UMA ADAPTAÇÃO DO MEF PARA ANÁLISE EM MULTICOMPUTADORES: APLICAÇÕES EM ALGUNS MODELOS ESTRUTURAIS

Engo. VALÉRIO DA SILVA ALMEIDA

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Estruturas.

ORIENTADOR: Prof. Assoc. JOÃO BATISTA DE PAIVA

São Carlos
1999

Conhecimento
é sede depois
de bem ter-se
bebido.

À minha família
que compreende a
minha ausência.

AGRADECIMENTOS

Sobretudo, a DEUS por ter me dado saúde e me feito forte para enfrentar os obstáculos que surgiram em toda a minha vida.

Ao meu companheiro e compreensivo orientador Prof. João Batista de Paiva pela atenção e dedicação dada a mim e ao meu trabalho em todos os momentos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e a Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP – por terem me concedido a bolsa de estudo.

A todos os companheiros que conquistei neste período, em especial a Carlos Humberto Martins e a Jeselay H. Cordeiro dos Reis.

Ao amigo tão passageiro Yansheng Jiang que além da amizade me ajudou no início de meu trabalho.

Ao Prof. José Alberto Cuminato que teve paciência comigo. Ao Prof. José Elias Laier que, acima de tudo, contribuiu à minha formação humana.

Aos funcionários do centro de processamento de dados da USP de São Carlos –CISC – em especial ao administrador da rede do SP2 Rogério Toshiaki Kondo pelas infindáveis dúvidas esclarecidas.

A Ariadine Schifino Lechner pela sua dedicação e pelo seu carinho dado a mim.

A todos do Departamento de Estruturas da EESC/USP, pela total infra-estrutura concedida.

Enfim, felizmente, inúmeras são as pessoas que deveriam ser citadas que aqui não caberiam. Ficam tácitas neste texto e, sobretudo, concretas em minhas demonstrações de agradecimentos.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ALU: Arithmetical Logic Unit

ASMGJ: Asynchronous Systolic Multiple Gauss Jordan

Bi- CG: Bi-Conjugate-Gradients

Bi-CGSTAB: Bi-Conjugate-Gradients Stabilised

CISC: Centro de Informática de São Carlos

CPU: Central Processing Unit

CST: Constant Strain Triangle

diag: diagonal

DKT: Discrete Kirchhoff Theory

Flops: Flow Point Operation Per Second

GC: Gradientes Conjugados

HPF: High Performance Fortran

IC: Incomplete Cholesky

ICCG: Incomplete Cholesky Conjugate-Gradient

I/O: IN and OUT

lim: limite

MEC: Método dos Elementos de Contorno

MEF: Método dos Elementos Finitos

MIMD: Multiple Instruction Multiple Data

MISD: Multiple Instruction Single Data

MPI: Message Passing Interface

ns: nanossegundo

PIM: Parallel Iterative Methods

POLY: Polynomial

PTV: Princípio dos Trabalhos Virtuais

PVM: Parallel Virtual Machine

SP: Scalable PowerParallel

SIMD: Single Instruction Multiple Data

SISD: Single Instruction Single Data

RESUMO

ALMEIDA, V. S. (1999). *Uma adaptação do MEF para análise em multicomputadores: aplicações em alguns modelos estruturais.* São Carlos, 1999. 126p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Neste trabalho, apresenta-se uma adaptação dos procedimentos utilizados nos códigos computacionais seqüenciais advindos do MEF, para utilizá-los em multicomputadores. Desenvolve-se uma rotina para a montagem do sistema linear particionado entre os diversos processadores. Resolve-se o sistema de equações lineares geradas mediante a rotina do PIM *(Parallel Iterative Method)*. São feitas adaptações deste pacote para se aproveitar as características comuns do sistema linear gerado pelo MEF: esparsidade e simetria. A técnica de resolução do sistema em paralelo é otimizada com o uso de dois tipos de pré-condicionadores: a decomposição incompleta de Cholesky (IC) generalizado e o POLY(0) ou Jacobi. É feita uma aplicação para a solução de pavimento com o algoritmo-base totalmente paralelizado. Também é avaliada a solução do sistema de equações de uma treliça. Mostram-se resultados de *speed-up*, de eficiência e de tempo para estes dois modelos estruturais. Além disso, é feito um estudo em processamento seqüencial da performance dos pré-condicionadores genéricos (IC) e do advindo de uma série truncada de *Neumann*, também generalizada, utilizando-se modelos estruturais de placa e chapa.

Palavras-chave: multicomputadores, método dos elementos finitos, processamento paralelo, pré-condicionadores, método dos gradientes conjugados.

ABSTRACT

ALMEIDA, V. S. (1999). *Multicomputer finite element method analysis of usual structures models*. São Carlos, 1999. 126p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

This work presents an adaptation of conventional finite element method (FEM) computing procedures to multicomputers. It is presented the procedure which the linear system of equations is split among the processor and its solution. It was improved a public software called PIM (Parallel Iterative Method) is used to solve this system of equations. These improvements explore efficiently the usual features of the FEM systems of equations: sparseness and symmetry. To improve the solution of the system, two different preconditioners are used: a generic Incomplete Cholesky (IC) and the Polynomial preconditioning (POLY(0) or Jacobi). It is carried out a full adaptation of the method to parallel computing with a program developed to analyse floor structures. The improved PIM is also used to solve the system of equations of a tri-dimensional truss. It is presented the speed-up, the efficiency and the time used in the resolution of the systems of equations for the floor and for the truss. It is also presented a study of performance in sequential processing of the generic (IC) and the generic Neumann series preconditioners in the analysis of plates in bending and in plane actions.

Keywords: multicomputers, finite element method, parallel processing, preconditioners, conjugate-gradients method.

Capítulo 1

INTRODUÇÃO

1.1 MECÂNICA DAS ESTRUTURAS E O PROCESSAMENTO EM PARALELO

A necessidade humana de interpretar, quantificar e qualificar o comportamento de tudo que está a sua volta é algo que o instiga desde os tempos mais remotos. Porém, nem sempre é possível quantificar os comportamentos de forma exata em função da complexidade do tratamento que deve ser levado em consideração para sua análise. Assim, metodologias simplificadas sempre foram adotadas para facilitar tais interpretações.

Um exemplo é o cálculo do comprimento da circunferência, que na antigüidade era objeto de interesse e fora feito primeiramente, mediante uma aproximação de polígonos inscritos com um número muito grande de lados, aproximando-se de uma circunferência.

Em ROCHA (s.d.), é apresentado o desenvolvimento do método da exaustão, que é o cálculo da área da região formada por uma curva qualquer definida com extremos em dois pontos, com continuidade garantida entre eles, e o eixo horizontal. Assim, a área é obtida pela divisão da região em pequenos retângulos.

Com estes exemplos citados, dentre outros, o uso de meios discretos para simular os comportamentos dos meios contínuos sempre foram usados pelos estudiosos para simplificar sua análise.

Como abordagem direcionada ao estudo do comportamento dos sólidos, na engenharia houve - e haverá - o interesse em simular os movimentos, as forças e/ou o

fluxo que aparecem nos corpos da natureza. Desta forma, mediante uma modelagem matemática e levando-se em consideração o seu estado de equilíbrio inicial, manutenção de continuidade e compatibilidade, sempre é possível explicitar um modelo matemático que representasse este estado do corpo. Tal tipo de equação, dependendo da abordagem que é feita, pode ser uma equação diferencial ou integral, mas estas equações geradas, em geral, não possuem respostas exatas, e caso haja, muitas são de difícil obtenção.

Contudo, uma forma de se determinar os movimentos dos corpos é a utilização dos processos discretos, que nada mais são do que divisões do seu domínio em partes discretas, e as equações são aproximadas por outras mais simples, mas, satisfazendo as condições de cada problema.

Dessa forma, com os conhecimentos da mecânica dos sólidos, de seus fenômenos que são modelados física e matematicamente, se desenvolveram alguns métodos os quais representam os meios contínuos por simplificados meios discretos. Em MCHENRY[1], HRENIKOFF[2] e NEWMARK[3] *apud* ZIENKIEWICZ (1977) são feitos os primeiros trabalhos neste sentido, comentando-se que razoáveis soluções para um meio contínuo podem ser obtidas por substituição em pequenas porções do contínuo por arranjos de simples barras elásticas.

Mais tarde, ARGYRIS (1960) fez um estudo dos princípios de energia (método das forças e dos deslocamentos) aplicados às estruturas das asas das aeronaves, sendo estas discretizadas em elementos de barra e a posterior manipulação da solução do sistema gerado é feita com o uso de um computador digital. Por fim, compara-se os dois métodos - de energia e energia complementar que foram implementados - com as soluções exatas.

---

[1] McHENRY, D. (1943). A lattice analogy for the solution of plane stress problems. J. *Inst. Civ. Eng.*, v. 21, p. 59-82.

[2] HRENIKOFF, A. (1941). Solution of problems in elasticity by the framework method. J. Appl. *Mech.*, A8, p. 169-144.

[3] NEWMARK, N. M. (1949). Numerical methods of analysis in bars, plates and elastic bodies. *Numerical Mech. Anal. Eng.*, (ed. L.E. Grinter), Macmillan.

Em seu trabalho, ARGYRIS (1960) utiliza um modelo com simples barras, simulando uma estrutura tridimensional. Mas, sabe-se que para simular comportamentos estruturais mais realísticos da mecânica dos sólidos, deve-se desenvolver modelos matemáticos mais gerais e ter ferramentas adequadas à sua realização.

Como método discreto bastante empregado na mecânica dos sólidos, o método dos elementos finitos (MEF) tem-se desenvolvido e principalmente tem sido implementado, desde a sua criação nominal (TURNER *et al.* (1956)), a partir do avanço da tecnologia dos computadores.

O método em si se baseia no emprego de equações polinomiais de fácil tratamento que satisfaçam as condições mecânicas e geométricas do domínio de um determinado problema contínuo. Como o equacionamento e a determinação dos parâmetros incógnitos advindos do MEF geram um número muito grande de variáveis e, principalmente, uma grande quantidade de operações que demandam considerável tempo computacional, geralmente são empregados modestos modelos na resolução dos problemas, os quais possam ser operacionalizados nos computadores convencionais.

Em contrapartida, as modelagens mais sofisticadas e complexas deste método só se tornaram exeqüíveis graças ao avanço dos computadores, que possibilitaram a manipulação eficiente de complexas instruções e da armazenagem de um grande número de dados.

Assim, o crescente uso do método dos elementos finitos, mediante modelos cada vez mais sofisticados, depende diretamente do aprimoramento dos computadores e, nas últimas cinco décadas, estudos ligados ao seu aperfeiçoamento e impacto que estas máquinas têm causado no mundo científico é algo notório.

Conforme NOOR (1997), o avanço tecnológico dos computadores pode ser dividido em cinco gerações, sendo que estas gerações têm aumentado a velocidade para mais do que um trilhão de vezes durante as últimas cinco décadas, enquanto tem-se reduzido o custo. Estas gerações têm características diferenciadas principalmente pelos componentes que as constituem . Assim, tem-se:

- *Primeira geração (1939-1950):* caracterizou o surgimento dos computadores com blocos de *hardware* de tubos de vácuo;
- *Segunda geração (1950-1960):* introduziu o uso de blocos de diodos discretos e transistores;
- *Terceira geração (1960-meados de 1975):* marcou o uso de blocos de circuitos integrados (pequenas e médias escalas);
- *Quarta geração (meados de 1975-1990):* implementou a utilização de circuitos integrados de grande e muito grande escala;
- *Quinta e última geração (1990-presente):* caracterizou o uso de *chips* com ultra grande escala de componentes integrados e microprocessadores poderosos.

A partir do início da década de 60, algoritmos numéricos começaram a ser desenvolvidos, de forma a serem aplicados não somente em máquinas de arquiteturas seqüenciais, mas também em paralelas, já que tais computadores poderiam oferecer maior recursos em termos de velocidade e de tamanho de memória.

Neste sentido, podemos citar os trabalhos de CARROLL & WETHERALD (1967) e LEHMAN (1966), que fizeram um estudo inicial do uso destas técnicas para tais máquinas. Mas foi na metade da década de 70 que trabalhos mais aplicados a engenharia foram desenvolvidos, como o estudo pioneiro realizado por MIRANKER (1971), sobre algoritmos na análise numérica em computação paralela, estudos estes que eram os de otimização matemática, encontro de raízes, equações diferenciais e solução de sistemas lineares.

Assim, após este trabalho, o avanço nas aplicações, mediante o método dos elementos finitos em processamento paralelo, começou a ser feito naturalmente, explorando tais algoritmos ou readaptando-os, de forma a resolver seus problemas intrínsecos. Citam-se os trabalhos de NOOR & HARTLEY (1978), que avaliaram a montagem da matriz de rigidez e o de NOOR & LAMBIOTTE (1979), que analisaram as respostas dinâmicas com o método dos elementos finitos.

A partir da metade da década de 80, as implementações em computação paralela começaram a ser realizadas de maneira volumosa, como por exemplo, o desenvolvimento de um pseudocódigo realizado por ZOIS (1988a e 1988b), que

desenvolveu um programa computacional chamado de PARFERS. Este programa fazia uma análise estrutural em MEF, avaliava a forma de montagem da matriz de rigidez, dos carregamentos e a solução do sistema linear aplicado a um tipo de paradigma computacional: o MIMD. A montagem da matriz era feita particionando-se os dados entre os processos, mas não era utilizada a simetria para os dados que estão em memórias diferentes. Assim, os subsistemas de cada processo não aproveitavam a simetria e a esparsidade dos problemas intrínsecos da análise, que fora feita em teoria estática linear. Para resolver o sistema, utilizou-se de um método direto chamado de "Asynchronous Systolic Multiple Gauss Jordan" (ASMGJ), que é uma variante do método de Gauss.

Técnicas para otimizar as aplicações dos modelos de MEF começaram a ser objeto de interesse para melhorar a potencialidade oferecida pelos computadores de natureza paralela. Por exemplo, uma técnica precursora da utilização do tratamento da estrutura em subdomínio para computação paralela foi o trabalho de LAW (1986), onde é apresentado o método dos elementos finitos e não se constrói a matriz de rigidez global da estrutura, e sim atualiza-se os valores de cada processador vizinho até encontrar um erro menor que o estimado. A estratégia da solução proposta pode ser estendida para subestruturação, onde cada subestrutura é tomada como um superelemento e acessado por um processador. Comenta-se, finalmente, que a implementação para problemas não-lineares é muito simples de se realizar.

Uma outra forma de otimização em tais computadores é a solução do sistema, que é uma das etapas mais importantes de todo o processo e que despende mais tempo para conclusão. A resolução do sistema linear empregada comumente é com o uso de métodos diretos (Gauss, Cholesky). Entretanto, nos últimos quinze anos, têm-se despontado o uso de métodos iterativos que, como maior vantagem, oferecem menos dependência entre os dados dos diferentes processadores.

Assim, BARRAGY & CAREY (1988), fizeram um estudo o qual foi desenvolvido para computadores de memória compartilhada, onde a solução do sistema é feita com o método dos gradientes conjugados e a parte mais relevante - para processamento paralelo - do método é o produto matriz-vetor, onde para cada elemento é gerada sua matriz local/global. Segundo eles, o seu trabalho é altamente

paralelizável para este tipo de arquitetura e o pré-condicionador usado (Jacobi) não é o melhor que se dispunha, mas é de fácil paralelização.

O grande problema gerado com o emprego de um método iterativo é a respeito do condicionamento da matriz de rigidez, que leva a uma demora na convergência para os deslocamentos. De sorte, uma outra maneira de otimizar a resolução do sistema linear através de métodos iterativos é acelerar a convergência de sua resposta, utilizando um pré-condicionador.

O trabalho de NOOR & PETERS (1989) apresenta-se um estudo de análise dinâmica não-linear para abordagem em multiprocessadores, por meio de estratégias de subestruturação com o método dos gradientes conjugados, usando pré-condicionadores. A implementação foi feita em um computador do tipo CRAY X-MP/4 e resultados de *speed-up* e tempo de CPU são obtidos para alguns casos.

Já no final da década de 80, surgia o interesse em utilizar as técnicas da aplicação de inteligência artificial e, como estudo introdutório, GROSSO & RIGHETTI (1988) abordaram estas técnicas e sugeriram um possível caminho para resolver os problemas do MEF. Constrói um meta-algoritmo como simples exemplo e conclui-se que este tipo de algoritmo tem vantagens para serem paralelizáveis.

A partir da década de 90, já se pensava em aplicar todas estas técnicas de otimização juntas, as quais foram apresentadas anteriormente para resolução de problemas estruturais. Assim, alguns trabalhos implementados com a junção destes processos de otimização são descritos em seguida.

JOHNSSON & MATHUR (1990), descreveram uma formulação do método dos elementos finitos com a discretização da estrutura em domínios, onde cada domínio está em um processador. Desta forma, evita-se a montagem da matriz de rigidez global do sistema e a resolução do sistema linear é feita concorrentemente em cada domínio. Além disso, atualiza-se globalmente *apenas* o vetor resíduo, via método dos gradientes conjugados com acelerador de convergência (pré-condicionador).

Segundo eles, as malhas regulares geram comunicação apenas entre processos vizinhos, e o método dos gradientes conjugados para solução do sistema é altamente concorrente quando há bom balanceamento de cargas, e que o produto matriz-vetor é

feito no próprio domínio, sem comunicação. Implementa-se, enfim, com uma estrutura tridimensional e avalia-se os tempos de processamento e os tempos de pico para a montagem do sistema de rigidez do elemento e para a resolução do sistema neste supercomputador, o qual utiliza o modelo de arquitetura CM-2.

Em ADELI & KAMAL (1992) é analisado o uso da técnica de subestruturação para análise de barras. Para cada subdomínio é feita a condensação estática lançando-se sua contribuição para os nós da interface dos domínios, e então calcula-se este sistema voltando para cada subdomínio e obtendo seus deslocamentos internos. Usou-se o computador com arquitetura de memória compartilhada e uma aplicação para um pórtico plano é apresentado.

Com o emprego da técnica de decomposição em domínio, é de fundamental importância otimizar a forma de como discretizar a malha (*grid*), pois tem que garantir uma divisão eqüitativa dos dados entre os diversos processadores, evitando-se ociosidade no sistema. Assim, nos trabalhos de FARHAT & ROUX (1991) e de SAXENA & PERUCCHIO (1992) foram tratados este assuntos e apresentam um algoritmo para dividir a malha de maneira equilibrada.

Para o primeiro trabalho - FARHAT & ROUX (1991) – tem-se o estudo da decomposição de domínios implementados em computadores de memória local, apresentando-se a melhor maneira de dividir a malha para reduzir os tempos de trocas de mensagens. Aplicações a problemas estruturais em análise estática são feitas.

Em SAXENA & PERUCCHIO (1992) um trabalho de decomposição automática em domínio da malha de um sólido é tratado. Pseudocódigos e uma maneira ótima de distribuir esta decomposição são mostrados com suas características intrínsecas de paralelização, até então não alcançado anteriormente.

Com o uso cada vez maior dos métodos iterativos para solução do sistema linear, muitos pesquisadores continuaram a desenvolver técnicas de resolução de sistemas via métodos diretos em paralelo, podendo ser citado, na década de 90, o trabalho de RAO *et al.* (1993), o qual apresenta um estudo comparativo de eficiência entre os tipos de armazenagem de matriz: em banda e do tipo *skyline*. Mostra-se que, a eficiência de ambos, sendo resolvidos pelo método direto de Gauss, é muito

próxima, e tem também um pseudocódigo para resolução do sistema tanto para computadores de memória compartilhada como para memória distribuída.

O trabalho de LAW & MACKAY (1993) abordam resolução do sistema linear em computador de memória distribuída, aproveitando a esparsidade da matriz com a estratégia de se armazenar em banda e resolvendo o sistema pela decomposição LDL^T. Um exemplo de um avião (com 4.000.000 graus de liberdade) e os tempos de processamentos são mostrados para a montagem dos fatores L e D, para a substituição direta e para a retrosubstituição.

Nos últimos cinco anos vários tipos de pré-condicionadores têm sido objeto de interesse para solução dos problemas via método dos gradientes conjugados. Na literatura, uma gama muito grande de pré-condicionadores têm sido analisados e implementados para a abordagem de problemas estruturais.

Assim, trabalhos como o de SCHMIT & LAI (1994) apresentam um eficiente modo de resolver o sistema de equações através do estudo de três tipos de pré-condicionadores (decomposição incompleta de Cholesky, polinomiais e fatorização elemento por elemento), assim como o estudo de otimização para melhor aproximar o vetor inicial. São apresentados também exemplos de treliças espaciais com diferentes pré-condicionadores. Segundo eles, o uso do método do gradiente conjugado com o adequado pré-condicionador e a correta otimização em computação paralela, em conjunto, significam uma atrativa e eficiente forma de resolver problemas de otimização estrutural.

O trabalho de BITZARAKIS *et al.* (1997) apresenta três técnicas de subdomínio, onde os nós internos e as interfaces são calculados com o método dos gradientes conjugados, com pré-condicionador do tipo polinomial: uma série truncada de Neumann. Exemplos com vigas em balanço, cascas cilíndricas e placas são apresentados com a eficiência das iterações alcançadas e os tempos de CPU.

O uso de computadores do tipo multiprocessadores é ainda de domínio restrito para centros de computação de universidades e empresas desta área computacional. Assim, com a necessidade crescente do uso deste tipo de paradigma computacional em quase todas as áreas científicas, tem-se desenvolvido sub-rotinas que possam ser implementadas nos programas convencionais, através dos quais

possam ser aproveitadas as vantagens oferecidas nos programas para uso em computação paralela. Estas vantagens são trocas de mensagens, transmissão global de dados e sincronização dos vários processadores em pontos específicos do programa.

Os pacotes mais comuns para se realizar as trocas de mensagens são o PVM, ver GEIST *et al.* (1994) e MPI, ver IBM (1995), que são, respectivamente, abreviações de *Parallel Virtual Machine e Message Passing Interface*, os quais podem ser usados em *workstation*, ou seja, em conjuntos de computadores, até mesmo pessoais do tipo Pentium, que estejam ligados em rede. Isto abre uma grande possibilidade de pequenos centros computacionais estudarem e/ou readaptarem seus pseudocódigos para este tipo de paradigma a um custo baixo.

Como trabalho usando uma destas sub-rotinas, pode ser citado o artigo de CHEN & BYREDDY (1997), que aborda a paralelização em computadores de memória distribuída, utilizando as sub-rotinas de trocas de mensagens chamada de PVM. Dois tipos de abordagens são feitas com os modelos existentes para esta troca: o usado modelo mestre-escravo e o modelo de mestre-escravo em multinível. Implementa-se estes tipos de procedimentos na análise de placas do modelo de Reissner-Mindlin, mediante o método das faixas finitas. Conclui-se, então, que o tratamento da programação através do modelo de mestre-escravo em multinível é mais eficiente.

1.2 OBJETIVO

Tendo em vista a crescente necessidade de se estudar modelos estruturais cada vez mais complexos e que demandem grande quantidade de memória e tempo computacional, é necessário o desenvolvimento de ferramentas que suportem estas análises, já que as máquinas de processamento existentes têm demostrado que não conseguem acompanhar a taxa de crescimento no tratamento de modelos complexos da mecânica dos sólidos.

Desta forma, a utilização de máquinas para processamento paralelo é a opção para a possível aplicação de sistemas estruturais mais sofisticados em análise

numérica via, por exemplo, método dos elementos finitos (MEF). No item 1.1, percebe-se que a "migração" do MEF para este novo tipo de arquitetura já tem sido feita desde a metade da década de 70, MIRANKER (1971). No Brasil, este campo tem sido pouco explorado nos centros de pesquisas ligados a resolução de problemas da engenharia estrutural, mostrando uma carência de estudo e de aplicação nesta área.

Assim, o trabalho aqui apresentado teve o intuito de mostrar uma forma para resolução de um problema estrutural analisado via MEF em sistema de computadores de memória distribuída, também conhecido por multicomputadores. Com isso, o trabalho foi desenvolvido alterando-se as características intrínsecas do código, ou seja, dividindo-se todas as etapas do algoritmo dentre os vários processadores (nós), sendo que todos os procedimentos em paralelo foram desenvolvidos para funcionarem com qualquer número de nós. Destaca-se que deu-se ênfase à montagem e a resolução do sistema linear para análise em paralelo.

Para o primeiro procedimento, utilizou-se a técnica desenvolvida em REZENDE (1995). Esta técnica consiste em se montar a matriz de rigidez linha a linha e não da forma convencional: elemento por elemento. Demonstrando a extrema eficiência com o uso deste procedimento para a abordagem em computadores de memória distribuída.

A resolução do sistema linear já não apresenta vantagens quanto a sua paralelização como no procedimento anterior. Assim, tem sido objeto de estudo a busca de metodologias cada vez mais eficientes na sua otimização em arquitetura paralela.

Hoje em dia a técnica dos Gradientes Conjugados, com o uso de um adequado pré-condicionador, tem se destacado na aplicação da resolução do sistema linear em paralelo. Por isso, este trabalho pretendeu utilizar desta técnica com emprego do pré-condicionador denominado decomposição incompleta de Cholesky (IC), que aqui fora generalizado para a influência de qualquer combinação de espectros das colunas não negligenciadas.

Apresenta-se, também, um estudo numérico em processamento seqüencial de convergência com aplicação de dois pré-condicionadores : IC e um advindo de uma série truncada de Neumann, também generalizada. Apresentou-se, então, resultados

de convergência com o uso dos dois pré-condicionadores, aplicados a três modelos estruturais: placa, chapa e pavimento.

Para aplicação em paralelo, o código totalmente paralelizado é de um modelo de pavimento. Também foi adaptado para a resolução do sistema em paralelo o código advindo do modelo de treliça tridimensional. Tomou-se medidas de desempenho de *speed-up*, eficiência e tempo de CPU das várias etapas em paralelo, variando-se os números de equações de cada modelo estrutural e o número de processadores do sistema utilizado.

O computador que foi usado para a realização deste trabalho é o IBM 9900/SP, que possui três nós com 256 Mbytes em cada um (ver TUTORIAL do IBM SP (1998)). O sistema de troca de mensagens adotado é o PVM desenvolvido pela IBM, que é denominado de PVMe (ver TUTORIAIS do CISC (1998)).

Deixa-se esclarecido que este trabalho é o precedente para o entendimento das dificuldades e dos possíveis “gargalos” encontrados para a paralelização do MEF em multicomputadores neste centro de pesquisa.

1.3 ORGANIZAÇÃO DO TRABALHO

No capítulo 1 apresenta-se uma contextualização do que se tem feito na área numérica aplicada ao MEF com o uso do processamento paralelo. Assim, apresenta-se um sucinto estad
o da arte sobre o desenvolvimento do método dos elementos finitos, junto com o avanço de máquinas computacionais que pudessem executar estes procedimentos inerentes ao método dos elementos finitos (MEF).

No segundo capítulo, faz-se um estudo resumido dos computadores de alto desempenho. Inicialmente, são abordadas as técnicas mais utilizadas para se melhorar a performance na realização dos processamentos. Em seguida, apresenta-se o porquê da necessidade de se utilizar computadores com vários processadores, dando-se ênfase à arquitetura de memória distribuída e às suas limitações.

No capítulo 3 comenta-se brevemente o método dos elementos finitos, dando-se maior atenção às técnicas de otimização do método. Assim, é apresentada a

técnica da subestruturação, que também é aplicada para processamento em série. Em seguida, a técnica da decomposição em domínio, bastante empregada na análise em paralelo e a terceira forma, realizada neste trabalho - que é o simples particionamento dos dados entre os diversos processadores.

No capítulo 4, apresenta-se os métodos de resolução do sistema linear em paralelo, comentando-se brevemente as vantagens e desvantagens do procedimento direto e iterativo. Em seguida, é apresentado um estudo do método dos gradientes conjugados, e assim, aborda-se o pacote PIM utilizado para resolver o sistema em paralelo. Então, mostra-se os procedimentos adotados para implementá-lo.

No capítulo 5 relata-se o estudo de pré-condicionadores, apresentando duas técnicas de pré-condicionamento: decomposição incompleta de Cholesky (IC) e a advinda de uma séria truncada de Neumann (POLY). Mostra-se, em seguida, os procedimentos utilizados para paralelizar a técnica de IC. No final deste capítulo alguns exemplos da programação seqüencial são mostrados apenas para se verificar o comportamento de cada modelo estrutural frente aos pré-condicionadores.

No capítulo 6 são realizados os exemplos aplicados em processamento paralelo. Assim, dois modelos estruturais considerados: treliça tridimensional e pavimento. Seus resultados de desempenho, para o número de nós existentes, são mostrados para o método de IC, variando-se o número de colunas na montagem da fatorização de [L] e compara-se estes resultados com o uso do pré-condicionador Poly(0) ou também conhecido Jacobi.

Capítulo 2

INTRODUÇÃO À COMPUTAÇÃO PARALELA

2.1 INTRODUÇÃO

Este capítulo será desenvolvido com o intuito de explicar de maneira geral os tipos de arquiteturas existentes em computadores. Assim, inicialmente serão tratadas as diferentes classificações existentes para tais máquinas, dando-se ênfase à classificação mais comum: a máquina de von-Neumann. Será apresentado então um estudo direcionado para computadores de alto desempenho com uma abordagem mostrando os três principais fatores inerentes a este tipo de máquina: velocidade de ciclo, comprimento de palavra e capacidade de memória.

Em seguida, se mostrará o porquê da necessidade da procura de utilização da concorrência para agilizar o processamento de instruções, mostrando-se os parâmetros para aumentar as taxas de execução, ou seja, as quantidades de operações de dados em ponto flutuante por ciclo. As formas de se aumentar a concorrência também serão apresentadas por técnicas ligadas à otimização em nível de *hardware* (*pipeline* e/ou tipos de memória) e em nível de *software* (processamento em paralelo de dados por computadores conectados em rede). Assim, será mostrado que, de maneira geral, a forma de execução das tarefas (instruções) para os diversos tipos de concorrência, é feita conforme a taxonomia proposta por FLYNN (1966).

Serão descritas, então, as duas formas mais comum de processamento de dados para computadores de alto desempenho: o processamento vetorial e o processamento paralelo. O primeiro pode ser resumido, genericamente, como o

emprego de técnicas de *pipeline* e/ou de memória intercalada/local para manipulação de dados em massa. O segundo é descrito como o particionamento físico dos dados em vários processadores (nós), sendo que estes podem ser controlados por uma única unidade de processamento (CPU) ou por vários sistemas independentemente, mas sincronizados e trocando dados uns com outros quando necessário.

Deve ser ressaltado que este capítulo não têm o intuito de elucidar todos os fenômenos, técnicas e procedimentos específicos necessários para o estudo que aqui será abordado, mas sim de posicionar este estudo sucinta e globalmente a respeito das diversas formas de arquiteturas de computadores existentes e, principalmente, da forma de realizações de instruções.

Para uma maior compreensão do assunto aqui tratado, ALMEIDA & ÁRABE (1991), DeCEGAMA (1989) e FREEMAN & PHILLIPS (1992) podem ser mais elucidativos do que o trabalho presente, pois este capítulo é tratado de maneira a compreender harmoniosamente o desenvolvimento e as técnicas empregadas as quais melhor se ajustem a um determinado conjunto de problemas numéricos.

2.2 ARQUITETURAS DE COMPUTADORES

O projeto na arquitetura de computadores engloba tanto o estudo da estrutura isolada de cada um de seus componentes, como por exemplo, os tipos e a capacidade de memória, os modelos de processadores, etc., como também o estudo entre essas estruturas funcionando como um conjunto interdependente e sistematicamente organizado no sistema computacional. Com relação a forma de organizar e interligar esses componentes para realização de instruções, surgiram, a partir do início da década de 40, vários conceitos (paradigmas) de organização de instruções para computadores. Em seguida, os conceitos mais comuns serão abordados resumidamente.

a) Máquina de von-Neumann (instruções dirigidas): a maioria dos computadores, de alguma forma, foram baseados neste tipo de sistema convencional de computação idealizado por NEUMANN (1945). Nele, conforme ALMEIDA & ÁRABE (1991), o

computador é organizado em cinco unidades, figura (2.1), onde cada unidade pode ser resumida como:

Entrada: transmite dados e instruções do ambiente 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.

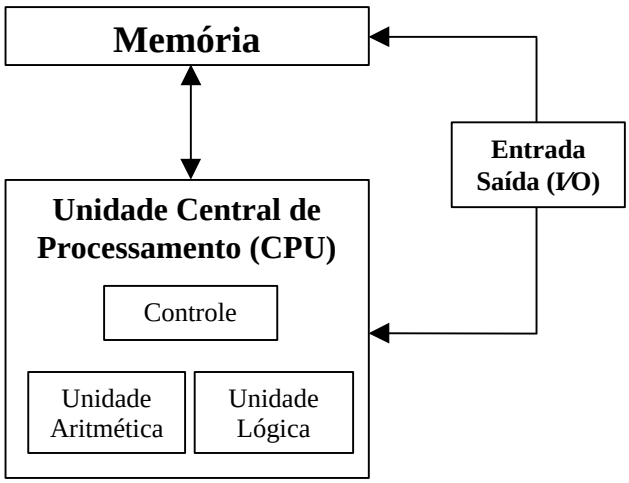


FIGURA 2.1 - Esquema da máquina de von-Neumann

As instruções, neste tipo de paradigma, são realizadas pelo processador seqüencialmente, e cada instrução, conforme ALMEIDA & ÁRABE (1991), percorre um ciclo que esquematicamente pode ser representado pela figura (2.2).

Com este tipo de arquitetura, a busca de instruções e/ou dados na memória, que está fora da unidade central de processamento, é um dos fatores mais relevantes para a perda de velocidade de um ciclo, ou seja, conforme ALMEIDA & ÁRABE (1991) comenta-se que no IBM 3090, o tempo de acesso à memória varia de 200 *ns* a 500 *ns*, enquanto o ciclo de execução de máquina é de 18,5 *ns*, demonstrando que o ciclo depende quase que exclusivamente do tempo de busca em memória, sendo que este obstáculo é conhecido como “gargalo de von-Neumann”.

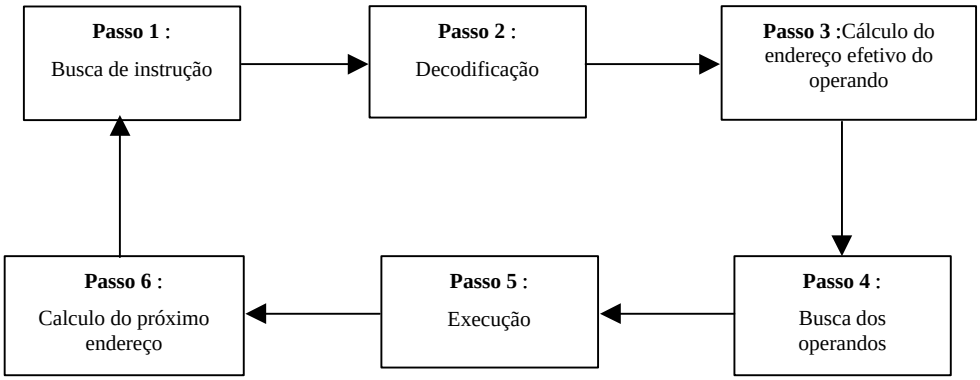


FIGURA 2.2 - Ciclo de execução de instruções na máquina de von-Neumann

b) Data flow (dados dirigidos): neste tipo de arquitetura as instruções se realizam tão logo os operandos são obtidos ou que estejam prontos para sua utilização, ao contrário do esquema da "máquina de von-Neumann", onde as instruções são realizadas passo a passo.

c) Reduction (exigência dirigida): este paradigma consiste em realizar as instruções quando os dados são exigidos para os outros cálculos, e não quando os dados estiverem prontos como no caso do conceito de data flow.

Assim, como a maioria dos computadores seguem o paradigma da "máquina de von-Neumann", neste trabalho serão abordadas as formas de aumentar o desempenho deste tipo de arquitetura. Desta maneira, métodos para evitar o atraso na busca de dados na memória principal, técnicas para a realização de várias instruções em um mesmo ciclo e a tentativa de diminuir o ciclo do processamento de instruções no processador são alvo de estudo a seguir.

2.3 COMPUTADORES DE ALTO DESEMPENHO

O conceito de desempenho em computação está ligado diretamente com o termo taxa de execução, que representa o número de operações em ponto flutuante realizadas por segundo pelo processador. Este termo também é conhecido como flops (*flow point operation per second*) ou seu múltiplo Mflops, ou seja, um milhão de flops. Esta taxa de execução depende basicamente de três parâmetros, os quais influenciam na maior ou menor taxa de execução, que são relacionados basicamente da seguinte forma:

$$\text{taxa de execução} = \frac{(\text{multiplicidade}).(\text{Concorrência})}{Ciclo} \qquad (2.1)$$

O ciclo indica o período em que estas operações são realizadas no processador, e com o desenvolvimento cada vez maior do setor de microeletrônicos este período tem diminuído notoriamente. No item seguinte, será apresentado o desenvolvimento dessa tecnologia nas últimas décadas.

A multiplicidade refere-se ao número de operações que se realiza em um ciclo. Assim, o quociente entre a multiplicidade e o ciclo define a freqüência do processador. Esta freqüência determina a velocidade do processador para executar uma operação.

O último fator que influencia no aumento de desempenho é a concorrência, a qual indica o número de *instruções* que serão realizadas em um mesmo ciclo, ou seja, a quantidade de tarefas (instruções) realizadas *simultaneamente*.

Simplificadamente, pode-se diferenciar a multiplicidade da concorrência com o seguinte exemplo:

- Seja a tarefa (instrução) de uma pessoa construir um muro. Para a execução deste muro, o operário executa 3 operações: pegar o tijolo na pilha, passar massa no seu extremo inferior e assentá-lo. Treina-se esta pessoa para que ela realize cada uma destas operações mais rápido, isto é, aumentando-se sua freqüência (multiplicidade por ciclo) em cada operação, assim ela realizará a construção do muro em menos tempo. Uma outra forma de se otimizar a

construção do muro seria acrescentando mais operários, introduzindo assim o conceito de trabalho em paralelo. Percebe-se que ambas as formas de otimização podem ser aplicadas juntas. Destaca-se que a rapidez de cada operário está limitado fisicamente, bem como a freqüência de cada processador é limitada pela velocidade da luz no vácuo. Mas, aumentar operários ou processadores para realizarem instruções iguais ou diferentes não tem limitação, apenas tem que adequar o ambiente de trabalho para otimizar a distribuição de tarefas dentre estes operários ou entre os processadores.

O nível de concorrência pode se dar em termos de *hardware* que é o emprego de técnicas que serão descritas posteriormente (técnicas de *pipeline*) e em nível de software, onde a concorrência se dá com o emprego de diferentes processadores trabalhando em conjunto.

Com isso, na busca cada vez maior de computadores de alto desempenho, estes dois fatores (aumento da freqüência e de concorrência) têm sido alvo de constantes estudos para a extração da maior taxa de execução possível. Desta forma, tecnologia ligada a otimização destes fatores têm sido alvo de interesse nas indústrias de computadores, como por exemplo, na IBM, Intel, Cray, Fujistu, Hitachi, NEC entre outras. Cada um destes fatores e sua parcela de importância no crescimento de computadores de alto desempenho serão abordados em seguida.

2.3.1 Tecnologia de processadores

A necessidade de diminuir o tempo de ciclo de uma operação no processador é aparentemente o mais imediato modo de otimizar o poder de resposta de um sistema computacional. Assim, o estudo do aperfeiçoamento destes componentes é algo que instiga os pesquisadores na área da microeletrônica há anos. Mas especificamente nas últimas 3 décadas, o desenvolvimento tecnológico da microeletrônica tem causado um avanço notável em componentes de *hardware*, podendo ser citado o projeto de circuitos mais rápidos e mais densos e também de sistemas de memória cada vez mais potentes.

Apresenta-se, assim, um gráfico na figura (2.3) que demostra, esquematicamente, o desenvolvimento dos circuitos microeletrônicos para computadores de alto desempenho com o avanço no ciclo de processamento de dados.

Em DeCEGAMA (1989) é comentado que com a atual forma de transmissão de dados, o limite da velocidade da transmissão de sinais está limitado à velocidade de propagação da luz no vácuo (que é da ordem de $3x10^8$ m/s). Como exemplo desta limitação física, pelo gráfico da figura (2.3), nota-se que para o período compreendido entre 1976 e 1985, o avanço no ciclo dos processadores foi de 3 vezes (passou de 12,5 *ns* a 4,1 *ns*), e para o período de 1985 a 1992, alcançou-se um ganho de 2 vezes (de 4,1 *ns* a 2,2 *ns*). Mostra-se, então, que procurar obter computadores de alto desempenho apenas com arquitetura otimizada no ciclo de um processador, hoje

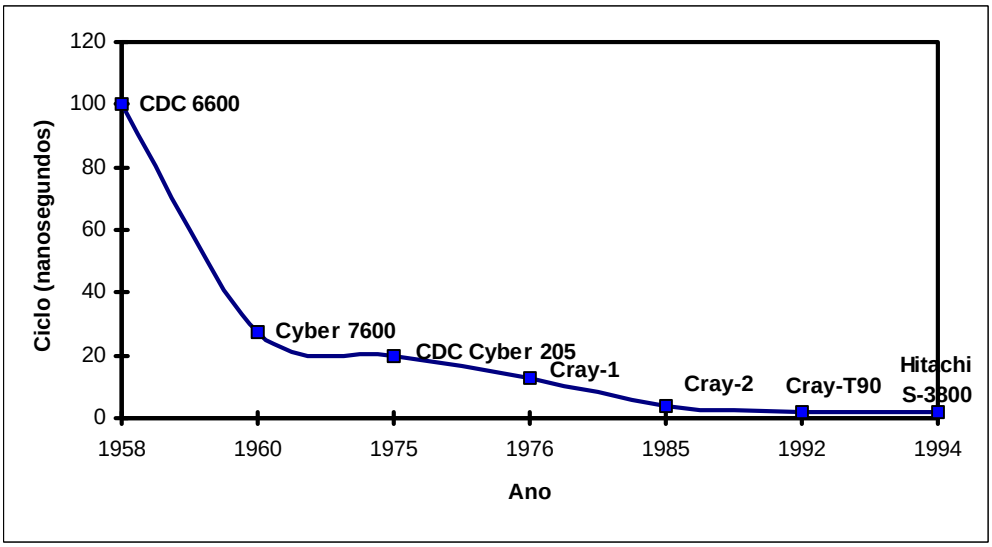


FIGURA 2.3 - Avanço da velocidade de processamento dos computadores de alto desempenho

em dia, é uma forma pouco eficiente, restrita e de custo elevado para a busca de mais tecnologia às empresas de computadores de alto desempenho.

Desta maneira, como exemplo colhido neste gráfico, nota-se que em 1976 com o computador Cray-1 seu ciclo era de 12,5 *ns*. Nove anos depois (1985) com o Cray-2 este ciclo passou a ser de 4,1 ns e, finalmente, em 1992 o ciclo atingiu com o Cray-T90 o valor de 2,2 *ns*. Mostra-se, então, que para os computadores da empresa Cray atingir um fator 3 vezes menor fora preciso 9 anos, e que quase para este

mesmo período (7 anos) alcançou-se uma tecnologia relativa de pouco menos do que 2 vezes.

2.3.2 Modelos de concorrência

A concorrência ou paralelismo é a forma de se obter com mais eficácia poderosos computadores de alto desempenho. Paralelismo significa executar tarefas (*jobs*) de um ou mais programas simultaneamente. Existem basicamente duas formas de executar essa concorrência: o paralelismo descrito pela técnica de *pipeline* ou com o uso de dois ou mais processadores interligados para executar uma mesma tarefa.

A técnica de *pipeline* consiste em realizar a concorrência das instruções paralelamente em um processador, sendo que este tipo de paralelismo está intrínseco na arquitetura do processador, não sendo necessário ao programador adaptar ou desenvolver seu programa para este tipo de otimização, já que esta forma de ganho de concorrência está em nível de *hardware*. De forma geral, a técnica é descrita como: dado um conjunto de instruções qualquer todas divididas em *n* subtarefas, e onde cada subtarefa *n* é realizada em um único ciclo *t*, esta técnica consiste em que para este mesmo período *t* pode-se *r*ealizar *n* subtarefas das diferentes instruções. Conclui-se então, que para um mesmo ciclo *t*, com esta técnica é possível realizar *n* subtarefas sob várias instruções, enquanto que do modo comum (sem *pipeline*), neste mesmo período *t*, realiza-se *n* subtarefas para apenas *uma única* instrução. Lembrando-se do exemplo do operário construindo o muro, esta técnica seria representada como se atribuíssemos mais mãos para este mesmo operário, sem necessidade de se readaptar o serviço e sim, o mesmo operário realizando o seu "auto-paralelismo".

A outra forma de paralelismo, com o uso de vários processadores, se caracteriza pelo ganho na taxa de execução obtido apenas dependente de se acoplar vários processadores existentes no mercado para realizarem uma mesma tarefa. Mas, para este tipo de concorrência, há a necessidade do programador desenvolver seu programa explorando esta arquitetura.

Para esses modelos de concorrência, o grau de paralelismo, resumido na figura (2.4) extraída de ALMEIDA & ÁRABE (1991), pode dar em diferentes níveis de tarefas: de paralelização de programas, de processos, rotinas, laços de instruções e em instruções. Para cada um desses níveis, existe uma classificação decorrente dessa forma de distribuição de *jobs* descritos a seguir:

- ***Processamento distribuído***: a paralelização nesse tipo de processamento está em nível de programa, ou seja, diferentes programas realizando diferentes tarefas, e as variáveis (matrizes, vetores ou escalares) necessárias em outro (s) processador (es) são transferidos pela interconexão existente no sistema de trocas de mensagens;

- ***Processamento paralelo****:* ocorre quando a paralelização está em nível de rotinas e processos. Assim, para um mesmo programa, pode-se dividir suas subrotinas e/ou partes do processo entre os vários processadores;

- ***Processamento vetorial:*** a paralelização está em nível de laços e/ou instruções, sendo que a técnica mais empregada neste tipo de procedimento é a técnica de *pipeline*.

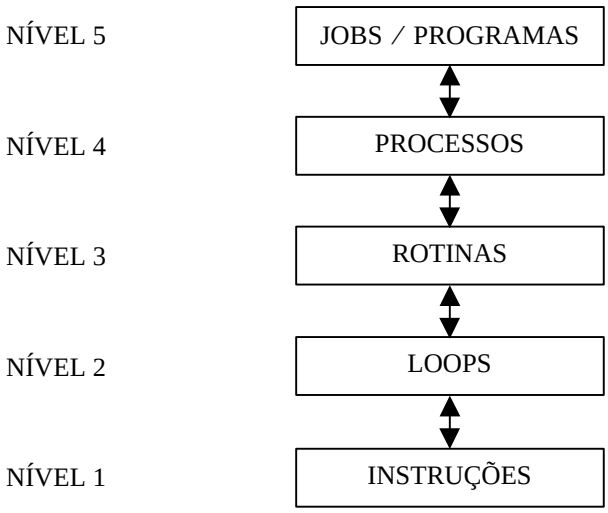


FIGURA. 2.4 Níveis de paralelismo

Para esses modelos de concorrência citados surgiram várias formas de execução dos processos. Uma dessas formas é a utilização de iguais (ou diferentes) instruções e com os mesmos (ou diferentes) dados. Esta questão de executar tanto diferentes como iguais instruções com iguais ou diferentes dados está relacionado com o conceito de diferentes controles de instruções e diferentes níveis de concorrência de execução. Assim, em função dessas possibilidades de fluxo de instrução e/ou de dados, várias classificações e taxonomias têm sido propostas, sendo que a mais conhecida é a taxonomia proposta por FLYNN (1966):

SISD (Single Instruction Single Data): é o processamento de uma seqüência de instruções sobre uma única seqüência de dados, ou seja, modelo clássico da máquina de von Neumann, como por exemplo os computadores que aplicam a técnica de *pipeline*;

SIMD (Single Instruction Multiple Data): refere-se ao processamento de uma seqüência de instruções sobre diferentes dados. Este modelo é usado em processamento vetorial;

MISD (Multiple Instruction Single Data): trata-se do processamento paralelo de diferentes instruções com a mesma seqüência de dados e em FREEMAN & PHILLIPS (1992) é relatado que ainda não foram desenvolvidos máquinas nessa categoria;

MIMD (Multiple Instruction Multiple Data): é o processamento de diferentes instruções ou iguais sobre diferentes dados. Nessa categoria se engloba uma grande gama de modelos de máquinas e conforme REZENDE (1995):" (...) 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 (...), permite subdivisões da categoria MIMD. Os computadores pertencentes à categoria MIMD são também conhecidos pelo nome de Multiprocessadores".

Assim, em função desses vários modelos de concorrência existentes para computadores de alto desempenho, algumas dessas arquiteturas têm se destacado como uso no meio cientifico e no meio industrial. São apresentados os dois principais modelos de computadores em paralelo utilizados: os computadores de processamento vetorial e os multiprocessadores.

2.3.2.1 Processadores vetoriais

A grande vantagem de se usar esse tipo de arquitetura é a sua aplicação em problemas com grande número de operações matemáticas, como, por exemplo, a multiplicação entre dois vetores ou mesmo entre duas matrizes. O fator que diferencia os processadores vetoriais (SIMD) das máquinas simples de von-Neumann (SISD), é a possibilidade de se fazer simultaneamente uma alta taxa de operações matemática elementares (produtos escalares ou vetoriais em *pipeline*, por exemplo) realizada por este modelo.

Assim, como exemplo, para a realização da soma de dois vetores de comprimento N em um computador escalar, esta operação levaria um tempo conforme mostrado na equação (2.2):

$$t_{ESCALAR} = K \times N \times t_{CICLO} \tag{2.2}$$

onde K é o número de passos necessários para executar a operação desejada (em geral igual a 3) e o t_{CICLO} é o tempo de ciclo do processador. Está operação, sendo realizada em um processador vetorial consumiria o tempo mostrado na equação (2.3), onde $t_{INICIAL}$ é o tempo de preparação da operação para a técnica de *pipeline*.

$$t_{VETORIAL} = t_{INICIAL} + N \times t_{CICLO} \tag{2.3}$$

A arquitetura básica deste tipo de computador é mostrada na figura (2.5), que será descrita a seguir. Quando uma instrução é buscada e decodificada no processamento de instruções, faz-se necessário verificar se esta instrução é uma operação escalar ou vetorial. Se a operação for escalar seu comprimento de palavra,

por exemplo de 32 bits, é dividido vetorialmente, como por exemplo em 4 partes de 8 bits, e então enviados para serem operacionalizados nas várias unidades aritméticas existentes em *pipeline* (*pipeline* aritméticos). Se a instrução for com operações vetoriais, envia-se para o controlador vetorial e também realiza-se suas operações nas unidades aritméticas em *pipeline*.

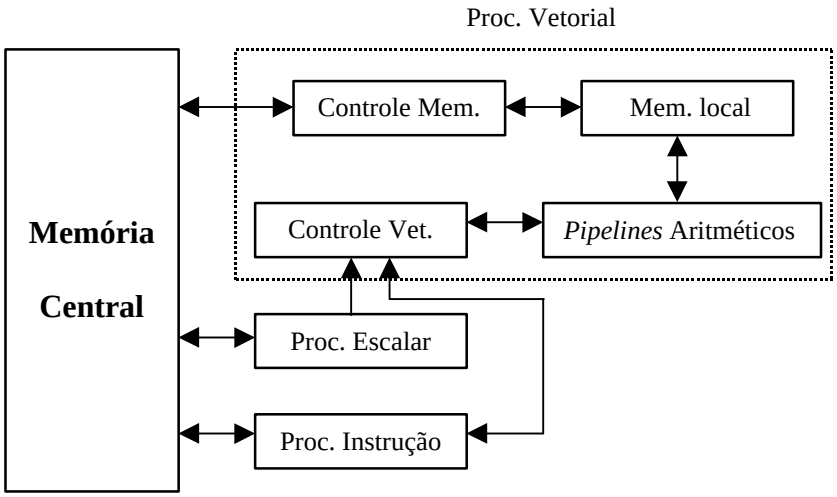


FIGURA 2.5 - Organização básica da forma de execução de instruções de um computador vetorial

A grande dificuldade no desenvolvimento desses computadores, conforme ALMEIDA & ÁRABE (1991), é o fato de que exige-se que as memórias dos processadores tenham que sustentar uma alta taxa de fluxo de dados exigida pela velocidade de operações deste sistema vetorial. Dessa maneira, mesmo com o avanço dos componentes de memória, ver NOOR (1997), ainda o custo para seu emprego eleva-se bastante para o volume de operações que são exigidos hoje em dia.

Com os empecilhos encontrados para este tipo de processamento, os pesquisadores têm buscado sistemas de computação com modelos MIMD em multiprocessadores no qual se tem sistemas independentes de memórias.

2.3.2.2 Multiprocessadores

O ganho de desempenho nos computadores vetoriais está restrito à concepção de nova tecnologia dos componentes, tanto no aumento da capacidade de memória

como na diminuição do ciclo do processador. Assim, uma maneira de aumentar o nível de paralelismo é realizar diferentes instruções (ou iguais) sobre os diferentes dados com o uso de vários processadores independentes simultaneamente.

Uma gama muito grande de computadores enquadram-se nesta classificação, sendo que a forma de conexão entre eles (interconexão), o número de processadores interligados (granulometria) e, principalmente, a arquitetura da memória (local ou global) são parâmetros que distinguem os vários grupos de multiprocessadores existentes.

O conceito de granulometria e de interconexão não são questões tão relevantes para o interesse do programador no desenvolvimento de programas que aproveitem este tipo de arquitetura. Ressalta-se, porém, que estes dois fatores podem ser importantes com relação a execução do *software* com um número muito grande (mais de cem) de processadores trabalhando em conjunto, neste caso é aconselhável verificar o estudo apresentado em REZENDE (1995) e em ALMEIDA & ÁRABE (1991) para maior compreensão destes parâmetros.

O tipo de acesso à memória é um fator que abre caminho para duas novas formas bem distintas de desenvolvimento de algoritmos para o maior aproveitamento de sua arquitetura específica. Por um lado, há os computadores com **memória global** (memória compartilhada), onde todos os processadores têm acesso a ela. A busca ou a mudança do valor de uma determinada variável é feita por qualquer processador a qualquer momento, desde que seja bem controlado pelo programador. Por outro lado, existem os computadores onde cada processador tem sua **memória local** (memória distribuída). Assim, as alterações são feitas localmente e os dados são conhecidos pelos demais processadores mediante a troca de mensagem ocorrida pela rede de interconexão.

Dessa forma, com estes dois tipos de modelos de sistemas computacionais existentes, são apresentadas, a seguir, algumas vantagens e desvantagens inerentes a ambos os paradigmas de memória:

Memória Compartilhada: conforme DeCEGAMA (1989) três problemas principais ocorrem com este tipo de memória: conflito de *software*, conflito de *hardware* e alto custo para escalabilidade.

O conflito de *software* ocorre quando dois ou mais processadores alteram a mesma variável sem a devida sincronização das instruções. Um exemplo deste tipo de problema é quando dois processadores (A e B) buscam uma variável K_i quase ao mesmo tempo para sua atualização numa mesma instrução. Assim, o procedimento correto do código deveria fazer com que o processador A altere K_i e posteriormente B a atualize novamente. Mas em virtude das diferentes velocidades de cada processador, pode-se ocorrer que A altere K_i e B a modifique, sem levar em consideração a atualização de A, já que os dois processadores já tinham buscado esta mesma variável em memória única. Desta forma, percebe-se que a sincronização dos processos entre os vários processadores tem fator importante para evitar este problema, que é conhecido como *racing condition*.

O conflito de *hardware* está ligado ao acesso simultâneo de um ou mais processadores à memória, devendo-se também usar métodos de sincronização para o controle de busca e envio de dados na memória pelos diferentes processadores. Neste caso, este fator é solucionado por sistemas de controle eletrônicos à memória.

O terceiro e maior empecilho que tem causado o não avanço deste tipo de arquitetura refere-se ao alto custo para aumentar o número de processadores que podem ser ligados para acesso à memória. Isto porque para se fazer uma arquitetura em que se tenha, por exemplo, 30 processadores próximos e que tenham acesso à memória, é necessário sistemas eficientes de controle de temperatura e de ligação entre processador e memória.

***Memória Distribuída**:* ainda conforme DeCEGAMA (1989), é mostrado que os computadores de memória distribuída têm dois tipos de conflito que devem ser ressaltados.

O primeiro estaria relacionado ao fato de que a forma geral de se resolver um problema, com este tipo de arquitetura é particionando os dados entre os vários processadores e trocando mensagens para atualizar as dependências entre as várias partes divididas do mesmo problema. Contudo, estas trocas de mensagens acarretam uma grande perda de tempo no processo e quanto mais dependente for o algoritmo numérico mais lento é o envio e o recebimento de dados entre os processadores.

A outra forma de conflito para esta arquitetura, conforme DeCEGAMA (1989), é a pouca disponibilidade de compiladores autoparalelizantes (*parallelising compilers*), ou seja, compiladores que averiguem no *software* os pontos que possam ser implicitamente paralelizados sem a necessidade de adaptar os programas existentes para computadores convencionais, ou seja, seqüenciais. Alega-se assim, a insuficiência de tais compiladores devido a complexidade que existe para este procedimento.

Mas estes dois problemas de conflito, destacados por DeCEGAMA (1989), têm sido objeto de estudo para serem superados por pesquisadores do mundo inteiro desde há última década (80). Para o conflito ocasionado pela troca de mensagem, podem ser lembrados os trabalhos de NOOR & PETERS (1989), JOHNSSON & MATHUR (1990), FARHAT & ROUX (1991), ADELI & KAMAL (1992), entre outros.

Nesses trabalhos, os autores procuram basicamente diminuir tal empecilho, estudando novos procedimentos numéricos com o intuito de se obter o mínimo possível de trocas de mensagens. Isto fez com que novas rotinas, até então já bem definidas, fossem abordadas sob outro enfoque, como por exemplo, a resolução de sistemas lineares advindos do MEF, que de maneira geral, era resolvido mediante o emprego do método direto de Gauss ou qualquer uma variante deste, como o Método de Cholesky. Começou-se assim, a serem explorados métodos iterativos para se resolver o mesmo problema, podendo-se citar os trabalhos de BITZARAKIS *et al.* (1997) e SCHMIT & LAI (1994).

Por fim, o problema da falta de compiladores autoparalelizantes têm também sido alvo de pesquisa pelas empresas do setor de computadores de alto desempenho, destacando-se a CRAY (1998) e a IBM (1998), que têm desenvolvido *software* do tipo HPF, sigla de *High Performance Fortran* com esta finalidade. Acredita-se que futuramente, o mercado estará dominado por computadores de alto desempenho em memória distribuída com o uso de compiladores autoparalelizantes. Vale ressaltar que mesmo com a utilização de tais procedimentos denominados de implícitos, o desempenho de um programa não fica totalmente alcançado, já que tais autoparalelizadores apenas paralelizam situações (processos, instruções, laços) em casos bem óbvios para estes procedimentos. Portanto, para maior ganho de

desempenho, cabe ao programador paralelizar manualmente as características intrínsecas de seu programa.

Este tipo de paralelização manual - conhecida como paralelização explícita - é uma alternativa de otimização de programa para a arquitetura de computadores de alto desempenho em memória distribuída.

Capítulo 3

O MEF ADAPTADO AOS MULTICOMPUTADORES

3.1 INTRODUÇÃO

Este capítulo tem o intuito de fazer a ligação entre o estudo de computadores de alto desempenho de memória distribuída (MIMD) e sua aplicação no campo da engenharia das estruturas com o uso do método dos elementos finitos (MEF). Dessa forma, apresenta-se primeiramente o esboço geral do método dos elementos finitos, e os procedimentos necessários para a solução de um problema do MEF na análise de estruturas estáticas.

Em seguida, atenção especial é dada as metodologias empregadas para computadores de arquitetura paralela de memória distribuída, principalmente dando-se ênfase à técnica empregada neste trabalho: o simples particionamento dos graus de liberdade da matriz de rigidez entre os vários processadores para a resolução do sistema linear particionado, com o emprego de um método iterativo.

3.2 O MÉTODO DOS ELEMENTOS FINITOS

Na análise numérica aplicada à resolução dos problemas no campo da engenharia das estruturas, dois métodos discretos têm se destacado no meio acadêmico e industrial: *o método dos elementos finitos (MEF) e o método dos elementos de contorno (MEC).*

O MEC, assim primeiramente denominado por BREBBIA (1978), trata do equacionamento em forma de equações integrais de contorno do meio contínuo, de forma a explicitar as incógnitas apenas em pontos discretos *do contorno* e, posteriormente, obtêm-se os valores do domínio a partir dos valores do contorno. Mesmo sendo um método recente, suas aplicações têm sido feitas largamente no campo da mecânica dos sólidos, destacando-se o tratamento de problemas de domínios infinitos, como por exemplo, nas aplicações em solos.

O MEF, genericamente, se baseia na divisão de todo o domínio em diversos elementos, tais que estes representem o campo de deformação ou de tensões do domínio por equações simplificadas, mas que satisfaçam as condições mecânicas e geométricas iniciais. As aplicações deste método são diversas, podendo ser citadas as análises de edifícios empregando-se elementos de barras que simulam os pilares e as vigas e elementos bidimensionais de placa ou de casca para simular o pavimento.

Assim, o MEF nada mais é do que dividir (discretizar) o domínio da estrutura em finitos elementos que possam ter nós internos ou externos do seu domínio. Estes elementos finitos que formam o meio contínuo devem representar o comportamento mecânico – mesmo que de maneira simplificada – do meio a ser considerado mediante a aproximação de seu domínio por funções aproximadoras, geralmente polinomiais.

A idéia do método dos elementos finitos é antiga: em 1956 TURNER *et al.* já apresentaram esta técnica no meio científico, mas sua aplicação corrente só se tornou possível graças ao surgimento dos computadores, pois tal procedimento gerava a necessidade de se manipular milhares de dados repetitivamente, o que combinava com o uso em tais máquinas.

Neste trabalho, os sistemas gerados são para resolver problemas da mecânica dos sólidos, os quais foram formulados pelo método dos elementos finitos baseados no método dos deslocamentos obtidos pelo princípio dos trabalhos virtuais (PTV). Mas, conforme BATHE (1982), também equivale ao uso do método dos resíduos ponderados e ao método de Ritz.

Sabendo-se que minimizar um funcional é o mesmo que resolver um sistema linear, o tratamento do funcional com as simplificadas equações destes elementos em conjunto leva a formação de um sistema linear do tipo

$$[K]\cdot\{U\}=\{F\} \tag{3.1}$$

onde [K] é conhecido como matriz de rigidez que caracteriza a resistência da estrutura perante as ações externas. {U} é o vetor incógnito dos deslocamentos e {F} é o vetor de ações nodais originados, por exemplo, pela variação de temperatura do sistema ou por cargas aplicadas na superfície de contato da estrutura.

Neste trabalho foram utilizados os elementos finitos conhecidos: DKT para a placa, CST para a chapa, elemento reticulado de viga com torção e um elemento advindo da análise matricial de treliça.

A seguir, são apresentadas de forma sucinta, as principais hipóteses de cada elemento, sendo que maiores detalhes podem ser encontrados nos trabalhos indicados em cada item.

3.3 ELEMENTO DKT

O elemento DKT é um elemento triangular de placa, que pode ser melhor compreendido no trabalho de BATOZ *et al.* (1980) e MESQUITA (1998). Apresenta nove graus de liberdade (uma translação e duas rotações) localizadas três a três nos vértices (ver figura 3.1). Ele, resumidamente, pode ser exposto como sendo formulado, primeiramente, partindo-se da teoria de placas espessas sobre a hipótese de REISSNER (1944,1945) e MINDLIN (1951), e negligenciando-se nos pontos discretos do contorno o funcional da energia potencial devido ao esforço cortante, considerando-se, então, apenas a parcela devida à flexão.

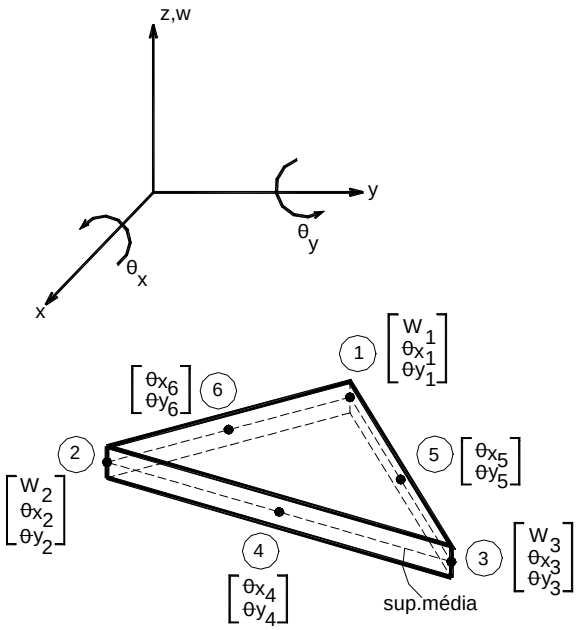


FIGURA 3.1 – Elemento finito DKT com os graus de liberdade interno e externo

A formulação do elemento DKT é baseada em quatro hipóteses, que são:

1) as rotações variam quadraticamente sobre o elemento;
2) as hipóteses de Kirchhoff são impostas nos nós dos vértices e nos nós do meio de cada lado do elemento;
3) a variação do deslocamento *w* ao longo dos lados é cúbica;
4) a variação da rotação da normal varia linearmente no ponto médio de cada lado do elemento.

No artigo de JEYACHANDRABOSE *et al.* (1985), é formulado a matriz de rigidez do elemento DKT [K_{DKT}] em coordenadas globais e um código em linguagem FORTRAN é apresentado para sua implementação explícita.

Para as ações aplicadas na superfície média da placa, conforme BATOZ *et al.* (1980), considera-se o carregamento distribuído como forças equivalentes concentradas em cada nó.

$$\{F^{e}_{DKT}\}^{T} = \frac{q \cdot A}{3} \cdot \{1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\} \qquad (3.2)$$

onde

q : carga uniformemente distribuída;

A: área de cada elemento.

Os esforços internos gerados para o problema de placa delgada (placa de Kirchhoff) são os momentos fletores (M_x e M_y) e o momento volvente (M_{xy}), os quais podem ser obtidos mediante as equações apresentadas em BATOZ *et al.* (1980) e MESQUITA (1998).

O vetor $\{U^{e}_{PLACA}\}^{T}$ referente aos graus de liberdade do elemento de placa é dado por:

$$\{U^{e}_{PLACA}\}^{T} = \begin{bmatrix} w_1 & q_{X_1} & q_{Y_1} & w_2 & q_{X_2} & q_{Y_2} & w_3 & q_{X_3} & q_{Y_3} \end{bmatrix} \qquad (3.3)$$

3.4 ELEMENTO FINITO RETICULADO DE VIGA

O elemento reticulado de viga é associado ao elemento de placa com a compatibilização dos deslocamentos desse com os deslocamentos da placa, sem consideração de excentricidade. Assim, para o desenvolvimento da matriz de rigidez da viga [K_{VIGA}] de formação da grelha, consideram-se os seus graus de liberdade de rotação no plano médio (θ_x e θ_y) e a translação (w) iguais aos da placa. Desta forma, para um mesmo nó onde há elementos de placa e de viga, a rigidez de ambos são somadas algebricamente para simular uma estrutura de pavimento - ver figura (3.2).

Na formulação do elemento reticulado, parte-se de um polinômio aproximador cúbico do deslocamento e de um polinômio linear para o efeito de torção, relacionando-se estes com suas rotações, suas curvaturas, esforços cortantes e momentos mediante as relações com as derivadas de ordem superior. Nos trabalhos

de CHAVES (1996) ou SAVASSI (1996) é apresentado a matriz de rigidez e o vetor de ação para a barra.

O vetor $\{U^{e}_{VIGA}\}^{T}$ referente aos graus de liberdade do elemento de viga é dado por:

$$\{U^{e}_{VIGA}\}^{T} = \begin{bmatrix} w_1 & q_{x_1} & q_{y_1} & w_2 & q_{x_2} & q_{y_2} \end{bmatrix} \tag{3.4}$$

3.5 MATRIZ DE RIGIDEZ DO PAVIMENTO

Com os elementos de placa e de viga já definidos separadamente pode-se agora montar a matriz de rigidez e o vetor de ações do pavimento. Isto é feito, como já citado anteriormente, com a soma nos nós - em comum - da rigidez e das ações de ambos os elementos independentemente. Esquematicamente, este procedimento pode ser interpretado como:

$$[K_{PAVIMENTO}] = \sum_{I=1}^{(N^{\circ}NO)} ([K_{DKT_{no_i}}] + [K_{VIGA_{no_i}}]) \tag{3.5}$$

$$\{F_{PAVIMENTO}\} = \sum_{I=1}^{(N^{\circ}NO)} (\{F_{DKT_{no_i}}\} + \{F_{VIGA_{no_i}}\}) \tag{3.6}$$

A matriz $[K_{PAVIMENTO}]$ gerada é quadrada de dimensão igual ao seu número de graus de liberdade por nó vezes o número de nós. Em outras palavras, possui uma dimensão de (3N X 3N) e o vetor de dimensão (3N), onde N é o número de nós.

Com a matriz de rigidez e com o vetor de ações do pavimento, pode-se resolver o sistema linear dado por:

$$[K_{PAVIMENTO}] \cdot \{U_{PAVIMENTO}\} = \{F_{PAVIMENTO}\} \tag{3.7}$$

onde $\{U_{PAVIMENTO}\}$ são os deslocamentos de todo o pavimento.

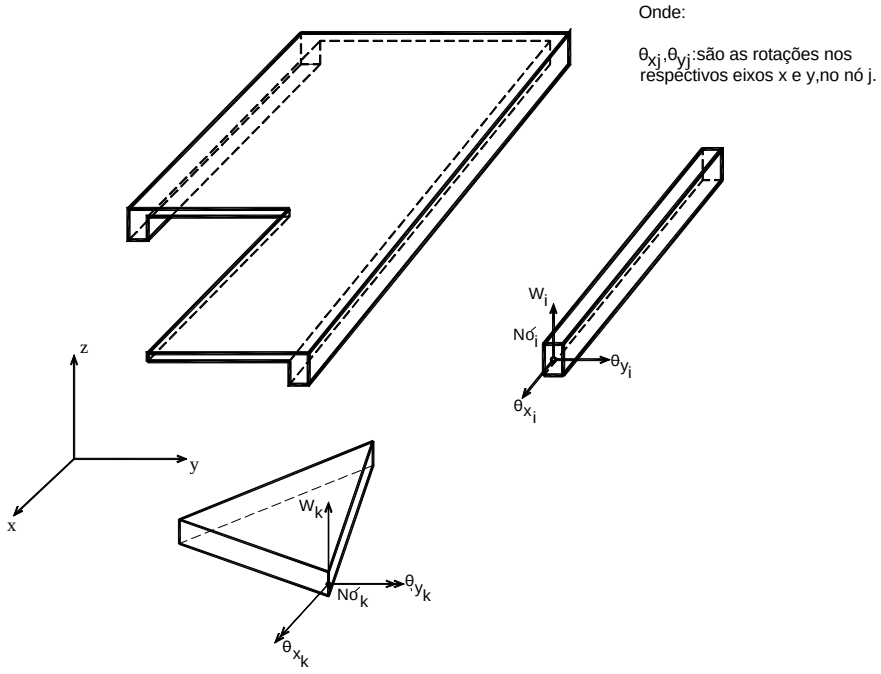


FIGURA 3.2 - Pavimento com os dois elementos estruturais constituintes: placa e viga

3.6 ELEMENTO TRIANGULAR CST (Constant Strain Triangle)

Este elemento formado pela aproximação linear para os deslocamentos, conforme SAVASSI (1996), é o elemento de chapa em que foi mostrado por TURNER *et. al.* (1956) como apresentação para o método dos elementos finitos.

O elemento triangular apresenta 2 graus de liberdade por vértice, sendo os deslocamentos em x (u) e em y (v) - ver figura (3.3). Assim, o polinômio aproximador é uma expressão algébrica completa do primeiro grau.

Em SAVASSI (1996) é apresentado todas as expressões e deduções que levam a formação da matriz de rigidez do elemento $[K_{CST}]$ e do vetor de forças equivalentes $\{F_{CST}\}$ para o caso de variação linear das forças externas sobre um lado genérico, bem como a forma de se obter as tensões e as deformações específicas com os deslocamentos conhecidos.

O vetor $\{U^e_{CST}\}^T$ referente aos graus de liberdade do elemento de chapa é dado por:

$$\{U^{e}_{CST}\}^{T} = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & u_3 & v_3 \end{bmatrix} \tag{3.8}$$

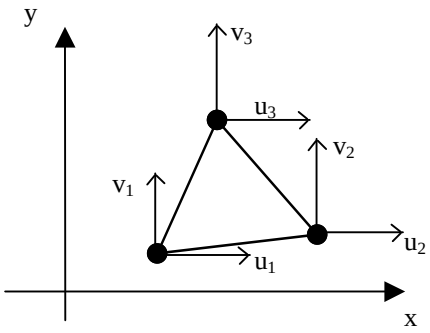


FIGURA 3.3 – Graus de liberdade do elemento CST

3.7 TRELIÇA TRIDIMENSIONAL

Para a utilização do programa que analisa estruturas por elementos de treliça, foi utilizado o algoritmo desenvolvido por SOUZA (1998) em seu trabalho de mestrado. Este algoritmo foi desenvolvido utilizando-se o procedimento matricial convencional para resolver um problema formado por elementos de treliças coplanares ou não.

Utiliza-se, assim, o método dos deslocamentos para se obter a matriz de rigidez de cada elemento e então, com a contribuição adequada de barra na formação do problema global, monta-se a matriz de rigidez da estrutura [$K_{TRELIÇA}$] bem como o vetor de ação nodal {$F_{TRELIÇA}$}.

Por fim, os esforços nas barras e as reações de apoio podem ser obtidos pelas relações matriciais, respectivamente, pela lei constitutiva de tensão-deformação e por equilíbrio.

O vetor $\{U^{e}_{TRELIÇA}\}^{T}$ referente aos graus de liberdade do elemento de treliça é dado por:

$$\{U^{e}_{TRELIÇA}\}^{T} = \begin{bmatrix} u_1 & v_1 & w_1 & u_2 & v_2 & w_2 \end{bmatrix} \tag{3.9}$$

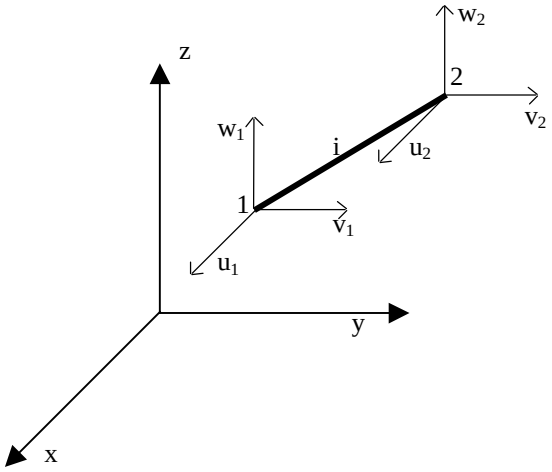


FIGURA 3.4 –Graus de liberdade do elemento de treliça

3.8 CONDIÇÕES DE CONTORNO

Para se resolver o sistema linear (3.1), deve-se, inicialmente, impor as condições de contorno geométricas da estrutura, já que os elementos foram construídos partindo-se de considerações de equilíbrio, de compatibilidade e de relações constitutivas para cada elemento, sendo que as condições de contorno ainda não foram prescritas.

Duas maneiras, em geral, são adotadas para prescrever o deslocamento: a técnica de zeros e um e a da penalidade (técnica do número grande).

A técnica de zeros e um consiste em eliminar as equações do sistema linear relativas às posições que são prescritas. Assim, anulam-se os valores da matriz de rigidez que contribuem para a parcela desta posição.

A técnica da penalidade ou método do número grande se baseia na hipótese de que a restrição ao deslocamento pode ser fisicamente interpretada como a imposição de uma rigidez “infinita” na direção restrita. Assim, o deslocamento encontrado para esta direção será quase nulo. Para melhor compreensão destes métodos cita-se o trabalho de SAVASSI (1996) e de CHAVES (1996).

3.9 PROCEDIMENTOS COMPUTACIONAIS USUAIS APLICADOS AO MEF

Para a resolução de estruturas em análise estática via MEF, o procedimento numérico necessário para se obter os parâmetros do domínio da estrutura estudada é, sucintamente, dado por:

a) ***montagem da matriz de rigidez (K_{EL}) dos elementos de uma estrutura***: para cada elemento, calcula-se sua matriz de rigidez local e esta é alocada na posição global da matriz de rigidez da estrutura (***K***). Faz-se isso, genericamente, elemento por elemento;

b) ***obtenção das ações nodais equivalentes (F)***: com as ações aplicadas na estrutura, estas são transformadas em forças nodais, montando-se assim o vetor de ação (F);

c) ***solução do sistema linear e obtenção dos deslocamentos nodais (U):*** resolve-se o sistema linear gerado para a estrutura ([K]{U}={F}), obtendo os deslocamentos do domínio da estrutura;

d) ***cálculo das tensões de cada elemento*****:** com as relações entre tensão-deformação e deformação-deslocamento usadas pelo modelo analisado, avaliam-se as tensões de cada elemento.

Assim, esquematicamente mostra-se na figura (3.5) o procedimento descrito nos itens **a, b, c** e **d**, onde η representa o número de graus de liberdade vezes o número de nós.

Como este método (MEF) é baseado em parâmetros que incluam todo o domínio da estrutura, isto faz com que o sistema linear, principalmente a formação da matriz de rigidez da estrutura, tenha dimensões proporcionais ao número de nós vezes o grau de liberdade do modelo usado.

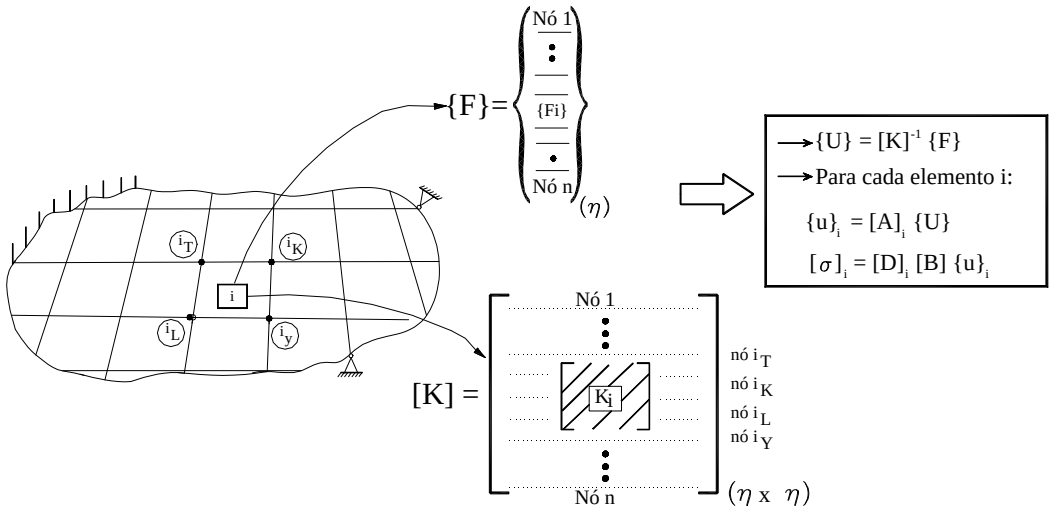


FIGURA 3.5 - Procedimento convencional da formação do sistema, elemento por elemento

Assim, como exemplo, é apresentado em JIANG *et al*. (1997) uma simulação numérica da parte de uma máquina onde se processa a conversão de polímeros em pó para polímeros em forma de paletas liquidas à alta temperatura. Essa simulação requer um espaço em disco de 8 Gbytes (com a armazenagem da matriz de rigidez de 255 mil linhas por 3400 colunas), onde tal requerimento de espaço livre em disco é praticamente impossível de ser feito da maneira apresentada na figura (3.5), mesmo em um supercomputador.

Chega-se, então, a uma dicotomia interessante na análise via MEF: por um lado, a maior discretização do problema leva a resultados mais próximos do real e, por outro lado, quanto maior esta discretização, maior a necessidade de espaço físico de memória no computador e de manipulação de grande quantidade desses dados pelo processador. Isto causa uma limitação de custo e tempo, fato este que não é interessante para as empresas e para os centros de pesquisa.

Desta forma, o impacto do processamento em paralelo com o uso de metodologias de divisão do problema em subproblemas, ou mesmo de se condensar em um único processador seus parâmetros incógnitos, têm aberto um outro campo de interesse na engenharia, onde sabe-se que o desenvolvimento de projetos depende diretamente do menor tempo de obtenção das respostas e com maior precisão possível.

Em seguida, é apresentado, sucintamente, três formas de se otimizar a resolução de um problema com o emprego respectivo das técnicas de subestruturação, de decomposição em domínio e o simples particionamento dos graus de liberdade da estrutura. Este último é o que foi desenvolvido neste trabalho para ser utilizado na resolução do pavimento. Assim, será dado mais atenção a este procedimento numérico.

3.9.1 TÉCNICA DA SUBESTRUTURAÇÃO

A técnica da subestruturação consiste em se dividir o domínio da estrutura em subestruturas entre pontos internos (nós internos) e pontos do contorno dessa sub-região (nós externos). Dessa forma, para não se montar a matriz de rigidez com todos os parâmetros do domínio da estrutura, relacionam-se os nós internos do domínio com os externos mediante a “condensação” da rigidez desses primeiros e associando-as aos parâmetros externos do contorno da sub-região. Assim, tem-se no final, uma matriz de rigidez apenas em função das incógnitas dos deslocamentos desses nós externos, diminuindo relevantemente o sistema linear a ser resolvido. Após esses deslocamentos dos nós externos de cada sub-região serem conhecidos, calcula-se os deslocamentos e as tensões dos pontos internos.

Um dos primeiros trabalhos a se utilizar dessa técnica foi o desenvolvido por PRZEMIENIECKI (1962). Nele se apresenta uma teoria geral da subestruturação, mostrando como a matriz de rigidez interna e o vetor de ação interno são lançados para os parâmetros do contorno, e, neste caso, os deslocamentos são impostos como nulos. Após faz-se globalmente a realocação desses deslocamentos no contorno. Depois, para cada subestrutura, calculam-se as suas deformações internas e as suas tensões. Como exemplo de aplicação, uma análise estática linear de dois modelos de aviões são apresentados.

Recentemente em BEZERRA (1995), é apresentada essa mesma técnica aplicada à análise de edifícios, onde cada pavimento representa uma subestrutura, sendo que os elementos não ligados diretamente aos pilares são os nós internos, e os

nós de ligação entre os pilares e as lajes de cada pavimento representam os nós externos.

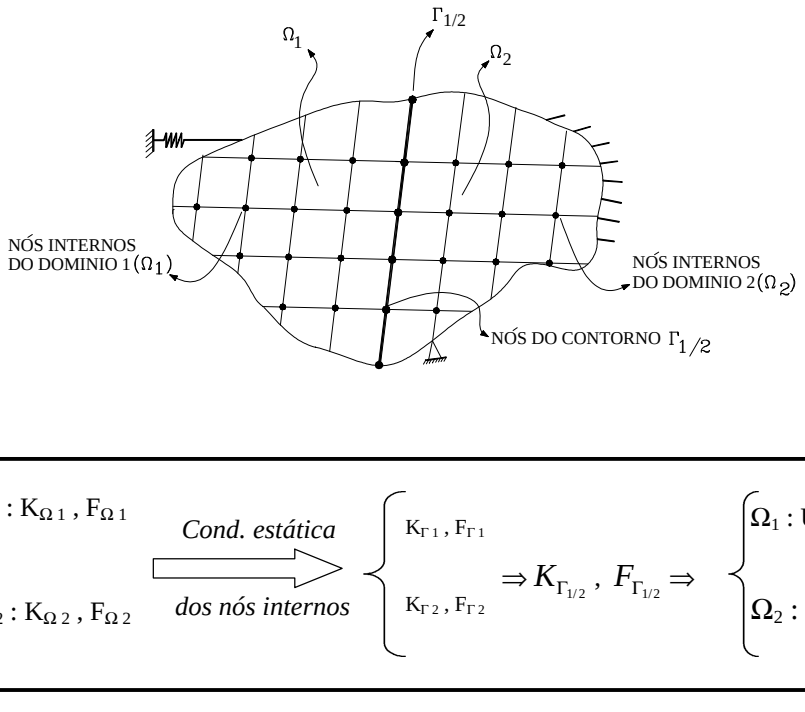


FIGURA 3.6 - Esquema da técnica de subestruturação em dois subdomínios

Esta técnica de otimização é empregada, em geral, em problemas que são resolvidos em computadores de arquitetura convencional (SISD) como, por exemplo, nos PC's. Entretanto, nas últimas duas décadas têm sido empregada no auxilio à técnica de decomposição em domínio para computadores com arquitetura paralela (SIMD e MIMD), a qual é descrita no item a seguir.

3.9.2 DECOMPOSIÇÃO EM DOMÍNIO

Mesmo com o emprego da técnica de subestruturação para se resolver um problema de grande escala via MEF, a necessidade do meio científico e industrial, com pode ser visto em JIANG *et al*. (1997) e LAW & MACKAY (1993) respectivamente, tem-se requerido o uso cada vez maior da capacidade de processamento e de armazenamento de centenas de milhares de variáveis. Assim, o

crescimento do processamento paralelo tem-se dado constantemente, e novas técnicas para a implementação desta arquitetura têm surgido.

Uma das formas mais imediatas de se usar o processamento paralelo em elementos finitos é dividindo-se a malha gerada em sub-regiões. Este procedimento é conhecido como a técnica da decomposição em domínio, a qual é empregada subdividindo-se o domínio de um corpo em subdomínios tais que cada um ou vários são alocados em processadores diferentes, independentemente.

Esta forma de se resolver um problema de MEF tem sido utilizado por muitos pesquisadores, citando-se os trabalhos de FARHAT & ROUX (1991), SAXENA & PERUCCHIO (1992), BITZARAKIS *et al*. (1997) e ADELI & KAMAL (1992), que aplicam a técnica de decomposição em domínio.

Para se otimizar a utilização da técnica da decomposição em domínio, deve-se ter em mente que a divisão da malha não deve ser feita aleatoriamente. Isto porque, como cada conjunto de subdomínios vai estar em um determinado processador, nada impede que em cada um desses haja uma maior quantidade de graus de liberdade do que em outro processador, podendo acarretar uma divisão da carga de trabalho desigual. Outro critério também importante para a melhor utilização da técnica de decomposição em domínio, é procurar gerar o menor número possível de nós na interface para que a comunicação entre os processadores seja mínima.

Entretanto, para se evitar esses problemas, vários programas de pré-processamento automático têm sido desenvolvidos para se resolver automaticamente a questão do balanceamento de cargas e do número de nós da interface. Como exemplo de um desses programas, em NASCIMENTO *et al.* (1997), é feito um estudo de algoritmos heurísticos para a otimização no particionamento de malhas. Três aplicações em malhas com a utilização dos aplicativos denominados de *Jostle* e *Métis* desenvolvidos respectivamente por WALSHAW *et al.* (1995) e KARYPIS & KUMAR (1995) são apresentados.

Aproveitando-se as técnicas da subestruturação, estas têm sido implementadas recentemente no método da decomposição em domínio. O procedimento para esta otimização consiste em se fazer com que as variáveis de cada domínio interno sejam lançadas para as interfaces das subestruturas, sendo que os

parâmetro da interface podem ou não estar em um mesmo processador. Resolve-se, então, o sistema particionado entre os processadores apenas com os nós do contorno, fazendo as trocas de mensagens necessárias entre os processos. Finalmente, para cada processador com posse dos deslocamentos das interfaces dos subdomínios, calculam-se seus parâmetros internos.

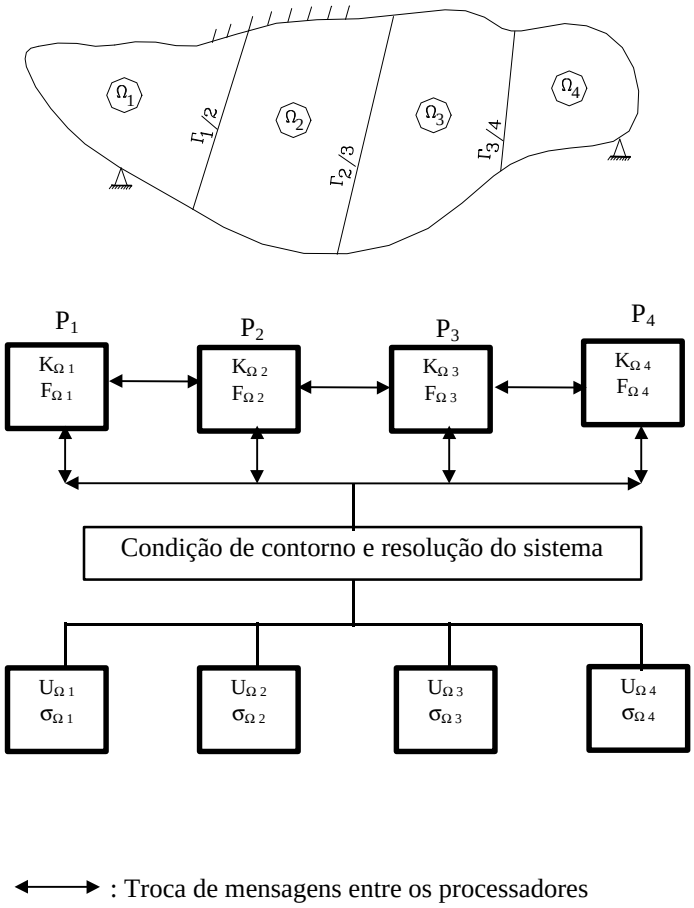


FIGURA 3.7 - Decomposição em domínio em memória distribuída, com quatro processadores

3.9.3 SIMPLES PARTICIONAMENTO DO SISTEMA LINEAR

No procedimento anterior de decomposição em domínio, mostra-se que para o emprego adequado daquela técnica deve se preocupar, principalmente, com a forma em que a malha é dividida. Com isso, quando se altera o número de processadores, o domínio é dividido de forma diferente, comprometendo o

desempenho do algoritmo desenvolvido. Assim, conforme JIANG *et al*. (1997), quanto maior a escalabilidade de um sistema, ou seja, a possibilidade de se variar o número de processadores, mais difícil é de se manter um sistema com desempenho ótimo para qualquer número de processadores utilizados.

Mediante esse "gargalo" apresentado acima e aproveitando-se o trabalho de REZENDE (1995), propõe-se aqui uma outra alternativa de particionamento do domínio para a abordagem em computadores de arquitetura paralela com memória distribuída: o simples particionamento do sistema linear.

Antes de se descrever o procedimento propriamente dito, apresenta-se, de forma genérica, o que foi desenvolvido por REZENDE (1995).

3.9.3.1 Abordagem Nodal

Em REZENDE (1995) é apresentado a necessidade do processamento paralelo para análise via MEF, os paradigmas de execução de instruções, os modelos de arquiteturas mais comuns, e o ponto mais importante de seu trabalho é o desenvolvimento de um algoritmo que monte a matriz de rigidez linha a linha. Isto porque para os computadores de memória compartilhada, existe conflito entre os vários processadores em um mesmo espaço da memória. Já que tal fato, como anteriormente descrito no capítulo 2, é um dos maiores empecilhos no avanço de tais máquinas.

Esse conflito citado aconteceria em função do método convencional de se montar a matriz de rigidez, ou seja, mediante a variação do número de elementos do domínio, onde vai sendo alocado sua contribuição nas posições na matriz de rigidez da estrutura, fato este que leva o conflito de memória (também conhecido de *racing condition*) entre os processadores.

É proposto, então, a montagem da matriz de rigidez via abordagem nodal. Assim, determinam-se todos os elementos associados a um mesmo nó e então, finalmente, uma varredura de todos os nós é feita e, desta forma, vai se contabilizando a contribuição de todos os elementos para cada nó, montando então a

matriz de rigidez da estrutura linha a linha (grau de liberdade por grau de liberdade). Mostra-se esquematicamente na figura (3.8) o que é proposto em REZENDE (1995).

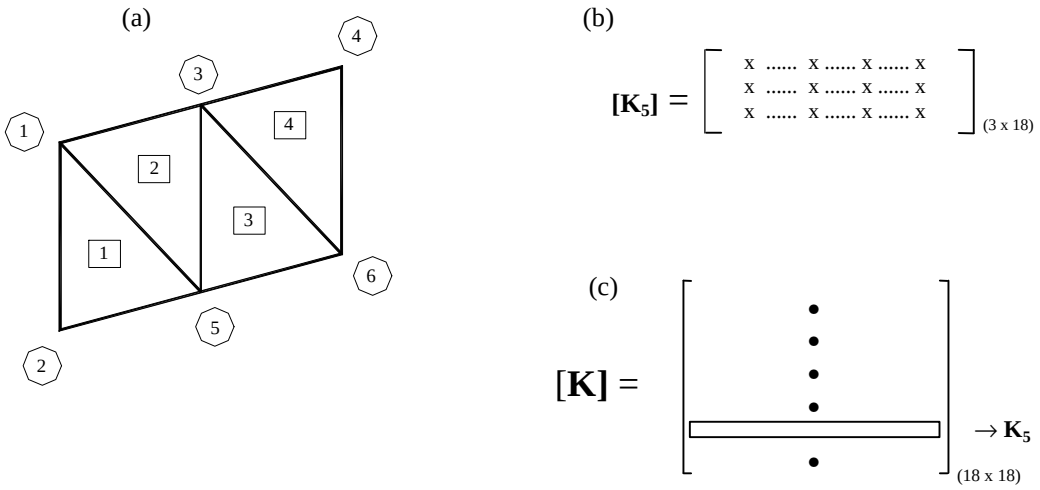


FIGURA 3.8 - Montagem da matriz de rigidez c/ abordagem nodal em um modelo c/ três graus de liberdade (Adaptado de Rezende (1995)).

Assim, como é observado na figura 3.8a, apresenta-se uma malha plana de uma estrutura genérica com a numeração dos elementos e dos nós. Em 3.8b, mostra-se a montagem da matriz de rigidez apenas dos parâmetros do nó 5, e na figura 3.8c esquematiza-se a alocação da parcela referente ao nó 5 na matriz global.

3.9.3.2 Montagem da matriz de rigidez em paralelo via abordagem nodal

Mesmo o estudo realizado por REZENDE (1995) tendo sido feito pensando em computadores de memória compartilhada, ou seja, todos os processadores têm conhecimento de todas as variáveis do sistema, aproveita-se neste trabalho sua técnica de abordagem nodal para a montagem da matriz de rigidez da estrutura em computadores paralelo de memória distribuída.

Tal fato é possível porque em computadores MIMD de memória distribuída, para se aproveitar os vários processadores trabalhando concorrentemente, pode-se dividir os graus de liberdade de toda a estrutura dentre os vários processadores

impostos e assim, via abordagem nodal, ir montando linha a linha em cada processo uma subparte da matriz de rigidez global.

Este procedimento difere do apresentado pela técnica da decomposição em domínio, porque neste último divide-se o domínio físico e no que é proposto aqui divide-se o *domínio dos graus de liberdade*. A grande vantagem da técnica realizada neste trabalho é que o balanceamento de cargas pode ser feito facilmente, já que apenas com o número total de graus de liberdade e com o número de processadores obtêm-se uma divisão bem equilibrada. Como a quantidade total de graus de liberdade não necessariamente é um valor múltiplo do número de processadores requeridos, a diferença existente entre cada processador, no balanceamento da distribuição, é dada pela seguinte relação apresentada na figura (3.9):

η = (graus de liberdade) $*$ (n° nós); (total de equações)
n = n° de processadores; (n << η)
k = int(η/n)
resto = η- $k * n$
$P_1 \rightarrow P_n$ = k (*)
Se resto $\neq$ 0
  $P_1 \rightarrow P_{resto}$ = 1 (**)
Fim de Se

Processador (n)	P_1	P_2	.....	P_{resto}	....	P_{N-1}	P_N
n° de graus de liberdade	k 1	k 1	....	k 1	....	k	k (*) (**)
Total de (η / n)	(k+1)	(k+1)	....	(k+1)	....	k	k

FIGURA 3.9 – Pseudo-algoritmo e balanceamento da divisão de dados entre os processadores

Mostra-se, então que a máxima diferença entre os processadores é de apenas uma linha, fato este que é imperceptível, dada a dimensão dos problemas a serem resolvidos.

A implementação numérica desta forma é realizada com o paradigma mestre/escravo (ver apêndice I), sendo que após a determinação do tamanho de cada submatriz de cada processador, cria-se um vetor para indicar e controlar qual parte da matriz de rigidez será montada por cada processador. Com isso, a abordagem nodal é realizada simultaneamente em todos os processos. O mesmo procedimento é feito

com o vetor de ações e com o vetor deslocamento. Esquematicamente é mostrado na figura (3.10) uma forma de se montar o sistema linear, e neste caso dividindo-se a matriz em colunas.

É evidente que existem muitas formas de se armazenar a matriz em cada processador, podendo esta ser dividida por colunas, por linhas ou qualquer outra forma, desde que seja conhecido a posição real que esta represente na estrutura. Como exemplo, em PIM, ver CUNHA & HOPKINS (1998), é apresentado uma maneira de armazenamento da matriz por coluna.

Neste trabalho, no capítulo 4 item (4.3.1), mostra-se a forma adotada para armazenar a matriz em cada processador. Destaca-se que o procedimento comentado em REZENDE (1995) facilita tanto para a montagem por coluna como por linha, já que a matriz é simétrica.

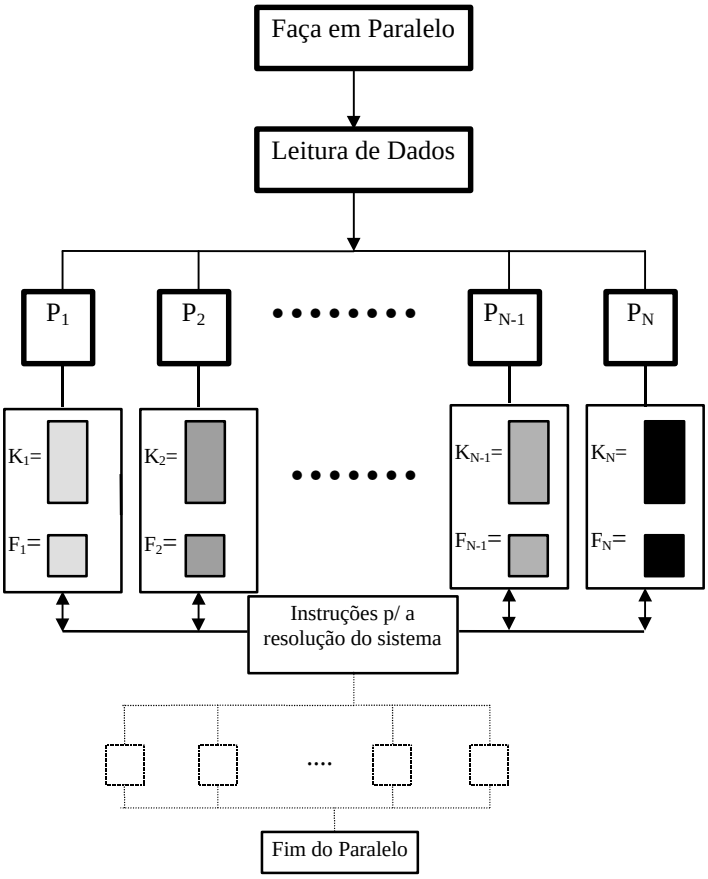


FIGURA 3.10 - Esquema geral da técnica utilizada para o simples particionamento

3.9.4 Obtenção dos esforços em paralelo

Para este trabalho existem dois tipos de esforços que devem ser obtidos. O primeiro em nível de elemento: sendo assim, calcula-se as deformações e as tensões locais, ou seja, ε_i e σ_i, onde *i* simboliza um elemento genérico. O segundo são os esforços médios obtidos nos nós, os quais dependem dos elementos que concorrem neste ponto.

A seguir serão apresentados as duas formas usadas para se obter estes valores em paralelo, onde cada processador já conhece o seu deslocamento global referente aos seus graus de liberdade de domínio.

3.9.4.1 Esforços por elemento

Os esforços que devem ser obtidos em nível de elemento são de fácil obtenção em paralelo. Assim, neste trabalho usou-se os seguintes procedimentos:

- fazer com que todos os processadores conheçam os deslocamentos de toda estrutura;
- para cada processador, aplicando-se as relações entre deslocamento, deformação e tensão-deformação, obter as deformações e os esforços elemento por elemento;
- todos os processadores (escravos), menos o primeiro (mestre), enviam seus valores de deformação e esforços para o mestre imprimir.

Na figura (3.11), mostra-se esquematicamente o procedimento discutido neste item.

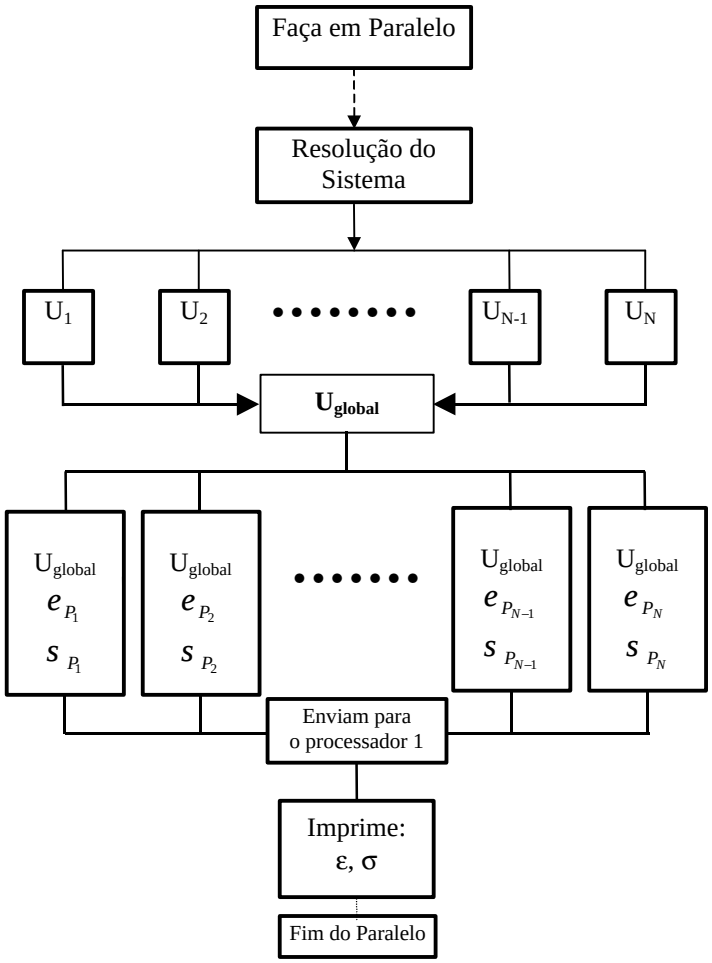


FIGURA 3.11 - Esquema geral utilizado para a obtenção dos esforços em cada elemento

3.9.4.2 Esforços por nó

Os esforços obtidos por nós, aproveitando-se o procedimento realizado para se obter os esforços por elementos, foram feitos da seguinte maneira:

- ao se obter os esforços por elemento em cada processador, obtêm-se os esforços nodais deste elemento, alocando estes em um vetor {Aux} em uma posição referente ao seu nó;
- ao final, todos os processadores enviam este vetor para o processador mestre e assim faz-se uma somatória global destes esforços;

- um vetor é criado {Elem_Conc} para obter o número de elementos que concorrem em cada nó;
- tira-se a média dos esforços por nó.

Esquematicamente, na figura (3.12), é apresentado o que fora feito neste item.

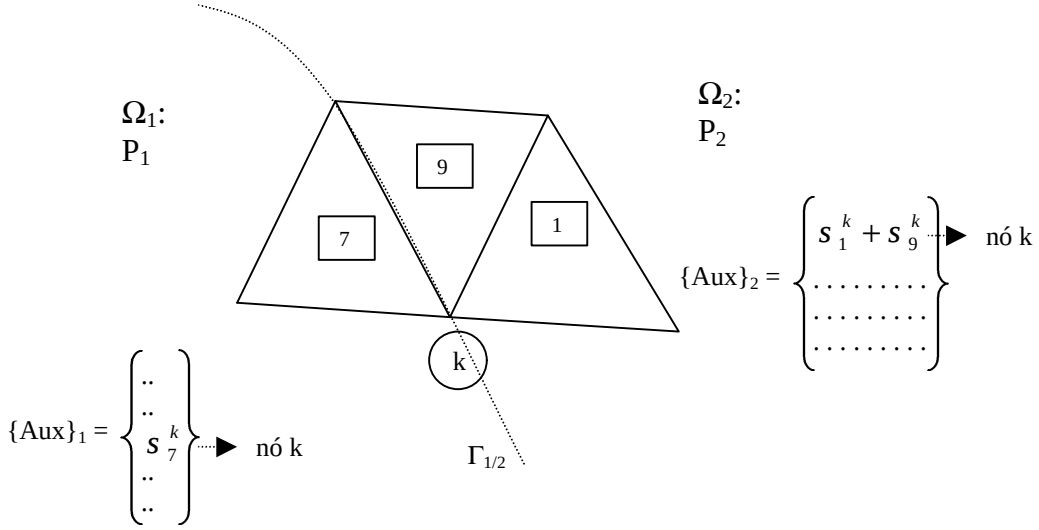


FIGURA 3.12 - Obtenção do vetor {Aux} nos diferentes processadores

Após todos os processadores (*np*) montarem o vetor auxiliar ({Aux}), enviam-no para o processador mestre, fazendo então a somatória dos esforços e, linha a linha, obtendo os esforços nodais médios, ou seja:

- processador mestre: $\{\text{Aux}\} = \sum_{j=1}^{np} \{Aux\}_j$

$$\{\text{Média\_esforços}\} = \sum_{i=1}^{n\_nós} \left(\frac{\{Aux\}}{\{Elem\_Conc\}}\right)_i$$

Capítulo 4

RESOLUÇÃO DO SISTEMA LINEAR

4.1 INTRODUÇÃO

A resolução do sistema linear do tipo

$$[K]\cdot\{U\}=\{F\} \tag{4.1}$$

gerada pelo MEF, onde [K] é a matriz de rigidez simétrica e positiva definida, {U} o vetor de deslocamentos e {F} é o vetor das ações, pode ser resolvido pelo dois métodos conhecidos na literatura: método direto e iterativo.

Conforme BATHE (1982), no início das aplicações do MEF, metade da década de 50, os algoritmos iterativos eram comumente utilizados.

A utilização dos métodos iterativos requer iterativamente a realização de um produto matriz-vetor, sendo que nesta matriz armazena-se apenas os valores diferentes de zero e sem alterá-los ao longo do método. Assim, para sistema esparsos - que são característicos do MEF - torna-se uma grande vantagem sua utilização, pois, este fator era imprescindível para a época em virtude de que os computadores tinham baixa capacidade de armazenamento de dados.

Mas, como desvantagem destes métodos iterativos, destaca-se a dificuldade de se estimar o tempo de convergência do sistema, já que a convergência do método

é fator dependente do número de condição (κ) [1]da matriz [K]. Isto faz com que, muitas vezes, este método alcance convergência em um número elevado de iterações (*ite*), sendo que isto torna o método ineficiente.

Para diminuir o número de iterações do método, pode-se, então, aplicar as técnicas de pré-condicionamento sobre o sistema originário dado pela equação (4.1). Este procedimento aplicado ao método é conhecido como métodos iterativos com pré-condicionadores. Estas técnicas aceleram a convergência do sistema, mas são procedimentos que dependem do tipo de problema a ser tratado, portanto, específicos e, portanto, é necessário se estudar um determinado pré-condicionador para acelerar cada conjunto de problemas.

Os métodos diretos têm como vantagens a estabilidade e por saber-se, *a priori*, o número de operações necessárias para a sua utilização. Este valor é da ordem de $\frac{1}{2} \cdot n \cdot m^2$, também conhecido pela forma $O(\frac{1}{2} \cdot n \cdot m^2)$, onde *n e m* são, respectivamente, o número de graus de liberdade e o comprimento de semibanda. A estabilidade do método é garantida, pois, a convergência do método independe do número de condição e assim, não precisam de estudos complementares para sua aplicação.

Como maior desvantagem para o método direto, sabe-se que os elementos da matriz armazenada [K], são alterados ao longo do algoritmo. Assim, a esparsidade é perdida ao longo das operações, portanto, tendo-se que armazená-la por inteiro, ou seja, com dimensão $(n \cdot m)$. Acrescenta-se também que o número de operações necessárias para o método varia quadraticamente em (m), ou seja, sistemas onde o comprimento de banda é alto torna-se ineficiente seu uso.

Assim, pelos motivos apresentados anteriormente, é que no início das aplicações do MEF, em função dos computadores terem pequena capacidade de memória, fator este limitante, optou-se pela utilização dos métodos iterativos.

Mas, com o desenvolvimento dos componentes microeletrônicos, dando-se ênfase a maior capacidade de armazenagem de dados, a partir da metade da década de 60, com as desvantagens oferecidas pelos métodos iterativos, pela boa

[1] O numero de condição é um fator determinado pelo quociente entre o maior e o menor autovalor da matriz.

estabilidade apresentada nos métodos diretos, estes últimos começaram a ser empregados em substituição aos métodos iterativos para resolução dos problemas advindos do MEF.

Ressalta-se, finalmente, que isto só ocorreu porque os modelos estruturais implementados na época, geravam sistemas lineares que eram possíveis de serem armazenados nos computadores existentes e pelo tipo de arquitetura destas máquinas, basicamente seqüencial ou vetorial, os métodos diretos ofereciam maiores vantagens em comparação aos métodos iterativos.

4.2 RESOLUÇÃO DO SISTEMA LINEAR EM PROCESSAMENTO PARALELO

Com o início da utilização do processamento paralelo aplicado ao campo da engenharia, principalmente para a resolução de um sistema linear, os métodos iterativos começaram a se destacar.

Isto pelo fato que estes métodos possuem características adequadas a este tipo de arquitetura, como boa versatilidade para se armazenar os sistemas esparsos e o procedimento onde se faz, a cada iteração, o produto matriz-vetor, o qual possui vantagens para paralelização sem muita perda de eficiência. Além de que estes métodos realizam muitas operações sobre vetores, fator também que se beneficia para tais computadores pelo sistema de *pipeline*.

A boa versatilidade no tratamento de sistemas esparsos significa que para sistemas muito grandes, pode-se ao invés de se armazenar toda a matriz de rigidez em banda de dimensão $(n \cdot m)$, guarda-se - em vetores – apenas as posições diferentes de zero, já que a matriz [K] não irá ser alterada.

Em relação ao produto matriz-vetor, a vantagem oferecida pelo método se apresenta, principalmente, para aplicação em sistemas de arquiteturas de memória distribuída, já que pode-se dividir os dados da matriz entre os diversos processadores e se realizar o produto sob estes diferentes dados em cada processador local, tendo-se que apenas após este produto, atualizar globalmente o vetor resultante desta operação, trocando-se mensagens.

Alguns trabalhos que comentam as vantagens de se aplicar os métodos iterativos para análise em processamento em paralelo são: CUMINATO & MENEGUETTE (1998), GROSSO & RIGHETTI (1988), LAW (1986) e BITZARAKIS *et al.* (1997).

Neste trabalho, mais adiante, será mostrado o procedimento desenvolvido para se calcular este produto em processamento paralelo com memória distribuída.

A grande vantagem da implementação do método direto em computadores com memória compartilhada se dá no fato de que os diversos processadores podem realizar todas as instruções do método sob diferentes dados. Tem-se que apenas estabelecer pontos do código computacional onde deve-se colocar comandos de sincronização entre estes processadores. Evita-se, assim, o problema já tratado no capítulo 2, de *racing condition.* Destacam-se alguns trabalhos onde é aplicado o método direto para análise em computadores paralelos com memória compartilhada: ADELI & KAMAL (1992), REZENDE (1995) e ZOIS (1988a e 1988b).

A grande desvantagem para paralelização do método direto aparece para implementação em computadores de memória distribuída. Neste tipo de arquitetura, os dados do sistema (4.1) são divididos entre os diversos processadores e durante a execução do algoritmo, em determinado momento, um ou vários processadores não realizam instruções com os seus dados armazenados, ou seja, ficam ociosos, não aproveitando, então, o potencial total do paralelismo neste método.

Um procedimento bastante adotado para otimizar resolução do sistema linear em processamento paralelo pelo método direto, tanto para computadores com memória compartilhada para se evitar o problema do *racing condition*, como para máquinas com memória distribuída, para se superar a questão da ociosidade, é utilizando as técnicas de subestruturação e decomposição em domínio, como já explicados no capítulo 3.

O emprego de uma destas técnicas acarreta numa sensível redução do número de variáveis globais do sistema a ser resolvido, lembrando-se que em cada processador após a resolução dos pontos de contorno (*Shur Complement*), obtêm-se as incógnitas de cada subestrutura alocada em cada processador, ver capítulo 3. Diminui-se, com isso, o tempo de sincronização para implementação em

computadores de memória compartilhada e menor ociosidade dos processadores para as máquinas com memória distribuída.

Citam-se assim alguns trabalhos onde é aplicado estas técnicas: ZOIS (1988a e 1988b), ADELI & KAMAL (1992), RAO *et al*. (1993), LAW & MACKAY (1993), TOPPING & KHAN (1996) e JIANG *et al*. (1997).

Dentre estes trabalhos, um que deve ser destacado é o realizado por JIANG *et al*. (1997) Em seu trabalho, JIANG *et al*. (1997) fazem a análise de uma estrutura tridimensional sob o efeito de temperatura, necessitando-se de 8 Gbytes para armazenar a matriz de rigidez. A técnica utilizada foi a técnica da decomposição em domínio em computadores com memória distribuída. Armazena-se, assim, as sub-regiões da estrutura nos diferentes processadores, ao todo são 22 nós, condensando os graus de liberdade nos nós do contorno entre cada sub-região. Desta forma, reduz o sistema e para resolvê-lo, utiliza-se um algoritmo híbrido direto-iterativo.

Para o trabalho aqui apresentado, a resolução do sistema foi realizada via um método iterativo denominado de método dos gradientes conjugados. Este método tem-se destacado para aplicação em processamento paralelo graças a sua facilidade de implementação e pela sua boa convergência quando se utiliza um pré-condicionador adequado. Assim, o próximo item mostrará em detalhes o método.

Ressalta-se que existem muitos outros métodos iterativos que oferecem vantagens para paralelização. Cita-se os trabalhos de BATHE (1982), DEMMEL (1993) e CIMERMAN (1996), os quais apresentam em detalhes outros métodos iterativos, como: Gauss-Seidel, Jacobi e Quase-Newton (BFGS).

4.2.1 MÉTODO DOS GRADIENTES CONJUGADOS

O Método dos Gradientes Conjugados é aplicado sobre a matriz [K], sendo esta simétrica e positiva definida. A simetria é atendida tal que para um valor da matriz [K], por exemplo, k_{ij} tem-se que $k_{ji} = k_{ij}$, para qualquer i e j.

Para uma matriz [K] ser considerada positiva definida, tem-se que para todo vetor {U} não nulo, é válido $\{U\}^T[K]\{U\} > 0$. Geometricamente, isto significa que existe um ponto crítico $\{U_k\}$ na equação

$$\Phi(\mathrm{U}) = \frac{1}{2}\{U\}^T \cdot [K] \cdot \{U\} - \{U\}^T \cdot \{F\} \tag{4.2}$$

que minimize o sistema linear da equação (4.2).

O Método dos Gradientes Conjugados (GC) é baseado na estratégia do método da *descida íngreme*. Nessa estratégia, é escolhido um ponto $\{U_0\}$, como valor inicial, e mediante uma série de aproximação $\{U_1\}$, $\{U_2\}$, $\{U_3\}$...., $\{U_n\}$ que é realizada até encontrar um ponto numericamente próximo do mínimo da equação (4.1).

Assim, aplicando as técnicas do cálculo variacional na equação (4.2), chega-se a expressão:

$$\Phi(\mathrm{U_p}+\alpha \mathrm{r_p}) = \frac{1}{2}(\{U_p\} + a \cdot \{r_p\})^T \cdot [K] \cdot (\{U_p\} + a . \cdot \{r_p\}) - (\{U_p\} + a . \cdot \{r_p\}^T) \cdot \{F\} \tag{4.3}$$

onde $\{r_p\}$ é o vetor resíduo, dado por:

$$\{\mathrm{r_p}\} = \{F\} - [K] \cdot \{U_0\} \tag{4.4}$$

Derivando e minimizando convenientemente a expressão (4.3) , o valor do escalar α para que $\Phi(\{U_p\}+\alpha\{r_p\})$ seja mínimo é

$$a_{CRITICO} = \frac{\{r_p^T\} \cdot \{r_p\}}{\{r_p^T\} \cdot [K] \cdot \{r_p\}} \tag{4.5}$$

e então, a próxima aproximação do vetor {U} será,

$$\{U_{p+1}\} = \{U_p\} + a_{critico} \cdot \{r_p\} \tag{4.6}$$

e o pseudo- algoritmo pode ser visto na figura (4.1).

→ j = 0, $U_0 = 0$, $r_0 = F$

→ **Enquanto** $\|r_j\| < \varepsilon$ (critério de parada)

→ j =j + 1

→ $a_j = \dfrac{r_{j-1}^T \cdot r_{j-1}}{r_{j-1}^T \cdot K \cdot r_{j-1}}$

→ $U_j = U_{j-1} + \alpha_j r_{j-1}$

→ $r_j = F - K U_j$

→ **Fim Enquanto**

FIGURA 4.1 – Pseudo-algoritmo do método da descida íngreme

Um problema que surge na estratégia da descida íngreme, é que a convergência pode ser lenta, pois a estratégia não impede que em determinado passo, chegue-se a uma mesma direção que um outro passo anterior, isto causa uma oscilação no mesmo ponto que não é o de mínimo.

Como avanço, no Método dos Gradientes Conjugados (GC) é usado a estratégia de sempre ir buscando direções ortogonais $\{p_0\}$, $\{p_1\}$, $\{p_2\}$,....., $\{p_{n-1}\}$ às direções já calculadas no passo anterior. Para cada uma dessas direções, se encontrará uma das coordenadas de {U}, com isso, após *n* passos, que é a dimensão de {U}, o processo terminará e a solução procurada é encontrada.

A aproximação do vetor {U} fica neste processo:

$$\{u_{j+1}\} = \{u_j\} + a_j \{p_j\} \tag{4.7}$$

e a garantia para atingir a ortogonalidade das direções de $\{p_i\}$ e $\{p_{i+1}\}$ de forma que sejam K- ortogonais é:

$$\{p_i^T\} \cdot [K] \cdot \{p_j\} = 0 \text{ , para } i \neq j \tag{4.8}$$

e o pseudo- algoritmo do Método dos Gradientes Conjugados é apresentado na figura (4.2).

Os teoremas (4.1) e (4.2), que têm suas respectivas demonstrações em CUMINATO & MENEGUETTE (1998) e LUENBERGER (1969), complementam os critérios de convergência do método:

Teorema 4.1: Se as direções de busca $\{p_0\}$, $\{p_1\}$, $\{p_2\}$,....., $\{p_{n-1}\}$ são K-conjugados (K-ortogonais) e o escalar α_j é escolhido da forma que: $a_j = -\frac{\{p_j^T\}\cdot[K]\cdot\{r_j\}}{\{p_j^T\}\cdot[K]\cdot\{p_j\}}$

então o processo termina em, no máximo, *n* passos, onde *n* é a dimensão da matriz quadrada [K] do sistema da expressão (4.1).

Teorema 4.2: Seja [K] simétrica e definida positiva, então o algoritmo dos gradientes conjugados produz uma seqüência de vetores $\{U_0\}$, $\{U_1\}$, $\{U_j\}$..., com a seguinte propriedade:

$$\left\|\{U\}-\{U_j\}\right\|_K \leq 2\cdot\left\|\{U\}-\{U_0\}\right\|_K \cdot \left(\frac{\sqrt{k}-1}{\sqrt{k}+1}\right)^k \qquad (4.9)$$

onde $\kappa = \frac{l_{\max}}{l_{\min}}$ e $\lambda_{máx}$ e $\lambda_{mín}$ são, respectivamente, o maior e o menor autovalor de [K] e κ é conhecido como número de condição.

Em relação ao que foi exposto nos teoremas (4.1) e (4.2), duas questões têm que ser salientadas:

1) Pelo teorema (4.1), o Método dos Gradientes Conjugados é matematicamente um método direto. Isto porque se conhece inicialmente o número máximo de iterações. Mas, numericamente, em virtude da limitada precisão numérica computacional existente, as direções conjugadas não são precisamente alcançadas, fazendo com que - muitas vezes – o limite de convergência não seja finito. Assim, costuma-se chamá-lo em muitos trabalhos, de método iterativo.

$\rightarrow p_0 = r_0 = F - K\,U_0$

$\rightarrow j = 1$

$\rightarrow$ **Enquanto** $\left\|r_j\right\| < \varepsilon$ (critério de parada)

$$\rightarrow a_j = \frac{r_j^T \cdot r_j}{p_j^T \cdot K \cdot p_j}$$

$\rightarrow U_{j+1} = U_j + \alpha_j\, p_j$

$\rightarrow r_{j+1} = r_j - \alpha_j\, K\, p_j$

$$\rightarrow b_j = \frac{r_{j+1}^T \cdot r_{j+1}}{r_j^T \cdot r_j}$$

$\rightarrow p_{j+1} = r_{j+1} + \beta_{j+1}\, p_j$

$\rightarrow j = j + 1$

$\rightarrow$ **Fim Enquanto**

FIGURA 4.2 – Pseudo-algoritmo do método dos gradientes conjugados

2) Pelo teorema (4.2) e pela equação (4.9) dois fatores influenciam na convergência do método dos gradientes conjugados: a aproximação inicial $\{U_0\}$ e no número de condição (κ).

- Em relação a aproximação inicial, em geral, na literatura é comentado que este fator é de pouca importância para a eficiência do método. Entretanto, no trabalho de SCHMIT & LAI (1994) é apresentado um estudo de otimização onde se obtêm uma aproximação inicial para o vetor $\{U_0\}$. E nos últimos anos, a pesquisa na área de inteligência artificial tem dado atenção a este item, onde a estratégia é criar bancos de dados de deformações e/ou de esforços de determinados modelos estruturais e, assim, aplicar algoritmos especiais - conhecidos como meta-algoritmos – para se otimizar a aplicação

deste método. No trabalho de GROSSO & RIGHETTI (1988) é feito um estudo introdutório neste assunto para aplicação no MEF.

- Pela equação (4.9), nota-se que a influência da variação do número de condição (κ) na convergência do método, é mais significativo do que a aproximação inicial $\{U_0\}$, assim, quanto maior é a diferença entre os autovalores da matriz [K], mais demorada é a convergência do método. Assim, o principal empecilho para a utilização deste método, é justamente a necessidade de se ter um número de condição (κ) de [K] o mais próximo do valor unitário.

Nos últimos cinco anos, extensos estudos têm sido dedicados à busca de técnicas matemáticas para se obter um número de condição (κ) o mais próximo do valor um.

A idéia básica empregada, é de ao invés de se resolver o sistema $[K]\cdot\{U\}=\{F\}$, resolve-se um sistema do tipo $[M]\cdot[K]\cdot\{U\}=[M]\cdot\{F\}$, tal que a matriz obtida do produto $([M]\cdot[K])$ seja o mais próximo possível da matriz identidade, onde sabe-se que $\kappa_{identidade}=1$. Esta matriz [M], é conhecida como pré-condicionador, que nada mais é do que um acelerador de convergência para o método.

Finalmente, ressalta-se que esta análise numérica de otimização da resolução de sistemas lineares via Método dos Gradientes Conjugados com o uso da técnica de pré-condicionamento, é a área que deve ser dada mais atenção para implementação no MEF, ou em qualquer outra aplicação com este método de resolução de sistemas. Sem o seu adequado uso em computação paralela, não se obtêm ganho de desempenho de tais arquiteturas, pois como o método é sensível ao condicionamento da matriz, em geral, a convergência sem a sua utilização é lenta.

4.3 PACOTE PIM

Para a implementação em paralelo do método dos Gradientes Conjugados (GC) utilizou-se o pacote de subrotinas PIM, ver CUNHA & HOPKINS (1998), que

foi desenvolvido para a resolução em ambiente paralelo de sistemas lineares simétricos e não-simétricos. Este pacote oferece além do método dos GC, outras alternativas de métodos iterativos como, por exemplo, o Bi-Gradiente Conjugado (Bi-GC), Gradiente Conjugado Quadrado (QGC), a versão estabilizada do Bi-Gradiente Conjugado (Bi-CGSTAB), o Resíduo Mínimo Generalizado (GMRES), dentre outros.

O pacote PIM, tem o intuito de dar total liberdade ao usuário no que se refere ao armazenamento da matriz, seu acesso e particionamento, bem como oferecer portabilidade aos diversos tipos de ambientes de programação em computadores paralelos.

Para se atingir estes dois fatores citados anteriormente, três pontos que são de responsabilidade do usuário para serem implementados no método são:

- O produto matriz-vetor;
- A implementação com o pré-condicionador (se houver);
- Produtos internos dos escalares e critério de parada.

4.3.1 Produto matriz-vetor

No que se refere ao produto matriz-vetor, o pacote PIM deixa por conta do usuário sua implementação. Assim, para manter a portabilidade do pacote, o usuário deve conhecer as características especiais (se houver) de sua matriz, seja ela, por exemplo tridiagonal, pentadiagonal, cheia ou em banda, e desenvolver um procedimento heurístico ou não heurístico para utilizá-lo em processamento paralelo. Assim, apresenta-se aqui a metodologia heurística desenvolvida para se poder aproveitar as características advindas do modelo do MEF comum: simetria e esparsidade.

No que diz respeito ao armazenamento da matriz de rigidez negligenciando-se os valores fora do espectro do comprimento de semibanda, emprega-se o procedimento convencional para a montagem da matriz, onde gera-se seus valores começando pela diagonal principal e os alocam à partir da primeira coluna de cada linha, transladando-se estas colunas.

Para se realizar o produto matriz-vetor de um sistema em banda é necessário efetuar o produto de cada valor da matriz duas vezes (menos para os valores da primeira coluna que representam a diagonal principal), ou seja:

- Para a linha *i*, o valor $[K_{ij}]$ multiplica o valor do vetor $\{U_j\}$, armazenando este valor na linha *i (se i ¹ j)*;

- Para a linha *j*, o valor $[K_{ij}]$ multiplica o valor do vetor $\{U_i\}$ e armazena-o em *j (se i ¹ j)*.

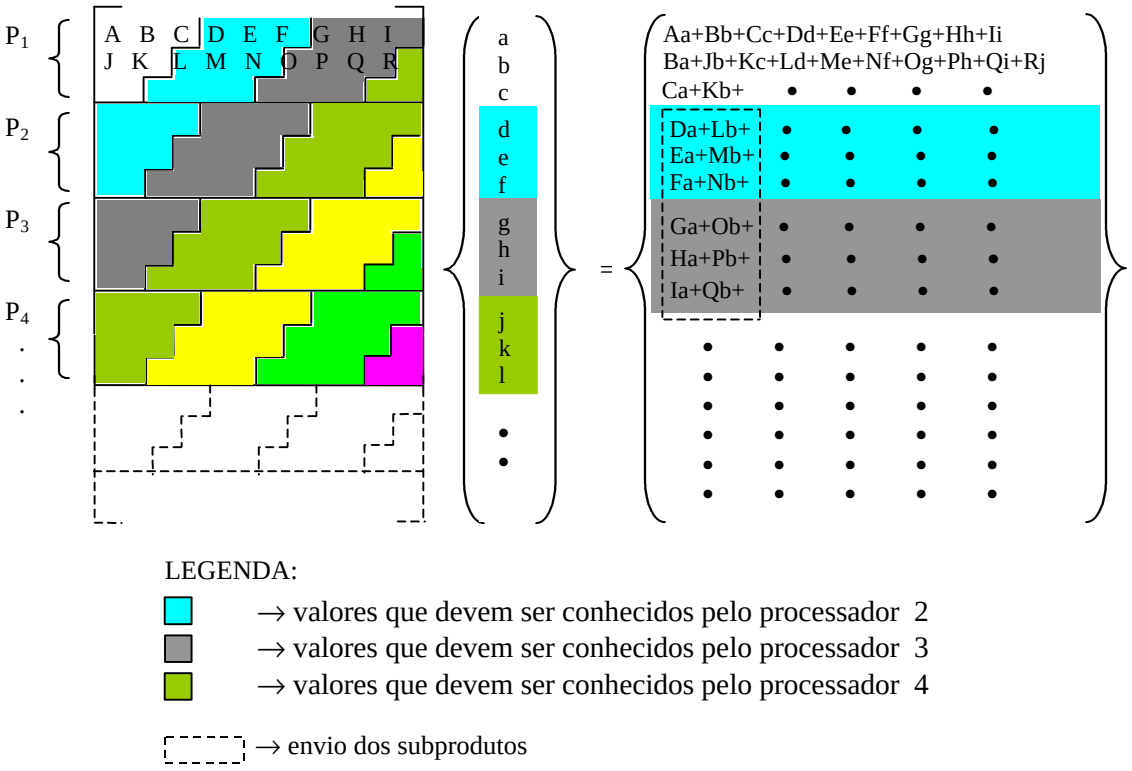


FIGURA 4.3 – Produto matriz-vetor para multicomputadores

Desta forma, na figura (4.3) mostra-se esquematicamente o procedimento empregado para se executar o produto matriz-vetor sendo a matriz em semibanda em paralelo. Ressalta-se que nesta figura é dado ênfase à metodologia empregada para contornar os problemas gerados por este tipo de armazenamento em banda. Deixa-se esclarecido que o maior e o menor trabalho para se transferir vetores para outros

processadores está relacionado com a relação entre o número de grau de liberdade existente em cada processador e com o espectro do comprimento da semibanda.

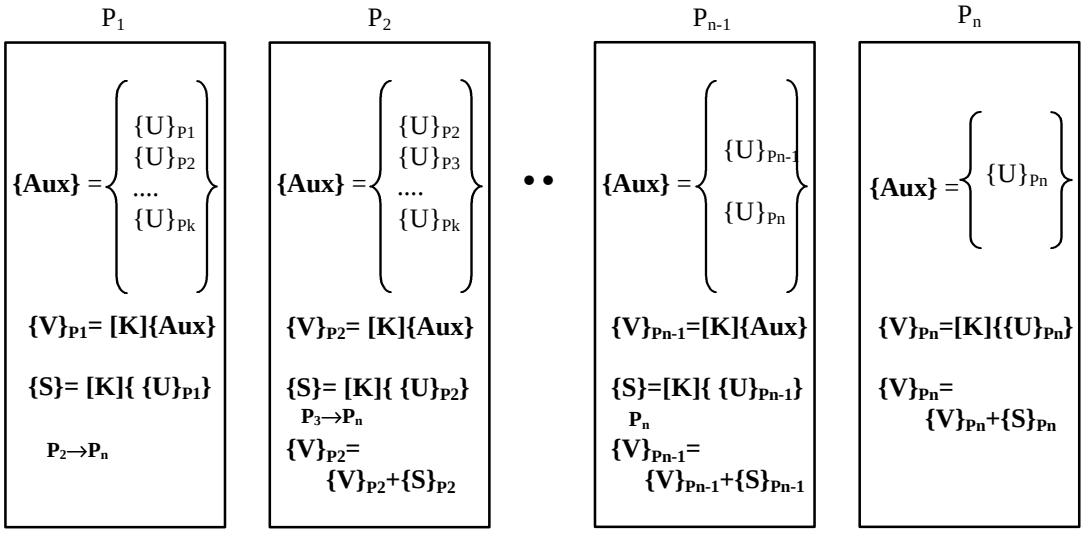
Ou seja, quanto maior o comprimento de semibanda, maior a influência entre linhas, portanto maior a troca de mensagens entre os diversos processadores em processo.

Na figura (4.4), apresenta-se esquematicamente o procedimento em forma de pseudocódigo da metodologia empregada para se fazer o produto matriz-vetor com a matriz em semibanda. Resumidamente, tal procedimento é explicado como:

- Para cada processador em cada passo, gera-se um vetor auxiliar (**{Aux}**) que contenha as características do deslocamento (**{U}**), que é referente aos valores das posições armazenadas em sua memória local e as posições que estão nos processadores adjacentes que influenciam em seu produto local. Desta forma é necessário fazer troca de mensagem destes valores não conhecidos por cada processador, obviamente o último processador não precisa receber estes dados, pois os valores necessários já estão armazenados em sua própria memória;

- Para cada processador, obtêm-se os valores do vetor (**{V}**) resultante do produto local, os quais foram gerados pelo próprio processo. Cria-se um outro vetor auxiliar (**{S}**) que armazena os valores de um produto local, advindo da simetria, mas, que devem ser conhecidos pelos processadores adjacentes;

- Finalmente, em cada processador - menos o primeiro – somam-se os seus valores resultantes locais com os produtos gerados pela simetria, ou seja o vetor (**{S}**), que foram obtidos pelos outros processadores adjacentes.



$P_i \rightarrow P_j$: processador i envia para os processadores (i+1) até j

FIGURA 4.4 – Procedimento esquemático do algoritmo para o produto matriz-vetor

4.3.2 Produto Interno

Conforme apresentado pelo algoritmo da figura (4.2), verifica-se a necessidade de se realizar um produto interno do tipo $a = \{u^T\} \cdot \{v\} = \sum_{i=1}^{n} \{u_i\} \cdot \{v_i\}$, onde $\{u\}$ e $\{v\}$ são vetores distribuídos dentro dos diversos processadores. Sem perda de generalidade, assume-se que cada processador armazena n/p elementos, onde n é o total de graus de liberdade da estrutura e p é número de processadores.

Este procedimento computacional em paralelo pode ser dividido em três partes:

1. A computação local de cada processador é dada por $b_j = \sum_{i=1}^{n/p} \{u_i\} \cdot \{v_i\}$ para cada processador;
2. A acumulação dos valores de b_j é feita através da seguinte forma apresentada na figura (4.5), com um exemplo com 7 processadores em grupo:

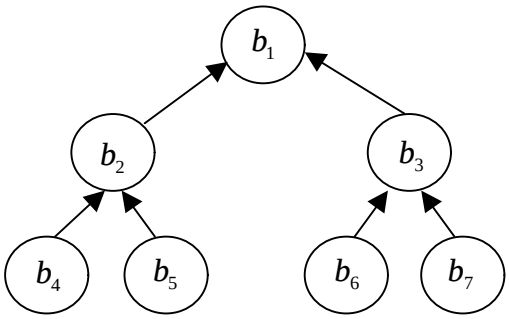


FIGURA 4.5 – Modelo de transferência de dados em árvore

e no final de tal procedimento, obtêm-se:

$a = \sum_{j=1}^{p} b_j$ e armazena-se no processo pai.

3. Transmite-se, assim, para todos os processadores o valor de a .

4.4 Exemplos de instabilidade do método GC frente ao número de condição

Para mostrar a sensibilidade do método dos gradientes conjugados em relação aos diferentes autovalores gerados da matriz de rigidez, é mostrado então dois exemplos de diferentes modelos estruturais que foram implementados para a resolução de seus sistemas com o método dos gradientes conjugados sem pré-condicionador.

Assim, tem-se aqui o intuito de mostrar a sensibilidade mostrada pela equação (4.9) em relação a variação do número de condição (κ) ou ao vetor inicial $\{U_0\}$ perante a convergência do método.

Ressalta-se que estes exemplos foram implementados em processamento seqüencial, já que eles tem o objetivo apenas de mostrar as características do método dos GC em relação aos diferentes valores físicos e geométricos destas estruturais, mostrando que mesmo para um tipo de estrutura com a mesma geometria, a convergência pode ser mais ou menos eficiente .

4.4.1 Placa quadrada com a técnica do número grande

Para o primeiro exemplo, aplicou-se ao modelo como condição de contorno o método do número grande.

Deste modo, implementou-se o programa utilizando esta forma de restrição e aplicou-se diferentes valores simulando a rigidez “infinita” sobre o ponto de restrição, onde este valor será representado pela letra K^*. Assim, sem alterar os demais valores da matriz de rigidez da estrutura, tem-se que os autovalores gerados nesta matriz serão afetados apenas pela imposição da rigidez “infinita” no (s) grau (s) de liberdade imposto (s). Com isso, quanto maior esta rigidez, maior será o número de condição e, conforme a equação (4.9), a convergência nos método dos gradientes conjugados fica prejudicada.

Implementou-se, em processamento em série, para maior compreensão deste fenômeno matemático, dois modos de se resolver o sistema com o método do número grande: o primeiro, usando o método direto de Gauss e o segundo, o método dos gradientes conjugados.

A tabela (4.1) apresenta os valores encontrados do deslocamento transversal do meio da placa apoiada e, a tabela (4.2), apresenta os deslocamentos para a placa engastada.

A geometria da placa é apresentada na figura (4.5), possuindo 512 elementos em uma única direção com os seguintes parâmetros com unidades compatíveis :

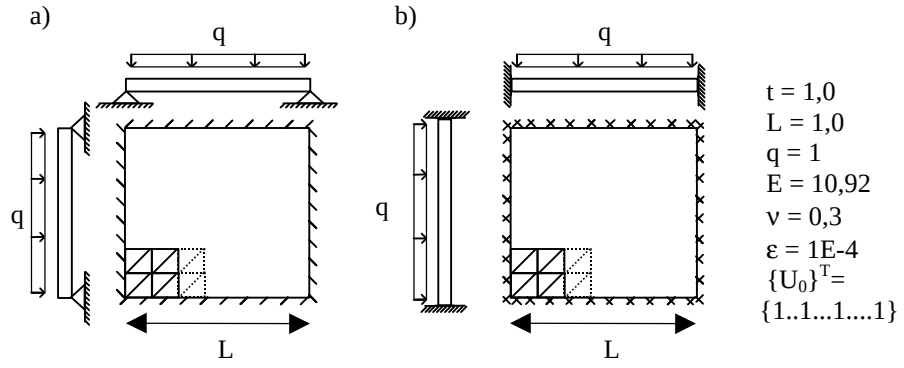


FIGURA 4.6 – Placas analisadas com suas características e os critérios do MGC

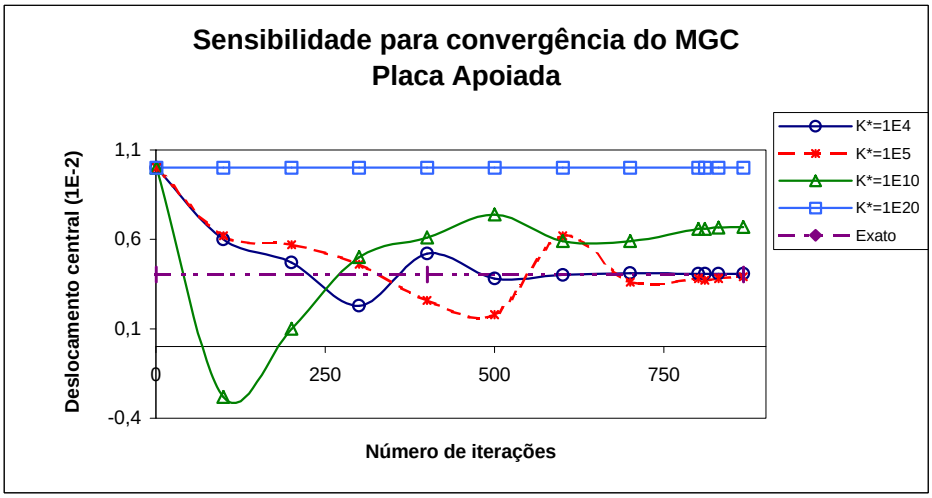


FIGURA 4.7 – Convergência p/ a placa apoiada (a) c/ diferentes valores de K^*

Tabela 4.1 Deslocamento central obtido para placa apoiada (a) em todo o bordo.

Número Grande		*K*= 1e4*	*K*= 1e10*	*K*= 1e15*	*K*= 1e20*
Desloca Mento (w) Central	W_{GC}	***0,004067***	***0,066820***	***1,000000***	***1,000000***
	W_{Gauss}	***0,004053***	***0,004053***	***0,004052***	***0,004053***

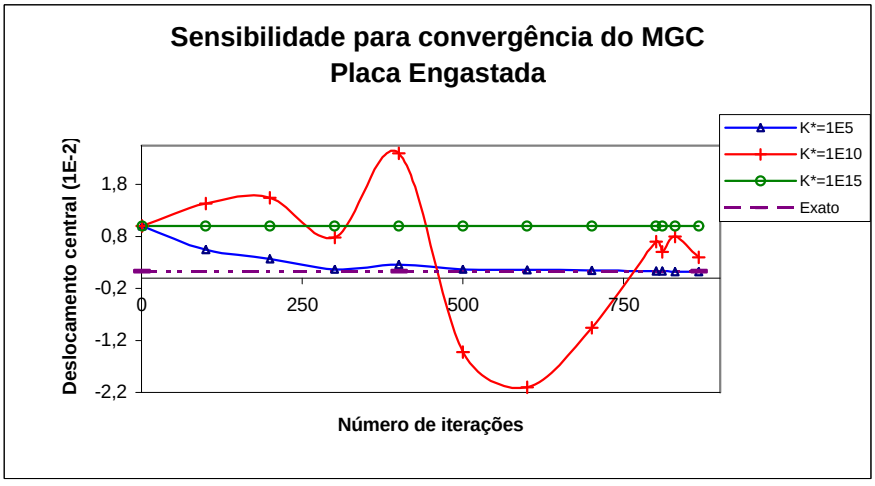


FIGURA 4.8 – Convergência p/ a placa engastada (b) c/ diferentes valores de K^*

Tabela 4.2 Deslocamento obtido para placa engastada (b) em todo o bordo.

Número Grande		$K^* = 1e5$	$K^* = 1e10$	$K^* = 1e15$	$K^* = 1e20$
Desloca Mento (w) central	W_{GC}	***0,001280***	***0,137342***	***1,000000***	***1,000000***
	W_{Gauss}	***0,001275***	***0,001275***	***0,001275***	***0,001275***

Mostra-se pelas tabelas (4.1) e (4.2), que o método de Gauss (e possivelmente qualquer uma de seus variantes) é bem mais estável com relação ao aumento do número de condição, do que o método dos gradientes conjugados, não aconselhando utilizar esta forma de restrição para a análise do sistema via método dos gradientes conjugados. Em DEMMEL (1993) é provado a independência do número de condição da matriz em relação a resolução pelo método direto.

4.4.2 Chapa retangular de lados paralelos

Este exemplo esquematizado na figura (4.9) é apresentado em SAVASSI (1996). O elemento finito utilizado é um elemento triangular com aproximação linear para os deslocamentos, conhecido como CST.

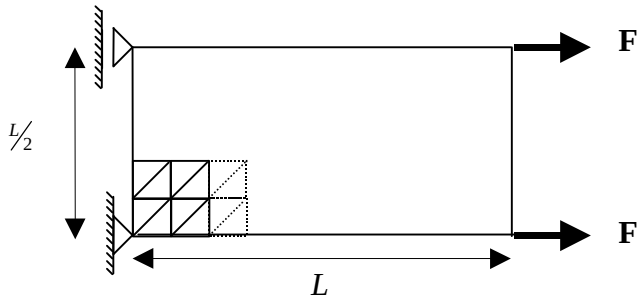
Assim, foi discretizado a rede em malhas com a orientação e as características indicadas a seguir.

Esse exemplo tem o intuito de mostrar a sensibilidade do método dos gradientes conjugados perante a variação do número de condição e a variação do vetor inicial ($\{U_0\}$).

Para se levar em condição a variação do número de condição, alterou-se a ordem de grandeza dos valores envolvidos na matriz de rigidez com a análise de dois diferentes modelos para a chapa, apresentados na figura (4.9).

A estimativa do vetor inicial foi feita para 3 casos com uma aproximação do vetor com ordem de grandeza de 10, assim estimou-se em 0,1, 0, 10 os valores de $\{U_0\}$.

Também, variou-se o valor da precisão para o critério de parada adotado, sendo que este foi a norma euclidiana sobre o vetor resíduo. As figuras (4.10) e (4.11) mostram os valores de convergência para os dois casos considerados.



Modelo 1

E = 1,0
ν = 0,25
L = 4,0
F = 1,0
t = 1,0

Modelo 2

E = 21000 kN/cm^2
ν = 0,30
L = 400,0 cm
F = 100 kN
t = 2,5 cm

FIGURA 4.9 – Modelo da chapa analisada, orientação e caracteristicas adotadas

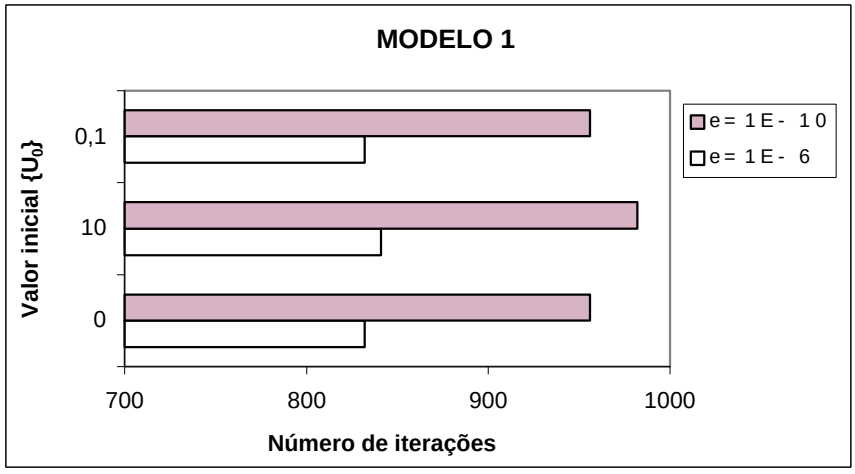


FIGURA 4.10 - Variação de convergência do modelo 1 de chapa

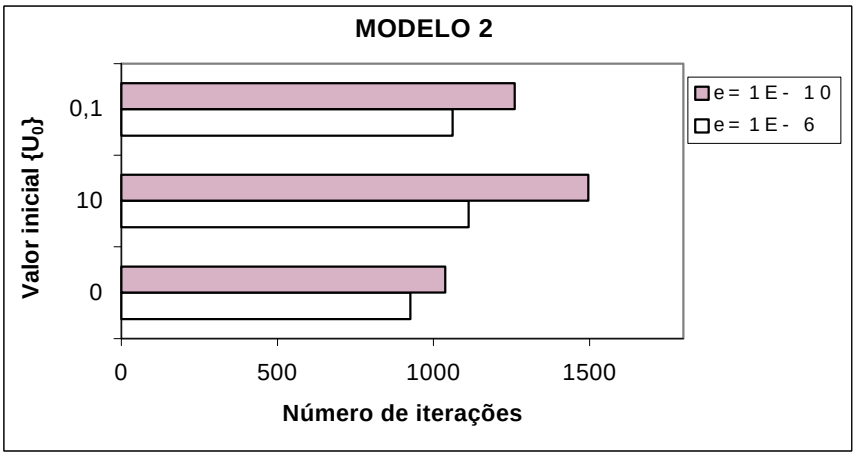


FIGURA 4.11 - Variação de convergência do modelo 2 de chapa

Como é mostrado nas figuras (4.10) e (4.11), a variação aleatória do valor inicial $\{U_0\}$ altera muito pouco a convergência do método, ficando esta discrepância em torno de 5% para o mesmo modelo comparado.

Nota-se que a variação do número de condição (κ), como é mostrado pela equação (4.9), mediante o emprego de valores diferentes de entrada em um mesmo modelo geométrico leva a uma maior variação na convergência, sendo que esta variação foi em torno de 25% comparando-se com a mesma precisão e com o mesmo vetor inicial.

Nestes exemplos, por serem de pequeno porte, essa variação parece pouco significativa, mas conforme o aumento dos graus de liberdade, leva a uma diferença relevante.

Capítulo 5

PRÉ-CONDICIONADORES

5.1 INTRODUÇÃO

Como fora mostrado no capítulo 4 pela equação (4.9), uma forma de se chegar a uma convergência mais rápida pelo método dos gradientes conjugados é tentar diminuir o número de condição (κ) da matriz de rigidez [K].

Tal objetivo é alcançado transformando o sistema dado pela equação (4.1) em um sistema equivalente, porém melhor condicionado, de modo que os autovalores da matriz do novo sistema estejam o mais agrupado possível. Isto faz com que a diferença entre o maior e o menor autovalor se aproximem mais, e assim, o seu quociente - que é o valor de (κ) – fiquem mais próximo de 1. Dessa maneira, o sistema alcança convergência mais rápido que o sistema originário, conforme apresentado na equação (4.9).

Assim, a técnica do pré-condicionamento consiste em transformar a equação (4.1) em um sistema equivalente do tipo:

$$[\tilde{K}]\cdot[\tilde{U}] = [\tilde{F}] \tag{5.1}$$

e o efeito do mau-condicionamento da matriz [K] é diminuído conforme a maior ou menor eficácia do pré-condicionador utilizado.

Na literatura existem muitas formas de se transformar o sistema da equação (4.1) mau-condicionado em um sistema bem-condicionado. Para isto, inúmeros pré-

condicionadores têm sido propostos e, basicamente, todos tem a mesma finalidade: obter um número de condição o mais próximo do valor unitário, sem perda de eficiência e sem causar instabilidade para resolução do sistema. Por exemplo, transformando a matriz do sistema linear equivalente em não positiva definida.

Para que isto não ocorra, as características da matriz pré-condicionadora [M] tem quer ser as mesmas que a da matriz de rigidez [K], ou seja, simétrica e positiva-definida, para garantir a equivalência de sistemas e com isso as mesmas propriedades necessárias para o método dos gradientes conjugados. Além disso, esta matriz [M] tem que ser uma boa aproximação da matriz [K], mas que tenha uma estrutura tal que seja fácil obter a sua inversa, pois, na prática, a idéia do pré-condicionamento consiste em se obter uma aproximação eficiente de $[K]^{-1}$.

Em resumo, a técnica de pré-condicionamento consiste em se pré-multiplicar o sistema dado pela equação (4.1), de modo que a matriz gerada pelo produto $[\tilde{K}] = [M]^{-1} \cdot [K]$ seja melhor condicionada que a matriz original [K]. Isto é possível fazendo com que a matriz $[M]^{-1}$ se aproxime de $[K]^{-1}$. Assim, o pré-condicionador tem o objetivo de transformar a matriz de rigidez [K] em uma matriz $[\tilde{K}]$ que seja a mais próxima da matriz identidade [I] e, desta forma, bem condicionada.

Ou seja, do sistema inicial dado pela equação (4.1), pré-multiplica-se pela inversa da matriz pré-condicionadora:

$$([M]^{-1} \cdot [K]) \cdot \{U\} = [M]^{-1} \cdot \{F\} \tag{5.2}$$

a matriz de rigidez alterada ($[\tilde{K}]$) melhor condicionada é dada por:

$$[\tilde{K}] = [M]^{-1} \cdot [K] \tag{5.3}$$

de modo que quanto mais $[M]^{-1}$ se aproxime de $[K]^{-1}$, tem-se a relação

$$\kappa(\tilde{K}) << \kappa(K) \quad \text{e} \quad \kappa(\tilde{K}) \to 1 \tag{5.4}$$

fica-se, então, no limite, quando $[M]^{-1} = [K]^{-1}$, a convergência do método torna-se direta e a iteração (*ite*) é imediata, ou seja, na representação em forma de limite:

$$\lim_{M^{-1}\to K^{-1}} k(\tilde{K}) = 1 \quad \text{ou} \quad \lim_{M^{-1}\to K^{-1}} ite = 1 \tag{5.5}$$

esquematicamente, resume-se a equação (5.5) na figura (5.1).

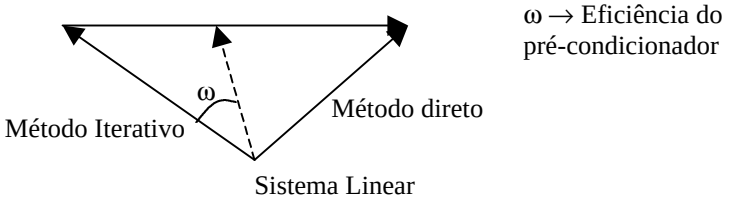


FIGURA 5.1 –Eficiência do pré-condicionamento sobre o sistema

A figura (5.1) compara, de maneira simplificada, as direções tomada pelos métodos diretos e iterativos para a solução do sistema de um determinado problema. Enquanto o método direto já parte para a direção da solução correta, o método iterativo toma diferentes direções que vão convergindo para a direção procurada. Mas, em geral, é lento o encontro desta direção.

Nesta figura é mostrado também que, conforme a eficiência do pré-condicionador, a direção procurada é alcançada mais ou menos rápida. Se o pré-condicionador for extremamente eficiente, o sistema equivalente, dado pela equação (5.1), se transforma em um sistema do tipo $\{U\}=[K]^{-1}\cdot\{F\}$ e a convergência é direta.

Mas para se obter a matriz inversa exata de [K] tem-se um esforço computacional muito maior que o necessário para se resolver o sistema sem o uso de pré-condicionador. Assim, a estratégia numérica buscada no estudo destes pré-condicionadores é a obtenção de uma matriz especial que seja a mais próxima possível da inversa de [K], contudo, com um reduzido esforço computacional para a sua montagem e a garantia da eficácia para a convergência do sistema equivalente.

Inúmeras são as técnicas existentes para se conseguir um eficiente pré-condicionamento. Mas, sabe-se que uma técnica pode ser eficiente para um dado conjunto de problemas, mas ineficientes para outros, ou então, uma mesma técnica sobre um mesmo problema, por exemplo, apenas se variando as condições de contorno ou a ordem de grandeza dos parâmetros do material empregado na estrutura analisada, pode não se apresentar tão eficiente quanto se encontrou neste mesmo exemplo.

Enfim, o estudo dos pré-condicionadores, por si só, é alvo de intensos estudos no meio acadêmico e em algumas aplicações industriais, sendo que para o meio acadêmico cita-se o trabalho de SCHMIT & LAI (1994) e para o campo industrial pode ser averiguado aplicações em JIANG *et al.* (1997).

Ressalta-se que para se aprofundar neste estudo e, principalmente, encontrar um pré-condicionador que se adeqüe e otimize *genericamente* os problemas advindos da mecânica dos sólidos é, no mínimo, prematuro. Portanto, este trabalho tem apenas o intuito de se estudar e apresentar 2 pré-condicionadores e aplicá-los a alguns exemplos estruturais, sem generalizar sua eficiência no campo do MEF.

5.2 ALGORITMO DO MÉTODO GC COM PRÉ-CONDICIONAMENTO

A adaptação do pseudo-algoritmo apresentado pela figura (4.2) do método dos gradientes conjugados puro, considerando-se o efeito do pré-condicionamento, pode ser feita de modo que, ao invés de se aplicar o pré-condicionador na equação (4.1), como mostrado na equação (5.2), realiza-se a contribuição da matriz pré-condicionadora ao vetor resíduo $\{r_p\}$. Esta é dada pela equação (4.4), já que este vetor é a diferença entre o vetor F e o vetor resultante do produto de $[K]\cdot\{U\}$, ou seja, lembrando-se que o vetor $\{r_p\}$ é dado por:

$$\{r_p\} = \{F\} - [K]\cdot\{U\} \tag{5.6}$$

da equação (5.2) aplicada em (5.6), tira-se um vetor $\{\tilde{r}_p\}$ já pré-condicionado, da seguinte forma:

$$\{\tilde{r}_p\} = [M]^{-1} \cdot \{F\} - [M]^{-1} \cdot [K] \cdot \{U\} \tag{5.7}$$

Ou, aproveitando-se o vetor $\{r_p\}$, dado pela equação (5.6), implicitamente, pode se escrever a equação (5.7) da seguinte maneira:

$$\{\tilde{r}_p\} = [M]^{-1} \cdot \{r_p\} \tag{5.8}$$

Este procedimento é também conhecido na literatura como método implícito dos gradientes conjugados. Assim, a figura (5.2) apresenta o pseudo-algoritmo implícito para a implementação com um pré-condicionador genérico.

→ **r_0 = F - K U_0**
→ montar $[M]^{-1}$ → $z_0 = [M]^{-1}. r_0$
→ **p_0 = z_0 / j = 0**
→ Enquanto **(algum critério de parada)**
→ $a_j = \dfrac{r_j^T \cdot z_j}{p_j^T \cdot K \cdot p_j}$
→ **U_{j+1} = U_j + α_j p_j**
→ **r_{j+1} = r_j - α_j K p_j**
→ $z_{j+1} = [M]^{-1}. r_{j+1}$
→ $b_j = \dfrac{r_{j+1}^T \cdot z_{j+1}}{r_j^T \cdot z_j}$
→ **p_{j+1} = z_{j+1} + β_{j+1} p_j**
→ **j =j + 1**
→ Fim Enquanto

FIGURA 5.2 – Pseudo-algoritmo do método GC com pré-condicionador

Foram estudados e implementados neste trabalho 2 tipos de pré-condicionadores: *decomposição incompleta de Cholesky*, que também é conhecido pela sua sigla em inglês *IC* (Incomplete Cholesky), e o advindo de uma *série polinomial truncada*, conhecido na literatura como *POLY*.

Para o primeiro modelo (IC), tem-se os seguintes trabalhos que podem ser consultados: MEIJERINK & van der VORST (1977), CIMERMAN (1996), SCHMIT & LAI (1994) e MANTEUFFEL (1980). Para o segundo: SCHMIT & LAI (1994), DUBOIS *et al*. (1979) e JOHNSON *et al*. (1983).

Esclarece-se que o pré-condicionador advindo da técnica de *decomposição incompleta de Cholesky* (IC) deve ser implementado tanto em *processamento seqüencial* como em *paralelo*, analisando assim sua eficiência com a variação do número de processadores. Portanto, verificando sua performance em ambientes de trocas de mensagens, além de se aplicá-lo a dois modelos estruturais: pavimentos e em treliças espaciais.

O pré-condicionador obtido pela *série polinomial* será implementado para análise em *código computacional seqüencial.*

No final deste capítulo serão mostrados alguns resultados de convergência, tanto em termos de tempo como em número de iterações para alguns modelos estruturais com os dois pré-condicionadores. Deixa-se claro que estas respostas tem o objetivo de mostrar a eficiência e comparar os dois pré-condicionadores em sistemas computacionais seqüenciais, mas que em processamento paralelo pode-se até inverter a eficiência destes em função das características próprias de cada técnica. Assim, como a implementação do pré-condicionador polinomial não será feita para processamento paralelo, os resultados que serão mostrados ao final deste capítulo, tém o intuito apenas de enriquecer o trabalho neste assunto, e mostrar qualitativamente a convergência dos dois métodos nos exemplos apresentados.

5.3 DECOMPOSIÇÃO INCOMPLETA DE CHOLESKY (IC)

Em MEIJERINK & van der VORST (1977) é primeiramente apresentado o método da decomposição (ou também fatorização) incompleta de Cholesky para aplicação com o método dos gradientes conjugados. No trabalho destes pesquisadores, é discutido a possibilidade do uso desta técnica para resoluções de sistemas lineares simétricos e positivo-definidos para o caso onde a matriz é do tipo M-matriz[1]. Além disso, é apresentado teoremas e provas que garantam a existência e a eficiência do método para o tipo de sistema analisado.

Em seguida, no trabalho de MANTEUFFEL (1980), é mostrado que a matriz [K] gerada pelo MEF, mesmo não tendo as características de M-matriz, pode estender esta técnica apresentada MEIJERINK & van der VORST (1977), e obter bons resultados de convergência para estes problemas.

5.3.1 Montagem do pré-condicionador Incompleto de Cholesky

O método da decomposição incompleta de Cholesky parte da idéia de se obter a matriz pré-condicionadora pela fatorização de [K] em uma decomposição de Cholesky, ou seja, fatorando a matriz [K] sob a forma:

$$[K] = [L] \cdot [L^T] \tag{5.9}$$

onde [L] é uma matriz triangular inferior de dimensão ($n \cdot n$), tal que [L] = (l_{ij}) e (l_{ij}) = 0 se i<j e [L^T] é a sua transposta.

Esta decomposição, apresentada anteriormente, nada mais é do que a conhecida decomposição-LU, sendo que esta técnica é gerada pelo método da eliminação de Gauss, que pode ser melhor entendido em BATHE (1982).

[1]Conforme MEIJERINK & van der VORST (1977), uma matriz [K]=(k_{ij}) é uma M-matriz se $k_{ij} \leq 0$ p/ $i \neq j$, [K] sendo não-singular e $[K^{-1}] \geq [0]$, sendo que estas matrizes são geradas freqüentemente pela discretização de equações diferenciais elípticas e parabólicas.

Porém, fatorar a matriz [K] em [L] ou [L^T] faz com que estas duas matrizes triangulares não sejam tão esparsas dentro do campo dado pelo comprimento em banda quanto a matriz [K]. Isto é um "gargalo" para a resolução numérica do método, já que há a necessidade de se armazenar estas matrizes cheias, ou seja, $(n \cdot m)$, acarretando mais uso de memória e maior quantidade de dados a serem manipulados em seu processamento.

Desta forma, perante este "gargalo", o método da decomposição *incompleta* se baseia em se fazer uma fatorização de [L] ou [L^T] de [K], de maneira que *alguns elementos são negligenciados* nas matrizes [L] e [L^T] em *posições apropriadas*. Estas posições são escolhidas de modo que se possa manter a esparsidade de [K] sobre as matrizes triangulares. Por isso, o método é denominado de *decomposição incompleta de Cholesky*.

Em MEIJERINK & van der VORST (1977), é apresentado os teoremas que garantem a existência e a estabilidade numérica desta decomposição incompleta.

Nestas matrizes triangulares, os valores negligenciados podem ocorrer em lugares arbitrários[2] (mas fora da diagonal principal) e as posições (i,j) não nulas serão indicadas por um conjunto P que é descrito a seguir:

$$P \subset P_N \equiv \{ (i , j) \mid i \neq j,\ 1 \leq i \leq n,\ 1 \leq j \leq n \} \tag{5.10}$$

onde P_N contêm todos os pares de índices de entradas da matriz.

Desta forma, pode-se escolher um campo de preenchimento das matrizes [L] ou [L^T], indicado por P, onde o par (i,j) indicará as posições de todos os valores diferentes de zero.

Por exemplo, seja o campo dado pelo conjunto:

$$P^1 \equiv \{ (i , j) \mid |i - j| \neq 0, 1, 2 \} \quad (i, j = 1 \ldots n) \tag{5.11}$$

[2] Não tão arbitrários assim, pois dependendo da escolha destas posições, a matriz incompleta [L] pode não ser positiva-definida.

Sendo este campo esquematicamente representado pelas seguintes posições indicadas em P^1:

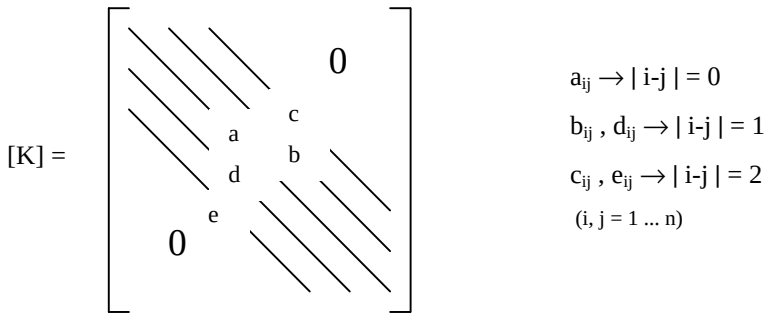


FIGURA 5.3 –Campo considerado de [K] para montar [L]

Assim, ao se montar as matrizes [L] ou $[L^T]$, apenas as posições dadas pelo conjunto P^1 serão consideradas e as demais serão anuladas.

Apresenta-se na figura (5.4), o pseudo-algoritmo conhecido na literatura – por exemplo em GOLUB & LOAN (1984) - que constrói a matriz triangular inferior [L] através da matriz original do sistema [K], sendo que este pseudo-algoritmo foi adaptado para se armazenar a matriz [L] em banda, onde **m** é o comprimento de semibanda.

Conforme CIMERMAN (1996), o algoritmo apresentado para se obter a matriz incompleta [L] pode falhar no momento de se calcular o valor do pivô, se este for negativo (raiz negativa). Isto pode ser superado adotando o primeiro valor de pivô positivo logo imediatamente acima da linha da posição do pivô. Isto costuma ocorrer, como já salientado, pois a matriz gerada pode não ser positiva-definida.

No trabalho de MEIJERINK & van der VORST (1977) é ressaltado que as posições não negligenciadas não precisam estar adjacentes, já que para a grande maioria dos problemas advindos da mecânica dos sólidos, tanto utilizando o método das diferenças finitas como o MEF, mostram que as influências entre os parâmetros da matriz não estão apenas próximas da diagonal principal. Em geral, as posições próximas da diagonal principal e da última diagonal não nula são preenchidas. Desta forma, em seu trabalho esclarece-se que é comum apenas relevar o conjunto P para estas posições.

```
→ Faça i = 1, n
  → Faça j = 1 , m
    → L(i,j) = K(i,j)
  →Fim de Faça
→Fim de Faça
Faça k=1, n
        L(k,1)= √L(k,1)
         Faça i =2, m
           L(k,i) =L(k,i) / L(k,1)
         Fim de Faça
         aux=1
         Faça j = k+1, n
                aux = aux+1
                Faça i=1, m
                 se (aux ≤ m)     a1=L(k,aux)
                 senão  a1=0
                 se (j-k+i > m)  a2=0
                 senão     a2 = L(k,j-k+i)
                 se (L(j,i) = 0)  L(j,i)=0
                 senão
                 L(j,i) =L(j,i) - a2 · a1
                Fim de faça
         Fim de faça
Fim de faça
```

FIGURA 5.4 – Pseudo-algoritmo da montagem de [L] no método IC

Para exemplificar, mostra-se aqui um exemplo onde não se negligencia as 3 primeiras posições e as 3 últimas não nulas da matriz do sistema. Assim, o conjunto P é expresso por:

$$P^3 \equiv \{ (i , j) \mid |i - j| \neq 0, 1, 2, m-3, m-2, m-1 \} \qquad (5.12)$$

ou seja:

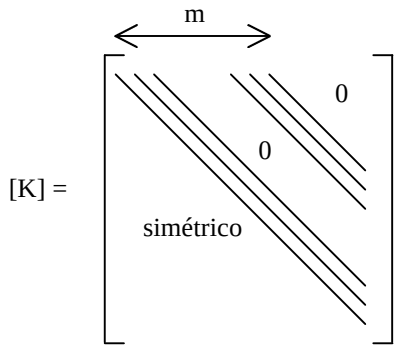


FIGURA 5.5 – Exemplo do campo de [K] sobre P^3

É comum referenciar-se ao método dos gradientes conjugados com o uso do pré-condicionamento da decomposição incompleta de Cholesky pela abreviação de seu nome em inglês (*Incomplete Cholesky Conjugate-Gradient)* de ICCG(η_b), onde η_b é o comprimento de banda referente ao número de diagonais extras, além das 3 diagonais determinadas pelo conjunto P^1 (equação (5.11)). Ou seja, a partir da diagonal principal as duas adjacentes incluindo ela. Assim, para o último exemplo apresentado, figura (5.5), sua notação seria ICCG(3).

5.3.2 Variações dos espectros sobre L

Para o presente trabalho, desenvolveu-se um algoritmo em que o espectro de influência sobre as posições não nulas na matriz [L] são variáveis de entrada do problema. Pode-se, assim, ser escolhido o comprimento de influência das diagonais a partir da diagonal principal, como também a partir da última diagonal não nula, conforme figura (5.6).

Desta forma, utiliza-se o pseudo-algoritmo mostrado na figura (5.4), adaptando-o para que as posições de influência determinada pelo campo de m_1 e m_2 sejam consideradas, negligenciando os valores fora deste espectro para se montar [L].

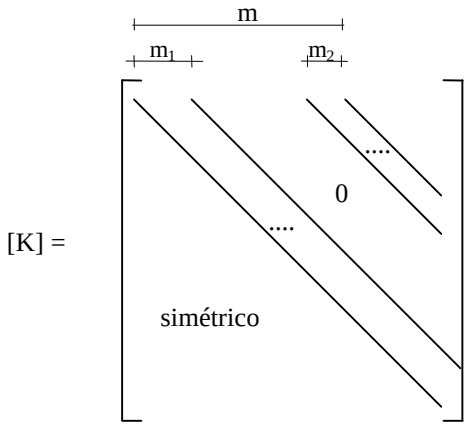


FIGURA 5.6 – Espectro de influência de m_1 e m_2 sobre [K]

Como os valores de m_1 e m_2 podem não ser iguais, neste trabalho não poderia ser utilizado a abreviatura apresentada em MEIJERINK & van der VORST (1977). Por isso, criou-se uma referência específica para a abreviação do método para se incluir os parâmetros m_1 e m_2.Como exemplo, considere um campo de P tal que m_1 = 4 e m_2 = 3. Assim, o conjunto P será denominado de $P_{m_1}^{m_2}$ e será dado por:

$$P_4^3 \equiv \{ (i , j) \mid |i - j| \neq 0, 1, 2,3,m-3,m-2,m-1 \} \qquad (5.13)$$

A notação para a utilização do método dos gradientes conjugados com este pré-condicionador com influência genérica, será indicada por ICCG(m_1,m_2) e, para este último exemplo, tem-se ICCG(4,3).

No programa desenvolvido, realiza-se o algoritmo de montagem da matriz pré-condicionadora com m_1 e m_2 como variável, destacando-se, também, que o tamanho a ser alocado para esta matriz [L] é de (n, m_1+m_2). Ou seja, neste algoritmo translada-se as posições influenciadas pelo espectro m_2, economizando assim memória e, principalmente, operações desnecessárias.

5.3.3 Vetor pré-condicionado

Conforme apresentado no pseudocódigo da figura (5.2), além de criar a matriz pré-condicionadora - neste caso a matriz [L] – é necessário, também, resolver um sistema do tipo:

$$[M]\cdot\{z\}=\{r\} \tag{5.14}$$

Para o método da decomposição incompleta de Cholesky, nota-se uma valiosa vantagem quanto a resolução deste sistema. Isto porque, ao invés de se resolver o sistema mostrado na equação (5.14), e pelo fato de se aproximar [M] de [K], pode-se, conforme equação (5.9), decompo-la da seguinte forma:

$$[L]\cdot[L^T]\cdot\{z\}=\{r\} \tag{5.15}$$

Como [L] e [L^T] são matrizes triangulares, respectivamente inferior e superior, pode-se aplicar o procedimento realizado no processo da eliminação de Gauss. Ou seja, obtêm-se primeiramente um vetor auxiliar {y} pela substituição direta de:

$$[L]\cdot\{y\}=\{r\} \tag{5.16}$$

Em seguida, faz-se a retrosubstituição com a transposta de [L], assim:

$$[L^T]\cdot\{z\}=\{y\} \tag{5.17}$$

Mostra-se, então, que realizar este procedimento é direto para a obtenção do vetor pré-condicionado {z}.

5.3.4 Paralelização do método IC

O método da decomposição incompleta de Cholesky, para implementação em ambiente em paralelo, não apresenta características vantajosas para este tipo de ambiente. Isto ocorre tanto para a montagem da matriz triangular inferior [L] como para se obter o vetor pré-condicionado.

No tocante a montagem da matriz [L], nota-se pelo pseudo-algoritmo da figura (5.4) que este é próprio para se realizar em programação seqüencial, já que a técnica em si é feita percorrendo-se *todas as linhas da matriz [K]* e, então, a partir de cada linha, que chamaremos de linha-base, altera-se as demais linhas e colunas abaixo desta linha–base, procedendo assim para cada variação da linha-base.

Assim, para os diversos processadores que têm domínio sobre as linhas que estão abaixo da linha-base, estes tem que esperar o processador que tem controle da linha-base realizar a instrução sobre sua linha. Em seguida, este processador envia esta linha para os processadores inferiores. Só então, estes podem realizar sua instrução.

Percebe-se, também, que se o (s) processador (es) tem (têm) domínio sobre as linhas que estão acima da linha-base, este (s) processador (es) tem que ficar ocioso (s) esperando o final do processo. A seguir, mostra-se de forma esquemática o procedimento discutido.

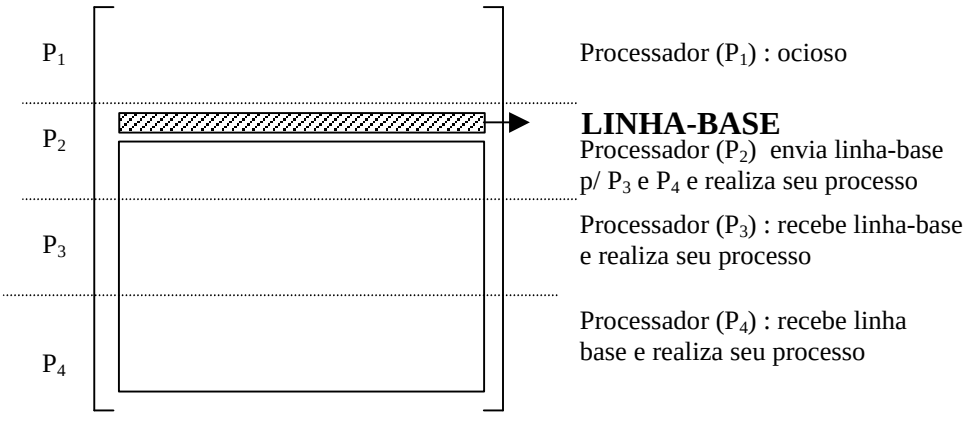


FIGURA 5.7 – Obtenção da matriz [L] em programação paralela

A resolução do sistema apresentado na equação (5.14) possui também características próprias para programação seqüencial. Pois, para se obter o vetor resposta {z}, realiza-se, respectivamente, uma substituição direta e uma retrosubstituição sobre o sistema (5.15), procedimentos estes que para a análise em paralelo ficam ineficientes.

Isto ocorre porque a matriz de rigidez e os vetores utilizados foram divididos em sub-blocos e cada sub-bloco armazenado em um processador diferente. Assim, cada processador tem que aguardar a realização das instruções sobre o processador adjacente, seja referente ao bloco superior para o caso da substituição direta ou sobre o bloco inferior para o caso da retrosubstituição. Demostra-se, assim, a ociosidade causada para a análise em paralelo também para este procedimento.

Esquematicamente, na figura (5.8), é apresentado o procedimento para resolução do sistema dado pela equação (5.14) e descrito anteriormente, sendo que (*np)* representa o número de processadores do sistema:

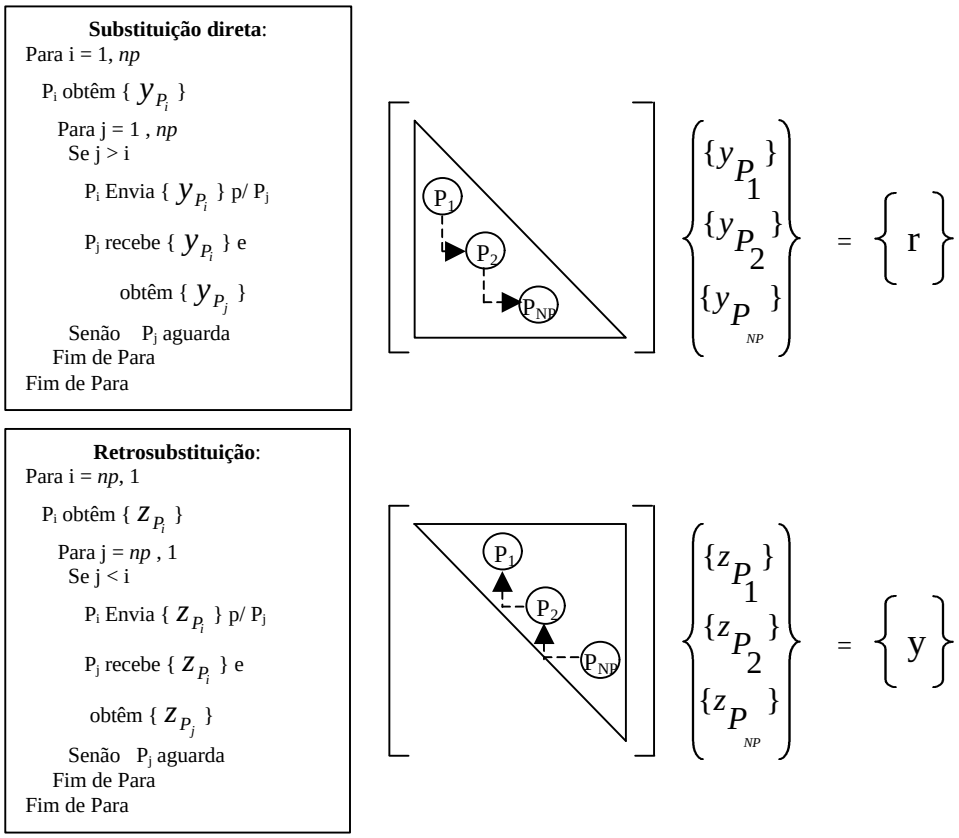


FIGURA 5.8 – Esquema para obter o vetor {z} pré-condicionado em paralelo

Destaca-se que no programa desenvolvido neste trabalho, na sub-rotina PRECONL, é explicitado o algoritmo apresentado na figura (5.8), ressaltando que no ponto onde se envia o vetor $\{y_i\}$ ou $\{z_i\}$ para os demais processadores, em função do espectro de influência do sub-bloco de um processador *i* em relação a um processador *j*, o envio deste sub-vetor pode não precisar ser feito, já que sua influência pode não existir.

Isto é importante principalmente para sistemas onde o comprimento de banda do sistema original é pequeno. Assim, a influência (se houver) dos processadores se torna entre processadores adjacentes, tornando o envio aos demais processadores desnecessário. Dessa maneira, quanto menor o comprimento de (*m*), menor a comunicação entre processadores e, desta forma, mais eficiente é o processamento paralelo para este procedimento

Por fim, em função do que foi apresentado, nota-se que a técnica de pré-condicionamento pelo ICCG pode parecer ineficiente na análise em processamento paralelo. Mas, conforme os trabalhos de MANTEUFFEL (1980) e SCHMIT & LAI (1994), se for empregado esta técnica com um pequeno campo tomado de *P*, por exemplo P_3^3, em função da rápida convergência desta técnica e com a baixa comunicação dada por este campo, é ainda uma técnica bastante empregada para o pré-condicionamento no método dos gradientes conjugados para a resolução do sistema originado pelo MEF.

5.4 PRÉ-CONDICIONADOR POLINOMIAL

O pré-condicionador polinomial é uma outra técnica clássica para se obter a aproximação da inversa da matriz [K], mediante uma aproximação para a inversa desta matriz com o uso de uma expansão polinomial em potências de [K].

Assim, como vantagem deste método quando comparado com o método da decomposição incompleta, é que nesta aproximação da inversa de [K], tem-se que realizar apenas um produto matriz-vetor para obter o vetor condicionado {z}, não tendo que aplicar a substituição e a retrosubstituição como no caso anterior. Em

DUBOIS *et al.* (1979) mostram-se todas as hipóteses e provas matemáticas necessárias para aplicar o método nos problemas aqui tratados.

Parte-se, inicialmente, da hipótese de que a matriz [K] tem que ser simétrica e positiva-definida e, então, pode-se reescrevê-la da seguinte forma:

$$[K] = [D_S] + [\bar{K}] \tag{5.18}$$

tal que [D_S] é a matriz formada apenas pelos elementos da diagonal principal de [K] e $[\bar{K}]$ é a matriz dos elementos de [K] com valores nulos em toda a diagonal principal.

Colocando em evidência a matriz diagonal [D_S] na equação (5.18), tem-se:

$$[K] = [D_S] \cdot ([I] + [D_S]^{-1} \cdot [\bar{K}]) \tag{5.19}$$

E, elevando os dois membros a (-1) na equação (5.19) e aplicando as propriedades matriciais, chega-se a :

$$[K]^{-1} = ([I] + [D_S]^{-1} \cdot [\bar{K}])^{-1} \cdot [D_S]^{-1} \tag{5.20}$$

E conforme DUBOIS *et al.* (1979) e JOHNSON *et al*. (1983), se a matriz [K] for *estritamente diagonal dominante*, pode se escrever a primeira parcela do segundo membro da equação (5.20) como uma série polinomial da seguinte maneira:

$$([I] + [D_S]^{-1} \cdot [\bar{K}])^{-1} = \sum_{J=0}^{\infty} \{(-1)^J \cdot ([D_S]^{-1} [\bar{K}])^J\} \tag{5.21}$$

onde [I] é a matriz identidade.

Ou seja, substituindo a equação (5.21) em (5.22), a aproximação da inversa de [K] é dada por:

$$[K]^{-1} = \{\sum_{J=0}^{\infty}\{(-1)^J \cdot ([D_S]^{-1}[\bar{K}])^J\}\}[D_S]^{-1} \tag{5.22}$$

O fato da matriz [K] ter que ser estritamente diagonal dominante é porque, sabendo-se que a aproximação em série da expressão

$$\frac{1}{1+x} \quad \text{tal que } (0 \leq x < 1) \tag{5.23}$$

é dada por

$$\frac{1}{1+x} \cong 1 - x + x^2 - x^3 + \cdots \tag{5.24}$$

Analogamente esta idéia, considerando-se as devidas diferenças, pode ser estendida para a aplicação em matrizes, fazendo-se:

$$[\tilde{X}] = \frac{[K]}{[D_S]} \tag{5.25}$$

onde $[\tilde{X}]$ é a matriz resultante do quociente entre [K] e [D_S]. Resulta-se, então, na aproximação polinomial

$$\frac{[I]}{[I]+[\tilde{X}]} \cong [I] - [\tilde{X}] + [\tilde{X}]^2 - [\tilde{X}]^3 + \cdots \tag{5.26}$$

que é equivalente a equação (5.21).

Se a matriz [K] é *estritamente diagonal dominante*, na matriz resultante $[\tilde{X}]$ pode ser mostrado que seu determinante é menor que 1. Assim, vale a aproximação e a série é convergente.

Na literatura, para a aplicação da série, é comum truncar o polinômio em, no máximo, potências de cinco. Comenta-se que valores acima destes é comum começar a ocorrer instabilidade no sistema tratado, conforme o aumento na ordem da potência.

Em SCHMIT & LAI (1994), por exemplo, faz-se aplicações com expansões de potências de ordem um e dois e, em geral, esta técnica de pré-condicionamento é denominada de POLY (n_T), tal que n_T é o números de potências retidas no truncamento da série.

Um pré-condicionador que é fácil de ser implementado tanto em ambiente em série como em paralelo é o POLY(0), que também é denominado de pré-condicionador Jacobi. Ele é dado pela matriz formada apenas pela diagonal principal onde os termos são calculados pelo inverso do valor da referida posição da matriz [K]. Ou seja:

$$[K]^{-1} \cong diag(1/k_{ii}) \quad i = 1, n \tag{5.27}$$

5.4.1 Pré-Condicionador Polinomial Incompleto

Neste trabalho, extrapolando o conceito da fatorização incompleta apresentada no item anterior, tentou-se então aplicá-la sobre a técnica da série polinomial.

Com isso, para cada aproximação da série pelo truncamento da potência, realizou-se também um *truncamento sobre o número de colunas* da matriz que está sendo multiplicada por ela mesma, ou seja, $[K]^J$. Monta-se, então, a aproximação pela série, considerando apenas as colunas que estejam mais próximas da diagonal principal, já que são estes elementos que tem valores mais relevantes para obtenção da inversa de [K].

Em função da possibilidade de se variar o espectro de influência sobre o produto da matriz por ela mesma, foi necessário desenvolver aqui uma outra notação para especificar o tipo de série polinomial e o número de diagonais (m_1), a partir da diagonal principal, que serão consideradas. Desta forma utilizando a notação

apresentada anteriormente incluiu-se apenas o número de diagonais, ou seja, a notação fica POLY (n_T , m_1).

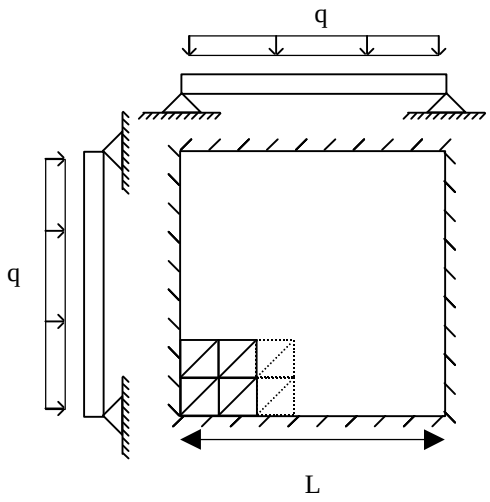
5.5 APLICAÇÕES

A seguir são apresentados alguns exemplos em que se comparam os dois métodos de pré-condicionamento, variando-se os parâmetros intrínsecos de cada método. Ou seja, variando os espectros da influência de m_1 e m_2 em ICCG e a influência ordem no número de potências consideradas na série e a quantidade de colunas (m_1) de POLY. Explica-se que quando forem consideradas todas as colunas para o POLY, se indicará a forma POLY(n_T , m), onde m representa o comprimento de semibanda.

Os resultados aqui apresentados foram obtidos em um microcomputador do tipo PENTIUM II 300 com 64 Mbytes e em **programação seqüencial**. Tais resultados tem o intuito apenas de comparar a eficiência entre os métodos com suas respectivas variações.

5.5.1 Placa quadrada simplesmente apoiada

O modelo aqui utilizado para verificar os dois métodos é o advindo de um problema do MEF aplicado a uma estrutura bidimensional de placa com elementos DKT, sendo que as características geométricas do material e o tipo de carregamento é apresentado a seguir. Também tem-se os resultados para o sistema sem o uso de pré-condicionadores, considerando a matriz identidade como pré-condicionador.



t = 1,0
L = 1,0
q = 1
E = 10,92
ν = 0,3
ε = 1E-4
$\{U_0\}^T = \{0..0...0....0\}$
norma euclidiana
aplicado ao resíduo

(Valores em unidades compatíveis)

FIGURA 5.9 – Esquema da placa, sua orientação e parâmetros de geometria e do material

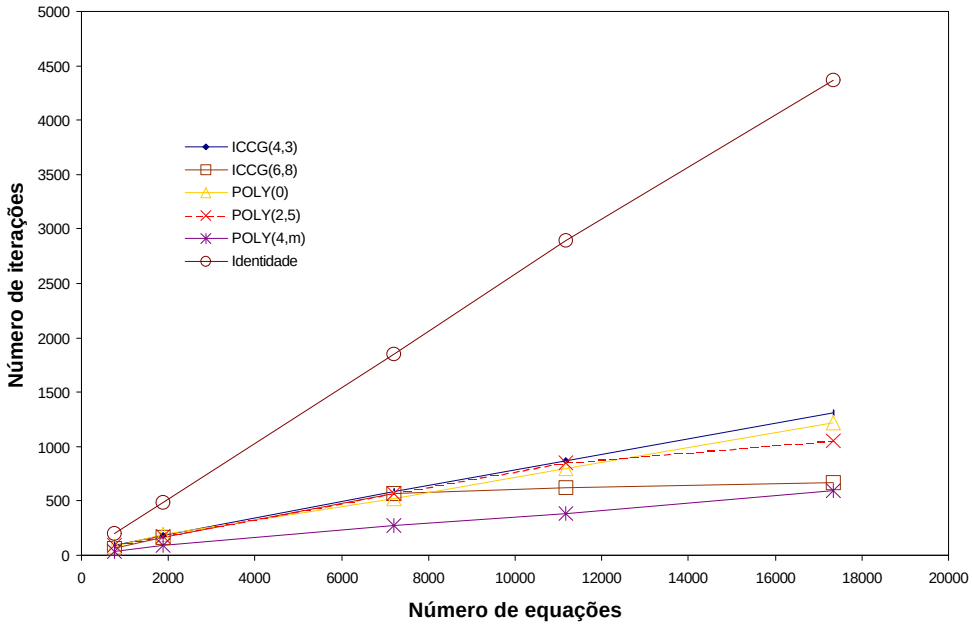


FIGURA 5.10 – Curva (N° de iterações x N° de equações) para a placa apoiada ao longo do bordo

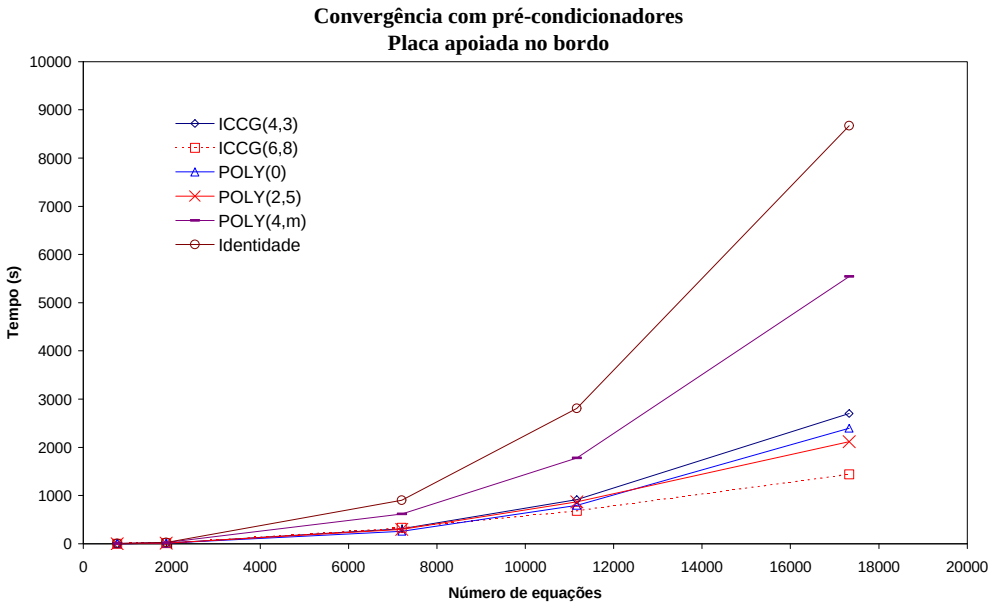


FIGURA 5.11 – Curva (tempo x N° de equações) para aplaca apoiada ao longo do bordo

Os valores plotados nas figuras (5.10) e (5.11) dos pré-condicionadores foram os que apresentaram variações significativas dentre uma gama de combinações analisadas, mais de 10 combinações para cada tipo.

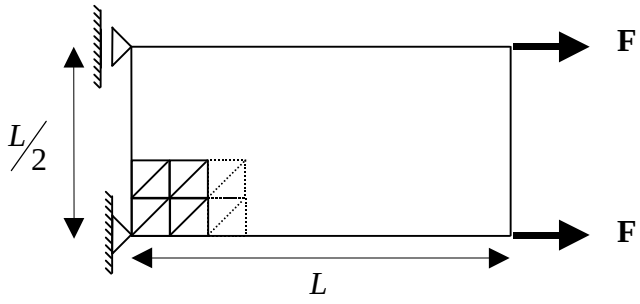
Destaca-se o uso do *ICCG*(6,8) , que apresentou uma alta convergência tanto em termos de número de iterações quanto em tempo.

O POLY(4,m), ou seja, o polinomial considerando-se todos os valores da matriz [K] na montagem implícita da inversa de [K], demostrou alta convergência para o número de iteração. Mas em função da grande quantidade de operações necessárias o seu tempo de resposta foi deficitário.

A adaptação adotada aqui para a montagem incompleta do POLY(2,5), mostrou-se eficaz neste exemplo, sendo efetiva – principalmente - em termos do baixo tempo necessário para convergência.

5.5.2 Chapa retangular com lados paralelos

Neste exemplo são apresentado os resultados de convergência, tanto em número de iterações quanto em tempo, de uma estrutura plana formada por elementos de chapa. O elemento finito utilizado para discretizar o modelo é o CST. Assim, na figura (5.12) é mostrado a estrutura discretizada, a orientação dos elementos e as características adotadas.



Modelo

E = 1,0
ν = 0,25
L = 4,0
F = 1,0
t = 1,0
ε = 1E-4
$\{U_0\}^T$= {0..0...0....0}
norma euclidiana aplicado ao resíduo

FIGURA 5.12 – Esquema da chapa, sua orientação e parâmetros de geometria e do material

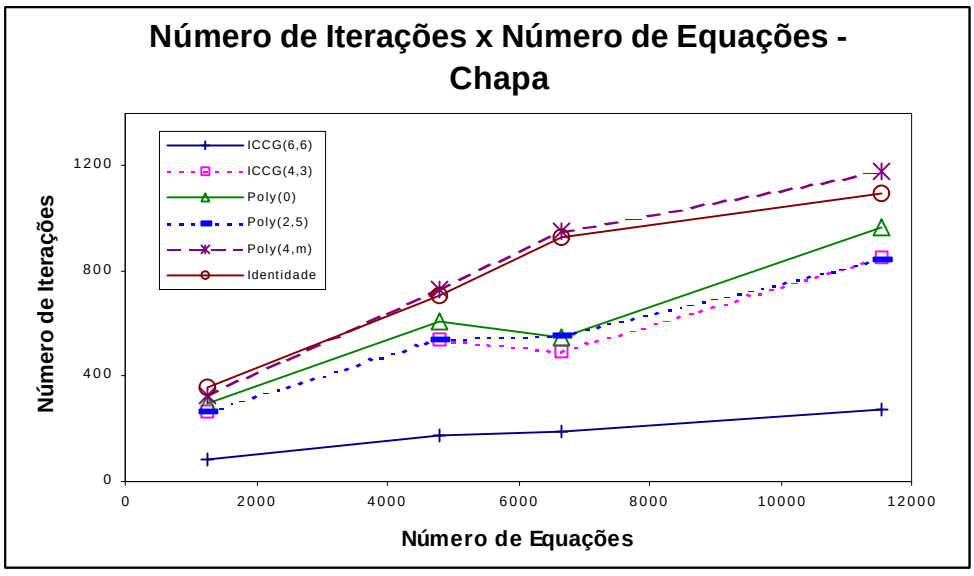


FIGURA 5.13 – Curva (N° de iterações x N° de equações) para a chapa

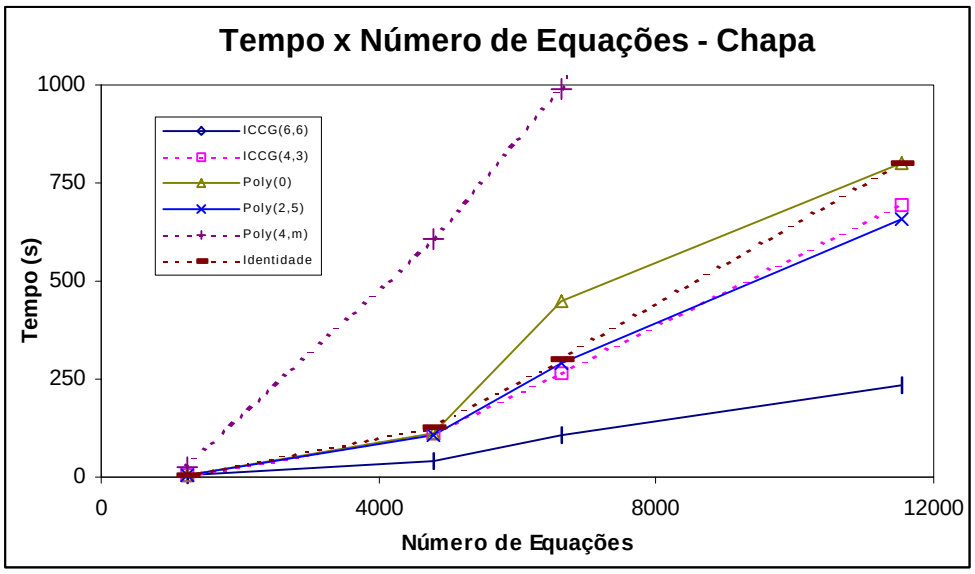


FIGURA 5.14 – Curva (tempo x N° de equações) para a chapa

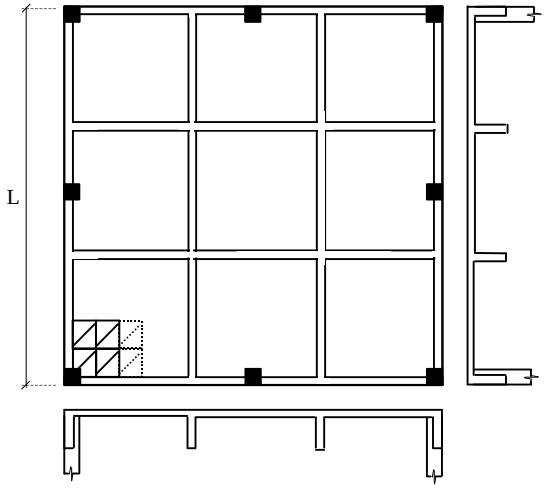
Para este exemplo, os resultados encontrados com o uso do pré-condicionador ICCG(6,6) mostrou ser bem mais efetivo que os encontrados pelos demais pré-condicionadores. O tempo de convergência com o uso do ICCG(6,6) foi aproximadamente 60% menor que o encontrado pelo segundo melhor pré-condicionador - Poly(2,5).

Neste exemplo, o Poly(0) mostrou-se pouco eficiente quando comparado com o exemplo anterior, apresentando nenhuma vantagem em relação ao método puro dos gradientes conjugados (sem pré-condicionador). O Poly(4,m) não revelou bons resultados de convergência nem em tempo e nem em número de iterações.

5.5.3 Pavimento quadrado

Neste exemplo apresenta-se uma estrutura de pavimento formada por elementos de placa DKT e por elementos de viga. Pretende-se neste item, comparar a eficiência dos pré-condicionadores (IC) e do Poly em relação a duas concepções de valores adotados.

Para o primeiro modelo do pavimento adotou-se valores simplificados, enquanto que para o segundo correspondem valores reais de um pavimento, utilizando concreto armado como material e uma carga típica de edifício. A configuração da estrutura, as condições de apoio, os parâmetros físicos e geométricos e o tipo de carregamento são esquematizados na figura (5.12).



Modelo 1
L = 1,0
I_{VIGA} = 5,0
H_{LAJE} = 1,0
$J_{torção}$ = 1E-4
E = 10,92
ν = 0,30
q = 1,0

(Valores em unidades compatíveis)

Modelo 2
L = 600,0 cm
I_{VIGA} = 104666,67 cm^4
H_{LAJE} = 7,0 cm
$J_{torção}$ = 1E-4 cm^4
E = 2500,0 kN/cm^2
ν = 0,25
q = 1E-3 kN/cm^2

FIGURA 5.15 –Orientação, parâmetros de geometria e de material para os modelos do pavimento

Assim, os resultados de convergência em termos de número de iterações e em termos de tempo são apresentados nas figuras (5.13) e (5.14).

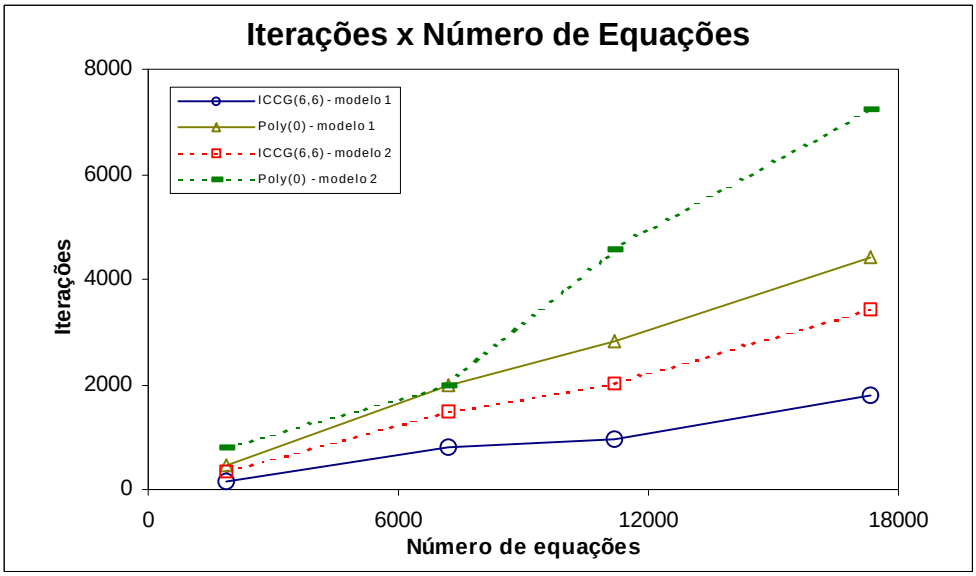


FIGURA 5.16 – Curva (N° de iteração N° de equações) para o pavimento

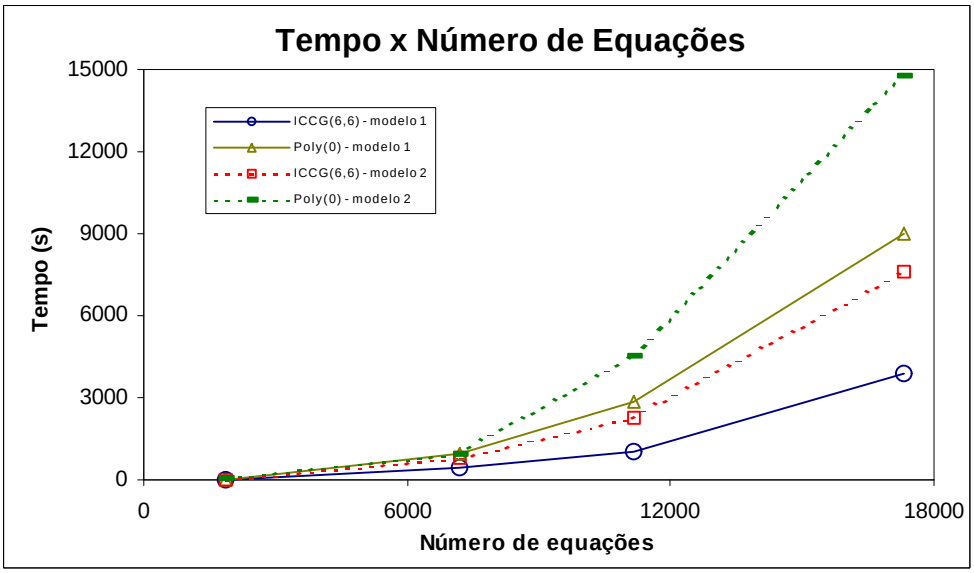


FIGURA 5.17 – Curva (tempo x N° de equações) para o pavimento

Para os casos da técnica da decomposição incompleta de Cholesky (IC) quando a escolha dos valores de m_1 eram menores que os valores de m_2, para a montagem do pré-condicionador tinha-se que realizar pivoteamento para evitar raiz quadrada negativa no algoritmo. Acredita-se que isto tenha ocorrido em virtude de que a matriz incompleta [L] montada no método não era positiva definida. Portanto, contrariando as hipóteses para a sua utilização. Assim, aconselha-se escolher apenas valores do (IC) em que $m_1 \geq m_2$.

Para os gráficos apresentados anteriormente não ficarem com muitas curvas para os dois modelos analisados, plotou-se apenas as curvas para ICCG(6,6) e Poly(0), já que estes dois pré-condicionadores serão avaliados nos exemplos de processamento em paralelo.

Mesmo para as duas diferentes concepções de valores adotadas para este pavimento, o pré-condicionador ICCG(6,6) mostrou ser bem mais eficiente do que o Poly(0), tanto em termos de tempo como em termos de iterações.

Mas, percebe-se que os valores adotados refletem diretamente a variação do número de condição nos dois modelos. Assim, na convergência pelo modelo 1, o

número de iterações necessárias fica em torno de 10% do número de incógnitas, enquanto que para o segundo caso este valor é da ordem de 20%.

Este valor percentual para análise em processamento paralelo tem significado extremamente relevante pois quanto maior o número de iterações, maior a dependência entre os vários processadores, e assim pode-se prejudicar a eficiência da análise deste segundo modelo em paralelo, principalmente usando-se o ICCG, já que este necessita de maior dependência dentre os diversos processadores no sistema.

Capítulo 6

APLICAÇÕES EM PROCESSAMENTO PARALELO

6.1 INTRODUÇÃO

Este capítulo apresenta dois exemplos de estruturas que foram analisadas em processamento paralelo: pavimento e treliça tridimensional.

Para o primeiro caso, faz-se uma adaptação completa do código sequencial deste modelo para computadores de arquitetura paralela. Assim, a ambientalização em paralelo começa a partir da leitura de dados até a obtenção dos esforços de cada elemento.

Para o algoritmo do modelo de treliça, realizou-se uma adaptação apenas para verificar o potencial do paralelismo na resolução do sistema, ponto este que é o mais importante para caracterizar o desempenho global do modelo em análise paralela.

Com os resultados de tempo conseguidos com a variação do número de processadores, fez-se assim a medida de desempenho de cada conjunto: número de processador e pré-condicionadores. Tal análise é necessária para mostrar a menor ou maior eficiência do uso do paralelismo nos dois exemplos com os pré-condicionadores utilizados.

Em suma, este capítulo foi dividido de tal modo que o item (6.2) mostra a forma convencional de se tirar as medidas de desempenho do código. Em seguida, apresentam-se dois exemplos de modelos estruturais, sendo que para cada exemplo onde se varia o número de elementos são mostrados gráficos de “speed-ups”, de eficiência e de tempo sob diversos pré-condicionadores.

No final de cada série de exemplos faz-se uma análise de resultados local.

Ressalta-se que os resultados são obtidos na máquina do CISC localizado no campus da USP em São Carlos, sendo um computador com arquitetura paralela do tipo IBM 9900/SP com 3 processadores com 256 Mbytes por nó, ver TUTORIAL do IBM (1998) e TUTORIAIS do CISC (1998).

6.2 MEDIDA DE DESEMPENHO

A medida usual de desempenho no processamento de programas ou de trechos paralelos é o fator conhecido como *Speed-up*. Este eqüivale a relação do quociente entre o tempo necessário para sua execução em programação sequencial e o tempo em processamento paralelo com *n* processadores.

Assim, esta relação é expressa por:

$$Speed - up = \frac{\text{Tempo com 1 processador}}{\text{Tempo com n processadores}} \tag{6.1}$$

Esta relação mostra que para valores acima de 1 tem-se um ganho na execução do processamento com *n* processadores. Aliada à medida de *speed-up*, tem-se a medida de eficiência do processo, a qual é dada pela relação

$$Eficiência = \frac{100 \cdot (Speed - up)}{\text{número de processadores}} \quad (\%) \tag{6.2}$$

Em AMDAHL (1967) é apresentado que o limite superior do *speed-up* é dado pela relação

$$Speed - up \leq \frac{1}{s + \frac{r}{n}} \tag{6.3}$$

de tal modo que r representa a fração em tempo do processo que é paralelizável dentro programa e s é dado por s = r-1.

Assim, por exemplo, onde o tempo de análise em paralelo é bem superior que o tempo sequencial, como no caso da resolução do sistema com o emprego do método dos gradientes conjugados, pode-se, sem perda de generalidade, afirmar que r = 1, ou seja, 100% do programa é paralelizável. Portanto, nestes casos o *Speed-up* é limitado por

$$(Speed - up) \leq n \tag{6.4}$$

O *Speed-up* como já salientado, é um medidor de desempenho do processamento que compara a performance do programa sequencial com sua versão em paralelo. Entretanto, conforme mostrado pela equação (6.4), simplificamente é estabelecido que seu valor esta limitado pelo número de processadores. E, em geral, o valor do *speed-up* ficará bem abaixo deste valor, podendo até ser um valor menor que 1, para um número reduzido de processadores.

Isto ocorre, principalmente, em virtude do "gargalo" surgido quando da necessidade de espera dos processadores para trocas de mensagens e da sincronização dos processadores dentro dos processos e/ou das instruções, principalmente para o caso aqui tratado, onde se tem poucos processadores para verificar os fatores de desempenho.

Para destacar esta perda de eficiência do processamento paralelo, em JIANG *et al. (1997)* obtêm-se um *speed-up* em torno de 10 para a análise em paralelo com 22 computadores em memória local de um sistema com a fração em paralelo bem próximo de 1. A eficiência obtida em seu trabalho foi de 45%.

6.3 TRELIÇA TRIDIMENSIONAL

Apresentam-se aqui os resultados de *speed-up*, eficiência e do tempo de processamento de cada rede variando-se os pré-condicionadores e o número de processadores.

Na figura (6.1) é mostrado o modelo esquemático da treliça gerada. Para este modelo variou-se o tamanho da rede com a utilização de um gerador próprio

desenvolvido. Nesta figura também é apresentado as características geométricas e físicas do material. Este exemplo foi extraído do trabalho de SOUZA (1998), bem como as características adotadas, sendo que aplicou-se uma carga concentrada no nó central da estrutura, no banzo superior, de valor indicado na tabela da figura (6.1).

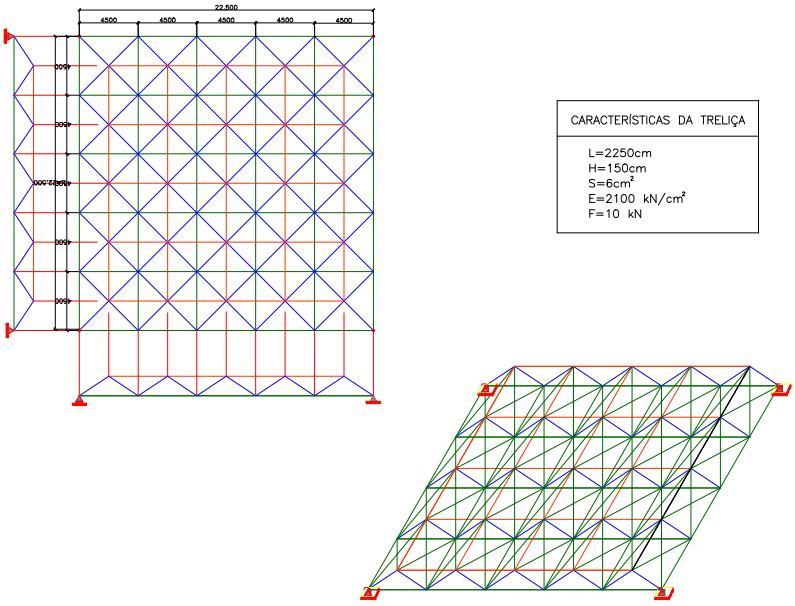


FIGURA 6.1 - Planta e perspectiva do modelo de treliça

Rede A: 19208 elementos e 4901 nós

Rede B: 63368 elementos e 16021 nós

Rede C: 95048 elementos e 23981 nós

A seguir, nas tabelas de (6.1) a (6.4), são apresentados os gráficos de *speed-up* por número de processadores e de eficiência por número de processadores.

Para estes exemplos, após uma análise com exemplos menores em processamento seqüencial, decidiu-se em verificar a convergência com 4 tipos diferentes de pré-condicionadores.

Para o caso dos pré-condicionadores advindo da decomposição incompleta de Cholesky, escolheu-se 3 pré-condicionadores: ICCG(6,8), ICCG(4,3) e ICCG(3,1).

O primeiro, o ICCG(6,8), se deu por este apresentar a melhor performance em processamento seqüencial para os exemplos averiguados.

A segunda escolha - ICCG(4,3) - se deu por este apresentar uma razoável eficiência para os exemplos, menor que a do ICCG(4,3), mas esperando-se que para sistemas maiores, este pudesse se destacar pelo fato de se realizar menos operações por iteração e pela menor quantidade de trocas de dados entre processadores.

A terceira escolha - ICCG(3,1) - foi feita, pois mesmo por este apresentar uma má convergência nos sistemas analisados, pode ser que em processamento paralelo sua performance possa se destacar em função de se realizar menos operações para cada iteração e a interdependência dos diversos processadores seja bem menor, comparado com os outros pré-condicionadores deste mesmo estilo.

Como já salientado, a técnica de pré-condicionamento pelo modelo polinomial não foi implementada em processamento paralelo. Mas, pelo fato de o POLY(0) apresentar bons resultados, ver figura (5.10) e (5.11), e, também, por ser de fácil implementação, este pré-condicionador foi utilizado para estes exemplos. Assim, a seguir apresenta-se os resultados de *speed-up* e de eficiência para os modelos *A* e *B* para a resolução do sistema em paralelo.

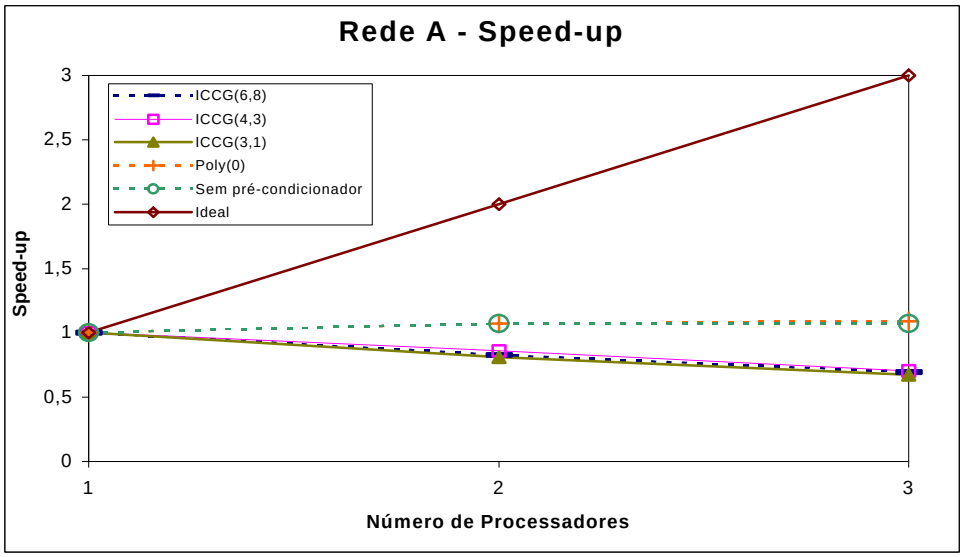


FIGURA 6.2 –Speed-up da rede A

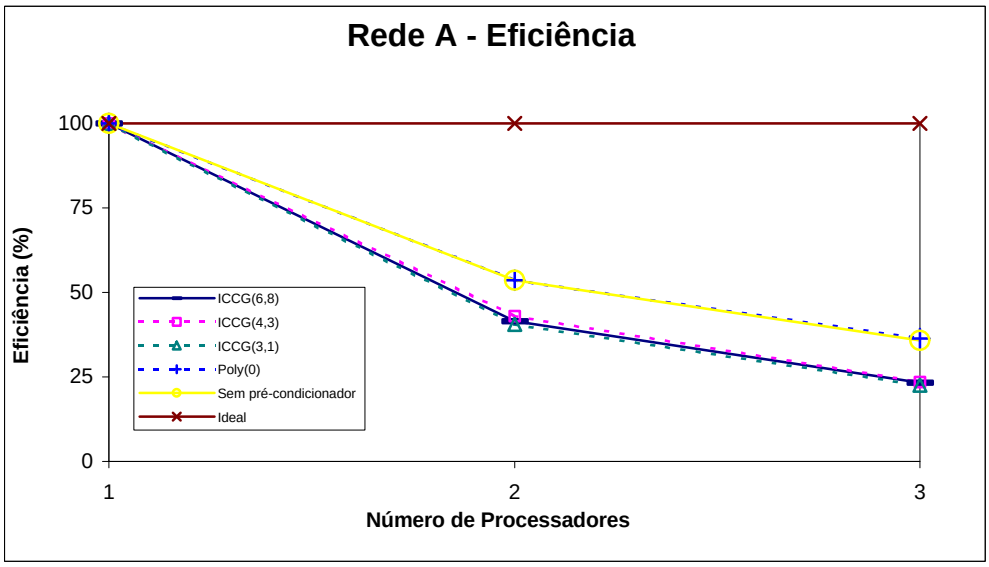


FIGURA 6.3 –Eficiência da rede A

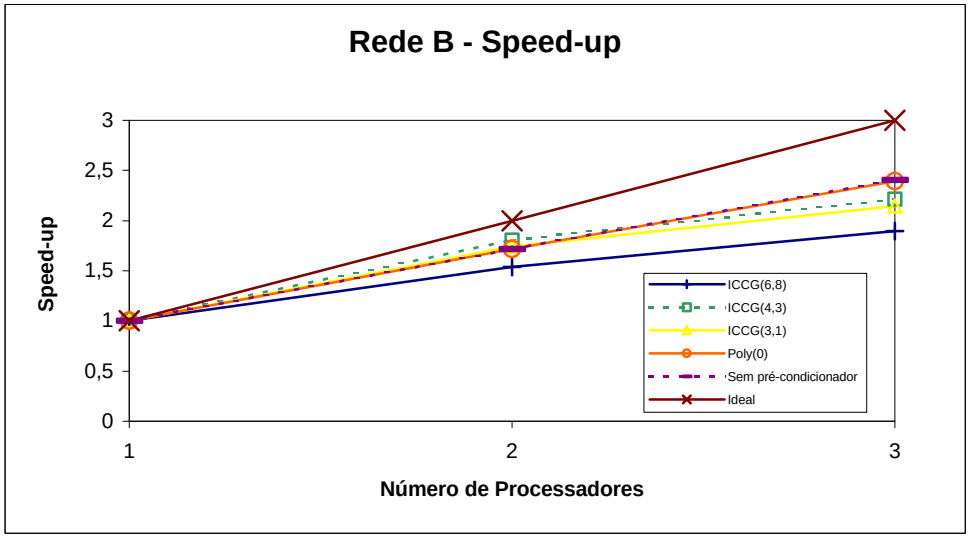


FIGURA 6.4 –Speed-up da rede B

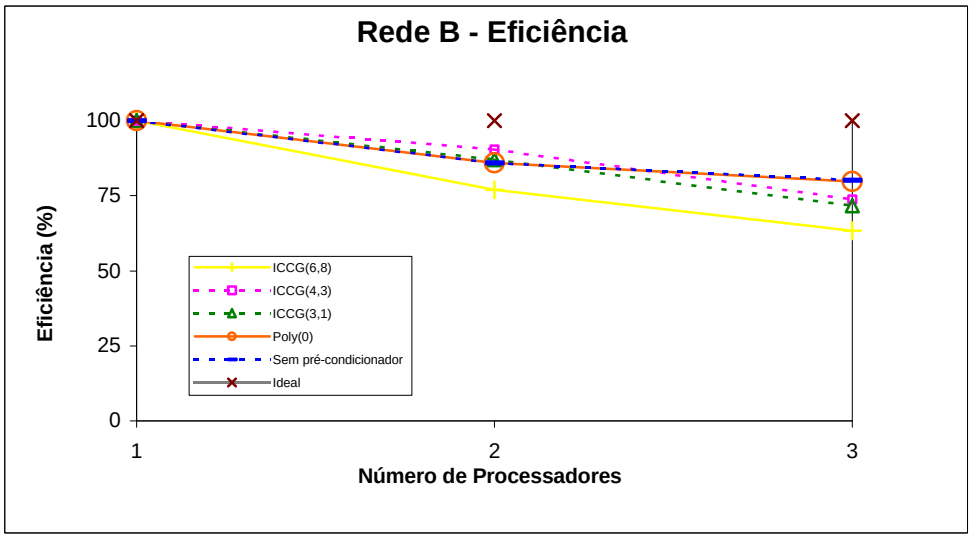


FIGURA 6.5 – Eficiência da rede B

Os valores de *speed-up* e de eficiência encontrados para a treliça demonstram, como era de se esperar, que o uso do paralelismo só se torna significativo quando tem-se um número de equações do sistema relativamente grande, ou seja, a partir do modelo B.

Pelos valores encontrados para o modelo B, tanto para *speed-up* como para eficiência, nota-se que com a maior quantidade de trocas de mensagens e também com a maior dependência entre os diversos processadores, que ocorre com o uso dos pré-condicionadores do tipo ICCG, a performance do sistema diminui a medida que aumenta-se o número de processadores, no caso de 1, 2 e 3.

Por outro lado, um outro fator que, também, deve ser considerado na performance do programa é o seu tempo comparativo de convergência para os diversos pré-condicionadores sob os diversos processadores. Já que os medidores de desempenho são parâmetros de comparação de eficiência do paralelismo no código computacional, mas *não mostrando a performance relativa de tem*po entre os diferentes pré-condicionadores.

Por exemplo, conforme a figura (6.4), o valor do *speed-up* para a rede B *com o pré-condicionador* ICCG(6,8) foi de *1,9* enquanto que *sem o uso de pré-condicionador* obteve-se um *speed-up* de *2,4*, ambos com 3 processadores. Por outro lado, o *tempo* requerido para resolver o sistema *com o pré-condicionador* ICCG(6,8)

foi em torno de *5000* segundos e o tempo *sem pré-condicionador* foi da ordem de *15000 segundos.*

Assim, nas figuras (6.6) a (6.8) são apresentados os tempos de resolução das diversas redes sob os diversos aceleradores de convergência.

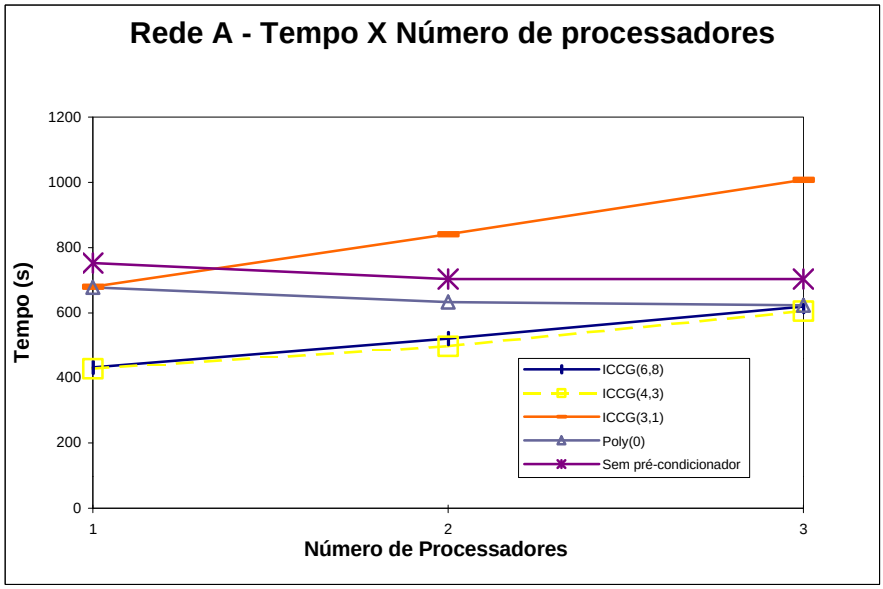


FIGURA 6.6 –Convergência em termos de tempo da rede A

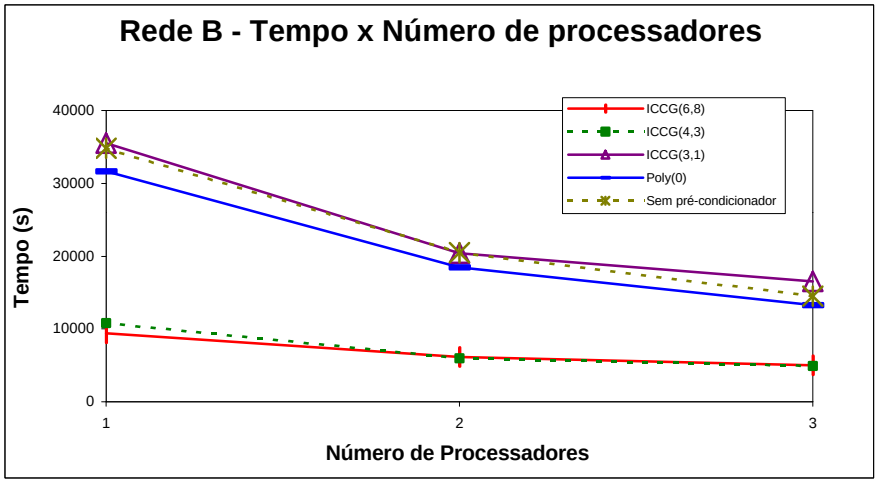


FIGURA 6.7 –Convergência em termos de tempo da rede B

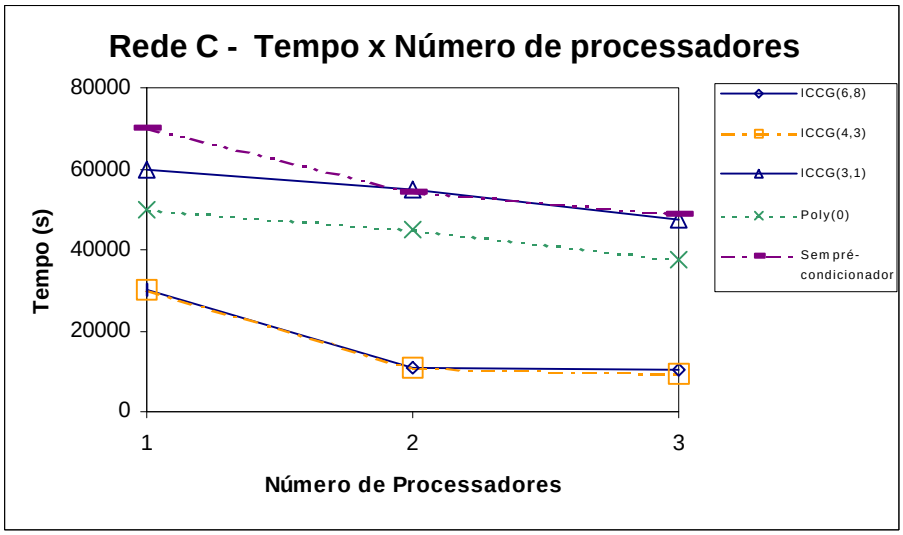


FIGURA 6.8 –Convergência em termos de tempo da rede C

Pelo apresentado pelas figuras (6.7) e (6.8), o tempo de resposta com os pré-condicionadores ICCG(6,8) e ICCG(4,3), em função da alta taxa de convergência destes pré-condicionadores, mesmo estes perdendo eficiência paralelismo, se tornaram melhores pré-condicionadores do que com o uso do POLY(0) para este modelo, com este número de processadores.

Para confirmar a eficiência em termos de tempo dos pré-condicionadores ICCG(6,8) e ICCG(4,3) com, no máximo, 3 processadores sob o modelo apresentado, na tabela (6.9) é plotado para as diversas redes deste modelo, uma curva associando o número de graus de liberdade *versus* o tempo de processamento, sendo que isto foi feito para três processadores.

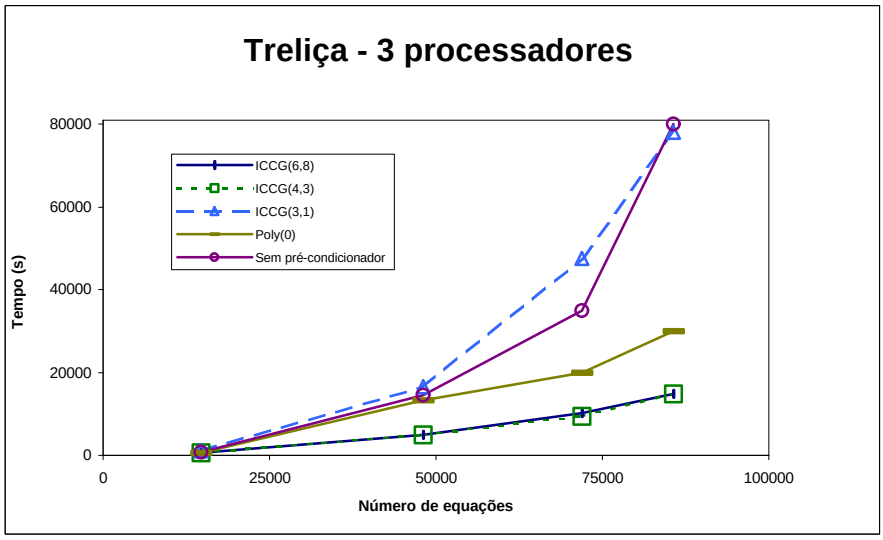


FIGURA 6.9 –Curva (Tempo x N° de equações) para a treliça

O último ponto da curva da figura (6.9) foi realizado em uma rede com 113288 elementos e 28561 nós, ou seja, com 85683 incógnitas no sistema linear, levando a formação de uma matriz de rigidez com 85683 linhas e 720 colunas (em semibanda) ou aproximadamente 0,5 Gbytes de memória.

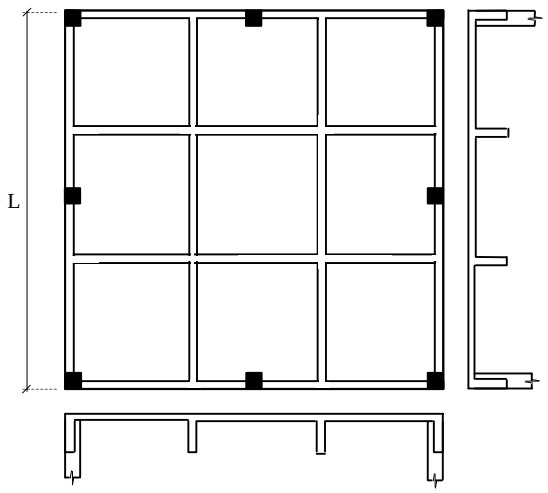
6.4 PAVIMENTO QUADRADO

Este exemplo é o mesmo apresentado no capítulo 5 no item 5.5.2, diferenciando-se pelos parâmetros geométricos e físicos da estrutura e do material. Assim, adotou-se valores compatíveis a uma estrutura real. Na figura (6.10) é esquematizado a sua forma e seus parâmetros.

Para este modelo, discretizou-se a estrutura em elementos finitos de placa e de viga formando o pavimento e foi considerado duas redes, dadas por

- ***Pavimento A: 11250 elementos de placa e 314 elementos de viga;***
- ***Pavimento B: 33800 elementos de placa e 534 elementos de viga;***

A norma utilizada no método dos gradientes conjugados é a norma euclidiana aplicada sobre o vetor resíduo e a precisão é de 1E-4 (ε = 1E-4).



$L = 600$ cm
$H_{VIGA} = 50$ cm
$b_{VIGA} = 10$ cm
$H_{LAJE} = 7$ cm
$J_{torção} = 1E\text{-}4$ cm^4
$E = 2500$ kN/cm^2
$\nu = 0{,}25$
$q = 1E\text{-}3$ kN/cm^2

FIGURA 6.10 –Esquema do pavimento analisado e suas características

Nas figuras (6.11) a (6.14), apresenta-se os valores de *speed-up* e de eficiência para as duas redes adotadas. É mostrado estes valores de eficiência para todos os processos relevantes para resolução do problema em MEF. Assim, obtêm-se valores de tempo em ambiente paralelo para:

- Leitura de dados e montagem do sistema linear;
- Montagem do pré-condicionador do tipo ICCG (MPC);
- Resolução do sistema;
- Obtenção dos esforços.

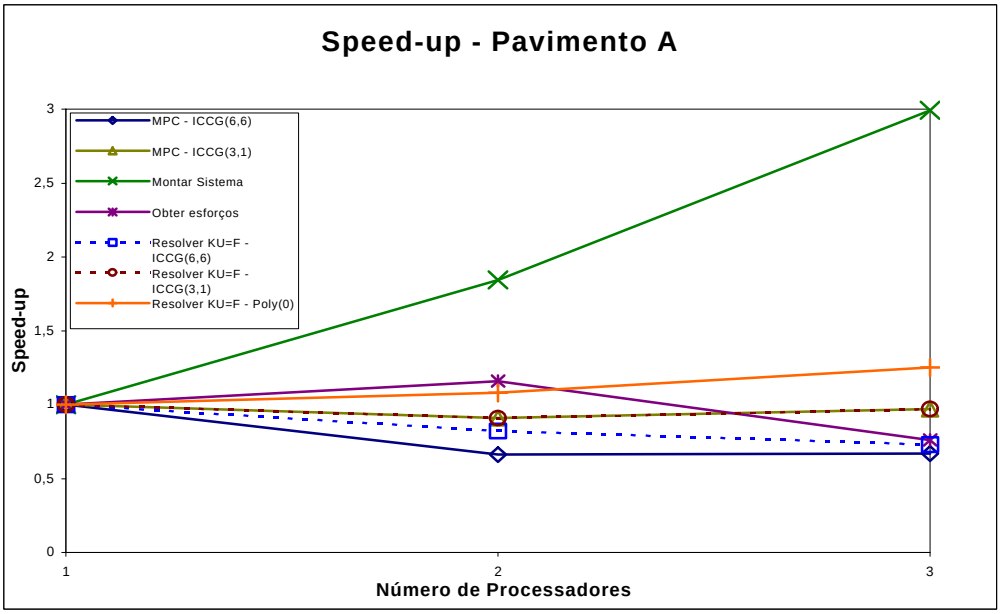


FIGURA 6.11 –*Speed-up* para o pavimento A

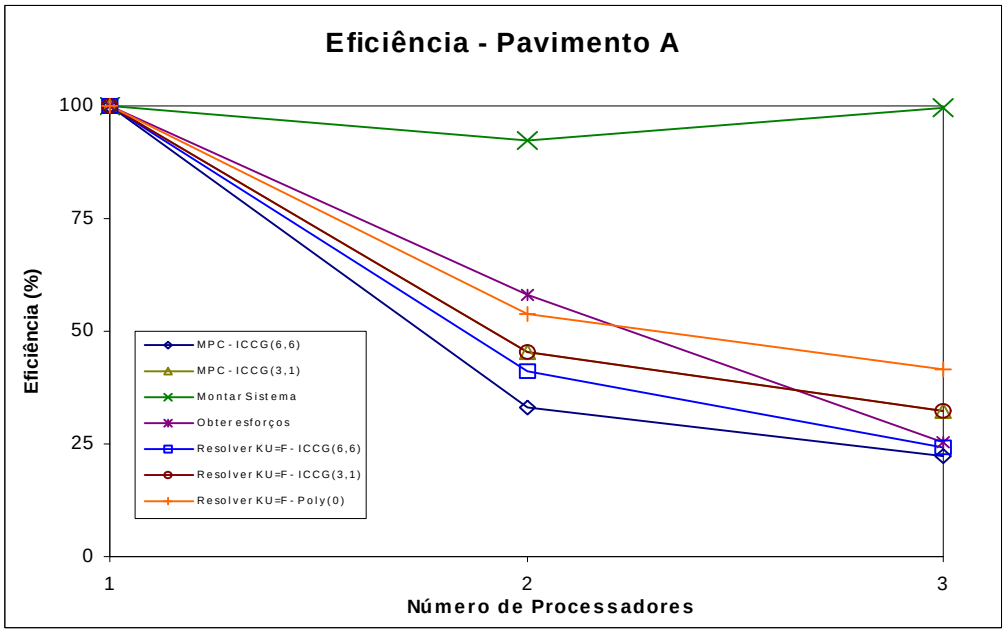


FIGURA 6.12 –Eficiência para o pavimento A

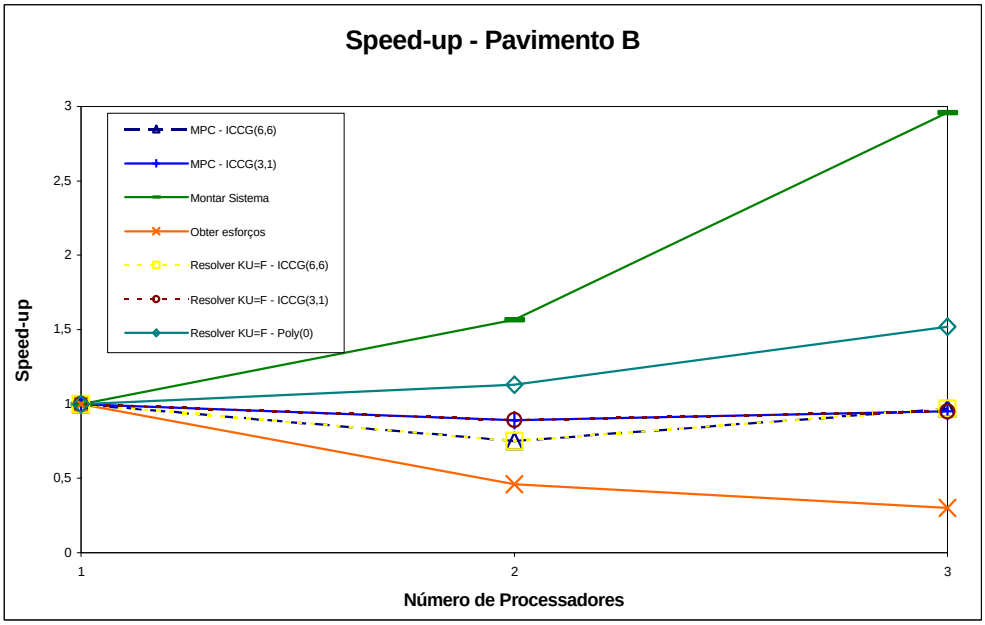


FIGURA 6.13 –*Speed-up* para o pavimento B

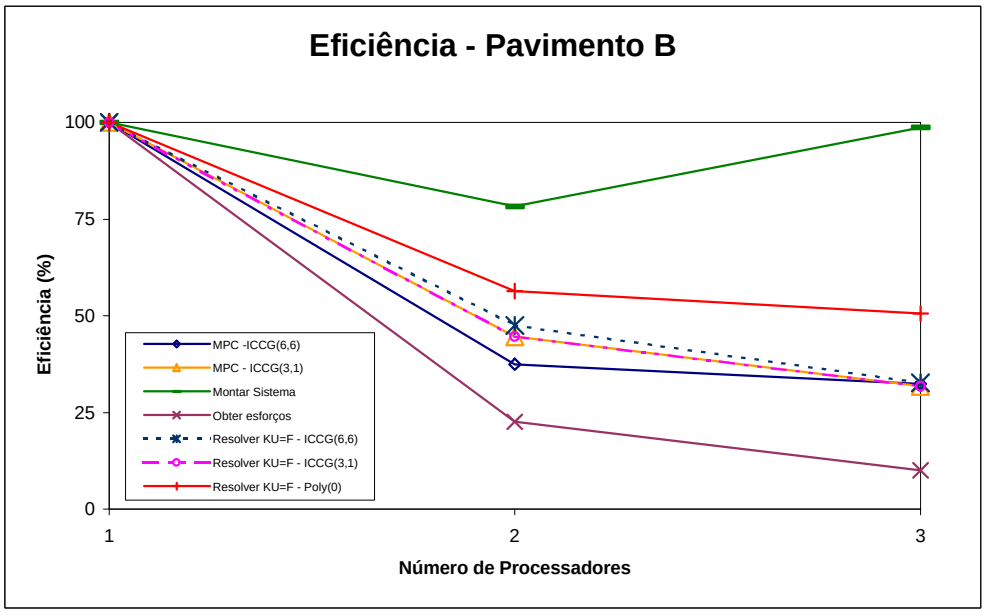


FIGURA 6.14 –Eficiência para o pavimento B

Em relação aos valores obtidos de eficiência para os diferentes procedimentos na análise deste pavimento, apresenta-se a seguir conclusões preliminares destes resultados encontrados:

- ***Leitura de dados e montagem do sistema linear***: Para este caso, averigua-se que em função deste procedimento ser um caso em que o paralelismo é realizado sem a necessidade de trocas de mensagens entre os processadores, ou seja, uma completa independência no tratamento dos dados, os valores de eficiência tanto para o pavimento A quanto para B, levaram a excelentes resultados de performance. Sem perda de generalidade, pode-se afirmar que para este caso a eficiência é ideal para a abordagem em paralelo com multiprocessadores.

- ***Montagem do pré-condicionador do tipo ICCG***: Como já salientado no capítulo 5, a montagem deste tipo de pré-condicionador é um procedimento ineficiente no tratamento em ambiente paralelo, principalmente a medida em que se aumenta a escalabilidade do sistema. Isto ocorre, pois a montagem de uma matriz do tipo ICCG leva a ociosidade dos processadores que já realizaram as mudanças sobre as suas linhas na matriz (ver item 5.3.4).

- ***Resolução do sistema:*** Para este caso, o pré-condicionador do tipo POLY(0) mostrou ser mais eficiente do que o do tipo ICCG, chegando a ter para o pavimento B um *speed-up* de 1,52, ou seja, uma eficiência de 51%.

  Acredita-se que a eficiência para o pré-condicionador do tipo ICCG venha a se evidenciar com o aumento do sistema linear e do número de processadores no ambiente paralelo, conforme pode ser observado nas figuras de (6.13) a (6.14).

- ***Obtenção de esforços:*** Conforme as figuras de (6.13) a (6.14), a obtenção dos esforços de todo o pavimento e o envio destes valores para um único processador para impressão, mostrou ser o procedimento mais ineficiente em todo o processamento paralelo neste modelo.

  Isto deve ter ocorrido, pois para a obtenção dos esforços médio nos nós, tem-se que os elementos que concorrem em um determinado nó não estejam armazenados no mesmo processador, assim há a necessidade de trocas de mensagens, e em virtude de se realizar paralelismo em um número pequeno de dados, isto causa um “gargalo” no tratamento destes dados em paralelo,

acreditando que em função da forma de análise destes dados, seja mais interessante analisa-los em um único processador.

A seguir, mostra-se as curvas de tempo *versus* número de processadores para as duas redes analisadas sob os diferentes pré-condicionadores.

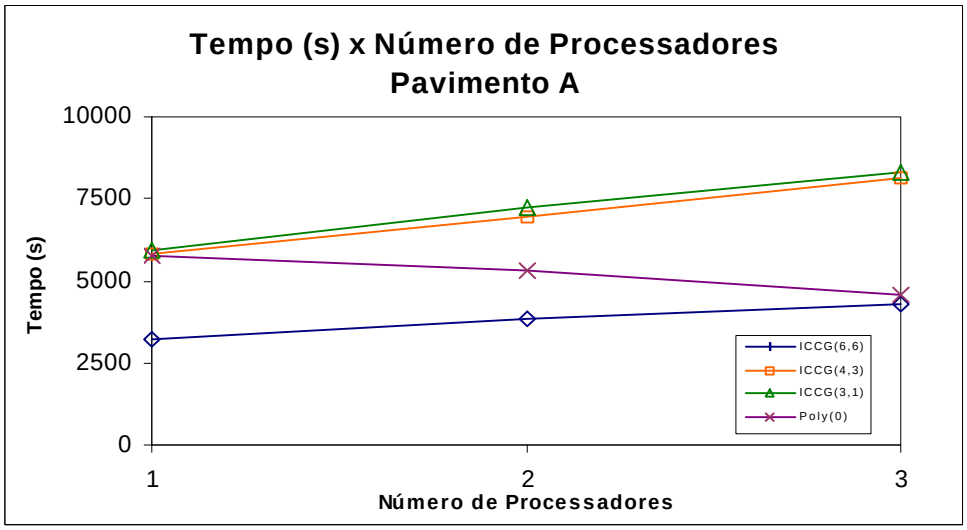


FIGURA 6.15 – Convergência em termos de tempo do pavimento A

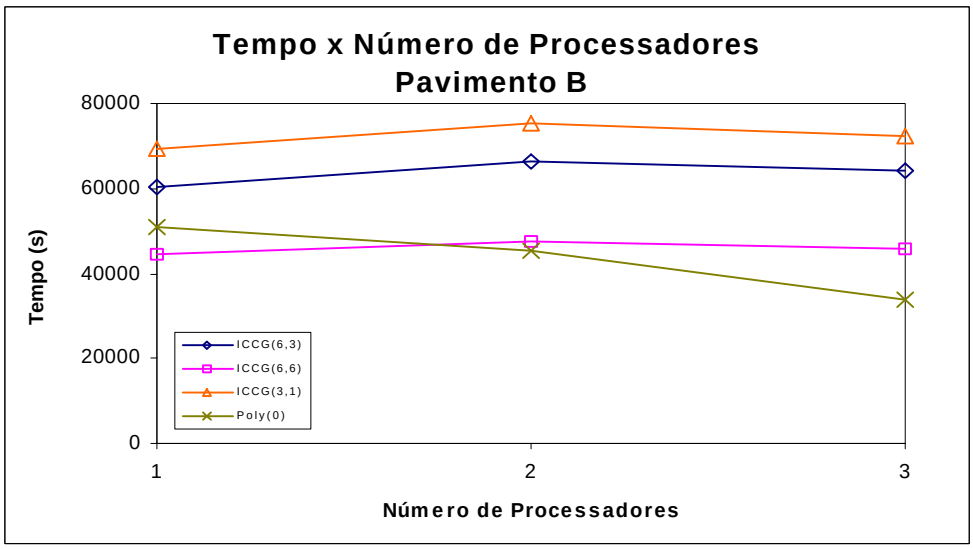


FIGURA 6.16 – Convergência em termos de tempo do pavimento B

Conforme mostra as figuras (6.15) e (6.16), o pré-condicionador POLY(0) demostra ser o mais eficiente para análise em paralelo para este modelo. Assim a

medida que aumenta-se o número de incógnitas do sistema a ser resolvido e com aumento do número de processadores, percebe-se que o tempo para resolução do sistema vai diminuindo.

O pré-condicionador ICCG(6,6) mostra-se mais eficiente em um único processador em relação ao POLY(0), mas vai perdendo sua performance no sistema com 2 processadores e começa lentamente a melhorar o tempo de resposta para 3 processadores, mas ainda não alcançando a performance em paralelo que o pré-condicionador POLY(0) demostra.

Capítulo 7

CONCLUSÃO

O presente trabalho teve o intuito de adaptar um código computacional advindo do método dos elementos finitos aplicado na análise de uma estrutura em multicomputadores.

Em virtude da arquitetura deste tipo de máquina, teve-se que modificar alguns procedimentos numéricos já consagrados pelo método convencional aplicado ao MEF. Destaca-se, assim, a alteração para a montagem e a resolução do sistema linear.

Convêm ressaltar que o ponto de "gargalo" para a otimização do método em processamento paralelo ocorre justamente na resolução do sistema. Assim, para se adaptar o MEF para este tipo de arquitetura, é necessário dar ênfase, principalmente, a esta fase, sendo os demais procedimentos, de certa forma, corriqueiros na paralelização. A importância do estudo da resolução do sistema em multicomputadores se torna mais evidenciada quando se aplica a problemas envolvendo não-linearidades, pois nestes processos, em geral, para cada incremento de passo, é necessário resolver um novo sistema.

Para a montagem do sistema linear, utilizou-se o procedimento proposto por REZENDE (1995), o qual fora desenvolvido para computadores de memória compartilhada. Porém, este se adaptou plenamente aos multicomputadores. Esta alteração, chamada em seu trabalho de abordagem nodal, consiste em montar a matriz de rigidez linha por linha e não pela forma convencional elemento por elemento. Assim,

para multicomputadores, isto é aproveitado de forma eficiente, pois divide-se a matriz de rigidez dentre os diversos processadores e então, paralelamente, e mais importante ainda, de maneira independente.

A grande vantagem oferecida por esta técnica, comparada com a técnica da decomposição em domínio, é que para esta última é necessário aplicar um pré-processador para se redividir a rede, de tal modo a se obter um balanceamento equilibrado dentre os vários processadores. Na técnica aqui empregada, o balanceamento de carga é obtido naturalmente.

Para a resolução do sistema linear, utilizou-se um método iterativo denominado de gradiente conjugado. Esta técnica adequou-se bem na análise em multicomputadores, pois, para cada passo, é necessário realizar um produto matriz-vetor, sendo este procedimento bem adequado na realização em paralelo. Implementou-se este produto levando-se em consideração a simetria e a esparsidade características do sistema. Com isso, aumentou-se a dependência entre os diversos processadores, mas procurando-se não perder o potencial oferecido pelo produto sob diversos processadores, e em virtude dos resultados encontrados, este resultado demonstrou ser eficiente.

Além disso, implementou-se também a soma global de um escalar, que é originado do produto interno local de cada processador, advindo tanto para a procura da direção que resolva o sistema quanto para o escalar originado pela norma utilizada.

O grande obstáculo que existe na utilização do MGC é a dependência para convergência do método em relação ao número de condição da matriz de rigidez. Assim, o método pode se tornar eficiente ou ineficiente conforme as características intrínsecas deste sistema.

Para solucionar este problema, é comum aplicar o pré-condicionamento sobre o sistema, a fim de tornar o método mais eficaz. Assim, neste trabalho, inicialmente foi proposto o estudo e a implementação de um tipo de pré-condicionador em paralelo: decomposição incompleta de Cholesky (IC); considerando-se 3 e 6 diagonais na matriz incompleta ([L]). Mas, generalizou-se este modelo em paralelo para poder considerar qualquer combinação de diagonais extras na montagem de [L], além de se fazer um

estudo de convergência numérica em processamento seqüencial para o pré-condicionador polinomial também generalizado advindo de uma série truncada (POLY).

Para este segundo caso, além de se utilizar o procedimento convencional para aplicar o POLY, testou-se aqui em processamento seqüencial - como extensão da idéia da decomposição incompleta - a montagem da inversa da matriz de rigidez pela aproximação da série, levando-se em consideração *apenas* algumas colunas da matriz originária, diminuindo-se, assim, o número de instruções a serem realizadas. Mostrou esta adaptação resultados interessantes a serem estudados e implementados em processamento paralelo, propondo-se a trabalhos futuros a sua utilização.

Aplicou-se, então, as duas técnicas generalizadas a alguns modelos estruturais em processamento seqüencial. O que mostrou-se é que estes pré-condicionadores trazem uma grande vantagem em termos de convergência, tanto em número de iterações quanto em termos de tempo, se comparados a resolução do sistema sem o seu uso.

Para os exemplos em processamento paralelo, foi utilizada a técnica da decomposição incompleta, além do pré-condicionador mais simples de se paralelizar da série polinomial: POLY(0). Dessa forma, analisou-se 2 modelos estruturais: treliça tridimensional e pavimento.

Os resultados, embora sejam restritos a um pequeno número de computadores (no máximo 3) aqui não são generalizados em virtude do restrito universo de computadores utilizados. Tem-se sim o intuito de verificar qualitativamente a validade dos modelos implementados neste campo e a potencialidade dos pré-condicionadores em processamento paralelo.

Para o exemplo da treliça tridimensional, percebe-se que o tempo de convergência com o uso do $ICCG(4,3)$ é um pouco inferior ao obtido pelo *ICCG(6,8)*. O segundo mostra uma convergência em termos de iteração um pouco menor que o primeiro. Porém, em virtude da quantidade de operações serem maior no *ICCG(6,8)*, em geral, o *ICCG(4,3)* mostrou-se mais adequado para grandes sistemas. Ambos, quando comparados com POLY(0), e sem o pré-condicionador, apresentaram valores da ordem de $\frac{1}{3}$ inferiores aos dois últimos. Vê-se também que a eficiência destes dois primeiros foi de 63% para 3 processadores.

Para o pavimento analisado, por outro lado, o pré-condicionador POLY(0) demostrou ser mais eficiente que o advindo da decomposição incompleta de Cholesky. A eficiência alcançada com o uso deste pré-condicionador foi de aproximadamente 51% com 3 processadores, ou seja, um *speed-up* de 1,52, enquanto que para a pequena quantidade de processadores utilizados, a eficiência do ICCG não tenha se evidenciado. Acredita-se que, em virtude do alto número de iterações para convergência deste segundo modelo (aproximadamente 20% do total de incógnitas) o ICCG deve ter se tornado ineficiente para este pequeno número de processadores, em virtude da sua caraterística seqüencial. Todavia, conforme o aumento do número de processadores, a sua potencialidade deve se evidenciar.

Ressalta-se que, os resultados encontrados para o pré-condicionador *ICCG(6,6)* foram bem próximas as encontradas pelo *ICCG(6,8)*. Contudo, como já salientado, muitas vezes este último pré-condicionador pode levar a montagem da matriz decomposta [L] com características positivas não-definidas, enquanto que o *ICCG(6,6)* não apresentou este problema, preferindo-se então o uso deste.

De acordo com os resultados obtidos nos capítulos 5 e 6, acredita-se que o uso dos pré-condicionadores *ICCG(6,6)* e *ICGG(4,3)* se tornaram eficientes frente ao pré-condicionador POLY(0) ou Jacobi. Isto é verdade desde que o número de iterações fique em torno de 5 a 10% do número de graus de liberdade do sistema. Nos exemplos numéricos apresentados, em geral, isso ocorreu como demonstrado. Mas para o caso do pavimento, conforme já salientado, este valor ficou na ordem de 20%, mostrando então a não potencialidade desse tipo de pré-condicionador ou a pouca eficiência do POLY(0) frente a este pequeno universo de processadores.

Supõe-se que isto tenha ocorrido em virtude do acoplamento de dois elementos finitos, que atribuam diferentes valores em ordem de grandeza à matriz de rigidez. Para o caso do pavimento isto se evidencia, já que a rigidez da viga é bem superior ao da placa. Essa diferença relativa de rigidez leva a um aumento no número de condição (κ) da matriz do pavimento, e com isso a convergência desse sistema se torna lento, acarretando um maior tempo nos procedimentos paralelos a serem executados. Isto

demonstra ser mais ineficiente ainda nas técnicas onde o paralelismo se evidencia, ou seja, no método da decomposição incompleta do Cholesky.

Como trabalhos futuros, sugere-se a implementação do produto matriz-vetor, de tal modo que seja apenas armazenado em um vetor os valores da matriz diferentes de zero, fazendo com que haja um ganho tanto em termos de memória quanto na diminuição da quantidade de operações a serem realizadas.

Também seria interessante implementar, neste programa desenvolvido em paralelo, o pré-condicionador genérico POLY(n_T,m), variando tanto o número de polinômios truncados como o espectro de influência para a montagem da inversa da matriz de rigidez. Acredita-se que os pré-condicionadores POLY(2,5) e POLY(4,m) tragam resultados interessantes na análise em processamento paralelo.

Por fim, em virtude da incerta eficácia que os pré-condicionadores trazem para a convergência, nos diferentes modelos estruturais que possam vir a ser implementados com este algoritmo aqui desenvolvido, sugere-se um estudo do comportamento das características da matriz de rigidez originada pelo sistema. Assim, como sugestão, poderia se calcular o número de condição de cada sistema, e desta forma encontrar uma relação entre qual tipo de pré-condicionador seria mais eficiente na convergência em processamento paralelo para um determinado modelo estrutural.

REFERÊNCIAS BIBLIOGRÁFICAS

ADELI, H. (1992). *Parallel processing in computational mechanics*. M. Dekker, New York.

ADELI, H. ; KAMAL, O. (1992). *Concurrent Analysis of large structures-I. Algorithms*. *Comput. Struct.*, v. 42, n. 03, p. 413-424.

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

ARGYRIS, J.H. ; KELSEY, S (1960). *Energy theorems and structural analysis*. Butterworths, London.

BARRAGY, E. ; CAREY , G. F. (1988). *A Parallel element-by-element solution scheme*. International Journal for numerical methods in engineering, v. 26, p. 2367-2382.

BATHE, K. J. (1982). *Finite element procedures in engineering analysis*. Englewood Cliffs, Prentice-Hall.

BATOZ, J. L. ; BATHE, K. J. ; HO, L. W. (1980). *A study of three–node triangular plate bending elements*. International Journal for numerical methods in engineering, v. 15, p. 1771-1812.

BEZERRA, Dermival P. (1995). *Análise de estruturas tridimensionais de edifícios altos considerando a rigidez transversal à flexão das lajes*. São Carlos. 138p.

Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

BITZARAKIS, S. ; PAPADRAKAKIS, M. ; KOTSOPULOS, A. (1997). *Parallel solution techniques in computational structural mechanics.* Comput. Methods Appl. Mech. Engr., v. 148, p. 75-104.

BREBBIA, C.A. (1978)."*The boundary element method for engineers*", Pentech, London.

CARROLL, A. B. ; WETHERALD, R. T (1967). *Application of parallel processing to numerical weather prediction.* Journal of the ACM, vol.14, n. 3, p.591-614, Julho.

CHAVES, E.W.V. (1996). *Sistema de Resolução de Pavimentos pelo Método dos Elementos Finitos*. EESC-USP, Depto. de Engenharia de Estruturas.

CHEN, H. C. ; BYREDDY, V. (1997). *Solving plate bending problems using finite strips on networked workstations*. Comput. Struct., v. 62, n. 2, p. 227-236.

CIMERMAN, Marcia. (1996). *Resolução de Sistemas Lineares Via Métodos Iterativos com Pré-Condicionadores – Aplicação em Problemas de Engenharia de estruturas.* São Paulo. 111p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo.

CRAY, *An optimizing HPF Compiler Conformant with the HPF Language Draft Standard 1.1*. (1998). http://www.cray.com/products/applications/directory/codes/pghpf_HPF.html (1 Agosto)

CUMINATO, J. A. ; MENEGUETTE, M. Jr. (1998). *Discretização de equações diferenciais parciais: Técnicas de Diferenças finitas.* Instituto de Ciência e Matemática de São Carlos, Universidade de São Paulo.

CUNHA, Rudnei D.; HOPKINS, Tim (1998*). PIM 1.1: The Parallel Iterative Methods package for Systems of Linear Equations User's Guide (Fortran 77 version).* Computing Laboratory, University of Kent at Canterbury, United Kingdom.

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

DEMMEL, James W. (1993). *Berkeley Lecture Notes on Numerical Linear Algebra.* Mathematics Department and Computer Science Division, University of California.

DUBOIS, P. F., GREENBAUM, A., RODRIGUE, G. H. (1977). Aproximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors. *Computing*, v.22, p. 257-268.

FARHAT, C. ; ROUX, F. X. (1991). *A method of finite element tearing and interconnecting and its parallel solution algorithm*. Int. J. Numer, Methods Engrg., v. 32, p. 1205-1227.

FLYNN, M. (1966). *Some computer organizations and their effectiveness*. IIEE Transactions, vol. 21.

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

GEIST, Al et al. (1994). *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press Cambridge, Massachusetts, London, England.

GOLUB, G. H. ; LOAN, Van C. F. (1984). *Matrix Computat*ion, Johns Hopkins Press, Baltimore.

GROSSO, A. DEL ; RIGHETTI, G. (1988). *Finite element techniques and artificial intelligence on parallel machines.* . Comput. Struct., v. 30, n.04, p. 999-1007.

IBM Parallel Environment for AIX: *MPI Programming and Subroutine Reference* (1995). University of Tennessee, Knoxville, Tennessee.

IBM, *IBM Software: XL High Performance Fortran for AIX* (1998). http://199.246.40.10/homepage_search.mod?q=HPF (1 Agosto).

JEYACHANDRABOSE, C. ; KIRKHOPE, J. ; BABU, C.R. (1985).*An alternative explicit formulation for the DKT plate-bending element*. International Journal for Numerical Methods in Engineering, v. 21, p. 1289-1293.

JIANG, Y. et al. (1997). *Parallel Computation of Polymer Melt Flow through an extruder*. Polyflow, s.a., Place de l'Université 16, B-1348 Louvain-la-Neuve, Belgium.

JOHNSON, O. G. ; MICCHELLI, C. A. , PAUL, G. (1983). Polynomial preconditioners for conjugate gradient calculations. *SIAM J. Num. Anal*., v.20, p. 362-376.

JOHNSSON, S. L. ; MATHUR, K. K. (1990). *Data structures and algorithms for the finite element method on a data parallel supercomputer*. International Journal for numerical methods in engineering, v.29, p. 881-908.

KARYPIS, G. ; KUMAR, V. (1995). *Unstructured graph Partioning and Sparse Matrix Ordering System, Version 2.0*. Department of Computer Science , University of Minnesota.

KIRCHHOFF, G. (1850). *Uber das gleichgewicht und die bewegung einer elastischen scleibe*. J. Math., n. 40, p. 51-58.

LAW, K. H. (1986). *A parallel finite element solution method*. Comput. Struct,. v. 23, n. 6, p. 845-858.

LAW, K. H.; MACKAY, D. R. (1993). *A parallel row-oriented sparse solution method for finite element structural analysis*. International Journal for numerical methods in engineering, v. 36, p. 2895-2919.

LEHMAN, M. (1966). *A survey of problems and preliminary results concerning parallel processing and parallel processors*. Proc. IIEE, v. 54, p. 1889-1901.

LUENBERGER, D. G. (1969). *Optimization by vector space methods.* John Wiley, New York.

MANTEUFFEL, T. A. (1980). An incomplete factorization MEIJERINK, J. A. ; VORST, H. A van der (1977). An iterative solution method for linear systems of wich the coefficient matrix is a symmetric M-matrix. *Mat. Comp.*, v.31 (137), p.148-162.

MESQUITA, A. D. (1998). *Uma formulação do método dos elementos finitos aplicada á analise elastoplástica de cascas.* São Carlos. 144p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

MEIJERINK, J. A. ; VORST, H. A van der (1977). *An iterative solution method for linear systems of wich the coefficient matrixis a symmetric M-matrix*. Mat. Comp., v.31 (137), p.148-162.

MINDLIN, R. D. (1951). *Influence of rotatory inertia and shear on flexural motions of isotropic, elastic plates*. J. Appl. Mech., n. 13, p. 31-38.

MIRANKER, W.L. (1971). *A survey of paralelism in numerical analysis*. SIAM, v. 13, n.4, p. 524-547.

NASCIMENTO, W. D. ; ALVES, J. L. D. ; COUTINHO, A. L. G. A. (1997). *Um estudo sobre algoritmos para particionamento de malhas não estruturadas.* Colégio Técnico Universitário, Universidade Federal de Juiz de Fora.

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

NOOR, A. K. (1997). *New Computing systems and future high-performance computing environment and their impact on structural analysis and design.* Comput. Struct. ,v.64, n. 1-4, p. 1-30.

NOOR, A. K. ; HARTLEY, S.J. (1978). *Evaluation of element stiffiness matrices on CDC STAR-100 computer*. Comput. Struct. ,v. 9, p. 151-160.

NOOR, A. K. ; LAMBIOTTE, J.J. (1979). *Finite element dynamic analysis on CDC STAR-100 computer*. Comput. Struct. ,v. 10, p. 7-12.

NOOR, A . K. ; PETERS, Jeanne M. (1989). *A partitioning strategy for efficient nonlinear finite element dynamic analysis on multiprocessor computers*. Comput. Struct., v. 31, n.05, p. 795-810.

PRZEMIENIECKI, J.S. (1962). *Matrix Structural Analysis of Substructures*. AIAA Journal, v. 1, n. 1, p. 138-147.

RAO, A. R. M. ; LOGANATHAN, K. ; RAMAN, N. V. (1993). *Studies on two concurrent solvers for finite element analysis*. Advances in Engineering Software. v. 18, p. 161-166.

REISSNER, E. (1944). *On bending of elastic plates*, J. Math. Physics, n.23, p. 184-191.

REISSNER, E. (1945). *The effect of transverse shear deformation on bending of elastic plates*. J. Appl. Mec*h.* , n. 12.

REZENDE, M. N. (1990). *Análise de pavimentos de edifícios pelo método dos elementos finitos em microcomputador*. São Carlos. 87p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

REZENDE, Marcelo Novaes (1995). *Processamento paralelo em análise estrutural.* São Carlos. 112p. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

ROCHA, L. M. (s.d.). *Cálculo: Integrais*. (s.n.).

SAMPAIO, M.C.; SAUÉ, J. P.; MOURA, J. B. (1988). *Unix – Guia do Usuário*. Makron Books do Brasil editora, São Paulo, Mc Graw-Hill.

SAXENA, M. ; PERUCCHIO, R. (1992*). Parallel fem algorithms based on recursive spatial decomposition – I. Automatic mesh generation.* . Comput. Struct., v. 45, n. 5/6, p. 817-831.

SAVASSI, Walter (1996). *Introdução ao método dos elementos finitos : em análise linear de estruturas.* São Carlos. Escola de Engenharia de São Carlos, Universidade de São Paulo.

SCHMIT, L. A. Jr. ; LAI, Y. C. (1994). *Structural optimization based on preconditioned conjugate gradient analysis methods*. International Journal for numerical methods in engineering, v. 37, p. 943-964.

SOUZA, A. S. C. (1998). *Contribuição ao estudo das estruturas metálicas espaciais.* São Carlos. 147p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

TOPPING, B. H. V. , KHAN, A I. (1996). *Parallel Finite Element Computations.* 1ª ed., Saxe-Coburg Publications, Edinburgh.

TURNER, M. J. , CLOUGH, R. W. , MARTIN, H. C. (1956). *Stiffness and Deflection Analysis of Complex Structures*. Journal of the aeronautical sciences, v.23, n.9, p. 805-823.

TUTORIAL do IBM SP. http://www.cisc.sc.usp.br/servicos/sp2.html (30 Julho/1998).

TUTORIAIS do CISC (1998). http://www.cisc.sc.usp.br/tutoriais/ (1 Agosto)

WALSHAW, C. ; CROSS, M. ; EVERETT, M. G. (1995). "*A Parallelisable Algorithm for Optimising Unstructured Mesh Partitions*". http:// www.gre.ac.uk/~wc06/papers/walshaw95.ps.gz (1 Agosto)

ZIENKIEWICZ, O. C. (1977). *The finite element meth*od. 3ª ed., McGraw-Hill, Maidenhead.

ZOIS, D. (1988a). *Parallel processing techniques for FE analysis: stiffiness, loads and stresses evaluation*. Comput. Struct,. v. 28, p. 247-260.

ZOIS, D. (1988b). *Parallel processing techniques for FE analysis: system solution*. Comput. Struct, v. 28, p. 261-274.

APÊNDICE I

I.1 O que é o PVM

Processamento paralelo, o método de se ter muitas pequenas tarefas para se resolver um grande problema, têm emergido como uma chave de possibilidade tecnológica na computação moderna. Os últimos anos têm testemunhado um avanço da aceitação e adoção do processamento paralelo, ambos para computação científica de alta performance e para as aplicações em “propósitos genéricos”, sendo que isto foi um resultado da exigência para mais alta performance, mais baixo custo e produtividade sustentada. A aceitação tem sido facilitada por dois grandes desenvolvimentos: processadores massivamente paralelos (PMPs) e o uso muito difundido da computação distribuída.

Os PMPs são agora os mais poderosos computadores do mundo. Estas máquinas combinam de uma pequena centena à algumas milhares de CPUs em um simples gabinete grande conectado à centenas de *gigabytes* de memória. PMPs oferecem um enorme poder computacional e são usados para resolver problemas de grande desafio computacional, tal como modelagem global do clima e preparação de remédios. Como as simulações têm se tornado realísticas, o poder computacional necessário para produzi-los cresce rapidamente. Assim, pesquisadores buscando maior eficiência investem, em ordem, em PMPs e em processamento paralelo para se obter os maiores poderes computacionais possíveis.

O segundo maior desenvolvimento que afeta a resolução de problemas científicos é a computação distribuída. Computação distribuída é o processo no qual um conjunto de computadores conectados por uma rede são usados coletivamente para solucionar um simples problema extenso. Assim, mais e mais organizações têm redes de área local de alta velocidade interconectando muitas estações de trabalho (*workstations*) para propósitos gerais, e os recursos computacionais combinados podem exceder o poder de um simples computador de alta performance. Em alguns casos, vários PMPs têm sido combinados usando computação distribuída para produzir inigualável poder computacional.

O mais importante fator em computação distribuída é o custo. Grandes PMPs tipicamente custam mais de 10 milhões de dólares. Em contrapartida, usuários vêm muito pouco custo em executar os seus problemas em um conjunto local de computadores existentes. É incomum para usuários de computação distribuída compreenderem o poder computacional bruto de um grande PMPs, mas eles são capazes de solucionar problemas bem maiores do que poderiam usando um de seus computadores locais.

As similaridades entre computação distribuída e PMP é a noção de passagem de mensagem. Em todos os processamentos paralelos, dados devem ser trocados entre os diversos processadores que cooperam em uma mesma tarefa. Vários paradigmas têm sido tentados incluindo memória compartilhada, compiladores paralelizantes, e passagem de mensagem. O modelo de passagem de mensagem tem se tornado o paradigma escolhido, pela perspectiva do número e variedade de multiprocessadores que ele suporta, assim como em termos de aplicações, linguagens e sistemas de *software* que ele usa.

O sistema *Parallel Virtual Machine* (PVM) usa o modelo de passagem de mensagem para permitir programadores explorarem a computação distribuída através de uma extensa variedade de tipos de computadores, incluindo PMPs. O conceito chave no PVM é que ele faz com que uma coleção de computadores pareçam como uma grande máquina virtual, por isso o seu nome.

I.2 Paradigmas comuns na programação em paralelo

Computação paralela que usa um sistema do tipo do PVM, pode ser aproximado de 3 formas, baseadas na organização das tarefas computacionais. Dentro de cada uma, diferentes estratégias de alocação da carga de trabalho são possíveis. O primeiro e mais comum modelo para aplicação no PVM, é conhecido como computação cheia *(crowd computing):* que é uma coleção de processadores interrelacionados e que tipicamente executam o mesmo código; realizando instruções sob diferentes partes da carga total de trabalho, geralmente envolvendo a constante troca de resultados intermediários, quer escalares, vetores ou matrizes. Para este paradigma, existem

ainda duas categorias que pode ser dividida esta forma de execução de tarefas em paralelo: o modelo mestre-escravo e o modelo de processador único (comumente é chamado cada processador de nó). Será dado ênfase neste apêndice apenas o primeiro modelo, já que é o utilizado neste trabalho.

I.3 Modelo Mestre-Escravo

O modelo mestre-escravo possui um programa principal, nomeado de mestre, responsável pela geração (*spawning*) do processo, inicialização das instruções em paralelo, obtenção dos resultados dos outros processadores e pela saída destes em arquivos. Enfim, ele é responsável pelo gerenciamento das tarefas que serão executadas pelos diferentes processadores. Os programas escravos executam a computação necessária, sendo que eles são gerados e controlados pelo processo mestre.

Toda esta computação envolve 3 fases:

- A primeira é a inicialização do processo no grupo, ou seja, no programa onde se aplicará as subrotinas do PVM, o mestre aloca o número de processos escravos requeridos pelo usuário e coloca todos (mestre mais escravos) em um grupo.

- A Segunda fase é a computação em si, onde todos do grupo executam as tarefas atribuídas a cada um deles e, quando necessário, trocam, transmitem ou recebem dados de outros processos, sendo que esta comunicação pode ser feita entre todos do grupo.

- A terceira fase é a obtenção e a apresentação dos resultados e, finalmente, o grupo participante de todo o processo é desfeito, terminando assim o processamento em paralelo. Podendo, ainda, ser executado instruções pelo processador mestre, em série e, se necessário, iniciar a geração novamente com um novo grupo.

O sistema PVM foi desenvolvido para implementação em programas que sustentam as seguintes linguagens: C, C++ e FORTRAN. As subrotinas existentes para essas linguagens somam mais de 100, sendo que será apresentado aqui apenas as mais comuns, e que conseguem atender aos problemas convencionais do usuário. Caso seja necessário, pode ser obtido tanto o sistema PVM com subrotinas em C e em FORTRAN, como o seu manual contendo todas as subrotinas, via *ftp* pela rede, mediante o seguinte endereço: ***netlib2.cs.utk.edu; cd pvm3/book; get refcard.ps)***.

I.4 Rotinas do PVM 3 para a linguagem FORTRAN

A seguir, apresenta-se uma lista das rotinas para implementação usando a linguagem FORTRAN:

1) PVMFMYTID

Função: prepara o sistema para as chamadas do PVM. O valor de retorno desta rotina indica o número do processo que será o mestre.

Chamada: *call pvmfmytid (tid)*

Parâmetro

tid: identificador inteiro do processador mestre, criado assim que se inicia a geração. Tid menor que zero indica erro.

2) PVMFPARENT

Função: comando que retorna o número do processo mestre gerado pelo comando 1.

Chamada: *call pvmfparent (num)*

Parâmetros

num: valor do processador mestre gerado.

3) PVMFJOINGROUP

Função: associa todos os processos no mesmo grupo.

Chamada: *call pvmfjoingroup ('grupo', num)*

Parâmetros

grupo: caracter do tipo string que nomeia o grupo existente.

num: inteiro que é retornado da rotina. Controla a formação do grupo, já que não se pode criar dois grupos ao mesmo tempo. Se num for menor que zero então indica um erro.

4) PVMFSPAWN

Função: gera os processos escravos.

Chamada: *call pvmfspawn (tarefa, flag, onde, ntarefas, tids, numt)*

Parâmetros

tarefa: string que indica o nome do programa a ser executado em paralelo.

flag: inteiro que específica o tipo de geração. Geralmente atribui-se o tipo default, que é 0.

onde: string que indica onde o processo PVM deve começar. Isto porque pode-se alocar uma máquina via rede. Para casos gerais, atribui-se o valor * que indica uma seleção aleatória dentre os processadores conectados no sistema PVM.

ntarefas: inteiro especificando o numero de geração dos escravos a serem feitas.

tids: vetor inteiro do tamanho de ntarefas, que retorna o número do processador gerado.

numt: inteiro que retorna o número de processadores gerados. Se menor que zero indica um erro na geração. Se positivo, mas menor que ntarefas indica que houve uma falha na geração.

5) PVMFINITSEND

Função: esta rotina limpa o buffer e o prepara para empacotar uma nova mensagem.

Chamada: *call pvmfinitsend (codigo, info)*

Parâmetros

*codigo***:** inteiro que formata a mensagem a ser enviada. Por default atribui-se 0 a este valor.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro na identificação no buffer.

6) PVMFPACK

Função: empacota a mensagem ativada pelo comando 5, onde o tipo de dado pode ser qualquer.

Chamada: *call pvmfpack (tipo, dados, nitem, stride, info)*

Parâmetros

tipo: representa o tipo de variável que será enviada. String=1; Integer2=2;Real4=4,Real8=6.

dados: ponteiro que indica o inicio do bloco dos dados a ser enviado. Se for vetor e quiser enviar todos os seus valores, indicar apenas o seu nome.

nitem: indica o número total de dados a serem enviados, em conformidade com o tamanho dos valores de dados.

stride: inteiro que indica o tipo de valor a ser empacotado. Se complex, stride=2, senão, stride=1.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro.

7) PVMFMCAST

Função: comando que transmite os dados em buffer a todos os processos indicados.

Chamada*:* *call pvmfmcast (nproc, tids, msgtag, info)*

Parâmetros

nproc: indica o número de processadores que receberão os dados.

tids: indica o número dos processadores, já que é associado para cada processador um número (tid).

msgtag: inteiro que associa o tipo de mensagem enviada e que deve ser recebida.

8) PVMFRECV

Função: este comando prepara a área de seu buffer (memória) para receber uma mensagem.

Chamada: *call pvmfrecv (tid, msgtag, info)*

Parâmetros

tid: inteiro que indica o processador que esta enviando mensagem. Se tid = –1, então recebe-se de qualquer processador.

msgtag: inteiro que associa o tipo de mensagem enviada e que deve ser recebida com a sua chamada. Se msgtag = -1 recebe-se qualquer mensagem enviada a ele.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro na identificação no buffer.

9) PVMFUNPACK

Função: desempacota a mensagem ativada pelo comando 8.
Chamada: *call pvmfunpack (tipo, dados, nitem, stride, info)*

Parâmetros

tipo: representa o tipo de variável que será recebida. String=1; Integer4=3;Real4=4,Real8=6.

dados: ponteiro que indica o inicio do bloco dos dados a ser enviado, mas deve combinar com o tipo que foi empacotado.

nitem: indica o número total de dados a serem recebidos, em conformidade com o tamanho dos valores de dados.

stride: inteiro que indica o tipo de valor a ser empacotado. Se complex, stride=2, senão, stride=1.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro.

10) PVMFSEND

Função: comando que envia os dados que estão ativos em buffer.
Chamada: *call pvmfsend (processo, msgtag, info)*

Parâmetros

processo: indicador do processo para o qual enviará a mensagem.

msgtag: inteiro que associa o tipo de mensagem enviada e que deve ser recebida.

11) PVMFBARRIER

Função: comando que sincroniza todos os processos do grupo até sua chamada.

Chamada: *call pvmfbarrier (grupo, cont, info)*

Parâmetros

grupo: string associado ao grupo para sincronização.

cont: inteiro que específica o numero total de processadores no grupo, incluindo o mestre.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro.

12) PVMFPERROR

Função: comando que fornece o status do erro ocorrido no PVM. Este comando pode ser colocado após a chamada de outra rotina do PVM. Se a verificação de erro desta outra rotina ocorrer, então esta rotina finaliza a tarefa.

Chamada: *call pvmfperror (aviso)*

Parâmetros

aviso: qualquer aviso indicando em qual comando houve falha.

13) PVMFLVGROUP

Função: comando para finalização do grupo.

Chamada: *call pvmflvgroup (grupo, info)*

Parâmetros

grupo: string associado ao grupo que será finalizado.

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro.

14) PVMFEXIT

Função: comando para sair do PVM.

Chamada: *call pvmfexit (info)*

Parâmetros

info: inteiro que retorna da rotina. Valores menores quer zero indica um erro.

I.5 Exemplo de um programa em FORTRAN usando o sistema PVM

Apresenta-se a seguir o código em FORTRAN de um simples exemplo, em que divide-se uma matriz quadrada A_{3x3} entre três processadores, sendo que cada linha da matriz é lida em um processador assim como um vetor B_3. O objetivo é obter o vetor resultante C_3 do produto entre A e B. Desta forma, faz-se localmente o produto de cada linha da matriz pelo vetor e ao final, os processos escravos enviam para o mestre que aloca cada subparte do vetor C na posição correta e imprimi o resultado.

Este código foi extraído de um exemplo feito como forma de exercício no curso de programação paralelo ministrada no CISC em Junho de 1997.

```
      program matf

c Exemplo de um simples programa para multiplicar
c matriz vetor em paralelo.
c
c Uso de grupos/barreiras


c declaração das rotinas externas do PVM (headears)
      include 'fpvm3.h'

c numero maximo de processos a gerar
      parameter (NMAX=3)

c subrotinas externos que sao do PVM

      EXTERNAL PVMFMYTID, PVMFJOINGROUP, PVMFSPAWN,
     &  PVMFBARRIER
      EXTERNAL PVMFLVGROUP, PVMFEXIT, PVMFRECV,
     &  PVMFINITSEND
      EXTERNAL PVMFPACK, PVMFUNPACK, PVMFPARENT,
     &PVMFPERROR
      EXTERNAL PVMFNRECV, PVMFGETINST, PVMFGSIZE, PVMFSEND

c variaveis de controle de tarefas
      integer mytid,nprocs,ibola
      integer tids(NMAX),me,me_arr,me0

c variaveis comuns a todos os processos
      double precision a(3),b(3),c(3),tmp
```

```
c******************  CODIGO COMECA AQUI  ******************
c entrando no PVM
      call pvmfmytid(mytid)

c veja quem e o mestre
      call pvmfparent(tids(1))

c ponha-me no grupo 'foo'
      call pvmfjoingroup('foo',me)
      if (me.lt.0) then
            call pvmfperror('Problema na criacao!', me)
      endif

c se sou o processo mestre, gere outros
      if( me.eq.0) then

c esse e o n# do tid do mestre(tid(1))
            tid(1)=mytid

c fixa o numero de processos escravos em 2
            nprocs=2

c gere os nprocs processos agora
            print *,'Processo #1 c/ tids=',tids(1)
         do j=2, nprocs+1
                  print *,'Gerando o processo #',j
                  call pvmfspawn('matf',pvmtaskdefault,'*',tids(j),ierr)
                  if (ierr.ne.1) then
                        print *,'Nao consegui gerar o processo',j
                        print *,'Encerrando...'
                        call pvmfexit(ierr)
                  endif
            enddo
            print *,'Todos os processos gerados!'

c soma o mestre ao numero de processos
            nprocs=nprocs+1

c envia o numero de processos para todos os processos
            call pvmfinitsend(pvmdatadefault, ibuf)
            call pvmfpack(integer4,nprocs,1,1,ierr)
            call pvmfmcast(nprocs,tids,1,ierr)
      else

c os escravos leem o numero de processos para chegarem
c na barreira...
            call pvmfrecv(tids(1),1,ibuf)
            call pvmfunpack(integer4, nprocs,ierr)
```

```
      endif

c sincronizando todos na barreira
      call pvmfbarrier('foo',nprocs,ierr)

c pega todos os tids correspondentes aos me's na
c ordem em que eles chegaram (e nao na que foram gerados...)
      do j=1,nprocs
            call pvmfgettid('foo',j-1,tids(j))
      enddo

c****************** TRABALHO EM PARALELO **********************

c cada processo inicializa a matriz e o vetor

c so o mestre realiza esta parte
      if (me.eq.0) then
            a(1)=5.0d0
            a(2)=1.0d0
            a(3)=9.0d0
      endif

c so o escravo 1 realiza esta tarefa
      if (me.eq.1) then
            a(1)=3.0d0
            a(2)=7.0d0
            a(3)=4.0d0
      endif

c so o escravo 2 realiza esta tarefa
      if    (me.eq.2) then
            a(1)=1.0d0
            a(2)=2.0d0
            a(3)=5.0d0
      endif

c todos leem o vetor b..
      b(1)=1.2d0
      b(2)=2.4d0
      b(3)=3.4d0

c e todos fazem o produto de sua linha pelo vetor b
      tmp=0.0d0
      do i=1,3
            tmp=tmp+a(i)*b(i)
      enddo
c escravos mandam e mestre recebe os dados
      if (me.ne.0) then
```

```
c escravos executam este pedaco
c mandam para o mestre
c inicializa buffer
            call pvmfinitsend(pvmdatadefault,ibuff)

c empacotam os resultados (tmp)
            call pvmfpack(real8,tmp,1,1,ierr)

c mandam mensagem para o mestre (tids(1) com valor, p.e.,5
            call pvmfsend(tids(1),5,ierr)

      else
c so o mestre executa este pedaco
c recebendo os dados dos escravos
            do i=2,3

c recebe dados de tids(i), com valor de mensagem 5
                  call pvmfrecv(tids(i),5,ierr)

c mestre guarda no vetor c, posicao i
                  call pvmfunpack(real8,c(i),1,1,ierr)
            enddo

c mestre poe seu resultado em c(1)
            c(1)=tmp

c mestre imprime o vetor c
            do i=1,3
                  print *,'c(',i,')=',c(i)
            enddo
      endif

c************** FIM DO TRABALHO EM PARALELO **************

c saia do grupo foo
      call pvmflvgroup('foo',ierr)

c saindo do PVM
      call pvmfexit(ierr)

c terminando o programa
      stop
      end
```