

Um Sistema Paralelo Integrado para Análise de Estruturas

C.O. Moretti¹, T.N. Bittencourt¹, J.C. André¹, L.F. Martha²

¹ *Laboratório de Mecânica Computacional
Departamento de Engenharia de Estruturas e Fundações
Escola Politécnica - USP*

² *Departamento de Engenharia Civil
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio*

1. Introdução

Neste trabalho será apresentado o protótipo de um sistema paralelo integrado para a análise de estruturas, que utiliza uma rede local de computadores como ambiente de processamento. O sistema desenvolvido é formado por diversos componentes integrados, cada um responsável por uma parte específica da análise. A Figura 1 apresenta as quatro unidades básicas desse sistema paralelo.

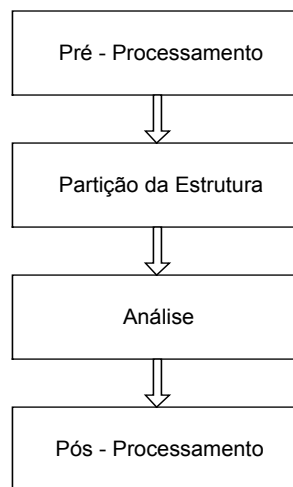


Figura 1: Unidades básicas do sistema paralelo de análise

Os principais objetivos observados durante o desenvolvimento desse sistema foram:

- reduzir o tempo gasto na análise de estruturas, através da divisão do trabalho entre vários processadores, possibilitando ao usuário analisar diferentes soluções para um problema em um menor tempo;
- possibilitar a análise de estruturas de grande porte, que não poderiam ser processadas em máquinas isoladas, devido à exigência de grande capacidade de memória e armazenamento de dados;
- possibilitar o acesso à análise paralela para a maioria dos usuários, com baixo custo, utilizando-se de uma rede local de computadores;

- melhorar o aproveitamento dos recursos disponíveis em uma rede local de computadores.

Nas seções seguintes, são apresentados todos os componentes do sistema paralelo, com destaque para as implementações feitas no programa de análise FEMOOP.

2. Pré-Processamento

Nesta fase, é gerado o modelo a ser analisado, através da definição de seus atributos. O pré-processador utilizado foi o MTOOL (*Bidimensional Mesh Tool*) [Mtool,92], desenvolvido pelo Grupo de Tecnologia em Computação Gráfica (TeCGraf) da PUC-Rio.

O MTOOL é um programa gráfico interativo para a geração de malhas de elementos finitos bidimensionais. Utilizando-se esse programa, definem-se os diversos atributos necessários para a análise de um modelo, como sua geometria, propriedades dos materiais, condições de suporte e de carregamento. Gera-se também a malha de elementos finitos e, por meio de sua interface gráfica, é possível visualizar e editar a malha resultante. A Figura 2 mostra a interface gráfica do programa MTOOL.

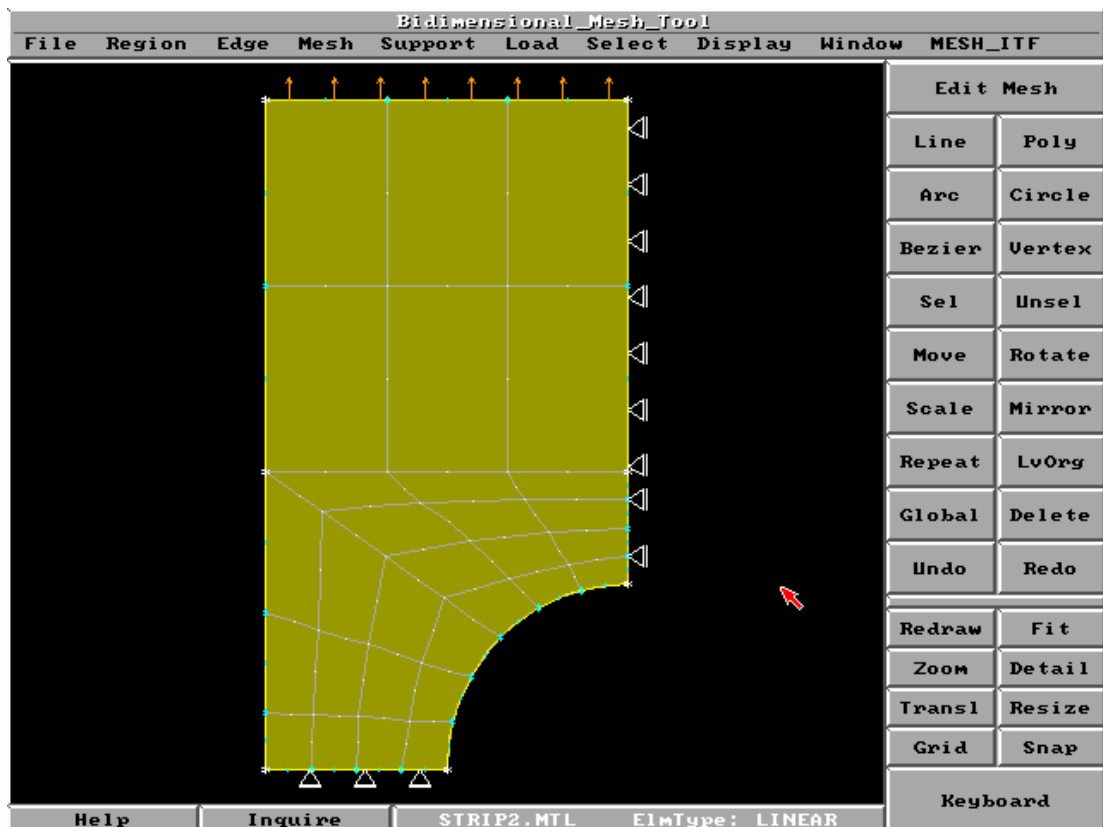


Figura 2: Interface gráfica do programa MTOOL

Ao final da criação do modelo, é gerado um arquivo de formato neutro que, de maneira padronizada, contém todas as informações do modelo necessárias para a análise. Por meio desse arquivo, será feita a integração entre os diversos componentes do sistema paralelo.

3. Partição da Estrutura

Após a geração do modelo, para se aproveitar as vantagens que o ambiente paralelo oferece, é necessária a partição da estrutura em um determinado número de subestruturas. Em geral, o número de subestruturas é igual ao número de processadores disponíveis. Nessa partição, busca-se distribuir de maneira equilibrada o trabalho entre os processadores e minimizar os graus de liberdade na fronteira entre as subestruturas.

Para essa tarefa, desenvolveu-se o programa PARTDOM [Moretti,97], o qual lê as informações do modelo contidas no arquivo neutro, gerado na fase de pré-processamento, e, utilizando-se de algoritmos automáticos de partição de domínio, divide a estrutura no número desejado de subestruturas. Ao final desse processo, são adicionadas ao arquivo neutro as informações relativas à partição da estrutura.

A Figura 3 mostra a interface gráfica do PARTDOM.

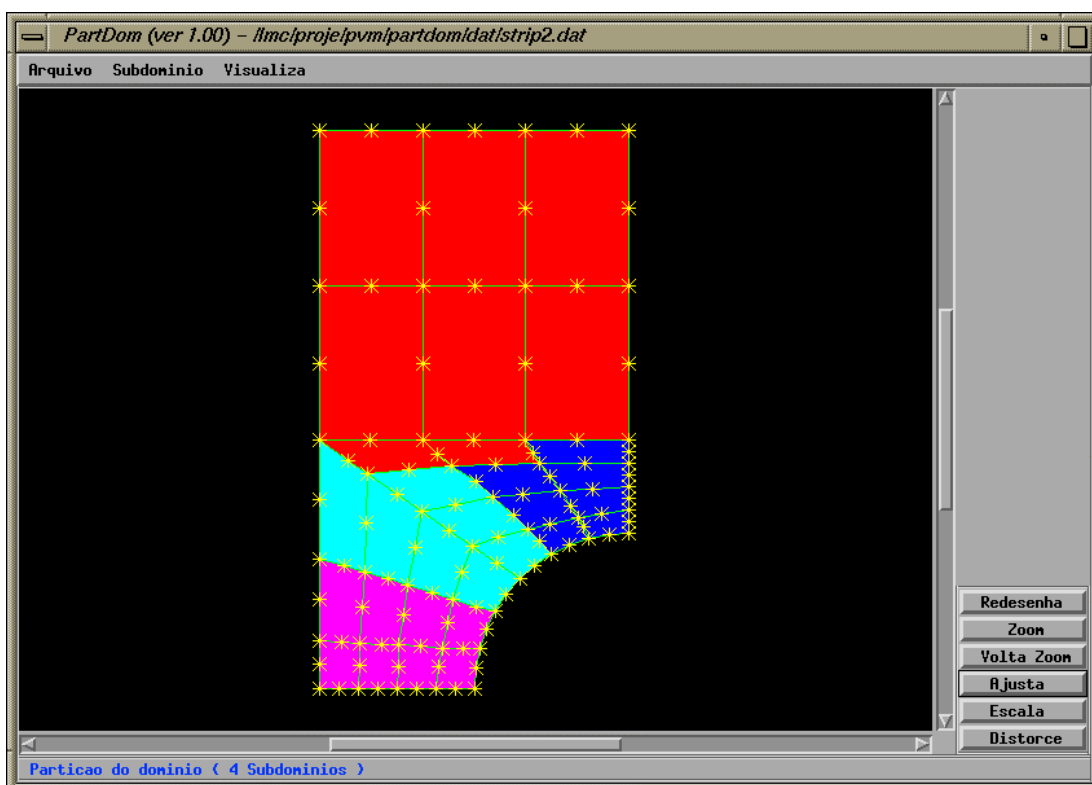


Figura 3: Interface gráfica do programa PARTDOM

4. Análise

Após a geração do modelo e a sua partição em subdomínios, o próximo passo consiste na análise em paralelo desse modelo, utilizando-se vários processadores, cada um responsável por uma parcela do trabalho computacional envolvido na sua resolução. Neste trabalho, foi utilizado o Método dos Elementos Finitos para a análise do modelo, por meio do programa FEMOOP (*Finite Element Method - Object Oriented Programming*) [Martha,96], desenvolvido conjuntamente pelo Departamento de

Engenharia Civil da PUC-Rio e pelo Laboratório de Mecânica Computacional (LMC) da Escola Politécnica da USP.

O programa FEMOOP é baseado no paradigma da programação orientada para objetos, e emprega a linguagem de programação C++ [Fujii,97] [Guimarães,92]. Um dos benefícios mais importantes da programação orientada para objetos é a extensibilidade do código, permitindo que novas implementações sejam feitas com pequeno impacto sobre o código já existente. Ao longo dessa seção, serão utilizados conceitos de programação orientada para objetos na apresentação das implementações feitas no FEMOOP. Basicamente, serão abordados os conceitos de classe, derivação de classe, e objetos. Para maiores informações sobre a programação orientada para objetos e seus conceitos fundamentais, recomenda-se consultar [Weiskamp,90].

Para adaptar o FEMOOP ao ambiente paralelo, foi necessária a criação de uma nova classe, responsável pela manipulação e armazenagem de dados referentes aos subdomínios. Também foi preciso implementar várias funções novas dentro das classes já existentes.

Nas seções a seguir, serão apresentadas as principais implementações feitas no FEMOOP, além de detalhes do funcionamento do programa ao se realizar uma análise utilizando vários processadores.

4.1 Biblioteca de funções paralelas

A primeira etapa no trabalho de adaptar o FEMOOP ao ambiente paralelo, consistiu na implementação de uma biblioteca de funções para gerenciar as trocas de mensagens entre os processadores. Essa biblioteca funciona basicamente como uma interface entre o FEMOOP e as funções do PVM (*Parallel Virtual Machine*) [Geist,94], gerenciador de troca de mensagens utilizado neste trabalho. Dessa forma, busca-se também isolar as funções do PVM em uma única área do programa, facilitando uma eventual mudança no gerenciador de trocas de mensagens, o que exigiria alterações apenas nessa pequena área do código.

A biblioteca possui todas as funções necessárias à tarefa de troca de mensagens em um ambiente de memória distribuída. As funções principais implementadas nessa biblioteca são as responsáveis pelo envio e recepção de mensagens, sendo que as mensagens podem conter vetores de valores inteiros, de dupla precisão ou ainda uma seqüência de caracteres. Existem também funções de inicialização do processamento paralelo e identificação do tipo de programa em execução (se é um programa mestre ou uma tarefa). Esses tipos de programa são característicos da programação paralela, onde um programa mestre é o responsável pela inicialização e disparo das tarefas, que são programas executados concorrentemente e que interagem entre si e com o programa mestre por meio de trocas de mensagens. A biblioteca possui também uma função específica para executar o disparo de tarefas.

Neste trabalho, para o envio e recepção das mensagens, foi utilizada uma área da memória (*buffer*) para a armazenagem da mensagem antes do envio ou depois da recepção. Assim, ao se enviar uma mensagem, os dados são copiados para esse *buffer* e então enviados. Esta é a forma mais segura de se trabalhar com troca de mensagens,

uma vez que não se corre o risco de um dado da mensagem ser alterado antes do seu envio.

Para receber as mensagens, utilizou-se a recepção do tipo bloqueada (*blocking receive*). Neste caso, o processamento é interrompido até que a mensagem seja recebida. Foi dessa forma que se controlou a sincronia entre os processos paralelos.

4.2 Seqüência das etapas da execução do FEMOOP

Nesta seção, serão descritas de maneira geral as diversas etapas da execução do programa FEMOOP ao se realizar uma análise em paralelo. O objetivo dessa descrição é mostrar o funcionamento do programa sem entrar em detalhes de programação, apresentando as diversas fases de uma análise pelo Método dos Elementos Finitos em um ambiente paralelo. Quando utilizadas, as funções da biblioteca de troca de mensagens serão identificadas. O nome dessas funções iniciam-se sempre com o prefixo 'paral_'.

Ao ser adaptado ao ambiente paralelo, o programa FEMOOP passou a ter a capacidade de funcionar tanto como programa mestre como tarefa. Assim, ao ser executado pelo usuário, o FEMOOP identifica-se como mestre e dispara outras unidades do próprio FEMOOP, as quais identificam-se e funcionam como tarefas. A identificação do tipo de programa é uma das primeiras etapas de execução. As etapas iniciais, que são comuns tanto para o programa mestre quanto para as tarefas, são descritas a seguir:

- *Inicialização do processamento paralelo*: cada programa, seja mestre ou tarefa, possui um número inteiro que o identifica dentro da máquina virtual paralela. Esse número, chamado de identificador da tarefa (*tid*), é obtido pela chamada a uma função da biblioteca de gerenciamento de trocas de mensagens (`paral_mytid`). Essa é sempre a primeira etapa a ser executada pelos programas do sistema paralelo;
- *Identificação do tipo de programa*: nessa etapa, o programa, por meio de uma chamada à função `paral_I_am_a_Parent`, identifica-se como mestre ou tarefa.

A partir desse ponto, a seqüência da execução do programa mestre é diferente da execução das tarefas. Assim, a descrição será dividida em duas partes: uma referente ao programa mestre e outra referente às tarefas.

4.2.1 Programa Mestre

Após a inicialização do processamento paralelo e a identificação do tipo de programa, a seqüência de atividades do programa mestre pode ser assim resumida:

- *Abertura do arquivo de dados*: ao ser executado, o programa pede o nome do arquivo de formato neutro referente ao modelo a ser analisado, e prepara-o para as diversas leituras que serão feitas ao longo da execução;
- *Identificação do tipo de análise*: por meio da leitura do arquivo neutro, o programa identifica se a análise é do tipo seqüencial ou paralela. Se o arquivo

possuir informações a respeito da partição do modelo em subdomínios, a análise a ser executada é paralela. Se nenhuma informação sobre os subdomínios for encontrada, a análise é feita de maneira seqüencial, sem a utilização das funções de troca de mensagens e das novas implementações.

- *Leitura dos dados referentes aos subdomínios e criação de objetos da classe `Subdom`*: ao ler o número de subdomínios ao qual o modelo foi dividido, o programa cria um número igual de objetos da classe `Subdom`, cada um correspondendo a um subdomínio do modelo. Essa classe foi criada para ser a responsável pela manipulação e armazenagem dos dados referentes aos subdomínios, e será descrita em detalhes na próxima seção. Após a criação dos objetos, prossegue-se com a leitura dos dados dos subdomínios, armazenando-se esses dados nos objetos correspondentes. São lidas informações sobre os elementos, nós internos, nós de fronteira e subdomínios vizinhos, referentes à cada subestrutura.
- *Disparo das tarefas e envio de dados*: uma vez conhecido o número de subdomínios, dispara-se um número igual de tarefas, por meio da função `paral_spawn`, e envia-se os dados lidos na etapa anterior para as tarefas correspondentes.
- *Leitura e envio das informações do modelo*: o próximo passo consiste na leitura das informações sobre a geometria, carregamentos, condições de suporte e deslocamentos impostos referentes ao modelo, e no envio dessas informações às tarefas correspondentes.
- *Período de espera*: após o envio de todas as informações às tarefas, essas serão responsáveis pela análise em paralelo do modelo. Durante esse processo, o programa mestre não executa nenhuma função, esperando pelos resultados da análise.
- *Recepção dos resultados da análise*: ao final da análise, as tarefas enviam os resultados para o programa mestre. Esses resultados incluem informações sobre os deslocamentos e tensões, além dos tempos gastos em cada etapa da análise.
- *Impressão dos resultados*: o programa mestre é responsável pela adição dos resultados no arquivo de formato neutro, o qual será lido pelo pós-processador para a visualização desses resultados. Além disso, os tempos gastos em cada etapa da análise são impressos na tela, podendo ser redirecionados para um arquivo.

4.2.2 Tarefas

Após a inicialização do processamento paralelo e a identificação do tipo de programa, a seqüência de atividades das tarefas pode ser assim resumida:

- *Recepção dos dados do subdomínio*: as informações, referentes ao subdomínio que está sendo analisado pela tarefa, são recebidas e armazenadas em um objeto da classe `Subdom`. São informações sobre os elementos, nós internos, nós de fronteira e subdomínios vizinhos.
- *Recepção das informações do modelo*: são recebidas as informações do modelo como geometria, carregamentos, condições de suporte e deslocamentos impostos, referentes ao subdomínio em análise pela tarefa.

- *Análise em paralelo*: tem início, então, a análise em paralelo do modelo. Nessa etapa, será utilizada a técnica de subestruturação [Nour,87], que possibilita a resolução do sistema de equações lineares quando este apresenta-se particionado entre várias tarefas.
- *Montagem da matriz de rigidez e do vetor de esforços*: cada tarefa monta a matriz de rigidez e o vetor de esforços correspondentes ao subdomínio pelo qual está responsável. Ao final dessa etapa, a matriz de rigidez e o vetor de esforços do modelo completo encontram-se particionados entre as várias tarefas.
- *Montagem da matriz de rigidez e do vetor de esforços reduzidos*: é feita, então, a eliminação das incógnitas correspondentes aos graus de liberdade dos nós internos do subdomínio, obtendo-se a matriz de rigidez e o vetor de esforços reduzidos, correspondentes apenas aos graus de liberdade dos nós de fronteira.
- *Resolução do sistema de equações lineares em paralelo*: utilizando-se o método dos gradientes conjugados pré-condicionados implementado em paralelo, resolve-se o sistema de equações lineares formado pela matriz de rigidez e vetor de esforços reduzidos de cada subdomínio, obtidos na etapa anterior. Ao final da resolução, obtêm-se os deslocamentos dos nós de fronteira entre subestruturas. A implementação em paralelo do método dos gradientes conjugados pré-condicionados será descrita neste trabalho.
- *Montagem do vetor de deslocamentos*: conhecidos os deslocamentos dos nós de fronteira, são calculados os deslocamentos dos nós internos da subestrutura, obtendo-se assim todos os deslocamentos do modelo.
- *Cálculo das tensões*: cada tarefa calcula, então, as tensões que aparecem no subdomínio pelo qual está responsável. Essa etapa é feita concorrentemente, sem a necessidade de troca de informações entre as tarefas. Não foram necessárias novas implementações para realizar o cálculo das tensões.
- *Envio dos resultados*: a última etapa consiste no envio dos valores dos deslocamentos e tensões para o programa mestre, o qual é responsável pela organização e impressão dos resultados. São enviados também os tempos gastos em cada etapa da análise, o que possibilita a monitoração de todo o processo de análise em paralelo.

4.3 Descrição da classe `Subdom`

No processo de adaptação do programa FEMOOP ao ambiente paralelo, foi criada uma nova classe, dentro da estrutura de classes já existentes, responsável pela manipulação e armazenagem dos dados correspondentes aos subdomínios do modelo. Essa nova classe foi chamada de `Subdom` e nesta seção serão apresentadas as principais funções e dados presentes nessa classe.

Os dados comuns a todos os objetos da classe `Subdom` são: número total de subdomínios, identificador (*tid*) do programa mestre, identificadores de todas as tarefas, e número total de graus de liberdade na fronteira entre os subdomínios.

Os dados exclusivos de cada objeto da classe `Subdom`, referentes a cada subdomínio são: número de nós, número e a lista de elementos, número e a lista de nós

internos, número e a lista de nós de fronteira, e número e a lista de subdomínios vizinhos.

As principais funções da classe `Subdom` são descritas resumidamente a seguir (todas as funções citadas nessas descrições pertencem à classe `Subdom`):

- `Init`: é a função de inicialização dos objetos e de envio de dados às tarefas. Executada pelo programa mestre, essa função lê, a partir do arquivo de formato neutro, o número de subdomínios e todas as informações sobre os subdomínios (elementos, nós internos, nós de fronteira e subdomínios vizinhos), utilizando-se a função `ReadAll`, também pertencente à classe `Subdom`. Criam-se, então, os objetos correspondentes aos subdomínios, onde são armazenados os dados lidos. A seguir, utilizando-se a função `SpawnProcesses`, são disparadas as tarefas, e, utilizando-se a função `SendTaskInfo`, os dados armazenados nos objetos são enviados às tarefas correspondentes.
- `InitTask`: executada pelas tarefas, essa função cria um objeto da classe `Subdom`, onde são armazenados os dados enviados pelo programa mestre por meio da função `SendTaskInfo`. Para receber esses dados, utiliza-se a função `ReceiveTaskInfo`.
- `MountKsMatrix`: realiza a condensação da matriz de rigidez, montando a matriz de rigidez reduzida da subestrutura, correspondente apenas aos graus de liberdade dos nós da fronteira do subdomínio.
- `MountFsVector`: realiza a condensação do vetor de esforços, montando o vetor de esforços reduzido da subestrutura, correspondente apenas aos graus de liberdade dos nós da fronteira do subdomínio.
- `MountLVector`: monta um vetor correspondente a matriz booleana de ligação **L**, que estabelece a relação entre os graus de liberdade do subdomínio e da estrutura completa.
- `MountFVector`: conhecidos os deslocamentos na fronteira do subdomínio, essa função calcula os deslocamentos dos graus de liberdade internos do subdomínio.

4.4 Implementações feitas nas outras classes

Nesta seção, serão mostradas as principais implementações feitas nas classes já existentes no FEMOOP. Essas implementações, que consistiram na criação de novas funções, foram necessárias para adaptar as classes ao ambiente paralelo. As funções criadas, basicamente, controlam o envio e a recepção de dados entre o programa mestre e as tarefas.

As seções a seguir descrevem, de maneira breve, a funcionalidade das classes para as quais foram feitas novas implementações, além de conter a descrição das novas funções. Uma descrição mais detalhada das classes pertencentes à estrutura do programa FEMOOP encontra-se em [Fujii,97].

4.4.1 Classe AnModel

Contém as informações associadas ao modelo de análise escolhido. Entre os tipos de análise já implementados estão: viga, treliça, estado plano de tensão, estado plano de deformação, placas submetidas à flexão e ao cisalhamento, e sólido. As novas funções implementadas nessa classe foram:

- `Send`: executada pelo programa mestre, lê, a partir do arquivo de formato neutro, o modelo de análise global e envia essa informação para todas as tarefas.
- `Receive`: executada pelas tarefas, recebe a informação sobre o modelo de análise global, enviada pelo programa mestre, e cria um objeto da classe correspondente ao modelo de análise.

4.4.2 Classe Ctrl

Contém as informações que são comuns a diversos tipos de controle de soluções. Podem ser controladores de deslocamento, deformações, forças e tempo. As novas funções implementadas nessa classe foram:

- `Send`: executada pelo programa mestre, lê, a partir do arquivo de formato neutro, o tipo de análise (estática ou dinâmica, linear ou não-linear) e envia essa informação para todas as tarefas.
- `Receive`: executada pelas tarefas, recebe a informação sobre o tipo de análise, enviada pelo programa mestre, e cria um objeto da classe correspondente ao tipo de análise.

4.4.3 Classe Drv

Contém os gerenciadores para os tipos de análises e as funções necessárias para a análise de um problema estrutural, como, por exemplo, numeração dos graus de liberdade da estrutura, montagem da matriz de rigidez, montagem do vetor de esforços, montagem do vetor de esforços internos, montagem do vetor de deformações e resolução do problema. As novas funções implementadas nessa classe foram:

- `Send`: executada pelo programa mestre, lê o modelo de análise global e envia essa informação para todas as tarefas.
- `Receive`: executada pelas tarefas, recebe a informação sobre o modelo de análise global, enviada pelo programa mestre, e cria um objeto da classe correspondente, que funcionará como um gerenciador do tipo de análise.

4.4.4 Classe Element

Contém várias funções que são comuns a todos os tipos de elementos, por exemplo, montagem da matriz de rigidez do elemento, localização do pontos de Gauss e montagem do vetor de esforços internos. As novas funções implementadas nessa classe foram:

- `SendAll`: executada pelo programa mestre, gerencia todo o processo de envio das informações dos elementos às tarefas, utilizando-se as funções de envio descritas a seguir.
- `ReceiveAll`: executada pelas tarefas, gerencia todo o processo de recepção das informações dos elementos, utilizando-se as funções de recepção descritas a seguir.
- `SendThickness`: executada pelo programa mestre, lê o número de espessuras diferentes presentes no modelo e os valores dessas espessuras, e envia essas informações para todas as tarefas.
- `ReceiveThickness`: executada pelas tarefas, recebe e armazena as informações sobre as espessuras presentes no modelo.
- `SendIntegrationOrder`: executada pelo programa mestre, lê o número de ordens de integração diferentes presentes no modelo e os valores dessas ordens de integração, e envia essas informações para todas as tarefas.
- `ReceiveIntegrationOrder`: executada pelas tarefas, recebe e armazena as informações sobre as ordens de integração presentes no modelo.
- `SendSectionProperty`: executada pelo programa mestre, lê o número de seções diferentes presentes no modelo e os valores das propriedades dessas seções, e envia essas informações para todas as tarefas.
- `ReceiveSectionProperty`: executada pelas tarefas, recebe e armazena as informações sobre as propriedades das seções presentes no modelo.
- `SendBeamOrientVector`: executada pelo programa mestre, lê o número de vetores de orientação dos elementos de viga presentes no modelo e os valores desses vetores de orientação, e envia essas informações para todas as tarefas.
- `ReceiveBeamOrientVector`: executada pelas tarefas, recebe e armazena as informações sobre os vetores de orientação dos elementos de viga presentes no modelo.
- `SendBeamEndLiberation`: executada pelo programa mestre, lê o número de liberações das extremidades dos elementos de viga presentes no modelo e os valores dessas liberações das extremidades, e envia essas informações para todas as tarefas.
- `ReceiveBeamEndLiberation`: executada pelas tarefas, recebe e armazena as informações sobre as liberações das extremidades dos elementos de viga presentes no modelo.

Também foram criadas duas novas funções para cada tipo de elemento, as quais são descritas a seguir. Os nomes dessas funções são compostos inicialmente pela palavra `Send` ou `Receive`, seguida pelo tipo do elemento; por exemplo, para elementos do tipo Q8, as funções são nomeadas como `SendQ8` e `ReceiveQ8`.

- `SendTipo_do_Elemento`: executada pelo programa mestre, lê as informações de cada elemento do tipo `Tipo_do_Elemento` e envia essas informações apenas à tarefa que está responsável pelo subdomínio ao qual pertence o elemento. As informações enviadas variam com o tipo de elemento, e podem incluir: número do elemento, material, espessura, ordem de integração, conectividade e tipo de modelo de análise.

- `ReceiveTipo_do_Elemento`: executada pelas tarefas, recebe as informações dos elementos do tipo `Tipo_do_Elemento`, e cria objetos da classe correspondente.

4.4.5 Classe `Fem`

Funciona como um gerenciador da aplicação do Método dos Elementos Finitos, fazendo as chamadas necessárias para a montagem do sistema de equações de equilíbrio do problema em análise.

Foram feitas alterações apenas no construtor dessa classe. Agora, ao se criar um objeto dessa classe, é feita a verificação do tipo de análise (seqüencial ou paralela) e também do tipo de programa (mestre ou tarefa), selecionando-se dessa forma as funções adequadas a serem executadas.

4.4.6 Classe `FemMech`

Contém dados e métodos dos objetos pertencentes ao gerenciador da análise mecânica de uma estrutura. As novas funções implementadas nessa classe foram:

- `SendPrintStress`: executada pelas tarefas, envia os resultados do cálculo das tensões para o programa mestre.
- `ReceivePrintStress`: executada pelo programa mestre, recebe os resultados do cálculo das tensões, enviados pelas tarefas. Ordena, então, os resultados de acordo com a numeração do modelo completo e adiciona esses resultados ao arquivo de formato neutro.

4.4.7 Classe `LoadElement`

Contém os métodos relacionados à classe de carregamentos dos elementos. As novas funções implementadas nessa classe foram:

- `SendAll`: executada pelo programa mestre, gerencia todo o processo de envio das informações dos carregamentos dos elementos, utilizando-se as funções de envio descritas a seguir.
- `ReceiveAll`: executada pelas tarefas, gerencia todo o processo de recepção das informações dos carregamentos dos elementos, utilizando-se as funções de recepção descritas a seguir.
- `SendLineForceUniform`: executada pelo programa mestre, lê as informações sobre as forças uniformemente distribuídas em cada elemento, e envia essas informações apenas à tarefa responsável pelo subdomínio ao qual pertence o elemento.
- `ReceiveLineForceUniform`: executada pelas tarefas, recebe as informações sobre as forças uniformemente distribuídas nos elementos e, para cada carregamento, cria um objeto da classe `ForUniform`.
- `SendLineMomentUniform`: executada pelo programa mestre, lê as informações sobre os momentos uniformemente distribuídos em cada

elemento, e envia essas informações apenas à tarefa responsável pelo subdomínio ao qual pertence o elemento.

- `ReceiveLineMomentUniform`: executada pelas tarefas, recebe as informações sobre os momentos uniformemente distribuídos nos elementos e, para cada carregamento, cria um objeto da classe `MomUniform`.

4.4.8 Classe `Material`

Contém os dados e métodos para as diferentes classes de materiais. As novas funções implementadas nessa classe foram:

- `SendAll`: executada pelo programa mestre, gerencia todo o processo de envio das informações dos materiais, utilizando-se as funções de envio descritas a seguir.
- `ReceiveAll`: executada pelas tarefas, gerencia todo o processo de recepção das informações dos materiais, utilizando-se as funções de recepção descritas a seguir.
- `SendMaterial`: executada pelo programa mestre, lê o número de materiais diferentes presentes no modelo e envia essa informação para todas as tarefas.
- `ReceiveMaterial`: executada pelas tarefas, recebe e armazena a informação sobre o número de materiais diferentes presentes no modelo.

Também foram criadas duas novas funções para cada tipo de material, as quais são descritas a seguir. Os nomes dessas funções são compostos inicialmente pela palavra `Send` ou `Receive`, seguida pelo tipo do material; por exemplo, para materiais elásticos isotrópicos, as funções são nomeadas como `SendElasticIsotropic` e `ReceiveElasticIsotropic`.

- `SendTipo_do_Material`: executada pelo programa mestre, lê as informações de cada material do tipo `Tipo_do_Material` e envia essas informações para todas as tarefas. As informações enviadas variam com o tipo de material, podendo incluir, por exemplo, o módulo de elasticidade e o coeficiente de Poisson.
- `ReceiveTipo_do_Material`: executada pelas tarefas, recebe as informações dos materiais do tipo `Tipo_do_Material`, e cria objetos da classe correspondente.

4.4.9 Classe `Node`

Contém os dados e métodos relacionados aos nós. As novas funções implementadas nessa classe foram:

- `SendAll`: executada pelo programa mestre, gerencia todo o processo de envio das informações dos nós, utilizando-se as funções de envio descritas a seguir.
- `ReceiveAll`: executada pelas tarefas, gerencia todo o processo de recepção das informações dos nós, utilizando-se as funções de recepção descritas a seguir.

- `SendNodeCoord`: executada pelas tarefas, lê as coordenadas de cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceiveNodeCoord`: executada pelas tarefas, recebe as coordenadas dos nós e as armazena nos objetos correspondentes.
- `SendNodeSupport`: executada pelas tarefas, lê as informações do suporte de cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceiveNodeSupport`: executada pelas tarefas, recebe as informações do suporte dos nós e as armazena nos objetos correspondentes.
- `SendNodeConstraint`: executada pelas tarefas, lê as informações das restrições impostas a cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceiveNodeConstraint`: executada pelas tarefas, recebe as informações das restrições impostas aos nós e as armazena nos objetos correspondentes.
- `SendNodeForces`: executada pelas tarefas, lê os valores das forças aplicadas em cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceiveNodeForces`: executada pelas tarefas, recebe os valores das forças aplicadas nos nós e os armazena nos objetos correspondentes.
- `SendPrescDispl`: executada pelas tarefas, lê os deslocamentos impostos em cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceivePrescDispl`: executada pelas tarefas, recebe os valores dos deslocamentos impostos aos nós e os armazena nos objetos correspondentes.
- `SendSkewNode`: executada pelas tarefas, lê as condições de suporte inclinado de cada nó do modelo e envia para a tarefa responsável pelo subdomínio ao qual pertence o nó.
- `ReceiveSkewNode`: executada pelas tarefas, recebe as condições de suporte inclinado dos nós e as armazena nos objetos correspondentes.

4.4.10 Classe `Tdp`

Contém as informações necessárias para a análise estática e dinâmica da estrutura. Nessa classe, foram incluídos contadores de tempo ao longo do código para a monitoração da análise em paralelo. Além disso, a função `Solver` foi adaptada para realizar todo o processo de resolução em paralelo do sistema de equações lineares:

- `Solver`: inicialmente, essa função verifica se a análise é seqüencial ou paralela e, conforme o tipo de análise, são executadas as funções adequadas. Se a análise for do tipo paralela, a seqüência de etapas realizadas por essa função é a seguinte:
 - Montagem da matriz de rigidez do subdomínio
 - Montagem do vetor de esforços
 - Consideração dos deslocamentos impostos
 - Montagem da matriz de rigidez reduzida: utiliza-se a função `MountKsMatrix` da classe `Subdom`.

- Montagem do vetor de esforços reduzido: utiliza-se a função `MountFsVector` da classe `Subdom`.
- Resolução em paralelo do sistema de equações lineares: utiliza-se a função `Parallel_PCG_Solver`, implementada neste trabalho, e que será descrita na seção seguinte.
- Montagem do vetor de deslocamentos: utiliza-se a função `MountFVector` da classe `Subdom`.
- Cálculo das tensões
- Envio dos resultados ao programa mestre: utiliza-se a função `SendPrintStress` da classe `FemMech`.

4.5 Resolução do sistema de equações lineares

Para se obter o sistema de equações lineares, que permite encontrar entre todas as soluções compatíveis aquela que também é equilibrada, utilizou-se a técnica de subestruturação [Nour,87]. Na resolução desse sistema de equações lineares, será utilizado o método dos gradientes conjugados pré-condicionados [Golub,83], adaptado para o ambiente paralelo. Nesta seção, serão apresentados a versão paralela do algoritmo do método dos gradientes conjugados pré-condicionados e detalhes específicos da aplicação da técnica de subestruturação.

Como já visto, ao se utilizar a técnica de subestruturação, a estrutura é dividida em um determinado número de subestruturas, as quais são distribuídas entre os processadores. Para a utilização da versão paralela do algoritmo dos gradientes conjugados pré-condicionados, ao dividir-se a estrutura, deve-se fazer uma partição dos nós de fronteira em nós de fronteira primários e secundários, como mostra a Figura 4.

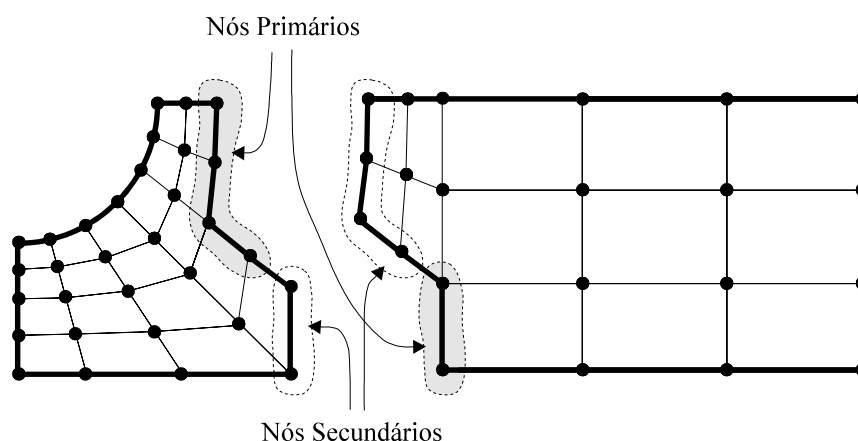


Figura 4: Partição dos nós de fronteira

Nessa partição, cada nó de fronteira é marcado como primário em apenas um dos subdomínios e como secundário nos demais. Como os nós de fronteira pertencem a dois ou mais subdomínios, essa partição é utilizada para evitar-se a duplicação dos componentes dos vetores, associados a esses nós, nas operações de produto escalar. Assim, ao se efetuar um produto escalar entre vetores divididos entre vários processadores, cada processador realiza essa operação apenas sobre os componentes desses vetores associados aos nós de fronteira primários. Os resultados das operações

são então trocados entre os processadores, e, com a soma desses resultados, obtém-se o produto escalar entre os vetores.

Para se aplicar a técnica de subestruturação, foram implementadas funções na classe `Subdom`, que são responsáveis por diversas etapas desse processo. Essas funções já foram descritas anteriormente e são responsáveis pela montagem da matriz de rigidez (`MountKsMatrix`) e do vetor de esforços (`MountFsVector`) associados aos graus de liberdade dos nós de fronteira, e do vetor de ligação \mathbf{L} (`MountLVector`), que contém a correspondência entre a numeração dos graus de liberdade do subdomínio e a numeração da estrutura completa. Ao final da aplicação dessa técnica, obtém-se o sistema de equações lineares a ser resolvido em paralelo.

A versão em paralelo do algoritmo dos gradientes conjugados pré-condicionados foi implementada como uma nova função na biblioteca de funções matemáticas, já presente no FEMOOP, e recebeu o nome de `Parallel_PCG_Solver`. Essa biblioteca possui várias funções para a manipulação de vetores e matrizes, além de dois métodos diretos e seqüenciais de solução de sistemas de equações lineares: um baseado na decomposição de Crout (`CroutSolver`) e outro baseado no método de eliminação de Gauss (`GaussSolver`).

Basicamente, essa nova implementação consiste na paralelização das operações entre matrizes e vetores, sendo que a seqüência das operações permanece a mesma da versão seqüencial.

A Figura 5 apresenta o algoritmo implementado nesse trabalho, e mostra, para cada processador, a seqüência de operações a ser executada. Destacam-se também as etapas que exigem a troca de mensagens entre os processadores, sendo que, a cada iteração do algoritmo, são necessárias quatro operações de troca de mensagens: duas dessas operações envolvendo números de dupla precisão, e duas operações envolvendo vetores de dupla precisão. Nessa figura, N_{peq} é o número de graus de liberdade associados aos nós primários da fronteira do subdomínio e p é o número de processadores.

O pré-condicionador utilizado foi o de balanceamento diagonal (*diagonal scaling*) [Golub,83]. Nesse caso, a matriz de pré-condicionamento possui elementos não-nulos apenas na sua diagonal. Neste trabalho, a matriz de pré-condicionamento foi formada utilizando-se os elementos da diagonal da própria matriz de rigidez da estrutura. Para se obter essa diagonal, é necessária a troca da diagonal das matrizes de rigidez de cada subdomínio entre os processadores. Ao receber uma diagonal de outro subdomínio, o processador adiciona, aos elementos de sua diagonal, os elementos que correspondem aos mesmos graus de liberdade, utilizando-se o vetor de ligação \mathbf{L} , obtendo-se, ao final do processo, a parte da diagonal da matriz de rigidez da estrutura completa correspondente ao seu subdomínio.

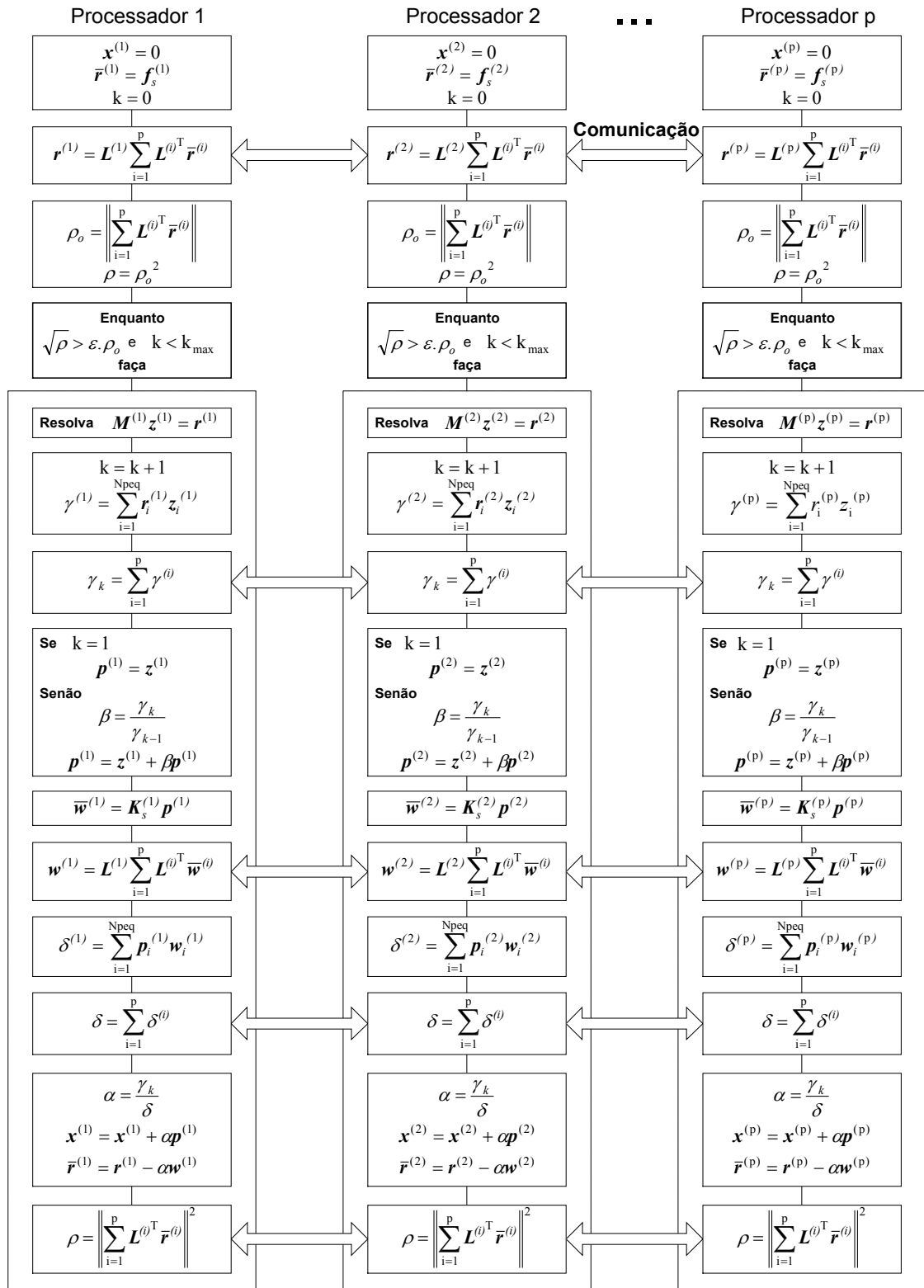


Figura 5: Algoritmo paralelo dos gradientes conjugados pré-condicionados

5. Pós-Processamento

Essa etapa consiste na visualização dos resultados da análise. Para isso, foi utilizado o programa MVIEW (*Bidimensional Mesh View*) [Mview,93], desenvolvido pelo TeCGraf da PUC-Rio.

O MVIEW é um programa gráfico interativo para visualização de resultados de uma análise por elementos finitos. São fornecidas informações qualitativas e quantitativas da malha e dos resultados obtidos. Entre suas funções, destacam-se a visualização da configuração deformada do modelo e a obtenção das isofoixas de resultados escalares para nós e em pontos de Gauss. A Figura 6 mostra a interface gráfica do programa MVIEW.

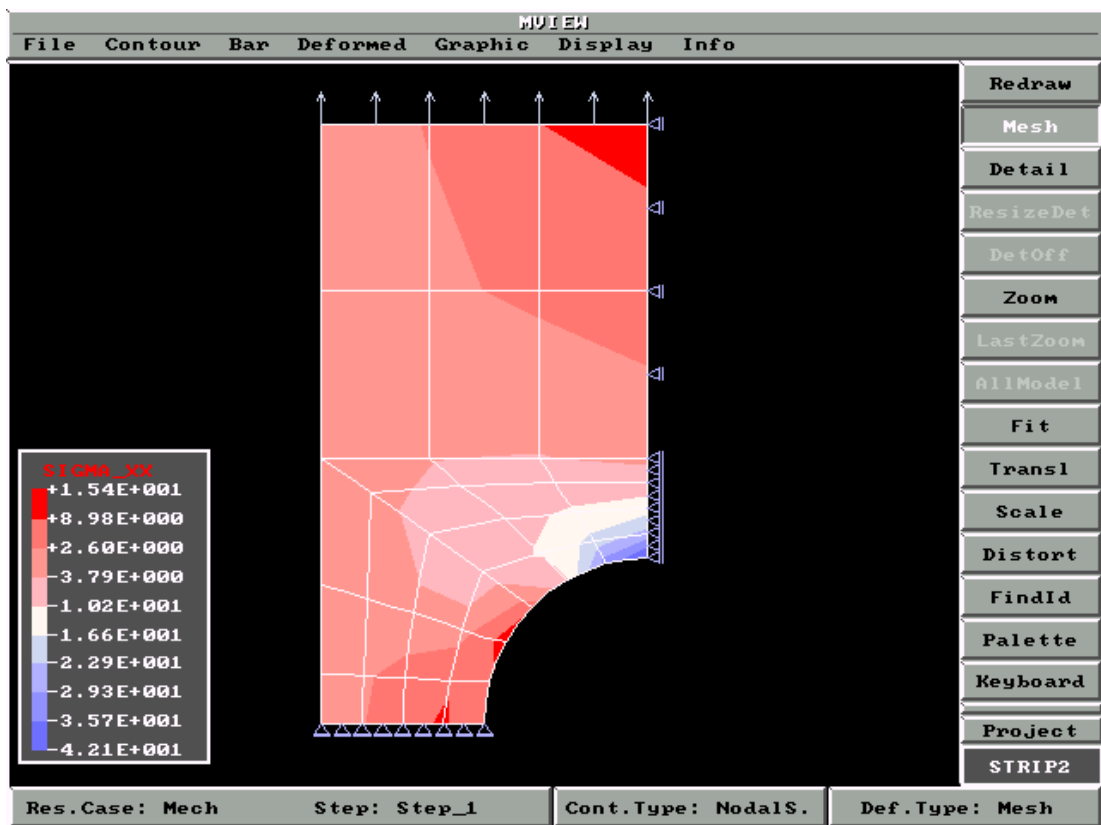


Figura 6: Interface gráfica do programa MVIEW

6. Resumo

Foram apresentados os diversos componentes do sistema paralelo de análise de estruturas, implementado neste trabalho. Foram descritas as etapas de pré-processamento, partição da estrutura, análise e pós-processamento. Maior destaque foi dado às implementações feitas no programa de análise FEMOOP.

Referências Bibliográficas

- [Fujii,97] Fujii, G., “*Análise de Estruturas Tridimensionais: Desenvolvimento de uma Ferramenta Computacional Orientada para Objetos*”, Dissertação de Mestrado, Dep. de Engenharia de Estruturas e Fundações (PEF), Escola Politécnica, USP, 1997.
- [Geist,94] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderman, V., *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.
- [Golub,83] Golub, G.H. e Van Loan, C.F., *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.
- [Guimarães,92] Guimarães, L.G.S., Menezes, I.F.M. and Martha. L.F., “*Object Oriented Programming Discipline for Finite Element Analysis Systems*” (in Portuguese), Proceedings of XIII CILAMCE, Porto Alegre, RS, Brasil, Vol. 1, pp. 342-351, 1992.
- [Martha,96] Martha, L.F., Menezes, I.F.M., Lages, E.N., Parente Jr, E. and Pitangueira, R.L.S., “*An OOP Class Organization for Materially Nonlinear Finite Element Analysis*”, Join Conference of Italian Group of Computational Mechanics and Ibero-Latin American Association of Computational Methods in Engineering, pp. 229-232, University of Padova, Padova, Italy, 1996.
- [Moretti,97] Moretti, C.O., Bittencourt, T.N., André, J.C., e Martha, L.F., “*Algoritmos Automáticos de Partição de Domínio*”, Boletim Técnico BT/PEF, Departamento de Engenharia de Estruturas e Fundações, Escola Politécnica - USP, 1998 (a ser publicado).
- [Mtool,92] “*MTOOL - Bidimensional Mesh Tool (Versão 1.0) - Manual do Usuário*”, Grupo de Tecnologia em Computação Gráfica - TeCGraf / PUC-Rio, 1992.
- [Mview,93] “*MVIEW - Bidimensional Mesh View (Versão 1.1) - Manual do Usuário*”, Grupo de Tecnologia em Computação Gráfica - TeCGraf / PUC-Rio, 1993.
- [Nour,87] Nour-Omid, B., Raefsky, A., e Lyzenga, G., “*Solving Finite Element Equations on Concurrent Computers*”, in A.K. Noor, Ed., *Parallel Computations and Their Impact on Mechanics*, pp. 209-227, ASME, New York, 1987.
- [Weiskamp,90] Weiskamp, K., Flaming, *The Complete C++ Primer*, Academic Press, 1990.