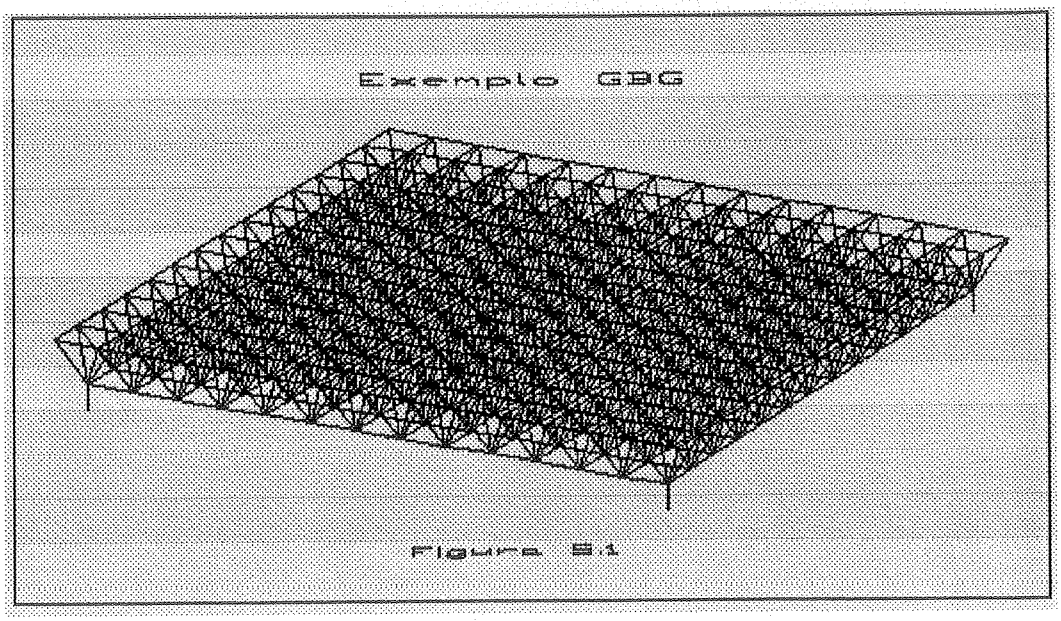
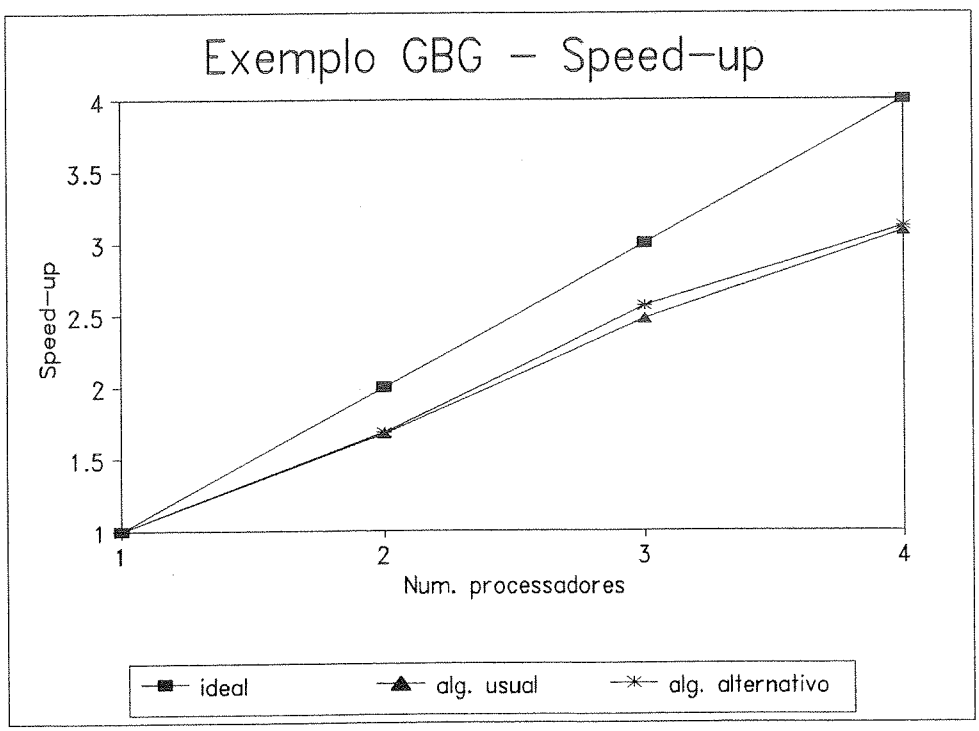
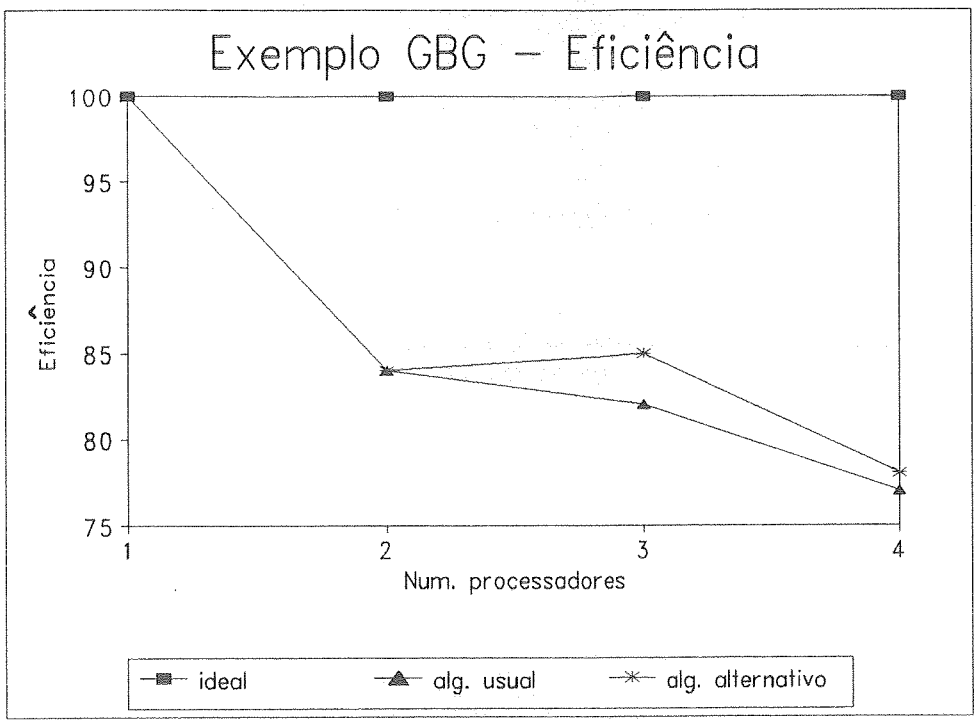

Exemplo GBG

Figura 5.1


Exemplo GBG - Speed-up
Speed-up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo


Exemplo GBG – Eficiência
100
95
90
85
80
75
Eficiência
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo

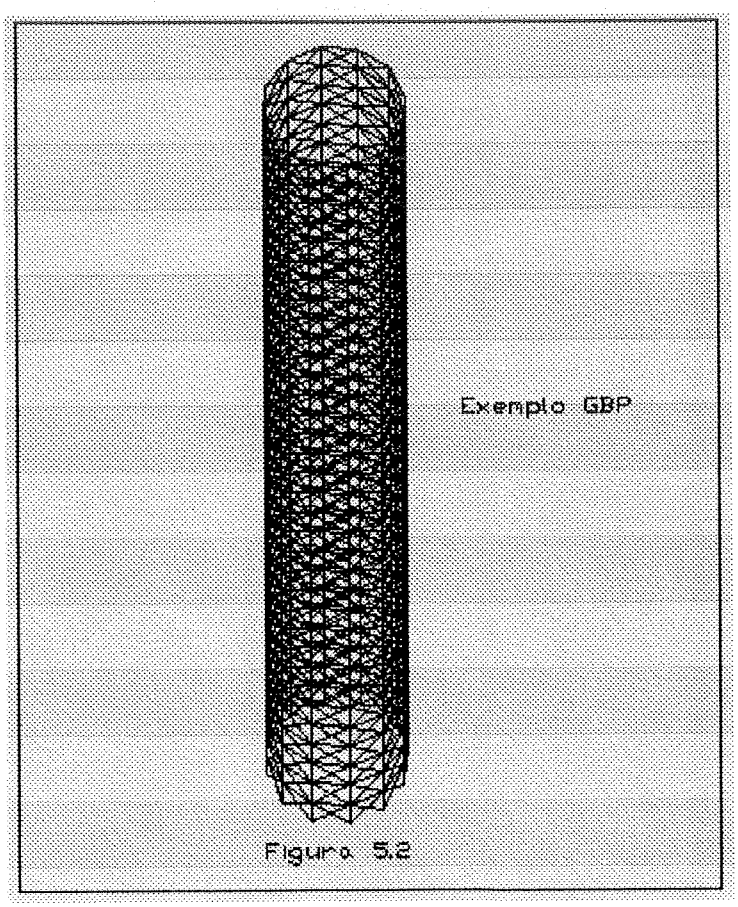
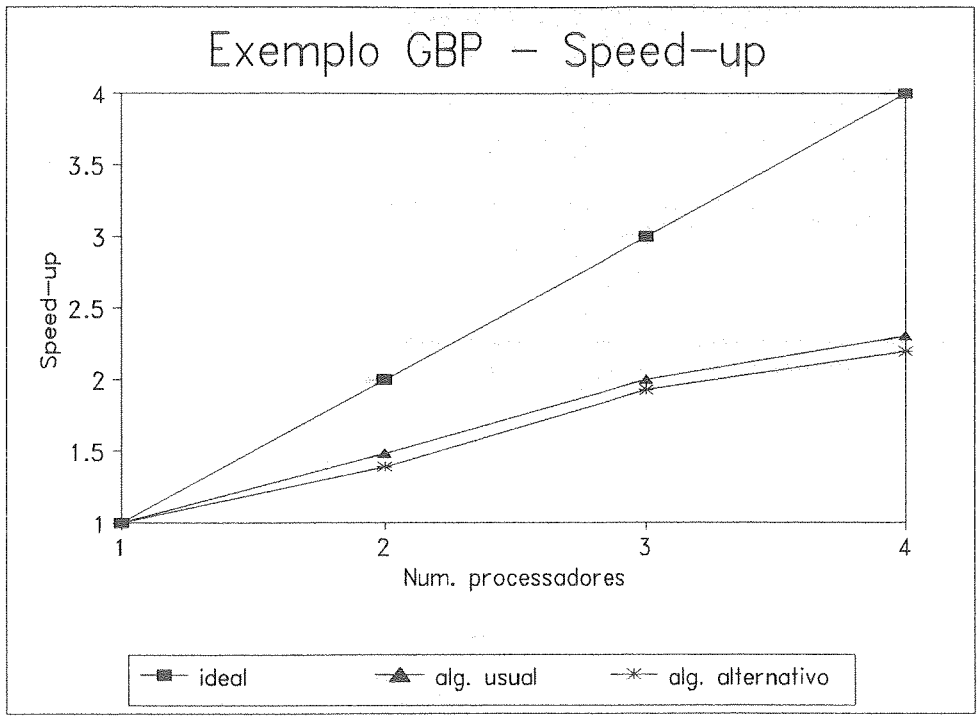
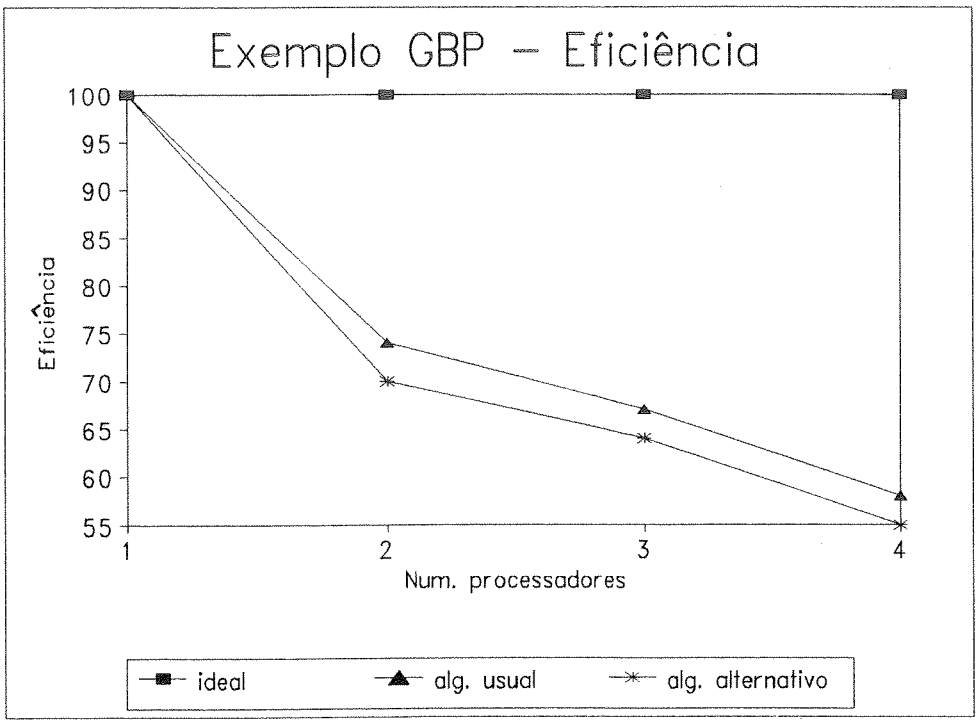

Exemplo GBP

Figura 5.2


Exemplo GBP – Speed–up
Speed–up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo


Exemplo GBP – Eficiência
Eficiência
100
95
90
85
80
75
70
65
60
55
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo

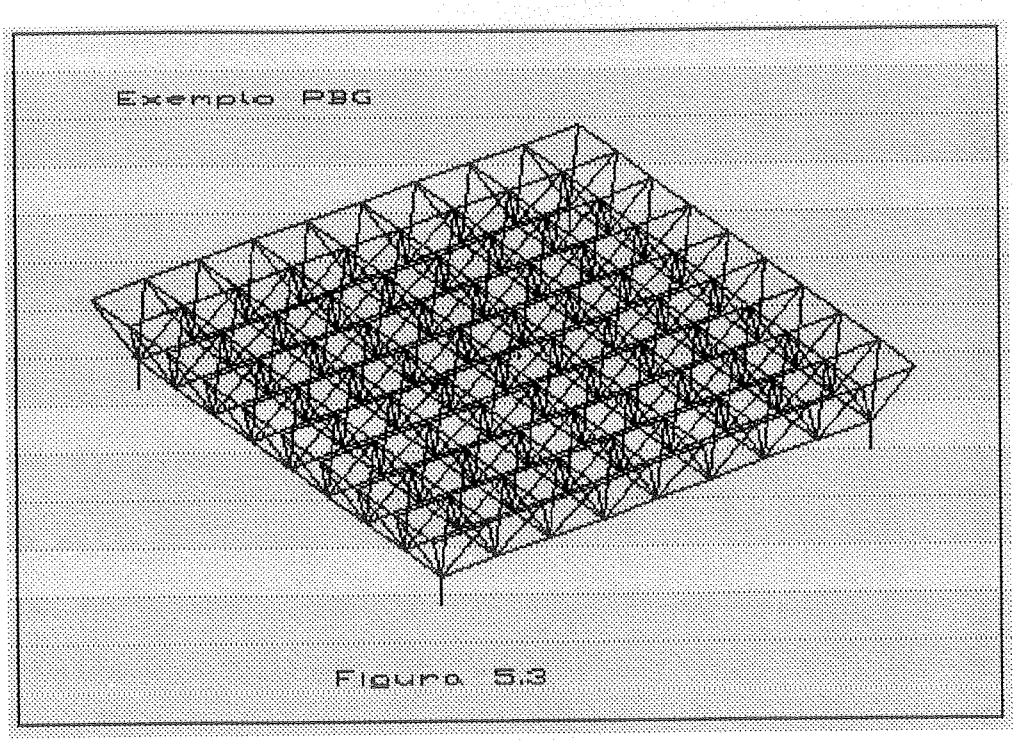
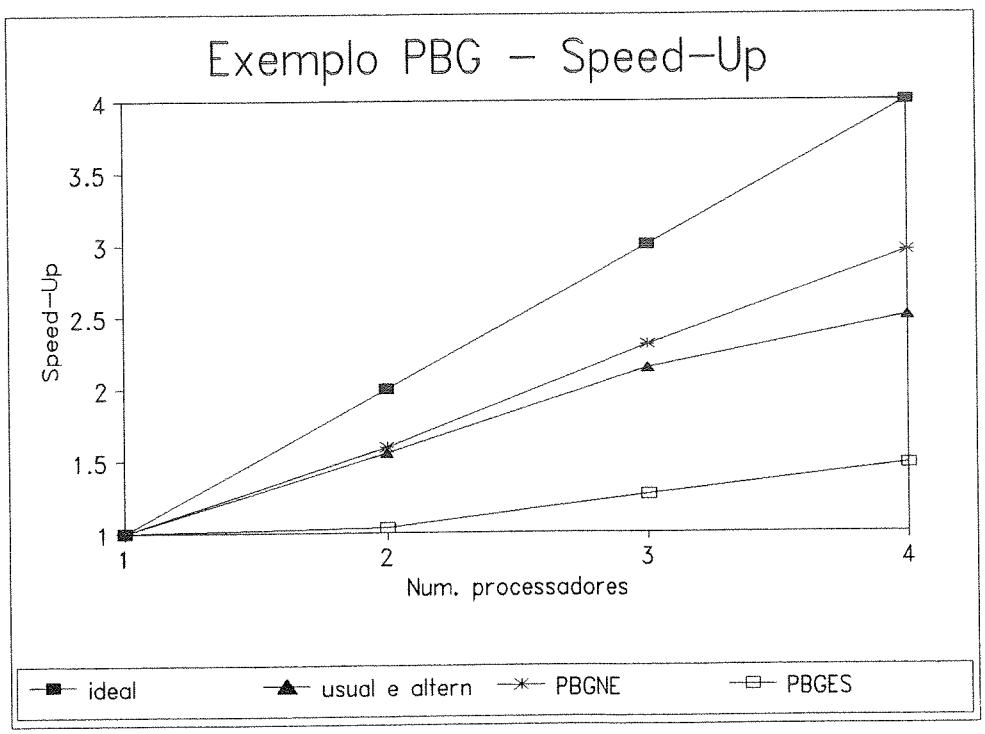
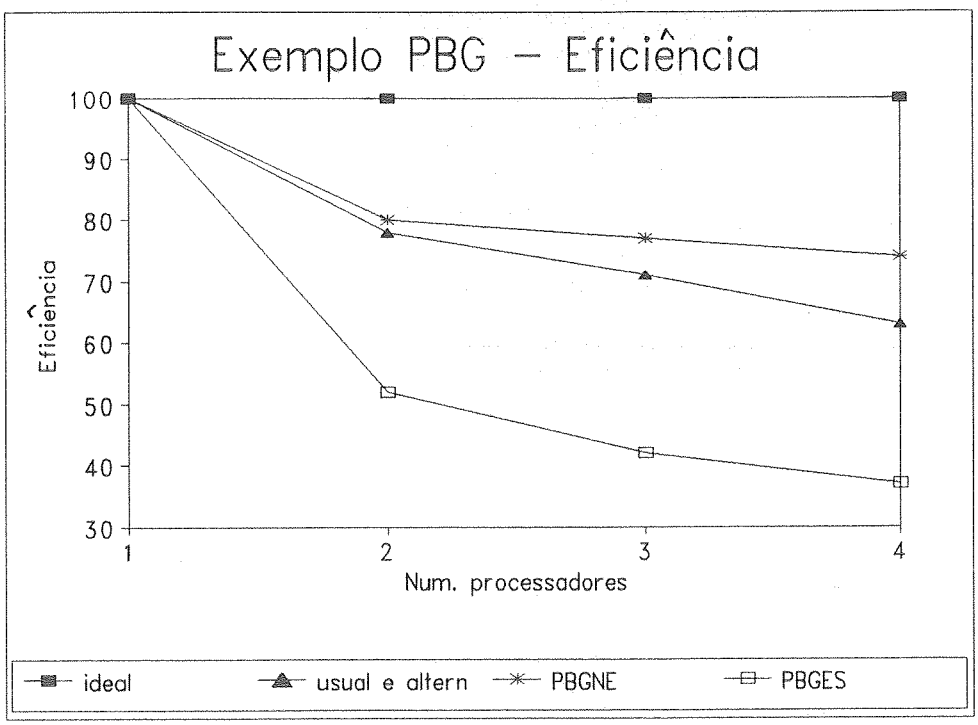

Exemplo PBG

Figura 5.3


Exemplo PBG - Speed-Up
Speed-Up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
usual e altern
PBGNE
PBGES


Exemplo PBG – Eficiência
100
90
80
70
60
50
40
30
Eficiência
1
2
3
4
Num. processadores
ideal
usual e altern
PBGNE
PBGES

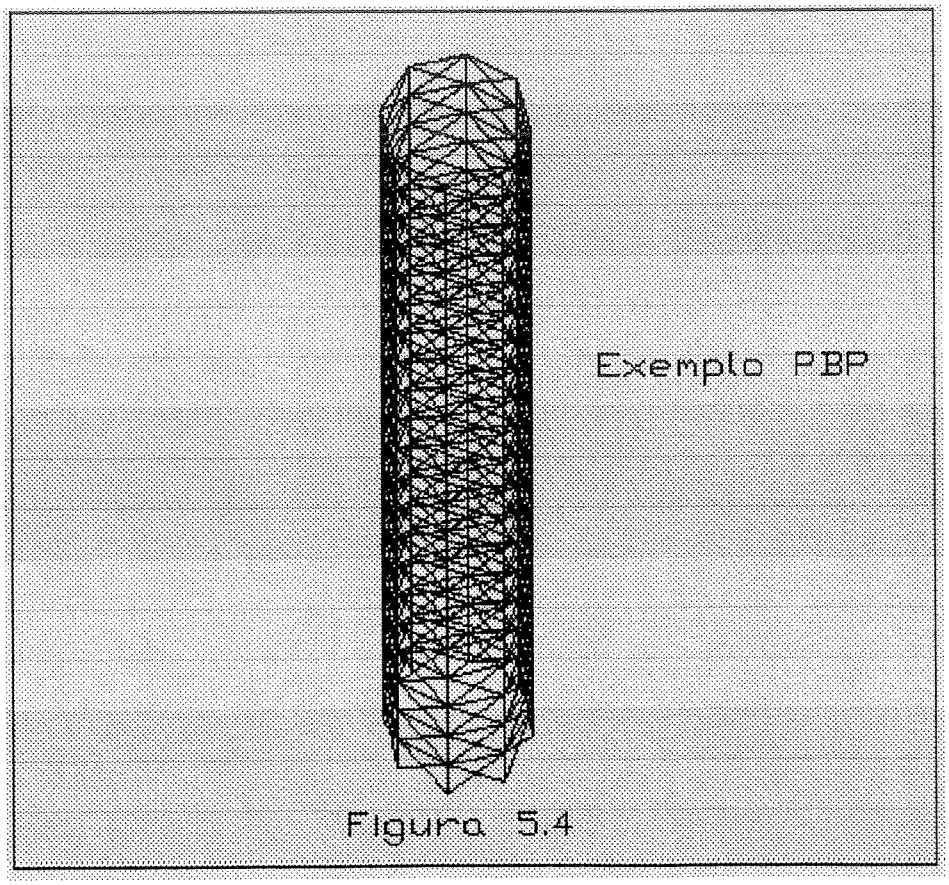
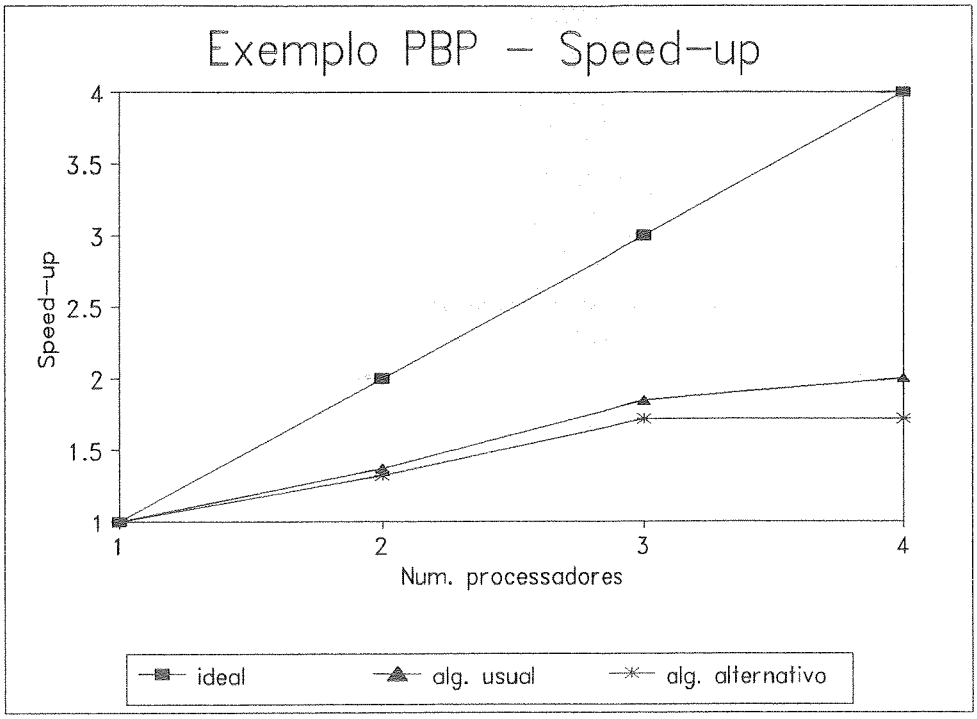
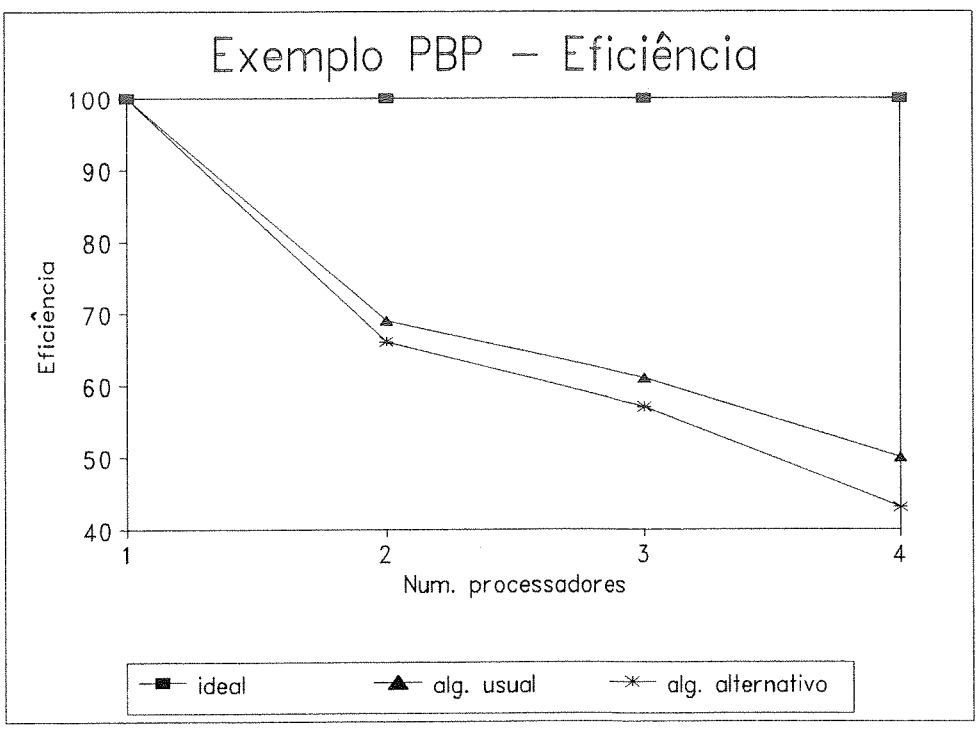

Exemplo PBP

Figura 5.4


Exemplo PBP – Speed-up
Speed-up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo


Exemplo PBP – Eficiência
Eficiência
100
90
80
70
60
50
40
1
2
3
4
Num. processadores
ideal
alg. usual
alg. alternativo

5.6 - ANÁLISE DOS RESULTADOS OBTIDOS

Nos testes efetuados, o algoritmo alternativo mostrou-se superior ao usual em sistemas com largura de semi banda mais elevada, fato que se inverteu nos sistemas de pequena largura de semi banda. Assim, nos exemplos processados, o esforço computacional da sincronização (algoritmo alternativo) foi se tornando menor com o aumento da largura de semibanda do sistema do que o esforço da repetida criação de processos da triangularização (algoritmo usual). É interessante observar que esses esforços computacionais ligados a cada um dos algoritmos são mais dependentes da máquina (esforço de criação de processos paralelos) no caso do algoritmo usual e mais dependentes do esquema de sincronização usado no programa, para o caso do algoritmo alternativo. Assim, o algoritmo alternativo pode ser otimizado com a utilização de um mais eficiente esquema de sincronização, o qual pode inclusive ser variável, em função das características do sistema a ser resolvido.

Além da comparação entre o algoritmo usual e alternativo, deve também ser observado o incremento de desempenho de ambos com o aumento da largura de semibanda e consequentemente, com o aumento do número de operações paralelizadas. Esse fato, já por várias vezes comentado neste trabalho, reflete a diminuição proporcional do custo computacional de paralelização com o aumento da complexidade (em número de operações) do problema.

O exemplo PBGES é um caso onde mostra-se vultosa a importância do balanceamento de trabalho. Com o escalonamento simples utilizado na triangularização, as iterações de maior esforço computacional concentraram-se em um processador, refletindo notoriamente na degradação do desempenho.

5.7 COMPARAÇÃO COM RESULTADOS CONHECIDOS

Conforme já abordado no capítulo 4, a comparação direta entre resultados de *speed-up* produzidos por máquinas diferentes não é conveniente, porém como referência, citar-se-á um resultado de speed-up utilizando estratégia usual de paralelização do método de eliminação de Gauss: Em FREEMAN(1992), foi obtido *speed-up* 3,0 para quatro processadores com matriz quadrada de dimensões 256 x 256 não esparsa e computador Encore Multimax 520. Na mesma publicação citam-se exemplos de melhor desempenho, porém em arquitetura de memória local (hipercubo).

6- UTILIZAÇÃO DE ALGORITMOS PARALELOS NA ANÁLISE NÃO LINEAR DE TRELIÇAS TRIDIMENSIONAIS

6.1 INTRODUÇÃO

Neste capítulo, os algoritmos paralelos propostos para a montagem da matriz de rigidez da estrutura e solução de sistema de equações lineares são utilizados na implementação de um programa para análise não linear física e geométrica de treliças tridimensionais. Os conceitos teóricos envolvendo a teoria não linear de estruturas são apresentados resumidamente, visando unicamente proporcionar o entendimento básico desses problemas e, em seguida, das paralelizações efetuadas no programa desenvolvido. Um aprofundamento teórico pode ser obtido em MATTHIES & STRANG(1979) e em OWEN(1980). As implementações foram realizadas em computador Silicon Graphics 4S - 440D sendo que as características do mesmo, da linguagem utilizada e da forma como foram feitas medidas de tempo de processamento encontram-se no anexo deste trabalho.

6.2 PROBLEMAS ESTRUTURAIS NÃO LINEARES

Diz-se que um problema estrutural é não linear quando não há linearidade entre a magnitude das forças aplicadas à estrutura e a consequente magnitude dos deslocamentos verificados na mesma. A não linearidade nos problemas estruturais manifesta-se em duas classes de problemas : a não linearidade geométrica (abreviadamente NLG) e a não linearidade física (abreviadamente NLF), sendo que NLG e NLF podem se apresentar isoladamente ou associadas no mesmo problema.

A não linearidade geométrica está relacionada à deformabilidade da estrutura, a qual implica em variações da geometria da mesma ao longo da aplicação das forças, levando ao aparecimento de esforços e deslocamentos possivelmente muito distintos

dos que se obteriam em uma análise de primeira ordem (onde desprezam-se tais variações). As treliças construídas em alumínio (material de baixo módulo de elasticidade, se comparado ao do aço, por exemplo) representam usual exemplo de estrutura que pode requerer uma análise de NLG para uma avaliação mais confiável de deslocamentos e esforços.

A não linearidade física por sua vez, aparece quando da utilização de materiais em níveis de tensão tais que não se configura a obediência à lei de Hooke, ou seja, o material é submetido a um trecho não linear de seu diagrama tensão x deformação. As treliças de aço nas quais se permite a plastificação do material de algumas de suas barras é um exemplo passível de análise de NLF.

6.3 MATRIZ DE RIGIDEZ TANGENTE E MÉTODOS INCREMENTAIS ITERATIVOS DE ANÁLISE NÃO LINEAR

Em uma análise linear, a matriz de rigidez da estrutura não depende dos deslocamentos nodais da mesma, os quais são obtidos diretamente pela solução do sistema de equações lineares (6.1) :

$$\mathbf{Ka} = \mathbf{f} \qquad 6.1$$

onde **K** é matriz de rigidez da estrutura, **a** é o vetor de deslocamentos e **f** o vetor de cargas nodais equivalentes. No caso de análise de problemas onde impera a NLG, a relação entre o vetor de cargas e o vetor de deslocamentos se dá por intermédio de uma matriz de rigidez que é função dos próprios deslocamentos a serem obtidos :

$$\mathbf{K_s(a)\ a} = \mathbf{f} \qquad 6.2$$

A matriz $\mathbf{K_s(a)}$ é denominada matriz de rigidez secante. A figura 6.1.a traz um exemplo de diagrama força (P) x deslocamento (d) onde ocorre a não linearidade. Pode-se observar que, na figura 6.1.a, uma reta intercepta a curva nos pontos P=f e d=a, representando a solução do sistema da expressão 6.2, simbolizando a atuação da matriz $\mathbf{K_s(a)}$ e evidenciando a denominação "matriz de rigidez secante".

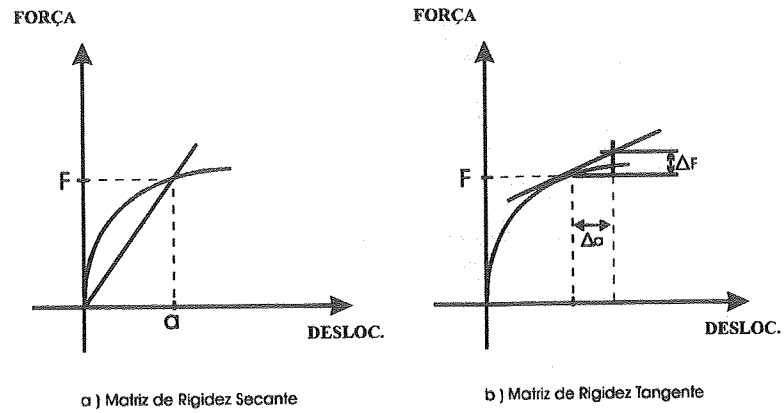


Fig. 6.1: Comportamento Não-Linear

Conforme mostrado em RUBERT(1993), ao se aplicar à expressão 6.2 um procedimento semelhante a se desevolver $\mathbf{K_s(a)}$ na forma de série de Taylor truncada no termo linear, obtém-se uma relação entre um acréscimo de força Δf e o consequente acréscimo de deslocamento Δa :

$$\mathbf{K_t(a)\Delta a = \Delta f} \qquad 6.3$$

A matriz $\mathbf{K_t(a)}$ é denominada "matriz de rigidez tangente" . A figura 6.1.b exemplifica a relação entre um acréscimo de carga Δ**f** e o correpondente acréscimo de deslocamento Δ**a** relacionados pela matriz tangente, ilustrando também a razão do nome "matriz de rigidez tangente".

O conceito de "matriz de rigidez tangente" é a essencia do tratamento de problemas não lineares por uma abordagem incremental iterativa. Nesse tipo de abordagem, (exemplificando para o caso de uma estrutura sujeita a um conjunto de forças) , as forças aplicadas são divididas em uma série de incrementos. Para cada incremento de força obtém-se uma aproximação para o incremento de deslocamento correspondente via matriz de rigidez tangente. Na posição deslocada aproximada

obtida, a resultante das tensões na estrutura gerará uma forças que diferem das forças aplicadas originalmente (no equilíbrio seriam iguais). Essas diferenças entre as forças aplicadas na estrutura e a forças resultantes das tensões na estrutura em sua posição deformada constituem o resíduo. Obtido o resíduo (vetor de resíduos no caso de vários graus de liberdade), o mesmo é reaplicado à estrutura visando o restabelecimento do equilíbrio, gerando um novo resíduo. Esse processo iterativo continua até que o resíduo seja, por algum critério métrico, inferior a um valor estabelecido de precisão. Ocorrendo a convergência pela minimização do resíduo, um novo incremento de forças é aplicado. Assim, o deslocamento total da estrutura se dá por uma somatória dos deslocamentos obtidos em cada incremento de forças.

6.4 ALGORITMO DO MÉTODO INCREMENTAL ITERATIVO NEWTON RAPHSON MODIFICADO 2

A diferenciação entre os diversos métodos incrementais iterativos situa-se na maneira como se processa a atualização da matriz de rigidez global ao longo do carregamento. Em RUBERT(1993) encontra-se um ótimo estudo comparativo entre diversos métodos incrementais iterativos. Nos diversos exemplos abordados nesse trabalho pôde-se notar que o método NEWTON RAPHSON MODIFICADO TIPO 2 (abreviadamente NMR2) apresentou convergência aos resultados esperados, porém mostrou-se estar posicionado medianamente entre métodos eficientes (BFGS, por exemplo) e ineficientes (método da rigidez inicial, por exemplo) em termos de esforço computacional requerido. A opção pelo NRM2 teve por objetivo evidenciar a importância da paralelização (e consequente incremento de desempenho computacional) de algoritmos eficientes em termos da precisão de resultados numéricos, porém limitados no tocante à rapidez de processamento.

O algoritmo incremental iterativo NRM2 caracteriza-se por efetuar a atualização da matriz de rigidez global na segunda iteração de cada incremento, além da primeira montagem da matriz global, usualmente realizada por ocasião do início do processamento. A partir da segunda iteração a matriz de rigidez global é mantida constante no incremento. A figura 6.2 apresenta um diagrama carga x deslocamento não linear e, paralelamente, uma ilustração de como tal comportamento seria tratado

pelo método NRM2. As retas tangentes à curva correspondem a cada iteração; e a variação do ângulo que as mesmas fazem com o eixo das abcissas representa as atualizações da matriz de rigidez global. O algoritmo 6.1 traz em "pseudo código" um exemplo de esquema básico de programa de análise não linear de treliças por método incremental iterativo utilizando NRM2.

ALGORITMO 6.1 - ANÁLISE NÃO LINEAR DE TRELIÇAS UTILIZANDO O MÉTODO INCREMENTAL ITERATIVO "NEWTON RAPHSON MODIFICADO TIPO 2 ".

```
1  INÍCIO
2   Leia_os_dados
3   *monte_a_matriz de rigidez_global
4  PARA i=1 até Número_de_incrementos FAÇA
5      monte_vetor_de_cargas_nodais
6      convergência = 0
7      iteração = 1
8     ENQUANTO convergência = 0 FAÇA
9       SE iteração = 2 ENTÃO
10            *monte_a_matriz de rigidez_global
11      FIM_SE
12      resolva_sistema_de_equações_lineares
13      calcule_esforços_nas_barras
14      calcule_vetor_resíduos
15      calcule_norma_vetor_resíduos
16      SE norma <= norma_esperada ENTÃO
17            Convergência =1
18      CASO_CONTRÁRIO
19            monte_vetor_de_cargas_nodais
20            iteração = iteração + 1
21      FIM_SE
22    FIM_ENQUANTO
23 FIM_FAÇA
24 Mostre_resultados
25 FIM
```

* Considera-se que o algoritmo de montagem da matriz de rigidez global inclui a imposição de condições de contorno.

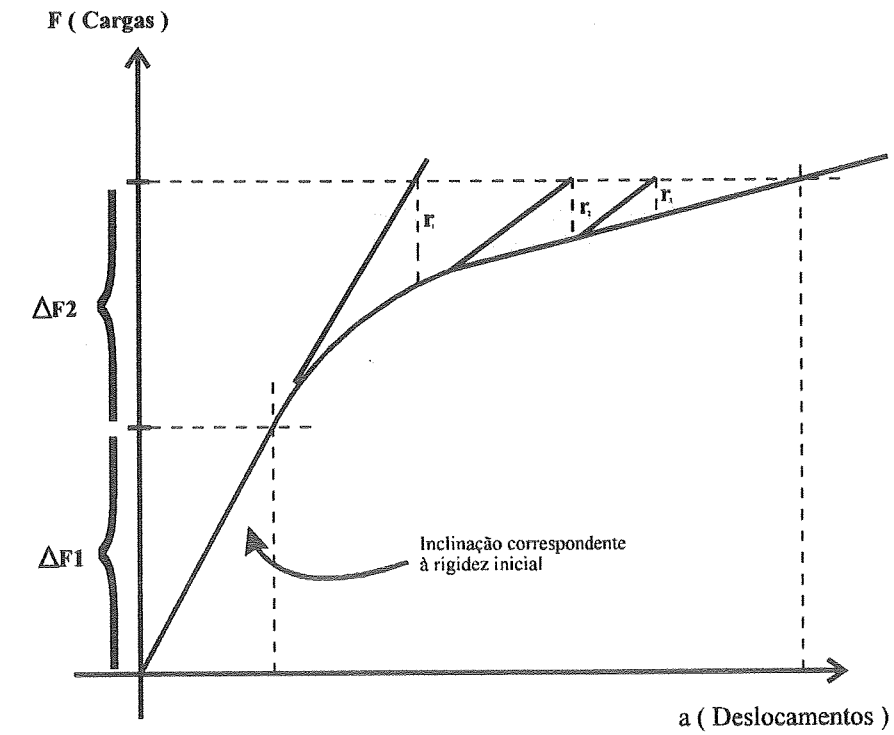


Fig. 6.2: NRM2

6.5 MATRIZ DE RIGIDEZ TANGENTE PARA BARRA DE TRELIÇA

Conforme pode ser observado em RUBERT(1993), a matriz de rigidez tangente de uma barra de treliça plana ($\mathbf{K}_t$) em seu sistema local de coordenadas (figura 6.3) é dada por :

$$\mathbf{K}_t = \mathbf{K}_0 + \Delta\mathbf{K}_0 + \mathbf{K}_g$$

onde $\mathbf{K}_0$ é a matriz de rigidez da teoria elástica linear,

$\Delta \mathbf{K}_0$ é a matria que efetua a correção do sistema local de coordenadas em função dos deslocamentos ocorridos e

$\mathbf{K}_g$ é a usual matriz geométrica, a qual é função do nível da solicitação axial da barra.

A correção $\Delta \mathbf{K}_0$ apresentada é necessária pois, com as seguidas iterações, vão

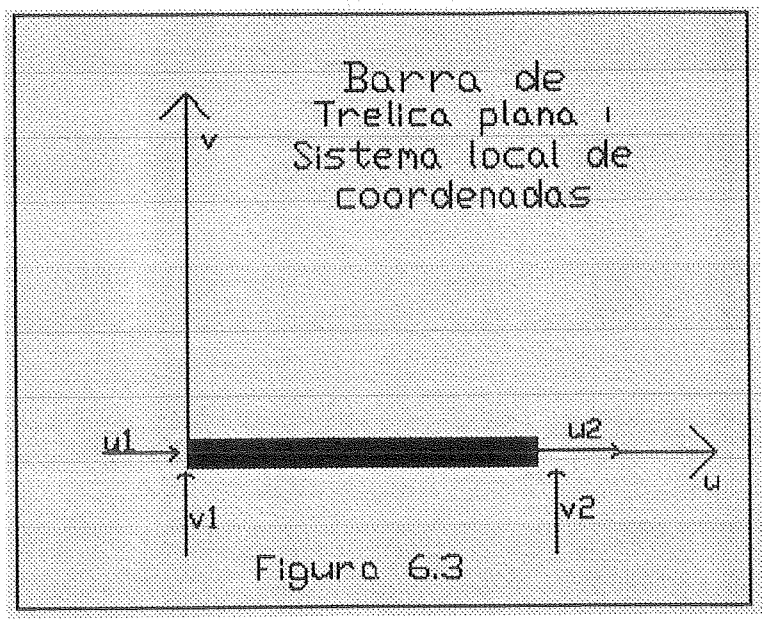


Figura 6.3

ocorrendo deslocamentos que alteram a geometria da estrutura, tornando o sistema inicial de coordenadas globais não representativo do estado corrente da estrutura. Impondo-se um giro ao sistema local e depois rotacionando-se do sistema local para o global (que é mantido fixo), consegue-se representar o histórico de deslocamentos da estrutura ao longo da aplicação das forças.

Alternativamente, neste trabalho optou-se por um sistema de coordenadas globais que altera-se a cada iteração, atualizando as coordenadas dos nós. Desta forma, na passagem do sistema local para o global, tal passagem se dá para um sistema global já deslocado pelas iterações anteriores e dispensa a correção $\Delta \mathbf{K}_0$ apresentada. Assim, com o sistema global "móvel", a matriz de rigidez tangente barra de treliça plana em seu sistema local de coordenadas fica dada simplesmente por :

$$\mathbf{K}_t = \mathbf{K}_0 + \mathbf{K}_g \qquad 6.5$$

A discussão apresentada para o caso de treliças planas é também obviamente válida para o caso de treliças tridimensionais. As matrizes locais $\mathbf{K}_0$ e $\mathbf{K}_g$ para esse tipo de barra de treliça são explicitamente apresentadas a seguir, para o sistema local de coordenadas (figura 6.4) .

$$[k_0] = \frac{EA}{L}\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[K_g] = \frac{N}{L}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

onde A é a área da seção transversal da barra de treliça,

E é o módulo de elasticidade do material,

N é o esforço axial na barra e

L é o comprimento da barra.

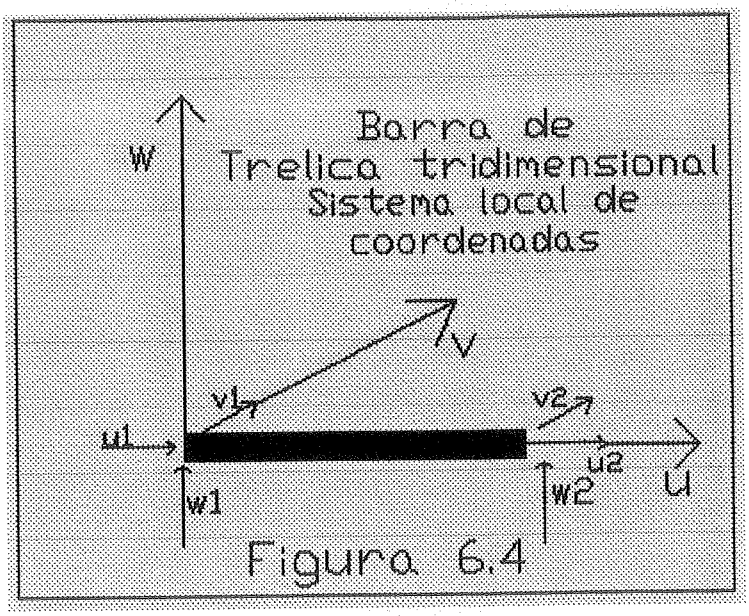


Figura 6.4

Para o tratamento da não linearidade física, além da geométrica, adotou-se, um módulo de elasticidade "E" variável (função do nível de deformação axial) para ser utilizado na expressão de $\mathbf{K}_0$, através da atualização do mesmo a cada cálculo das deformações da barra. O modelo adotado de diagrama tensão x deformação foi o "bilinear", ou seja, "E" constante e igual a "E1" para deformação (em módulo) de zero até uma deformação dada "ε1" e, para deformações maiores, "E" constante e igual a "E2". Os valores de E1, E2 e ε1 são definidos pelo usuário do programa.

6.6 PROGRAMA PARA ANÁLISE NÃO LINEAR DE TRELIÇAS TRIDIMENSIONAIS

6.6.1 PARALELIZAÇÕES REALIZADAS

O esquema básico do programa implementado partiu do algoritmo 6.1, o qual foi otimizado pela paralelização das seguintes etapas : montagem da matriz de rigidez global (linhas 3 e 10 do algoritmo 6.1), solução do sistema de equações lineares (linha 12 do algoritmo 6.1), cálculo do vetor de resíduos (linha 14 do algoritmo 6.1), cálculo da norma Euclidiana do vetor de resíduos (linha 15 do algoritmo 6.1) e cálculo do esforço axial das barras (linha 13 do algoritmo 6.1). Algumas etapas não explícitas no algoritmo também foram paralelizadas, como é o caso do cálculo dos comprimentos e co-senos diretores das barras e da atualização

das coordenadas do sistema global.

Conforme já comentado, a paralelização de etapas de menor esforço computacional deve ter sua viabilidade avaliada em função do custo de paralelização (variável de máquina para máquina). Em testes efetuados com o programa, mesmo para os menores problemas (treliças com 160 barras), as paralelizações isoladas das etapas mostraram ser eficazes, diminuindo o tempo de processamento e sugerindo terem sido frutíferas tais paralelizações.

A paralelização da montagem da matriz de rigidez global foi feita com o conceito de abordagem nodal plena apresentado no capítulo 4. É interessante notar que em problemas não lineares, onde a matriz de rigidez global será montada várias vezes, a montagem da estrutura de dados auxiliar (algoritmo 4.3) só é necesária por ocasião da primeira montagem da matriz global, permanecendo a estrutura na memória para as montagens seguintes e assim, aumentando a eficiência do programa. Na paralelização da montagem da matriz de rigidez global foi também incluída a imposição de condições de contorno, conforme mostrado no capítulo 4. Para essa etapa, analogamente, o cálculo das posições efetivas (algoritmo 4.7) só é feito na primeira imposição de condições de contorno, ficando tais posições na memória.

Com a eliminação de linhas e colunas durante a imposição de condições de contorno via posições efetivas, fica menos evidente a determinação da largura de semi-banda. Opcionalmente, esta pode ser feita também juntamente com a matriz de rigidez global e, portanto ser paralelizada. Para isso, deve-se utilizar um *array* MAXDIR[i] (de dimensão igual ao número de pontos) que indicará, para cada ponto montado, qual foi o maior índice de coluna gerado. Ao final da montagem da matriz global, basta a obtenção do maior valor encontrado em MAXDIR[i], o qual será a largura de semibanda. No programa implementado, essa foi a forma de obtenção da largura de semi banda, obviamente só realizada na primeira montagem da matriz de rigidez global.

A solução do sistema de equações lineares foi implementada utilizando-se o algoritmo paralelo alternativo (capítulo 5), embora os testes efetuados não tenham comprovado a superioridade desse algoritmo ou do usual. A opção pelo algoritmo alternativo se deu pelo interesse em submetê-lo a numerosas execuções e, desta

forma, testar o seu esquema de sincronização, o qual sempre mostrou-se correto.

O cálculo do vetor de resíduos foi outra etapa paralelizada do programa. O resíduo é, para cada grau de liberdade da treliça, a soma da ação externa aplicada e da ação conjunta das barras no referido grau de liberdade. Havendo o equilíbrio, o resíduo se anula, sendo portanto um indicador de tal situação. Assim como na montagem da matriz de rigidez global, no cálculo do vetor de resíduos pode fazer uso da "abordagem nodal", inserindo-se no vetor de resíduos a "contribuição" de cada nó (três graus de liberdade, no caso de treliças tridimensionais). Inicialmente, deve-se obter, para cada ponto nodal, a componente da ação das barras ligadas ao mesmo em cada um dos graus de liberdade. A componente de cada barra em cada grau de liberdade dada pelo produto de seu esforço axial pelo co-seno diretor (para a estrutura deslocada) da barra relacionado ao grau de liberdade. As três componentes (nas direções "X", "Y" e "Z" do sistema global) da ação das barras em cada ponto nodal podem ser facilmente calculadas com o uso da estrutura de dados auxiliar da etapa de montagem da matriz de rigidez (algoritmo 4.3), a qual indica quais barras estão ligadas aos pontos nodais e ainda qual é a numeração local do ponto nodal para a barra. O algoritmo 6.2 traz em "pseudo-código" o cálculo das componentes da ação das barras nos três graus de liberdade do ponto nodal. Obtidas, para cada ponto nodal, as componentes "X", "Y" e "Z" da ação das barras, basta, para cada um desses graus de liberdade a adição da ação externa correspondente, gerando os resíduos correspondentes ao ponto nodal. Criando-se um *loop* para o cálculo da contribuição para o vetor de resíduos de cada ponto nodal, as iterações desse *loop* alterarão variáveis diferentes, que são os elementos do vetor de resíduos correspondentes aos graus de liberdade do ponto nodal. Desta forma, o *loop* pode ser diretamente paralelizado (algoritmo 6.3).

ALGORITMO 6.2 - CÁLCULO DAS COMPONENTES DA AÇÃO DAS BARRAS NOS GRAUS DE LIBERDADE GLOBAIS DO PONTO NODAL

compx é a componente "X" da ação das barras no ponto nodal,
compy é a componente "Y" da ação das barras no ponto nodal,

(continuação do algoritmo 6.2)

compz é a componente "Z" da ação das barras no ponto nodal,
CX[i] = co-seno diretor da barra, relacionado ao grau de liberdade "X"
CY[i] = co-seno diretor da barra, relacionado ao grau de liberdade "Y"
CZ[i] = co-seno diretor da barra, relacionado ao grau de liberdade "Z"
NCONC[k] indica o número de elementos ligados ao ponto k
ELEM_CONC[k,i] indica o número do i-ésimo elemento ligado ao ponto k
NO_CORRESP[k,i] é o índice local do ponto k para o i-ésimo elemento a ele ligado
ESF[i] é o esforço axial na barra "i ".
k é o ponto nodal para o qual serão calculadas as componentes da ação das barras
Obs : NCONC[k], ELEM_CONC[k,i] e NO_CORRESP[k,i] são oriundos do processamento do algoritmo 4.3. CX[i], CY[i], CZ[i] e ESF[i], por hipótese, já estão calculados.

```
INÍCIO
compx = 0
compy = 0
compz = 0
PARA i = 1 até NELCNV[ k ] FAÇA
   elem = ELEMCO[ k , i ]
   SE  ( (NORELAT[ k , i ] = 1 )  ENTÃO  /* ponto nodal inicial da barra */
         compx = compx + ESF[ elem ] * CX[ elem ]
         compy = compy + ESF[ elem ] * CY[ elem ]
         compz = compz + ESF[ elem ] * CZ[ elem ]
   CASO CONTRÁRIO /* ponto nodal final da barra */
         compx = compx - ESF[ elem ] * CX[ elem ]
         compy = compy - ESF[ elem ] * CY[ elem ]
         compz = compz - ESF[ elem ] * CZ[ elem ]
    FIM_SE
FIM_FAÇA
FIM
```

ALGORITMO 63 - CÁLCULO DO VETOR DE RESÍDUOS PARA A TRELIÇA TRIDIMENSIONAL

npon é o número de pontos nodais da treliça
EXT_X[i] é a ação externa no grau de liberdade "X" do ponto nodal "i "
EXT_Y[i] é a ação externa no grau de liberdade "Y" do ponto nodal "i "
EXT_Z[i] é a ação externa no grau de liberdade "Z" do ponto nodal "i "
RESÍDUO[k] é o vetor de resíduos

```
INÍCIO
PARA j  = 1 até NPON FAÇA_EM_PARALELO
     k = j
     algoritmo 6.2
     RESÍDUO[ 3*( j - 1 ) +1 ] = compx + EXT_X[ j ]
     RESÍDUO[ 3*( j - 1 ) +2 ] = compy + EXT_Y[ j ]
     RESÍDUO[ 3*( j - 1 ) +3 ] = compz + EXT_Z[ j ]
FIM_FAÇA_EM_PARALELO
FIM
```

O cálculo paralelo da norma Euclidiana do vetor de resíduos foi realizado utilizando-se estratégia semelhante à apresentada no exemplo de uso de barreiras (*barriers*) no ítem 3.2.3.1. Para se calcular a norma Euclidiana de um vetor, é feita a somatória do quadrado de cada um dos elementos, sendo posteriormente extraída a raiz quadrada dessa somatória. No cálculo paralelo da somatória, cada processador efetua uma parte da mesma (elevando ao quadrado alguns dos elementos do vetor) e inserindo o resultado em uma variável "soma parcial" própria (uma para cada processador). Terminados os processos paralelos, a somatória é obtida pela soma das contribuições das "somas parciais" dos processadores. A existência de uma soma parcial em variável exclusiva do processador é essencial para se evitar a atualização simultânea da somatória. Obtida a somatória, a norma Euclidiana é dada pela raiz quadrada da mesma.

As paralelizações do cálculo de variáveis ligadas às barras (esforço axial, comprimento e co-senos diretores) e a paralelização da atualização das coordenadas globais foram feitas diretamente nos *loops* sequenciais originais, já que as iterações dos mesmos alteram diferentes variáveis. No caso do cálculo de variáveis ligadas às barras, cada iteração altera as variáveis referentes à própria barra e, no caso da atualização das coordenadas globais, cada iteração altera as coordenadas correspondentes a um ponto nodal e, assim, em ambos os casos não há a possibilidade de ocorrência da atualização simultânea (*racing condition*).

6.6.2 EXEMPLO DE PROCESSAMENTO

6.6.2.1 DESCRIÇÃO DO PROBLEMA

Para exemplificar a utilização do sistema implementado, processou-se um problema envolvendo uma treliça tridimensional (figura 6.5) onde aparecem associadas a não linearidade física e a não linearidade geométrica. A treliça possui as seguintes características :

✓ Dimensões : largura = comprimento = 1400cm; altura 100cm.

✓ Material das barras : é suposto um material de diagrama tensão x deformação bilinear, partindo de módulo de elasticidade E1 = 21000 KN/cm^2 até a deformação $\epsilon 1$ = 0,00114, onde o módulo de elasticidade passa a ser E2 = 2100 KN/cm^2. Com

esses dados o material deixa o comportamento elástico linear ao atingir tensão σ = 23,9 KN/cm^2 e tem características semelhantes às de alguns aços estruturais usuais.

✓ Área da seção transversal das barras : as barras do plano superior da treliça foram definidas com área de 4cm^2, as do plano inferior com 6cm^2 e as diagonais (barras de ligação entre os planos superior e inferior da treliça) foram definidas com área de 2 cm^2.

✓ Sequência de aplicação das cargas : o carregamento total foi dividido em três incrementos, no primeiro deles aplicou-se 50% da carga e nos dois incrementos subsequentes foram aplicados 25% da carga.

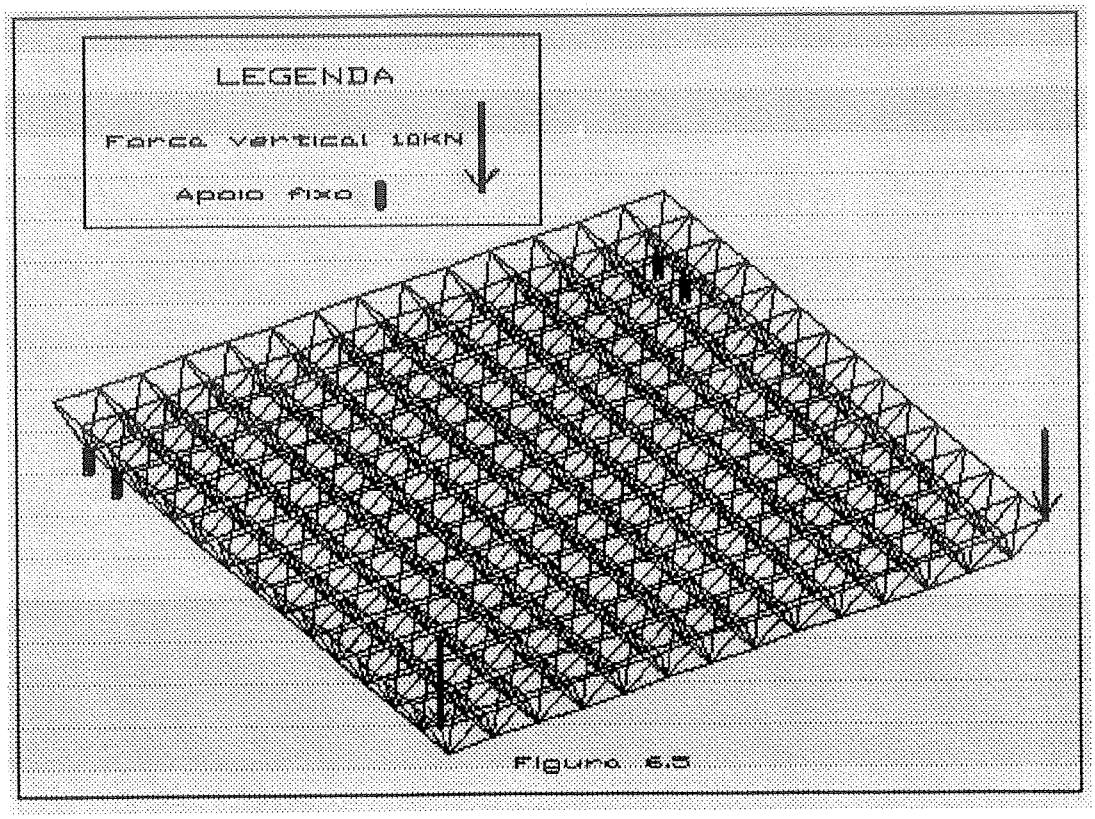


Figura 6.5

✓ Precisão utilizada na verificação do equilíbrio : quociente entre a norma Euclidiana do vetor de resíduos e a correspondente norma do vetor de cargas menor que 0.01.

6.6.2.2 - RESULTADOS OBTIDOS

Foram necessárias 20 iterações para a solução do problema, sendo 5 para o primeiro incremento (50% da carga), 8 para o segundo (75% da carga) e 7 para o terceiro (100% da carga). Com a totalidade da carga aplicada ocorreu a plastificação de 8 diagonais (figura 6.6). A flecha final sob as cargas foi de 17,03cm.

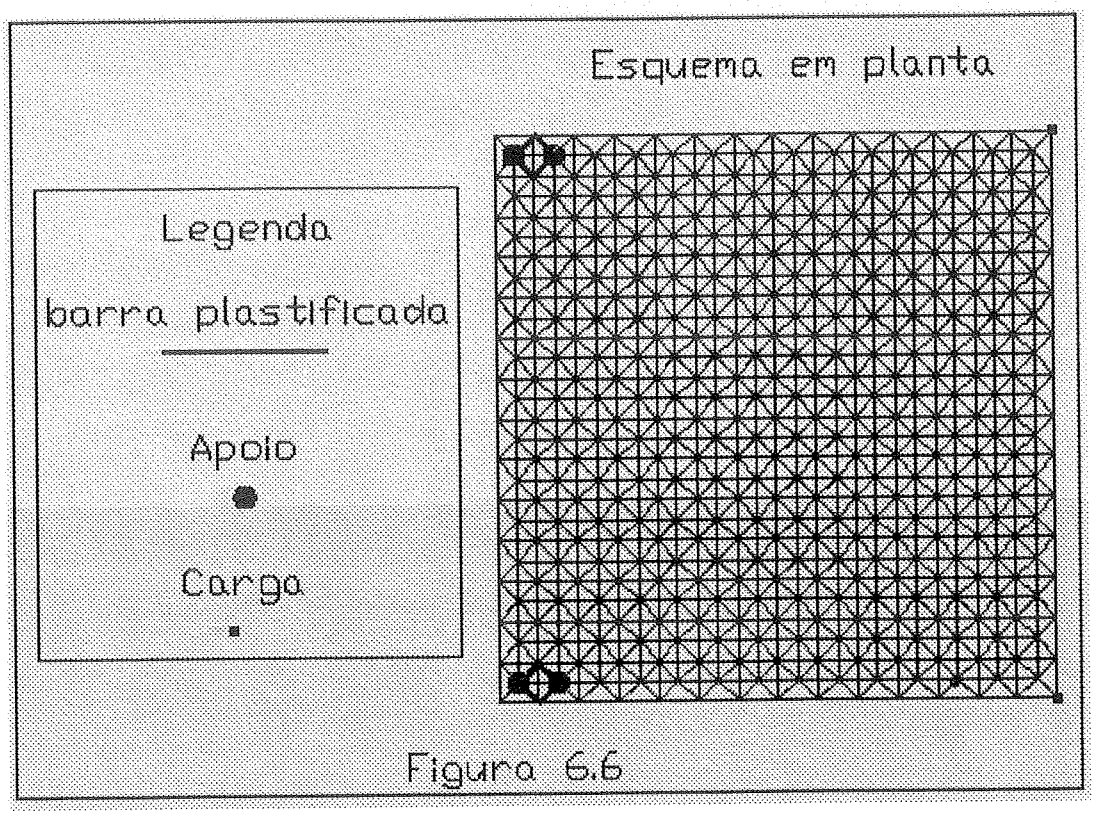


Figura 6.6

6.6.2.3 - DESEMPENHO PARCIAL E GLOBAL DO PROGRAMA

Além da medida de tempo de processamento de todo o programa , foram feitas medidas de tempo de processamento das seguintes etapas : montagem da matriz de rigidez global da estrutura, solução do sistema de equações lineares, cálculo do vetor de resíduos e cálculo da norma Euclidiana do vetor de resíduos.

Na primeira solução do sistema de equações lineares, estando a estrutura indeformada, e no caso com muitas barras com sistema local paralelo ao global, a matriz de rigidez global é bastante esparsa, fazendo com que o tempo de

processamento seja menor do que para as próximas iterações, quando as deformações eliminam os eventuais paralelismos entre eixos locais e globais. O tempo de solução do sistema de equações lineares na segunda iteração foi o considerado nas tabelas e gráficos de *speed-up* apresentados a seguir. O algoritmo utilizado foi o alternativo (capítulo 5) para processamento com 2,3 e 4 processadores, e o usual para 1 processador (o algoritmo usual apresenta menor tempo de processamento sequencial, conforme já mostrado no capítulo 5). A tabela 6.1 traz as duplas "**tempo obtido(segundos) / *speed-up***", seguindo a notação :

TT = tempo total de processamento do programa ,

MG = tempo de montagem da matriz de rigidez global,

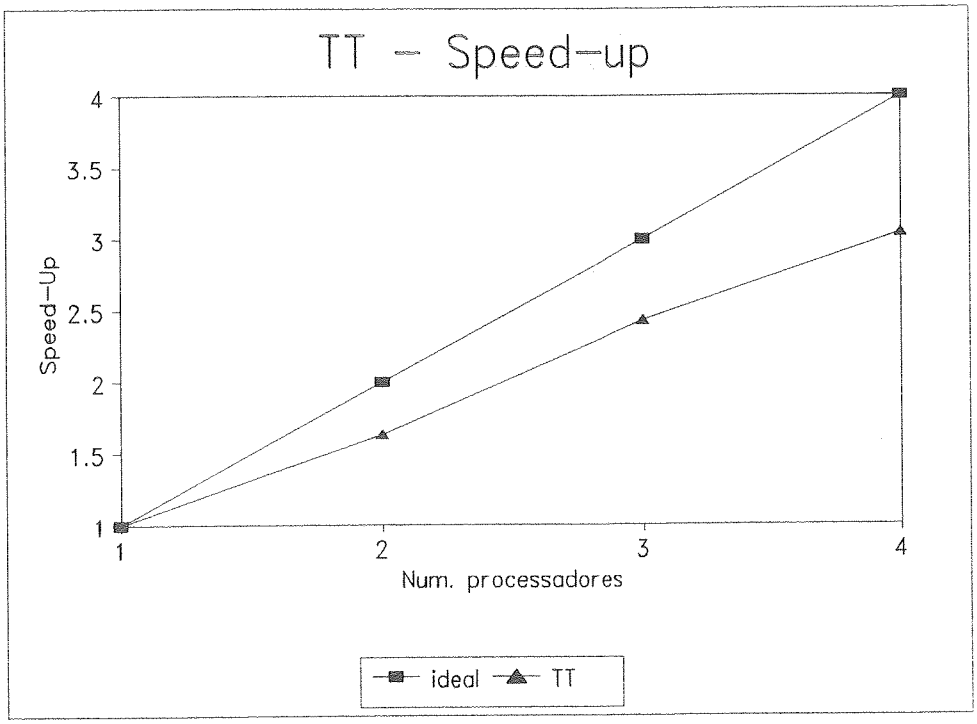
SE = tempo de solução do sistema de equações lineares,

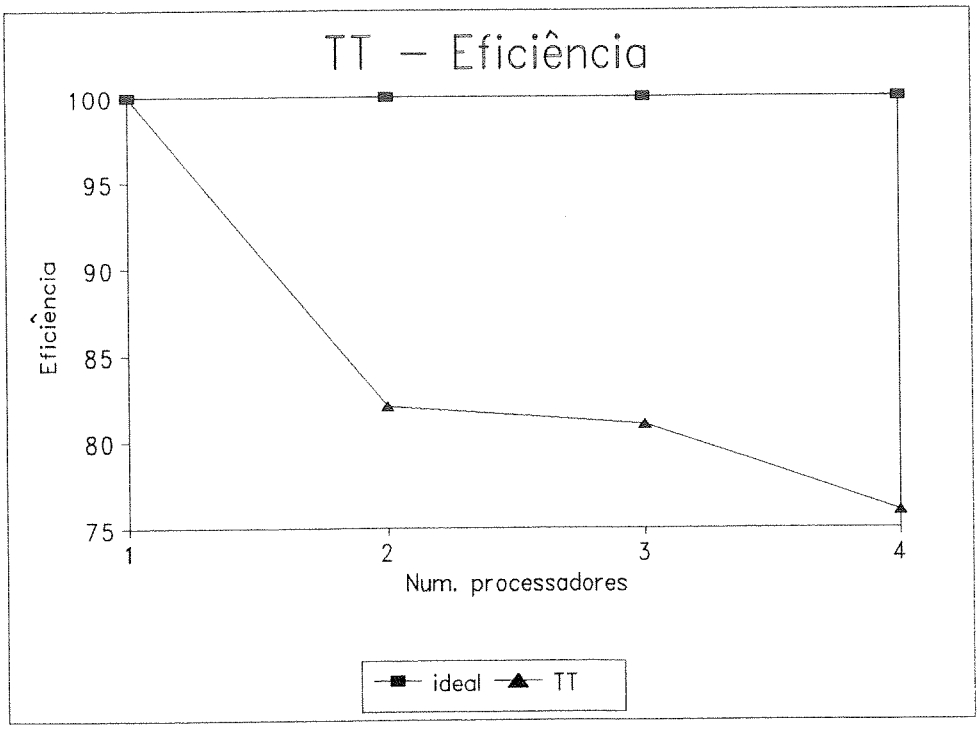
VR = tempo de cálculo do vetor de resíduos e

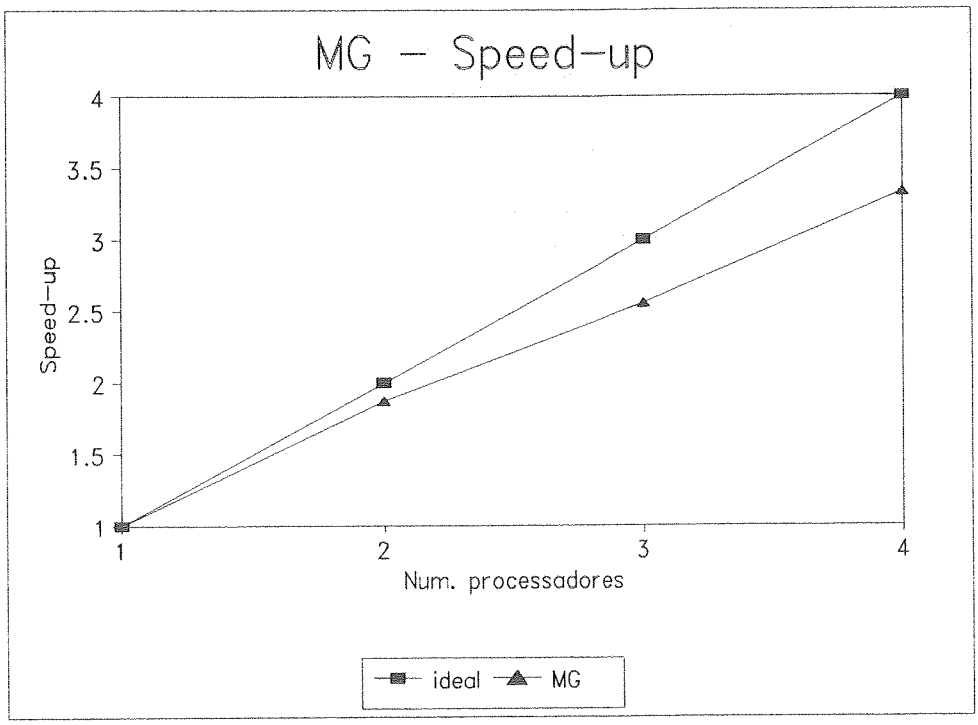
NE = tempo de cálculo da norma Euclidiana do vetor de resíduos.

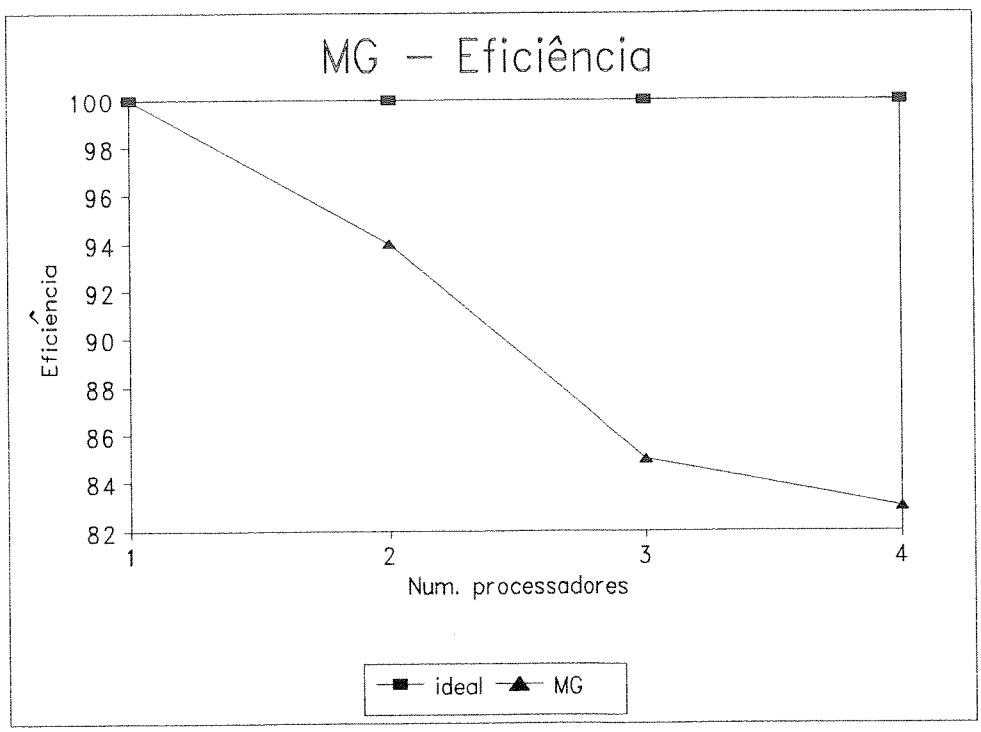
Como os tempos de execução de algumas etapas são extremamente reduzidos, foram feitas repetições das mesmas para se chegar ao tempo de um único processamento. Essas repetições também estão indicadas na tabela 6.1 (Rep). Os gráficos correspondentes de *speed-up* são apresentados após a tabela.

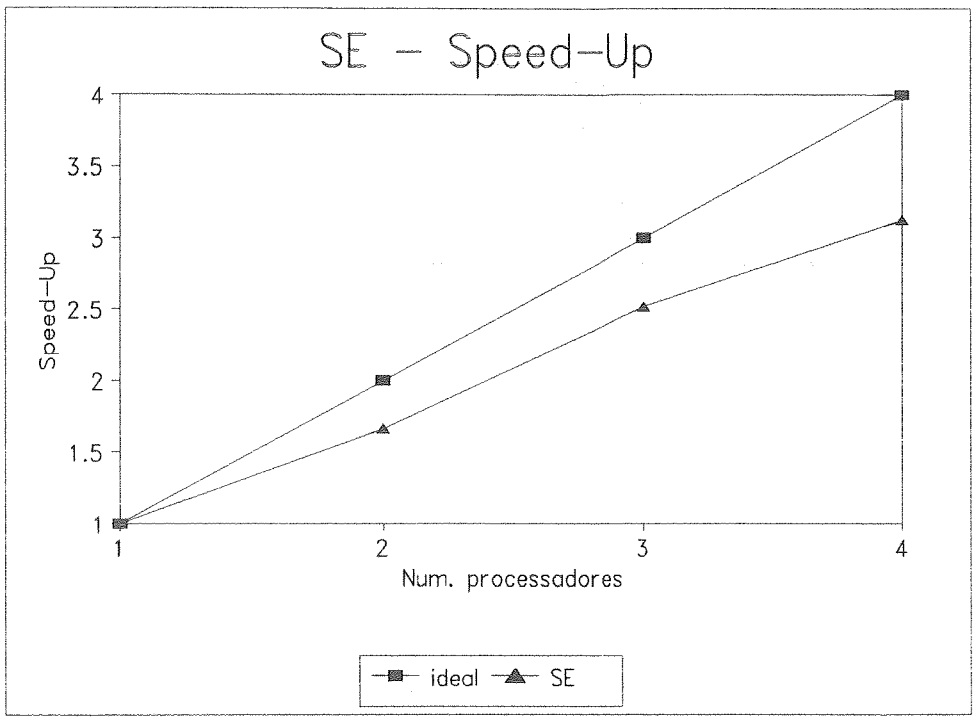
TAB 6.1	Número de processadores				
Refer.	1	2	3	4	Rep
TT	11260/ 1,00	6915/1,63	4625/2,43	3710/3,04	1
MG	0,28/ 1,00	0,15/1,87	0,11/2,55	0,084/3,33	500
SE	561/ 1,00	334/1,68	222/2,52	178/3,15	1
VR	0,0113/1,00	0,0058/1,94	0,0038/2,97	0,0029/3,90	6000
NE	7.2e-4/1,00	4.07e-4/1,77	3,00e-4/2,4	2,33e-4/3,09	150000

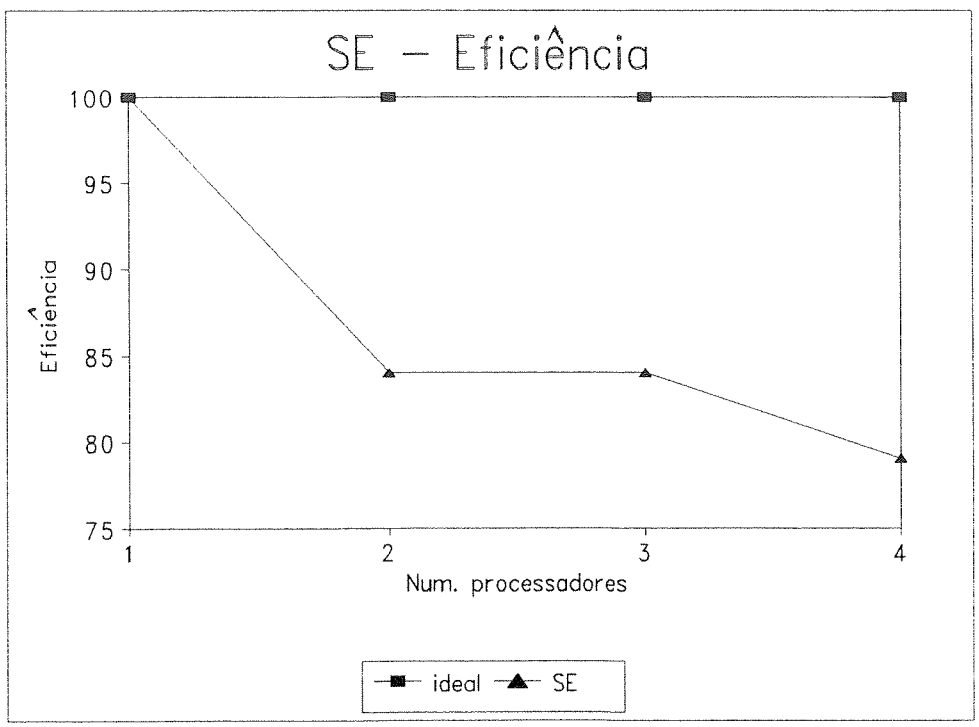

TT - Speed-up
Speed-Up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
TT

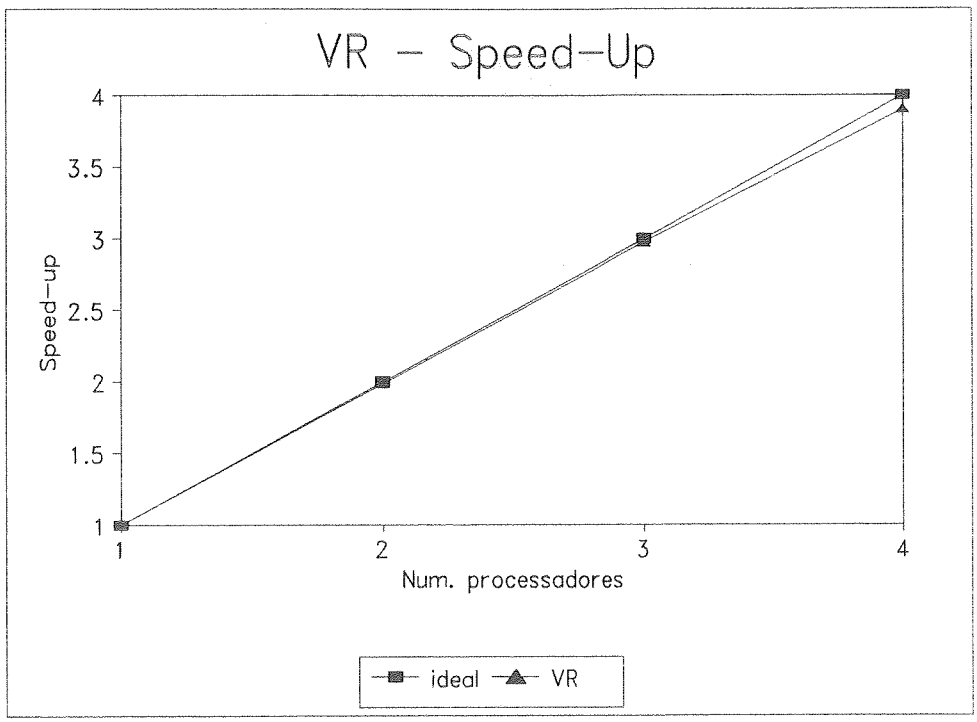

TT - Eficiência
Eficiência
100
95
90
85
80
75
1
2
3
4
Num. processadores
ideal
TT

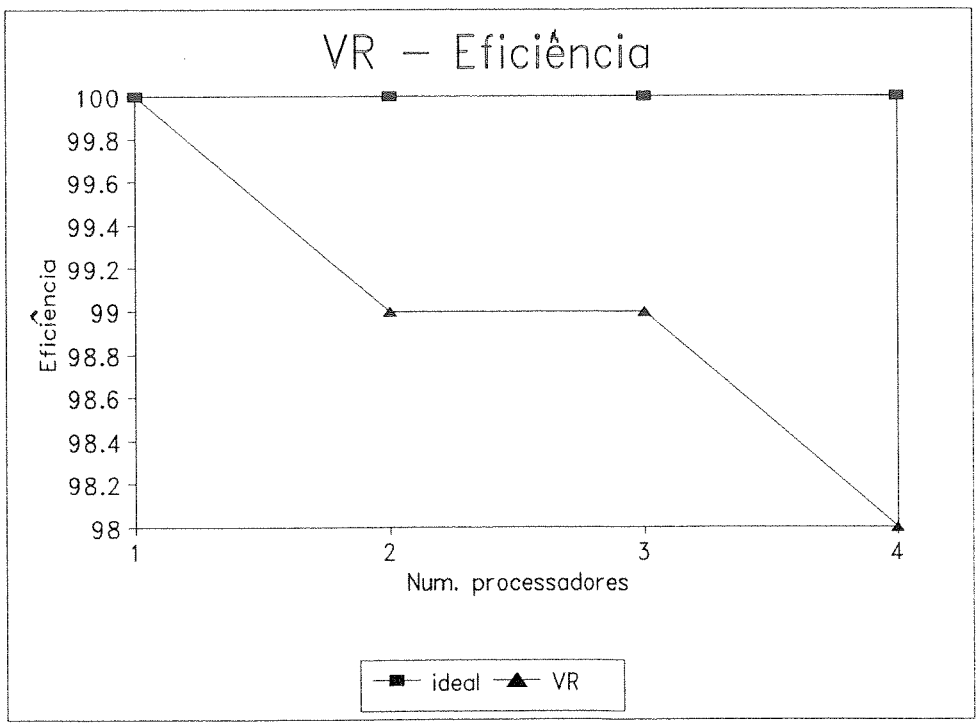

MG – Speed-up
Speed-up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
MG

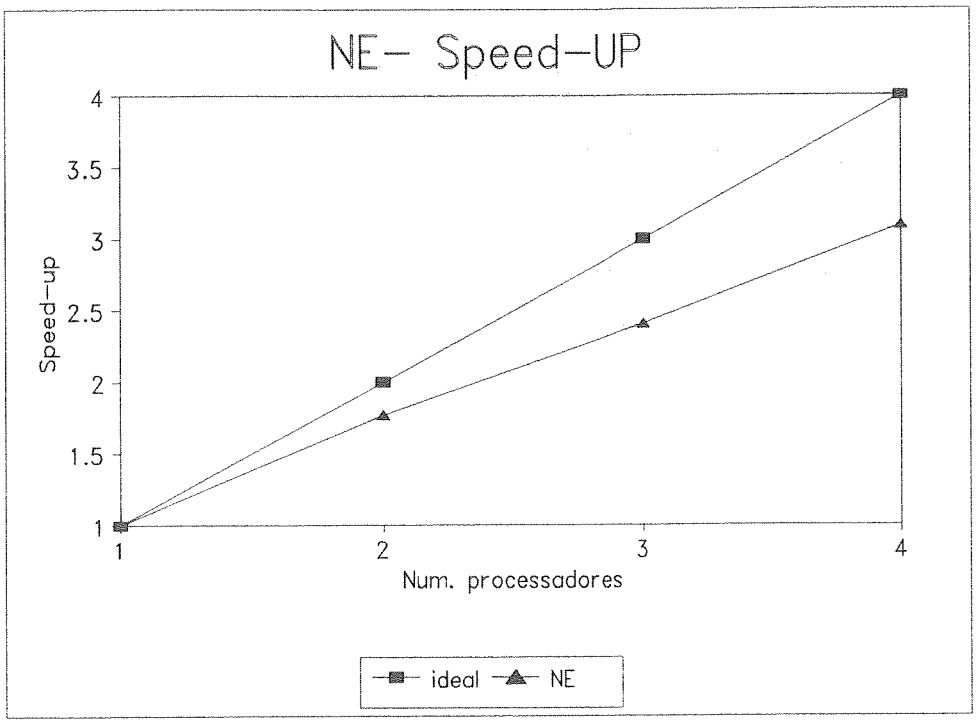

MG – Eficiência
Eficiência
100
98
96
94
92
90
88
86
84
82
1
2
3
4
Num. processadores
ideal
MG

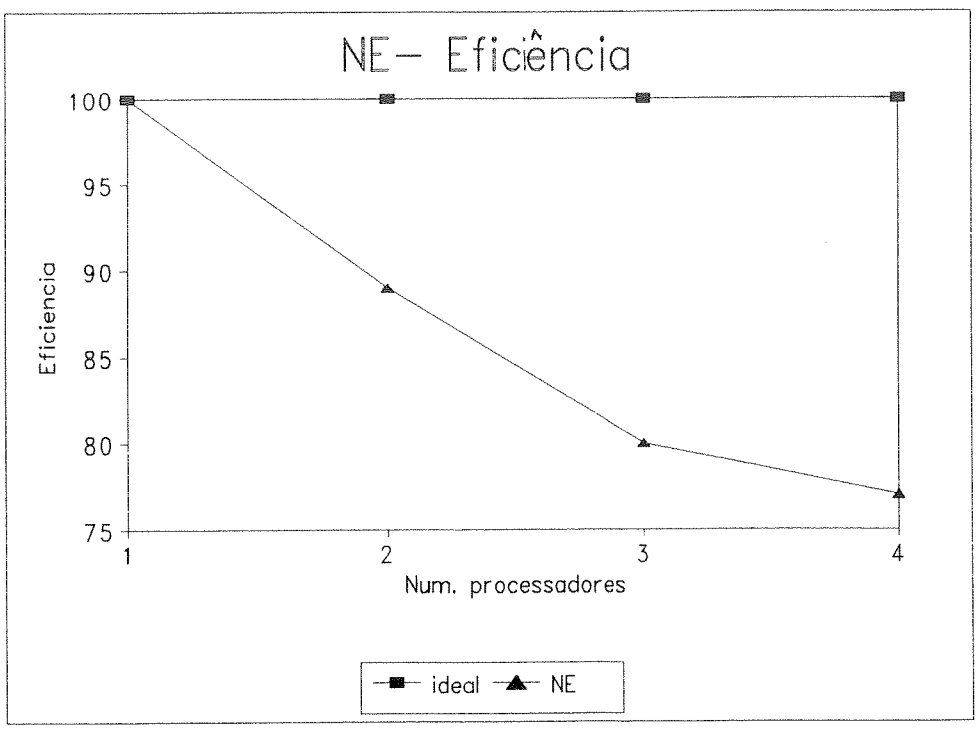

SE – Speed–Up
Speed–Up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
SE


SE – Eficiência
Eficiência
100
95
90
85
80
75
1
2
3
4
Num. processadores
ideal
SE


VR - Speed-Up
Speed-up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
VR


VR - Eficiência
Eficiência
100
99.8
99.6
99.4
99.2
99
98.8
98.6
98.4
98.2
98
1
2
3
4
Num. processadores
ideal
VR


NE- Speed-UP
Speed-up
4
3.5
3
2.5
2
1.5
1
1
2
3
4
Num. processadores
ideal
NE


NE- Eficiência
Eficiencia
100
95
90
85
80
75
1
2
3
4
Num. processadores
ideal
NE

6.6.2.4- ANÁLISE DOS RESULTADOS

Os resultados apresentados na tabela 6.1 mostram a notória predominância, em termos de esforço computacional, da etapa da solução do sistema de equações em relação às outras etapas. Esse fato se reflete na semelhança entre os gráficos de *speed-up* dessa etapa predominante (**SE**) e do gráfico correspondente ao tempo total de processamento (**TT**).

É também interessante observar-se a alta eficiência obtida na etapa de cálculo do vetor de resíduos (**VR**), indicando bom balanceamento de trabalho. Para a etapa de cálculo da norma Euclidiana do vetor de resíduos (**NE**) seria também esperado bom resultado, pois , em princípio, o balanceamento de trabalho é excelente, porém dado o pequeno esforço computacional da mesma, houve degradação do desempenho pelo custo computacional da montagem dos processos paralelos.

A etapa de montagem da matriz de rigidez (**MG**) apresentou desempenho semelhante ao exemplo "**M**" do capítulo 4, estando portanto de acordo com o esperado.

7- CONCLUSÕES

Se é quase inimaginável a utilização de métodos numéricos em análise estrutural sem o auxílio de computadores, fica também difícil de se imaginar o incremento da eficácia de tais métodos sem o correspondente incremento de desempenho dos computadores. Assim, no futuro próximo, se tornará imperativa a parceria entre métodos numéricos de análise e computadores de arquitetura paralela.

Hoje entretanto, a computação paralela nas diversas áreas nas quais a mesma é aplicável, ainda é uma ciência em nascimento, devido a dois fatos principais : a recente disponibilidade dos computadores paralelos e a multiplicidade de conceitos que envolve a computação paralela.

Particularmente no que se refere à análise estrutural, são relativamente escassos os trabalhos envolvendo computação paralela. Tal fato levou a se proporcionar a este trabalho um caráter introdutório, procurando focalizar aspectos básicos da computação paralela e peculiaridades da aplicação da mesma na análise de etapas típicas da aplicação do método dos elementos finitos em análise estrutural. As conclusões obtidas no estudo da paralelização dessas etapas típicas e na implementação do programa paralelo de análise não linear de treliças tridimensionais são apresentadas em seguida.

No estudo da montagem paralela da matriz de rigidez da estrutura, a abordagem nodal plena mostrou-se bastante adequada para a referida etapa, eliminando os complexos esquemas de sincronização inerentes às montagens paralelas oriundas do algoritmo sequencial usual. Cabe, no entanto, destacar-se a necessidade de uma formulação que gere eficientemente apenas as linhas da matriz

de rigidez dos elementos correspondentes a cada ponto nodal do mesmo, tarefa não elementar para matrizes de rigidez geradas a partir de integração numérica.

Na paralelização da solução do sistema de equações lineares, o algoritmo alternativo proposto, embora seja ainda passível de otimização, mostrou-se pouco superior ao usual em matrizes de grande largura de banda e um pouco inferior no caso de matrizes de pequena largura de banda. Essa proximidade de resultados entre os dois algoritmos aponta para uma suposta limitação do próprio algoritmo do qual ambas as soluções são baseadas : o método de eliminação de Gauss. Resultados superiores obtidos em máquinas MIMD de arquitetura baseada em memória local que aparecem em FREEMAN(1992) também sugerem um estudo sobre qual arranjo de memória é mais conveniente na utilização do método de eliminação de Gauss , ou ainda, dada a importância dessa etapa na análise estrutural, torna-se recomendável uma análise dos diversos métodos de solução (diretos e iterativos) em arquiteturas MIMD de memória compartilhada e distribuída.

A implementação do programa paralelo para análise não linear de treliças tridimensionais foi grandemente facilitada pela existência dos algoritmos paralelos já desenvolvidos para a montagem da matriz de rigidez da estrutura e para a solução do sistema de equações lineares. Os algoritmos que tratam de etapas típicas possuem duas qualidades apreciáveis : a generalidade, por se adaptarem em princípio a qualquer programa de análise estrutural e o desempenho, pelo fato de serem algoritmos já otimizados. Tais algoritmos contribuem também para a produtividade no desenvolvimento de software pois permitem ao programador se concentrar em seu trabalho específico, não tendo que se ocupar com o desenvolvimento de todas as etapas.

Finalmente, como sugestão complementar para prosseguimento de pesquisa, cabe ressaltar que considerável parte do tempo dispendido em uma análise estrutural via método dos elementos finitos concentra-se na etapa de entrada de dados e que o incremento de desempenho proporcionado pelos computadores paralelos pode ser beneficamente aplicado na referida etapa, não só diminuindo o tempo total da análise como também tornando mais agradável tal tarefa para o usuário.

ANEXO: Recursos computacionais utilizados

A1 Introdução

Neste anexo são apresentadas as características técnicas do computador utilizado, as peculiaridades da linguagem de computação usada na implementação dos programas e ainda o processo de obtenção do *speed-up* dos algoritmos.

A2 Sobre o computador

O computador utilizado foi o SILICON GRAPHICS modelo POWER 4S-440D de arquitetura MIMD com memória global, pertencente à Universidade Federal de São Carlos. A máquina contava com a seguinte configuração :

✓ Processadores : 4 processadores MIPS R3000 (40MHz) VLSI *floating point chip revision 4.0.*

✓ Memória *cache* para instruções : 64 KB

✓ Memória *cache* secundária : 1MB

✓ Memória principal : 64 MB

✓ Conexão entre processadores : barramento (*bus*)

✓ Sistema operacional : IRIX 5.2 (UNIX)

A3 Sobre a linguagem de programação

A linguagem de programação utilizada foi uma extensão de FORTRAN77 com possibilidade de criação de *loops* concorrentes. A definição de *loop* concorrente é feita pelo comando (imediatamente antes do loop) "C$DOACROSS". Como esse comando começa pela letra "C", a qual é indicação de comentário em FORTRAN, o programa fonte pode ser compilado também em máquinas sequenciais.

O comando "C$DOACROSS" possui vários parâmetros opcionais. Os principais são :

✓ LOCAL(a,b,c) : indica que as variáveis a,b e c são locais aos processadores.

✓ MP_SCHEDTYPE = tipo_de escalonamento : indica qual será o escalonamento utilizado nas iterações. Se o tipo_de_escalonamento for "SIMPLE", as iterações do *loop* serão simplesmente divididas entre os processadores, ou seja, ao primeiro processador caberá executar as "n" primeiras iterações e ao último, as "n" últimas, sendo "n" o resultado da divisão do número de iterações do *loop* pelo número de processadores. Se o tipo_de_escalonamento for "INTERLEAVE", o primeiro processador fará as iterações 1,NP+1,2NP+1, o segundo 2,NP+2,.. e etc, sendo "NP" o número de processadores. Opcionalmente, associada à opção "INTERLEAVE" pode ser usada a cláusula "CHUNK= c", fazendo com que o primeiro processador execute as iterações de 1 até "c", depois, de NP*c+1 até NP*c+c, ou seja, escalonamento intercalado de "pedaço "c". A definição do escalonamento é essencial para se promover um bom balanceamneto de trabalho. Na máquina utilizada, o pedaço (*chunk*) do escalonamento intercalado recomendado era 4 ou 8 para favorecer a utilização da memória *cache*.

✓ IF(condição) = Se a condição for verdadeira, o loop é executado concorrentemente, caso contrário é executado sequencialmente. Essa opção é interessante para se evitar a paralelização de loops de pouco esforço computacional, os quais podendo implicar em aumento do tempo de processamento devido ao custo computacional da paralelização.

Conforme apresentado, o modelo de paralelização dessa extensão de FORTRAN 77 é o *loop* concorrente. Embora seja a possível a utilização de alguns instrumentos de sincronização (barreiras, bloqueios e semáforos), tal procedimento é um tanto complexo, sendo aconselhável a utilização de outra linguagem, como o próprio "C", também disponível para a máquina.

A3 Obtenção de *Speed-Up*

A máquina utilizada operava como servidor de rede, prestando serviços como correio eletrônico, acesso à rede *Internet*, processamento de programas científicos e etc. Assim, estava sempre com um número mínimo de processos em operação e nunca totalmente disponível para que fossem efetuadas as medidas de tempo de

procesamento. A solução adotada foi efetuar as medidas quando o número e a complexidade dos processos em execução era mínima e monitorar esse fato através do comando "*top*" do sistema, o qual fornece a ocupação dos processadores.

Estando em uma situação favorável (poucos processos de pequeno esforço computacional), definia-se o número de processadores a ser utilizado com o comando "MP_NUMTHREADS=n" sendo "n" o número de processadores e executava-se o programa. Os tempos eram obtidos por inserções do comando "*time*" do sistema no código fonte dos programas. Para se chegar a medidas confiáveis foram feitos vários processamentos de cada programa em dias diferentes.

A ocupação dos processadores pelos processos interferentes mostrou-se bem reduzida nas monitorações efetuadas, validando os resultados obtidos, principalmente para processamento com até 3 processadores, quando a influência de pequenos processos é praticamente nula.

REFERÊNCIAS BIBLIOGRÁFICAS

ADELI, H; KAMAL O.(1992) *Concurrent analysis of large structures-I. Algorithms.* Computers & Structures Vol 42, No.3, p413-424.

ALMEIDA, V.A.F; ÁRABE, J.N.C. (1991). *Introdução à supercomputação.* São Paulo, LTC.

AMDAHL, G.(1967). *Validity of the single processor aproach to achieving large scale computing capabilities.* AFIPS Proceedings of the sprint joint computer conference, No.30, p438-485.

CHIEN, L.S; SUN, C.T(1989). *Parallel processing techniques for finite element analysis of nonlinear large truss structures.* Computers & Structures Vol 31, No.6, p 1023-1029.

CODENOTTI, B; LEONCINI, M..(1991). *Parallel complexity of linear system solution.* Singapore, World Scientific Publishing.

De_CEGAMA, A.L (1989). *Parallel processing architetures and VLSI hardware.*EUA, Prentice Hall.

DEKKER, T.J; HOFFMANN, W; POTMA, K(1994). *Parallel algorithms for solving large linear systems.* Journal of Computational and Applied Mathematics Vol 50 pp 221-232.

DIJKSTRA, E. W.(1968). *Cooperating sequential processes.* N. York, Academic Press.

FREEMAN, L; PHILIPS, C.(1992). *Parallel numerical algorithms.* Inglaterra, Prentice Hall International.

FLYNN, M. (1972). *Some computer organizations and their effectiveness.* IEEE Transactions, Vol 21.

HOCKNEY, R(1985). *(r_∞, $n_{1/2}$, $s_{1/2}$) Measurements on the 2-CPU Cray X-MP. Parallel Computing*, Vol 2, pp 1 14.

LAKSHMIVARAHAN, S; DHALL,S.K(1990). *Analysis and design os parallel algorithms.* EUA, McGraw Hill.

MATHIES, H; STRANG, G.(1979). *The solution of nonlinear finite element equations.* International Journal for numerical methods in engineering, Vol 14, pp 1613-1626.

OWEN, D.R,J; HINTON, E(1980). *Finite elements in plasticity: theory and pratice. U.K, Pineridge Press Limited*

QUINN, M.J(1987). *Designing efficient algorithms for parallel computers.* EUA, McGraw Hill Book Company.

REZENDE, M.N(1990). *Análise de pavimentos de edifícios pelo método dos elementos finitos em microcomputador.* Dissertação de Mestrado apresentada à Escola de Engenharia de São Carlos-USP, São Carlos.

RUBERT, J.B.(1991). *Estudo do desempenho de algoritmos numéricos na solução de sistemas não lineares de estruturas formadas por barras de treliça.* Dissertação de Mestrado apresentada à Escola de Engenharia de São Carlos-USP, São Carlos.

SORIANO, H..L(1981). *Sistemas de equações lineares em problemas estruturais.* Laboratório Nacional de Lisboa (apostila), Lisboa.

STONE, H.(1987). *High-performance computer architeture*, EUA, Addison-Wesley.

von NEUMANN, J.(1945). *First draft of report on the EDVAC*. Technical Report. EUA,University of Pennsylvania.

ZIENKIEWICZ, O.C.(1967). *The finite element method*. EUA, McGraw Hill.