Pontifícia Universidade Católica
do Rio de Janeiro

Marcelo Rodrigues Leão Silva

Aplicação da programação orientada a objetos e da computação distribuída ao MEF para análise de estruturas

Tese de Doutorado

Tese apresentada ao Programa de Pós-graduação em Engenharia Civil do Departamento de Engenharia Civil da PUC-Rio como parte dos requisitos parciais para obtenção do titulo de Doutor em Engenharia Civil.

Orientador: Luiz Fernando Campos Ramos Martha

Rio de Janeiro

Setembro de 2005

PUC-Rio - Certificação Digital Nº 0221221/CA

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Marcelo Rodrigues Leão Silva

Aplicação da Programação Orientada a Objetos e da Computação Distribuída ao MEF para Análise de Estruturas

Tese apresentada como requisito parcial para obtenção do título de Doutor pelo Programa de Pós-Graduação em Engenharia Civil da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Luiz Fernando Campos Ramos Martha
Orientador
Departamento de Engenharia Civil - PUC-Rio

Dr. Marcello Goulart Teixeira
IME

Dr. Luiz Felippe Estrella Júnior
CEPEL

Prof. Raul Rosas e Silva
Departamento de Engenharia Civil - PUC-Rio

Prof. Paulo Batista Gonçalves
Departamento de Engenharia Civil - PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 19 de setembro de 2005

PUC-Rio - Certificação Digital Nº 0221221/CA

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Marcelo Rodrigues Leão Silva

Nasceu na cidade do Rio de Janeiro, RJ, em 28 de julho de 1966.Graduou-se em Engenharia de Fortificação e Construção pelo Instituto Militar de Engenharia em 1989, tendo ingressado no Quadro de Engenheiros Militares do Exército Brasileiro. Concluiu o Mestrado em Engenharia Mecânica do Instituto Militar de Engenharia em 1994, passando a integrar o Corpo Docente da Instituição.

Ficha Catalográfica

Silva, Marcelo Rodrigues Leão

Aplicação da programação orientada a objetos e da computação distribuída ao MEF para análise de estruturas / Marcelo Rodrigues Leão Silva ; orientador: Luiz Fernando Campos Ramos Martha. – Rio de Janeiro : PUC, Departamento de Engenharia Civil, 2005.

117 f. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Civil.

Inclui referências bibliográficas.

1. Engenharia civil – Teses. 2. Elementos Finitos. 3. Programação Orientada a Objetos. 4. Computação Paralela. 5. Plasticidade. I. Martha, Luiz Fernando Campos Ramos. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Civil. III. Título.

CDD: 624

PUC-Rio - Certificação Digital Nº 0221221/CA

PUC-Rio - Certificação Digital Nº 0221221/CA

À Minha esposa Beatriz e aos meus filhos Thiago e Lucas.

Agradecimentos

À minha esposa Beatriz e aos meus filhos Thiago e Lucas, pelo carinho e apoio.

Ao professor Luiz Fernando Martha, pela orientação do trabalho.

Aos demais professores do Departamento de Engenharia Civil da PUC, pelos ensinamentos transmitidos.

Ao Professor Antônio Carlos Areias Neto, pelo apoio.

Aos funcionários do Departamento de Engenharia Civil da PUC, especialmente Ana Roxo, pelo apoio administrativo.

Aos meus amigos Luiz Antônio Vieira Carneiro e Alexander Mazolli, pelo apoio e estímulo.

PUC-Rio - Certificação Digital Nº 0221221/CA

Resumo

Silva, Marcelo Rodrigues Leão, Martha, Luiz Fernando Campos Ramos (Orientador). **Aplicação da Programação Orientada a Objetos e da Computação Distribuída ao MEF para Análise de Estrutura**. Rio de Janeiro, 2005. 117p. Tese de Doutorado. Departamento de Engenharia Civil. Pontifícia Universidade Católica do Rio de Janeiro.

O objetivo deste trabalho é o de apresentar uma proposta de metodologia para a análise de estruturas pelo Método dos Elementos Finitos, utilizando-se na sua implementação as técnicas de programação orientada a objetos e computação distribuída. A utilização das técnicas de programação orientada a objetos permite a implementação de um código compacto, portável e de fácil adaptação. Para a implementação do código optou-se pela utilização da linguagem C++, que possui os recursos mais importantes da programação orientada a objetos, destacando-se a herança, o polimorfismo e a sobrecarga de operadores, e da biblioteca MPI de computação paralela. Inicialmente serão apresentados os procedimentos necessários à implementação orientada a objetos da análise de estruturas pelo método dos elementos finitos, sendo posteriormente apresentadas às alterações necessárias à inclusão das técnicas de processamento paralelo, empregando-se duas técnicas de paralelização. A grande quantidade de operações matriciais envolvidas na análise de estruturas pelo método dos elementos finitos motivou ainda o desenvolvimento de uma biblioteca de classes para a representação destas operações. Os exemplos apresentados têm a finalidade de verificar a exatidão dos resultados obtidos com o código implementado, e as vantagens de se empregar a programação orientada a objetos e a computação distribuída

PUC-Rio - Certificação Digital Nº 0221221/CA

Palavras-chave

Elementos Finitos; Programação Orientada a Objetos; Computação Paralela, Plasticidade

Abstract

Silva, Marcelo Rodrigues Leão; Martha, Luiz Fernando Campos Ramos (Advisor). **Application of the Object-Oriented Programming and Distributed Computing to the Structural Analysis by The Finite Element Method**. Rio de Janeiro, 2005. 117p. D.Sc. Thesis. Department of Civil Engineering. Pontifícia Universidade Católica do Rio de Janeiro.

This work focuses on a methodology for the analysis of structures based on the Finite Element Method (FEM) using on its implementation object-oriented programming techniques, together with parallel programming. The usage of object-oriented programming techniques allows the implementation of a compact, portable and of easily adaptable source code. The implementation was carried out using C++ language, which has the main features of the object-oriented programming, such as inheritance, polymorphism and operator overloading, and the MPI library for parallel computing. The procedures taken into account on object-oriented implementations for analysis of structures using the Finite Element Method are presented, followed by the modifications needed for including parallel computing, using two strategies. Also, the large amount of matrix operations involved on the structures analysis using Finite Element Method motivated the development of a class library which represents such operations. The examples presented have the purpose of verify the accuracy of the results obtained with the code, and the advantages of the use of object-oriented programming and parallel computing.

PUC-Rio - Certificação Digital Nº 0221221/CA

Keywords

Finite Elements; Object-Oriented Programming; Parallel Computing; Plasticity

Sumário

PUC-Rio - Certificação Digital Nº 0221221/CA

PUC-Rio - Certificação Digital Nº 0221221/CA

Lista de figuras

PUC-Rio - Certificação Digital Nº 0221221/CA

PUC-Rio - Certificação Digital Nº 0221221/CA

1
Introdução

1.1
Motivação e Objetivos

Desde que Clough [1] apresentou seus primeiros trabalhos de implementação computacional para a solução de estruturas pelo método dos elementos finitos, a complexidade das soluções de problemas de Engenharia tem sido sempre limitada pela capacidade de processamento disponível.

À medida que se aumentava esta capacidade de processamento, crescia não só a complexidade dos problemas cuja solução se tornava possível, mas também a do código computacional, cuja compreensão e manutenção se tornava cada vez mais difícil.

A fim de se contornar estas dificuldades, novas técnicas de programação foram introduzidas, sendo a mais recente a de programação orientada a objetos, cuja implementação permite uma representação computacional do problema mais próxima do modelo estrutural real que está sendo analisado.

No que se refere à limitação da capacidade de processamento, a distribuição do esforço computacional entre diversos processadores oferece uma possível alternativa a esta limitação, desde que o tempo gasto na comunicação entre os processadores não se sobreponha ao ganho obtido com sua distribuição. Nestes casos, torna-se possível à solução de problemas complexos em uma fração do tempo que seria necessário á solução obtida empregando-se um único processador.

Este trabalho tem como objetivo propor uma metodologia para a solução de problemas de análise estrutural pelo método dos elementos finitos empregando-se as técnicas de programação orientada a objetos e de programação paralela, visando simplificar a compreensão e a manutenção do código e a contornar as limitações impostas pela capacidade de processamento do *hardware*. O trabalho consiste no desenvolvimento de uma biblioteca de classes a ser disponibilizada na forma de *software* livre, empregando compiladores e

PUC-Rio - Certificação Digital Nº 0221221/CA

bibliotecas de livre distribuição, na apresentação de duas estratégias de paralelização, e visando também atender as novas diretrizes do governo federal adotadas pelo Instituto Militar de Engenharia (IME), instituição à qual o autor está vinculado.

1.2
Revisão Bibliográfica

A aplicação das técnicas de programação orientada a objetos tem sido considerada desde o final da década de oitenta por pesquisadores que desejavam implementá-las na solução de problemas de análise estrutural pelo método dos elementos finitos.

Uma das primeiras aplicações destas técnicas na análise de estruturas pelo método dos elementos finitos foi apresentada em 1989 por J. W. Baugh *et al.* [2], que apresentaram um sistema orientado a objetos para a análise elástica linear de estruturas pelo método dos elementos finitos implementado em Common Lisp Object System (CLOS) - um dialeto da linguagem Lisp - com base em um modelo geométrico descrito a partir de duas classes principais denominadas *Vertex* e *Edge*, capazes de representar geometricamente as entidades que são usadas na modelagem de estruturas pelo método dos elementos finitos. O modelo físico ou estrutural a ser analisado era composto por três classes: *Element*, *Node* e *Material*. A classe *Element*, que definia uma superclasse genérica para a representação de elementos, era definida por sua topologia, tipo de material, tipo de elemento e parâmetros complementares como área e espessura, sendo sua informação topológica herdada da classe *Edge*. A classe *Material*, como o próprio nome indica, continha informações relacionadas com as propriedades constitutivas do modelo. A classe *Node* herdava os atributos que definiam suas coordenadas e conectividade da classe *Vertex*, e implementava os atributos que representam seus graus de liberdade e condições de contorno (restrições a deslocamentos e cargas nodais aplicadas). Os diferentes tipos de elementos eram implementados como subclasses da classe *Element*, sendo responsáveis pela implementação dos atributos e métodos necessários à obtenção da sua matriz de rigidez local e transferência de suas contribuições à matriz de rigidez global. A análise da estrutura era feita por uma aplicação desenvolvida pelo usuário, sendo esta

PUC-Rio - Certificação Digital Nº 0221221/CA

aplicação responsável por armazenar todas as informações, e após o processamento os resultados eram armazenados nos objetos da classe *Node*. Abordagem semelhante foi adotada posteriormente por Archer [3].

Em 1990, Fenves [4] apresentava as vantagens das técnicas de programação orientada a objetos no desenvolvimento de softwares para a área de engenharia, destacando o fato de que suas técnicas de abstração dos dados produziam códigos mais flexíveis e modulares, com substancial reaproveitamento.

Ainda em 1990 Forde *et al.* [5] apresentaram uma implementação em que abstraíram os principais componentes empregados neste tipo de análise (elementos, nós, materiais, condições de contorno e cargas aplicadas) em uma estrutura de classes que também incluía classes destinadas a representar entidades associadas ao processamento numérico da solução. Apresentaram um programa orientado a objetos para a análise elástico-linear de estruturas pelo método dos elementos finitos empregando elementos planos isoparamétricios, com o objetivo de desenvolver uma biblioteca que pudesse ser expandida por outros usuários, com a finalidade de resolver problemas mais complexos, ou ser incorporada a outros sistemas. O programa apresentado era formado por seis classes principais: Três classes com funcionalidades semelhantes às classes *Element*, *Material* e *Node* apresentadas por Baugh & Rehak, e duas classes destinadas a tratar as condições de contorno, denominadas *DispBC* (responsável por manipular as informações relacionadas a deslocamentos prescritos) e *ForceBC* (responsável por manipular as informações relacionadas às cargas nodais aplicadas), além de uma classe chamada *Domain*, usada na representação da estrutura. Nesta implementação cada subclasse da classe *Element* era, no entanto, uma especialização de uma classe chamada *List*, responsável pelo armazenamento da matriz de rigidez e das cargas diretamente aplicadas a cada elemento e das suas respectivas contribuições às matrizes e vetores globais da estrutura. Esta classe, por sua vez, manipulava objetos de classes usadas nas representações dos pontos de Gauss (classe *Gausspoint*) e das funções de forma (classe *Shapefcn*). A estrutura como um todo era representada por um objeto da classe *Domain*, que manipulava listas de nós, elementos, materiais e condições de contorno, sendo ainda responsável pelo armazenamento da matriz de rigidez global da estrutura e dos vetores de forças globais. O programa foi implementado em uma linguagem híbrida, usando a linguagem C para a parte numérica e Object Pascal para a

PUC-Rio - Certificação Digital Nº 0221221/CA

definição das classes. Posteriormente, uma versão em linguagem C++ foi implementada por Scholtz [6].

Diversos autores [7,8,9,10,46,50] procuraram destacar as vantagens das técnicas de programação orientada a objetos sobre as tradicionalmente usadas na implementação feita em linguagem FORTRAN ou outras que à época usavam apenas as técnicas de programação procedural. Mackie [7], por exemplo, utilizou Turbo Pascal na sua primeira implementação, embora posteriormente tenha adotado a linguagem C++, como a maioria dos autores. O mesmo ocorreu com Zimmerman *et al* [21], que inicialmente empregaram a linguagem SmallTalk.

Muitos pesquisadores concentraram seus trabalhos no desenvolvimento de classes relacionadas ao processamento numérico da solução dos problemas. Scholz [6] apresentou exemplos detalhados da codificação de classes destinadas a representar vetores e matrizes. Zeglinski *et al.* [11] desenvolveram uma completa biblioteca de classes relacionadas ao tema da álgebra linear, incluindo classes para a representação de matrizes esparsas, de banda e triangulares, bem como uma descrição da semântica dos operadores empregados na linguagem C++. Lu *et al.* [12] apresentaram uma biblioteca numérica de classes desenvolvida em C++ com tipos adicionais de matrizes, registrando um desempenho compatível com as implementações feitas na linguagem C, e criticando a falta de abstração de dados e de encapsulamento da biblioteca LAPACK [13], desenvolvida em FORTRAN. Dongarra *et al.* [14] apresentaram então uma versão orientada a objetos desta biblioteca, implementada em C++ e denominada LAPACK++, que incluía classes capazes de representar os diversos tipos de matrizes, incluindo simétricas e de banda, com a possibilidade de inclusão de novos tipos e com velocidade de processamento e eficiência compatíveis com códigos já desenvolvidos em linguagem FORTRAN. Yu [51] apresentou um conjunto de modelos orientados a objetos para análise numérica pelo método dos elementos finitos.

No que se refere à modelagem das propriedades constitutivas de materiais como uma classe, com a finalidade de incorporar efeitos da não-linearidade do material, um dos primeiros trabalhos a abordá-la em maiores detalhes foi apresentado por Zahlten *et al.* [15], em que um objeto desta classe possuía como atributos objetos de classes usadas na representação de superfícies de escoamento, de regras de encruamento e do algoritmo para a solução do problema de valor

PUC-Rio - Certificação Digital Nº 0221221/CA

inicial associado ao problema. Posteriormente, Dobais *et al* [48] implementaram classes destinadas especialmente à solução numérica de problemas não-lineares.

A incorporação das não-linearidades geométricas foi considerada, juntamente com as não-linearidades do material, por Zabaras *et al* [16], que implementaram classes cujos objetos tornavam possível o armazenamento do histórico de deformações, e de variáveis envolvidas em problemas de contato.

A incorporação de análise dinâmica não-linear em um sistema orientado a objetos foi apresentada por G. R. Miller *et al.* [17,18,19], que implementaram um sistema fortemente baseado em entidades geométricas que incluíam pontos, vetores e tensores em três dimensões, em um sistema de coordenadas livres, e voltado principalmente aos métodos iterativos de solução baseados na interação entre elementos. As propriedades constitutivas dos materiais eram armazenadas em objetos de uma classe criada especialmente com esta finalidade.

T. Zimmermann *et al.* [20,21,22,23] desenvolveram um sistema para a análise linear estática e dinâmica de estruturas pelo método dos elementos finitos empregando as técnicas de programação orientada a objetos, com possibilidade de ser expandido de forma a considerar as não-linearidades do material, mas que exigiam a redefinição de algumas classes. A concepção do sistema era muito similar aos tradicionalmente desenvolvidos com as técnicas de programação estruturada, considerando que os dados eram armazenados em termos dos graus de liberdade globais e que as propriedades da estrutura eram reunidas em um sistema de equações lineares, sendo também fornecida uma biblioteca numérica capaz de manipular matrizes e vetores de diversos tipos. Classes eram definidas para a representação de nós, elementos, materiais e pontos de Gauss. Inicialmente o sistema foi desenvolvido em Smalltalk [21], sendo posteriormente reescrito em C++ [20] devido à baixa eficiência daquela linguagem. O sistema foi expandido de forma a incorporar a solução de análise plástica por Menétrey e Zimmermann [22].

H. Adeli *et al.* [24] apresentaram um sistema orientado a objetos para a análise elástico-linear de estruturas pelo método dos elementos finitos em que cada nó possuía três graus de liberdade cuja orientação coincidia com as do sistema global de coordenadas. Incluía ainda uma biblioteca de classes para a manipulação de vetores e matrizes e uma classe independente, chamada *GlobalData*, destinada a armazenar dados globais disponibilizados a todos os

PUC-Rio - Certificação Digital Nº 0221221/CA

objetos do sistema. Os nós armazenavam sua posição no sistema de coordenadas globais e os deslocamentos de seus graus de liberdade.

Hededal [25] apresentou uma implementação em que eram definidas classes para a representação de nós, elementos e materiais, sendo o modelo estrutural representado por listas de objetos destas classes. Foram também implementadas classes para a representação de matrizes e vetores usados na solução numérica do sistema de equações lineares.

Bittencourt [26] usou os recursos de *templates* da linguagem C++ na implementação de técnicas orientadas a objetos para a solução de estruturas pelo método dos elementos finitos com subestruturação.

Pode-se ainda destacar o desenvolvimento de completos ambientes para análise de estruturas pelo método dos elementos finitos empregando técnicas de programação orientada a objetos implementados por diversos autores [27,28,29,30,31,32,44, 45,49,52], destacando-se o mais recente trabalho de Mackie [32]

No que se refere á implementação de técnicas de programação paralela para a distribuição do esforço computacional, as principais estratégias consistem na decomposição do domínio do problema e na paralelização da solução do sistema de equações lineares (que nos problemas de análise estrutural pelo método dos elementos finitos costuma consumir a maior parte do tempo gasto no processamento).

No contexto da análise de estruturas pelo método dos elementos finitos, a decomposição do domínio consiste na subdivisão da estrutura em diversas partes, sendo cada uma destas partes ou subdivisões analisada em paralelo por um processador. Pode-se ainda considerar como parte desta estratégia a metodologia de análise denominada elemento-por-elemento, implementada neste trabalho, e que não requer que esta decomposição seja definida de forma explícita pelo usuário, o que faria com que a solução só fosse aplicável a modelos que possuíssem uma determinada topologia.

Uma visão geral das técnicas de decomposição do domínio é apresentada por Prieto *et al*. [33], e por Smith *et al* [34]. Aplicações específicas destas técnicas na análise paralela de estruturas pelo método dos elementos finitos são apresentadas por Topping [35].

PUC-Rio - Certificação Digital Nº 0221221/CA

No que se refere à paralelização da solução do sistema de equações lineares, métodos diretos de solução, como o método da eliminação de Gauss, podem ser modificados e paralelizados. A paralelização de métodos diretos de solução foi abordada em profundidade por Lin [36], enquanto Gupta *et al* [37] abordaram a solução em paralelo de sistemas representados por matrizes esparsas, simétricas e positiva-definidas. Scott [38,39] apresentou propostas para a paralelização de um método frontal de solução.

A metodologia denominada elemento-por-elemento foi usada inicialmente por Hughes *et al* [40] na solução serial de problemas de análise estrutural pelo método dos elementos finitos. Em 1997, Smith and Pettipher [41] usaram esta metodologia na solução em paralelo de estruturas pelo método dos elementos finitos, com o emprego da biblioteca MPI de troca de mensagens. Bane *et al* [42] apresentaram um trabalho semelhante ao de Pettipher e Smith [41], empregando a biblioteca OpenMP ao invés de MPI. Gullerud e Dodds [43] usaram o método dos Gradientes Condicionados com Pré-Condicionamento para resolver problemas tridimensionais de mecânica dos sólidos.

PUC-Rio - Certificação Digital Nº 0221221/CA

1.3 Organização do Trabalho

Este trabalho se compõe da presente introdução e mais sete capítulos:

O capítulo 2 apresenta um resumo do método dos elementos finitos, cuja compreensão é indispensável ao entendimento do código computacional que será apresentado.

O capítulo 3 apresenta os conceitos fundamentais da programação orientada a objetos, suas principais características e vantagens.

O capítulo 4 apresenta uma descrição das classes implementadas neste trabalho para a representação das diversas entidades presentes na solução de problemas de análise estrutural pelo método dos elementos finitos, usando a linguagem C++. Uma listagem completa da implementação dos seus métodos é fornecida como um anexo complementar.

O capítulo 5 apresenta os conceitos fundamentais da programação paralela e das bibliotecas de troca de mensagens.

O capítulo 6 apresenta os procedimentos necessários à incorporação das técnicas de programação paralela ao sistema já desenvolvido, empregando-se a biblioteca MPI e discute as técnicas de paralelização que serão empregadas na sua implementação.

O capítulo 7 apresenta exemplos cuja solução foi obtida usando o sistema implementado.

O capítulo 8 apresenta as conclusões do trabalho, além de sugestões para trabalhos futuros.

PUC-Rio - Certificação Digital Nº 0221221/CA

2
O Método dos Elementos Finitos

2.1
Considerações Gerais

O método dos elementos finitos é, atualmente, um dos métodos numéricos mais empregados na análise computacional de estruturas.

A aplicação deste método consiste na divisão da estrutura (domínio do problema) em regiões (subdomínios) denominadas elementos, interligadas através de pontos especiais denominados nós ou pontos nodais.

Em cada elemento, considera-se que as componentes de deslocamento de qualquer ponto interior pode ser obtido por interpolação dos seus respectivos valores nos pontos nodais. Uma vez obtidas as expressões matemáticas que definem as componentes de deslocamento em qualquer ponto, as componentes de deformações e tensões podem então ser obtidas considerando-se as equações estabelecidas pela teoria da elasticidade para cada tipo de análise.

Fica evidente, portanto, a importância da obtenção adequada do campo de deslocamentos nodais e da precisão da interpolação destes valores na análise estrutural. Uma interpolação pobre pode ser compensada com uma rica decomposição do domínio, e vice-versa.

Conforme será mostrado no próximo tópico, para cada elemento uma matriz [K], denominada matriz de rigidez local ou do elemento é definida, relacionando os deslocamentos dos seus pontos nodais e as forças externas aplicadas a estes pontos, considerando-se isoladamente o equilíbrio do elemento. Uma análise do equilíbrio global da estrutura mostra que a sua matriz de rigidez (denominada matriz de rigidez global) pode ser obtida a partir da contribuição das matrizes de rigidez dos seus elementos.

PUC-Rio - Certificação Digital Nº 0221221/CA

2.2
Formulação Matemática

Este tipo de formulação está presente na quase totalidade dos livros sobre elementos finitos, dentre os quais pode-se destacar as referências [53,54]. Neste tópico algumas fórmulas são deduzidas utilizando equações definidas para problemas bidimensionais, sem que isso comprometa a sua aplicabilidade aos problemas tridimensionais, desde que empregadas as equações correspondentes.

Pode-se considerar como ponto de partida desta formulação o princípio dos trabalhos virtuais, que oferece a seguinte equação para a aplicação de um campo de deslocamentos virtuais $\delta\{u\}$ a uma estrutura em equilíbrio e submetida a um conjunto de forças externas diretamente aplicadas:

$$\delta W = \delta W_i - \delta W_e = \int_V \delta\{\varepsilon\}^T \{\sigma\} dV - \delta\{U\}^T \{F\} = 0 \qquad (2.1)$$

Onde

δW: Trabalho virtual total.

δWi: Energia de deformação interna.

δWe: Trabalho virtual das forças externas.

$\delta\{\varepsilon\}$: Vetor que representa as deformações virtuais, correspondentes ao deslocamento virtual $\delta\{u\}$.

$\{\sigma\}$: Vetor que representa as componentes reais de tensão.

$\delta\{u\}$: Vetor que representa os deslocamentos virtuais aplicados a estrutura.

$\delta\{U\}$: Vetor que representa os deslocamentos virtuais aplicados aos pontos nodais da estrutura.

$\{F\}$: Vetor que representa as cargas externas diretamente aplicadas aos pontos nodais da estrutura.

Considerando-se que internamente a qualquer elemento o campo de deslocamentos será obtido por interpolação dos valores calculados nos pontos nodais da estrutura, pode-se escrever a seguinte equação:

$$\delta\{u\} = [N]\delta\{U\} \qquad (2.2)$$

PUC-Rio - Certificação Digital Nº 0221221/CA

onde [N] é uma matriz cujos elementos são as funções de interpolação definidas para o elemento.

Sendo o vetor de deformações (no campo das pequenas deformações, normalmente adotado para o caso de análise linear elástica) da estrutura definido em um problema bidimensional por:

$$\{\varepsilon\} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} \tag{2.3}$$

As equações anteriores permitem que se escreva:

$$\{\varepsilon\} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \\ \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{bmatrix} = \lfloor H \rfloor \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \\ \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{bmatrix} \tag{2.4}$$

onde:

$$\lfloor H \rfloor = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.5}$$

Utilizando-se as funções de interpolação, têm-se:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} N1 & 0 & N2 & 0 & \ldots & \ldots \\ 0 & N1 & 0 & N2 & \ldots & \ldots \end{bmatrix} \begin{bmatrix} u1 \\ v1 \\ u2 \\ v2 \\ \ldots \\ .. \end{bmatrix} = [\mathrm{N}]\{\mathrm{U}\} \tag{2.6}$$

Logo:

PUC-Rio - Certificação Digital Nº 0221221/CA

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial y} & 0 \\ 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} N1 & 0 & N2 & 0 & \dots & \dots \\ 0 & N1 & 0 & N2 & \dots & \dots \end{bmatrix} \begin{bmatrix} U1 \\ V1 \\ U2 \\ V2 \\ \dots \\ .. \end{bmatrix} =$$

$$\begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & \dots & \dots \end{bmatrix} \begin{bmatrix} U1 \\ V1 \\ U2 \\ V2 \\ \dots \\ .. \end{bmatrix} \tag{2.7}$$

Finalmente

$$[\varepsilon] = [H] \begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & \dots & \dots \end{bmatrix} \begin{bmatrix} U1 \\ V1 \\ U2 \\ V2 \\ \dots \\ .. \end{bmatrix} = [H][G]\{U\} = [B]\{U\} \tag{2.8}$$

onde

$$[G] = \begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & \dots & \dots \end{bmatrix} \tag{2.9}$$

Portanto, [B] = [H][G] é a matriz que relaciona deformações e deslocamentos (no campo das pequenas deformações).

Para o caso de deslocamentos virtuais têm-se:

$$\delta\{\varepsilon\} = [H] \begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & \dots & \dots \end{bmatrix} \begin{bmatrix} \delta U1 \\ \delta V1 \\ \delta U2 \\ \delta V2 \\ \dots \\ .. \end{bmatrix} = [H][G]\delta\{U\} = [B]\delta\{U\} \tag{2.10}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

Conseqüentemente:

$$W = W_i - W_e = \int_V \delta\{\varepsilon\}^T \{\sigma\} dV - \delta\{U\}^T \{F\} = 0 \tag{2.11}$$

$$W = \int_V \delta\{U\}^T [B]^T \{\sigma\} dV - \delta\{U\}^T \{F\} = 0 \tag{2.12}$$

$$W = \delta\{U\}^T \int_V [B]^T \{\sigma\} dV - \delta\{U\}^T \{F\} = 0 \tag{2.13}$$

$$W = W_i - W_e = \delta\{U\}^T \left(\int_V |B|^T \{\sigma\} dV - \{F\} \right) = 0 \tag{2.14}$$

Como o deslocamento virtual $\delta\{U\}$ é arbitrário e a equação anterior deve ser válida para qualquer campo de deslocamentos virtuais $\delta\{U\}$, deve-se ter:

$$\int_V [B]^T \{\sigma\} dV - \{F\} = 0 \tag{2.15}$$

Para o caso em análise, em que é admitida a existência de uma relação linear entre tensões e deformações, pode-se escrever:

$$\{\sigma\} = [\mathrm{D}]\{\varepsilon\} = [\mathrm{D}][\mathrm{B}]\{U\} \tag{2.16}$$

Conseqüentemente, a equação anterior pode ser reescrita da seguinte maneira:

$$\left(\int_V [B]^T [D][B] dV \right) \{\mathrm{U}\} - \{F\} = 0 \tag{2.17}$$

ou, finalmente:

$$[K]\{\mathrm{U}\} = \{F\} \tag{2.18}$$

onde $[K] = \int_V [B]^T [D][B] dV$ é denominada a matriz de rigidez da estrutura

Evidentemente, estando a estrutura subdividida nas diversas regiões denominadas elementos, a integração anterior pode ser efetuada da seguinte maneira, onde a integral é aplicada a cada um dos *Ne* elementos:

$$[K] = \int_V [B]^T [D][B] dV = \sum_{e=1}^{Ne} \int_{Ve} [B]^T [D][B] dV_e \tag{2.19}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

2.3
Funções de interpolação

Neste trabalho são considerados elementos quadriláteros isoparamétricos, derivados de um "elemento-padrão" quadrado, mostrado na figura 1, cujos vértices possuem as coordenadas (-1,-1), (1,-1), (1,1) e (-1,1), e para o qual as funções de interpolação são:

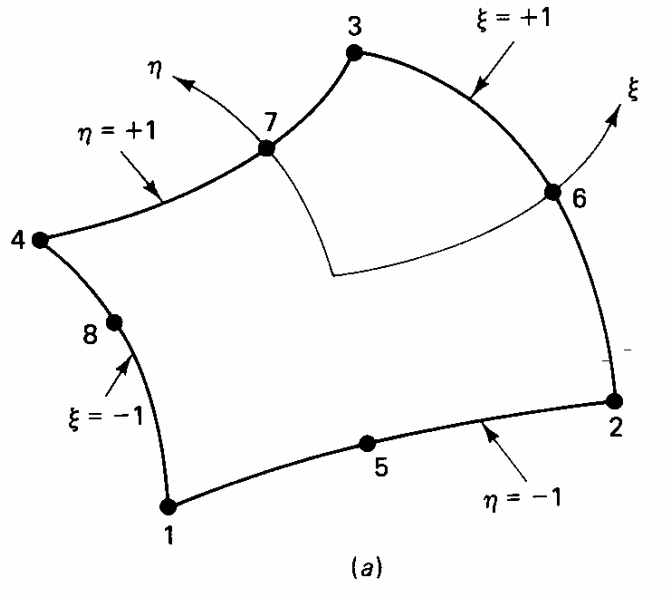

Figura 1 – Definição do Elemento-Padrão

- Para elementos com quatro pontos nodais:

$$N_1 = 0{,}25(1 - \xi)(1 - \eta) \tag{2.20}$$

$$N_2 = 0{,}25(1 + \xi)(1 - \eta) \tag{2.21}$$

$$N_3 = 0{,}25(1 + \xi)(1 + \eta) \tag{2.22}$$

$$N_4 = 0{,}25(1 - \xi)(1 + \eta) \tag{2.23}$$

- Para elementos com oito pontos nodais:

$$N_1 = -0{,}25(1 - \xi)(1 - \eta)(1 + \xi + \eta) \tag{2.24}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

$$N_2 = -0{,}25(1 + \xi)(1 - \eta)(1 - \xi + \eta) \tag{2.25}$$

$$N_3 = -0{,}25(1 + \xi)(1 + \eta)(1 - \xi - \eta) \tag{2.26}$$

$$N_4 = -0{,}25(1 - \xi)(1 + \eta)(1 + \xi - \eta) \tag{2.27}$$

$$N_5 = 0{,}5(1 - \xi^2)(1 - \eta) \tag{2.28}$$

$$N_6 = 0{,}5(1 + \xi)(1 - \eta^2) \tag{2.29}$$

$$N_7 = 0{,}5(1 - \xi^2)(1 + \eta) \tag{2.30}$$

$$N_8 = 0{,}5(1 - \xi)(1 - \eta^2) \tag{2.31}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

Neste tipo de formulação, as funções usadas na interpolação dos deslocamentos são também usadas na interpolação da geometria. Para que a integração possa ser feita no domínio do "elemento-padrão", deve-se considerar as seguintes transformações, conseqüência da aplicação da "regra da cadeia" para a diferenciação de funções de várias variáveis:

$$\begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{pmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} = [J] \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \tag{2.32}$$

A equação anterior fornece:

$$\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} = [J]^{-1} \begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{pmatrix} \tag{2.33}$$

Como a integração será feita considerando-se os diversos valores de interesse das funções de forma e suas derivadas para o sistema local ao "elemento-

padrão", as integrais presentes na equação (2.18) devem ser multiplicadas pelo determinante da matriz [J], denominado Jacobiano, e que é numericamente igual à relação entre o volume infinitesimal real do elemento sobre o qual está sendo feita a integração e o volume infinitesimal do elemento-padrão

$$[K] = \int_V [B]^T [D][B] dV = \sum_{e=1}^{Ne} \int_{Ve} [B]^T [D][B] dxdy = \sum_{e=1}^{Ne} \int_{Ve} [B]^T [D][B] \det([J]) d\xi d\eta \tag{2.34}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

3
Programação Orientada a Objetos

3.1
Considerações Gerais

Até o início da década de noventa, as linguagens de programação mais empregadas na implementação de programas de análise de estruturas pelo método dos elementos finitos eram a linguagem FORTRAN e a linguagem C. A linguagem FORTRAN foi inicialmente empregada por ter sido a primeira linguagem científica de alto nível disponível nas universidades e grandes centros de pesquisa, principalmente na época dos computadores de grande porte *(mainframes)*. Exemplos da utilização destes códigos podem ser encontrados em [1,2]. A grande quantidade de bibliotecas matemáticas e de elementos já desenvolvidas e disponíveis nesta linguagem é um dos principais argumentos utilizados por muitos profissionais para justificar a sua utilização até os dias atuais, a despeito das facilidades disponíveis em linguagens de programação mais modernas, e que oferecem as técnicas e recursos da programação orientada a objetos.

A linguagem C, inicialmente disponibilizada para o ambiente UNIX, foi adotada por muitos pesquisadores como alternativa à linguagem FORTRAN. Suas principais vantagens em relação à versão da linguagem FORTRAN disponível à época do lançamento da linguagem C eram: a possibilidade de criação de novos tipos de dados estruturados *(struct)* e o recurso de alocação dinâmica de memória (usado em substituição ao emprego dos "vetores de trabalho" tradicionalmente empregados em FORTRAN).

O emprego destas duas linguagens, no entanto, produz uma implementação com código bastante extenso, de difícil manutenção e adaptação. Qualquer alteração requer, por parte de quem não participou originalmente da sua codificação, um grande esforço no sentido de compreender o significado de cada bloco de código.

PUC-Rio - Certificação Digital Nº 0221221/CA

As técnicas de modelagem e de programação orientada a objetos (em oposição às técnicas de programação estruturada utilizadas em C e FORTRAN) presentes na linguagem C++, permitem que a modelagem de um problema seja feita de forma mais intuitiva, com a geração de um código mais compacto e cuja adaptação e manutenção se torna muito mais simples.

Na programação estruturada (também denominada procedural), um problema é subdividido em dois conjuntos de entidades principais e distintas:

- Um conjunto de dados.
- Um conjunto de funções e procedimentos que manipularão estes dados.

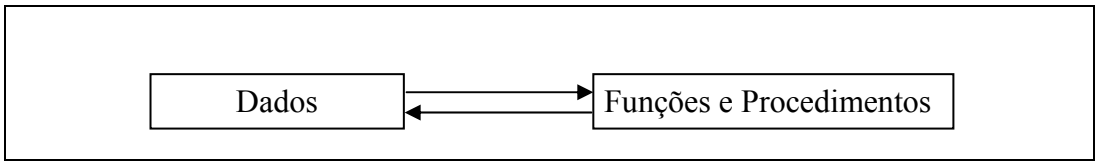

Figura 2 – Comunicação entre Dados e Funções na Programação Estruturada

Neste tipo de abordagem, os dados são passados como parâmetros às funções e procedimentos que os manipulam, como apresentado no diagrama da figura 2. Desta maneira, por exemplo, um elemento integrante de uma malha de elementos fititos e sua matriz de rigidez são entidades completamente distintas, embora uma matriz de rigidez seja uma característica intrínseca a um elemento.

Por outro lado, na programação orientada a objetos os dados e as funções e procedimentos que os manipulam são reunidos em uma única entidade denominada CLASSE, como mostra a figura 3.

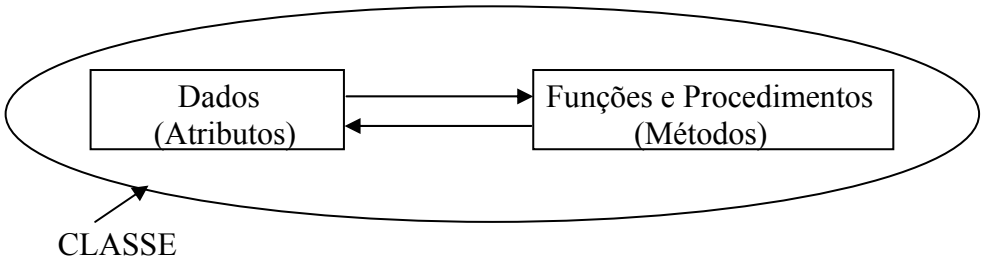

Figura 3 – Representação do Conceito de Classe

PUC-Rio - Certificação Digital Nº 0221221/CA

Desta maneira, um elemento integrante de uma malha de elementos finitos pode ser representado como uma instância de uma classe, reunindo as informações necessárias à sua completa definição (informações estas denominadas atributos) e as funções e procedimentos que manipulam tais informações (denominadas métodos no linguajar da programação orientada a objetos). Sua matriz de rigidez, por exemplo, pode ser definida como um de seus atributos, e as funções utilizadas na sua obtenção podem ser definidas como seus métodos.

Neste contexto, um objeto é definido como uma instância ou ocorrência real de uma classe. Pode-se estabelecer, portanto, uma analogia segundo a qual na programação orientada a objetos um objeto está para uma classe assim como na programação estruturada uma variável está para um tipo de dado. Pode-se afirmar, portanto, que um objeto é uma "variável" de uma classe (embora o correto, neste caso, seja afirmar que um objeto é uma instância de uma classe).

PUC-Rio - Certificação Digital Nº 0221221/CA

3.2
Características e Vantagens da Programação Orientada a Objetos

São características importantes da programação orientada a objetos:

- O conceito de Herança, que permite um reaproveitamento mais eficiente do código. A herança permite que se definam novas classes a partir de uma classe já existente (e denominada classe-base ou superclasse). Neste caso, todas as características da classe-base estarão também presentes nas classes dela derivadas por herança (também denominadas subclasses). Ao se criar uma nova classe derivada por herança de uma classe-base, apenas as alterações nas características já existentes na classe-base e a incorporação de novas características precisam ser implementadas, aproveitando-se integralmente os atributos e métodos já definidos.

- A sobrecarga de funções (ou métodos), que permite que se definam com um mesmo nome funções e procedimentos que recebem conjuntos de parâmetros distintos, mas que realizem tarefas semelhantes. Estas funções podem inclusive possuir parâmetros com valores padrão ou "default" (valores que serão assumidos caso não seja fornecido explicitamente um valor para o parâmetro).

- O conceito de polimorfismo: Característica segundo a qual métodos de mesmo nome podem realizar processamentos distintos, dependendo do objeto ao qual se aplicam.

- Sobrecarga de operadores, que nos permite definir de forma compacta as operações a serem realizadas sobre os objetos de uma classe. Esta característica é especialmente útil na implementação de uma classe criada para a representação de matrizes de números reais, apresentada no próximo capítulo. A sobrecarga de operadores permite que operações matriciais complexas sejam realizadas em uma única linha de código, facilitando o seu entendimento e manutenção.

Neste trabalho, apresenta-se uma proposta de modelagem de classes a serem utilizadas na análise de estruturas pelo método dos elementos finitos, adotando-se em sua implementação a linguagem C++, que possui todos os recursos descritos anteriormente. Considerando-se ainda que este tipo de implementação envolve uma grande quantidade de operações matriciais, desenvolveu-se também uma classe destinada a representar matrizes de números reais e suas operações.

PUC-Rio - Certificação Digital Nº 0221221/CA

4
Apresentação das Classes Implementadas

4.1
Introdução

Neste capítulo será apresentada uma descrição de cada uma das classes implementadas, além de sua funcionalidade no sistema. São apresentadas as classes criadas para a representação de entidades matriciais, a estrutura a ser analisada, e os pontos nodais, elementos, pontos de Gauss e materiais que a definem.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.2
Definição de uma classe destinada a representar entidades matriciais

A implementação de um programa para a análise de estruturas pelo método dos elementos finitos requer, inevitavelmente, um grande número de operações matriciais, como por exemplo:

- Montagem da matriz de rigidez local dos elementos.

- Montagem da matriz de rigidez global da estrutura, obtida a partir da reunião das contribuições das matrizes de rigidez dos diversos elementos que a compõem, considerando-se suas respectivas incidências.

- Solução do sistema de equações lineares, para obtenção dos deslocamentos correspondentes a um determinado carregamento aplicado.

- Montagem do vetor de forças externas aplicadas.

Considerando a grande quantidade de operações matriciais envolvidas, desenvolveu-se como parte deste trabalho uma classe destinada a representar matrizes de números reais e suas operações.

O recurso da sobrecarga de operadores presente na linguagem C++ permite que se definam, para a classe matriz, operadores que representem:

- A soma e subtração de matrizes, de maneira que se **A, B** e **C** forem definidos como objetos da classe matriz, representando matrizes de mesmas

dimensões, a definição de **C** como a soma ou subtração de **A** e **B** possa ser representada da seguinte maneira (em uma única linha de código):

C = A+B;

Ou

C = A-B;

- A multiplicação de matrizes, de maneira que se **A, B** e **C** forem definidos como objetos da classe matriz, onde **A** possui dimensões m x n, **B** possui dimensões n x p e **C** possui dimensões m x p, a definição de **C** como o produto das matrizes **A** e **B** possa ser representada da seguinte maneira (em uma única linha de código):

C = A*B;

- A solução de sistemas de equações lineares, de maneira que se **A, B** e **X** forem definidos como objetos da classe matriz, onde **A** representa uma matriz quadrada de ordem n, e **B** e **X** representa vetores-coluna com dimensões n x 1, a definição de **X** como a solução do sistema **A*X = B** possa ser representada da seguinte maneira (em uma única linha de código):

X = B/A;

- A transposição de matrizes, de maneira que se **A e B** forem definidos como objetos da classe matriz, onde **A** possui dimensões m x n e **B** possui dimensões n x m, a definição de **B** como a transposta de **A** possa ser representada da seguinte maneira (em uma única linha de código):

B = !A;

- A inversão de matrizes, de maneira que se **A e B** forem definidos como objetos da classe matriz, onde **A** e **B** representam matrizes quadradas de ordem n, a definição de **B** como a inversa de **A** possa ser representada da seguinte maneira (em uma única linha de código):

B = ~A;

- A multiplicação de uma matriz por um escalar, de maneira que se **A e B** forem definidos como objetos da classe matriz, onde **A** e **B** possuem dimensões m x n, e **c** for um número real (escalar) a definição de **B** como o produto do escalar **c** pela matriz **A** possa ser representada da seguinte maneira (em uma única linha de código):

B = c*A;

PUC-Rio - Certificação Digital Nº 0221221/CA

No que se refere à implementação da classe matriz, é importante observar que na linguagem C++ a alocação dinâmica de memória é efetuada utilizando-se o operador *new*. No caso de se desejar alocar dinamicamente um vetor de N objetos de um tipo predefinido ou de uma classe, a sintaxe a ser empregada é:

Nome_Da_Classe * Nome_Array;

Nome_Array = *new* Nome_Da_Classe[N];

É importante ainda considerar que na linguagem C++ o índice de um vetor é indexado a partir de zero. Conseqüentemente, o último elemento de um vetor unidimensional de dimensão N será aquele correspondente ao índice N-1. Embora esta seja uma fonte comum de erros, principalmente para os que estão começando a programar em C++, no caso da classe matriz esta característica estará oculta na definição da classe. Desta maneira, a declaração de uma matriz A de quatro linhas e cinco colunas pode ser feita em uma linha de código com a seguinte sintaxe:

matriz A(4,5);

No caso de se utilizar vetores de outros tipos de objetos, uma solução alternativa consiste em se alocar um elemento adicional e desprezar o elemento de índice 0, trabalhando apenas com os elementos cujo índice varia entre 1 e N (ocorrendo, no entanto, um pequeno desperdício de memória, correspondente ao elemento de índice 0). Esta alternativa será utilizada na implementação posterior de vetores de objetos das diversas classes usadas na definição de problema, optando-se por simplificar a sua codificação e posterior manutenção.

Na implementação da classe matriz, o armazenamento dos elementos da matriz será feito em um vetor unidimensional definido internamente à classe. Este vetor será denominado *elementos*, e se *m* e *n* forem números inteiros que representam respectivamente o número de linhas e de colunas da matriz, a alocação dinâmica de memória para os *m* x *n* elementos da matriz será feito em uma linha de código com a seguinte sintaxe:

elementos = *new* double[m*n];

Observe que este código será definido internamente à classe, sendo transparente para o usuário que irá utilizá-la em uma posterior implementação. Faz-se um mapeamento interno entre os elementos de uma matriz bidimensional e os elementos deste vetor unidimensional.

A implementação da classe matriz envolve a definição de seus atributos, métodos e operadores sobrecarregados. Com base no que foi exposto

PUC-Rio - Certificação Digital Nº 0221221/CA

anteriormente fica clara, portanto, a necessidade de se definir os seguintes atributos para a classe matriz:

- Um campo denominado num_linhas, destinado a armazenar o número de linhas da matriz, e representado por uma variável inteira de mesmo nome.

- Um campo denominado num_colunas, destinado a armazenar o número de colunas da matriz, e representado por uma variável inteira de mesmo nome.

- Um campo denominado elementos, destinado a armazenar um ponteiro para uma variável do tipo double (real de precisão dupla), a partir do qual será alocado dinamicamente um vetor para armazenamento dos elementos da matriz.

A classe matriz terá os seguintes métodos:

- Um método construtor, destinado a atribuir os valores iniciais a diversos atributos da classe, e a alocar memória para o vetor de números reais destinado a armazenar os elementos da matriz.

- Um método destrutor, destinado a liberar a memória alocada para o vetor de números reais destinado a armazenar os elementos da matriz.

- Um método destinado a atribuir um valor a um elemento da matriz representada por um objeto da classe.

- Um método destinado a adicionar um valor a um elemento da matriz representada por um objeto da classe.

- Um método destinado a redimensionar a matriz representada por um objeto da classe (redefinindo o número de linhas e de colunas da matriz, alterando-se os valores armazenados nos atributos correspondentes e redefinindo-se a memória alocada para o vetor no qual os elementos da matriz serão armazenados).

- Um método destinado a calcular o valor do determinante da matriz representada por um objeto da classe, no caso de matrizes quadradas.

- Um método destinado a obter a inversa da matriz representada por um objeto da classe, no caso de matrizes quadradas.

- Um método destinado a obter a transposta da matriz representada por um objeto da classe.

- Um método destinado a transformar em uma matriz identidade a matriz representada por um objeto da classe.

A classe matriz terá ainda os seguintes operadores definidos (ou sobrecarregados):

PUC-Rio - Certificação Digital Nº 0221221/CA

- Um operador destinado a representar a soma de matrizes representadas por objetos da classe.

- Um operador destinado a representar a subtração de matrizes representadas por objetos da classe.

- Um operador destinado a representar a multiplicação de matrizes representadas por objetos da classe.

- Um operador destinado a representar a solução de sistemas de equações definidos por matrizes representadas por objetos da classe.

- Um operador destinado a representar a transposição de uma matriz representada por um objeto da classe.

- Um operador destinado a representar a inversão de uma matriz representada por um objeto da classe.

- Operadores destinados a permitir a atribuição direta de um valor a um elemento de uma matriz representada por um objeto da classe.

A definição da classe matriz, é reproduzida na figura 4:

```
class matriz
{
   public:
     int num_linhas, num_colunas; // Número de Linhas e de Colunas da matriz
     double *elementos; // Armazena os elementos da matriz
     matriz(); // Construtor Default
     ~matriz(); // Destrutor da Classe
     matriz(int m, int n = 1); // Construtor com Dimensões da Matriz
     double elemento(int i, int j = 1) const;// retorna um elemento da matriz
     void Atribui(int i, int j, double valor); // Atribui valor a um elemento da matriz;
     void Atribui(int i, double valor); // Atribui valor a um elemento da matriz;
     void Adiciona(int i, int j, double valor); // Adiciona valor a um elemento da matriz;
     void Adiciona(int i, double valor); // Adiciona valor a um elemento da matriz;
     void redim(int m, int n = 1); // Redimensiona a Matriz
     double Determinante(); // Retorna o Determinante da Matriz
     matriz Inversa(); // Obtém a matriz Inversa
     matriz Transposta();// Obtém a matriz Transposta
     void ZeraMatriz(); // Zera os elementos da Matriz
     void TransFormaEmIdentidade();// Transforma a Matriz em uma matriz Identidade.
     matriz operator = (matriz B);// Operador de Atribuição.
     matriz operator +(matriz B);
```

PUC-Rio - Certificação Digital Nº 0221221/CA

```
    void operator +=(matriz B);
    matriz operator -(matriz B);
    void operator -=(matriz B);
    matriz operator *(matriz B);
    void operator *=(matriz B);
    matriz operator *(double fator);
    void operator *=(double fator);
    matriz operator /(matriz A);
    void operator /=(matriz A);
    matriz operator -();
    bool operator ==(matriz B);
    bool operator !=(matriz B);
    matriz operator !(); // Transposta da matriz
    matriz operator ~(); // Inversa da matriz
    double& operator()(int i, int j = 1);
    double& operator()(double i, double j = 1);
    void operator()(int i, double valor);
    void operator()(int i, int j, double valor);
};
```

Figura 4 – Definição da classe matriz.

Repare que na definição da classe também foram sobrecarregados os operadores compostos da linguagem C++.

4.3
Definição das classes fundamentais à análise linear de estruturas

A análise de estruturas pelo método dos elementos finitos envolve fundamentalmente a consideração das seguintes entidades:

- Estrutura: Corpo sólido deformável a ser analisado, constituído por um ou mais tipos de materiais, submetido a um conjunto de forças externas diretamente aplicadas, e ocupando uma região do espaço (domínio) que pode ser subdividida em um conjunto de sub-regiões denominadas elementos (subdomínios), sendo estas sub-regiões interconectadas através de pontos especiais denominados nós ou pontos nodais. Serão também nestes pontos que estarão atuando as forças externas diretamente aplicadas à estrutura, bem como serão aplicadas as condições de contorno na forma de restrições a deslocamentos (e conseqüente reações de apoio).

Conseqüentemente, esta entidade (estrutura) é forte candidata a ser uma classe, possuindo entre os seus atributos iniciais os elementos e os nós que a

PUC-Rio - Certificação Digital Nº 0221221/CA

definem. Serão ainda considerados atributos adicionais auxiliares, como valores que definem o número de elementos, de pontos nodais e de tipos de materiais, e matrizes que representam os vetores de cargas externas aplicadas, deslocamentos nodais, e sua matriz de rigidez.

- Nós: Pontos da estrutura através dos quais serão interconectados os diversos elementos que representam suas sub-regiões, e nos quais serão aplicadas as forças externas e as condições de contorno, na forma de restrições a deslocamentos. Conseqüentemente, esta entidade (nó) é forte candidata a ser uma classe, possuindo inicialmente atributos que identifiquem sua posição inicial, as cargas diretamente aplicadas, seus deslocamentos e as restrições a eles impostas.

- Elementos: Conforme descrito anteriormente, cada uma das sub-regiões em que uma estrutura é dividida pode ser identificada como um elemento.

Estes elementos são constituídos por um tipo de material e estão interligados ou interconectados por pontos especiais denominados nós (já definidos anteriormente). Conseqüentemente, esta entidade (elemento) é também forte candidata a ser uma classe, possuindo inicialmente atributos que identifiquem os nós aos quais estão ligados, uma variável que identifique o seu tipo de material, e os diversos pontos de Gauss sobre os quais serão feitas as integrações numéricas necessárias à obtenção das matrizes características de cada elemento, particularmente a sua matriz de rigidez local.

- Material: Conjunto de características que definem o comportamento de pelo menos uma das sub-regiões (elementos) em que a estrutura é dividida. Estas características correspondem às propriedades constitutivas do material que constitui o elemento, como os módulos de elasticidade longitudinal e transversal e o coeficiente de Poisson.

- Ponto de Gauss: A integração das parcelas que definem a equação de governo do problema (no nosso caso, o equilíbrio da estrutura) é feita numericamente, utilizando-se o método da quadratura de Gauss, a partir de valores das funções que representam as grandezas de interesse em pontos específicos destas sub-regiões ou elementos, sendo estes pontos específicos denominados pontos de Gauss. Conseqüentemente esta entidade (ponto de Gauss) é também forte candidata a definir uma classe para o nosso problema.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.3.1
Definição da Classe Nó

A classe Nó, utilizada na representação dos pontos nodais da estrutura, terá inicialmente, os seguintes atributos:

- x , y: Atributos definidos como variáveis reais de precisão dupla, e destinados a armazenar as coordenadas x e y de cada nó da estrutura.

- dx , dy: Atributos definidos como variáveis reais de precisão dupla, e destinados a armazenar os deslocamentos de cada nó da estrutura.

- Fx , Fy: Atributos definidos como variáveis reais de precisão dupla, e destinados a armazenar as cargas externas diretamente aplicadas a cada nó da estrutura.

- restx , resty: Atributos definidos como variáveis booleanas, e destinados a informar se há ou não restrição de deslocamentos para cada nó da estrutura. Caso não haja restrição de deslocamento, será atribuído o valor 0. Um valor unitário identifica uma restrição.

- prescx , prescy: Atributos definidos como variáveis reais de precisão dupla, e destinados a armazenar os valores dos deslocamentos prescritos em cada nó da estrutura.

A classe Nó terá, inicialmente, os seguintes métodos:

- Um método construtor, destinado a atribuir valores iniciais aos atributos da classe, durante a criação de objetos.

- Um método destinado a gravar, no arquivo de resultados, os deslocamentos nodais.

A definição desta classe é reproduzida na figura 5:

```
class No
{
 public:
   double x, y;// Coordenadas
   double Fx, Fy; //Cargas Nodais
   bool restx, resty; // Restrições
   double dx, dy; //Deslocamentos Nodais
   double prescx, prescy; //Deslocamentos prescritos;
   No();// Construtor da Classe
   void GeraResultados(ofstream &arquivo_saida);// Geração de
resultados
};
```

Figura 5 – Definição da classe Nó.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.3.2
Definição da Classe Material

A definição desta classe é muito simples, pois não possui métodos (apenas atributos), sendo reproduzida na figura 6:

```
class Material
{
 public:
   double POISS, E;// Coeficiente de Poisson e Módulo de Elasticidade
};
```

Figura 6 – Definição da classe Material.

4.3.3
Definição da Classe PontoDeGauss

A classe PontoDeGauss, como o próprio nome indica, permite instanciar objetos que representarão os pontos de Gauss dos elementos, e sobre os quais será feita a integração numérica, inicialmente para a obtenção da matriz de rigidez de cada um destes elementos.

A classe PontoDeGauss terá, inicialmente, os seguintes atributos:

- XI, ETA: Atributos definidos como uma variável real de precisão dupla, que armazenam os valores das coordenadas locais do ponto de Gauss.

- WXI, WETA: Atributos definidos como uma variável real de precisão dupla, que armazenam os valores (pesos) a serem multiplicados na obtenção de diversas grandezas por integração numérica no ponto de Gauss.

- D: Atributo definido como um objeto da classe matriz, destinado a representar a matriz constitutiva do material do elemento ao qual pertence o ponto de Gauss. Evidentemente, esta matriz será função do tipo de material e do tipo de análise adotada (estado plano de tensões, estado plano de deformações ou axissimétrico).

- K: Atributo definido como um objeto da classe matriz, destinado a representar a parcela de contribuição do ponto de Gauss na obtenção da matriz de rigidez do elemento ao qual pertence o ponto de Gauss.

PUC-Rio - Certificação Digital Nº 0221221/CA

- Fint: Atributo definido como um objeto da classe matriz, destinado a representar a parcela de contribuição do ponto de Gauss na obtenção da matriz que representa o vetor de forças internas do elemento ao qual pertence o ponto de Gauss.

- LN: Atributo definido como um objeto da classe matriz, destinado a representar internamente o produto das matrizes que representam o operador [L] e as funções de interpolação [N], para a obtenção da matriz [B].

- B: Atributo definido como um objeto da classe matriz, destinado a representar a relação entre deformações e deslocamentos, para o caso de análise linear (pequenas deformações).

- u: Atributo definido como um objeto da classe matriz, na forma de um vetor (matriz unidimensional), e destinado a representar os deslocamentos dos pontos nodais pertencentes ao contorno do elemento ao qual pertence o ponto de Gauss.

- sigma: Atributo definido como um objeto da classe matriz, na forma de um vetor (matriz unidimensional), e destinado a representar as tensões atuantes no ponto de Gauss.

- J: Atributo definido como um objeto da classe matriz, destinado a representar o operador Jacobiano, usado na transformação das derivadas das funções de forma entre coordenadas globais e locais de pontos interiores ao elemento ao qual pertence o ponto de Gauss.

O operador Jacobiano, conforme descrito anteriormente, permite a transformação das derivadas das funções de forma entre coordenadas dos sistemas local a um elemento e global da estrutura.

Como a integração será feita considerando-se os diversos valores de interesse e das funções de forma e suas derivadas para o sistema local ao elemento ao qual pertence o ponto de Gauss, a expressão correta para a matriz [G] será:

$$[G]=\begin{bmatrix}[J^{-1}] & [0] \\ [0] & [J^{-1}]\end{bmatrix}\begin{bmatrix}\frac{\partial N1}{\partial \xi} & 0 & \frac{\partial N2}{\partial \xi} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial \eta} & 0 & \frac{\partial N2}{\partial \eta} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial \xi} & 0 & \frac{\partial N2}{\partial \xi} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial \eta} & 0 & \frac{\partial N2}{\partial \eta} & \dots & \dots\end{bmatrix} \tag{4.1}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

onde [0] representa uma matriz 2 x 2 em que todos elementos são iguais a zero.

Nesta implementação, conforme mostrado na equação anterior, a matriz que representa o operador Jacobiano será portanto considerada como uma matriz 4 x 4. Conseqüentemente, na integração efetuada para cálculo da matriz de rigidez no sistema local, o valor da contribuição no ponto de Gauss deverá ser multiplicado pela raiz quadrada do determinante do Jacobiano, e não pelo simples valor do seu determinante.

A classe PontoDeGauss terá, inicialmente, os seguintes métodos:

- Um método denominado Inicializa, que recebe como parâmetro o número de pontos nodais que definem o elemento e, a partir deste valor, dimensiona corretamente os diversos atributos definidos como sendo objetos da classe matriz.

- Um método denominado Calcula_D, destinado a calcular a matriz constitutiva no ponto considerado, função do material do elemento que contém o ponto de Gauss e do tipo de análise. Este método receberá como parâmetro um ponteiro para o objeto que representa o material do elemento, a partir do qual serão obtidas as informações necessárias referentes as constantes utilizadas na obtenção da matriz.

- Um método denominado Calcula_B, destinado a calcular a matriz [B] que relaciona os deslocamentos nodais em um ponto qualquer (no caso o ponto de Gauss) com suas respectivas deformações. Este método receberá como parâmetros um valor inteiro que identifica a posição do ponto de Gauss no elemento (o que permitirá a obtenção das suas coordenadas), um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss (este valor definirá as funções de forma a serem empregadas, bem como suas derivadas) e um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, para que possam ser obtidas as informações de interesse, correspondentes aos nós do elemento ao qual pertence o ponto de Gauss.

- Um método denominado Calcula_K, destinado a calcular a contribuição do ponto de Gauss representado pelo objeto à matriz de rigidez do elemento ao qual pertence o ponto de Gauss. Este método receberá como parâmetros um valor inteiro que identifica a posição do ponto de Gauss no elemento (o que permitirá a

PUC-Rio - Certificação Digital Nº 0221221/CA

obtenção das suas coordenadas), um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss (este valor definirá as funções de forma a serem empregadas, bem como suas derivadas), um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, para que possam ser obtidas as informações de interesse, correspondentes aos nós do elemento ao qual pertence o ponto de Gauss, e um ponteiro para o objeto que representa o material deste elemento. Inicialmente este método fará uma chamada aos métodos Calcula_D, descrito anteriormente, para a obtenção da matriz [D], e Calcula_Tensoes, que calculará a matriz [B] e as tensões no ponto de Gauss. A partir dos valores das matrizes [B] e [D], poderá ser obtido, por integração numérica, o valor da matriz [K].

- Um método denominado Calcula_J, destinado a calcular o operador Jacobiano no ponto considerado. Este método receberá como parâmetros um valor inteiro que identifica a posição do ponto de Gauss no elemento (o que permitirá a obtenção das suas coordenadas), um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss (este valor definirá as funções de forma a serem empregadas, bem como suas derivadas) e um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, para que possam ser obtidas as informações de interesse, correspondentes aos nós do elemento ao qual pertence o ponto de Gauss.

- Um método denominado Calcula_Tensoes, destinado a calcular as tensões no ponto de Gauss representado pelo objeto. Este método receberá como parâmetros um valor inteiro que identifica a posição do ponto de Gauss no elemento (o que permitirá a obtenção das suas coordenadas), um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss (este valor definirá as funções de forma a serem empregadas, bem como suas derivadas), um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, para que possam ser obtidas as informações de interesse, correspondentes aos nós do elemento ao qual pertence o ponto de Gauss, e um ponteiro para o objeto que representa o material deste elemento.

- Um método denominado GeraResultados, capaz de gravar no arquivo de saída os valores de interesse no ponto de Gauss.

- Um método denominado CalculaForcasInternas, destinado a calcular a contribuição do ponto de Gauss representado pelo objeto ao vetor de forças

PUC-Rio - Certificação Digital Nº 0221221/CA

internas do elemento ao qual pertence o ponto de Gauss. Este método receberá como parâmetros um valor inteiro que identifica a posição do ponto de Gauss no elemento (o que permitirá a obtenção das suas coordenadas), um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss (este valor definirá as funções de forma a serem empregadas, bem como suas derivadas), um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, para que possam ser obtidas as informações de interesse, correspondentes aos nós do elemento ao qual pertence o ponto de Gauss, e um ponteiro para o objeto que representa o material deste elemento. Inicialmente este método fará uma chamada ao método CalculaTensoes, descrito anteriormente, para a obtenção da matriz [B] e do vetor de tensões {σ} no ponto de Gauss. A partir dos valores das matrizes [B] e {σ} poderá ser obtido, por integração numérica, o valor da contribuição do ponto de Gauss ao vetor de forças internas do elemento - {Fint}.

- Um método denominado CoordenadasDoPontoDeGauss, que recebe como parâmetro um valor inteiro que identifica a posição do ponto de Gauss no elemento, e a partir da qual são definidos, neste método, os valores das suas coordenadas no sistema local ao elemento ao qual pertence o ponto de Gauss.

- Um método auxiliar, denominado COORDENADAXI, criado com a finalidade de simplificar a implementação de funções de forma. Este método recebe como parâmetro um valor inteiro que representa o número que identifica o ponto nodal no elemento e retorna um valor a ser usado como fator multiplicativo na função de forma.

- Um método auxiliar, denominado COORDENADAETA, criado com a finalidade de simplificar a implementação de funções de forma. Este método recebe como parâmetro um valor inteiro que representa o número que identifica o ponto nodal no elemento e retorna um valor a ser usado como fator multiplicativo na função de forma.

- Um método denominado SHAPE, que calcula o valor da função de forma correspondente a um dos pontos nodais nas coordenadas do ponto de Gauss. Este método receberá como parâmetros um valor inteiro que identifica o ponto nodal ao qual se refere a função de forma, um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss, e dois

PUC-Rio - Certificação Digital Nº 0221221/CA

números reais que definem as coordenadas do ponto de Gauss no sistema local ao elemento.

- Um método denominado LSHAPEXI, que calcula o valor da derivada da função de forma em relação à variável que define a abscissa no sistema local, e correspondente a um dos pontos nodais, nas coordenadas do ponto de Gauss. Este método receberá como parâmetros um valor inteiro que identifica o ponto nodal ao qual se refere a função de forma, um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss, e dois números reais que definem as coordenadas do ponto de Gauss no sistema local ao elemento.

- Um método denominado LSHAPEETA, que calcula o valor da derivada da função de forma em relação à variável que define a ordenada no sistema local, e correspondente a um dos pontos nodais, nas coordenadas do ponto de Gauss. Este método receberá como parâmetros um valor inteiro que identifica o ponto nodal ao qual se refere a função de forma, um valor inteiro que informa o número de pontos nodais que definem o elemento ao qual pertence o ponto de Gauss, e dois números reais que definem as coordenadas do ponto de Gauss no sistema local ao elemento.

É importante destacar que a classe PontoDeGauss não será usada para instanciar diretamente qualquer objeto. Ao invés disso, ela servirá como classe-base para as classes PontoDeGauss_EPT (usada na análise de problemas que envolvam um estado plano de tensões), PontoDeGauss_EPD (usada na análise de problemas que envolvam um estado plano de deformações), e PontoDeGauss_AXISSIMETRICO (usada na análise de problemas que envolvam um estado axissimétrico de tensões), que serão derivadas, por herança, da classe PontoDeGauss.

Nestas situações, em que uma classe não instancia nenhum objeto, a programação orientada a objetos permite que a mesma seja definida como uma classe composta apenas por métodos virtuais – definidos como métodos declarados na classe-base mas implementados apenas nas classes dela derivadas por herança. Neste caso, torna-se inclusive desnecessária a definição de métodos construtores e destrutores, já que a classe não instanciará nenhum objeto.

PUC-Rio - Certificação Digital Nº 0221221/CA

A definição desta classe é reproduzida na figura 7:

```
class PontoDeGauss
{
 public:
   double coordenada;
   double XI, ETA; // Coordenadas Locais do ponto de gauss.
   double WXI, WETA; // Pesos do ponto de gauss.
   matriz D; // Matriz Constitutiva do Material
   matriz K; // Matriz de Rigidez
   matriz Fint; // Vetor de Forças Internas
   matriz LN; // Matriz Resultante da Multiplicação do Operador [L]
            // Pela Matriz de Funções de Interpolação [N]
   matriz B; // Matriz que Associa Deformações a Deslocamentos
           // [B] = Inversa(J)*[L]{N]
   matriz J; // Matriz Jacobiano Responsável pela Mudança de
Coordenadas
   matriz u; // Matriz de Deslocamentos Dos Pontos Nodais
   matriz sigma;// Vetor de Tensões no Ponto de Gauss
   virtual void Inicializa(int NumeroDePontosNodais){};
   virtual void Calcula_D(Material &Mat){};// Calcula a Matriz
Constitutiva
   virtual void Calcula_B(int i, int NumeroDePontosNodais, No * Nos){};
   // Calcula a Matriz [B]
   virtual void Calcula_K(int i, int NumeroDePontosNodais, No * Nos,
Material &Mat){};
   // Calcula a Matriz de Rigidez
   virtual void Calcula_J(int i, int NumeroDePontosNodais, No * Nos){};
   // calcula o Jacobiano
   virtual void Calcula_Tensoes(int i, int NumeroDePontosNodais, No *
Nos, Material &Mat){};
   // Calcula as Tensões no Ponto de Gauss
   virtual void CalculaForcasInternas(int i, int NumeroDePontosNodais,
No * Nos, Material &Mat){};
   // Calcula as Forças Internas no Ponto de Gauss
   virtual void GeraResultados(){};
   virtual void CoordenadasDoPontoDegauss(int IGAUSS){};
   virtual double COORDENADAXI(int i){};
   virtual double COORDENADAETA(int i){};
   virtual double SHAPE(int i, int NumeroDePontosNodais, double XI,
double ETA){};
   virtual double LSHAPEXI(int i, int NumeroDePontosNodais, double
XI, double ETA){};
   virtual double LSHAPEETA(int i, int NumeroDePontosNodais, double
XI, double ETA){};
};
```

Figura 7 – Definição da classe PontoDeGauss.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.3.4
Definição da Classe PontoDeGauss_EPT

A classe PontoDeGauss_EPT permite instanciar objetos que representarão os pontos de Gauss de elementos submetidos a um estado plano de tensão, sendo derivada por herança da classe PontoDeGauss.

Esta classe herda todos os atributos e métodos já definidos para a classe PontoDeGauss, não havendo a necessidade de se redefinir qualquer atributo ou método adicional, excetuando-se apenas o seu método construtor.

Ocorre, no entanto, que como serão instanciados objetos desta classe, esta será responsável por implementar os métodos que foram apenas declarados na sua classe-base.

A classe PontoDeGauss_EPT é definida como uma classe derivada, por herança, da classe-base PontoDeGauss.

A definição desta classe é reproduzida na figura 8:

```
class PontoDeGauss_EPT : public PontoDeGauss
{
 public:
   PontoDeGauss_EPT();
   // Implementação dos Métodos Herdados da Classe-Base
   virtual void Inicializa(int NumeroDePontosNodais);
   virtual void Calcula_D(Material &Mat);
   virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No *
Nos);// Calcula a Matriz [B]
   virtual void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No *
Nos, Material &Mat);
   virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No *
Nos);
   virtual void Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais,
No * Nos, Material &Mat);
   virtual void CalculaForcasInternas(int IGAUSS, int
NumeroDePontosNodais, No * Nos, Material &Mat);
   virtual void CoordenadasDoPontoDegauss(int IGAUSS);
   virtual void GeraResultados();
   virtual double COORDENADAXI(int i);
   virtual double COORDENADAETA(int i);
   virtual double SHAPE(int i, int NumeroDePontosNodais, double XI,
double ETA);
   virtual double LSHAPEXI(int i, int NumeroDePontosNodais, double
XI, double ETA);
   virtual double LSHAPEETA(int i, int NumeroDePontosNodais, double
XI, double ETA);
};
```

Figura 8 – Definição da classe PontoDeGauss_EPT.

PUC-Rio - Certificação Digital Nº 0221221/CA

Apenas os métodos definidos como virtuais na classe-base são redeclarados, pois os atributos são integralmente herdados da sua classe-base.

A definição da classe PontoDeGauss_EPT como uma classe derivada da classe-base PontoDeGauss é indicada na seguinte linha de código:

class PontoDeGauss_EPT : public PontoDeGauss

4.3.5
Definição da Classe PontoDeGauss_EPD

A classe PontoDeGauss_EPD permite instanciar objetos que representarão os pontos de Gauss dos elementos submetidos a um estado plano de deformação.

A principal diferença deste tipo de análise em relação àquela em que se considera um estado plano de tensões está na definição da matriz constitutiva, que no caso de um estado plano de deformação é definida como:

$$[D] = \frac{E}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1-\upsilon & \upsilon & 0 \\ \upsilon & 1-\upsilon & 0 \\ 0 & 0 & \frac{(1-\upsilon)}{2} \end{bmatrix} \tag{4.2}$$

Deve-se considerar, ainda, a existência de um quarto componente de tensões σ_z, cujo valor depende daqueles calculados para as componentes σ_x e σ_y.

Esta classe herda todos os atributos e métodos já definidos para a classe PontoDeGauss_EPT, não havendo a necessidade de se definir qualquer atributo ou método adicional, havendo no entanto a necessidade de se redefinir os métodos Calcula_D, que obtém, a partir dos atributos do objeto da classe Material, os valores da matriz [D] para um estado plano de deformações, e o método Inicailiza, que redimensiona corretamente os atributos definidos como objetos da classe matriz, e o seu método construtor.

A classe PontoDeGauss_EPD é definida como uma classe derivada, por herança, da classe-base PontoDeGauss_EPT.

A definição desta classe é reproduzida na figura 9:

PUC-Rio - Certificação Digital Nº 0221221/CA

```
class PontoDeGauss_EPD : public PontoDeGauss_EPT
{
 public:
   PontoDeGauss_EPD();
   virtual void Calcula_D(Material &Mat);
   virtual void Inicializa(int NumeroDePontosNodais);
};
```

Figura 9 – Definição da classe PontoDeGauss_EPD.

4.3.6
Definição da Classe PontoDeGauss_AXISSIMETRICO

A classe PontoDeGauss_AXISSIMETRICO permite instanciar objetos que representarão os pontos de Gauss dos elementos submetidos a um estado axissimétrico de tensões.

A principal diferença deste tipo de análise em relação àquela em que se considera um estado plano de tensões está na definição dos campos de deformações e de tensões, e da matriz constitutiva do material que é, neste caso, definida pela expressão:

$$[D] = \frac{E(1-\upsilon)}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1 & \frac{\upsilon}{(1-\upsilon)} & 0 & \frac{\upsilon}{(1-\upsilon)} \\ \frac{\upsilon}{(1-\upsilon)} & 1 & 0 & \frac{\upsilon}{(1-\upsilon)} \\ 0 & 0 & \frac{(1-2\upsilon)}{2(1-\upsilon)} & 0 \\ \frac{\upsilon}{(1-\upsilon)} & \frac{\upsilon}{(1-\upsilon)} & 0 & 1 \end{bmatrix} \tag{4.3}$$

Neste tipo de análise, as matriz {ε} e {σ} são, respectivamente:

$$\{\sigma\} = \begin{bmatrix} \sigma_r \\ \sigma_z \\ \tau_{rz} \\ \sigma_\theta \end{bmatrix} \text{ e } \{\varepsilon\} = \begin{bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_{rz} \\ \varepsilon_\theta \end{bmatrix} \tag{4.4}$$

relacionados pela equação

PUC-Rio - Certificação Digital Nº 0221221/CA

$$\{\sigma\} = [D]\{\varepsilon\} \tag{4.5}$$

As deformações e deslocamentos, neste caso, estão relacionados pelas seguintes equações:

$$\{\varepsilon\} = \begin{bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_{rz} \\ \varepsilon_\theta \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial w} + \frac{\partial w}{\partial r} \\ \frac{u}{r} \end{bmatrix} == \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{r} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial z} \\ \frac{\partial w}{\partial r} \\ \frac{\partial w}{\partial z} \\ u \end{bmatrix} = \lfloor H \rfloor \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial z} \\ \frac{\partial w}{\partial r} \\ \frac{\partial w}{\partial z} \\ u \end{bmatrix} \tag{4.6}$$

onde, neste tipo de análise:

$$[H] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{r} \end{bmatrix} \tag{4.7}$$

logo:

$$\begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial z} \\ \frac{\partial w}{\partial r} \\ \frac{\partial w}{\partial z} \\ u \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial y} & 0 \\ 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial y} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} N1 & 0 & N2 & 0 & \dots & \dots \\ 0 & N1 & 0 & N2 & \dots & \dots \end{bmatrix} \begin{bmatrix} U1 \\ W1 \\ U2 \\ W2 \\ \dots \\ .. \end{bmatrix} =$$

$$\begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \dots & \dots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \dots & \dots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \dots & \dots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & .... & \dots \\ N1 & 0 & N2 & 0 & \dots & \dots \end{bmatrix} \begin{bmatrix} U1 \\ W1 \\ U2 \\ W2 \\ \dots \\ .. \end{bmatrix} \tag{4.8}$$

ou:

PUC-Rio - Certificação Digital Nº 0221221/CA

$$\begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial z} \\ \frac{\partial w}{\partial r} \\ \frac{\partial w}{\partial z} \\ u \end{bmatrix} = [G] \begin{bmatrix} \mathrm{U1} \\ \mathrm{W1} \\ \mathrm{U2} \\ \mathrm{W2} \\ \ldots \\ .. \end{bmatrix}, \text{ neste caso : } [\mathrm{G}] = \begin{bmatrix} \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & 0 & \ldots & \ldots \\ \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & 0 & \ldots & \ldots \\ 0 & \frac{\partial N1}{\partial x} & 0 & \frac{\partial N2}{\partial x} & \ldots & \ldots \\ 0 & \frac{\partial N1}{\partial y} & 0 & \frac{\partial N2}{\partial y} & .... & \ldots \\ N1 & 0 & N2 & 0 & \ldots & \ldots \end{bmatrix} \tag{4.9}$$

Entretanto, considerando-se que a integração numérica será feita no sistema local do elemento ao qual pertence o ponto de Gauss, devemos reescrever a matriz [G] da seguinte maneira:

$$[G] = \begin{bmatrix} [J^{-1}] & \begin{bmatrix} 0 \end{bmatrix} & 0 \\ \begin{bmatrix} 0 \end{bmatrix} & [J^{-1}] & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\partial N1}{\partial \xi} & 0 & \frac{\partial N2}{\partial \xi} & 0 & \ldots & \ldots \\ \frac{\partial N1}{\partial \eta} & 0 & \frac{\partial N2}{\partial \eta} & 0 & \ldots & \ldots \\ 0 & \frac{\partial N1}{\partial \xi} & 0 & \frac{\partial N2}{\partial \xi} & \ldots & \ldots \\ 0 & \frac{\partial N1}{\partial \eta} & 0 & \frac{\partial N2}{\partial \eta} & \ldots & \ldots \\ N1 & 0 & N2 & 0 & \ldots & \ldots \end{bmatrix} \tag{4.10}$$

onde [0] representa uma matriz 2 x 2 em que todos elementos são iguais a zero.

Fica clara, portanto, a necessidade de se redefinir alguns métodos da classe-base. Esta classe herda todos os atributos e métodos já definidos para a classe PontoDeGauss_EPT, não havendo a necessidade de se definir qualquer atributo adicional, havendo no entanto a necessidade de se redefinir os métodos Inicializa(), Calcula_B, Calcula_D, Calcula_J e Calcula_K. Além disso, deve-se também implementar explicitamente o seu construtor default.

A classe PontoDeGauss_AXISSIMETRICO é definida como uma classe derivada, por herança, da classe-base PontoDeGauss_EPT.

A definição desta classe é reproduzida na figura 10:

PUC-Rio - Certificação Digital Nº 0221221/CA

```
class PontoDeGauss_AXISSIMETRICO : public PontoDeGauss_EPT
{
 public:
   PontoDeGauss_AXISSIMETRICO();
   virtual void Calcula_D(Material &Mat);
   virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No *
Nos);// Calcula a Matriz [B]
   virtual void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No *
Nos, Material &Mat);
   virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No *
Nos);
   virtual void Inicializa(int NumeroDePontosNodais);
};
```

Figura 10 – Definição da classe PontoDeGauss_AXISSIMETRICO.

Para as classes implementadas, pode-se então estabelecer o diagrama de hierarquia mostrado na figura 11:

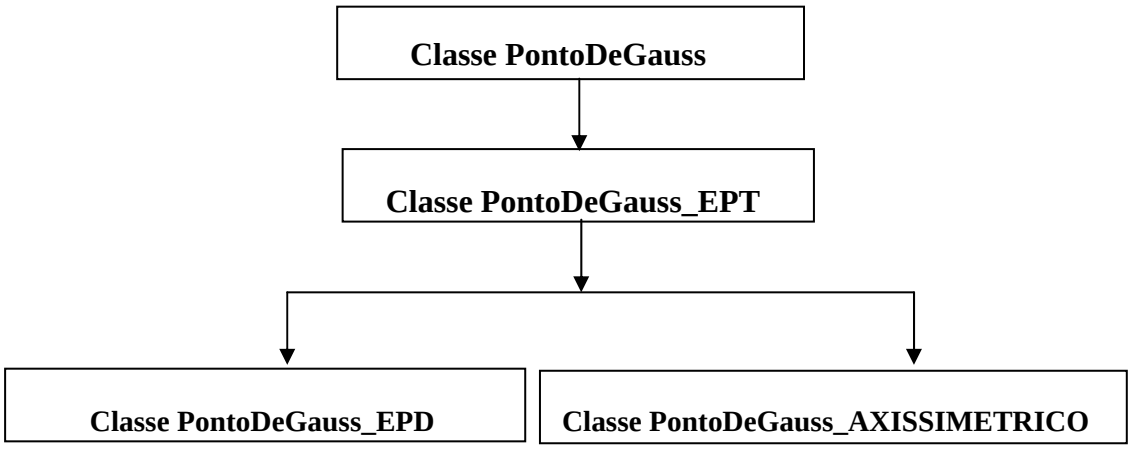

Figura 11 – Diagrama de hierarquia de classes.

4.3.7
Definição da Classe Elemento

A classe Elemento, como o próprio nome indica, permite instanciar objetos que representarão os elementos que representam as sub-regiões na qual a estrutura é subdividida.

Cada elemento será definido por um conjunto de pontos nodais e possuirá, em seu interior, pontos de Gauss sobre os quais serão calculadas as parcelas

PUC-Rio - Certificação Digital Nº 0221221/CA

necessárias ao cálculo numérico da sua contribuição, por exemplo, à matriz de rigidez da estrutura.

A classe Elemento terá, inicialmente, os seguintes atributos:

- NumeroDePontosNodais: Atributo definido como uma variável inteira, que armazena o número de pontos nodais que definem o elemento.

- NumeroDePontosDeGauss: Atributo definido como uma variável inteira, que armazena o número de pontos de Gauss definidos para o elemento.

- TipoMaterial: Atributo definido como uma variável inteira, que identifica o tipo de material do elemento.

- NP: Atributo definido como um ponteiro para um vetor de números inteiros, e que armazenará a incidência dos nós do elemento.

- NoLocal: Atributo definido como um ponteiro para um vetor de objetos da classe No, que armazenará os endereços dos objetos que representam os pontos nodais que definem o elemento.

- IND: Atributo definido como um objeto da classe matriz, destinado a armazenar as incidências do elemento, isto é, a relação entre coordenadas locais ao elemento e as globais à estrutura.

- K: Atributo definido como um objeto da classe matriz, destinado a representar a matriz de rigidez local dio elemento, ou sua parcela de contribuição na obtenção da matriz de rigidez da estrutura.

- PontosDeGauss: Atributo definido como um ponteiro para um vetor de objetos da classe PontosDeGauss (ou classes dela derivadas por herança), que armazenará os endereços dos objetos que representam os pontos de Gauss internos ao elemento.

- Fint: Atributo definido como um objeto da classe matriz, destinado a representar o vetor de forças internas do elemento, ou sua parcela de contribuição na obtenção do vetor de forças internas da estrutura.

A classe Elemento terá, inicialmente, os seguintes métodos:

- Um método denominado CalculaRigidez, destinado a calcular a sua matriz de rigidez local, e sua consequente contribuição à matriz de rigidez da estrutura. Sua matriz de rigidez é inicialmente calculada a partir das contribuições obtidas em cada um dos pontos de Gauss internos ao elemento. Este método receberá como parâmetros um ponteiro para o objeto que representa a matriz de rigidez da estrutura, um ponteiro para um vetor de objetos que representam os

PUC-Rio - Certificação Digital Nº 0221221/CA

pontos nodais da estrutura e um ponteiro para o objeto que representa o material do elemento.

- Um método denominado Incidência, destinado a calcular o vetor de incidências do elemento. Este método receberá como parâmetros um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura, e montará o vetor NoLocal descrito anteriormente.

- Um método denominado CalculaTensoes, destinado a calcular as tensões nos diversos ponto de Gauss internos ao elemento, mediante chamadas aos métodos dos objetos que representam estes pontos de Gauss. Este método receberá como parâmetros um ponteiro para o vetor de objetos que representam os pontos nodais da estrutura e um ponteiro para um vetor de objetos que representam os materiais da estrutura.

- Um método denominado LeDados, responsável pela leitura dos dados do elemento e pelo redimensionamento dos seus atributos que são definidos como vetor de objetos, ou como objetos da classe matriz. Este método recebe como parâmetros objetos (streams) que representam os arquivos de entrada e de saída.

- Um método denominado GeraResultados, responsável pela gravação, em um arquivo externo (representado por um stream de arquivo), dos resultados do processamento. Este método recebe como parâmetro um objeto que representa o arquivo de resultados.

- Um método denominado CalculaForcasInternas, destinado a calcular o seu vetor de forças internas local, e sua consequente contribuição ao vetor de forças internas global da estrutura. Seu vetor de forças internas é inicialmente calculado a partir das contribuições obtidas em cada um dos pontos de Gauss internos ao elemento. Este método receberá como parâmetros um ponteiro para o objeto que representa a matriz de rigidez da estrutura, um ponteiro para um vetor de objetos que representam os pontos nodais da estrutura e um ponteiro para o objeto que representa o material do elemento.

A classe Elemento, assim como a classe PontoDeGauss, não será usada para instanciar qualquer objeto. Ao invés disso, ela servirá como classe-base para as classes Elemento_EPT (usada na análise de problemas que envolvam um estado plano de tensões), Elemento _EPD (usada na análise de problemas que envolvam um estado plano de deformações), e Elemento_AXISSIMETRICO

PUC-Rio - Certificação Digital Nº 0221221/CA

(usada na análise de problemas que envolvam um estado axissimétrico), que serão derivadas, por herança, da classe Elemento.

Conseqüentemente esta classe será definida como uma classe composta apenas por métodos virtuais – definidos como métodos declarados na classe-base mas implementados apenas nas classes dela derivadas por herança. Neste caso, torna-se inclusive desnecessária a definição de métodos construtores e destrutores, já que a classe não instanciará nenhum objeto.

A definição desta classe é reproduzida na figura 12:

```
class Elemento
{
 public:
   int NumeroDePontosNodais;// Número de Nós Que Definem o Elemento
   int NumeroDePontosDeGauss;// Número de Pontos de Gauss do
Elemento
   int TipoMaterial;// Define o Tipo de Material do Elemento
   int* NP;// Incidência dos Pontos Nodais
   No * NoLocal; // Armazena Informações Globais Sobre os Nós do
Elemento
   PontoDeGauss *PontosDeGauss;// Armazena os Pontos de Gauss
   virtual void CalculaRigidez(matriz &KGLOBAL, No * Nos, Material
&Mat){};   // Método Para Cálculo da Rigidez
   virtual void Incidencia(No * Nos){};// Determina a Incidência Nodal do
Elemento
   virtual void CalculaTensoes(No * Nos, Material &Mat){};
   // Calcula as Tensões no Elemento
   virtual void LeDados(ifstream &arquivo_entrada, ofstream
&arquivo_saida){};
   virtual void GeraResultados(ofstream &arquivo_saida){};
 protected:
   matriz IND; // Vetor de Incidências
   matriz K; // Matriz de Rigidez
   matriz Fint; // Vetor de Forças Internas
};
```

Figura 12 – Definição da classe Elemento.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.3.8
Definição da Classe Elemento_EPT

A classe Elemento _EPT permite instanciar objetos que representarão os elementos submetidos a um estado plano de tensão, sendo derivada por herança da classe Elemento.

Esta classe herda todos os atributos e métodos já definidos para a classe Elemento, havendo a necessidade de se definir apenas seu método construtor e. como serão instanciados objetos desta classe, esta será responsável por implementar os métodos que foram apenas declarados na sua classe-base.

A classe Elemento _EPT é definida como uma classe derivada, por herança, da classe-base Elemento, e sua definição é reproduzida na figura 13:

```
class Elemento_EPT : public Elemento
{
 public:
   Elemento_EPT();
   virtual void Incidencia(No * Nos);
   virtual void CalculaRigidez(matriz &KGLOBAL, No * Nos, Material
&Mat);
   virtual void LeDados(ifstream &arquivo_entrada, ofstream
&arquivo_saida);
   virtual void CalculaTensoes(No * Nos, Material &Mat);
   virtual void CalculaForcasInternas(matriz &FintGLOBAL, No * Nos,
Material &Mat);
   virtual void GeraResultados();
};
```

Figura 13 – Definição da classe Elemento_EPT.

4.3.9
Definição da Classe Elemento_EPD

A classe Elemento_EPD permite instanciar objetos que representarão os elementos submetidos a um estado plano de deformação, sendo derivada por herança da classe Elemento_EPT.

PUC-Rio - Certificação Digital Nº 0221221/CA

Esta classe herda todos os atributos e métodos já definidos para a classe Elemento_EPT, havendo a necessidade apenas de se definir o seu método construtor

A definição desta classe é reproduzida na figura 14:

```
class Elemento_EPD : public Elemento_EPT
{
 public:
   Elemento_EPD();
};
```

Figura 14 – Definição da classe Elemento_EPD.

4.3.10
Definição da Classe Elemento_AXISSIMETRICO

PUC-Rio - Certificação Digital Nº 0221221/CA

A classe Elemento_AXISSIMETRICO permite instanciar objetos que representarão elementos submetidos a um estado axissimétrico, sendo derivada por herança da classe Elemento_EPT.

Esta classe herda todos os atributos e métodos já definidos para a classe Elemento_EPT, havendo a necessidade apenas de se definir o seu método construtor

A definição desta classe é reproduzida na figura 15:

```
class Elemento_AXISSIMETRICO : public Elemento_EPT
{
 public:
   Elemento_ AXISSIMETRICO ();
};
```

Figura 15 – Definição da classe Elemento_AXISSIMETRICO.

Para as classes implementadas, pode-se estabelecer o diagrama de hierarquia mostrado na figura 16:

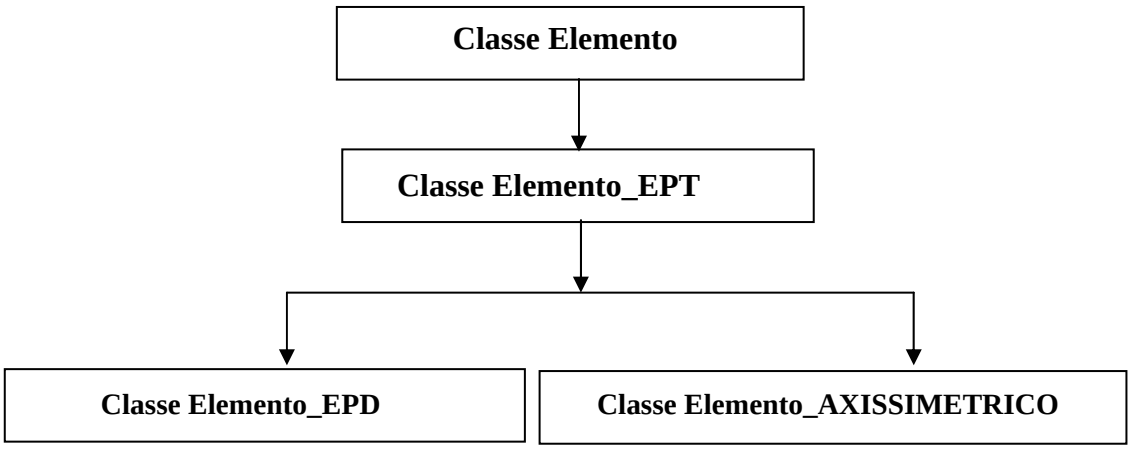

Figura 16 – Diagrama de hierarquia de classes.

4.3.11
Definição da Classe Estrutura

Esta classe é usada para representar a estrutura que será analisada. Será composta por um conjunto de elementos (representados por objetos de classes derivadas da classe elemento) e de pontos nodais (representados por objetos da classe No)

A classe Estrutura terá, inicialmente, os seguintes atributos:

- NumeroDeElementos: Atributo definido como uma variável inteira, que armazena o número de elementos em que a estrutura é subdividida.

- NumeroDePontosNodais: Atributo definido como uma variável inteira, que armazena o número de pontos nodais da estrutura, pontos estes que definem os locais em que as cargas serão diretamente aplicadas e os contornos dos elementos.

- NumeroDeNosComRestricao: Atributo definido como uma variável inteira, que armazena o número de pontos nodais com restrição de deslocamentos.

- NumeroDeTiposDeMaterial: Atributo definido como uma variável inteira, que armazena o número de tipos de material da estrutura.

- Nome: Atributo definido como um vetor de oitenta caracteres, destinado a armazenar o título do problema que será analisado.

PUC-Rio - Certificação Digital Nº 0221221/CA

- TipoDeProblema: Atributo definido como um vetor de oitenta caracteres, destinado a armazenar o identificador que define o tipo do problema que será analisado.

- Nos: Atributo definido como um ponteiro para um vetor de objetos da classe No, que armazenará os objetos que representam os pontos nodais da estrutura.

- d: Atributo definido como um objeto da classe matriz, que armazenará os deslocamentos nodais como resultado da solução do sistema que relaciona deslocamentos e cargas aplicadas à estrutura.

- K: Atributo definido como um objeto da classe matriz, que armazenará a matriz de rigidez global da estrutura.

- F: Atributo definido como um objeto da classe matriz, que armazenará o vetor de cargas nodais externas diretamente aplicadas.

- Fint: Atributo definido como um objeto da classe matriz, que armazenará o vetor de forças internas da estrutura.

- Elementos: Atributo definido como um ponteiro para um vetor de endereços de objetos da classe elemento, ou de classes dela derivadas por herança.

- Materiais: Atributo definido como um ponteiro para um vetor de objetos da classe Material, que armazena objetos que representarão as definições dos diversos tipos de materiais da estrutura.

- arquivo_entrada: Atributo definido como um objeto da classe ifstream, que representará o arquivo de dados do problema.

- arquivo_saida: Atributo definido como um objeto da classe ofstream, que representará o arquivo de resultados do problema.

A classe Estrutura terá, inicialmente, os seguintes métodos:

- Um método construtor, destinado a atribuir valores iniciais aos atributos da classe.

- Um método destrutor, destinado a liberar a memória alocada pelo método construtor da classe.

- Um método denominado LeDados, destinado a gerenciar a obtenção de dados do problema a partir de um arquivo de entrada. Este método será responsável por chamar os métodos destinados a leitura de dados referentes aos Nos, Materiais, elementos e restrições da estrutura, descritos a seguir.

PUC-Rio - Certificação Digital Nº 0221221/CA

- Um método denominado LeNos, destinado a leitura dos dados referentes aos pontos nodais da estrutura.

- Um método denominado LeMateriais, destinado a leitura dos dados referentes aos materiais da estrutura.

- Um método denominado LeRestricoes, destinado a leitura dos dados referentes às restrições de deslocamento aplicadas à estrutura.

- Um método denominado LeElementos, destinado a leitura dos dados referentes aos elementos da estrutura.

- Um método denominado CalculaRigidez, destinado a calcular a matriz de rigidez da estrutura a partir das contribuições dos diversos elementos.

- Um método denominado CalculaDeslocamentos, destinado a obter a solução do sistema de equações que relaciona deslocamentos e cargas aplicadas à estrutura, e conseqüente atualização dos deslocamentos nodais.

- Um método denominado Calcula, destinado a gerenciar o processo global de solução do problema, sendo responsável por redimensionar a matriz de rigidez da estrutura, definição do vetor de cargas externas aplicadas, aplicação das restrições aos deslocamentos da estrutura, cálculo dos deslocamentos e obtenção das tensões nos pontos de Gauss interiores aos elementos da estrutura.

- Um método denominado CalculaTensoes, destinado à obtenção das tensões nos pontos de Gauss interiores aos elementos da estrutura, executando o método correspondente para cada um dos objetos que representam os elementos da estrutura.

- Um método denominado MontaVetorDeCargas, destinado a definir cada um dos elementos da matriz F, que representa o vetor de forças externas diretamente aplicadas à estrutura.

- Um método denominado GeraResultados, destinado a gravar no arquivo de saída os resultados do problema analisado.

A definição desta classe é reproduzida na figura 17:

PUC-Rio - Certificação Digital Nº 0221221/CA

```
class Estrutura
{
 public:
   int NumeroDeElementos;// Número de Elementos da Estrutura
   int NumeroDePontosNodais;// Número de Nós da Estrutura
   int NumeroDeNosComRestricao;
        // Número de Nós com Restrição da Estrutura
   int NumeroDeTiposDeMaterial;
        // Número de Tipos de Material da Estrutura
   char Nome[81]; // Título do Problema
   char TipoDeProblema[81]; // Tipo de Problema
No* Nos;// Vetor de Nos da estrutura
   matriz d; // Vetor de Deslocamentos Nodais;
   matriz K; // Matriz de Rigidez
   matriz F; // Veror de Cargas Nodais Externas Aplicadas
   matriz Fint; // Veror de Forças Interna
   Elemento* Elementos; // Vetor de Elementos
   Material * Materiais; // Vetor de Materiais
   ifstream arquivo_entrada;
   ofstream arquivo_saida;
   Estrutura(); // Construtor da Classe
   ~Estrutura(); // Destrutor da Classe
   void LeDados(char * Dados, char * Resultados, char * Visualizacao);
   void LeNos();// Lê os dados dos Pontos Nodais da Estrutura
   void LeMateriais();// Lê os dados dos Materiais da Estrutura
   void LeRestricoes();// Lê os dados das Restrições da Estrutura
   void LeElementos();// Lê os dados dos Elementos da Estrutura
   void CalculaRigidez(); // Calcula a Matriz de Rigidez da estrutura
   void CalculaDeslocamentos();// Calcula os Deslocamentos da Estrutura
   void Calcula(); // Resolve a Estrutura
   void CalculaTensoes();// calcula as Tensões na Estrutura
   void AplicaRestricoes();// Aplica as Restrições à estrutura
   void MontaVetorDeCargas(double fator = 1.0);// Monta Vetor de
Cargas Externas
   void GeraResultados();
};
```

Figura 17 – Definição da classe Estrutura.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4
Incorporação da plasticidade (não-linearidades do material)

Neste tópico serão apresentados os procedimentos necessários à incorporação das não-linearidades do material, proposta inicialmente por Owen e Hinton [55] e, mais recentemente, numa abordagem mais moderna, por Crisfield [56,57], ambos considerando a não-linearidade entre tensões e deformações.

Nestas abordagens calcula-se, para cada passo de carga *dF* uma variação no campo de deslocamentos *d{U}*, considerando-se uma matriz de rigidez tangente.

A esta variação do campo de deslocamentos corresponde uma variação no campos de deformações $d\{\varepsilon\}$ e de tensões $d\{\sigma\}$ tal que

$$d\{\sigma\} = [Dep]\, d\{\varepsilon\} \tag{4.11}$$

A partir dos valores calculados para $d\{\sigma\}$ verifica-se se o novo valor do campo de tensões $\{\sigma\} = \{\sigma_{ant}\} + d\{\sigma\}$ permanece interior à superfície de escoamento. Caso esta condição não seja atendida, aplica-se o algoritmo de retorno, apresentado a seguir, para correção plástica.

4.4.1
Algoritmo de Retorno

Este algoritmo pode ser representado pela seguinte seqüência de passos:

1. A partir da variação no campo de deslocamentos, *d{U}*, calcula-se a variação total no campo de deformações, empregando-se a fórmula:

$$d\{\varepsilon\} = [B]\, d\{U\} \tag{4.12}$$

2. A partir do valor calculado para a variação total no campo de deformações, calcula-se a variação correspondente no campo de tensões, empregando-se a expressão (4.11).

3. Atualiza-se o valor do campo de tensões, empregando-se a fórmula:

$$\{\sigma\} = \{\sigma_{ant}\} + d\{\sigma\} \tag{4.13}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

4. Verifica-se se o novo valor das tensões satisfaz a função de escoamento, que para o critério de Von-Mises, em um estado triaxial é definido pela fórmula:

$$f = \sqrt{\frac{(\sigma_x - \sigma y)^2 + (\sigma_x - \sigma z)^2 + (\sigma_z - \sigma y)^2 + 6\tau_{xy}{}^2 + 6\tau_{xz}{}^2 + 6\tau_{zy}{}^2}{2}} - \sigma_0 \qquad (4.14)$$

Ou

$$f = \sigma_e - \sigma_0 \qquad (4.15)$$

5. Se f <= 0, nenhuma correção é necessária, pois ainda se verifica o regime elástico. Se f > 0, torna-se necessária uma correção no campo das tensões.

Neste caso, considerando-se as equações de Prandtl-Reuss, em conjunção com a expressão definida para f, obtém-se para a parcela plástica deste incremento de deformações:

$$d\left\{\varepsilon_p\right\} = d\begin{Bmatrix}\varepsilon_{px}\\ \varepsilon_{py}\\ \varepsilon_{pxy}\end{Bmatrix} = d\lambda\left\{\frac{\partial f}{\partial \sigma}\right\} = d\lambda\{\mathrm{a}\} = \frac{d\lambda}{2\sigma_e}\begin{Bmatrix}2\sigma_x - \sigma_y\\ 2\sigma_y - \sigma_x\\ 6\tau_{xy}\end{Bmatrix} \qquad (4.16)$$

Subtraindo-se esta parcela da deformação total, obtém-se a parcela elástica do incremento de deformações e, conseqüentemente, o incremento de tensões:

$$d\left\{\sigma\right\} = [D]\begin{Bmatrix}\varepsilon_{ex}\\ \varepsilon_{ey}\\ \varepsilon_{exy}\end{Bmatrix} = [D]\left(d\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \varepsilon_{xy}\end{Bmatrix} - d\begin{Bmatrix}\varepsilon_{px}\\ \varepsilon_{py}\\ \varepsilon_{pxy}\end{Bmatrix}\right) = [D]\left(d\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \varepsilon_{xy}\end{Bmatrix} - d\lambda\{a\}\right) \qquad (4.17)$$

Considerando-se que as tensões devem permanecer na superfície de escoamento (normalidade):

$$df = \{a\}^{\mathrm{T}}\, d\{\sigma\} = 0 \qquad (4.18)$$

Pré-multiplicando a equação (4.17) por a^T e combinando-a com a equação (4.18), obtém-se a expressão:

$$d\lambda = \frac{\{a\}^{\mathrm{T}}[\mathrm{D}]\{\varepsilon\}}{\{a\}^{\mathrm{T}}[\mathrm{D}]\{a\}} \qquad (4.19)$$

PUC-Rio - Certificação Digital Nº 0221221/CA

Substituindo esta expressão em (4.17) obtém-se:

$$d\left\{\sigma\right\} = [D]\left([I] - \frac{\{a\}\{a\}^{\mathrm{T}}[\mathrm{D}]}{\{a\}^{\mathrm{T}}[\mathrm{D}]\{a\}}\right) d\{\varepsilon\} \tag{4.20}$$

Para o caso em que se considera um encruamento isotrópico, pode-se escrever:

$$d f = \{a\}^{\mathrm{T}} d\{\sigma\} - H d\varepsilon_p = 0 \tag{4.21}$$

Pré-multiplicando a equação (4.17) por a^T e combinando-a com a equação (4.20), obtém-se a expressão:

$$d\left\{\sigma\right\} = [D]\left([I] - \frac{\{a\}\{a\}^{\mathrm{T}}[\mathrm{D}]}{\{a\}^{\mathrm{T}}[\mathrm{D}]\{a\} + H}\right) d\{\varepsilon\} \tag{4.22}$$

As matrizes anteriores podem então ser utilizadas na obtenção da matriz de rigidez tangente.

6. Faz-se:

$$\{\sigma\} = \{\sigma\}_{ant} + [D]d\{\varepsilon\} - d\lambda\{D\}\left\{\frac{\partial f}{\partial \sigma}\right\} = \{\sigma\}_{ant} + [D]d\{\varepsilon\} - d\lambda\{D\}\{\mathrm{a}\} \tag{4.23}$$

Onde:

$[D]d\{\varepsilon\}$ representa o preditor elástico.

$d\lambda\{D\}\left\{\frac{\partial f}{\partial \sigma}\right\}$ representa o corretor plástico.

Este algoritmo de retorno foi implementado para os casos de estado plano de tensão, estado plano de deformação e axissimétrico.

Neste caso, no entanto, por se tratar de um problema não-linear, será necessário empregar um método iterativo de solução, como Newton-Raphson, por exemplo.

O fluxograma da figura 18 apresenta, de forma esquemática, a solução do problema. A carga externa será aplicada em um número de incrementos denominado número de passos de carga. Conseqüentemente, para cada um dos

PUC-Rio - Certificação Digital Nº 0221221/CA

passos de carga deverá ser calculado o valor do vetor que representa o resíduo, definido como sendo a diferença entre o vetor de forças externas aplicadas e o de forças internas:

$$\{R\} = \int_V [B]^T \{\sigma\} dV - \{F\} = \{F\}_{\text{int}} - \{F\}_{\text{ext}} \tag{4.24}$$

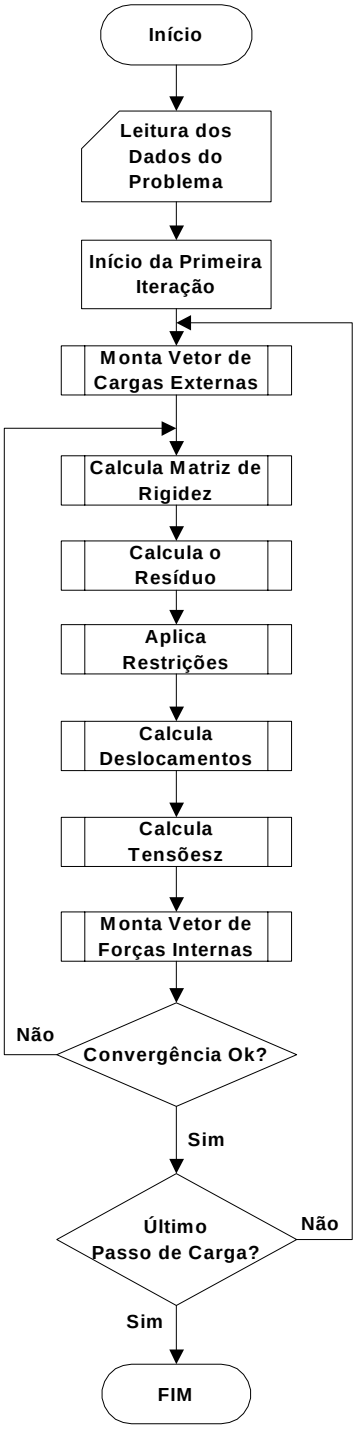

Figura 18 – Fluxograma para análise não-linear.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2
Alterações a serem Implementadas no Código

A seguir serão apresentadas as alterações a serem realizadas no código para a incorporação das não-linearidades do material.

4.4.2.1
Atributos Adicionais

A incorporação das não-linearidades do material requer que se façam as seguintes alterações:

- Inclusão, na classe Estrutura, de um atributo denominado NumeroDePassosDeCarga. Este atributo, definido como uma variável inteira, armazenará o número de passos de carga empregados na solução do problema. Conseqüentemente, a carga total a ser aplicada será dividida em incrementos de mesma intensidade.

- Inclusão, na classe Estrutura, de um atributo denominado R: Este atributo, definido como um objeto da classe matriz, armazenará o vetor que representa o resíduo em cada incremento do carregamento, sendo igual à diferença entre o vetor que representa a carga total aplicada e o de forças internas, de acordo com a equação 4.23.

- Inclusão, nas classes que representam os pontos de Gauss, de um atributo que armazene o vetor que representa as derivadas da função de escoamento em relação as componentes de tensões.

- Inclusão, nas classes que representam os pontos de Gauss, de um atributo que identifique se já ocorreu o escoamento do material naquele ponto.

- Inclusão, nas classes que representam os materiais da estrutura, de atributos que definam o valor inicial da tensão de escoamento e o módulo de elasticidade tangente no regime plástico (encruamento isotrópico).

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2.2
Métodos Adicionais

A incorporação das não-linearidades do material requer:

- Inclusão, nas classes que representam os pontos de Gauss, de métodos que determinem o valor da função de escoamento em função das tensões atuantes nos pontos de Gauss, bem como o respectivo algoritmo de retorno (predição elástica - correção plástica).

- Alteração, nas classes que representam os pontos de Gauss, dos métodos responsáveis pelo cálculo da matriz constitutiva do material de forma a considerar a parcela elastoplástica, e pelo cálculo das tensões atuantes no ponto de Gauss, chamando o respectivo algoritmo de retorno (se necessário).

- Alteração do método Lemateriais da classe Estrutura, de forma a considerar a inclusão dos novos atributos da classe Material. A classe Estrutura deverá ser acrescida dos seguintes métodos:

- Um método denominado MontaVetorDeForcasInternas, destinado a calcular as componentes do vetor de forças internas aplicadas à estrutura.

- Um método denominado Convergência, destinado a verificar se a diferença entre os módulos dos vetores de cargas externas aplicadas e de forças internas é inferior a uma determinada tolerância.

- Um método denominado CalculaResiduo, destinado calcular os componentes do vetor {R}, diferença entre os vetores que representam as cargas externas aplicadas - {F} - e as forças internas - {Fint}

Na redefinição das diversas classes, os nomes dos atributos e métodos que foram adicionados ou alterados para a inclusão das não-linearidades do material foram propositadamente colocados em negrito.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2.3
Redefinição da Classe Material

A classe Material, com as alterações mencionadas, passará a ter a definição mostrada na figura 19.

```
class Material
{
 public:
   double POISS, E;// Coeficiente de Poisson e Módulo de Elasticidade
   double fy; // Tensão de Escoamento Unidimensional
   double H; // Módulo de Elasticidade Longitudinaltangente  no regime
Plástico
};
```

Figura 19 – Redefinição da classe Material.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2.4
Redefinição da Classe PontoDeGauss

A classe PontoDeGauss, , com as alterações mencionadas, passará a ter a definição mostrada na figura 20.

```
class PontoDeGauss
{
 public:
   bool ESCOOU;
   double coordenada;
   double XI, ETA; // Coordenadas Locais do ponto de gauss.
   double WXI, WETA; // Pesos do ponto de gauss.
   matriz D; // Matriz Constitutiva do Material
   matriz K; // Matriz de Rigidez
   matriz Fint; // Vetor de Forças Internas
   matriz LN; // Matriz Resultante da Multiplicação do Operador [L]
            // Pela Matriz de Funções de Interpolação [N]
   matriz B; // Matriz que Associa Deformações a Deslocamentos
           // [B] = Inversa(J)*[L]{N]
   matriz J; // Matriz Jacobiano Responsável pela Mudança de
Coordenadas
   matriz u; // Matriz de Deslocamentos Dos Pontos Nodais
   matriz sigma;// Vetor de Tensões no Ponto de Gauss
   matriz a;// df/dsigma
```

```
   virtual void Inicializa(int NumeroDePontosNodais){};
   virtual void Calcula_D(Material &Mat){};// Calcula a Matriz Constitutiva
   virtual void Calcula_B(int i, int NumeroDePontosNodais, No * Nos){};// Calcula a Matriz [B]
   virtual void Calcula_K(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
   // Calcula a Matriz de Rigidez
   virtual void Calcula_J(int i, int NumeroDePontosNodais, No * Nos){};
   // calcula o Jacobiano
   virtual void Calcula_Tensoes(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
   // Calcula as Tensões no Ponto de Gauss
   virtual void CalculaForcasInternas(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
   // Calcula as Forças Internas no Ponto de Gauss
   virtual void GeraResultados(){};
   virtual void CoordenadasDoPontoDegauss(int IGAUSS){};
   virtual double COORDENADAXI(int i){};
   virtual double COORDENADAETA(int i){};
   virtual double SHAPE(int i, int NumeroDePontosNodais, double XI, double ETA){};
   virtual double LSHAPEXI(int i, int NumeroDePontosNodais, double XI, double ETA){};
   virtual double LSHAPEETA(int i, int NumeroDePontosNodais, double XI, double ETA){};
   virtual void Retorno(Material &Mat){};
   virtual double fsigma(double sigmax, double sigmay, double talxy, double TensaoLimite){};
};
```

Figura 20 – Redefinição da classe PontoDeGauss.

4.4.2.5

Redefinição da Classe PontoDeGauss_EPT

A classe PontoDeGauss_EPT, com as alterações mencionadas, passará a ter a definição mostrada na figura 21.

PUC-Rio - Certificação Digital Nº 0221221/CA

```
class PontoDeGauss_EPT : public PontoDeGauss
{
 public:
   PontoDeGauss_EPT();
   virtual void Inicializa(int NumeroDePontosNodais);
   void Calcula_D(Material &Mat);
   virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No *
Nos);// Calcula a Matriz [B]
   void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No * Nos,
Material &Mat);
   virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No *
Nos);
   void Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No *
Nos, Material &Mat);
   virtual void CalculaForcasInternas(int IGAUSS, int
NumeroDePontosNodais, No * Nos, Material &Mat);
   void CoordenadasDoPontoDeGauss(int IGAUSS);
   virtual void GeraResultados();
   double COORDENADAXI(int i);
   double COORDENADAETA(int i);
   virtual double SHAPE(int i, int NumeroDePontosNodais, double XI,
double ETA);
   virtual double LSHAPEXI(int i, int NumeroDePontosNodais, double
XI, double ETA);
   virtual double LSHAPEETA(int i, int NumeroDePontosNodais, double
XI, double ETA);
   virtual void Retorno(Material &Mat);
   virtual double fsigma(double sigmax, double sigmay, double talxy,
double TensaoLimite);
};
```

Figura 21 – Redefinição da classe PontoDeGauss_EPT.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2.6

Redefinição da Classe PontoDeGauss_EPD

A classe PontoDeGauss_EPD, com as alterações mencionadas, passará a ter a definição mostrada na figura 22.

```
class PontoDeGauss_EPD : public PontoDeGauss_EPT
{
 public:
   PontoDeGauss_EPD();
   virtual void Calcula_D(Material &Mat);
   virtual void Inicializa(int NumeroDePontosNodais);
   virtual void Retorno(Material &Mat);
   virtual double fsigma(double sigmax, double sigmay, double
sigmaz, double talxy, double TensaoLimite);
};
```

Figura 22 – Redefinição da classe PontoDeGauss_EPD.

4.4.2.7

Redefinição da Classe PontoDeGauss_AXISSIMETRICO

A classe PontoDeGauss_ AXISSIMETRICO, com as alterações mencionadas, passará a ter a definição mostrada na figura 23.

```
class PontoDeGauss_AXISSIMETRICO : public PontoDeGauss_EPT
{
 public:
   PontoDeGauss_AXISSIMETRICO();
   // Implementação dos Métodos Herdados da Classe-Base
   virtual void Calcula_D(Material &Mat);
   virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No *
Nos);// Calcula a Matriz [B]
   virtual void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No *
Nos, Material &Mat);
   virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No *
Nos);
   virtual void Inicializa(int NumeroDePontosNodais);
   virtual void Retorno(Material &Mat);
   virtual double fsigma(double sigmax, double sigmay, double
sigmaz, double talxy, double TensaoLimite);
};
```

Figura 23 – Redefinição da classe PontoDeGauss_AXISSIMETRICO.

PUC-Rio - Certificação Digital Nº 0221221/CA

4.4.2.8

Redefinição da Classe Estrutura

A classe Estrutura, com as alterações mencionadas, passará a ter a definição mostrada na figura 24.

```
class Estrutura
{
 public:
   int NumeroDeElementos;// Número de Elementos da Estrutura
   int NumeroDePontosNodais;// Número de Nós da Estrutura
   int NumeroDeNosComRestricao;
        // Número de Nós com Restrição da Estrutura
   int NumeroDeTiposDeMaterial;
        // Número de Tipos de Material da Estrutura
   char Nome[81]; // Título do Problema
   char TipoDeProblema[81]; // Tipo de Problema
No* Nos;// Vetor de Nos da estrutura
   matriz d; // Vetor de Deslocamentos Nodais;
   matriz K; // Matriz de Rigidez
   matriz F; // Veror de Cargas Nodais Externas Aplicadas
   matriz Fint; // Veror de Forças Interna
   Elemento* Elementos; // Vetor de Elementos
   Material * Materiais; // Vetor de Materiais
   ifstream arquivo_entrada;
   ofstream arquivo_saida;
   int NumeroDePassosDeCarga;// Número de Passos de Carga da Análise
   matriz R;// Residuo;
   Estrutura(); // Construtor da Classe
   ~Estrutura(); // Destrutor da Classe
   void LeDados(char * Dados, char * Resultados, char * Visualizacao);
   void LeDados(AnsiString Dados, AnsiString Resultados, AnsiString Visualizacao);
   void LeNos();// Lê os dados dos Pontos Nodais da Estrutura
   void LeMateriais();// Lê os dados dos Materiais da Estrutura
   void LeRestricoes();// Lê os dados das Restrições da Estrutura
   void LeElementos();// Lê os dados dos Elementos da Estrutura
   void CalculaRigidez(); // Calcula a Matriz de Rigidez da estrutura
   void CalculaDeslocamentos();// Calcula os Deslocamentos da Estrutura
   void Calcula(); // Resolve a Estrutura
   void CalculaTensoes();// calcula as Tensões na Estrutura
   void AplicaRestricoes();// Aplica as Restrições à estrutura
```

PUC-Rio - Certificação Digital Nº 0221221/CA

```
   void MontaVetorDeCargas(double fator = 1.0);// Monta Vetor de Cargas Externas
   void GeraResultados();
   void CalculaResiduo(double fator = 1.0);// Monta Vetor de Cargas Externas
   bool Convergencia(double fator = 1.0, double tolerancia = 1.0);

};
```

Figura 24 – Redefinição da classe Estrutura.

PUC-Rio - Certificação Digital Nº 0221221/CA

5
Fundamentos da Programação Paralela

5.1
Considerações Gerais

Neste capítulo serão apresentadas as técnicas de programação paralela que permitem a solução de problemas de análise estrutural em uma fração do tempo que seria necessário à sua solução através do emprego de máquinas compostas por um único processador, mediante a distribuição do esforço computacional entre diversos processadores.

Será dado maior enfoque às bibliotecas de troca de mensagens aplicadas a sistemas com memória distribuída, pois as implementações realizadas neste trabalho empregaram a versão disponível em linguagem "C/C++" de uma destas bibliotecas – a biblioteca MPI (Message Passing Interface).

5.2
Principais Limitações Associadas a Componentes de Hardware

Do ponto de vista da implementação de uma solução computacional destinada à solução de um modelo físico, como ocorre na solução de problemas de análise de estruturas pelo método dos elementos finitos, os principais componentes de hardware cujas limitações devem ser consideradas são: o processador e a memória.

O processador tem a função de executar operações matemáticas, lógicas e de transferência de valores de e para a memória em intervalos regulares de tempo. Este intervalo de tempo mínimo entre instruções sucessivas, medido em nano segundos, é definido como ciclo de *clock* ou ciclo de máquina, e varia de 1 a 5 nano segundos nos processadores mais rápidos. Outra medida comumente usada é a freqüência destes ciclos, medida em Megahertz (MHz), e que determina a velocidade do processador (*clock speed*), pois define a quantidade de instruções que podem ser realizadas ou processadas a cada segundo.

PUC-Rio - Certificação Digital Nº 0221221/CA

Embora os processadores possam iniciar a execução de uma nova instrução a cada nano segundo, o tempo de acesso à memória é de aproximadamente 100 nano segundos, o que pode ser considerado como um limitador adicional da velocidade de processamento. Além disso, é na memória que são armazenados os dados a serem processados durante a solução computacional de um modelo físico, limitando também a dimensão da solução que pode ser obtida.

A fim de se contornar estas limitações, foram desenvolvidos sistemas com múltiplos processadores, baseados em arquiteturas paralelas classificadas como memória compartilhada ou distribuída, de acordo com a distribuição da memória e da sua forma de acesso pelos diversos processadores.

Nas arquiteturas com memória compartilhada todos os processadores podem acessar diretamente toda a memória disponível no sistema, em um espaço de endereçamento único, como mostra a figura 25.

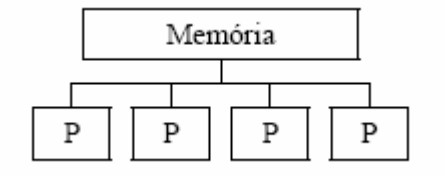

Figura 25 – Arquitetura de memória compartilhada.

Nas arquiteturas de memória distribuída cada processador pode acessar diretamente somente a sua memória local. O acesso à memória associada a outros processadores é feito através de uma rede interligando estes processadores, como ilustra a figura 26.

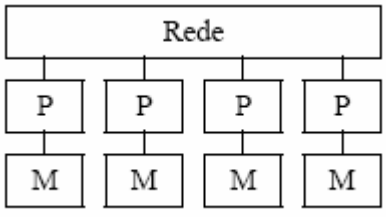

Figura 26 – Arquitetura de memória distribuída

PUC-Rio - Certificação Digital Nº 0221221/CA

Nestes casos, deverá ser considerado também o tempo de comunicação entre os diversos processadores, através da qual um processador enviará informações a serem armazenadas na memória local de outros processadores, ou terá acesso a valores armazenados nestes locais.

Os sistemas de memória compartilhada podem ainda ser classificados como SMP (*Symmetric Multiprocessors*) quando possuem processadores idênticos e acesso uniforme à memória ou NUMA (*Non Uniform Access Memory*) quando o acesso à memória não é uniforme. Por sua vez, os sistemas de memória distribuída são usualmente denominados MPP (*Massively Parallel Processors*) ou *clusters*.

5.3
Modelos de Programação Paralela

PUC-Rio - Certificação Digital Nº 0221221/CA

A arquitetura de memória determina a forma de programação utilizada na paralelização de programas. Dentre os vários modelos disponíveis, dois têm se destacado: O modelo de emprego de diretivas e o modelo de troca de mensagens.

No modelo de emprego de diretivas, comum em sistemas de memória compartilhada, diretivas para programação de *threads* - *thread programming* - podem ser inseridas em um código para definir regiões ou blocos definidos como estruturas de repetição - *laços* - a serem executados em paralelo.

No modelo de troca de mensagens (*message passing*), comum em sistemas de memória distribuída, cada processador pode acessar a memória pertencente a outros processadores enviando mensagens através de uma rede conectando os processadores.

Neste trabalho optou-se pelo emprego do modelo de troca de mensagens, tendo sido empregada a versão em linguagem “C/C++” da biblioteca MPI. Esta opção se deve ao fato de que os sistemas de memória distribuída formados por clusters de computadores são mais baratos e acessíveis à comunidade acadêmica e de profissionais que necessitam deste tipo de recurso, além da sua escalabilidade.

5.4
Avaliação de Desempenho

Uma das principais razões para a aplicação de técnicas de programação paralela é a redução do tempo total de processamento de uma aplicação. Embora não seja um critério suficiente na avaliação do desempenho, o tempo total de execução de um programa é certamente a informação inicial mais importante sobre o seu comportamento em um determinado sistema.

Uma das formas de determinar este tempo consiste na inserção de instruções especiais no início e final do programa, bem como no inicio e no final de cada rotina cujo tempo de execução se deseja determinar. Estes chamados *code timers* tem, no entanto, a desvantagem de acrescentar o seu próprio tempo de execução ao tempo total do programa.

Além do tempo de execução, o desempenho do programa é definido pelo uso eficiente dos recursos de hardware, em especial das suas unidades funcionais e da hierarquia de memória.

Usualmente utilizado como medida de desempenho de programas paralelos, o *speedup* é definido como a relação entre o tempo total de execução da versão serial ou em um único processador e o tempo total de execução da aplicação em paralelo, sendo definido pela fórmula:

$$S_p = \frac{T_1}{T_p} \tag{5.1}$$

Outra métrica bastante comum de avaliação de desempenho paralelo, denominada eficiência, é a relação entre o *speedup* e o número de processadores.

$$E_p = \frac{Sp}{p} \tag{5.2}$$

A conhecida Lei de Amdahl, quando aplicada a programas paralelos, define que o *speedup* de um programa é limitado pela fração paralelizável do código. Esta Lei pode ser representada pela equação a seguir, onde *f* representa a fração paralelizável de um programa:

$$S_p = \frac{p}{f + (1 - f)p} \tag{5.3}$$

PUC-Rio - Certificação Digital Nº 0221221/CA

O gráfico na Figura 27 mostra o *speedup* potencial para diferentes valores de *f*, a fração paralelizável do programa, e diferentes números de processadores. Este gráfico ilustra, por exemplo, que o tempo de execução de um programa com 80% de seu código paralelizável será reduzido em 4 vezes com o uso de 16 processadores (speedup igual a 4), uma eficiência paralela de 25%, enquanto que com 8 processadores a mesma aplicação tem um *speedup* 3.33 ou 41.6% de eficiência paralela.

Pode-se portanto deduzir que o simples acréscimo do número de processadores não resulta necessariamente em redução significativa do tempo de processamento de uma aplicação.

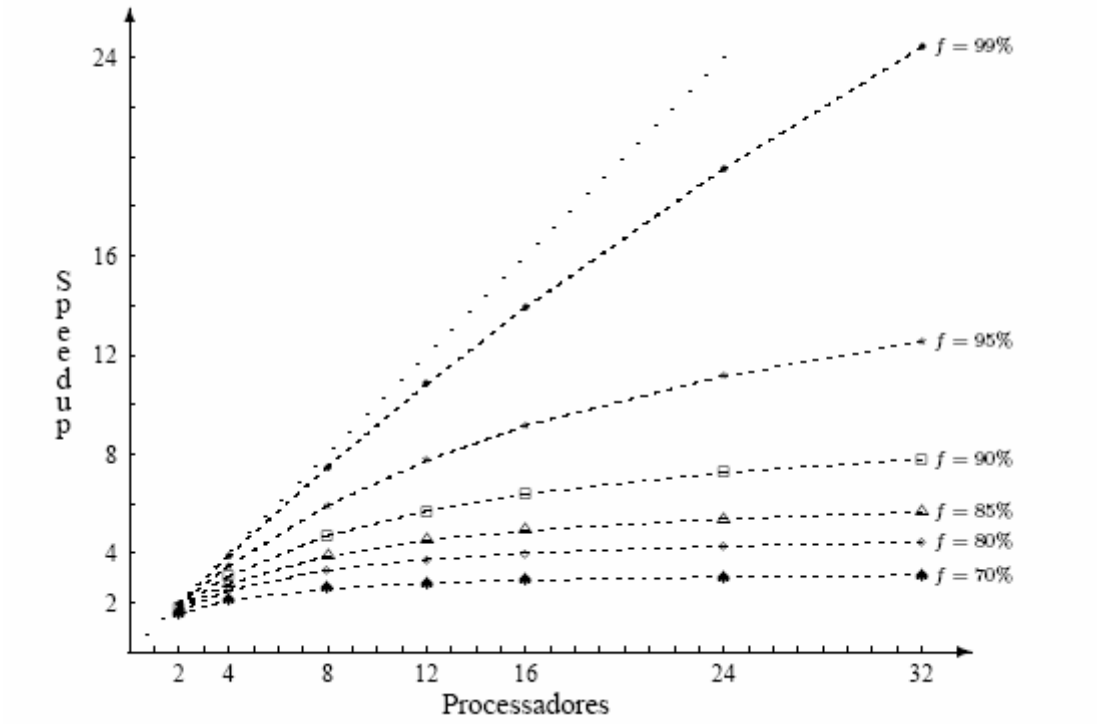

Figura 27 - Lei de Amdahl - Speedup Potencial

Evidentemente a situação ideal - correspondente a 100% de eficiência - só ocorreria se o código fosse completamente paralizável e se não houvesse consumo de tempo para a comunicação entre os diversos processadores, principalmente em sistemas de memória distribuída em que é empregado o modelo de troca de mensagens.

PUC-Rio - Certificação Digital Nº 0221221/CA

5.5
Classificação de Flynn

Michael Flynn classificou as máquinas ou arquiteturas computacionais em 4 categorias, de acordo com o fluxo ou seqüência de dados e instruções a serem processadas simultâneamente:

- SISD (Single Instruction Single Data): Uma única sequência de Instruções e uma única sequência de Dados. Um exemplo de arquitetura SISD é a máquina clássica de Von Neumann (composta por um único processador, memória e periféricos de entrada e saída de dados) em que as instruções são executadas sequencialmente, de acordo com a ordem estabelecida por uma unidade de controle. Este tipo de arquitetura não permite qualquer tipo de paralelismo.
- SIMD (Single Instruction Multiple Data): Corresponde ao paralelismo de dados, em que uma única sequência de Instruções é executada paralelamente usando vários dados de forma síncrona. São representadas pelos supercomputador vetoriais e os processadores matriciais, com uma única unidade de controle e diversas unidades lógico-aritméticas.
- MISD (Multiple Instruction Single Data): Multiplas sequências de Instruções e uma única sequência de Dados, ou seja, um conumto de máquinas executando diferentes conjuntos de instruções sobtre um único dado. Não correspondem a nenhum caso real.
- MIMD (Multiple Instruction Multiple Data): Multiplas sequências de Instruções e múltiplas sequências de Dados. São representadas pelos máquinas com múltiplos processadores com memória compartilhada ou distribuída e pelos clusters de computadores.

Neste trabalho emprega-se a arquitetura MIMD, mediante o emprego de clusters de computadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

5.6
O Modelo de Troca de Mensagens

O modelo de troca de mensagens (*message passing*) é caracterizado por um conjunto de processos (cada um associado a um processador independente) que possuem acesso direto à sua memória local, mas que podem enviar dados para a memória local associada a outros processadores (bem como receber dados destes processadores) através de um mecanismo de funções que permitem a sua intercomunicação. Pode-se considerar, portanto, que a comunicação dos processos é baseada no envio e recebimento de mensagens.

A transferência dos dados entre os processos requer operações de cooperação entre cada processo de forma que cada operação de envio deve casar com uma operação de recebimento.

O modelo computacional de troca de mensagens não inclui sintaxe de linguagem nem biblioteca (embora a partir deste modelo sejam criadas as bibliotecas de troca de mensagens, estas sim com sintaxe bem definida), e é completamente independente do hardware, apresentando as seguintes características:

- Generalidade: Pode-se construir um mecanismo de troca de mensagens para qualquer linguagem, como ferramentas de extensão das mesmas (bibliotecas). É o caso, por exemplo, da biblioteca MPI, disponível nas linguagens “C/C++” e FORTRAN.
- Adequação à ambientes distribuídos, como clusters de computadores.
- Necessidade de paralelização explícita do código, aumentando a responsabilidade do programador.
- Necessidade de se considerar os custos de comunicação entre processadores na implementação da solução paralela.

O padrão para troca de mensagens, denominado MPI (*Message Passing Interface*), foi projetado a partir das discussões realizadas em um forum de debates aberto, constituído de pesquisadores, usuários acadêmicos, programadores, usuários em geral e fornecedores de hardware, representando cerca de 40 organizações que se tornaram responsáveis pela sua aceitação e que definiram os seguintes itens:.

- Sintaxe

PUC-Rio - Certificação Digital Nº 0221221/CA

- Semântica
- Conjunto de rotinas padronizadas para *Message Passing*.

A documentação do MPI foi apresentada em Maio de 1994 (versão 1.0) e atualizada em Junho de 1995 (versão 1.1). O documento que define o padrão e denominado "MPI : A Message-Passing Standard" foi publicado pela Universidade de Tennesee e encontra-se disponível via World Wide Web na home-page do Laboratório Nacional de Argonne, no endereço:

http://www.mcs.anl.gov/mpi/

O padrão MPI apresenta as seguintes características:

- Eficiência: Foi cuidadosamente projetado para executar eficientemente em máquinas diferentes. Especifica somente o funcionamento lógico das operações, e deixa em aberto a sua implementação. Os desenvolvedores otimizam o código usando características específicas de cada máquina.
- Facilidade: Define uma interface não muito diferente de padões preexistentes. e acrescenta algumas extensões que permitem maior flexibilidade.
- Portabilidade: É compatível para sistemas de memória distribuída, clusters de computadores e uma combinação deles.
- Transparência: Permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas.
- Segurança: Provê uma interface de comunicação confiável. O usuário não precisa se preocupar com falhas na comunicação.
- Escalabilidade: O MPI suporta escalabilidade sob diversas formas, por exemplo: uma aplicação pode criar subgrupos de processos que permitem operações de comunicação coletiva para melhorar o alcance dos processos.

As bibliotecas de *Message Passing* possuem rotinas com finalidades bem específicas, como:

- Rotinas de gerência de processos: Estas rotinas têm por finalidade inicializar e finalizar processos, determinar número de processos utilizados pela aplicação e identificar cada um destes processos

PUC-Rio - Certificação Digital Nº 0221221/CA

- Rotinas de comunicação Ponto a Ponto: Permitem que a comunicação é feita entre dois processos, mediante a troca direta de mensagens entre dois processadores.
- Rotinas de comunicação de grupos: Rotinas para *Broadcast* (distribuição de uma informação entre diversos processos), sincronização de processos (barreiras) e outras

PUC-Rio - Certificação Digital Nº 0221221/CA

6
Incorporação das Técnicas de Programação Paralela

6.1
Considerações Gerais

Neste capítulo serão apresentadas os procedimentos necessários a incorporação das técnicas de programação paralela à biblioteca de classes já implementada nos capítulos anteriores. Serão apresentadas duas estratégias de paralelização, cujos resultados serão comparados no capítulo seguinte.

Será empregada a versão disponível em linguagem "C/C++" da biblioteca de troca de mensagens MPI (*Message Passing Interface*), pelas razões já apresentadas no capítulo anterior.

É importante considerar que neste modelo um mesmo programa é executado em paralelo nos diversos processadores, embora possam estar atuando sobre parcelas distintas dos dados do problema.

Neste contexto, é necessário definir alguns termos básicos relacionados à programação paralela empregando-se uma biblioteca de troca de mensagens:

- *Rank* ou Posto: Todo processo tem uma identificação única atribuída pelo sistema quando é inicializado em um ambiente distribuído. Essa identificação, denominada *rank* ou posto, é representada por um número inteiro que pode variar entre 0 e N-1, onde N é o número de processos paralelos nos quais a aplicação é executada. É utilizado para identificar um processo no envio ou recebimento de uma mensagem.

- Grupo: É o nome atribuído a um conjunto ordenado de M processos ($M \leq N$, onde N é o total de processos paralelos nos quais a aplicação é executada) e identificado por uma variável chamada "comunicador". É importante destacar que toda aplicação MPI possui um grupo *default*, ao qual estão associados todos os N processos, e que é identificado por um "comunicador" global denominado "MPI_COMM_WORLD".

- Mensagem: É o conteúdo de uma comunicação, formado de duas partes: "envelope" e "informação". Assim como ocorre em qualquer tipo de mensagem,

PUC-Rio - Certificação Digital Nº 0221221/CA

computacional ou não, um "envelope" contém os endereços (origem e destino) e a rota dos dados, sendo composto de três parâmetros: Identificação dos processos (transmissor e receptor), rótulo identificador da mensagem e comunicador do grupo ao qual se aplica. Já a "informação" corresponde ao conjunto de dados que se deseja enviar ou receber, sendo representado por três argumentos: O endereço inicial a partir do qual os dados se localizam; o número de elementos (dados) da informação que serão transmitidos na mensagem (aplicável qundo se deseja transmiir uma sequência de valores que ocupam posições contíguas de memória, como num array ou vetor) e o tipo dos dados que compõem esta informação..

- Comunicação ponto-a-ponto: Este tipo de comunicação corresponde à transferência de dados entre dois processos pertencentes a um mesmo grupo.

- Comunicação coletiva: Este tipo de comunicação corresponde a uma transferência de dados que envolve todos os processos pertencentes a um mesmo grupo.

PUC-Rio - Certificação Digital Nº 0221221/CA

6.2
Preparação do Ambiente Distribuído

Inicialmente deve-se destacar que toda aplicação distribuída desenvolvida com a biblioteca MPI deve sempre executar, antes de qualquer outra, a rotina MPI_Init, responsável por preparar a aplicação para ser executada em paralelo num ambiente distribuído.

A versão em linguagem C/C++ desta rotina apresenta a seguinte sintaxe:

int MPI_Init (int *argc, char *argv[])

Esta rotina deve, portanto, ser chamada no método construtor da classe que representa a estrutura, e este precisa ser redefinido de maneira a receber os parâmetros de linha de comando *argc* e *argv*.

Além disso, deve-se incluir na classe atributos que definam o número de processos e o processo corrente. Estes atributos, denominados *rank* e np, serão passados como parâmetros, por referência, às rotinas MPI_Comm_rank e MPI_Comm_size, responsáveis respectivamente por obter os valores que representam o processo corrente e o número de processos, e cuja implementação na linguagem C é reproduzida a seguir:

int MPI_Comm_rank (MPI_Comm comm, int *rank)

int MPI_Comm_size (MPI_Comm comm, int *size)

As chamadas a estas rotinas também deverão ser feitas no método construtor da classe.

Com estras alterações, a classe Estrutura passa a apresentar a definição mostrada na figura 28.

```
class Estrutura
{
 public:
   int np;// Número de Processadores
   int rank;//Posto ou Identificação do processo atual
   int NumeroDeElementos;// Número de Elementos da Estrutura
   int NumeroDePontosNodais;// Número de Nós da Estrutura
   int NumeroDeNosComRestricao;
         // Número de Nós com Restrição da Estrutura
   int NumeroDeTiposDeMaterial;
         // Número de Tipos de Material da Estrutura
   char Nome[81]; // Título do Problema
   char TipoDeProblema[81]; // Tipo de Problema
   No* Nos;// Vetor de Nos da estrutura
   matriz d; // Vetor de Deslocamentos Nodais;
   matriz K; // Matriz de Rigidez
   matriz F; // Veror de Cargas Nodais Externas Aplicadas
   matriz Fint; // Veror de Forças Interna
   Elemento* Elementos; // Vetor de Elementos
   Material * Materiais; // Vetor de Materiais
   ifstream arquivo_entrada;
   ofstream arquivo_saida;
   int NumeroDePassosDeCarga;// Número de Passos de Carga da Análise
   matriz R;// Residuo;
   Estrutura(int * argc, char ** argv[]); // Construtor da Classe
   ~Estrutura(); // Destrutor da Classe
   void LeDados(char * Dados, char * Resultados, char * Visualizacao);
   void LeDados(AnsiString Dados, AnsiString Resultados, AnsiString
Visualizacao);
   void LeNos();// Lê os dados dos Pontos Nodais da Estrutura
   void LeMateriais();// Lê os dados dos Materiais da Estrutura
   void LeRestricoes();// Lê os dados das Restrições da Estrutura
   void LeElementos();// Lê os dados dos Elementos da Estrutura
   void CalculaRigidez(); // Calcula a Matriz de Rigidez da estrutura
   void CalculaDeslocamentos();// Calcula os Deslocamentos da Estrutura
   void Calcula(); // Resolve a Estrutura
   void CalculaTensoes();// calcula as Tensões na Estrutura
```

PUC-Rio - Certificação Digital Nº 0221221/CA

```
   void AplicaRestricoes();// Aplica as Restrições à estrutura
   void MontaVetorDeCargas(double fator = 1.0);// Monta Vetor de
Cargas Externas
   void GeraResultados();
   void CalculaResiduo(double fator = 1.0);// Monta Vetor de Cargas
Externas
   bool Convergencia(double fator = 1.0, double tolerancia = 1.0);
};
```

Figura 28 – Redefinição da classe Estrutura.

6.3
Estratégia de Paralelização da Solução do Sistema de Equações

Esta estratégia tem por finalidade reduzir o tempo gasto na solução do sistema de equações lineares, principal etapa da solução de um problema de análise estrutural pelo método dos elementos finitos no que se refere ao consumo de tempo de processamento.

A implementação desta estratégia consiste em paralelizar o algoritmo de solução correspondente ao método dos gradientes conjugados, cuja descrição serial é apresentada na figura 29.

O método dos gradientes conjugados é um método iterativo para a solução de sistemas lineares em que a matriz dos coeficientes é positiva-definida, isto é, sistemas da forma:

$\mathbf{Ax} = \mathbf{b}$

Onde

$\mathbf{x}^T\mathbf{Ax} > \mathbf{0}$, para todo $\mathbf{x} \neq \mathbf{0}$

O método consiste em minimizar o resíduo:

$\mathbf{r}(\mathbf{x}) = \mathbf{b} - \mathbf{Ax}$

Ou, de forma equivalente, obter o valor mímimo da função:

$$\theta(x) = \frac{1}{2} x^T Ax - x^T b$$

Este método foi escolhido pela sua simplicidade no que se refere à sua codificação e complexidade computacional, além de ser perfeitamente adequado à biblioteca de classes implementada para a realização de operações matriciais,

PUC-Rio - Certificação Digital Nº 0221221/CA

permitindo que sua implementação, tanto serial como paralela, seja feita com poucas linhas de código.

Na análise de estruturas pelo método dos elementos finitos, a matriz de rigidez **K** desempenha o papel da matriz **A** dos coeficientes e o vetor de deslocamentos **d** desempenha o papel do vetor-solução **x**.

1. Adotar um valor inicial para o vetor **d** e uma tolerância Tol, para o módulo do resíduo
2. Calcular $\mathbf{p} = \mathbf{r} = \mathbf{F} - \mathbf{Kd}$
3. Calcular $\mathbf{c_1} = (\mathbf{r},\mathbf{r})$
4. Enquanto $\mathbf{c_1}^{1/2} > \text{Tol}$
    - 4.1. Calcular $\alpha_k = \mathbf{c_1} / (\mathbf{p},\mathbf{Kp})$
    - 4.2. Calcular $\mathbf{d} = \mathbf{d} + \alpha\mathbf{p}$
    - 4.3. Calcular $\mathbf{r} = \mathbf{r} - \alpha\mathbf{Kp}$
    - 4.4. Calcular $\mathbf{c_2} = (\mathbf{r},\mathbf{r})^{1/2}$
    - 4.5. Calcular $\beta = \mathbf{c_2} / \mathbf{c_1}$
    - 4.6. Calcular $\mathbf{p} = \mathbf{r} + \beta\mathbf{p}$
    - 4.7. Definir $\mathbf{c_1} = \mathbf{c_2}$

Figura 29 – Versão serial do Algoritmo dos Gradientes Conjugados.

É notável, neste algoritmo, a quantidade de operações que podem ser executadas em paralelo, principalmente a multiplicação da matriz **K** pelos vetores **p** e **d**, e alguns produtos escalares.

A figura 30 mostra, esquematicamente, a estratégia adotada para a paralelização da multiplicação da matriz **K** pelo vetor **d**, sugerida por Topping *et al* [35], em que o processamento do produto escalar das linhas da matriz **K** pelo vetor **d** é distribuído entre "n" processadores. Cada grupo de linhas nesta matriz é indicado por **Ki,** sendo "i" o processador no qual estas linhas serão processadas.

PUC-Rio - Certificação Digital Nº 0221221/CA

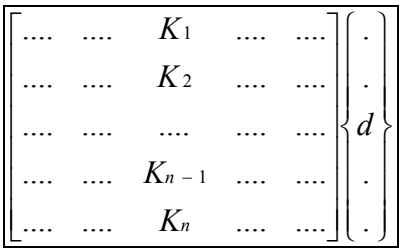

Figura 30 - Distribuição do produto da matriz K pelo vetor d

O cálculo do número de linhas a serem atribuídas a cada processador é realizado através do algoritmo apresentado na figura 31.

1. Calcular nlpp = N / np, onde N e o número de linhas da matriz.de rigidez global e np é o número de processadores.
2. Se o posto ou *rank* de um processo for inferior ao resto da divisão inteira entre o número de linhas da matriz de rigidez global e o número de processadores, o valor de nlpp deve ser incrementado em uma unidade.

Figura 31 - Cálculo do número de linhas atribuídas a cada processador

Este mesmo algoritmo é usado na distribuição dos elementos de vetores para o cálculo de produtos escalares em paralelo.

Desta maneira, a estratégia de paralelização do algoritmo dos gradientes conjugados é implementada da forma apresentada na figura 32.

1. Calcular o número de linhas por processador.
2. Distribuir a matriz de rigidez **K** e o vetor de cargas externas aplicadas **F** entre os diversos processadores.
3. Definir **d, p e z** como vetores de dimensão N.
4. Definir **r** como vetor de dimensão nlpp.
5. Adotar uma tolerância *Tol*, para o módulo do resíduo. O mesmo valor será adotado em todos os processos.
6. Definir, em cada processador, o valor inicial

PUC-Rio - Certificação Digital Nº 0221221/CA

p = **r** = **F**

7. Calcular em paralelo, em cada processador, o valor $\mathbf{c_1} = (\mathbf{r},\mathbf{r})$
8. Combinar, mediante soma, os valores de $\mathbf{c_1}$ de forma que todos os processos possuam o mesmo valor (comunicação entre os processos realizada usando rotinas de comunicação global).
9. Enquanto $\mathbf{c_1} \geq Tol$ faça:
   - 9.1. Calcular em paralelo, em cada processador, o valor da parcela correspondente de $\mathbf{z} = \mathbf{K_i p}$
   - 9.2. Calcular em paralelo, em cada processador, o valor da parcela correspondente de $\alpha = \mathbf{c_1} / (\mathbf{p},\mathbf{z})$.
   - 9.3. Combinar, mediante soma, os valores de α de forma que todos os processos possuam o mesmo valor (comunicação entre os processos usando rotinas de comunicação global).
   - 9.4. Calcular em paralelo, em cada processador, os valores correspondentes de $\mathbf{d} = \mathbf{d} + \alpha\mathbf{p}$
   - 9.5. Calcular em paralelo, em cada processador, os valores correspondentes de $\mathbf{r} = \mathbf{r} - \alpha\mathbf{z}$
   - 9.6. Calcular em paralelo, em cada processador, o valor $\mathbf{c_2} = (\mathbf{r},\mathbf{r})$
   - 9.7. Combinar, mediante soma, os valores de $\mathbf{c_2}$ de forma que todos os processos possuam o mesmo valor (comunicação entre os processos usando rotinas de comunicação global).
   - 9.8. Calcular em paralelo, em cada processador, o valor de $\beta = \mathbf{c_2} / \mathbf{c_1}$
   - 9.9. Calcular em paralelo, em cada processador, os valores correspondentes de $\mathbf{p} = \mathbf{r} + \beta\mathbf{p}$
   - 9.10. Combinar os valores de **p** de forma que todos os processos possuam o mesmo vetor (comunicação entre os processos usando rotinas de comunicação global).
   - 9.11. Definir $\mathbf{c_1} = \mathbf{c_2}$

Figura 32 – Versão paralela do Algoritmo dos Gradientes Conjugados.

PUC-Rio - Certificação Digital Nº 0221221/CA

A implementação deste algoritmo pode ser feita alterando-se apenas a codificação do operador da classe matriz usado na solução de sistemas lineares.

Na prática, empregou-se um pré-condicionamemto da matriz de rigidez, cuja matriz de escalonamento é a matriz diagonal cujos elementos são os elementos da diagonal principal da matriz de rigidez, como proposto por diversos autores, incluindo Gullerud *et al* [43].

6.4
Estratégia de Paralelização da Montagem da Matriz de Rigidez

Esta estratégia tem por finalidade reduzir o tempo gasto na obtenção da matriz de rigidez global, distribuindo esta tarefa entre os diversos processadores.

Hughes [40] propõe que o cálculo deste vetor seja feito com emprego da seguinte fórmula:

$$\{z\} = [K]\{p\} = \bigcup_{e=1}^{Nel} [Ce]^T [Ke][Ce]\{p_e\} \tag{6.1}$$

Onde

[Ce] é uma matriz de conectividade booleana (formada por elementos iguais a um ou zero) cujo número de linhas é igual ao número de graus de liberdade do elemento e cujo número de colunas é igual ao número de graus de liberdade total da estrutura.

Neste trabalho, a presença de vetores que representam as incidências nodais nos elementos permite que este cálculo seja feito sem a utilização da matriz [Ce].

Desta maneira, a equação 6.1 seria substituída por:

$$\{z\} = [K]\{p\} = \bigcup_{e=1}^{Nel} [Ke]\{p_e\} \tag{6.2}$$

A implementação desta estratégia, no entanto, requer alterações mais significativas no código do que a estratégia anterior, pois deverão ser criadas novas funções destinadas a distribuir, entre os processadores, os dados dos pontos nodais, elementos e materiais, funções estas que serão chamadas a partir do programa principal.

Além disso, deverão ser feitas pequenas alterações nos métodos CalculaRigidez e CalculaDeslocamentos da classe Estrutura.

PUC-Rio - Certificação Digital Nº 0221221/CA

A implementação desta estratégia de paralelização é apresentada na figura 33.

1. Calcular o número de elementos por processador.
2. Distribuir os elementos, os nós, os tipos de material e o vetor de cargas externas aplicadas **F** entre os diversos processadores.
3. Calcular em paralelo, em cada processador, a parcela correspondente da matriz de rigidez global.
4. Combinar, mediante soma, os valores da matriz de rigidez global, de forma que todos os processos possuam os mesmos valores (comunicação entre os processos usando rotinas de comunicação global).
5. Calcular o número de linhas por processador.
6. Distribuir a matriz de rigidez **K** e o vetor de cargas externas aplicadas **F** entre os diversos processadores.
7. Definir **d, p e z** como vetores de dimensão N.
8. Definir **r** como vetor de dimensão nlpp.
9. Adotar uma tolerância *Tol*, para o módulo do resíduo. O mesmo valor será adotado em todos os processos.
10. Definir, em cada processador, o valor inicial
    p = **r** = **F**
11. Calcular em paralelo, em cada processador, o valor $\mathbf{c_1} = (\mathbf{r},\mathbf{r})$
12. Combinar, mediante soma, os valores de $\mathbf{c_1}$ de forma que todos os processos possuam o mesmo valor (comunicação entre os processos realizada usando rotinas de comunicação global).
13. Enquanto $\mathbf{c_1} \geq Tol$ faça:
    - 13.1. Calcular em paralelo, em cada processador, o valor da parcela correspondente de $\mathbf{z} = \mathbf{K_i p}$
    - 13.2. Calcular em paralelo, em cada processador, o valor da parcela correspondente de $\alpha = \mathbf{c_1} / (\mathbf{p},\mathbf{z})$.
    - 13.3. Combinar, mediante soma, os valores de α de forma que todos os processos possuam o mesmo valor (comunicação entre os processos usando rotinas de comunicação global).
    - 13.4. Calcular em paralelo, em cada processador, os valores

PUC-Rio - Certificação Digital Nº 0221221/CA

correspondentes de $\mathbf{d} = \mathbf{d} + \alpha\mathbf{p}$

13.5. Calcular em paralelo, em cada processador, os valores correspondentes de $\mathbf{r} = \mathbf{r} - \alpha\mathbf{z}$

13.6. Calcular em paralelo, em cada processador, o valor $\mathbf{c_2} = (\mathbf{r},\mathbf{r})$

13.7. Combinar, mediante soma, os valores de $\mathbf{c_2}$ de forma que todos os processos possuam o mesmo valor (comunicação entre os processos usando rotinas de comunicação global).

13.8. Calcular em paralelo, em cada processador, o valor de $\beta = \mathbf{c_2}/\ \mathbf{c_1}$

13.9. Calcular em paralelo, em cada processador, os valores correspondentes de $\mathbf{p} = \mathbf{r} + \beta\mathbf{p}$

13.10. Combinar os valores de **p** de forma que todos os processos possuam o mesmo vetor (comunicação entre os processos usando rotinas de comunicação global).

13.11. Definir $\mathbf{c_1} = \mathbf{c_2}$

Figura 33 –Algoritmo Elemento por Elemento.

PUC-Rio - Certificação Digital Nº 0221221/CA

7
Exemplos de Aplicação

7.1
Considerações Gerais

Neste capítulo serão apresentados os resultados de alguns exemplos, destinados a verificar a validade do modelo adotado e da implementação realizada.

O primeiro exemplo, mais simples, destina-se a verificar a correção dos resultados obtidos com as análises elástica e plástica, cujos valores puderam ser verificados mediante análise estrutural convencional.

Os demais exemplos, destinados a verificar os resultados obtidos com emprego da computação paralela, foram executados num cluster comercial disponibilizado pela empresa Tsunami Technologies inc.

Optou-se pelo uso de um *cluster* comercial pelo fato de que os *clusters* não-comerciais a que o autor teve acesso, embora permitissem uma verificação inicial dos resultados, não permitiam acesso exclusivo aos processadores, produzindo resultados substancialmente diferentes ao se executar diversas vezes uma mesma aplicação.

Os modelos utilizados, embora modestos (devido ao alto custo do emprego de um *cluster* comercial), permitem que se verifique o comportamento dos resultados, principalmente no que se refere ao ganho de performance.

7.2
Verificação da Exatidão dos Resultados Obtidos com a Implementação

A fim de se verificar a correção dos resultados obtidos com as análises elástica e plástica, cujos valores possam ser verificados mediante análise estrutural convencional, adotou-se como exemplo uma viga isostática engastada e livre, reproduzida na figura 34, e com as seguintes características:

PUC-Rio - Certificação Digital Nº 0221221/CA

- Geometria: Vão de 4 m, altura de 0,5 m e largura de 10 cm.

- Material: Aço, com módulo de elasticidade longitudinal "E" igual a 210 GPa, coeficiente de Poisson igual a 0,3, tensão de escoamento igual a 250 MPa e adotando-se após o escoamento uma curva tensão-deformação linear com declividade igual a E/2). .
- Carregamento: Carga concentrada aplicada á extremidade livre da viga.

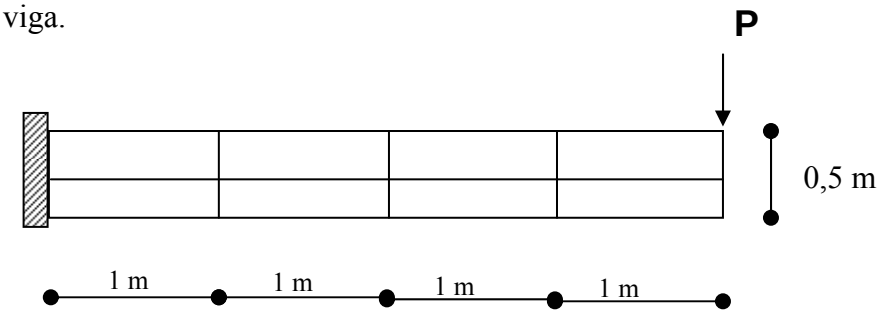

Figura 34 – Viga engastada e livre com oito elementos quadriláteros lineares.

Para este modelo, obtiveram-se os seguintes resultados:

- Considerando-se uma carga aplicada igual a 400 KN obteve-se um deslocamento máximo na extremidade livre igual a 0,039406 m, bem próximo do valor teórico de 0,039008 m (verificação da análise elástica).

A análise plástica foi realizada para o mesmo modelo, mas considerando um carregamento axial uniformemente distribuído (aplicado como carregamento nodal equivalente) com o objetivo de verificar a correção do algoritmo de retorno.

7.3
Resultados Obtidos com a Estratégia de Paralelização da Solução do Sistema de Equações

O modelo utilizado nestas análises, cujos resultados serão avaliados no próximo capítulo, é um pórtico plano hiperestático submetido a cargas concentradas em sua parte superior e na face externa da barra vertical esquerda, como mostrado na figura 35.

PUC-Rio - Certificação Digital Nº 0221221/CA

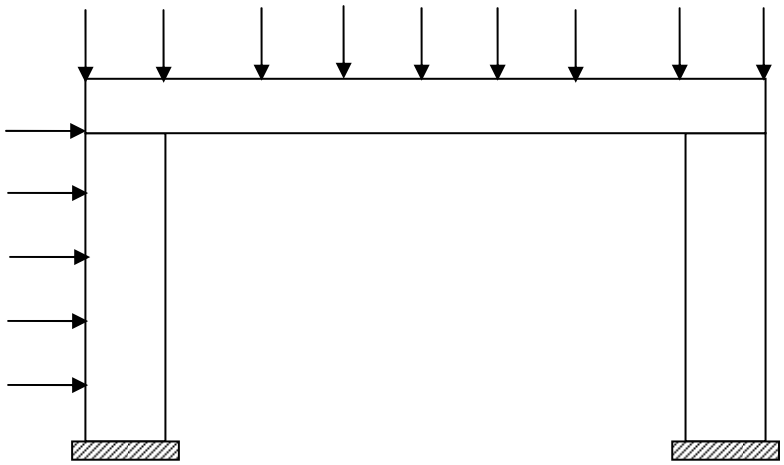

Figura 35 – Pórtico Hiperestático com elementos quadriláteros lineares.

Foram processados modelos com 100, 200 e 300 elementos, com a seguinte distribuição:

- No modelo com 100 elementos foram empregados 30 elementos nas barras verticais e 40 na barra horizontal.

- No modelo com 200 elementos foram empregados 60 elementos nas barras verticais e 80 na barra horizontal.

- No modelo com 300 elementos foram empregados 90 elementos nas barras verticais e 120 na barra horizontal.

Os gráficos apresentados a seguir mostram o ganho de performance obtido com a implementação desta estratégia de paralelização para modelos com 100, 200 e 300 elementos de estado plano de tensões, e empregando-se até 20 processadores. Na realidade, são apresentadas visualizações distintas dos mesmos resultados.

PUC-Rio - Certificação Digital Nº 0221221/CA

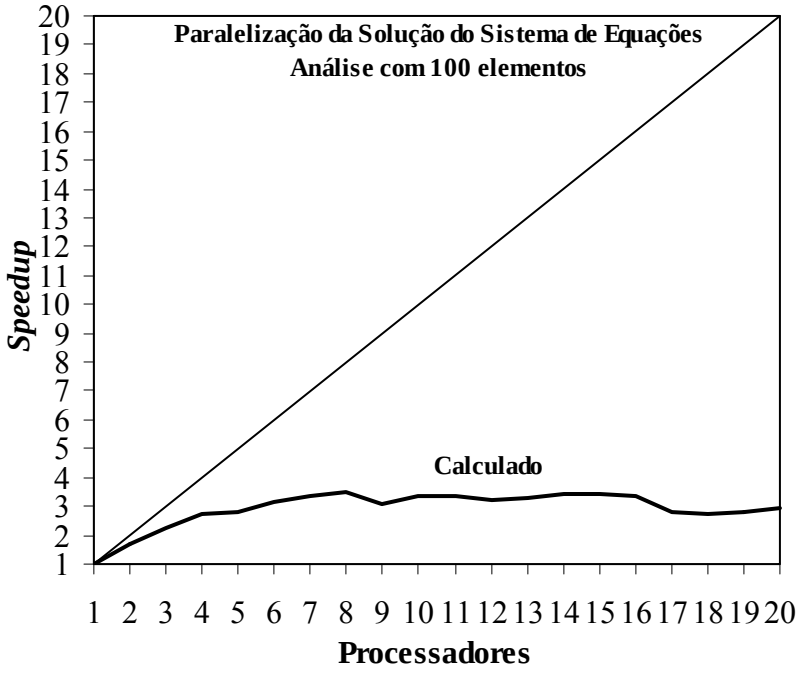

Figura 36 – Análise com 100 Elementos e 20 processadores.

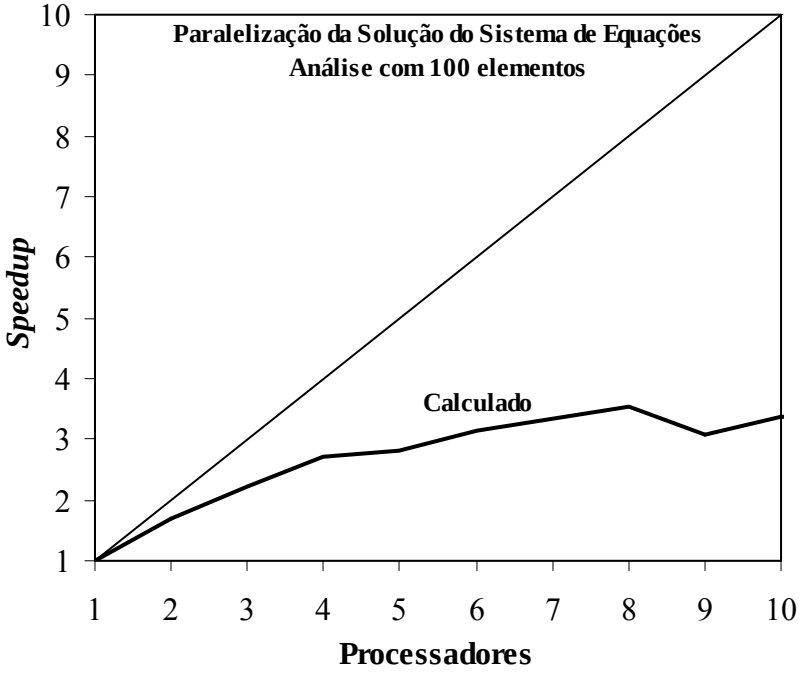

Figura 37 – Análise com 100 Elementos e 10 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

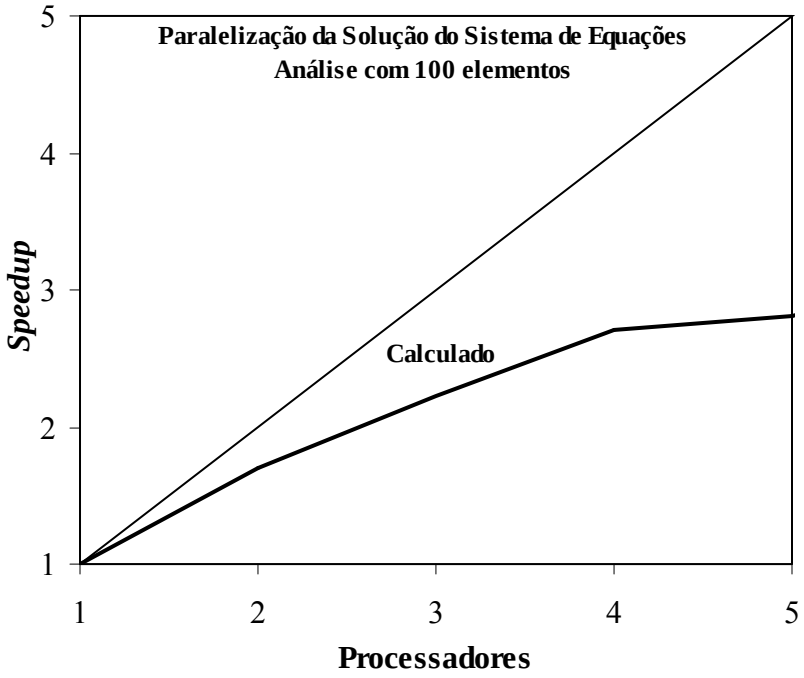

Figura 38 – Análise com 100 Elementos e 5 processadores.

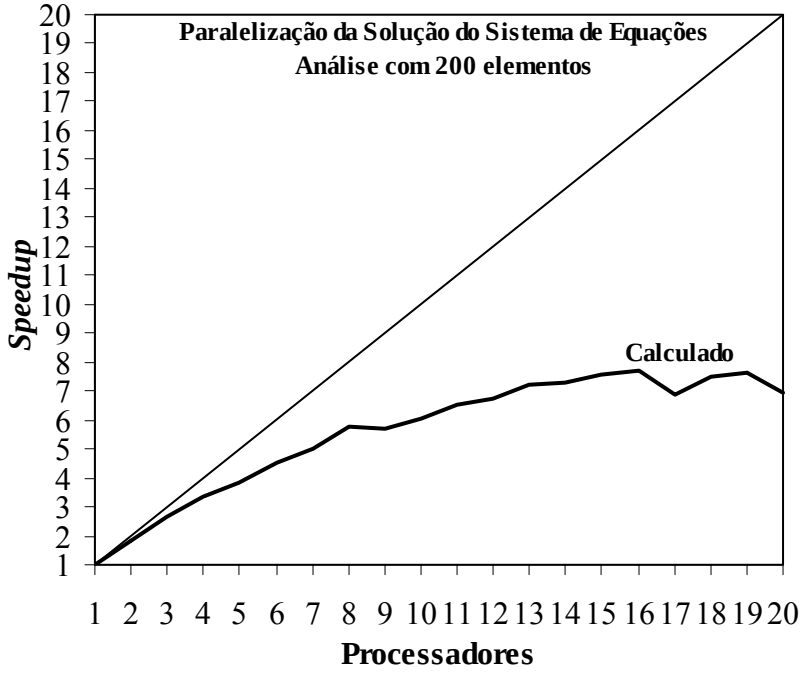

Figura 39 – Análise com 200 Elementos e 20 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

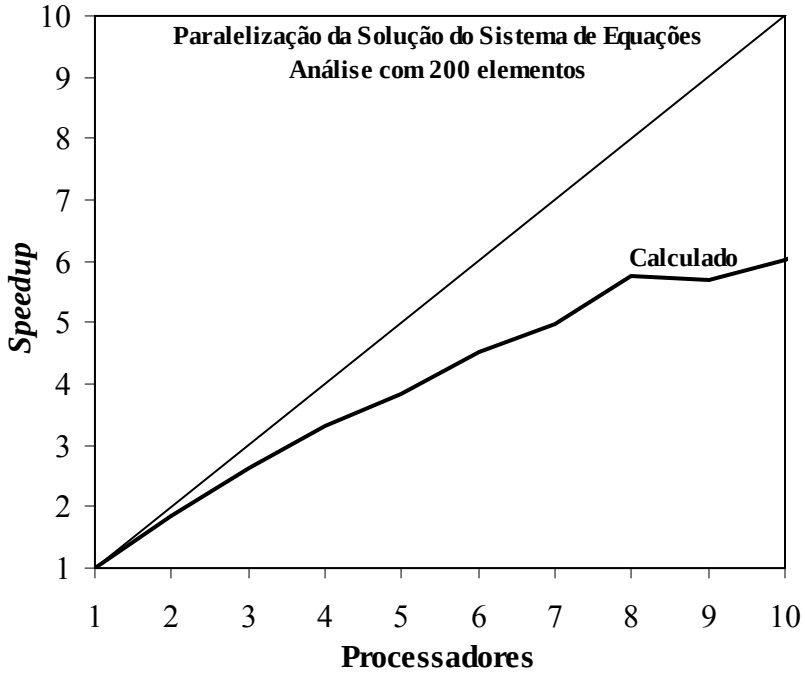

Figura 40 – Análise com 200 Elementos e 10 processadores.

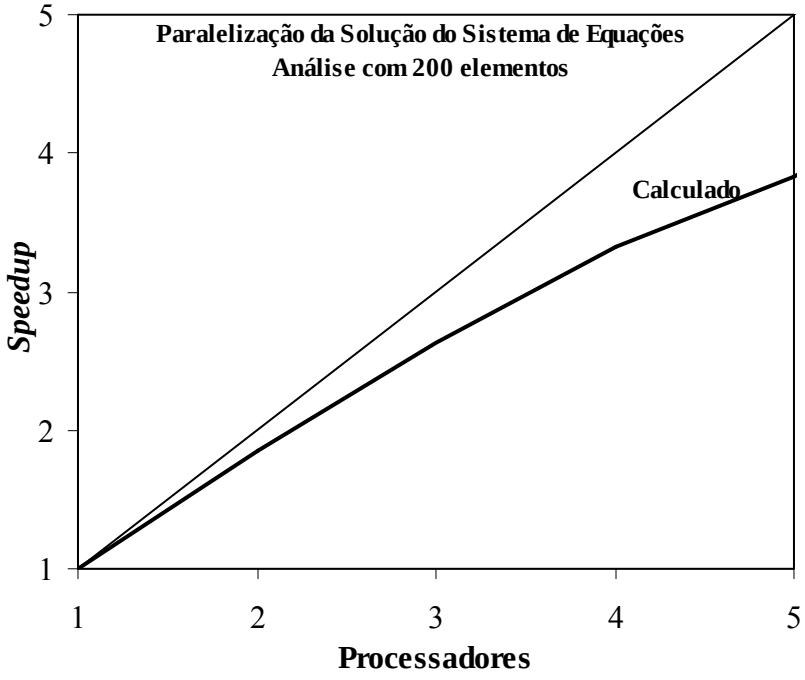

Figura 41 – Análise com 200 Elementos e 5 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

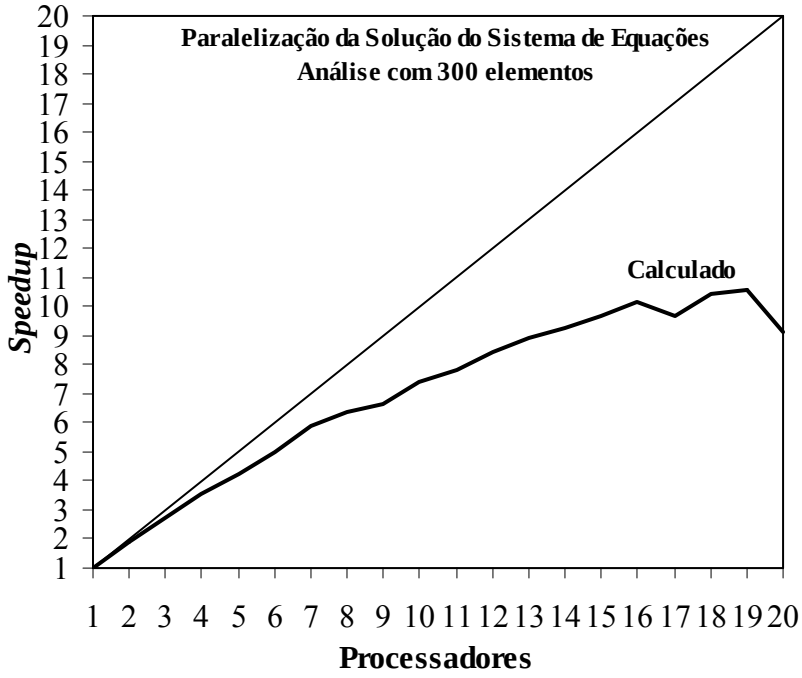

Figura 42 – Análise com 300 Elementos e 20 processadores.

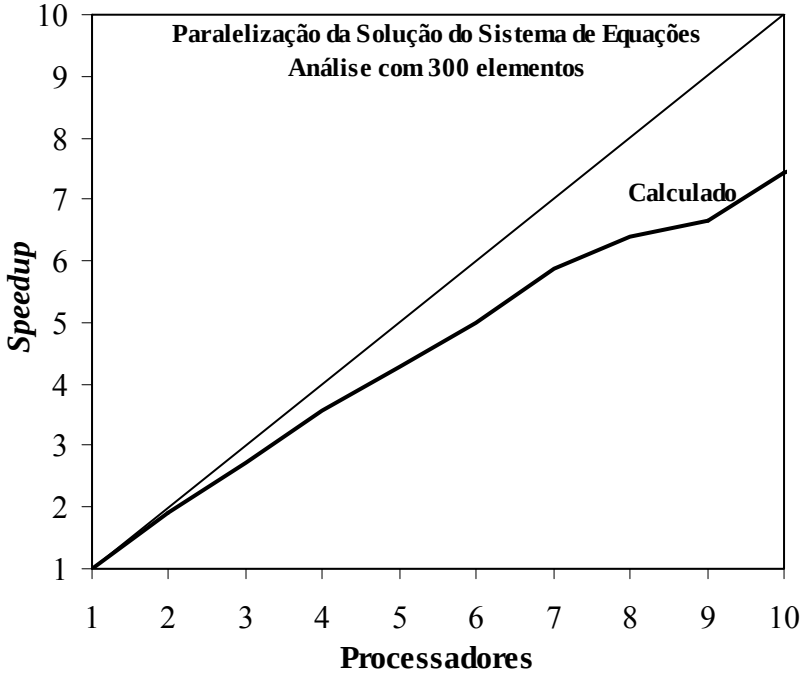

Figura 43 – Análise com 300 Elementos e 10 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

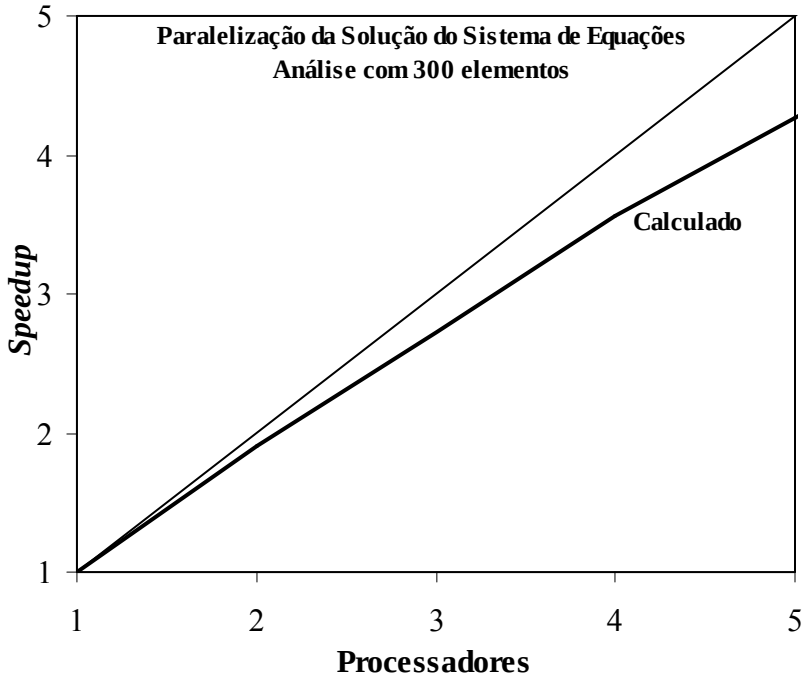

Figura 44 – Análise com 300 Elementos e 5 processadores.

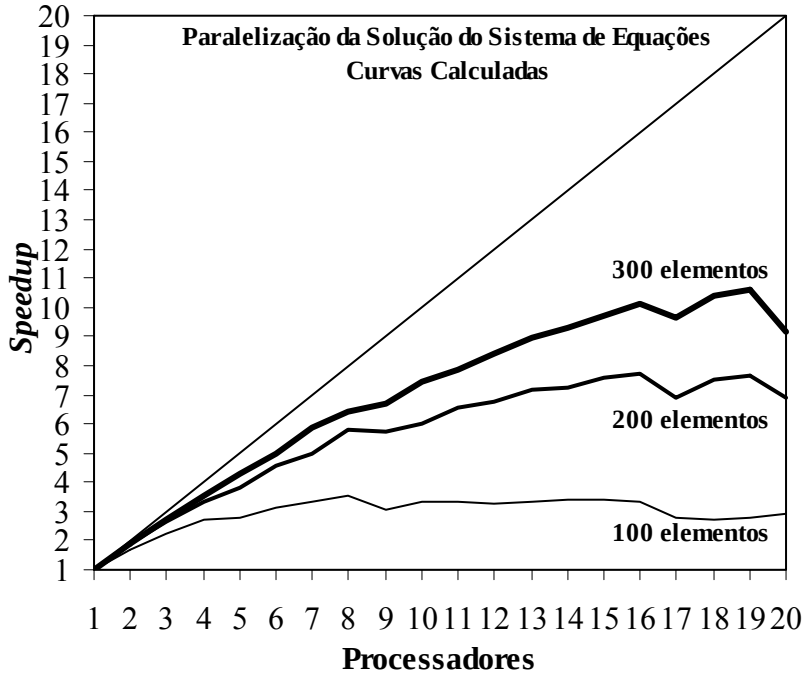

Figura 45 – Comparação dos Resultados com 20 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

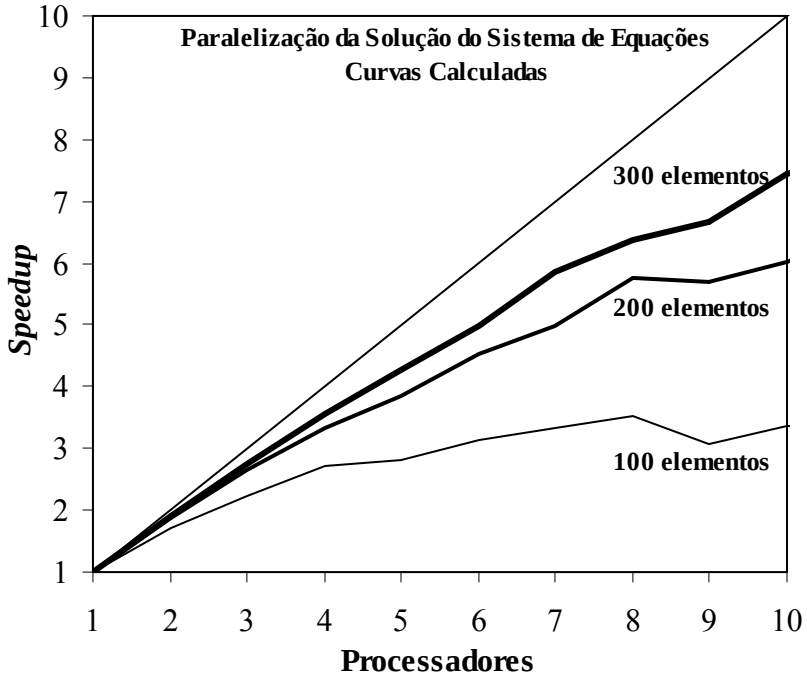

Figura 46 – Comparação dos Resultados com 10 processadores.

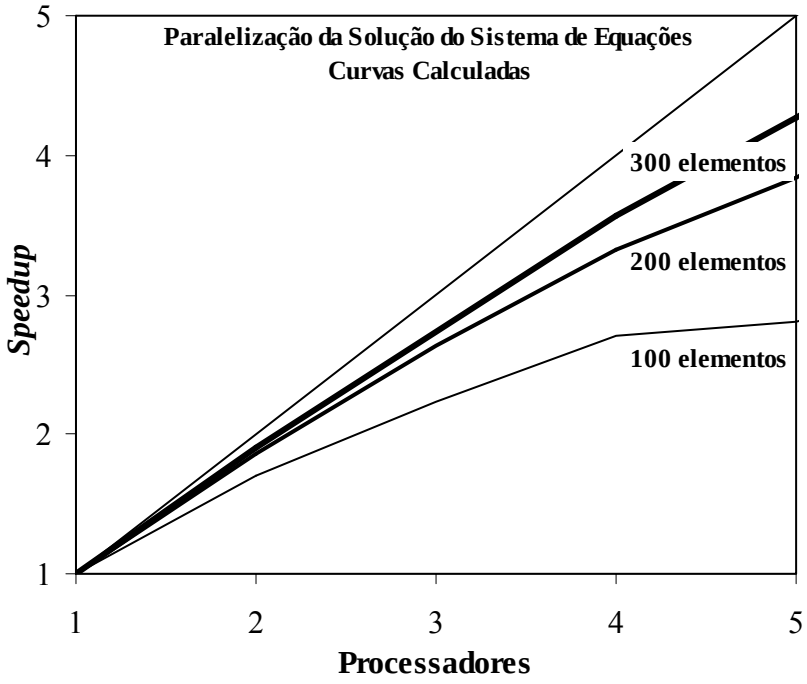

Figura 47 – Comparação dos Resultados com 5 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

7.4
Resultados Obtidos com a Estratégia de Paralelização da Montagem da Matriz de Rigidez

Os gráficos apresentados a seguir mostram o ganho de performance obtido com a implementação desta estratégia de paralelização para o mesmo exemplo processado com a estratégia de paralelização da solução do sistema de equações. (modelo com 100, 200 e 300 elementos, e empregando-se até 20 processadores). Na realidade, são apresentadas visualizações distintas dos mesmos resultados, para diferentes números de processadores. Estes resultados serão analisados no próximo capítulo.

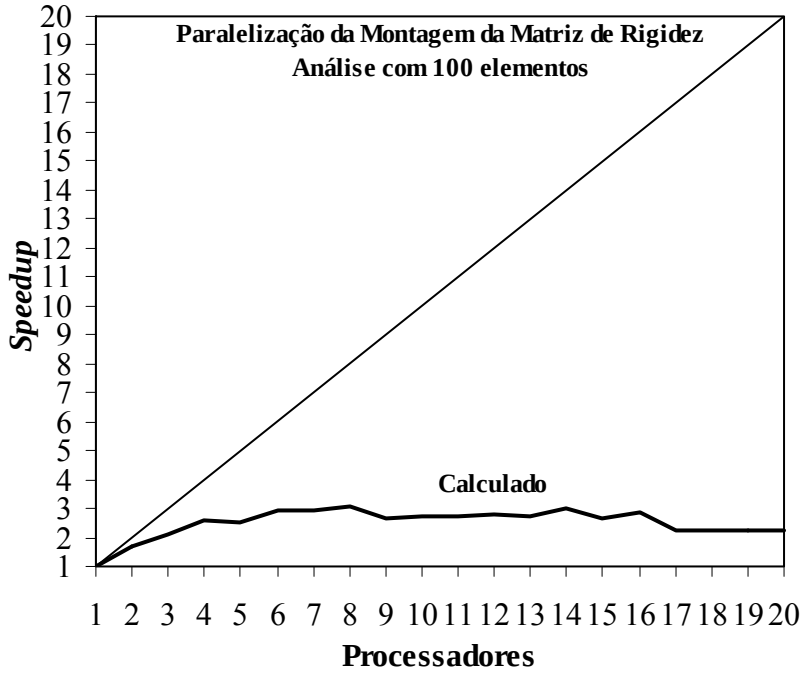

Figura 48 – Análise com 100 Elementos e 20 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

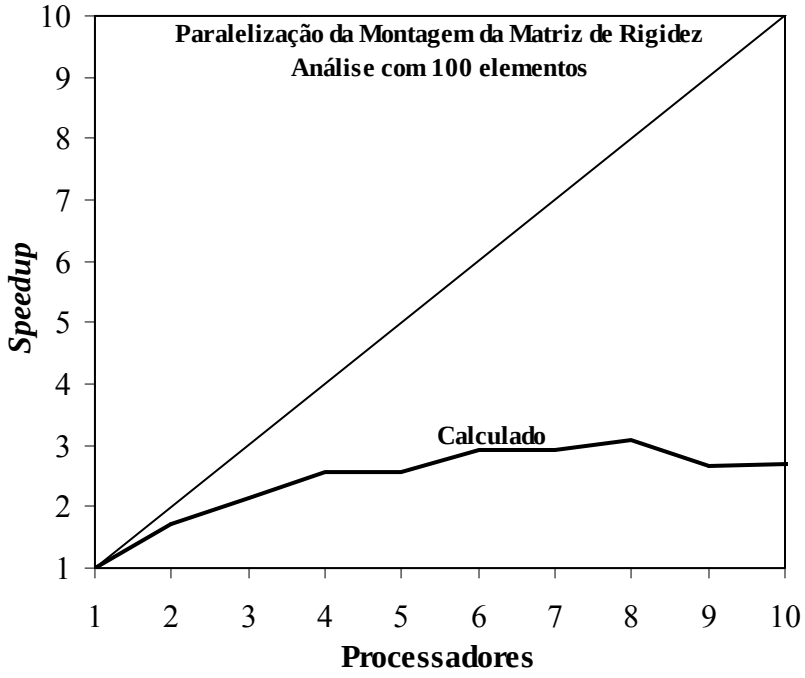

Figura 49 – Análise com 100 Elementos e 10 processadores.

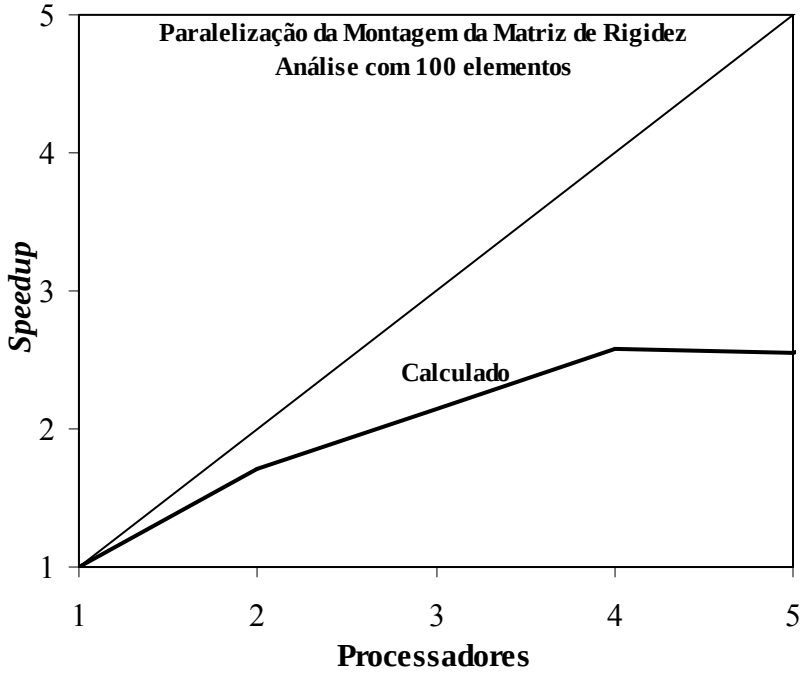

Figura 50 – Análise com 100 Elementos e 5 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

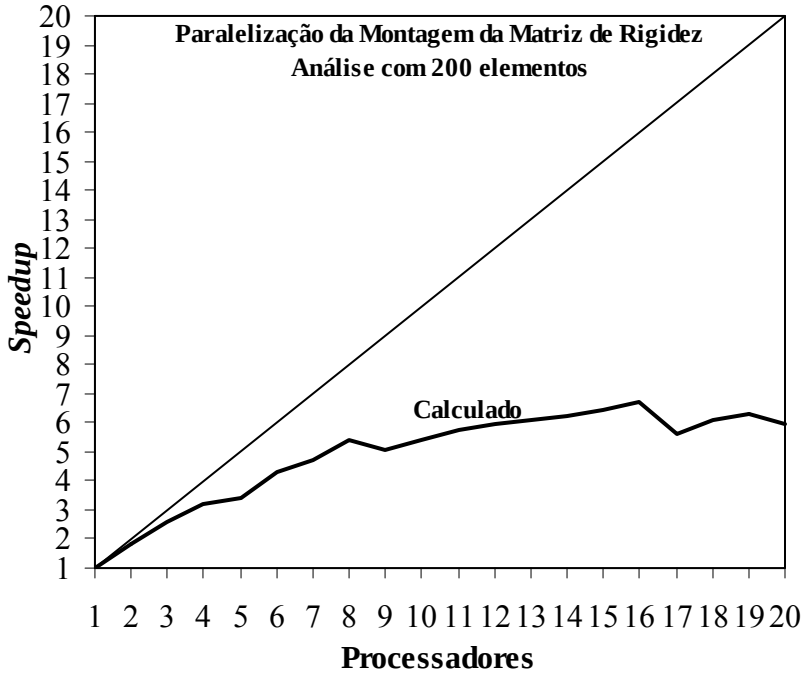

Figura 51 – Análise com 200 Elementos e 20 processadores.

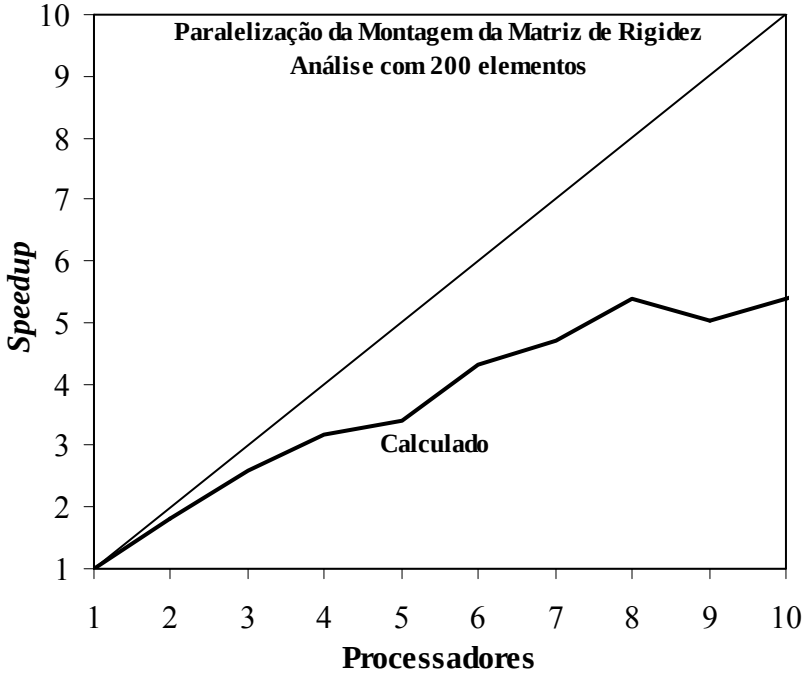

Figura 52 – Análise com 200 Elementos e 10 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

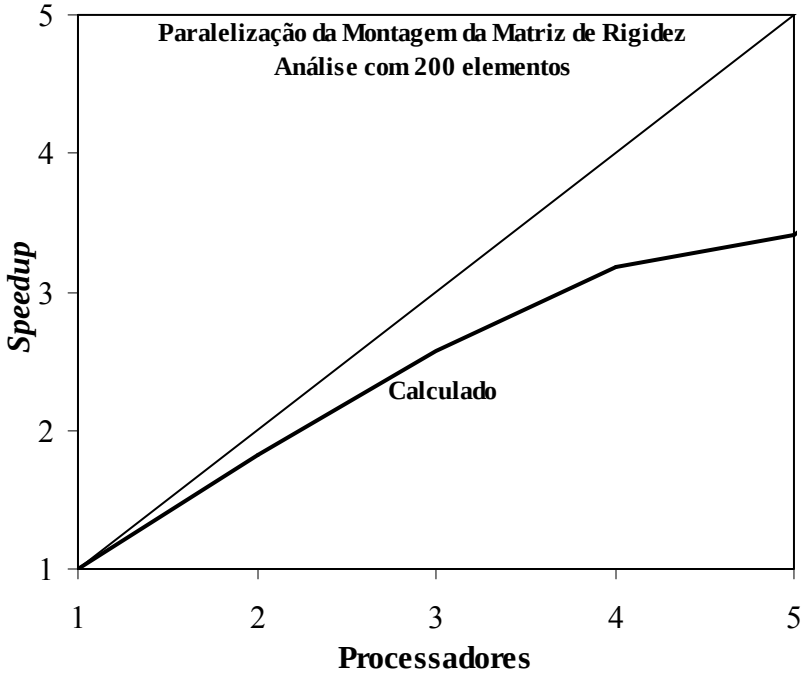

Figura 53 – Análise com 200 Elementos e 5 processadores.

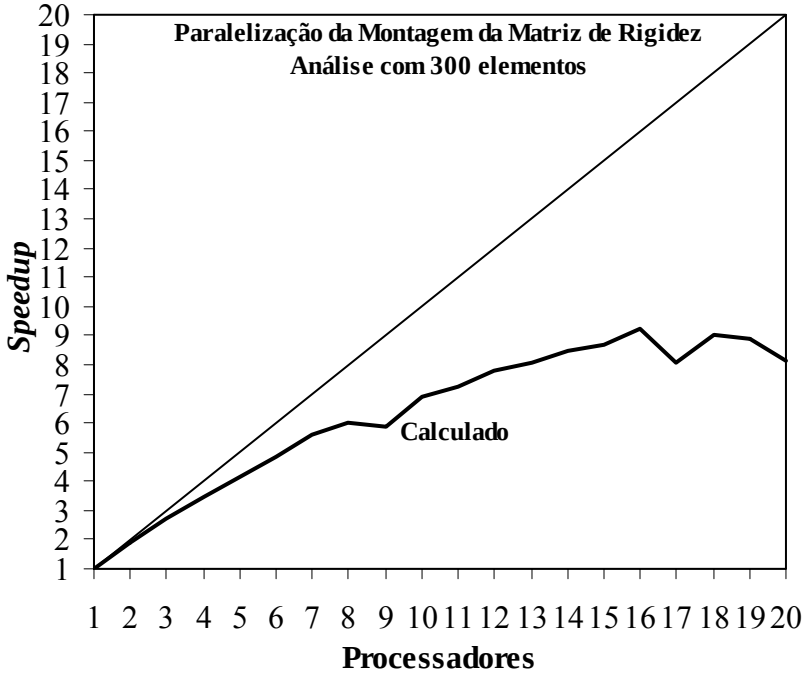

Figura 54 – Análise com 300 Elementos e 20 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

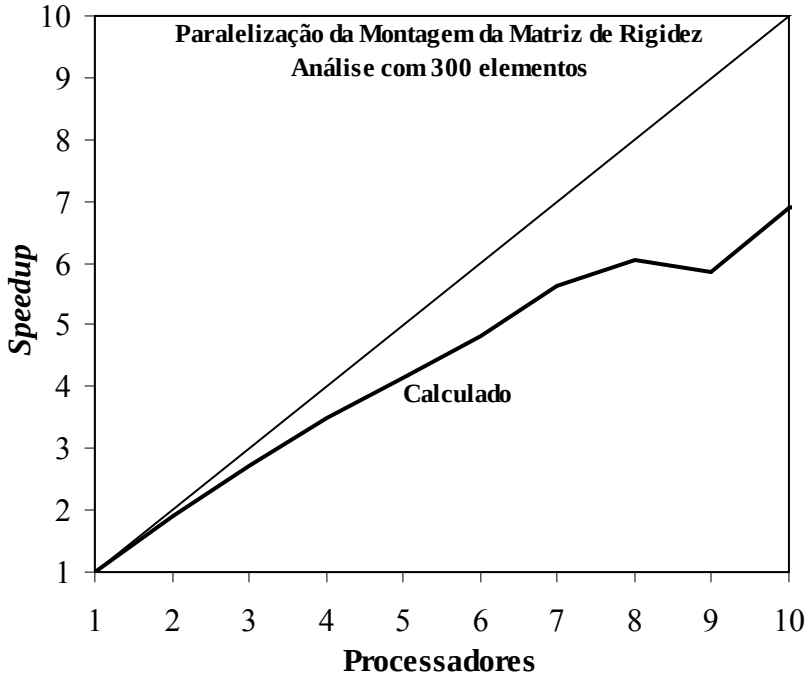

Figura 55 – Análise com 300 Elementos e 10 processadores.

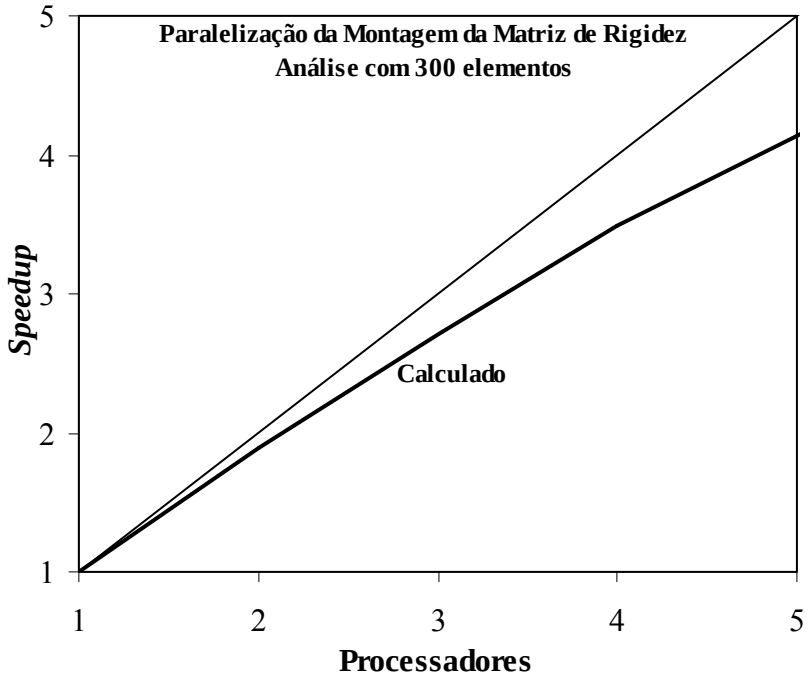

Figura 56 – Análise com 300 Elementos e 5 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

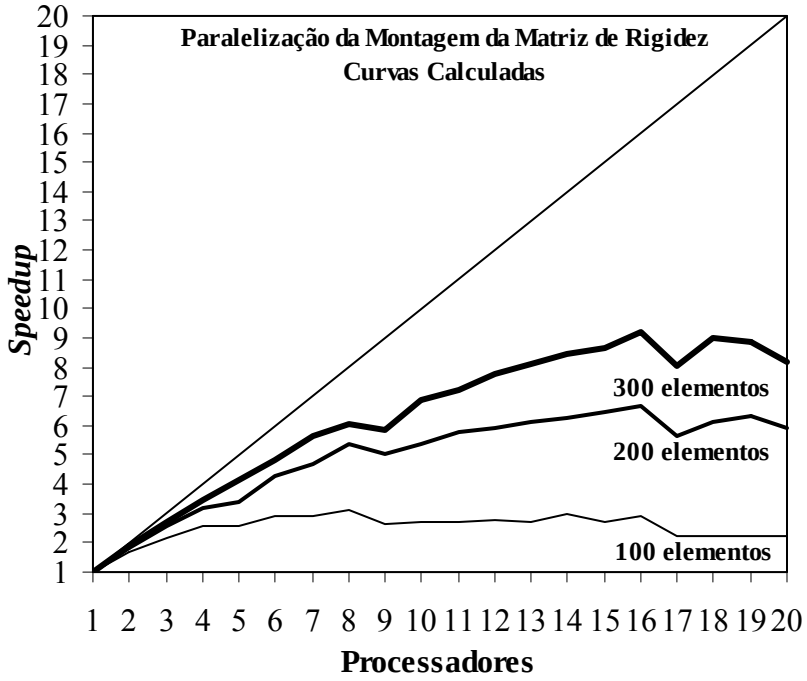

Figura 57 – Comparação dos Resultados com 20 processadores.

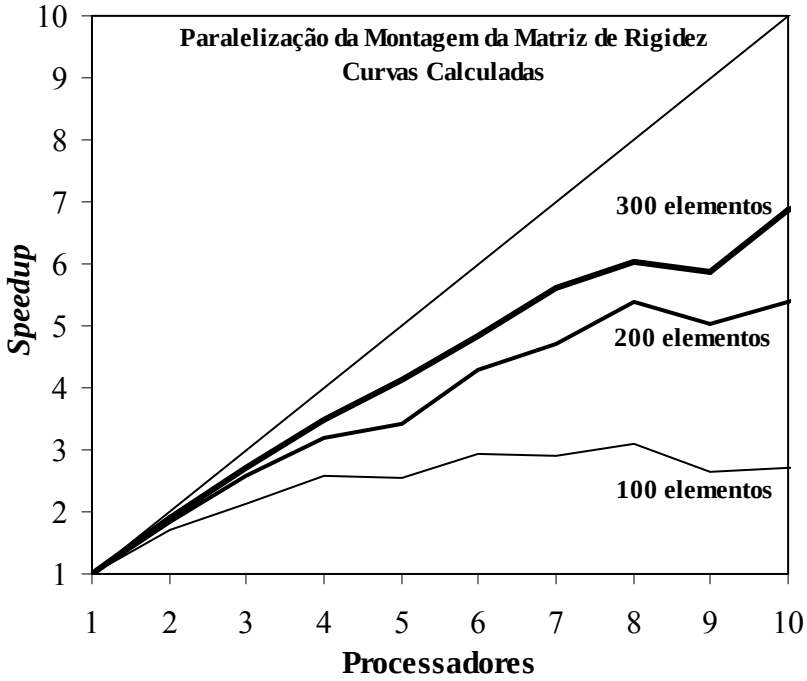

Figura 58 – Comparação dos Resultados com 10 processadores.

PUC-Rio - Certificação Digital Nº 0221221/CA

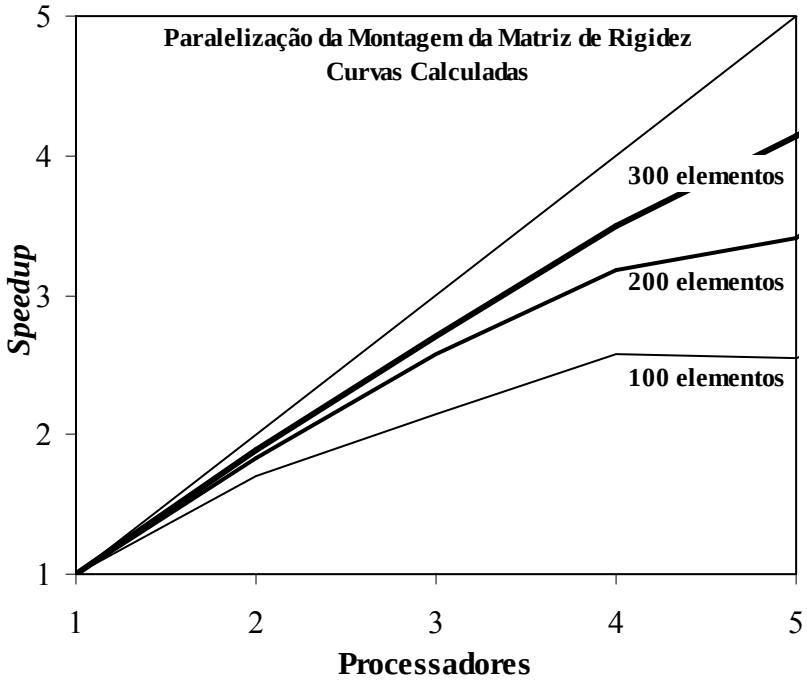

Figura 59 – Comparação dos Resultados com 5 processadores.

7.5
Comparação das Estratégias de Paralelização

Os gráficos apresentados a seguir mostram uma comparação do ganho de performance obtido com as duas estratégias de paralelização.

PUC-Rio - Certificação Digital Nº 0221221/CA

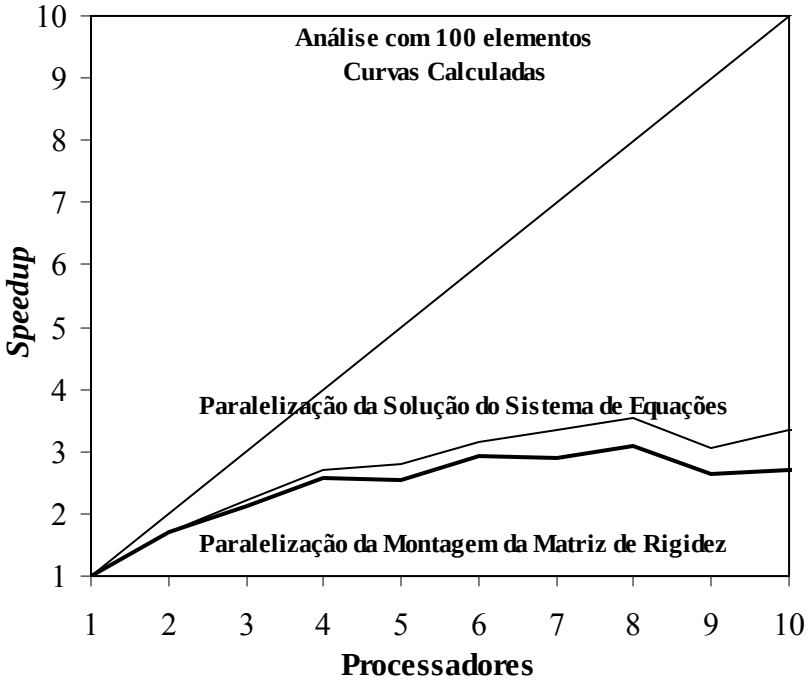

Figura 60 – Comparação dos Resultados com 100 Elementos.

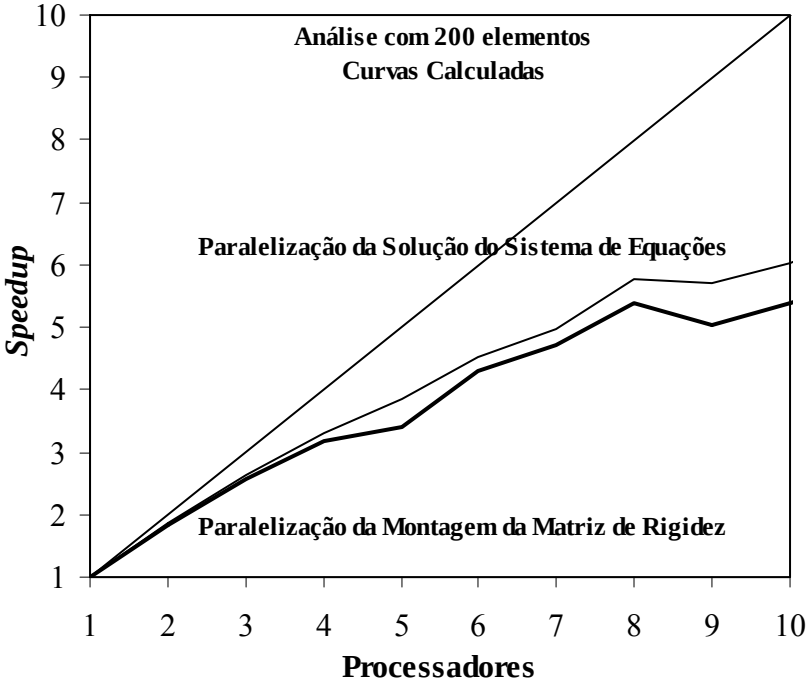

Figura 61 – Comparação dos Resultados com 200 Elementos.

PUC-Rio - Certificação Digital Nº 0221221/CA

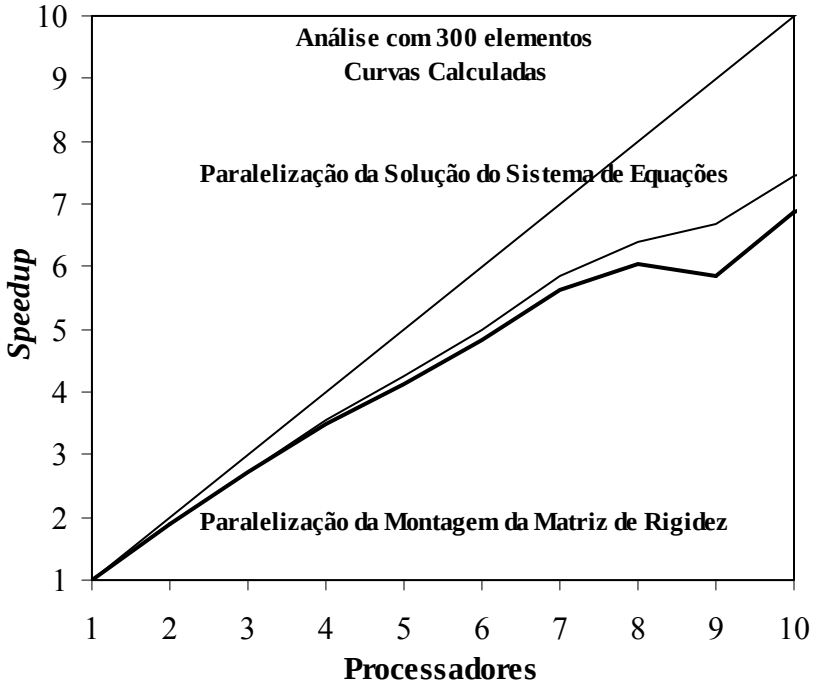

Figura 62 – Comparação dos Resultados com 300 Elementos.

PUC-Rio - Certificação Digital Nº 0221221/CA

8
Conclusões

8.1
Análise dos resultados

Os exemplos apresentados no capítulo anterior tiveram como objetivo verificar a correção dos resultados obtidos com o código implementado e o ganho de performance alcançado com o emprego das duas estratégias de paralelização apresentadas no capítulo 6.

É importante destacar que estes resultados foram obtidos empregando-se um cluster de computadores que garanta acesso exclusivo aos seus nós, de forma a garantir que os mesmos recursos computacionais estivessem disponíveis em todas as simulações consideradas.

Nos tópicos a seguir serão apresentadas as conclusões obtidas a partir destes resultados.

8.2
Ganho de Performance

Uma análise dos resultados obtidos empregando-se as duas estratégias de paralelização permite que se obtenham as seguintes constatações, para o modelo estrutural analisado:

a) Em todas as análises, para todos os modelos (com 100, 200 e 300 elementos) e nas duas estratégias de paralelização, verifica-se que o *speedup* aumenta com a quantidade de processadores empregados até um valor máximo, a partir do qual o ganho com o processamento paralelo é superado pelo tempo de comunicação entre os processos. Isto ocorre nas duas estratégias implementadas e está de acordo com o descrito na literatura consultada sobre o assunto.

b) Para uma determinada quantidade de processadores, o *speedup* aumenta com a complexidade do modelo (número de elementos). É importante destacar

PUC-Rio - Certificação Digital Nº 0221221/CA

que o que aumenta é o ganho de performance, e não o tempo absoluto de processamento.

c) O coeficiente de paralelização aumenta com o número de elementos do modelo, como pode ser verificado no gráfico da figura 63.

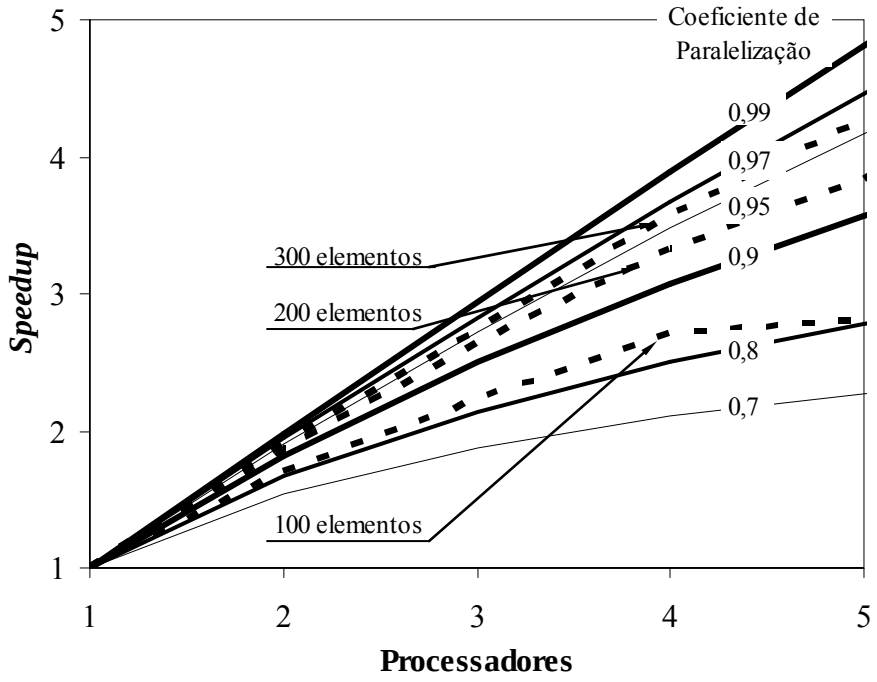

Figura 63 – Análise do Coeficiente de Paralelização.

É necessário considerar, no entanto, que estes resultados foram obtidos para um modelo estrutural particular, e empregando-se elementos quadriláteros lineares, e que resultados distintos poderiam ter sido obtidos com elementos mais complexos, com funções de interpolação de grau mais elevado, e nos quais o tempo de comunicação não fosse tão preponderante.

8.3
Comparação entre as Estratégias

Uma análise dos resultados obtidos empregando-se as duas estratégias de paralelização mostra que, na segunda estratégia, em que se paralelizou o processo de montagem da matriz de rigidez global, o tempo gasto na comunicação entre os processadores supera o ganho obtido ao se dividir esta tarefa entre os

PUC-Rio - Certificação Digital Nº 0221221/CA

processadores, e este fato foi verificado em todos os modelos analisados (com 100, 200 e 300 elementos)..

Deve-se, no entanto, considerar as particularidades do modelo adotado, antes de se considerar tal resultado como uma regra geral.

8.4
Conclusões Finais e Sugestões para Trabalhos Futuros

No que se refere ao emprego da programação orientada a objetos, pode-se concluir que o seu emprego fornece um código mais compacto e de manutenção simplificada.

No que se refere ao emprego da computação paralela os resultados apresentados mostram que, principalmente em modelos de grande complexidade, o ganho de performance obtido pode representar uma redução significativa do custo computacional envolvido na análise. Verifica-se, portanto, a viabilidade do emprego de *clusters* de computadores, nos quais se garanta a uma aplicação acesso exclusivo aos nós requisitados para o processamento distribuído.

É importante destacar, no entanto (como já mencionado anteriormente), que o aumento indiscriminado de processadores pode representar um desperdício de recursos computacionais, pois a partir de um determinado número de processadores este ganho de performance tende a se reduzir.

Como sugestões para trabalhos futuros podem ser consideradas as seguintes complementações a este trabalho:

- A avaliação dos resultados com o emprego de outros processos numéricos para a solução do sistema de equações, aproveitando as vantagens da manutenção simplificada de código proporcionada pela programação orientada a objetos.

- A incorporação de novos elementos à biblioteca, que permitam, por exemplo, a solução de modelos que tridimensionais..

- A comparação com os resultados obtidos com outras bibliotecas (como PVM e OpenMP, por exemplo).

PUC-Rio - Certificação Digital Nº 0221221/CA

9

Referências Bibliográficas

[1] CLOUGH, R.W. The Finite Element in plane stress analysis. **Proc. 2nd ASCE Conf. on Eletronic Computation.** Pittsburgh, Pa. Sept. 1960.

[2] BAUGH, J.W. and REHAK, D.R., Computational Abstractions For Finite Element Programming. In **Technical Report R-89-182**, Department of Civil Engineering, Carnegie Institute of Technology, 1989.

[3] ARCHER, G.C. **Object-oriented finite element analysis.** PhD Thesis. Univ of California, Berkeley; 1996.

[4] FENVES, G.L. Object-Oriented Programming For Engineering. **Software Development Engineering with Computers**, V6 N1: 1-15, 1990.

[5] FORDE, B.W.R.; FOSCHI, R.O. and STIEMER, S.F. Object-Oriented Finite Element Analysis. **Computers & Structures,** V34 N3: 355-374, 1990.

[6] SCHOLZ, S.P. Elements of an Object-Oriented FEM++ Program in C++. **Computers & Structures**, Vol. 43, No. 3, 517-529, 1992.

[7] FILHO, J.S.R.A. and DEVLOO, P.R.B. Object-Oriented Programming in Scientific Computations: The Beginning of a New Era. **Engineering Computations**, Vol. 8, 81-87, 1991.

[8] MACKIE, R.I. Object-Oriented Programming Of The Finite Element Method. **International Journal for Numerical Methods in Engineering**, V35 N2: 425-436, 1992.

[9] PIDAPARTI, R.M.V. and HUDLI, A.V., Dynamic Analysis of Structures Using Object-Oriented Techniques, **Computers & Structures**, Vol. 49, No. 1, 149-156, 1993.

[10] RAPHAEL, B. and KRISHNAMOORTHY, C.S. Automating Finite Element Development Using Object-Oriented Techniques, **Engineering Computations**, Vol. 10, 267-278, 1993.

[11] ZEGLINSKI, G.W.; HAN, R.P.S. and AITCHISON, P. Object-Oriented Matrix Classes For Use In A Finite Element Code Using C++. **International Journal for Numerical Methods in Engineering**, V37 N22: 3921-3937, 1994.

[12] LU, J.; WHITE, D.W.; CHEN, W.F. and DUNSMORE, H.E. A Matrix Class Library in C++ For Structural Engineering Computing, **Computers & Structures**, Vol. 55, No. 1, 95-111, 1995.

PUC-Rio - Certificação Digital Nº 0221221/CA

[13] HAMMARLING, S.; MCKENNEY, A.; OSTROUCHOV, S. and SORENSEN, D. **LAPACK User's Guide**. SIAM Publications, Philadelphia, 1992.
[14] DONGARRA, J.J.; POZO, R. and WALKER D.W. LAPACK++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra, **ACM** 0-8186-4340-4/93/0011, 1993.

[15] ZAHLTEN, W.; DEMMERT, P. and KRATZIG, W.B. An Object-Oriented Approach to Physically Nonlinear Problems in Computational Mechanics. **Computing in Civil and Building Engineering**, Rotterdam, 1995.

[16] SAMPATH, R. and ZABARAS, N. An object oriented implementation of a front tracking finite element method for directional solidification processes, **Int. J. Numer. Methods Eng.** 44 (9) 1227-1265, 1999.

[17] MILLER, G.R. An Object-Oriented Approach To Structural Analysis And Design. **Computers & Structures**, V40 N1: 75-82, 1991.

[18] MILLER, G.R. Coordinate-Free Isoparametric Elements. **Computers & Structures,** V49 N6: 1027-1035, 1993.

[19] MILLER, G.R. and RUCKI, M.D. A Program Architecture for Interactive NonLinear Dynamic Analysis of Structures, **Second Congress on Computing in Civil Engineering,** ASCE, 1994.

[20] DUBOISPELERIN, Y. and ZIMMERMANN, T. Object-Oriented Finite Element Programming .3. An Efficient Implementation In C++. **Computer Methods in Applied Mechanics and Engineering**, V108 N1-2: 165-183, 1993.

[21] DUBOISPELERIN, Y.; ZIMMERMANN, T. and BOMME, P. Object-Oriented Finite Element Programming .2. A Prototype Program In Smalltalk. **Computer Methods in Applied Mechanics and Engineering**, V98, N3: 361-397, 1992.

[22] MENETREY, P. and ZIMMERMANN, T. Object-Oriented Non-Linear Finite Element Analysis - Application to J2 Plasticity. **Computers & Structures,** V49 N5: 767-777, 1993.

[23] ZIMMERMANN, T.; DUBOISPELERIN Y. and BOMME, P. Object-Oriented Finite Element Programming .1. Governing Principles. **Computer Methods in Applied Mechanics and Engineering**, V98, N2: 291-303, 1992.

[24] YU, G. and ADELI, H. Object-Oriented Finite Element Analysis Using EER Model. **Journal of Structural Engineering** - ASCE, V119 N9: 2763-2781, 1993.

[25] HEDEDAL, O. **Object-oriented structuring of finite elements**. PhD Thesis. Aalborg University, Denmark; 1994.

[26] BITTENCOURT, M.L. Using C++ templates to implement finite element classes. **Eng Comput** 17(7): 775-88, 2000.

PUC-Rio - Certificação Digital Nº 0221221/CA

[27] MARTHA, L.F. and PARENTE, JR., E. An Object-Oriented Framework for Finite Element Programming, **Proceedings of the Fifth World Congress on Computational Mechanics**, IACM, Vienna, Austria, on-line publication (ISBN 3-9501554-0-6), http://wccm.tuwien.ac.at, Paper-ID: 80480, p. 10, 2002.

[28] ZIMMERMANN, T. et al. Aspects of an object-oriented finite element environment. **Comput Struct** 68(1/3): 1-16, 1998.

[29] YU, L. and KUMAR, A.V. An object-oriented modular framework for implementing the finite element method. **Comput Struct**; 79(9): 919-28, 2001..

[30] KOO, D. Object-oriented parser-based finite element analysis tool interface. **Proc. SPIE** 3833: 121-32, 1999.

[31] REIMANN, K.; GIL, L.; JENTSCH, M. and SANCHEZ, M. SCOPE, a framework of objects to develop structural analysis programs in C++, **Developments in engineering computational technology**, Civil-Comp Press, Edinburgh, UK, 2000.

[32] MACKIE, R.I. **Object oriented methods and finite element analysis**. Edinburgh: Saxe-Coburg Publ; 2000.

[33] PRIETO, M.; LLORENTE I. and TIRADO, F. A review of regular domain partitioning. **SIAM News**, 33, 1, 2000.

[34] SMITH, B.; BJORSTAD P. and GROPP, W. **Domain Decomposition: Parallel multilevel methods for elliptic partial differential equations**. Cambridge University Press, 1996.

[35] TOPPING, B.H.V and KHAN, A.I. **Parallel Finite Element Computations**. Saxe-Coburg, 1996.

[36] LIN, H.X. **A methodology for the parallel direct solution of finite element systems**. PhD Thesis, Delft University of Technology, 1993.

[37] GUPTA, A.; GUSTAVSON, F.; JOSHI M.; KARYPIS, G. and KUMAR, V. Design and implementation of a scalable parallel direct solver for sparse symmetric positive definite systems. **Proceedings of the Eighth SIAM Conference on Parallel Processing**, 1997.

[38] SCOTT, J.A. The design of a parallel frontal solver. **Rutherford Appleton Laboratory Technical Report**, RAL-TR-1999-075, 1999.

[39] SCOTT, J.A. A parallel frontal solver for finite element applications. **Numerical Methods in Engineering**, 50, 1131-1144, 2001.

[40] HUGHES, J.; LEVIT, I. and WINGET. An element by element solution algorithm for problems of structural and solid mechanics. **Computer Methods in Apllied Mechanics and Engineering**, 36, pp 241-254, 1983.

PUC-Rio - Certificação Digital Nº 0221221/CA

[41] PETTIPHER; M.A. and SMITH, I.M. The development of an MPP implementation of a suite of finite element codes. **Computer Science**, No 1225, pp 400-409, 1997.

[42] BANE, M.; KELLER, R.; PETTIPHER, M.A. and SMITH, I.M. **A comparison of MPI and OpenMP Implementations of a Finite Element Analysis Code**, 2000.

[43] GULLERUD, A.S. and DODDS, R.H. MPI-based implementation of a PCG solverusing an EBE architecture and preconditioner for implicit, 3D finite element analysis. **Computers and Structures**, 79, 553-575, 2001.

[44] JEREMIC, B. and STURE, S. Tensor objects in finite element programming, **Int. J. Numer. Methods Eng**. 41 (1) 113-126, 1998.

[45] ADELI, H. and YU, G. An Integrated Computing Environment for Solution of Complex Engineering Problems Using the Object-Oriented Programming Paradigm and a Blacboard Architecture, **Computers & Structures**, Vol. 54, No. 2, 255-265, 1995.

[46] KONG, X.A. and CHEN, D.P. An object-oriented design of FEM programs. **Comput Struct** 57(1): 157-66, 1995.

[47] KONG, X.A. A data design approach for object-oriented FEM programs. **Comput Struct** 61(3): 503-13,1996.

[48] DUBOIS-PELERIN Y. and PEGON, P. Object-oriented programming in nonlinear finite element analysis. **Comput Struct** 67(4): 225-41, 1998.

[49] ARCHER, G.C. et al. A new object-oriented finite element analysis program architecture, **Comput. Struct**. 70 (1) 63-75, 1999.

[50] MACKIE, R.I. Object-Oriented Finite Element Programming – The Importance of Data Modelling, **Advances in Engineering Software**, Vol. 30, pp 775-782, 1999.

[51] YU, G.G. **Object-oriented models for numerical and finite element analysis**, Ph.D. Thesis, The Ohio State Univ., 1994.

[52] SIMS, J.M. **An object-oriented development system for finite element analysis**, Ph.D. Thesis, Arizona State Univ., 1994.

[53] ZIENKIEWICZ, O.C. and TAYLOR, R.L. **The Finite Element Method**, 5a Ed, 3 Vol. Butterworth-Heinemann, 2000.

[54] COOK, R.D.; MALKUS, D.S.; PLESHA, M.E. and WITT, R.J. **Concepts and Applications of the Finite Element Method**, 4a Ed., Wyley & Sons, 2001.

PUC-Rio - Certificação Digital Nº 0221221/CA