UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Departamento de Engenharia de Estruturas

Implementação de Solução Elemento-por-Elemento em Ambiente Distribuído para o Método dos Elementos Finitos

Dissertação apresentada como requisito parcial

para obtenção do grau de Mestre

em Engenharia de Estruturas

por

Ramon Pereira da Silva

Outubro de 1993

Resumo

Um programa de solução de sistemas de equações baseado no método dos elementos finitos foi implementado em uma máquina paralela virtual usando um conjunto de estações de trabalho ligadas em rede. O método dos gradientes conjugados se mostrou apropriado nesta implementação, e os produtos envolvendo matrizes relativos à determinação do vetor direção de busca foram efetuados em paralelo. Esta abordagem mostrou ser eficiente, viável e promissora.

Abstract

An efficient finite element parallel solver has been implemented on a virtual parallel machine using a cluster of workstations networked. The conjugate gradient method has been shown to be appropriate on the parallel implementation, and the matrices products involved in the derivation of the search direction vector have been carried out on a parallel basis. This approach has been proven to be feasible for this class of problems and promises to be a powerful solver on parallel architecture systems.

Prefácio

Realizar um projeto de vida é uma tarefa árdua, e nos dias de hoje não a faríamos sem a colaboração paciente de algumas pessoas, que são mais que amigos, são parceiros de caminhada a quem dedico este trabalho.

- Estevam e Márcio, pela orientação competente, incentivo, paciência e apoio.
- Minha família, pela grande capacidade de renúncia.
- Meus amigos José Carlos & Ana pela torcida ímpar e adoção involuntária. Paulo e Edilson, pela amizade, apoio e "grande" incentivo.
- Jacira, ao meu lado nos bons e maus momentos. Sempre presente com palavras de afeto e estímulo. Meu agradecimento especial.
- Meus amigos do DEEs, Profs. Ana Maria G. Figueiredo, Elizabeth Vieira Maia e Ricardo Hallal Fakury, pelo apoio e incentivo.
- Profs. Leônidas C. Barroso e Alcebíades de Vasconcellos Filho, primeiros feiticeiros a me confiarem seus segredos.

Eduardo Loures, pelas "conversas paralelas". Aos professores e funcionários do Departamento de Engenharia de Estruturas, pelo apoio e infra-estrutura oferecidos. Ao Departamento de Ciência da Computação, pela utilização dos excelentes recursos computacionais. Ao Prof. Mário F. M. Campos e à Andréa Iabrudi, pela pronta resposta nas questões referentes à rede. À CAPES, pelo apoio financeiro. Ao LNCC/CNPq pelo sistema SDP. A todos agradeço pelo apoio, decisivo, para a realização deste trabalho.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Programas

Capítulo 1

Introdução

1.1 Motivação

Uma grande variedade de problemas físicos são modelados através de equações diferenciais ordinárias ou parciais, que muitas vezes são de difícil solução analítica requerendo então a utilização de métodos numéricos para esta solução.

Entre os métodos numéricos mais populares para a solução de tais equações estão, por exemplo, os métodos de Runge-Kutta, o método das diferenças finitas (MDF), os métodos variacionais e o método dos elementos finitos (MEF) [Carnahan *et al.*, 1969, Zienkiewicz e Morgan, 1983].

O mais conhecido destes métodos, o MDF, devido às dificuldades de generalização, tem sua utilização restrita a uma pequena gama de problemas. Entretanto, com o aparecimento dos computadores digitais, o método dos elementos finitos (MEF) ganhou notoriedade, e sua utilização, primeiramente na área de engenharia de estruturas (cálculo de aeronaves), espalhou-se por várias áreas.

Basicamente, o MEF aplicado a um problema cujo domínio (Ω) é contínuo, divide-o em vários outros sub-domínios (Ω^e), ou *elementos finitos*, cujo comportamento, segundo determinadas hipóteses, é conhecido. Assim, passa-se a analisar o problema em um número finito de pontos pré-selecionados, os chamados *nós*. A cada nó estão associadas várias incógnitas, às quais denominam-se *graus de liberdade*. À substituição do contínuo por um número finito de nós denomina-se *discretização*, e ao conjunto de elementos (que não se superpõem), de *malha de elementos finitos*.

Matematicamente, o problema pode ser descrito [Zienkiewicz e Morgan, 1983, Reddy, 1984, Stasa, 1985, Zienkiewicz e Taylor, 1989] como: encontre uma função u que satisfaça ao sistema de equações diferenciais lineares $\mathcal{L}$, tal que

$$\mathcal{L} = \left\{ \begin{array}{c} \mathcal{L}_1(u) \\ \mathcal{L}_2(u) \\ \vdots \end{array} \right\} + f = 0 \tag{1.1}$$

$u \in \Omega$, satisfazendo as condições de contorno

$$\mathcal{M} = \left\{ \begin{array}{c} \mathcal{M}_1(u) \\ \mathcal{M}_2(u) \\ \vdots \end{array} \right\} + g = 0 \tag{1.2}$$

no contorno Γ.

O MEF busca então aproximar u na forma

$$u \approx \hat{u} = \sum_{i=1}^{n} N_i a_i = Na \tag{1.3}$$

onde N_i são chamadas funções de forma, definidas apenas no domínio dos elementos e convenientemente escolhidas, e a_i parâmetros a serem determinados.

Dois processos distintos bastante utilizados para a obtenção de tal aproximação são: o método dos resíduos ponderados (MRP), e a determinação da formulação variacional do problema, cuja minimização do funcional equivalente é procurada [Stasa, 1985, Reddy, 1986].

O MRP consiste, basicamente, em utilizar uma função peso ω, tal que [Zienkiewicz e Morgan, 1983]

$$\int_\Omega \omega_i R_\Omega d\Omega + \int_\Gamma \bar{\omega}_i R_\Gamma d\Gamma = 0 \tag{1.4}$$

com $i = 1, 2, \ldots, n$; R_Ω e R_Γ são resíduos, respectivamente, no domínio Ω e contorno Γ que são calculados como

$$R_\Omega = \mathcal{L}(\hat{u}) + f \tag{1.5}$$

$$R_\Gamma = \mathcal{M}(\hat{u}) + g \tag{1.6}$$

e ω_i e $\bar{\omega}_i$ sendo as funções peso que, dependendo da escolha, resultam em variantes do MRP, tais como: Ritz, Mínimos Quadrados e Galerkin, entre outros.

Da aplicação do MRP no problema aproximado pelas funções de forma definidas em (1.3), resulta um sistema de equações lineares da forma

$$Ka = f \tag{1.7}$$

onde

$$K_{ij} = \sum_{e=1}^{n} K_{ij}^e \tag{1.8}$$

uma soma booleana no domínio Ω, e

$$f_i = \sum_{e=1}^{n} f_i^e \tag{1.9}$$

Para cada problema, os termos da equação (1.7) têm um significado diferente; em estruturas K é a matriz de rigidez, a o vetor de deslocamentos procurados e f o vetor de forças externas.

Observa-se também que a matriz (1.8) possui as seguintes características:

- simétrica positiva definida
- esparsa
- amplo espectro dos autovalores
- é construída através da contribuição de cada elemento

Mesmo que o operador diferencial seja não-linear, a utilização de métodos de linearização [Bathe, 1982], do tipo Newton-Raphson, conduzem a sistemas de equações lineares que têm de ser resolvidos.

Grande parte do esforço computacional dispendido na aplicação do MEF está concentrado na solução do sistema de equações resultante. Vários são os métodos disponíveis para a solução destes sistemas, que são classificados como diretos ou iterativos [Crisfield, 1986].

Nos métodos diretos a matriz global é invertida ou fatorada, sendo este último procedimento preferível, uma vez que a inversão é por demais dispendiosa, e assim que é obtida a fatoração, vários termos independentes podem ser resolvidos eficientemente, desde que a maior fração do tempo de solução seja gasta na etapa anterior. Por outro lado, na fatoração da matriz K, que é esparsa, perde-se a estrutura de armazenamento, porque no processo de fatoração são introduzidos valores nas posições não-nulas da matriz K, fenômeno este denominado preenchimento (*fill-in*), que pode ser minimizado com a adoção de estratégias de pivotação e utilização de estruturas de dados que exploram a esparsidade [Duff *et al.*, 1986, George e Liu, 1981].

Com a crescente demanda pela solução de sistemas de equações de grande porte, a utilização de métodos diretos pode ser inviabilizada devido ao grande espaço de armazenamento exigido e ao elevado tempo de computação requerido. Surgem então os métodos iterativos como alternativa.

Os métodos iterativos têm a seu favor um baixo requisito de armazenamento, uma vez que a estrutura de armazenamento original da matriz, que é esparsa e possui banda muito larga, é preservada, além de uma boa adequação aos computadores com capacidade de processamento vetorial ou paralelo, bem como aos processos adaptativos [Figueiredo, 1992]. Estes métodos se iniciam com uma estimativa do vetor solução que é refinado após sucessivas etapas, de acordo com as características do método, até que se atinja um grau de precisão pré-estabelecido. O esforço computacional exigido por estes métodos está associado à precisão da solução e ao condicionamento da matriz a ser resolvida. Este último aspecto pode ser melhorado utilizando-se técnicas de precondicionamento (ver, por exemplo, [Golub e van Loan, 1989]) que consistem em modificar convenientemente a matriz a ser resolvida de modo que a matriz transformada fique próxima da matriz identidade.

Um dos métodos iterativos mais explorados atualmente é o método dos Gradientes Conjugados (MGC). Sua popularidade se deve à sua simplicidade de implementação, possuir uma boa taxa de convergência, facilidade de implementação de esquemas de precondicionamento, além de adequação à nova geração de computadores.

Outro ponto favorável ao MGC, neste caso associado às matrizes geradas pela discretização via MEF, é a possibilidade de se utilizar a formulação elemento-por-elemento (EPE) como proposto por Hughes [Hughes *et al.*, 1983a, Hughes *et al.*, 1983b]. Nesta formulação a matriz de rigidez nunca é montada explicitamente, uma vez que cada termo da mesma é obtido pela associação dos termos da matriz de cada elemento da malha de elementos finitos.

O crescente desenvolvimento da tecnologia dos computadores tornou possível o aparecimento de máquinas não só mais rápidas, como também com capacidade de processamento vetorial ou paralelo, permitindo tratar problemas de grande porte. Todavia, o acesso a estas máquinas está restrito a um pequeno número de instituições ou pessoas devido ao seu alto custo. Uma alternativa atraente é a utilização de um conjunto de estações de trabalho (*workstations*) trabalhando simultaneamente. Esta alternativa, em comparação com o uso de supercomputadores, tem a vantagem de possuir uma relação custo/desempenho mais favorável [Loures *et al.*, 1992]. Justifica-se também, pelo fato de que já se tem notícias de implementação do PVM para máquinas IBM RS/6000 e CRAY, entre outras, configurando uma tendência de fortalecimento deste tipo de alternativa.

1.2 Objetivos

Este trabalho tem por objetivo verificar a eficiência de uma implementação paralela do método dos Gradientes Conjugados Precondicionado na solução de sistemas de equações lineares oriundos de discretização via MEF, utilizando-se um conjunto de *workstations* ligadas em rede como máquina paralela. Procura-se também identificar os pontos que limitam a performance do mesmo.

1.3 Contribuições

Pretende-se neste trabalho, demonstrar que a utilização deste tipo de arquitetura na solução de grandes problemas, em particular da engenharia, é competitiva, viável e promissora, uma vez que as estações de trabalho têm-se mostrado como a opção natural em *hardware* adotada não só por instituições de pesquisa, mas também por

empresas privadas, seguindo uma tendência mundial, o chamado *downsizing*[1]

Também, como consequência imediata, ficam a implementação do método de solução e da estratégia elemento-por-elemento, o estudo e utilização do PVM, e a estrutura básica para a implementação de novos precondicionadores, como plataforma para a continuação do uso e pesquisa da computação paralela distribuída nas áreas de elementos finitos, métodos numéricos, e em particular em engenharia de estruturas.

1.4 Organização do Texto

No capítulo 2 as idéias básicas do método dos Gradientes Conjugados são apresentadas. Discute-se também algumas técnicas de precondicionamento.

No capítulo 3 são discutidos alguns conceitos relacionados à computação concorrente, como *pipeline*, computação vetorial, distribuída e paralela.

No capítulo 4 os aspectos de implementação da formulação elemento-por-elemento, formas de paralelização do MGCP, o *software* e o *hardware* utilizados são apresentados e discutidos em detalhe.

Finalmente, o capítulo 5 analisa e discute os exemplos numéricos selecionados para avaliação de vários aspectos relacionados à implementação. Em seguida, no capítulo 6, algumas conclusões obtidas durante a realização deste trabalho são apresentadas, assim como são feitas sugestões para a continuação do mesmo.

[1] *Downsizing* é conhecido como o processo de substituição de computadores de grande porte (ou *mainframes*) por uma rede computadores de menor porte e mais baratos.

Capítulo 2

O Método dos Gradientes Conjugados

Na seção 2.1 os principais métodos iterativos serão brevemente revistos, uma vez que o objetivo principal deste capítulo é o estudo do método dos Gradientes Conjugados. Nas seções seguintes o MGC e o MGC precondicionado são vistos em detalhe ([Golub e van Loan, 1989]).

Em todo o capítulo será considerado um sistema de equações do tipo

$$Ax = b \tag{2.1}$$

onde $A \in \Re^{n \times n}$ é uma matriz simétrica positiva definida, $x, b \in \Re^n$ são o vetor de incógnitas e o vetor de termos independentes, respectivamente, e a matriz A pode ser decomposta na forma definida abaixo:

$$A = L + D + U \tag{2.2}$$

onde

$$L = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 \\ a_{21} & 0 & \dots & & \vdots \\ a_{31} & a_{32} & \ddots & & 0 \\ \vdots & & & 0 & 0 \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 0 \end{bmatrix} \tag{2.3}$$

$$D = \text{diag}(A) = (a_{11}, \dots, a_{nn}) \tag{2.4}$$

$$U = \begin{bmatrix} 0 & a_{12} & \dots & \dots & a_{1n} \\ 0 & 0 & \dots & & \vdots \\ 0 & 0 & \ddots & & a_{n-2,n} \\ \vdots & & & 0 & a_{n-1,n} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \tag{2.5}$$

2.1 Os Métodos Iterativos Clássicos

Os métodos iterativos são preferidos, em geral, na solução de grandes sistemas de equações, na maioria esparsos, que são provenientes da solução numérica de equações diferenciais parciais (EDP). Estes métodos produzem uma sequência de soluções aproximadas (x^k), que dependendo das características individuais de cada método, do condicionamento e tipo da matriz A (simétrica ou não, positiva definida, complexa, etc.) a ser resolvida, convergem mais rapidamente para a solução x desejada, dentro de uma tolerância especificada.

Em geral, nos métodos iterativos a matriz A é posta na forma:

$$A = M - N \tag{2.6}$$

onde M e N são chamadas de matrizes de iteração, e dependendo de como estas são construídas obtêm-se os vários métodos iterativos conhecidos.

2.1.1 Método de Jacobi

Este método é talvez o mais conhecido. Nele, as matrizes de iteração são dadas, respectivamente por, $M = M_J = D$ e $N = N_J = -(L + U)$, ou seja,

$$M_J x^{k+1} = N_J x^k + b \tag{2.7}$$

Assim, uma iteração típica de Jacobi é da forma:

repita para $i = 1$ **até** n
$x_i^{k+1} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k - \sum_{j=i+1}^{n} a_{ij} x_j^k\right) / a_{ii}$
fim repita

Uma vez que os valores de x^k são independentes dos valores de x^{k+1}, o método de Jacobi pode ser facilmente paralelizável. Uma desvantagem deste método é que ele frequentemente diverge, necessitando-se que a matriz A possua os autovalores dentro da faixa de $0 < \lambda < 2$ para que se garanta a convergência [Crisfield, 1986].

2.1.2 Método de Gauss-Seidel

O método de Gauss-Seidel pode ser visto como uma variante do método de Jacobi, se utilizarmos os valores das componentes de x^k calculadas previamente no cálculo das próximas componentes de x^k, i.e., x_j^k (com $j = 1, \ldots, i-1$) pode ser utilizado no cálculo de x_j^k (com $j = i+1, \ldots, n$), isto é conseguido fazendo-se as matrizes de iteração $M = M_G = (D + L)$ e $N = N_G = -U$, ou seja,

$$M_G x^{k+1} = N_G x^k + b \tag{2.8}$$

que pode ser descrito como abaixo:

repita para $i = 1$ **até** n

$x_i^{k+1} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k\right) / a_{ii}$

fim repita

Contrariamente ao método de Jacobi, a paralelização (ao nível de operação vetorial) do método de Gauss-Seidel não é facilmente conseguida, uma vez que existe uma dependência entre as componentes do vetor x^k.

Demonstra-se [Golub e van Loan, 1989], página 510, que se o raio espectral $\rho(M_G^{-1} N_G)$ for próximo da unidade, então a convergência do método de Gauss-Seidel pode ser bastante demorada.

2.1.3 Método das Sobre-Relaxações Sucessivas (SOR)

O método de Gauss-Seidel pode ser modificado introduzindo-se um parâmetro $\omega \in \Re$ tal que $M = M_\omega = (D + \omega L)$ e $N = N_\omega = (1 - \omega)D - \omega U$, ou seja

$$M_\omega x^{k+1} = N_\omega x^k + b \tag{2.9}$$

que pode ser descrito como abaixo:

repita para $i = 1$ **até** n

$x_i^{k+1} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k\right) / a_{ii} + (1 - \omega) x_i^k$

fim repita

Para algumas classes de problemas, o parâmetro ω que pode minimizar $\rho(M_\omega^{-1} N_\omega)$ é conhecido; entretanto, encontrar o valor ótimo de ω, de uma maneira geral, é uma tarefa árdua, requerendo por vezes o conhecimento dos autovalores de A.

Com relação à paralelização, valem as mesmas observações feitas na seção 2.1.2.

2.2 O Método do Gradiente

Pode-se demonstrar [Golub e van Loan, 1989], que resolver o sistema (2.1) equivale à minimização do funcional quadrático $\Phi(x)$ definido por:

$$\Phi(x) = \frac{1}{2} x^T A x - x^T b \tag{2.10}$$

ou

$$\min\{\Phi(x)\} = \frac{\partial \Phi(x)}{\partial x} = 0 \tag{2.11}$$

que resulta em $x = A^{-1} b$.

O método do gradiente (*steepest descent*) pode ser empregado para obter o mínimo de $\Phi(x)$. Assumindo que uma aproximação x^k tenha sido obtida, sabe-se que para um ponto $x^c = x^k$ a função Φ decresce mais rapidamente na direção negativa do gradiente de $\Phi(x^c)$, i.e., $-\nabla\Phi(x^k) = b - Ax^k$ que é igual ao resíduo

$$r^k = b - Ax^k. \tag{2.12}$$

Uma vez que existe uma direção conhecida na qual o resíduo (2.12) decresce, é razoável esperar que:

$$x^{k+1} = x^k + \alpha^k r^k \tag{2.13}$$

onde $\alpha \in \Re$ indica o quanto se "caminha" na direção de r^k na busca de uma nova aproximação para x, tal que $\Phi(x^{k+1}) = \Phi(x^k)$. Esta condição pode ser atendida impondo-se que:

$$\frac{d\Phi(x^k + \alpha^k r^k)}{d\alpha^k} = 0 \tag{2.14}$$

que resulta em:

$$\alpha^k = \frac{\langle r^k, r^k \rangle}{\langle r^k, Ar^k \rangle} \tag{2.15}$$

onde $\langle ., . \rangle$ denota um produto escalar em $\Re$, e o algoritmo é o descrito abaixo:

início
  $k = 0; x^0 = 0; r^0 = b$
  enquanto $r^k \neq 0$ **faça**
    $k = k + 1$
    $\alpha^k = \langle r^{k-1}, r^{k-1} \rangle / \langle r^{k-1}, Ar^{k-1} \rangle$
    $x^k = x^{k-1} + \alpha^k r^{k-1}$
    $r^k = b - Ax^k$
  fim enquanto
$x = x^k$
fim

Programa 2.1: Algoritmo do Método do Gradiente

Mostra-se [Golub e van Loan, 1989] que a desigualdade abaixo é válida para o método do gradiente:

$$\Phi(x^k) + \frac{1}{2} b^T A^{-1} b \leq \left(1 - \frac{1}{\kappa_2(A)}\right) \left(\Phi(x^{k-1}) + \frac{1}{2} b^T A^{-1} b\right) \tag{2.16}$$

Portanto, o método pode ter uma taxa de convergência muito baixa se $\kappa_2 = \lambda_n(A)/\lambda_1(A)$ for grande, caracterizando A como uma matriz mal-condicionada.[1]

[1] κ_2 é definido como o número de condição de A, λ_1 e λ_n são respectivamente, o menor e o maior autovalores de A.

Uma das características deste método é que $r^k \perp r^{k+1}$, ou mesmo que $r^{k-1} \perp r^k$, todavia, não se pode dizer que $r^{k-1} \perp r^{k+1}$, i.e., pode-se repetir uma direção previamente utilizada, podendo-se obter uma taxa de convergência extremamente baixa.

2.3 O Método dos Gradientes Conjugados

No método dos Gradientes Conjugados se admite uma direção de busca do tipo

$$p^k = r^{k-1} + \beta^k p^{k-1} \tag{2.17}$$

na qual $p^0 = r^0$, sendo que β^k é obtido impondo-se que os vetores p^k sejam mutuamente ortogonais a $Ap^{(1)}, \ldots, Ap^{k-1}$ o que resulta em

$$\alpha^k = \frac{\langle r^{k-1}, r^{k-1} \rangle}{\langle p^k, Ap^k \rangle} \tag{2.18}$$

e

$$\beta^k = \frac{\langle r^{k-1}, r^{k-1} \rangle}{\langle r^{k-2}, r^{k-2} \rangle} \tag{2.19}$$

Uma vez que o resíduo pode ser calculado recursivamente através de $r^k = r^{k-1} - \alpha^k Ap^k$, substitui-se esta expressão em (2.19) e obtem-se o algoritmo básico do método dos Gradientes Conjugados, mostrado abaixo:

início
  $k = 0; x^0 = 0; r^0 = b$
  enquanto $r^k \neq 0$ **faça**
    $k = k + 1$
    se $k = 1$
      $p^1 = r^0$
    senão
      $\beta^k = \langle r^{k-1}, r^{k-1} \rangle / \langle r^{k-2}, r^{k-2} \rangle$
      $p^k = r^{k-1} + \beta^k p^{k-1}$
    fim se
    $\alpha^k = \langle r^{k-1}, r^{k-1} \rangle / \langle p^k, Ap^k \rangle$
    $x^k = x^{k-1} + \alpha^k p^k$
    $r^k = r^{k-1} - \alpha^k Ap^k$
  fim enquanto
$x = x^k$
fim

Programa 2.2: Algoritmo de Gradientes Conjugados

Uma das características mais fortes do MGC é que, teoricamente, em aritmética infinita, ele converge para a solução exata em n iterações. Entretanto, aproximações aceitáveis da solução podem ser obtidas em até m iterações ($m \leq n$). Já em arimética finita, a condição de terminação finita já não é mais verdadeira devido aos erros de truncamento computacional. Para o vetor x^k, obtido na k-ésima iteração do MGC, a seguinte relação é válida [Golub e van Loan, 1989, Mesquita, 1992]:

$$\frac{\|x - x^k\|_A}{\|x - x^0\|_A} \leq 2\left(\frac{\sqrt{\kappa_2} - 1}{\sqrt{\kappa_2} + 1}\right)^k \tag{2.20}$$

onde $\|.\|_A$ é uma norma baseada na matriz A, definida como: $\|w\|_A = \sqrt{w^T A w}$.

Da expressão (2.20) pode-se mostrar [Mesquita, 1992], que para κ_2 grande, e para que

$$\frac{\|x - x^k\|_A^2}{\|x - x^0\|_A^2} \leq \epsilon \tag{2.21}$$

com ϵ sendo a tolerância desejada, vale a seguinte expressão:

$$k \geq \frac{\sqrt{\kappa_2}}{2} \ln(2/\epsilon) + 1 \tag{2.22}$$

demonstrando que o número de iterações é proporcional a raiz quadrada do número de condição de A, ou que o MGC converge rapidamente quando $\kappa_2(A) \approx 1$. Infelizmente, conforme já citado anteriormente, as matrizes geradas através da discretização via MEF, em geral não possuem esta característica.

Do ponto de vista computacional, duas operações do MGC sobressaem sobre as demais, a saber: os produtos escalares e o produto matriz-vetor Ap. Como a matriz K é construída elemento-por-elemento, pode-se tirar proveito desta característica implementando-se estas duas operações de forma a explorar um computador com capacidade de processamento vetorial ou paralelo [Carey e Jiang, 1986, Coutinho *et al.*, 1986, Hughes e Ferencz, 1987, Coutinho *et al.*, 1988, Barragy e Carey, 1988, Johnsson e Mathur, 1989, Coutinho *et al.*, 1991].

2.4 O Método dos Gradientes Conjugados Precondicionado

A fim de reduzir o número de condição de A é possível então lançar mão da técnica de precondicionamento. Esta técnica consiste em transformar (2.1) no sistema equivalente

$$\tilde{A}\tilde{x} = \tilde{b} \tag{2.23}$$

onde $\tilde{A} = C^{-1}AC^{-1}, \tilde{x} = Cx, \tilde{b} = C^{-1}b$, sendo C simétrica positiva definida. Aplicando o algoritmo 2.2 ao sistema (2.23) e definindo $\tilde{p}^k = Cp^k, \tilde{r}^k = C^{-1}r^k$, variáveis auxiliares, $M \in \Re^{n\times n}$, matriz precondicionadora simétrica positiva definida, $M = C^2$, e fazendo z^k a solução do sistema auxiliar $Mz^k = r^k$, resulta no algoritmo descrito abaixo:

início
$k = 0; x^0 = 0; r^0 = b$
construa M
enquanto $r^k \neq 0$ **faça**
resolva $Mz^k = r^k$
$k = k + 1$
se $k = 1$
$p^1 = z^0$
senão
$\beta^k = \langle r^{k-1}, z^{k-1}\rangle / \langle r^{k-2}, z^{k-2}\rangle$
$p^k = z^{k-1} + \beta^k p^{k-1}$
fim se
$\alpha^k = \langle r^{k-1}, z^{k-1}\rangle / \langle p^k, Ap^k\rangle$
$x^k = x^{k-1} + \alpha^k p^k$
$r^k = r^{k-1} - \alpha^k Ap^k$
fim enquanto
$x = x^k$
fim

Programa 2.3: Algoritmo de Gradientes Conjugados Precondicionado

Ressalta-se que o precondicionamento ideal é aquele que atende às seguintes condições:

- $\kappa_2(C^{-1}AC^{-1}) \ll \kappa_2(A)$
- a solução do sistema auxiliar $Mz = r$ seja eficiente e rápida
- M possua baixo requisito de armazenamento

portanto, o sucesso do algoritmo descrito em 2.3 depende da escolha adequada do precondicionador M.

2.5 Formas de Precondicionamento

A escolha do precondicionador tem efeito decisivo na taxa de convergência do MGCP. De uma maneira geral, pode-se dividir os precondicionadores em duas categorias: os de propósito geral e aqueles baseados na estratégia elemento-por-elemento (EPE) proposta por Hughes [Hughes *et al.*, 1983a, Hughes *et al.*, 1983b].

Este trabalho se limita a apresentar os precondicionadores clássicos, uma vez que não é seu objetivo o estudo dos mesmos. Os trabalhos de Muller [Muller e Hughes, 1986], Hughes [Hughes e Ferencz, 1987], Arruda [Arruda *et al.*, 1990] e Alquati [Alquati e Groehs, 1991, Alquati, 1991], estudam os precondicionadores EPE de uma forma mais crítica.

2.5.1 Precondicionadores de Propósito Geral

Os precondicionadores Diagonal e Incompleto de Cholesky são, de uma maneira geral, os mais importantes e conhecidos dentre os utilizados na solução de sistemas de equações do tipo dado em (2.1), e são suscintamente descritos aqui. Outros precondicionadores como o polinomial, o de fatoração incompleta de Cholesky por blocos (adequados para matrizes tri- ou pentadiagonais) e os baseados em decomposição de domínio têm uma descrição em [Golub e van Loan, 1989].

Precondicionador de Jacobi ou Diagonal

Define-se uma matriz de precondicionamento como:

$$M = \text{diag}(A) = D = D^{1/2} D^{1/2} \tag{2.24}$$

Por sua simplicidade de implementação, baixo requisito de armazenamento, obtenção simples e ter sua eficiência comprovada na solução de problemas de estruturas (ver [Alquati e Groehs, 1991]), foi adotado neste trabalho. Sua implementação computacional é efetuada como descrito abaixo:

$$D^{-1/2} A D^{-1/2} D^{1/2} x = D^{-1/2} b \tag{2.25}$$

resultando em um sistema na forma dada por (2.23). Neste caso, o sistema (2.25) é resolvido pelo algortimo 2.2, e ao final das iterações, o sistema original fica reestabelecido por:

$$x = D^{-1/2} x \tag{2.26}$$

Precondicionador Incompleto de Cholesky

A idéia por trás deste precondicionador é efetuar a fatoração incompleta de Cholesky da matriz A de modo a minimizar a quantidade de *fill-in* que pode ocorrer,

que é de difícil previsão. Uma boa estratégia é aproveitar a esparsidade da matriz A e utilizar a estrutura de dados a ela associada. Desta forma, é permitido somente o cálculo e armazenamento dos termos de L e L^T (fatores triangulares inferior e superior de Cholesky) quando $A_{ij} \neq 0$. Esta estratégia é conhecida como *fill-in* nível 0, $ILU(0)$.

2.5.2 Precondicionadores Elemento-por-Elemento

Como os precondicionadores EPE se referem à matriz de rigidez, a matriz A até agora utilizada será substituída por K.

Precondicionador EPE de Cholesky

Este precondicionador é definido como:

$$M = D^{1/2} \times \prod_{e=1}^{\mathtt{nel}} L_P \times \prod_{e=\mathtt{nel}}^{1} L_P^T \times D^{1/2} \tag{2.27}$$

onde se identificam os seguintes termos:

$$\bar{K}^e = L_P L_P^T \tag{2.28}$$

$$\bar{K}^e = I + D^{-1/2}(K^e - D^e)D^{-1/2} \tag{2.29}$$

$$D = \mathrm{diag}(K) \tag{2.30}$$

$$D^e = \mathrm{diag}(K^e) \tag{2.31}$$

sendo K^e a matriz do e-ésimo elemento, L_P e L_P^T as matrizes triangulares inferior e superior oriundas da fatoração de K^e pelo método de Cholesky, e `nel` o número de elementos da malha de elementos finitos. Embora $\bar{K}^e$ tenha a dimensão de K, as operações sobre $\bar{K}^e$ se dão apenas no conjunto de equações associadas ao elemento considerado.

Precondicionador EPE de Crout

Este precondicionador é definido similarmente ao de Cholesky, diferindo apenas no método utilizado para a fatoração de K^e que neste caso utiliza o método de Crout. Dessa forma este precondicionador fica assim definido:

$$M = D^{1/2} \times \prod_{e=1}^{\mathtt{nel}} L_P \times \prod_{e=1}^{\mathtt{nel}} D_P \times \prod_{e=\mathtt{nel}}^{1} L_P^T \times D^{1/2} \tag{2.32}$$

Os termos da equação (2.32) têm o mesmo significado do caso anterior, ressaltando-se que a fatoração de Crout impõe que:

$$\bar{K}^e = L_P D_P L_P^T. \tag{2.33}$$

Precondicionador EPE de Gauss-Seidel

Diferentemente dos precondicionadores citados anteriormente, a fatoração de K^e é baseada em decomposição por soma, da seguinte forma:

$$\bar{K}^e = I + L_S + L_S^T \tag{2.34}$$

que resulta no seguinte precondicionador:

$$M = D^{1/2} \times \prod_{e=1}^{\mathtt{nel}} (I + L_S) \times \prod_{e=\mathtt{nel}}^{1} (I + L_S^T) \times D^{1/2} \tag{2.35}$$

Implementação dos Precondicionadores EPE

O cálculo da matriz precondicionadora M elemento-por-elemento é feito em duas etapas. A primeira etapa consiste no cálculo da matriz $\bar{K}^e$ para cada elemento, de acordo com a expressão (2.29). A segunda etapa consiste na decomposição das matrizes K^e de cada elemento por Cholesky (2.27), Crout (2.32) ou Gauss-Seidel (2.35).

A solução do sistema auxiliar $Mz = r$ é da mesma forma para cada tipo de precondicionador EPE. Para exemplificar, suponha-se uma malha de dois elementos, cujo precondicionador adotado seja o EPE de Cholesky. Assim, de acordo com a expressão (2.27) a matriz M seria dada por [Crisfield, 1986]:

$$M = D^{1/2}(L_1 L_2)(L_2^T L_1^T) D^{1/2} \tag{2.36}$$

onde D é a matriz diagonal global desta malha e $L_1, L_2, L_1^T \mathrm{e} L_2^T$ são as matrizes triangulares inferior e superior oriundas da fatoração por Cholesky, correspondentes aos elementos 1 e 2 da referida malha. Assim, a cada iteração são realizadas as seguintes operações:

$$\begin{aligned}
D^{1/2} L_1 L_2 L_2^T L_1^T D^{1/2} z &= r && (2.37)\\
L_1 L_2 L_2^T L_1^T D^{1/2} z &= D^{-1/2} r = v_1 && (2.38)\\
L_2 L_2^T L_1^T D^{1/2} z &= L_1^{-1} v_1 = v_2 && (2.39)\\
L_2^T L_1^T D^{1/2} z &= L_2^{-1} v_2 = v_3 && (2.40)\\
L_1^T D^{1/2} z &= L_1^{-T} v_3 = v_4 && (2.41)\\
D^{1/2} z &= L_2^{-T} v_4 = v_5 && (2.42)\\
z &= D^{-1/2} v_5 && (2.43)
\end{aligned}$$

Nota-se três tipos de operações: escalonamento diagonal (2.38) e (2.43), substituições à frente, (2.39) e (2.40), e retrosubstituições (2.41) e (2.42).

Alquati [Alquati, 1991] implementou os precondicionadores Diagonal e os EPE de Cholesky, Crout e Gauss-Seidel, comparando o desempenho destes na solução de sistemas de equações oriundos de várias discretizações, via método dos elementos finitos, de uma gama de problemas de elasticidade linear, afirmando que o MGC sem precondicionamento apresentou desempenho satisfatório para problemas bem condicionados (problemas de estado plano de tensão e deformação, axissimétrico, sólido 3D, e placas com teoria de Midlin), e que a técnica de precondicionamento melhorou bastante a taxa de convergência (adotada no trabalho citado como sendo a razão entre o número de graus de liberdade e o número de iterações) na solução dos problemas mal condicionados (placas com elementos DKQ e cascas poliédricas).

De uma maneira geral, as melhores taxas de convergência foram alcançadas utilizando-se o precondicionador EPE de Cholesky.

Em relação ao tempo para atingir a convergência, o precondicionador de Jacobi, no geral, foi mais rápido, e em relação aos precondicionadores EPE, a escolha entre Cholesky ou Crout não resultou em alteração significativa, com ligeira vantagem para o Gauss-Seidel, embora seja menos eficiente em termos de melhora na taxa de convergência.

Capítulo 3

Processamento Paralelo

3.1 Introdução

A necessidade crescente de se resolver problemas cada vez mais complexos (ou maiores) implica na utilização de computadores de alto desempenho, os chamados supercomputadores. Historicamente, um supercomputador é conhecido como o computador mais potente utilizado em uma determinada época, uma vez que o desenvolvimento da tecnologia na área de *hardware* permite um significativo aumento na velocidade dos componentes. Todavia, existe uma limitação física imposta pela velocidade da luz que implica na utilização de novos recursos, como o processamento concorrente ou paralelo, para a obtenção de melhor performance.

De uma maneira genérica, processamento paralelo significa a utilizaçao de vários computadores (iguais ou não) trabalhando simultaneamente para atingir o mesmo objetivo, seja a nível de programas, seja a nível de instruções. A primeira situação configura-se como um caso de processamento distribuído e a última situação é conhecida como *pipeline*, que será visto mais adiante [Almeida e Árabe, 1991]. Outros níveis de paralelismo podem ser conseguidos com a execução de tarefas e rotinas e com a execução de *loops* em paralelo, o chamado processamento vetorial. Entretanto, quanto menor for o objeto (dados, sequências de instruções como os laços de iteração ou rotinas), maiores são as dificuldades em obter-se sincronização dos objetos atomizados.

Dentro deste contexto, os supercomputadores podem ser classificados em quatro grupos, embora computadores híbridos sejam frequentemente encontrados:

- SISD (*Single Instruction* x *Single Data*): este é o computador tradicional, o qual executa uma sequência de instruções em um dado por vez.
- SIMD (*Single Instruction* x *Multiple Data*): nesta classe de computadores todos os processadores executam a mesma sequência de instruções sobre vários

conjuntos de dados. Os computadores com capacidade de processamento vetorial se enquadram nesta classe.

- MISD (*Multiple Instruction* x *Single Data*): esta classe de máquina apenas existe conceitualmente e executa múltiplas instruções sobre um mesmo conjunto de dados.

- MIMD (*Multiple Instruction* x *Multiple Data*): este tipo de máquina pode ser encarado como o tipo mais geral de computador, no qual vários processadores executam múltiplas instruções sobre vários conjuntos de dados. Estes processadores podem atuar desacoplados ou não e são controlados por um único sistema operacional. Os processadores podem ser interconectados de diversas maneiras, podem ter memória compartilhada ou local (distribuída), o que será objeto de atenção mais à frente.

3.2 Recursos para Obtenção do Paralelismo

Dentre os recursos mais utilizados para aumentar a performance dos computadores estão, por exemplo, o processamento em *pipeline*, os processadores vetoriais e os multiprocessadores, que são suscintamente descritos abaixo:

3.2.1 Múltiplas Unidades Funcionais

Em essência um computador possui três componentes: memória principal, uma unidade central de processamento (UCP) e o subsistema de entrada e saída (E/S). A UCP consiste de um conjunto de registradores, um contador de instruções e uma unidade lógico-aritmética (ULA) na qual as operações, como multiplicações e comparações, são executadas uma a cada vez.

Uma das formas de obtenção de paralelismo é desmembrar a ULA em unidades funcionais que correspondam à operações como adição e multiplicação, com estas trabalhando simultaneamente. Logicamente haveria um *overhead* associado às operações de sincronização destas unidades que poderia ser minimizado com simplificações ou melhorias na forma de interconxeção e fluxo de dados entre as unidades funcionais.

3.2.2 Pipeline

A técnica de *pipeline* consiste em dividir uma tarefa em uma sequência de pequenas subtarefas, cada qual sendo executada concorrentemente em uma parte do *hardware*, na qual o resultado (ou saída) do processamento de uma subtarefa é a

entrada para o próximo estágio da computação. A contribuição da técnica de *pipeline* advém da capacidade de se iniciar novas tarefas antes do término daquelas que se encontram em execução.

Uma classificação para processamento em *pipeline* é encontrada em [Almeida e Árabe, 1991], qual seja:

- *pipeline* aritmético, onde as unidades lógico-aritméticas de um processador são segmentadas para a operação *pipeline*, i.e., numa operação como a adição, por exemplo, são identificadas as várias etapas que a constituem de modo a permitir sua execução em *pipeline*.

- *pipeline* de instruções, no qual a execução de uma sequência de instruções pode ser feita de modo a permitir uma superposição das fases que a constituem.

- *pipeline* de processadores, onde vários processadores dispostos em linha executam uma tarefa específica do processamento sobre uma sequência de dados.

Para aumentar o desempenho dos computadores várias arquiteturas incorporam os vários níveis de *pipeline* vistos anteriormente.

3.2.3 Processamento Vetorial

Os computadores vetoriais incorporam ao seu conjunto de instruções as *instruções vetoriais*, que consistem em executar somente uma operação por vez, em *pipeline*, sobre vários elementos de um ou mais vetores em um local à parte, ou o processador vetorial. Assim, a multiplicação de dois vetores a e b, de dimensão n, em um processador vetorial consiste, basicamente, na leitura dos vetores a e b para a memória local do processador vetorial, na execução da operação de multiplicação e no retorno do resultado em um outro vetor c, eliminando a busca e decodificação na memória da mesma operação, o que resulta em grande desempenho. Este é o típico exemplo de máquina do tipo SIMD. Ver [Almeida e Árabe, 1991, Dongarra *et al.*, 1991] para maiores detalhes.

3.2.4 Multiprocessadores

Os multiprocessadores são máquinas tipo MIMD compostas de vários processadores independentes e de um conjunto de memórias ligadas através de uma rede de interconexão. Os multiprocessadores podem ser divididos em sistemas fortemente e fracamente acoplados. Nos primeiros, também conhecidos como multiprocessadores, os diversos processadores possuem uma memória comum que pode ser compartilhada entre os mesmos. Já os sistemas fracamente acoplados, ou multicomputadores,

possuem a sua própria memória local. Nas duas categorias citadas a interação entre os processadores se dá através da troca de mensagens.

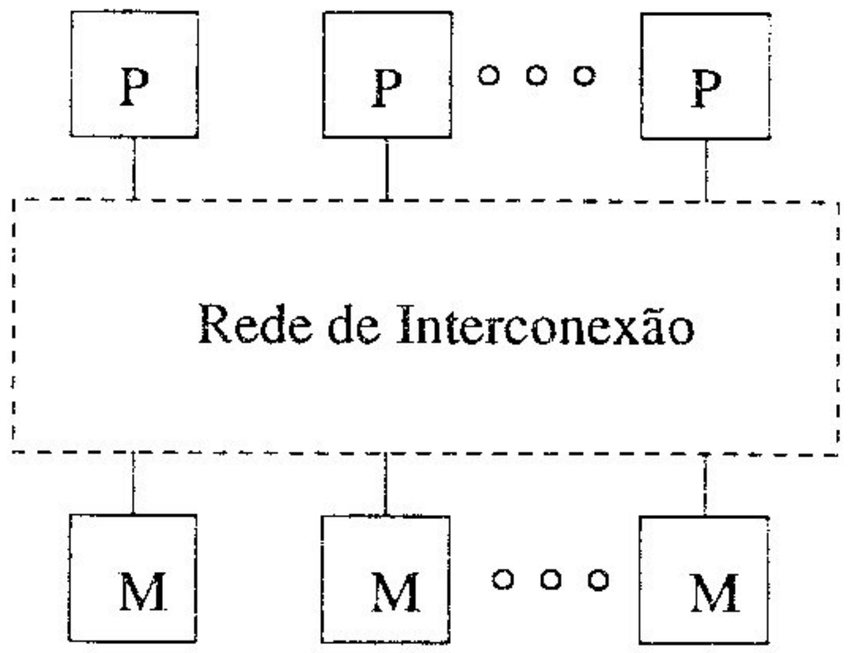


Figura 3.1: Arquitetura Geral dos Multiprocessadores

Dependendo de como é feita a interconexão entre processadores e memórias, surgem as diversas arquiteturas, que são brevemente descritas a seguir:

- barramento (*bus*): esta é a forma mais simples de conexão entre processadores, memórias e dispositivos de E/S. Através do barramento cada unidade funcional tem acesso aos dados na memória, um por vez, i.e., dois processadores não podem acessar (ler ou gravar) dados em um mesmo instante, o que somado ao reduzido número de conexões possíveis reduz bastante o desempenho deste tipo de arquitetura.

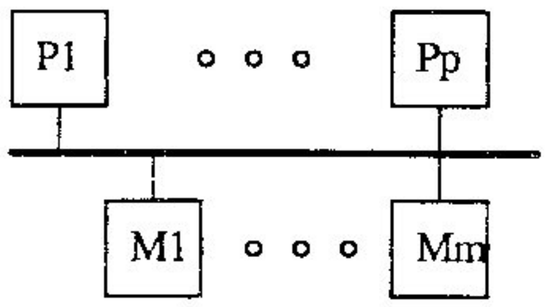


Figura 3.2: Arquitetura de Barramento

- barramentos cruzados (*crossbar*): este sistema permite total conectividade entre processadores e os módulos de memória. Todavia, se existem m memórias e p processadores, deverão existir mp dispositivos de chaveamento, o que configura uma desvantagem deste sistema.

- rede matricial: esta arquitetura forma uma malha bidimensional, onde cada processador está conectado a seus 4 vizinhos, i.e., à esquerda, direita, acima e abaixo. Uma variante desta topologia, é fazer a conexão dos processadores da primeira linha àqueles da última linha, e os processadores da primeira coluna àqueles da última coluna, formando um toróide. Um caso particular é o arranjo linear, i.e., os processadores ligados em linha ou coluna.

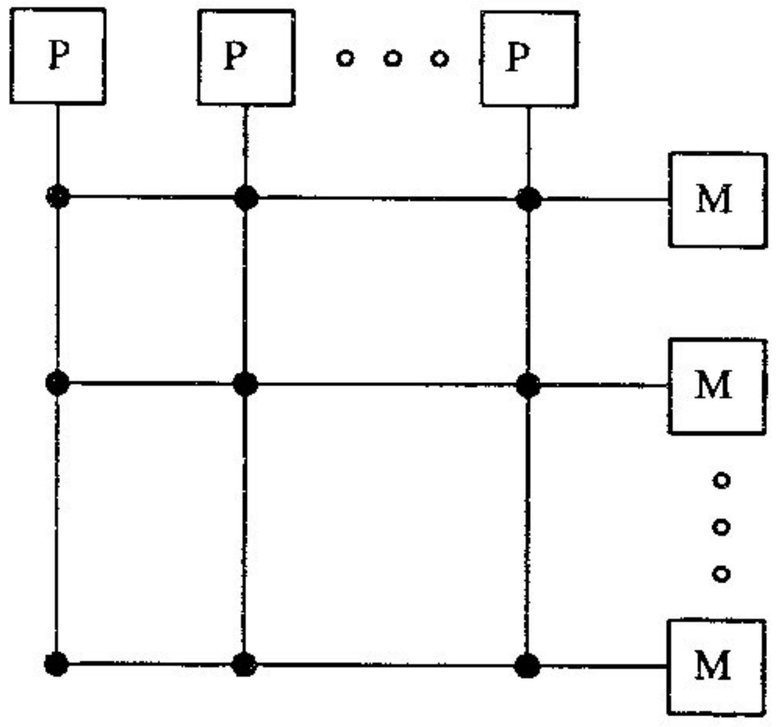


Figura 3.3: Arquitetura de Barramentos Cruzados

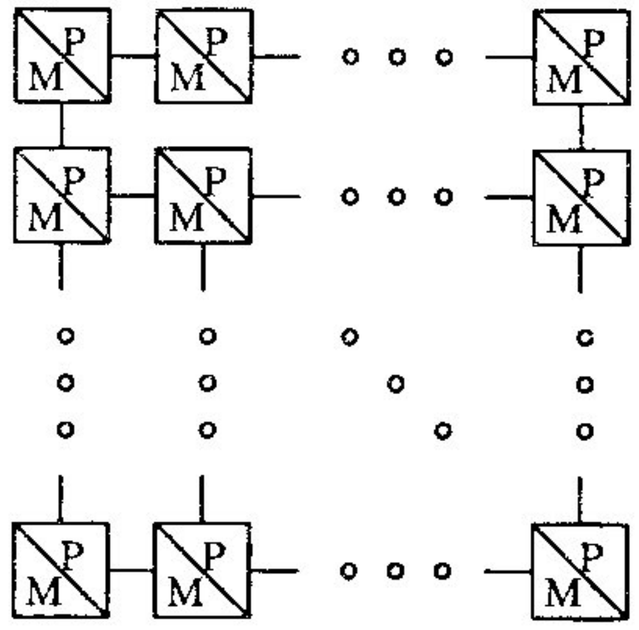
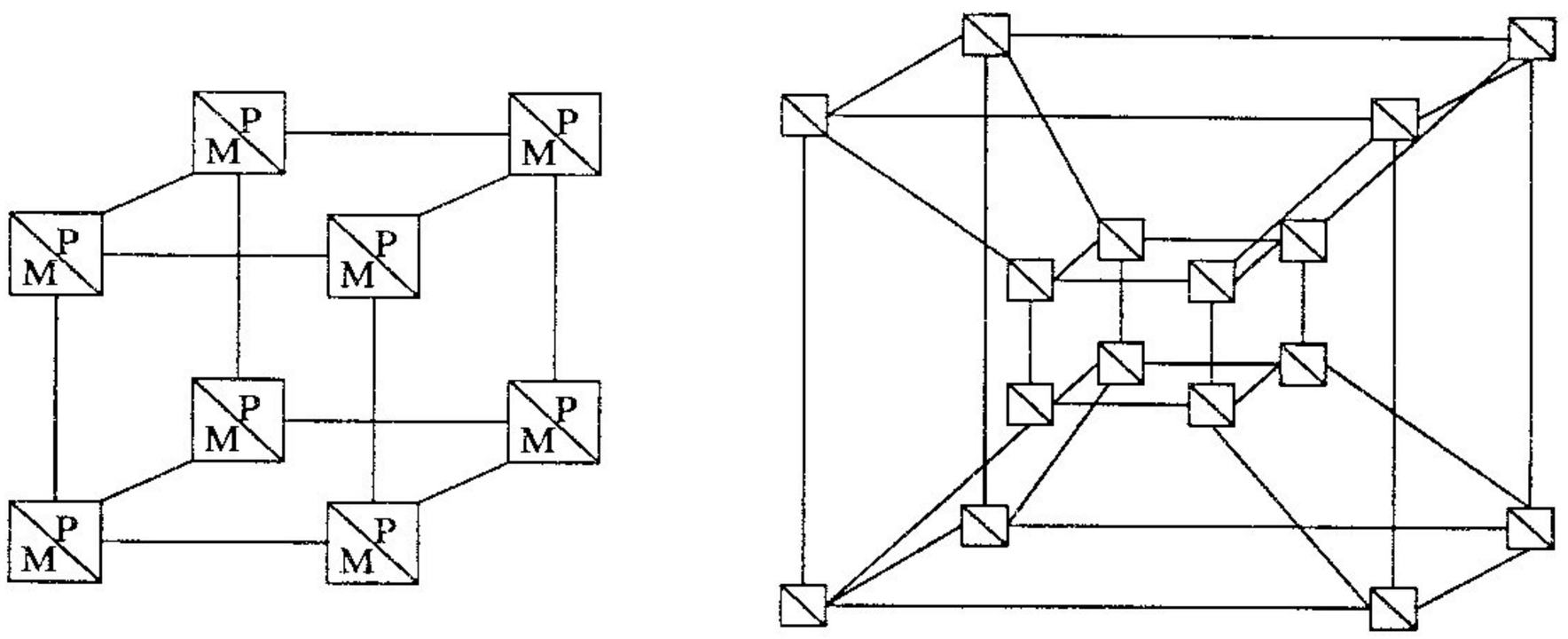


Figura 3.4: Sistema de Rede Matricial

- hipercubos: neste sistema cada processador está ligado a n outros processadores ou nós através de um canal de comunicação ponto-a-ponto, i.e., as conexões são definidas pelos possíveis arcos que possam ser traçados entre quaisquer dois nós cuja numeração difira de um bit na sua representação binária, resultando em $N = 2^n$ nós. O número n é conhecido como a dimensão do hipercubo. O hipercubo se caracteriza por se enquadrar na classificação MIMD das máquinas multiprocessadoras fracamente acopladas, possuir memória distribuída e comunicação entre processadores via troca de mensagens. A figura 3.5 ilustra hipercubos com dimensões 3 e 4.

- redes de interconexão multiestágio: esta arquitetura permite a conexão de um processador arbitrário a um módulo de memória também arbitrário, conforme ilustra a figura 3.6 que tipifica uma rede denominada *Omega*, como mencionado por [Almeida e Árabe, 1991].



a) Hipercubo de dimensão 3

b) Hipercubo de dimensão 4

Figura 3.5: Hipercubo de 3 e 4 Dimensões

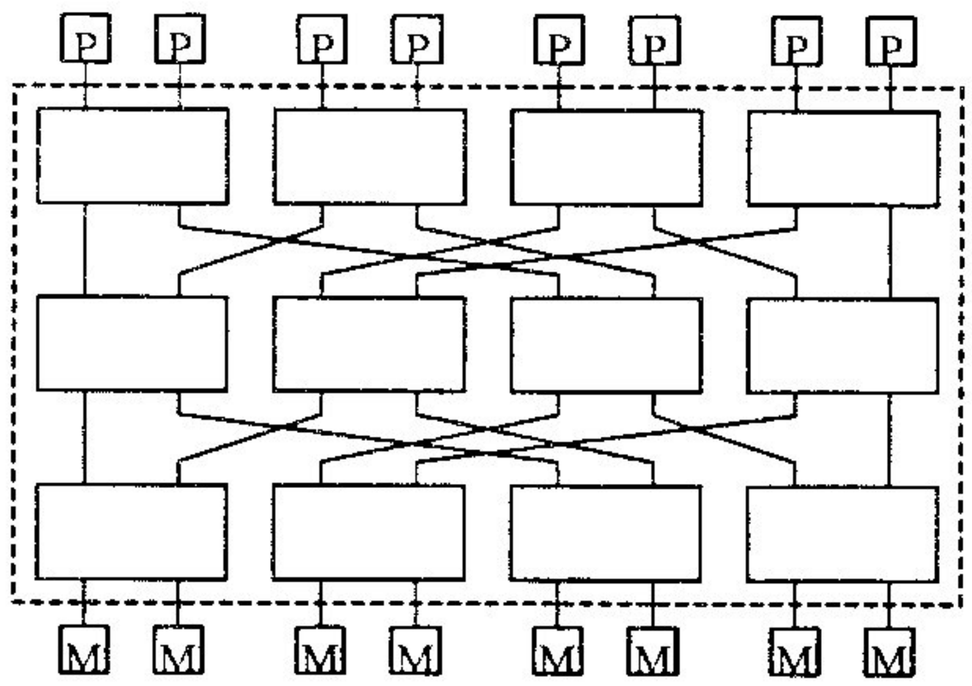


Figura 3.6: Rede de Conexão Multiestágio

3.3 Detalhes de Implementação de Programas Paralelos

O sucesso de um programa paralelo está diretamente relacionado com a precisa identificação das tarefas de um programa serial que poderiam ser executadas concorrentemente. Quanto maior é o número de partes em que um programa é dividido, maiores as chances de paralelização; entretanto, quanto maior for a complexidade do interrelacionamento destas partes, menor é a performance. Esta degradação é devida principalmente ao tempo extra requerido para o escalonamento, sincronização e comunicação dos processos. Esta seção examina os detalhes de implementação que podem minimizar o *overhead* e por conseguinte melhorar a performance de programas paralelos ou vetoriais, como a utilização de técnicas para decomposição de programas seriais em similares paralelos, sincronização de processos, balanceamento de carga entre processadores, etc.

3.3.1 Grafos de Dependência de Dados

Da eficiente e correta decomposição de programas paralelos decorre o seu sucesso. Uma maneira elegante para efetuar tal divisão é a utilização de grafo[1] de controle de fluxo de dados [Dongarra *et al.*, 1991]. Estes grafos podem representar tanto tarefas mais refinadas (paralelismo de grão fino), quanto macrotarefas (paralelismo de grão grosso). Este procedimento consiste em se ter os vértices representando processos que podem ser executados em paralelo, e a orientação das arestas, ou arcos, representando a existência de dependência dos dados em relação a estes processos. Um exemplo ilustra melhor esta técnica. Considere a existência de 5 processos denominados A, B, C, D e E relacionados como mostra a figura 3.7.

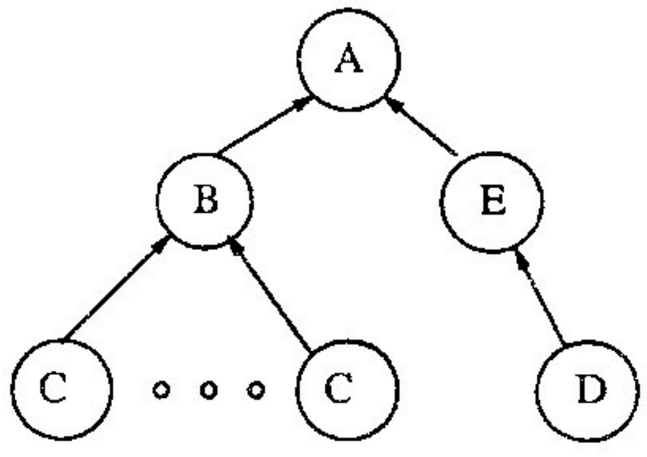


Figura 3.7: Exemplo de Grafo de Dependência de Dados

A existência de n processos C indica que eles operam sobre n conjuntos distintos de dados, que é o caso típico de uma operação vetorizada. A forma como estão relacionados os processos ressalta que os n processos C e o processo D podem ter sua execução feita em paralelo. Somente após o término da execução de C e D é que os processos B e E podem ser executados também em paralelo. Finalmente, o processo A só se inicia após os processos B e E terem chegado ao fim, e a computação é finalizada com o término da execução da tarefa A. A grande vantagem desta abordagem reside no fato de que o nível de detalhe e abstração a que se pode chegar corresponde àquele necessário para o programador desenvolver. Este procedimento permite detectar o *deadlock*, que é caracterizado quando máquinas tipo MIMD têm todos os seus processadores em estado de espera, quando todas as suas atividades cessam. Este fenômeno ocorre devido a uma dependência circular dos dados, i.e., por exemplo, o processo A espera pelos dados do processo B e vice-versa.

3.3.2 Sincronização

A Sincronização é exigida quando dois ou mais processos paralelos precisam se comunicar para, por exemplo, avisar de seu término, indicar a disponibilidade de

[1] Um grafo G é um par ordenado (V, A) onde $V = \{V_1, \ldots, V_r\}$ é um conjunto cujos elementos são chamados vértices e A é um conjunto de pares ordenados (V_i, V_j) de vértices, com $i \neq j$, chamados arestas que representam certas relações existentes entre os vértices [Callioli *et al.*, 1983].

novos dados ou mesmo fornecer informações sobre seu estado atual de execução.

A técnica de sincronização mais comum é a utilização de barreiras, onde um conjunto de processos tem de se reportar à barreira para decidirem se param ou continuam.

Um segundo tipo de sincronização, a nível mais baixo, é exigido, por exemplo, por computadores tipo SIMD que possuem memória compartilhada. A sincronização decorre da necessidade de um ou mais processadores acessarem a mesma porção de memória num mesmo instante, caracterizando o que é conhecido por região crítica. Neste caso, uma função tipo *lock* é utilizada, a qual testa o valor da variável de controle. Se este valor indicar a existência de restrição de acesso à região crítica, o processo é bloqueado até que a restrição seja retirada pelo processo que a originou.

3.3.3 Balanceamento de Carga

Basicamente, o balanceamento de carga procura assegurar que cada processador execute mais ou menos a mesma quantidade de computação de modo a explorar eficientemente os recursos do computador. O objetivo do balanceamento decorre da necessidade de se obter a máxima redução de tempo utilizando o menor número de processadores possível.

Quando é possível conhecer *a priori* o tamanho do problema a ser resolvido, é necessário apenas utilizar os chamados alocadores estáticos para obter o efeito desejado. Estes alocadores implementam políticas de balanceamento considerando, por exemplo, a carga de trabalho em cada processador e a velocidade relativa dos mesmos, no caso de se utilizar máquinas heterogêneas, possibilitando a divisão do problema de forma a contemplar os quesitos em avaliação.

Por outro lado, quando o tamanho das tarefas a serem executadas em paralelo não é conhecido antes de iniciada a computação ou varia no decorrer da execução, como é o caso de alguns problemas de inteligência artificial, é necessário utilizar-se os alocadores dinâmicos. Estes alocadores permitem que no decorrer da execução de uma tarefa paralelizada, a carga em cada processador possa ser atualizada continuamente.

O trabalho de Loures [Loures, 1993] descreve em maior nível de detalhe as diversas políticas de balanceamento existentes.

3.4 Métricas

Comumente a eficiência de um algoritmo em relação a outros se faz através de análise teórica da ordem de complexidade do mesmo ou através de testes de implementações. Com as oportunidades oferecidas pelo processamento paralelo, a neces-

sidade de reportar o desempenho das novas implementações reforça a utilização da experimentação como forma de medir as melhorias alcançadas, em geral, auferindo o tempo gasto para o término da execução de uma computação. A escolha do tempo decorrido como medida de tempo se justifica na medida em que esta abordagem inclui os tempos gastos em, por exemplo, entrada de dados e saída de resultados, *overhead* provocado por atividades do sistema como paginação da memória virtual, escalonamento de processos, uma vez que vários processos podem estar sendo executados em um mesmo processador, e o tempo de espera devido aos processos, relativos a uma mesma tarefa, que estão desbalanceados.

3.4.1 *Speed-up*

A medida de desempenho mais comum é o *speed-up*, que nada mais é do que o fator de ganho de uma implementação paralela em relação a uma similar sequêncial. As várias definições empregadas são fornecidas abaixo:

- *Speed-up*. O *speed-up*, $S(p)$ alcançado por um programa paralelo utilizando p processadores é dado por:

$$S(p) = \frac{\text{Tempo para resolver um problema na implementação serial mais rápida}}{\text{Tempo para executar o código paralelo em } p \text{ processadores}} \tag{3.1}$$

  Se $S(p) = p$ tem-se o chamado *speed-up* linear, o que corresponde a uma aplicação ideal.

- *Speed-up Relativo*: Esta medida é definida como:

$$SR(p) = \frac{\text{Tempo para executar o código paralelo usando um processador}}{\text{Tempo para executar o código paralelo em } p \text{ processadores}} \tag{3.2}$$

  Esta medida é utilizada em geral quando o tempo serial não é conhecido ou o tempo da versão "mono" processada domina as outras implementações.

- *Speed-up Absoluto*:

$$SA(p) = \frac{\text{Tempo mais rápido em qualquer computador serial}}{\text{Tempo para executar o código paralelo em } p \text{ processadores}} \tag{3.3}$$

  Embora esta métrica reflita o principal objetivo da computação paralela, a redução real de tempo, ela obriga o analista a testar os algoritmos em várias máquinas seriais, uma vez que o programa mais rápido é o reflexo do casamento de código e arquitetura.

3.4.2 Outras Métricas

Eficiência é outra medida de performance de programas paralelos, e é definida como a fração do *speed-up* obtido:

$$E(p) = \frac{S}{p} \tag{3.4}$$

onde $S = S(p), SR(p)$ ou $SA(p)$.

Eficiência Incremental é uma forma alternativa de se medir a eficiência de um algoritmo paralelo, e é definida como abaixo:

$$EI(p) = \frac{(p-1)(\text{Tempo para executar o código paralelo em } p-1 \text{ processadores})}{(p)\text{Tempo para executar o código paralelo utilizando } p \text{ processadores}} \tag{3.5}$$

Esta métrica fornece o ganho de tempo obtido ao se acrescentar mais um processador. Ela também pode ser utilizada quando o tempo para um processador não está disponível.

O trabalho de Barr e Hickman [Barr e Hickman, 1993] discute em profundidade a questão da utilização e validade das várias métricas nos relatos sobre performance de novos algoritmos. Também apresenta e discute outras métricas aqui não citadas.

Capítulo 4

Implementação Paralela do MGCP

Este capítulo aborda os vários aspectos da implementação paralela do módulo de solução do sistema de equações baseado no método dos Gradientes Conjugados utilizando a formulação EPE, quais sejam: o *software* e o *hardware* utilizados, implementação do produto matriz-vetor elemento-por-elemento, passagem da versão sequêncial do algoritmo de gradientes conjugados para a versão paralela.

4.1 O Sistema SDP

O código de elementos finitos utilizado foi o SDP e a implementação do módulo de solução é uma adaptação do programa original. O SDP (*Sistema para Desenvolvimento de Programas baseados no Método dos Elementos Finitos*) se originou do trabalho de Gouveia [Gouveia, 1987], desenvolvido no LNCC/CNPq. É um sistema que possui, mais que um conjunto de bibliotecas de rotinas (escritas em FORTRAN) — matemáticas, de utilitários e de elementos —, uma filosofia de programação e de documentação, que facilita o desenvolvimento, manutenção, depuração de programas técnicos-científicos e o intercâmbio entre pesquisadores de diferentes instituições de pesquisa.

Uma característica importante do SDP é a utilização de alocação dinâmica de variáveis, a partir de um vetor de trabalho previamente dimensionado, permitindo que o sistema possa ser portado para várias arquiteturas distintas com relativa facilidade. Em função desta implementação, qualquer variável, seja ela um vetor ou matriz, é vista no SDP como sendo uma tabela, à qual está associada um tipo (por exemplo, inteiro, real de dupla precisão, etc.), a sua dimensão e uma variável inteira que aponta para a localização da tabela no vetor de trabalho. A alocação de tabelas é conseguida com o auxílio de uma série de rotinas utilitárias que permitem

gerenciar a área de trabalho de modo eficiente.

A biblioteca de elementos, que pode ser expandida, é composta de elementos de barras bi e tridimensionais, bem como elementos isoparamétricos para estado plano de tensão e deformação, axissimétricos, flexão de placa e tetraedro para análise de tensão em sólidos 3D.

Na sua versão básica, o SDP é composto de 4 módulos que se comunicam entre si através de sistemas de arquivos criados quando da execução de um pré-processador, um processador, um módulo de solução de sistemas de equações e um pós-processador, nesta ordem. A função do pré-processador é ler um arquivo de dados preparado pelo usuário contendo a descrição da malha de elementos finitos. O processador, a partir do arquivo de dados criado anteriormente, calcula e grava as matrizes de rigidez e vetores de forças nodais equivalentes de cada elemento. O módulo de solução por sua vez, na versão básica, monta a matriz de rigidez armazenando-a segundo a técnica *skyline*, monta o vetor de forças globais, e procede ao cálculo dos deslocamentos nodais através do método de eliminação de Gauss. Finalmente, o pós-processador calcula as tensões nos nós dos elementos.

Esta estrutura modular permite que se atue em partes isoladas do sistema sem contudo afetar outros módulos, como foi o caso deste trabalho. Em contrapartida, se exige um grande espaço em disco para armazenamento, particularmente no caso das matrizes de rigidez de elementos mais elaborados, cuja dimensão pode ultrapassar a 60x60.

No caso de se utilizar programas sequenciais, esta abordagem é bastante razoável, apenas limitando a faixa de atuação na análise de problemas complexos, principalmente quando for crítico o espaço disponível em disco.

Com a utilização de computadores com capacidade de processamento paralelo, e particularmente com memória distribuída, a organização do sistema poderia ser repensada de forma a combinar os vários módulos, permitindo que se calculasse e armazenasse apenas as matrizes de rigidez dos elementos associados a cada processador e, obviamente adotando um método de solução do sistema de equações compatível com a arquitetura do computador utilizado. Neste caso, perderia-se em portabilidade e generalidade mas ganharia-se em eficiência. Esta abordagem é particularmente interessante no caso de solução de problemas que utilizam técnicas adaptativas, os quais requerem a atuação seletiva nas matrizes de rigidez dos elementos. Este é o caso também dos problemas não-lineares, que além de exigirem a solução de diversos sistemas de equações lineares, modificam seletivamente ou não as matrizes de rigidez dos elementos da malha analisada.

4.2 O Sistema PVM

O sistema PVM (*Parallel Virtual Machine*) [Sunderan, 1990, Geist *et al.*, 1993] foi desenvolvido no *Oak Ridge National Laboratory*. Este sistema permite que se utilize um conjunto de computadores heterogêneos, ligados através de uma rede, formando um único recurso computacional, podendo-se tirar vantagem das características individuais de cada máquina de uma maneira coerente e eficiente.

No PVM cada elemento de processamento é chamado nodo ou processador e uma instância em execução denomina-se processo, à qual também se associa um número inteiro positivo. Ele é composto de um núcleo, que gerencia a execução dos processos, e uma biblioteca de funções que permite ao usuário fazer a criação, sincronização e escalonamento de processos, verificação do estado da máquina, depuração, etc. Estas funções podem ser encaradas como uma extensão da linguagem de programação do usuário, neste caso foi utilizado as linguagens "C" e "FORTRAN".

O PVM permite a criação de vários processos, distintos ou não, que podem ser executados em uma mesma máquina, em várias máquinas distintas ou em uma combinação destas. Logicamente, cada máquina deve possuir uma configuração mínima de memória e poder de processamento para executar as tarefas a ela designadas.

O PVM implementa uma máquina paralela com arquitetura de memória distribuída (MIMD), e toda comunicação entre processos se dá através de troca de mensagens, que é realizada através da rede local quando os processos estão em máquinas diferentes. Neste particular, o PVM prevê a construção de canais de comunicação ponto-a-ponto entre pares de processos. Assim, cada mensagem carrega consigo uma identificação do processo origem e do processo destino. Desta forma, a troca de mensagens entre dois processos se dá diretamente, exceto no caso da primeira, porque o núcleo precisa estabelecer este canal.

Como se pode utilizar computadores de diferentes arquiteturas, é necessário que sejam geradas versões do programa de aplicação, assim como do núcleo do PVM, para cada uma das máquinas componentes da máquina virtual.

Antes de iniciar uma aplicação, o usuário deve acionar o núcleo do PVM informando-lhe as máquinas componentes do ambiente paralelo, que em seguida também executarão este núcleo. Desta forma, todos os núcleos têm conhecimento de tudo o que se passa na máquina paralela virtual, ficando ativos até que o usuário determine seu fim. Assim que a aplicação é iniciada, esta e os processos criados pelo PVM têm sua execução e escalonamento subordinados ao sistema operacional.

A versão utilizada nesta implementação é a 3.1.

4.3 A Rede do DCC

Todo o trabalho computacional foi realizado utilizando-se a rede de computadores instalada nos laboratórios do Departamento de Ciência da Computação da UFMG (DCC-UFMG), que está descrita na Fig. 4.1. Nos experimentos foi utilizada uma subrede, que é composta de estações de trabalho SUN, 11 modelo SLC e 2 modelos SPARC-2, com desempenhos nominais respectivamente de 2 e 4 MFLOPS, que estão conectadas através de uma rede tipo *Ethernet* com taxa de transmissão de 10 Mbits/seg.

4.4 O Produto Matriz-Vetor EPE

Como já foi observado anteriormente, pode-se explorar a natureza associativa da obtenção de cada entrada da matriz K na solução do sistema (1.7), utilizando-se o algoritmo 2.3, trabalhando-se ao nível dos elementos. Dessa forma, a matriz K nunca é montada, e o produto matriz-vetor Kp pode ser efetuado da seguinte forma, relembrando a equação (1.8)

$$Kp = \left(\sum_{e=1}^{\mathtt{nel}} K^e \right) p = \sum_{e=1}^{\mathtt{nel}} K^e p^e \tag{4.1}$$

onde `nel` é o número total de elementos, K^e é uma matriz de dimensão `nd*nd` referente ao e-ésimo elemento e p^e um vetor de dimensão `nd`, sendo `nd` o número de equações (graus de liberdade) associadas ao elemento. O vetor p^e é obtido extraíndo-se do vetor p apenas aqueles valores referentes ao elemento em questão, através de uma estrutura de dados que relaciona, para cada elemento, os graus de liberdade deste com os graus de liberdade globais. O vetor contendo o produto Kp é obtido de forma análoga, percorrendo o caminho contrário após a execução do referido produto.

Deve-se observar ainda que, utilizando-se a formulação elemento-por-elemento, não é necessário preocupar-se com a ordem de numeração dos nós, evitando-se assim, a utilização de algoritmos de renumeração nodal para efeito de armazenamento, no caso de se utilizar técnica de armazenamento tipo *skyline*, por exemplo.

4.4.1 Estrutura de Dados Utilizada

As matrizes de rigidez dos elementos, por serem simétricas, requerem apenas o armazenamento da parte supra-diagonal das mesmas ($D^e + U^e$), sendo necessário então uma tabela real de dimensão `nel*(nd*(nd+1)/2)`.

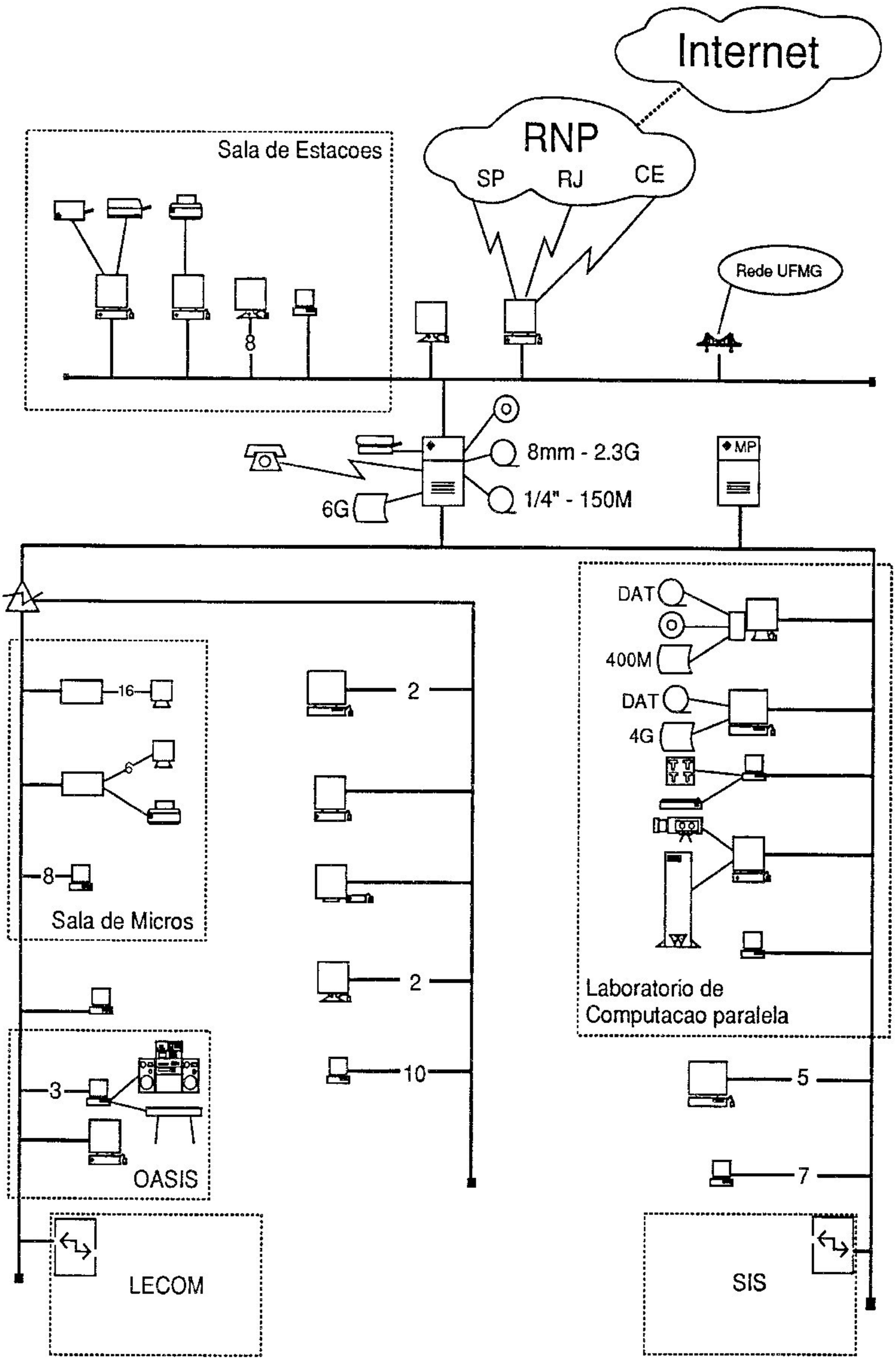
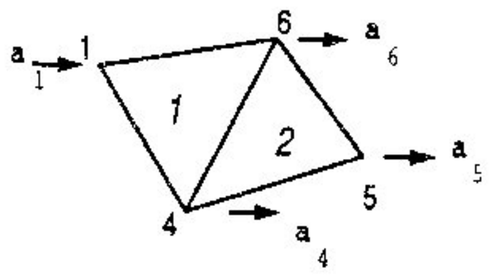


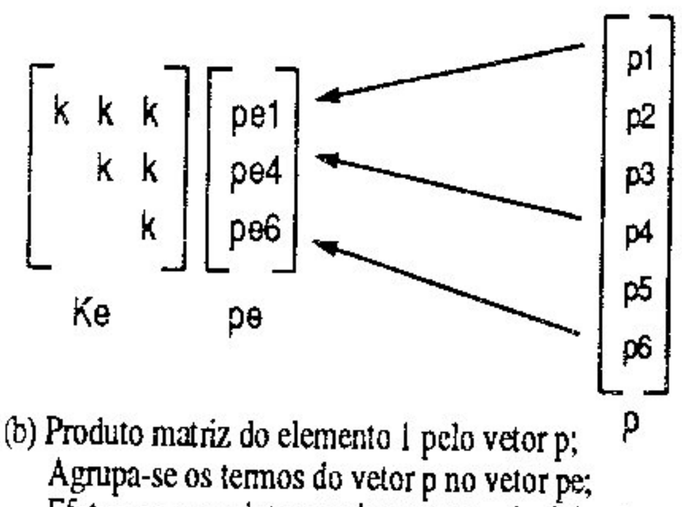
Figura 4.1: Rede de Computadores do DCC-UFMG

O mapeamento dos graus de liberdade locais para os correspondentes globais consiste em armazenar, para cada elemento, o número da equação global relativo à equação associada a cada nó do elemento. Assim, para `nel` elementos que possuam `nd` graus de liberdade cada, é necessário uma tabela, denominada `lm`, de dimensão `nel*nd`. No caso de haver restrição a algum grau de liberdade, este é colocado igual a zero.

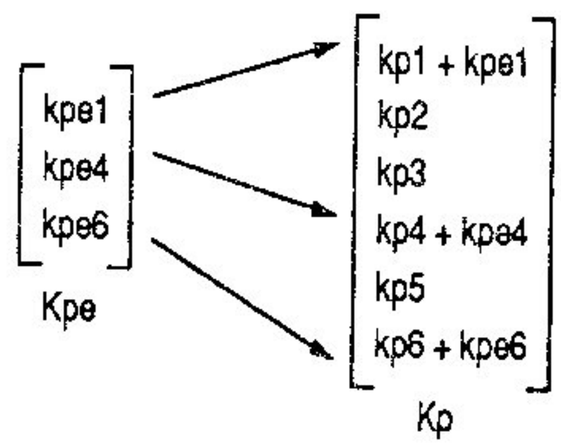
A título de exemplo, considere a malha ilustrada na Fig. 4.2, na qual cada nó possui 1 grau de liderdade apenas. A tabela `lm` neste caso fica: `lm(1)=1`, `lm(2)=4`, `lm(3)=6` para o elemento 1 e `lm(4)=6`, `lm(5)=4`, `lm(6)=5` para o elemento 2.



(a) Malha com 2 elementos; 1 grau de liberdade por no



(b) Produto matriz do elemento 1 pelo vetor p;
Agrupa-se os termos do vetor p no vetor pe;
Efetua-se o produto matriz-vetor ao nivel do elemento



(c) Efetuado o produto, atualiza-se o vetor Kp expandido-se os termos do vetor Kpe para posicoes corretas no vetor Kp

Figura 4.2: Produto Matriz-Vetor ao Nível do Elemento

4.4.2 Implementação

A implementação do produto matriz-vetor Kp consiste então na sequência de operações ilustrada no algoritmo 4.1 abaixo:

início
  repita para `i=1` **até** `nel`
    agrupe em `pe` os valores de `p` associados ao elemento `i` segundo `lm`
    multiplique a matriz `ke` do elemento `i` pelo vetor `pe` retorne o resultado em `kpe`
    acumule o vetor resultado `kpe` em `kp` segundo `lm`
  fim repita
fim

Programa 4.1: Produto Matriz-Vetor EPE

Nota-se, apesar de não se aplicar ao presente trabalho, que o segundo passo do algoritmo anterior é susceptível de vetorização, desde que seja feita na direção dos elementos, uma vez que `nel` $\gg$ `nd`.

Como este algoritmo é de caráter geral, é inevitável a ocorrência de um *overhead* devido à indexação indireta. O ideal, conforme notado também por Carey e Jiang [Carey e Jiang, 1986], seria a particularização deste procedimento para cada tipo de elemento e a consequente explosão dos laços de iteração relativos à multiplicação.

4.5 Derivação da Versão Paralela do MGCP

O objetivo desta seção é descrever a passagem da versão serial do algoritmo de gradientes conjugados para a versão paralela. Primeiramente apresenta-se o algoritmo sequêncial como implementado neste trabalho, passando-se pelas alternativas de paralelização e finalmente chegando-se à versão final paralelizada.

4.5.1 Versão Sequêncial do MGCP

O algoritmo 2.3 sofre algumas modificações quando é implementado em um computador. Visto que se trabalha com aritmética finita é necessário utilizar outro critério de parada, aqui adotado como sendo um limite (ϵ) especificado para a razão entre a norma euclidiana do resíduo da iteração corrente e a mesma norma do resíduo inicial. Outras alterações se fazem necessárias para atender à sintaxe da linguagem utilizada, o FORTRAN. Assim o referido algoritmo fica como ilustra o programa 4.2:

início
  `d=diag(K); k=0; x=0; r=b`
  `rr0=‖r‖`; *resolva* `Mz=r`
  `p=z; rz=⟨r,z⟩`
  repita
    `k=k+1`
    execute produto Kp utilizando algoritmo 4.1
    `pkp=⟨p,kp⟩; alfa=rz/pkp`
    `x=x+alfa*p; r=r-alfa*kp; rr=‖r‖`
    se `(rr/rr0` $\leq$ `epsilon)` **ou** `(k` $>$ `kmax)` **fim**
    resolva `Mz=r; rz2=⟨r,z⟩`
    `beta=rz2/rz; rz=rz2; p=beta*p+z`
  fim repita
fim

onde, para a k-ésima iteração, valem as seguintes definições:
b vetor de forças
d vetor contendo a diagonal da matriz K
M matriz precondicionadora
kp vetor contendo o produto Kp
p vetor direção de busca
r vetor que contém o resíduo
x vetor contendo a solução corrente
z vetor auxiliar
alfa e beta escalares
epsilon a tolerância especificada
kmax número máximo de iterações
pkp escalar contendo o produto escalar dos vetores p e kp
rr0 escalar contendo a norma do resíduo inicial
rr escalar contendo a norma do resíduo r
rz escalar contendo o produto escalar dos vetores r e z
rz2 idem
$\langle ., . \rangle$ denota produto escalar em $\Re$
$\|.\|$ denota norma euclidiana

Programa 4.2: Implementação Sequêncial do Algoritmo de MGCP

4.5.2 Grafo de Dependência de Dados

Uma primeira abordagem para a definição da versão paralela do algoritmo 4.2 é extrair seu grafo de dependência de dados, como visto na seção 3.3.1.

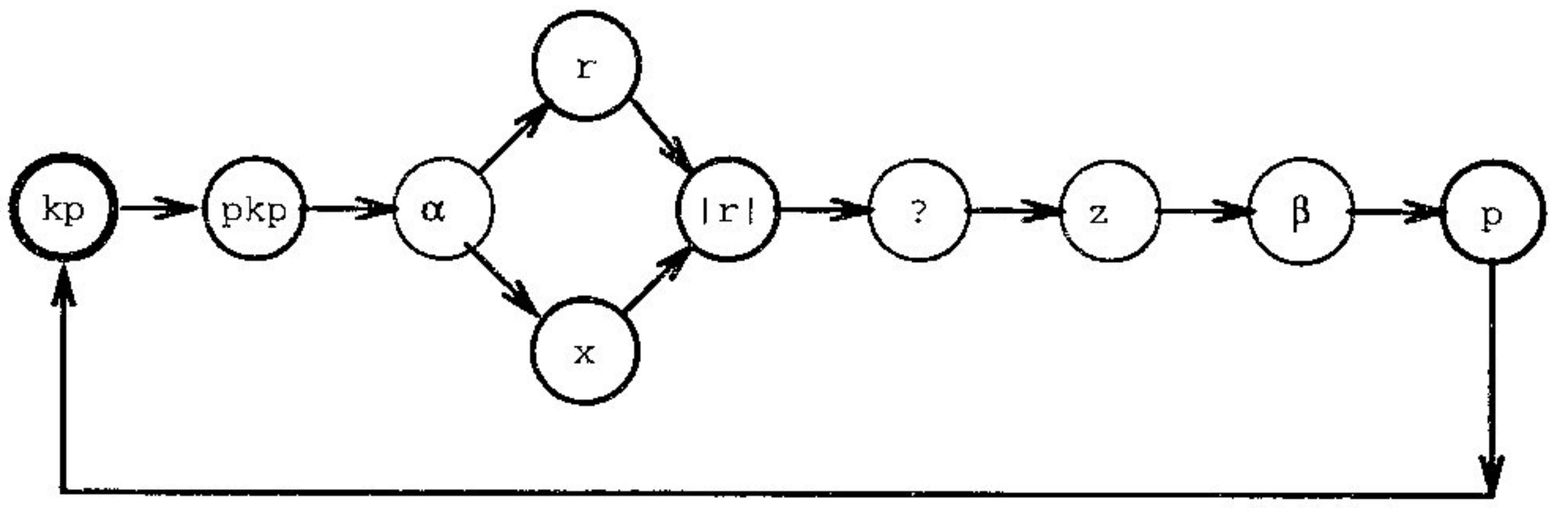


Figura 4.3: Grafo de Dependência de Dados

Para facilitar a análise, grafou-se as operações computacionalmente mais caras com linhas gradativamente mais espessas. Cotejando-se o grafo mostrado na Fig. 4.3

nota-se que ao nível de dependência de dados, o algoritmo 4.2 oferece poucas chances de paralelização, sendo altamente susceptível de vetorização. Observa-se porém, que a etapa mais restritiva desse algoritmo é o produto matriz-vetor, que é $O(n^2)$.

Uma vez que, uma máquina com memória distribuída está disponível, o problema reside então na identificação de uma determinada decomposição do algoritmo 4.2 que leve a uma implementação eficiente. Tal divisão tem que levar em consideração a quantidade de comunicação entre processos que o novo algoritmo exigirá, desde que toda comunicação é feita através de uma rede local. Todavia, como não se sabe o impacto que a quantidade de comunicação tem sobre a eficiência de um algoritmo desta natureza, é necessário pensar e implementar várias alternativas de modo a identificar aquela de melhor desempenho dentro das características e restrições impostas por este tipo de arquitetura.

Como paradigma de programação, adotou-se o modelo mestre-escravo, no qual o processo mestre delega tarefas a cada processo escravo, gerencia a execução das mesmas e pode ou não executar outras tarefas. O grafo na Fig. 4.4 ilustra este modelo de programação paralela, no qual os vértices representam processos ou tarefas e as arestas (arcos) representam a dependência de dados e/ou processos, que podem ou não estar no mesmo processador.

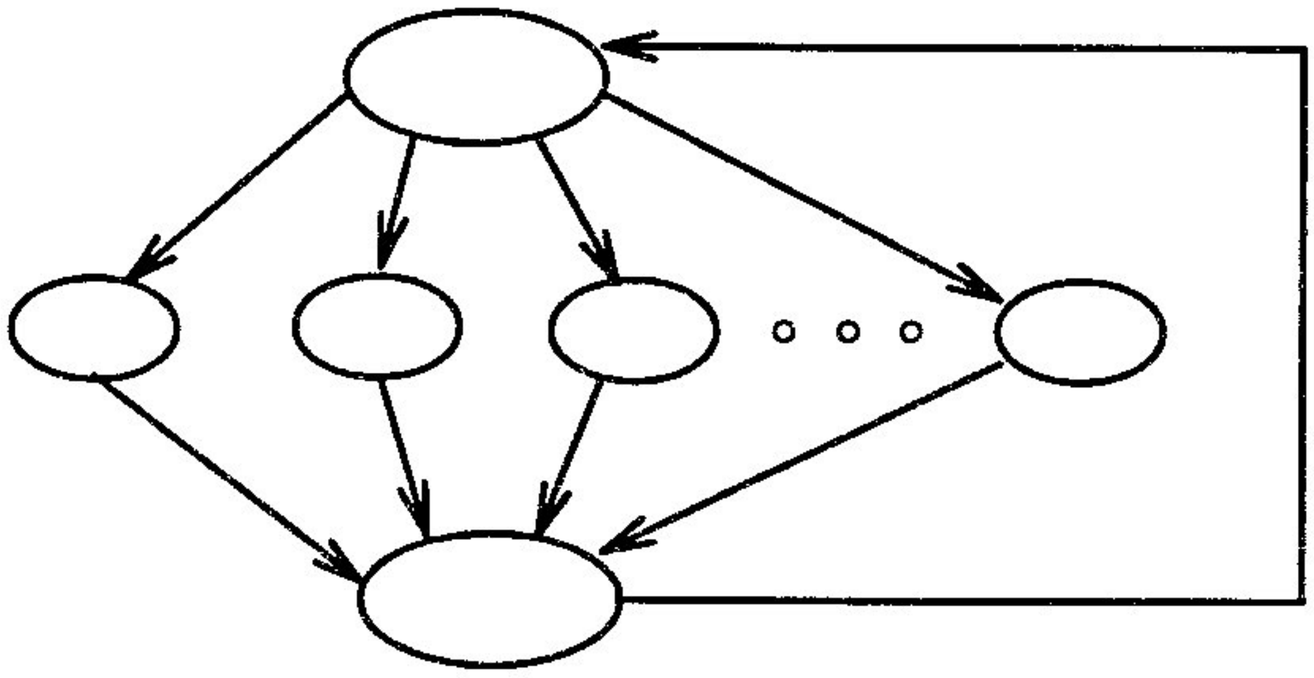


Figura 4.4: Grafo de Distribuição de Tarefas no Modelo Mestre-Escravo

4.5.3 Alternativas para Decomposição do Algoritmo MGCP

Na discussão que se segue, considere-se uma malha com 21121 nós, 6912 elementos isoparamétricos de 8 nós que resulta em um sistema de 42048 equações. Para efeito de comparação, a Fig. 4.5 ilustra como a malha em questão foi obtida.

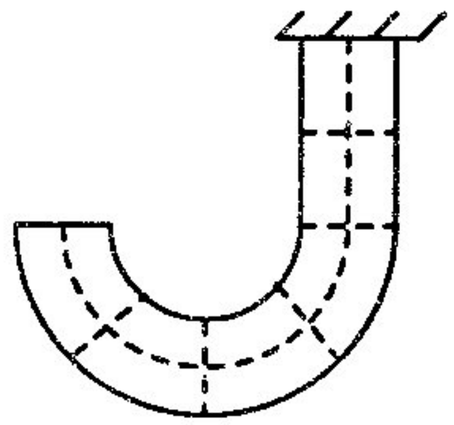


Figura 4.5: Malha para Discussão de Caso

Primeira Alternativa

Uma primeira análise do problema sugere que cada processador receba a matriz K^e e o vetor p^e efetue a multiplicação desta matriz pelo vetor p^e correspondente, a cada iteração, até que todos os elementos tenham sido contemplados, como mostrado pelos programas 4.3 e 4.4. A performance deste algoritmo se mostrou a mais medíocre possível, uma vez que exige uma quantidade enorme de bytes trafegando pela rede, senão veja-se os números: a cada iteração tem-se que comunicar um total de `nel` matrizes de elementos de dimensão `nd*(nd+1)/2` e 2 vetores de dimensão `nd` resultando em `nel*(nd*(nd+5)/2)*8 bytes`, o que em números, para a malha em questão, resulta em aproximadamente 9 Mbytes por iteração! Observa-se também que se consegue um excelente balanço de carga, mas o ganho de velocidade obtido com esta divisão não é suficiente para compensar o atraso imposto pela comunicação entre os processos, sendo necessário portanto, impor que cada processo escravo execute mais computação, bem como diminuir a quantidade de comunicação entre processos.

início *Processo Mestre*
  `i=1`
  enquanto *houver elementos a serem considerados* **faça**
    enquanto *houver processo escravo livre* **faça**
      agrupe em `pe` os valores de `p` associados ao elemento `i` segundo $\texttt{lm}^{(\texttt{i})}$
      envie a matriz `ke` e o vetor `pe` do elemento `i`
      `i=i+1`
    fim enquanto
    se *algum processo escravo executou tarefa* **então**
      receba `kpe`
      acumule o vetor resultado `kpe` em `kp` segundo $\texttt{lm}^{(\texttt{i})}$
    fim se
  fim enquanto
fim

Programa 4.3: Produto Matriz-Vetor EPE — Processo Mestre

início *Processo Escravo*
  enquanto *houver elementos a serem considerados* **faça**
    receba a matriz ke e o vetor pe de um determinado elemento
    execute produto Kp utilizando algoritmo 4.1
    envie o vetor kpe ao processo mestre
  fim enquanto
fim

Programa 4.4: Produto Matriz-Vetor EPE — Processo Escravo

Uma Segunda Alternativa

Uma vez que na análise linear de estruturas as matrizes de rigidez dos elementos não são alteradas a cada iteração, pode-se dividir a malha em questão em grupos de elementos, alocando-se em cada processador um certo número de elementos e consequentemente restrigindo-se a comunicação entre o processo mestre e os processos escravos apenas aos vetores pe e kpe dos elementos de cada grupo, conforme ilustram os programas 4.5 e 4.6. Neste novo algoritmo conseguiu-se aumentar enormente a quantidade de computação em cada processo, garantiu-se um bom balanço de carga entre os processos escravos e também diminuiu-se bastante a quantidade de comunicação entre processos, como já notado. Para efeito de comparação, para a mesma malha anteriormente utilizada, o número de bytes em tráfego na rede por conta desta nova implementação é: $(2 * \sum_{i=1}^{p} \texttt{nelg}_i * \texttt{nd}) * \texttt{8bytes} = 2 * \texttt{nel} * \texttt{nd} * \texttt{8bytes} \approx$ 1,7 Mbytes, ainda assim um número extremamente elevado, resultando em um algoritmo de pouquíssima eficiência.

início *Processo Mestre*
  repita para i=1 **até** ng
    agrupe em pe os valores de p associados aos elementos do grupo i
    segundo $\texttt{lm}^{(\texttt{i})}$
    envie vetor pe relativo ao grupo em questão
    i=i+1
fim repita
repita para i=1 **até** ng
  se *processo está livre* **então**
    receba kpe relativo ao grupo i
    acumule o vetor resultado kpe em kp segundo $\texttt{lm}^{(\texttt{i})}$

```
            i=i+1
        fim se
    fim enquanto
fim
```

onde ng é o número de grupos em que se dividiu a malha.

Programa 4.5: Produto Matriz-Vetor EPE por Grupos — Processo Mestre

```
início Processo Escravo
    receba pe relativo a um determinado grupo
    repita para i=1 até nelg
        execute produto Kp utilizando algoritmo 4.1
        i=i+1
    fim repita
    envie o vetor kpe ao processo mestre
fim
```

onde nelg é o número de elementos de um determinado grupo.

Programa 4.6: Produto Matriz-Vetor EPE por Grupos — Processo Escravo

A última solução encontrada, embora tenha diminuido enormemente a quantidade de comunicação entre os processos, ainda não conseguiu resultar em uma performance aceitável, sendo portanto necessário buscar nova solução.

A Alternativa Final

Observando-se o algoritmo 4.5 e a malha ilustrada na Fig. 4.5 mais detidamente, nota-se claramente que no caso específico do produto matriz-vetor, é necessário e suficiente que o processo responsável por determinado grupo de elementos comunique apenas os graus de liberdade associados aos nós dos elementos sob sua responsabilidade, que são comuns a elementos de outros grupos, uma vez que os elementos de kp relativos a estes graus de liberdade são obtidos através da contribuição de elementos que compartilham, digamos, uma mesma aresta, imaginando que se possa traçar uma linha representando a divisão da malha em grupos. Assim, os graus de liberdade restantes, i.e., que não pertencem a mais de um grupo, são tratados localmente em cada processo escravo. Procedendo deste modo, a tarefa de efetuar os produtos escalares pode ser delegada aos processos escravos, encarregados, em outras palavras, por certo número de graus de liberdade, o que faz com que cada

processo escravo comunique o resultado parcial de um determinado produto escalar para que seja efetuada a totalização do mesmo.

Fox *et al.* [Fox *et al.*, 1988] propõem algoritmo similar que foi implementado em um computador tipo hipercubo. Nesta implementação os autores assumem ser conhecidas as relações de vizinhança entre os grupos de elementos e sugerem que a comunicação entre processos seja feita — imaginando-se uma malha plana de elementos — da esquerda para direita e de baixo para cima, sendo que o último dos processos, após atualizar os valores recebidos, devolve cópias dos valores atualizados de `kp` seguindo caminho inverso, uma vez que, cada processador necessita destes valores para os cálculos subsequentes. Tome-se como exemplo o "trecho" de malha ilustrada na Fig. 4.6.

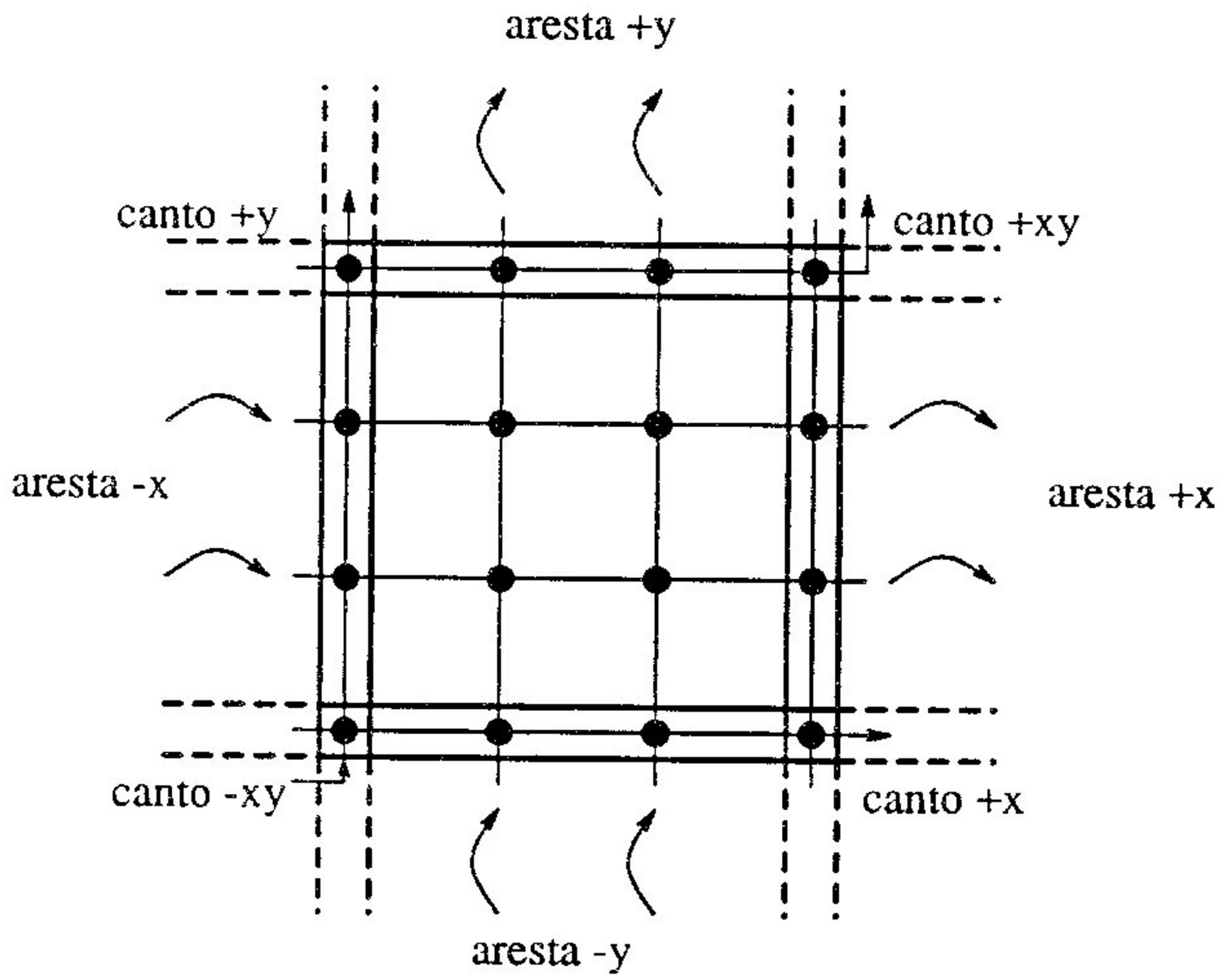
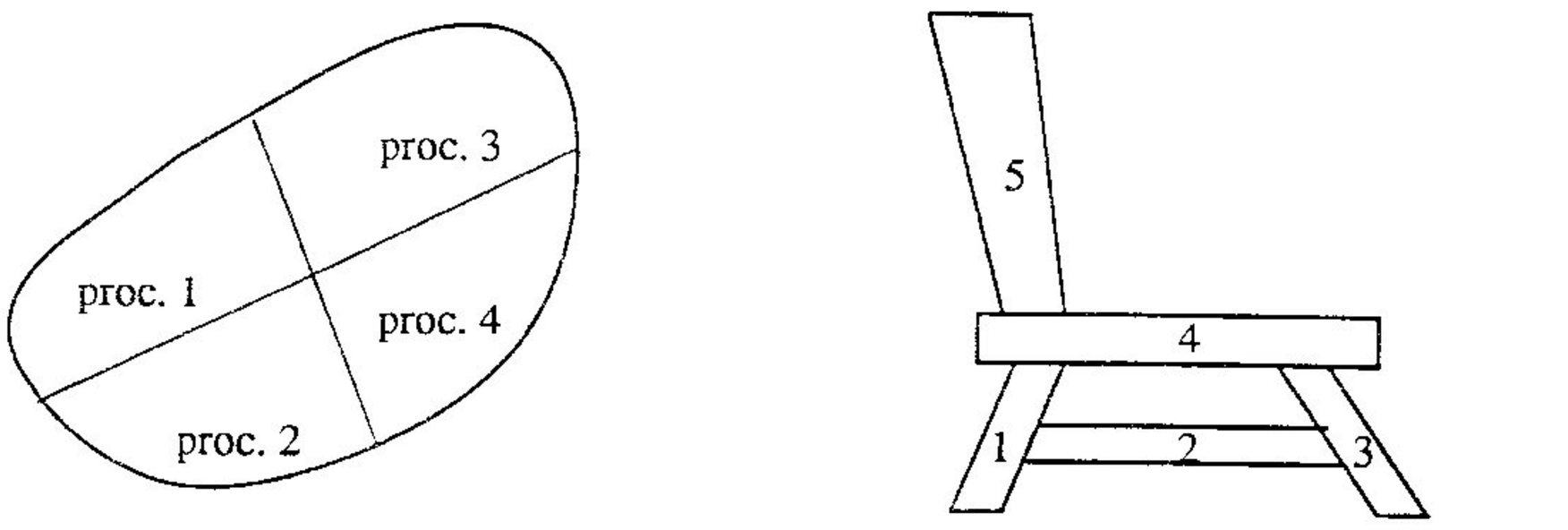


Figura 4.6: Esquema de Comunicação Adotado por Fox *et al.*

Os autores sugerem que todos os graus de liberdade de nós pertencentes às arestas $-x$ e $-y$, de um dado processador, sejam acumulados dentro deste processador. Contrariamente, os graus de liberdade dos nós restantes são enviados ao processador vizinho. Desta forma se garante que cada grau de liberdade seja acumulado uma única vez. Segundo os autores, a implementação paralela do algoritmo do método dos Gradientes Conjugados sem precondicionamento em um código bidimensional de elementos finitos, seguindo este esquema de comunicação, apresentou boa concordância com a eficiência teórica esperada, praticamente linear, para processos com mais que 10 elementos.

A eficiência deste esquema de comunicação depende tanto da arquitetura utilizada, quanto da maneira como a malha de elementos é dividida. Se o domínio a ser decomposto é simplesmente conexo, este esquema de comunicação funciona mesmo naqueles computadores que permitem comunicação somente entre processos vizinhos, afirmam os autores. Todavia, se o domínio é complexo ou multiplamente conexo, como ilustrado na Fig. 4.7, estas máquinas podem oferecer restrições adicionais, uma vez que existe limitações quanto ao número de ligações entre processos, afetando negativamente a performance do algoritmo.



a) Domínio Simplesmente Conectado

b) Domínio Multiplamente Conectado

Figura 4.7: Tipos de Domínio

4.5.4 Algoritmo Paralelo de MGCP

Neste trabalho optou-se por efetuar a comunicação dos graus de liberdade comuns entre processos escravos utilizando-se o processo mestre como um roteador, i.e., ao processo mestre foi delegada a função de receber, atualizar e enviar todos os graus de liberdade comuns aos vários grupos de elementos, seletivamente, ou seja, o processo mestre recebe/envia somente aqueles graus de liberdade associados aos nós sob a responsabilidade de determinado processador. Esta escolha se deve principalmente ao fato de que não há restrições quanto ao número de ligações existentes entre processos na arquitetura utilizada, simplificando muito o algoritmo de comunicação, permitindo o tratamento de problemas mais complexos, com domínios 3D. Abstraindo-se da existência de procedimento de identificação dos graus de liberdade comuns — que será objeto de discussão posteriormente — a versão paralela do algoritmo 4.2 fica como mostrado nos programas 4.7 e 4.8:

início *Processo Mestre*

inicie processos escravos

selecione equações comuns a cada grupo

recupere informações do grupo sob sua jurisdição

construa d=diag(K) dos elementos do grupo sob sua jurisdição
receba valores de d relativos aos graus de liberdade comuns *atualizando-os*
envie valores atualizados de d dos graus de liberdade comuns para o
    processo escravo adequado
k=0; x=0
r=b
rr0=‖r‖
receba rr0 de cada processo escravo *atualizando-o*
envie rr0 atualizado para todos os processos escravos
resolva Mz=r
p=z
rz=⟨r, z⟩
receba rz de cada processo escravo *atualizando-o*
envie rz atualizado para todos os processos escravos
repita
    k=k+1
    execute produto Kp utilizando algoritmo 4.1
    receba valores de kp relativos aos graus de liberdade comuns *atualizando-os*
    envie valores atualizados de kp dos graus de liberdade comuns para
        o processo escravo adequado
    pkp=⟨p, kp⟩
    receba pkp de cada processo escravo *atualizando-o*
    alfa=rz/pkp
    envie alfa para todos os processos escravos
    x=x+alfa*p
    r=r-alfa*kp
    rr=‖r‖
    receba rr de cada processo escravo *atualizando-o*
    envie rr atualizado para todos os processos escravos
    se (rr/rr0 ≤ epsilon) **ou** (k > kmax) **fim repita**
    resolva Mz=r
    rz2=⟨r, z⟩
    receba rz2 de cada processo escravo *atualizando-o*
    beta=rz2/rz
    envie beta atualizado para todos os processos escravos
    rz=rz2
    p=beta*p+z
fim repita

receba **x** de cada processo escravo *atualizando-o*
fim

Programa 4.7: Implementação Paralela do MGCP — Processo Mestre

início *Processo Escravo*
recupere informações do grupo sob sua jurisdição
construa d=diag(K) dos elementos do grupo sob sua jurisdição
envie valores de d relativos aos graus de liberdade comuns do grupo
sob sua jurisdição
receba valores atualizados de d realtivos aos graus de liberdade comuns do
processo mestre
k=0; x=0
r=b
rr0=‖r‖
envie rr0 ao processo mestre
receba rr0 atualizado do processo mestre
resolva Mz=r
p=z
rz=⟨r,z⟩
envie rz ao processo mestre
receba rz atualizado do processo mestre
repita
k=k+1
execute produto Kp utilizando algoritmo 4.1
envie valores de kp relativos aos graus de liberdade comuns do grupo
sob sua jurisdição
receba valores atualizados de kp relativos aos graus de liberdade comuns do
processo mestre
pkp=⟨p,kp⟩
envie pkp ao processo mestre
receba alfa do processo mestre
x=x+alfa*p
r=r-alfa*kp
rr=‖r‖
envie rr ao processo mestre
receba rr atualizado do processo mestre

se (rr/rr0 $\leq$ epsilon) ou (k > kmax) **fim repita**
resolva Mz=r
rz2=⟨r, z⟩
envie rz2 ao processo mestre
receba beta atualizado do processo mestre
p=beta*p+z
fim repita
envie **x** ao processo mestre
fim

Programa 4.8: Implementação Paralela do MGCP — Processo Escravo

Como pode ser observado nos algoritmos 4.7 e 4.8, a execução dos produtos escalares é feita paralelamente em cada processo responsável por determinada quantidade de graus de liberdade. Deve ser ressaltado também, que os processos escravos efetuam os referidos produtos escalares associados apenas aos graus de liberdade locais. Por outro lado, o processo mestre é responsável pela execução destes mesmos produtos escalares tanto dos graus de liberdade locais quanto dos graus de liberdade comuns a mais de um processo. Assim, é necessário que cada processo escravo envie o resultado parcial do produto escalar efetuado ao processo mestre, para que o mesmo faça a atualização, e envie o resultado a todos os processos escravos para os cálculos subsequentes. No que se refere à soma de vetores, esta é efetuada localmente, não necessitando nenhum tipo de comunicação entre processos. Procedendo assim, verifica-se que a comunicação entre processos requer, a cada iteração, $(6 * (p - 1) + 2 * \sum_{i=1}^{p-1} nglg_i) * 8$ bytes, onde nglg é o número de graus de liberdade comuns a cada grupo e p é o número de processadores. Em termos de números, o processamento da malha ilustrada na Fig. 4.5, resultaria no montante de bytes registrado na tabela abaixo:

	nº de processadores			
	3	6	9	12
soma das equações comuns	582	1746	2910	4074
nº de bytes transferidos	9408	28176	46944	65712

Tabela 4.1: Número de Bytes Transferidos Utilizando Comunicação Seletiva

Cotejando-se a Tab. 4.1 verifica-se prontamente a grande redução alcançada no número de bytes em tráfego com a utilização de comunicação seletiva.

4.5.5 Identificação dos Nós de Interface

O problema agora é, assumindo que a malha de elementos finitos esteja dividida em grupos, identificar os graus de liberdade de cada grupo e isolar aqueles que sejam comuns a mais de um grupo. Ressalta-se inicialmente que, a fim de facilitar a operação de comunicar estes dados, optou-se por armazená-los contiguamente. Para isto, basta proceder a uma reorganização dos mesmos dividindo os vetores em duas partes. A primeira delas contém os graus de liberdade comuns, sendo que a segunda parte contém os graus de liberdade locais.

A estratégia adotada neste trabalho consiste em armazenar um par de valores inteiros, representado por $L(n, g)$ onde L é uma lista encadeada, n é o número do nó e g é o número do grupo ao qual este nó pertence. Outros tipos de estrutura de dados (ED), como por exemplo, árvore binária, poderiam ser utilizados. Entretanto, a escolha deste tipo de ED se prende à relativa facilidade de implementação da mesma, tendo em vista que a linguagem de programação utilizada não oferece recursos extras, como ponteiros e alocação dinâmica de memória, para implementar facilmente outros tipos de ED.

Uma vez que a malha de elementos está organizada em grupos de elementos, a referida lista de informações relativas à topologia da malha é obtida percorrendo a lista de incidências dos elementos de cada grupo e extraindo da mesma o número do nó e do grupo ao qual ele pertence. Desde que os processos escravos necessitam conhecer os nós dos elementos associados a eles próprios, uma cópia da lista é enviada a cada processo escravo, entretanto, somente a parcela relativa ao grupo em questão.

Após a obtenção da lista, procede-se à ordenação topológica da mesma, i.e., utilizam-se duas chaves, denominadas primária e secundária. A chave primária é o número do nó e a chave secundária é o número do grupo. Assim sendo, pode-se identificar, unívocamente, se um mesmo nó pertence a grupos distintos pela obrigatória repetição do número do mesmo na lista ordenada. A ordenação foi feita pelo método *Quick Sort* utilizando-se uma rotina da biblioteca padrão da linguagem C encontrada no sistema UNIX, o `qsort()`.

A montagem da lista de nós comuns é efetuada em dois passos. O primeiro passo consiste em preencher uma lista auxiliar contendo apenas os nós repetidos que têm mais de um grupo associado ao mesmo, armazenando também os números dos grupos. O segundo passo corresponde a "enxugar" a lista gerada, retirando a repetição de nós. A existência destes dois passos se justifica devido à necessidade de se preservar a lista auxiliar, a partir da qual retirar-se-á uma identificação única que representa, digamos, a presença dos diversos grupos em um dado nó. Esta identificação se baseia na utilização da representação binária que um número inteiro possui, ou seja, cada bit vai indicar a presença ou não de determinado grupo.

Um exemplo ilustra melhor: sejam os números 0, 2 e 7, cuja representação binária (imaginando um número inteiro de 1 byte, ou 8 bits) de cada um deles é 00000000, 00000010 e 00000111, respectivamente. O número 0 indica que nenhum grupo está presente, e o número 2 indica que o grupo de número 2 faz parte da lista de nós. Por sua vez o número 7 indica que os grupos 1, 2 e 3 contêm algum nó, que neste caso é comum. A implementação deste artifício não requer mais do que uma linha de código C. Desta forma é necessário apenas um número inteiro, na presente implementação de 4 bytes, para identificar de modo único os vários grupos aonde determinado nó está presente. Este procedimento evita que se utilize uma ED que teria que ser capaz de armazenar tanto o número do nó, quanto uma quantidade variável dos números de grupos a ele associados. Este é o típico caso, por exemplo, de se utilizar uma lista duplamente encadeada.

Uma vez identificado em quais grupos estão os diversos nós comuns, é necessário enviar a cada processo escravo esta informação, de modo que cada um deles possa identificar dentre os seus nós aqueles que lhe são comuns.

A partir da identificação por cada processo dos seus nós comuns, pode-se proceder a uma renumeração das equações, numeradas globalmente, e obter uma numeração local, tendo também a preocupação de começar esta renumeração a partir dos nós comuns. Efetuada a renumeração, cada processo escravo comunica ao processo mestre a sua nova numeração, para que este faça um mapeamento das mesmas a fim de compatibilizar a numeração local dos nós comuns de um processo com aquela adotada pelo processo mestre.

A estrutura de dados neste caso, consiste em utilizar um vetor particionado em $p-1$ partes, cada qual associada a um determinado processador p. O mapeamento se faz então, relacionando as equações comuns renumeradas pelos processos escravos, com aquelas renumeradas pelo processo mestre. Exemplo: admita-se que as equações globais numeradas como 10, 15, 21, 30 e 45, de uma determinada malha, sejam comuns a dois ou mais processos. O processo mestre renumera-as em sequência, i.e., 1, 2, 3, 4 e 5. Suponha-se que ao processo 1 sejam comuns as equações 15 e 30, que renumera-as como 1 e 2. O mapeamento consiste em o processo mestre saber que as equações 1 e 2, do processo 1, correspondem às equações 2 e 4, por ele renumeradas.

A partir do exposto, inicia-se o procedimento de solução do sistema de equações, como já descrito anteriormente.

Capítulo 5

Análise dos Experimentos Numéricos

O objetivo deste capítulo é analisar os experimentos numéricos realizados a fim de verificar a eficiência da implementação paralela do Método de Gradientes Conjugados proposta neste trabalho. Primeiramente são feitas algumas considerações iniciais e em seguida dois exemplos são apresentados e discutidos.

5.1 Considerações Gerais

Segundo Loures *et al.* [Loures *et al.*, 1992] a utilização de uma rede *workstations* trabalhando concorrentemente apresenta uma relação custo-benefício favorável. Neste trabalho, os autores investigam o impacto da comunicação sobre a performance de programas paralelos, implementados em uma arquitetura tipo MIMD (no caso, utilizando o PVM), e identificaram que o emprego de uma rede heterogênea[1] de estações de trabalho conduz a melhor performance, desde que a aplicação seja estruturada na forma de tarefas de grande granularidade, como meio de minimizar o custo da comunicação, i.e., de acordo com a quantidade de mensagens trocadas entre os processos da aplicação, uma menor quantidade de máquinas, distribuídas de modo que os processos sequenciais executem nas *workstations* mais rápidas, pode reduzir parte da comunicação entre processos. Por outro lado, os mesmos autores afirmam que para tarefas de pouca granularidade, o ganho de tempo na utilização da configuração heterogênea em relação à homogênea não é bastante significativo, isto porque a diminuição no número de máquinas diminui o paralelismo da aplicação, não trazendo ganhos nem com a dimuinção da comunicação, nem na parte sequêncial do algoritmo.

[1] Uma rede heterogênea consiste, neste caso, na substituição de 2 estações de trabalho mais lentas, por uma mais veloz que possua desempenho equivalente.

Neste trabalho foi adotada uma rede heterogênea. Esta opção se deve ao fato de que o processo mestre tem uma fração sequêncial maior em relação aos processos escravos, uma vez que a ele foi delegada a tarefa de atualizar as equações comuns a todos os processos, bem como a responsabilidade por um grupo de elementos. Também, quando o número de elementos da malha analisada não era múltiplo do número de grupos em que se desejava efetuar uma partição, o critério adotado foi o de alocar no processo mestre a maior parcela de elementos. Tendo em vista estas considerações, manteve-se um processo escravo executando juntamente com o processo mestre na máquina mais rápida e os outros processos escravos nas demais.

A fim de validar a opção pela configuração heterogênea, foram testados, mas não registrados neste trabalho, alguns casos dos exemplos mostrados nas próximas seções, tanto para uma configuração homogênea quanto para a configuração heterogênea adotada, verificando-se que os tempos de execução nesta última, foram sempre um pouco menores que na configuração homogênea, conforme salientado no trabalho de Loures *et al.* [Loures *et al.*, 1992]. Isto se deve ao fato de que o tempo da execução de uma aplicação paralela é dominado por aquela máquina mais lenta, seja porque ela não é uma máquina veloz, ou porque a tarefa que lhe foi delegada é maior em relação às tarefas em outros processadores, ou em outras palavras, depende do balanço de carga em cada processador.

Com relação ao balanço de carga em termos de quantidade de computação, procurou-se alocar em cada processador o mesmo número de elementos. Mais especificamente, parte-se de uma malha sem nenhuma divisão, e considera-se como primeiro grupo, os primeiros `nel/p` elementos, sendo `nel` o número de elementos e `p` o número de processos em que se deseja resolver o problema em consideração, e assim por diante. Este critério foi o melhor encontrado, uma vez que o ideal seria levar em consideração também a quantidade de comunicação decorrente de tal divisão, quando se procuraria minimizar e equidistribuir a comunicação entre processos. Pode-se adiantar que esta não é tarefa fácil, constituindo hoje, um fértil campo para pesquisa, uma vez que este tipo preocupação faz parte do dia-a-dia de pesquisadores de outras áreas, como redes neuronais. Como referência, pode-se citar o trabalho de Justino Filho e Ebecken [Justino Filho e Ebecken, 1993], o qual propõe uma heurística para a decomposição de domínio automática de malhas de elementos finitos.

Deve ser ressaltado também, que devido à limitação de memória das SLC's (possuem 8 Mbytes), os tempos da versão sequêncial do algoritmo de MGCP, para alguns casos dos exemplos a seguir, foram obtidos utilizando-se uma SPARC-2 (32 Mbytes), sendo necessário então corrigi-los (seção 4.3, página 30). Conforme será visto nos exemplos aonde foi possível obter tempos de execução nas SLC's, este fator fica em torno de 2,3. Tendo em vista o exposto, o *speed-up* alcançado foi calculado

utilizando-se a expressão (3.2) como se segue:

$$SR = \frac{\text{tempo sequêncial na SLC}}{\text{tempo paralelo}} \tag{5.1}$$

e assim, a medida da eficiência é dada por:

$$E = \frac{SR}{\text{número de grupos}} \tag{5.2}$$

Como o SDP não possui um gerador automático de malhas, toda esta etapa foi realizada utilizando-se o programa ANSYS[2] como suporte, a partir do qual se cria um arquivo contendo as informações sobre a malha de elementos finitos, entretanto, com sintaxe própria, o que obrigou a proceder à conversão destes dados para o formato aceito pelo SDP, através de programa conversor escrito durante a realização deste trabalho.

Nos experimentos numéricos realizados foi adotado uma tolerância $\epsilon = 10^{-6}$ e admitiu-se um máximo de 5000 iterações como critérios de parada.

Nos exemplos que se seguem, são apresentadas informações adicionais acerca do armazenamento da matriz de rigidez global, caso ela fosse montada em algum momento, apenas para efeito de comparação. São fornecidos então, valores para armazenamento da referida matriz com utilização de algoritmo de renumeração nodal[3] com o respectivo valor da largura de banda média. Também para efeito de comparação, foram tomados, quando possível, os tempos de solução do sistema de equações utilizando-se o método de eliminação de Gauss em uma SLC, ou foi feita correção já citada.

Um dos objetivos comuns aos exemplos A e B é mostrar o quanto é importante o balanço de carga entre processadores, bem como salientar a necessidade de se possuir um algoritmo que faça a decomposição automática da malha de elementos finitos, conforme já citado.

5.2 Exemplo A

Este exemplo trata de um problema de estado plano de tensões, que consiste em um gancho chato redondo, preso em uma das extremidades, e suportando uma carga distribuída vertical de 0.3 kN/cm na extremidade livre, conforme mostra a Fig. 5.1.

Para o mesmo problema foram geradas malhas gradativamente crescentes tanto em número de elementos quanto em número de nós. O balanço de carga foi assegurado procurando-se manter o mesmo número de elementos em cada processador.

---

[2]Marca registrada da Swanson Analysis Systems, Inc.

[3]Neste caso, Reverse Cuthill-Mckee

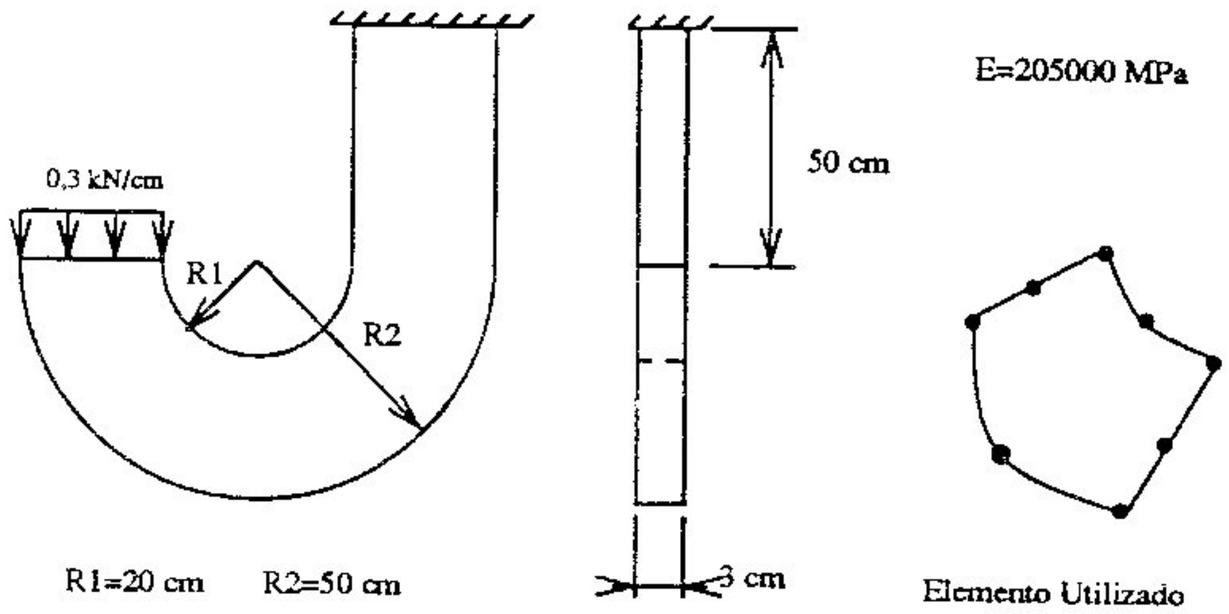


Figura 5.1: Exemplo A – Gancho Redondo

A Tab. 5.1 mostra o número de elementos (isoparamétricos, quadrangular de 8 nós) utilizado, o número de graus de liberdade, e as quantidades relativas ao armazenamento da matriz de rigidez. A partir da mesma, verifica-se imediatamente as vantagens em se utilizar a formulação EPE no que diz respeito aos requisitos de armazenamento.

Os tempos médios (3 execuções), relativo apenas à etapa de solução do sistema de equações, utilizando o método de Eliminação de Gauss, método de Gradientes Conjugados com precondicionamento diagonal sequêncial e paralelo, estão na Tab. 5.2 que se segue.

malha	número de			matriz de rigidez		
	elementos	nós	equações	nro. entradas	banda média	elementos
06x18	108	373	720	23.168	32	14.688
08x24	192	641	1248	51.176	41	26.112
10x30	300	981	1920	95.720	50	40.800
12x36	432	1393	2736	160.640	59	58.752
16x48	768	2433	4800	366.968	76	104.448
18x54	972	3061	6048	516.056	85	132.192
24x72	1728	5377	10656	1.193.096	112	235.008
32x96	3072	9473	18816	2.775.320	147	417.792
36x108	3888	11953	23760	3.926.768	165	528.768
40x120	4800	14721	29280	5.359.400	183	652.800
48x144	6912	21121	42048	9.191.096	219	940.032

Tabela 5.1: Exemplo A – Características das Malhas

A partir da Tab. 5.2 pode-se determinar o *speed-up* e eficiência alcançados na implementação proposta, como mostrado na Tab. 5.3 e ilustrado nos gráficos das Figs. 5.2 e 5.3, a partir dos quais podem-se fazer duas considerações importantes: o *speed-up*, de uma maneira geral, é tanto maior quanto maior é o número de processadores, contudo, não se pode afirmar que isto ocorra continuamente, uma vez

malha	direto	Gradientes Conjugado EPE						
		sequêncial		iterações	paralelo			
	SLC	SLC	SPARC		3	6	9	12
06x18	4	67	28	339	52	87	77	142
08x24	7	161	68	460	115	111	136	146
10x30	15	322	135	586	194	181	217	291
12x36	29	571	241	717	334	267	236	281
16x48	68	1391	582	978	609	485	498	478
18x54	97	1984	834	1107	812	559	544	724
24x72	295	4774	2118	1496	1864	1176	1154	1268
32x96		11452	4853	2016	4136	2452	2063	2198
36x108		*15904	6915	2278	6023	3473	2636	3279
40x120		*23078	10034	2539	8648	4551	4489	3159
48x144		*38295	16650	3070	17764	10543	7396	4852

(*) – estimado

Tabela 5.2: Exemplo A – Tempos Médios de Execução

que, cotejando-se o gráfico da Fig. 5.3, nota-se que as curvas que representam cada conjunto de processadores tendem a crescer, passar por um ponto de máxima eficiência e a partir dali decrescer. A existência de um ponto de máximo, é indicativo de que se conseguiu equilibrar tanto a carga de trabalho em cada processador, bem como manteve-se a comunicação em níveis que não prejudicam a performance do algoritmo. A única diferença entre as curvas é que o ponto de máximo ocorre em ocasiões diferentes, i.e., existe sempre uma situação, que depende do tamanho da malha, na qual se consegue obter máxima eficiência para determinado número de processadores. Salienta-se, que neste caso, a eficiência nunca ultrapassará a 100%. Isto acontece porque existe uma fração sequencial inerente ao programa, limitando o paralelismo que se pode obter.

A segunda consideração diz respeito à melhor eficiência obtida. Verifica-se prontamente, por inspeção do gráfico da Fig. 5.3, que a melhor eficiência é alcançada utilizando-se o menor número de processadores. Isto se deve ao fato de que nesta situação, se mantêm os processadores com a máxima carga de trabalho, e ainda, com um mínimo de comunicação interprocessos. Isto pode ser explicado porque a relação entre o tempo de computação e o tempo de comunicação é elevada. As tabelas a seguir registram a variação da quantidade de comunicação com o crescimento do tamanho do problema, bem como, com o aumento no número de divisões da malha.

A observação do conteúdo destas tabelas, revela que o critério adotado para a divisão da malha em grupos foi satisfatório, mantendo um certo equilíbrio de carga em cada processador, i.e. com um mínimo de acoplamento entre os processos.

malha	*speed-up*				eficiência (%)			
	3	6	9	12	3	6	9	12
06x18	1,3	0,8	0,9	0,5	42,9			
08x24	1,4	1,5	1,2	1,1	46,7	24,2	13,2	9,2
10x30	1,7	1,8	1,5	1,1	55,0	29,7	16,5	9,2
12x36	1,7	2,1	2,4	2,0	57,0	35,6	26,9	16,9
16x48	2,3	2,9	2,8	2,9	76,1	47,8	31,0	24,3
18x54	2,4	3,5	3,6	2,7	81,4	59,2	40,5	22,8
24x72	2,6	4,1	4,1	3,8	85,4	67,7	46,0	31,4
32x96	2,8	4,7	5,6	5,2	92,3	77,8	61,7	43,4
36x108	2,6	4,6	6,0	4,9	88,0	76,3	67.0	40,4
40x120	2,7	5,1	5,1	7,3	90,0	84,5	57,1	60,1
48x144	2,2	3,6	5,2	7,9	71,9	60,5	57,5	65,8

Tabela 5.3: Exemplo A - *Speed-up* e Eficiência Alcançados

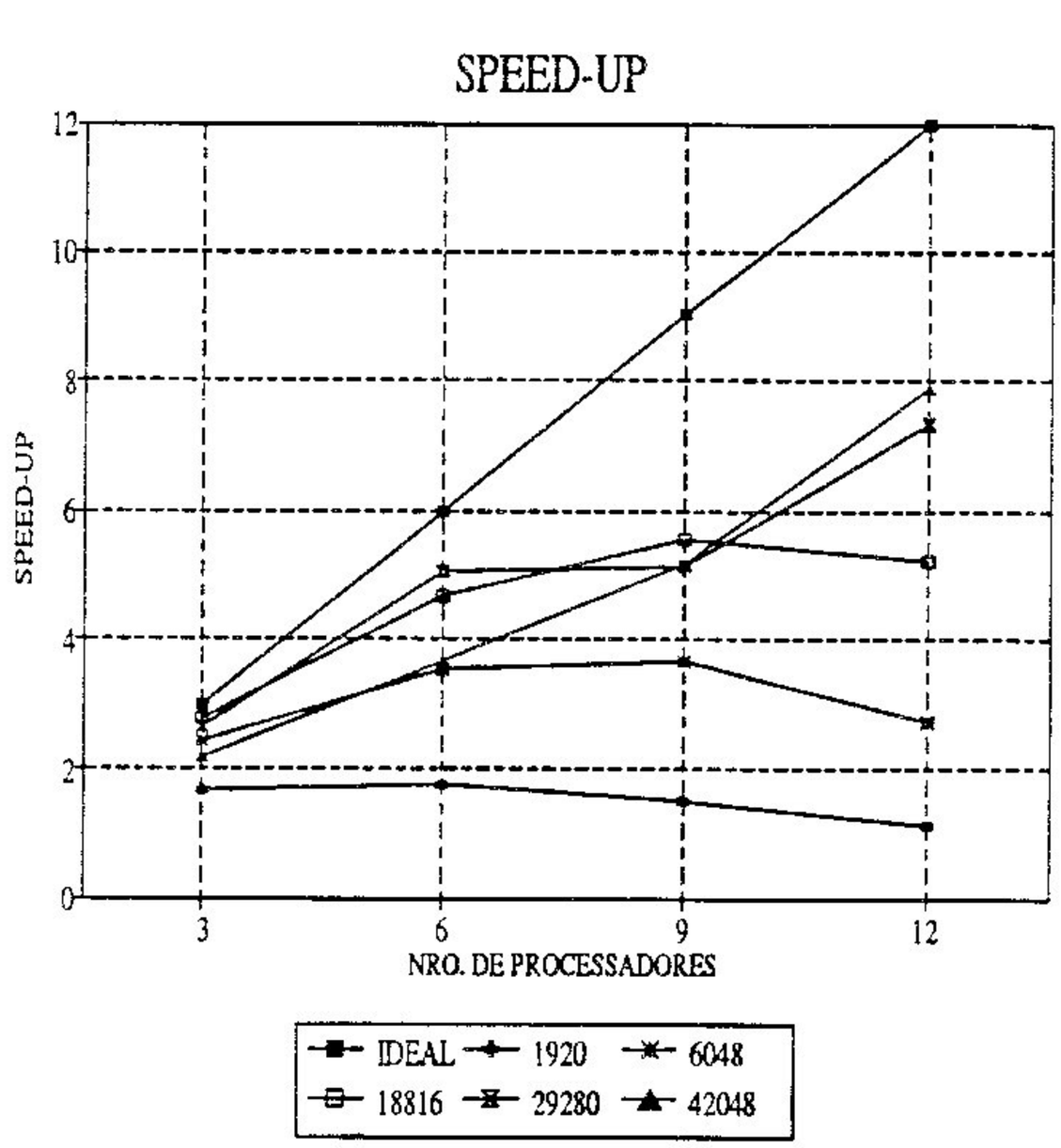


Figura 5.2: Exemplo A – Gráfico do *Speed-up*

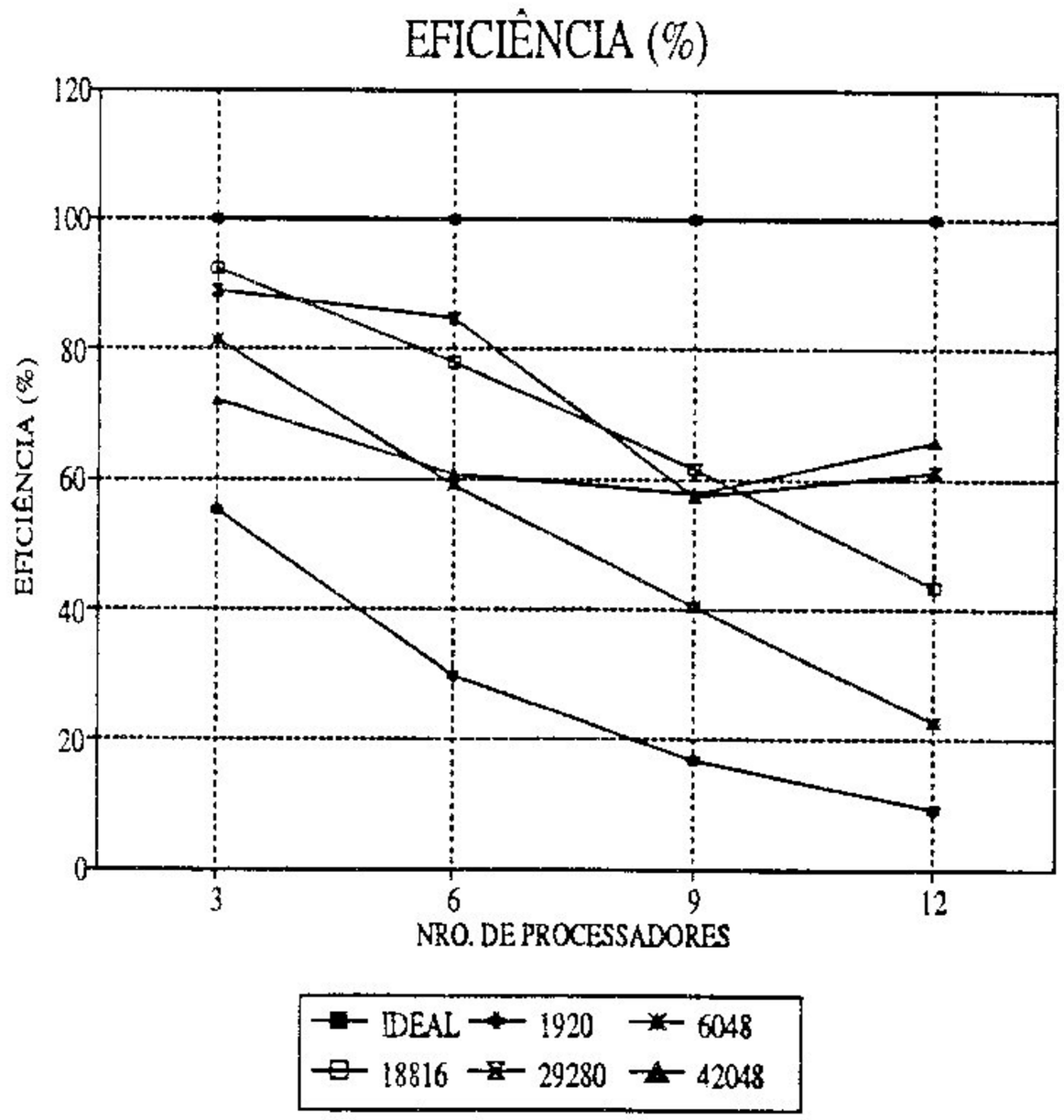


Figura 5.3: Exemplo A – Gráfico da Eficiência

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	36	133	13	214	26	1344
	2	36	133	26	214	52	
	3	36	133	26	240	52	
6	1	18	73	13	94	26	3984
	2	18	73	26	94	52	
	3	18	73	26	94	52	
	4	18	73	26	94	52	
	5	18	73	26	94	52	
	6	18	73	65	120	130	
9	1	12	53	13	54	26	6624
	2	12	53	26	54	52	
	3	12	53	26	54	52	
	4	12	53	26	54	52	
	5	12	53	26	54	52	
	6	12	53	26	54	52	
	7	12	53	26	54	52	
	8	12	53	26	54	52	
	9	12	53	104	80	208	
12	1	9	33	13	14	26	9264
	2	9	33	26	14	52	
	3	9	33	26	14	52	
	4	9	33	26	14	52	
	5	9	33	26	14	52	
	6	9	33	26	14	52	
	7	9	33	26	14	52	
	8	9	33	26	14	52	
	9	9	33	26	14	52	
	10	9	33	26	14	52	
	11	9	33	26	14	52	
	12	9	153	143	280	286	

Tabela 5.4: Exemplo A – malha 06x18: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	64	225	17	382	34	1728
	2	64	225	34	382	68	
	3	64	225	34	416	68	
6	1	32	121	17	174	34	5136
	2	32	121	34	174	68	
	3	32	121	34	174	68	
	4	32	121	34	174	68	
	5	32	121	34	174	68	
	6	32	121	85	208	170	
9	1	22	69	17	70	34	8544
	2	22	69	34	70	68	
	3	22	69	34	70	68	
	4	22	69	34	70	68	
	5	22	69	34	70	68	
	6	22	69	34	70	68	
	7	22	69	34	70	68	
	8	22	69	34	70	68	
	9	16	225	136	416	272	
16	1	16	69	17	70	34	11952
	2	16	69	34	70	68	
	3	16	69	34	70	68	
	4	16	69	34	70	68	
	5	16	69	34	70	68	
	6	16	69	34	70	68	
	7	16	69	34	70	68	
	8	16	69	34	70	68	
	16	16	69	34	70	68	
	10	16	69	34	70	68	
	11	16	69	34	70	68	
	12	16	69	187	104	374	

Tabela 5.5: Exemplo A – malha 08x24: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	100	341	21	598	42	2112
	2	100	341	42	598	84	
	3	100	341	42	640	84	
6	1	50	181	21	278	42	6128
	2	50	181	42	278	84	
	3	50	181	42	278	84	
	4	50	181	42	278	84	
	5	50	181	42	278	84	
	6	50	181	105	320	210	
9	1	30	117	21	150	42	10464
	2	30	117	42	150	84	
	3	30	117	42	150	84	
	4	30	117	42	150	84	
	5	30	117	42	150	84	
	6	30	117	42	150	84	
	7	30	117	42	150	84	
	8	30	117	42	150	84	
	9	60	213	168	384	336	
12	1	25	85	21	86	42	14640
	2	25	85	42	86	84	
	3	25	85	42	86	84	
	4	25	85	42	86	84	
	5	25	85	42	86	84	
	6	25	85	42	86	84	
	7	25	85	42	86	84	
	8	25	85	42	86	84	
	9	25	85	42	86	84	
	10	25	85	42	86	84	
	11	25	85	42	86	84	
	12	25	277	231	512	462	

Tabela 5.6: Exemplo A – malha 10x30: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	144	481	25	862	50	2432
	2	144	481	50	862	100	
	3	144	481	50	912	100	
6	1	72	253	25	406	50	7440
	2	72	253	50	406	100	
	3	72	253	50	406	100	
	4	72	253	50	406	100	
	5	72	253	50	406	100	
	6	72	253	125	456	250	
9	1	48	177	25	254	50	12384
	2	48	177	50	254	100	
	3	48	177	50	254	100	
	4	48	177	50	254	100	
	5	48	177	50	254	100	
	6	48	177	50	254	100	
	7	48	177	50	254	100	
	8	48	177	50	254	100	
	9	48	177	200	304	400	
12	1	36	139	25	178	50	17328
	2	36	139	50	178	100	
	3	36	139	50	178	100	
	4	36	139	50	178	100	
	5	36	139	50	178	100	
	6	36	139	50	178	100	
	7	36	139	50	178	100	
	8	36	139	50	178	100	
	9	36	139	50	178	100	
	10	36	139	50	178	100	
	11	36	139	50	178	100	
	12	36	139	275	228	550	

Tabela 5.7: Exemplo A – malha 12x36: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
1	1	256	833	33	1534	66	3264
	2	256	833	66	1534	132	
	3	256	833	66	1600	132	
6	1	128	433	33	734	66	9744
	2	128	433	66	734	132	
	3	128	433	66	734	132	
	4	128	433	66	734	132	
	5	128	433	66	734	132	
	6	128	433	165	800	330	
12	1	80	283	33	434	66	16624
	2	80	283	66	434	132	
	3	80	283	66	434	132	
	4	80	283	66	434	132	
	5	80	283	66	434	132	
	6	80	283	66	434	132	
	7	80	283	66	434	132	
	8	80	283	66	434	132	
	9	128	433	264	800	528	
	1	64	233	33	334	66	17328
	2	64	233	66	334	132	
	3	64	233	66	334	132	
	4	64	233	66	334	132	
	5	64	233	66	334	132	
	6	64	233	66	334	132	
	7	64	233	66	334	132	
	8	64	233	66	334	132	
	9	64	233	66	334	132	
	10	64	233	66	334	132	
	11	64	233	66	334	132	
	12	64	233	363	400	726	

Tabela 5.8: Exemplo A – malha 16x48: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
1	1	324	1045	37	1942	74	3648
	2	324	1045	74	1942	148	
	3	324	1045	74	2016	148	
6	1	162	541	37	934	74	10896
	2	162	541	74	934	148	
	3	162	541	74	934	148	
	4	162	541	74	934	148	
	5	162	541	74	934	148	
	6	162	541	185	1008	370	
9	1	108	373	37	598	74	18144
	2	108	373	74	598	148	
	3	108	373	74	598	148	
	4	108	373	74	598	148	
	5	108	373	74	598	148	
	6	108	373	74	598	148	
	7	108	373	74	598	148	
	8	108	373	74	598	148	
	9	108	373	296	672	592	
12	1	81	261	37	374	74	25392
	2	81	261	74	374	148	
	3	81	261	74	374	148	
	4	81	261	74	374	148	
	5	81	261	74	374	148	
	6	81	261	74	374	148	
	7	81	261	74	374	148	
	8	81	261	74	374	148	
	9	81	261	74	374	148	
	10	81	261	74	374	148	
	11	81	261	74	374	148	
	12	81	597	407	1120	814	

Tabela 5.9: Exemplo A – malha 18x54: Balanço de Carga e Comunicação

:

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	576	1825	49	3454	94	4800
	2	576	1825	98	3454	196	
	3	576	1825	98	3552	196	
6	1	288	937	49	1678	98	14352
	2	288	937	98	1678	196	
	3	288	937	98	1678	196	
	4	288	937	98	1678	196	
	5	288	937	98	1678	196	
	6	288	937	245	1776	490	
9	1	192	641	49	1086	98	23904
	2	192	641	98	1086	196	
	3	192	641	98	1086	196	
	4	192	641	98	1086	196	
	5	192	641	98	1086	196	
	6	192	641	98	1086	196	
	7	192	641	98	1086	196	
	8	192	641	98	1086	196	
	9	192	641	392	1184	784	
12	1	144	493	49	790	98	34456
	2	144	493	98	790	196	
	3	144	493	98	790	196	
	4	144	493	98	790	196	
	5	144	493	98	790	196	
	6	144	493	98	790	196	
	7	144	493	98	790	196	
	8	144	493	98	790	196	
	9	144	493	98	790	196	
	10	144	493	98	790	196	
	11	144	493	98	790	196	
	12	144	493	539	888	1078	

Tabela 5.10: Exemplo A – malha 24x72: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	1024	3201	65	6142	130	6336
	2	1024	3201	130	6142	260	
	3	1024	3201	130	6272	260	
6	1	512	1633	65	3006	130	18960
	2	512	1633	130	3006	260	
	3	512	1633	130	3006	260	
	4	512	1633	130	3006	260	
	5	512	1633	130	3006	260	
	6	512	1633	325	3136	650	
9	1	342	1110	67	1956	134	32544
	2	342	1112	134	1956	268	
	3	342	1110	134	1952	268	
	4	342	1112	134	1956	268	
	5	342	1112	134	1956	268	
	6	342	1110	134	1952	268	
	7	342	1112	134	1956	268	
	8	342	1112	134	1956	268	
	9	336	1119	536	2104	1072	
12	1	256	849	65	1438	130	44208
	2	256	849	130	1438	260	
	3	256	849	130	1438	260	
	4	256	849	130	1438	260	
	5	256	849	130	1438	260	
	6	256	849	130	1438	260	
	7	256	849	130	1438	260	
	8	256	849	130	1438	260	
	9	256	849	130	1438	260	
	10	256	849	130	1438	260	
	11	256	849	130	1438	260	
	12	256	849	715	1568	1430	

Tabela 5.11: Exemplo A – malha 32x96: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	1296	4033	73	7774	146	6336
	2	1296	4033	146	7774	292	
	3	1296	4033	146	7920	292	
6	1	648	2053	73	3814	146	21264
	2	648	2053	146	3814	292	
	3	648	2053	146	3814	292	
	4	648	2053	146	3814	292	
	5	648	2053	146	3814	292	
	6	648	2053	365	3960	730	
9	1	432	1393	73	2494	146	35424
	2	432	1393	146	2494	292	
	3	432	1393	146	2494	292	
	4	432	1393	146	2494	292	
	5	432	1393	146	2494	292	
	6	432	1393	146	2494	292	
	7	432	1393	146	2494	292	
	8	432	1393	146	2494	292	
	9	432	1393	584	2640	1168	
12	1	324	1063	73	1834	146	49584
	2	324	1063	146	1834	292	
	3	324	1063	146	1834	292	
	4	324	1063	146	1834	292	
	5	324	1063	146	1834	292	
	6	324	1063	146	1834	292	
	7	324	1063	146	1834	292	
	8	324	1063	146	1834	292	
	9	324	1063	146	1834	292	
	10	324	1063	146	1834	292	
	11	324	1063	146	1834	292	
	12	324	1063	803	1980	1606	

Tabela 5.12: Exemplo A – malha 36x108: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	1600	4961	81	9598	162	7872
	2	1600	4961	162	9598	324	
	3	1600	4961	162	9760	324	
6	1	800	2521	81	4718	162	23568
	2	800	2521	162	4718	324	
	3	800	2521	162	4718	324	
	4	800	2521	162	4718	324	
	5	800	2521	162	4718	324	
	6	800	2521	405	4880	810	
9	1	520	1667	81	3010	162	39264
	2	520	1667	162	3010	324	
	3	520	1667	162	3010	324	
	4	520	1667	162	3010	324	
	5	520	1667	162	3010	324	
	6	520	1667	162	3010	324	
	7	520	1667	162	3010	324	
	8	520	1667	162	3010	324	
	9	640	2033	648	3904	1296	
12	1	400	1301	81	2278	162	49584
	2	400	1301	162	2278	292	
	3	400	1301	162	2278	292	
	4	400	1301	162	2278	292	
	5	400	1301	162	2278	292	
	6	400	1301	162	2278	292	
	7	400	1301	162	2278	292	
	8	400	1301	162	2278	292	
	9	400	1301	162	2278	292	
	10	400	1301	162	2278	292	
	11	400	1301	162	2278	292	
	12	400	1301	891	2440	1782	

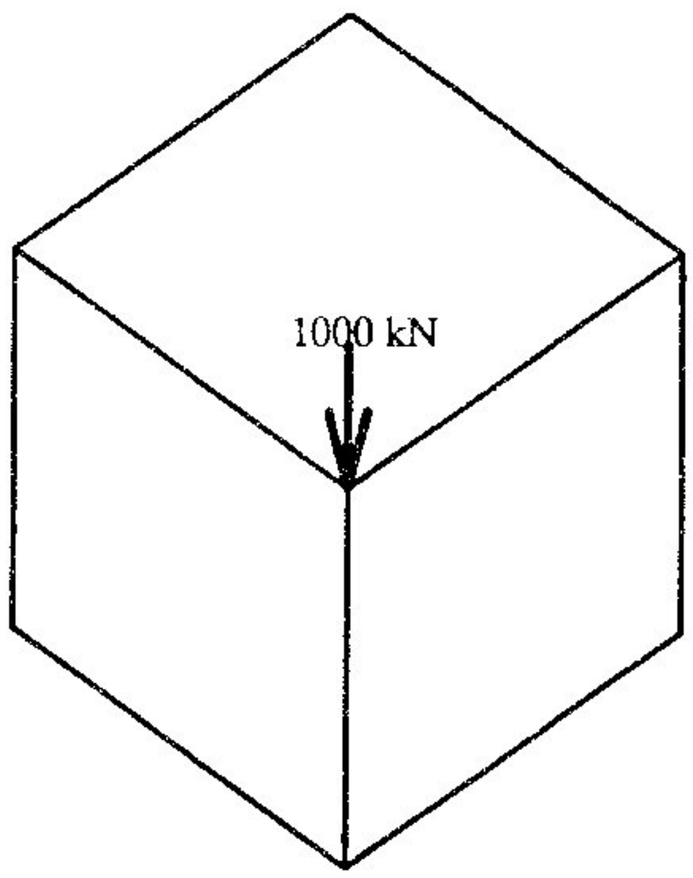
Tabela 5.13: Exemplo A – malha 40x120: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	2304	7105	97	13822	194	9408
	2	2304	7105	194	13822	388	
	3	2304	7105	194	14016	388	
6	1	1152	3601	97	6814	194	28176
	2	1152	3601	194	6814	388	
	3	1152	3601	194	6814	388	
	4	1152	3601	194	6814	388	
	5	1152	3601	194	6814	388	
	6	1152	3601	485	7008	970	
9	1	768	2433	97	4478	194	46944
	2	768	2433	144	4478	388	
	3	768	2433	144	4478	388	
	4	768	2433	144	4478	388	
	5	768	2433	144	4478	388	
	6	768	2433	144	4478	388	
	7	768	2433	144	4478	388	
	8	768	2433	144	4478	388	
	9	768	2433	776	4672	1552	
12	1	576	1849	97	3310	194	65712
	2	576	1849	194	3310	388	
	3	576	1849	194	3310	388	
	4	576	1849	194	3310	388	
	5	576	1849	194	3310	388	
	6	576	1849	194	3310	388	
	7	576	1849	194	3310	388	
	8	576	1849	194	3310	388	
	9	576	1849	194	3310	388	
	10	576	1849	194	3310	388	
	11	576	1849	194	3310	388	
	12	576	1849	1067	3504	2134	

Tabela 5.14: Exemplo A – malha 48x144: Balanço de Carga e Comunicação

5.3 Exemplo B

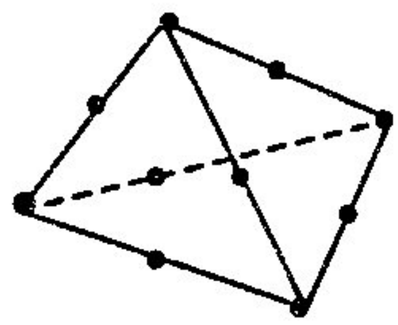
Este exemplo trata-se da análise de um cubo cujas arestas são iguais a 120 cm, suportando uma carga concentrada em uma extremidade, como mostrado na Fig. 5.4. As restrições de apoio estão aplicadas nas arestas que não aparecem no referido desenho. Verifica-se um pequeno desbalanceamento de carga relativo ao número de nós em cada processador, decorrente da falta de uma organização dos elementos dentro do domínio, que por sua vez decorre do algoritmo utilizado para a sua geração, que evidentemente, não prevê o particionamento da malha gerada.



E=205.000 Mpa

n=0,3

arestas = 120 cm



Elemento Utilizado

Figura 5.4: Exemplo B – Cubo

Da mesma forma como no exemplo anterior, são apresentadas tabelas que mostram os requisitos de armazenamento, os tempos de execução, e o *speed-up* e eficiência alcançados. Ressalta-se que não foi possível obter alguns tempos de execução, principalmente dos programas sequenciais, devido à falta de memória para conter as matrizes de rigidez dos elementos. Todavia, isto evidencia as vantagens de se utilizar processamento distribuído.

Os gráficos contidos nas Figs. 5.5 e 5.6 revelam pontos similares aos já notados, como: melhor eficiência utilizando-se poucos processadores, maior *speed-up* com mais processadores, e curvas de eficiência com pontos de máximo. Devido à forma como a malha de elementos é subdividida, verifica-se um aumento acentuado na comunicação. Entretanto, neste caso, o tempo de computação foi superior ao tempo gasto em comunicação, visto que as matrizes de elementos possuem ordem mais elevada, e daí, ainda obter-se boa eficiência.

malha	número de			matriz de rigidez		
	elementos	nós	equações	nro. entradas	banda média	elementos
04	490	811	2361	7 63.773	323	227.850
05	1061	1746	5148	2.552.976	496	493.365
06	1832	2949	8739	6.552.540	750	851.880
07	2781	4342	12903	12.368.424	959	1.293.165
08	4412	6867	20460	26.501.982	1295	2.051.580
09	6538	9939	29658	51.353.883	1732	3.040.170
10	8817	13188	39390			4.099.905

Tabela 5.15: Exemplo B – Características das Malhas

malha	direto	Gradientes Conjugado EPE						
		sequêncial		iterações	paralelo			
		SLC	SPARC		3	6	9	12
04		*696	301	237	279	204	229	255
05		*2024	880	326	774	481	465	599
06		*4177	1816	390	1561	971	821	788
07		*5180	2252	315	1965	1082	908	794
08		*13917	6051	529		2884	2262	2239
09				635		4947	3878	3324
10				511			3963	3414

(*) – estimado

Tabela 5.16: Exemplo B – Tempos Médios de Execução

malha	*speed-up*				eficiência (%)			
	3	6	9	12	3	6	9	12
04	2,5	3,4	3,0	2,7	83,2	56,9	33,8	22,7
05	2,6	4,2	4,4	3,4	87,2	70,1	48,4	28,2
06	2,7	4,3	5,1	5,3	89,2	71,7	56,5	44,2
07	2,6	4,8	5,7	6,5	87,9	79,8	63,4	54,4
08		4,8	6,2	6,2		80,4	68,4	51,8
09								
10								

Tabela 5.17: Exemplo B - *Speed-up* e Eficiência Alcançados

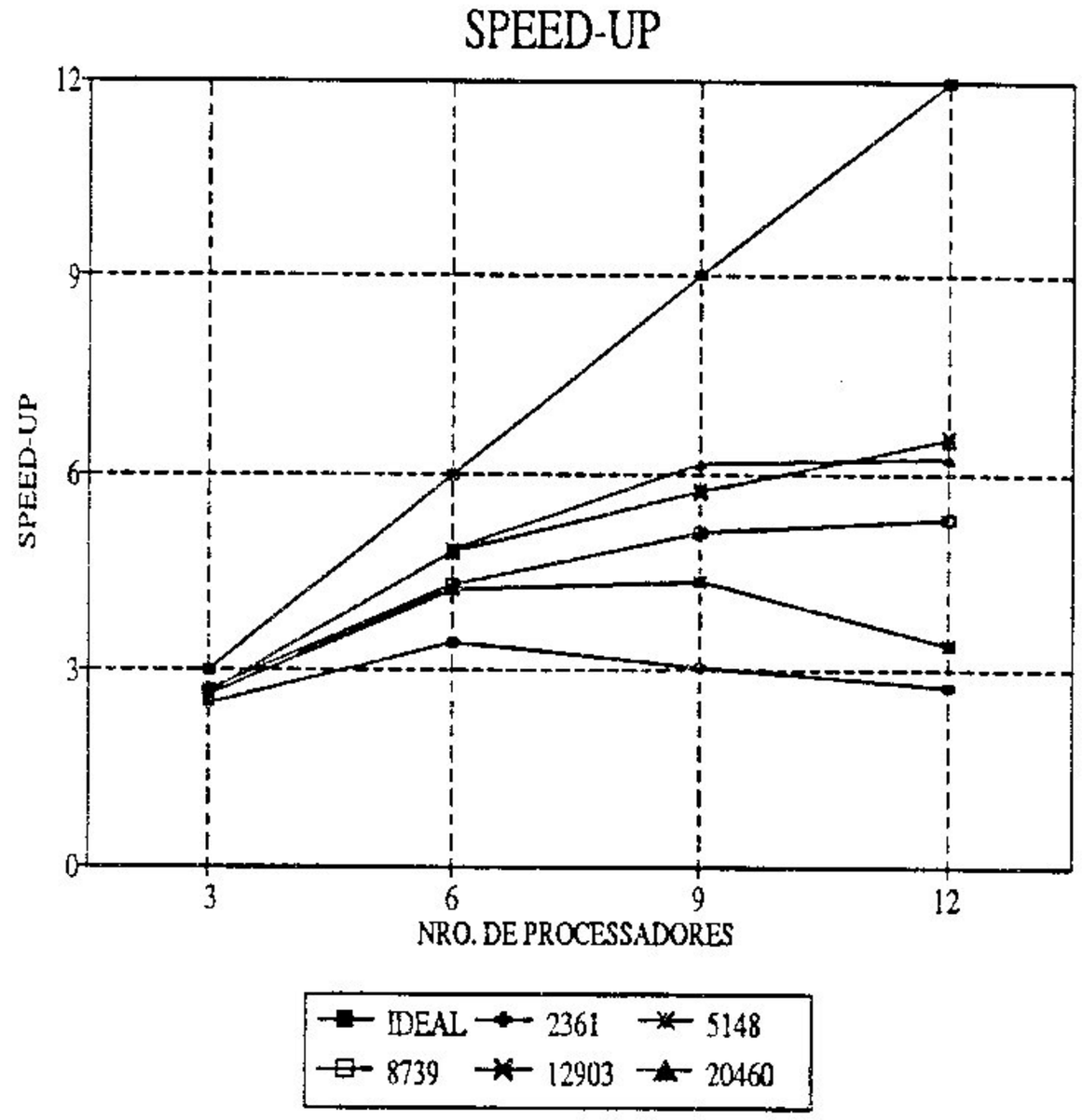


Figura 5.5: Exemplo B – Gráfico do *Speed-up*

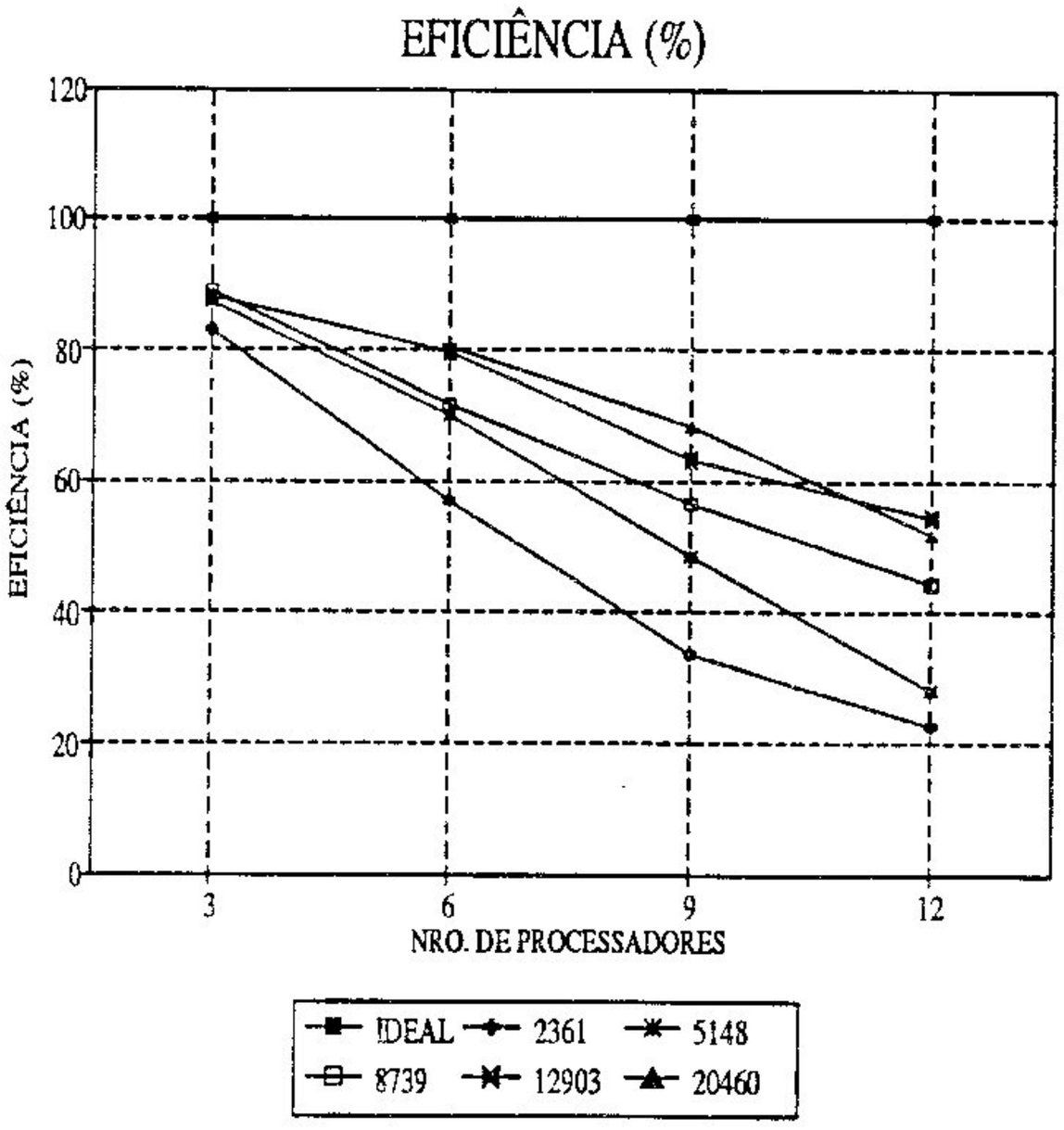


Figura 5.6: Exemplo B – Gráfico da Eficiência

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	163	352	145	570	432	19536
	2	163	380	265	342	783	
	3	164	348	265	666	783	
6	1	81	234	149	222	426	51072
	2	81	266	251	45	732	
	3	81	262	247	42	735	
	4	81	256	236	60	705	
	5	81	223	195	81	579	
	6	85	223	518	384	1527	
9	1	54	189	137	129	384	77232
	2	54	217	215	6	618	
	3	54	233	230	9	672	
	4	54	229	222	18	660	
	5	54	230	225	15	672	
	6	54	214	211	9	630	
	7	54	201	192	24	570	
	8	54	205	200	15	597	
	9	58	185	657	198	1938	
12	1	40	158	121	87	333	93792
	2	40	196	196	0	558	
	3	40	193	192	3	585	
	4	40	199	199	0	555	
	5	40	200	198	6	585	
	6	40	194	190	12	561	
	7	40	194	189	15	564	
	8	40	180	177	9	528	
	9	40	181	176	12	522	
	10	40	170	169	3	504	
	11	40	182	179	9	534	
	12	50	176	695	159	2046	

Tabela 5.18: Exemplo B – malha 04: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	353	702	193	1458	576	27072
	2	353	722	372	1047	1110	
	3	355	694	372	1533	1110	
6	1	176	422	165	705	495	76992
	2	176	444	344	297	1029	
	3	176	475	385	267	1149	
	4	176	457	376	243	1125	
	5	176	438	335	306	999	
	6	181	429	815	786	2544	
9	1	117	345	201	378	591	133200
	2	117	411	397	42	1179	
	3	117	398	383	42	1146	
	4	117	379	354	75	1059	
	5	117	384	357	78	1065	
	6	117	364	330	102	987	
	7	117	387	358	84	1068	
	8	117	421	403	54	1206	
	9	125	361	1320	354	3939	
12	1	133	305	218	219	630	164976
	2	133	335	316	57	924	
	3	133	329	323	18	969	
	4	133	352	341	30	1020	
	5	133	329	311	54	930	
	6	133	339	328	30	978	
	7	133	336	333	9	996	
	8	133	330	316	42	945	
	9	133	350	331	54	987	
	10	133	310	301	27	900	
	11	133	346	334	36	999	
	12	138	328	1454	243	4329	

Tabela 5.19: Exemplo B – malha 05: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	610	1175	307	2520	918	13136
	2	610	1216	588	1878	1758	
	3	612	1146	588	2583	1758	
6	1	305	695	282	1161	843	119712
	2	305	762	574	561	1716	
	3	305	754	559	582	1674	
	4	305	728	534	579	1599	
	5	305	734	547	558	1635	
	6	307	693	1360	1230	4068	
9	1	203	529	253	756	750	193728
	2	203	588	525	189	1566	
	3	203	628	575	156	1719	
	4	203	606	543	186	1626	
	5	203	583	527	168	1581	
	6	203	607	536	210	1605	
	7	203	602	536	195	1602	
	8	203	601	546	165	1635	
	9	208	539	2001	729	5985	
12	1	152	486	327	411	966	249552
	2	152	532	508	72	1509	
	3	152	505	470	102	1404	
	4	152	516	490	78	1467	
	5	152	519	499	57	1494	
	6	152	505	463	126	1389	
	7	152	497	465	93	1392	
	8	152	513	487	78	1458	
	9	152	508	481	78	1437	
	10	152	499	480	57	1437	
	11	152	563	538	75	1611	
	12	160	502	2374	414	7098	

Tabela 5.20: Exemplo B – malha 06: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1	927	1687	383	3819	1146	52944
	2	927	1726	720	3003	2157	
	3	927	1649	720	3924	2157	
6	1	463	1008	411	1704	1230	157152
	2	463	1089	778	930	2328	
	3	463	1055	719	999	2151	
	4	463	1018	679	1014	2034	
	5	463	1038	689	1044	2064	
	6	466	973	1798	1830	5382	
9	1	309	736	320	1167	951	237696
	2	309	771	600	510	1788	
	3	309	803	654	447	1959	
	4	309	822	685	408	2049	
	5	309	803	670	396	2004	
	6	309	811	671	417	2010	
	7	309	819	688	390	2061	
	8	309	806	671	405	2010	
	9	309	733	2515	1239	7524	
12	1	231	660	424	645	1245	333264
	2	231	771	733	114	2172	
	3	231	705	655	147	1959	
	4	231	689	619	210	1854	
	5	231	690	635	162	1899	
	6	231	687	616	210	1842	
	7	231	709	661	144	1980	
	8	231	711	661	147	1980	
	9	231	683	615	204	1845	
	10	231	711	641	207	1920	
	11	231	742	701	123	2100	
	12	240	676	3309	702	9888	

Tabela 5.21: Exemplo B – malha 07: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1						
	2						
	3						
6	1	735	1545	587	2775	1758	229248
	2	735	1626	1066	1671	3192	
	3	735	1644	1058	1755	3168	
	4	735	1620	1064	1665	3186	
	5	735	1605	1004	1794	3009	
	6	737	1487	2633	2913	7887	
9	1	490	1090	393	1992	1176	316656
	2	490	1114	793	960	2373	
	3	490	1186	881	912	2637	
	4	490	1219	937	843	2805	
	5	490	1222	928	882	2781	
	6	490	1198	926	813	2772	
	7	490	1184	901	846	2697	
	8	490	1139	843	885	2526	
	9	492	1121	3425	2070	10257	
12	1	367	930	424	1425	1263	472368
	2	367	1038	934	312	2793	
	3	367	1078	988	267	2958	
	4	367	1038	899	414	2691	
	5	367	1002	879	369	2634	
	6	367	1099	966	396	2892	
	7	367	1086	981	312	2937	
	8	367	1027	915	336	2742	
	9	367	1057	924	396	2766	
	10	367	1053	952	300	2853	
	11	367	1077	987	270	2961	
	12	375	962	4804	1275	14388	

Tabela 5.22: Exemplo B – malha 08: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1						
	2						
	3						
6	1	1089	2092	581	4419	1740	275952
	2	1089	2207	1241	2895	3717	
	3	1089	2250	1295	2856	3879	
	4	1089	2245	1307	2811	3915	
	5	1089	2246	1329	2742	3981	
	6	1093	2104	3204	4332	9597	
9	1	726	1568	606	2778	1815	457440
	2	726	1731	1300	1290	3894	
	3	726	1754	1343	1230	4023	
	4	726	1708	1258	1347	3768	
	5	726	1694	1241	1356	3717	
	6	726	1731	1277	1359	3825	
	7	726	1712	1280	1293	3834	
	8	726	1670	1232	1311	3690	
	9	730	1577	4954	2850	14838	
12	1	544	930	424	1425	1263	
	2	544	1038	934	312	2793	
	3	544	1078	988	267	2958	
	4	544	1038	899	414	2691	
	5	544	1002	879	369	2634	
	6	544	1099	966	396	2892	
	7	544	1086	981	312	2937	
	8	544	1027	915	336	2742	
	9	544	1057	924	396	2766	
	10	544	1053	952	300	2853	
	11	544	1077	987	270	2961	
	12	554	962	4804	1275	14388	

Tabela 5.23: Exemplo B – malha 09: Balanço de Carga e Comunicação

processos		elementos	nós		equações		bytes transferidos
			locais	comuns	locais	comuns	
3	1						
	2						
	3						
6	1						
	2						
	3						
	4						
	5						
	6						
9	1	979	2098	867	3573	2601	559056
	2	979	2269	1673	1785	5016	
	3	979	2218	1609	1824	4821	
	4	979	2164	1473	2070	4413	
	5	979	2129	1398	2184	4188	
	6	979	2131	1474	1968	4416	
	7	979	2229	1579	1947	4731	
	8	979	2194	1550	1929	4731	
	9	985	2039	6100	3831	18279	
12	1	734	1611	589	2946	1767	688512
	2	734	1657	1193	1389	3576	
	3	734	1756	1338	1251	4008	
	4	734	1817	1478	1017	4431	
	5	734	1823	1446	1128	4332	
	6	734	1777	1389	1161	4161	
	7	734	1814	1446	1101	4332	
	8	734	1809	1465	1029	4389	
	9	734	1843	1469	1119	4401	
	10	734	1717	1321	1188	3960	
	11	734	1666	1216	1347	3642	
	12	743	1651	7222	3072	21642	

Tabela 5.24: Exemplo B – malha 10: Balanço de Carga e Comunicação

5.4 Considerações Finais

De uma maneira geral, pode-se afirmar que a utilização de poucos processadores conduz a melhor eficiência, isto porque a relação entre o tempo de computação e o tempo de comunicação é grande. Por outro lado, obtêm-se melhores ganhos de velocidade com o aumento do número de processadores (divisão em grupos), como notado na seção 4.5.3. Portanto, conclui-se que, garantindo-se grande quantidade de computação em cada processador e mantendo-se um pequeno volume de comunicação, a utilização deste tipo de arquitetura conduz a excelentes resultados, como visto.

Ao longo da realização dos experimentos numéricos, foi observado que o número de iterações de um mesmo problema, variava de acordo com o número de processadores que se utilizava. A explicação para este fenômeno está no fato de que, as operações efetuadas no programa paralelo são executadas em uma ordem diferente daquelas correspondentes na versão sequêncial, o que resulta em propagação diferente do erro de truncamento, e daí uma pequena discrepância no número de iterações.

Outra observação, de ordem prática, diz respeito à carga das estações de trabalho. Procurou-se realizar todos os experimentos à noite e durante os fins de semana, visando uma carga mais baixa, uma vez que durante o dia, dependendo da utilização da rede, era inviável qualquer execução do programa. Entretanto, isto não é condição suficiente, uma vez que em sistemas multiusário e multitarefa como o UNIX, existe a possibilidade de outros processos estarem disputando a mesma máquina. Daí a necessidade de realizar várias execuções do mesmo problema a fim de se obter um média confiável.

Capítulo 6

Conclusões e Sugestões

Neste trabalho foi proposta uma implementação paralela do método dos gradientes conjugados para a solução de sistemas de equações oriundos da aplicação do método dos elementos finitos a problemas lineares de engenharia de estruturas, que mostrou ser eficiente e geral, uma vez que a arquitetura paralela utilizada não restringe o número de ligações entre processos, permitindo então, tratar problemas os mais variados.

A arquitetura utilizada, ou seja, o conjunto *workstations* e sistema PVM, se revelou adequado, poderoso, coerente e eficiente.

Foi identificado que a utilização de poucas máquinas resulta em melhor eficiência, e os pontos de estrangulamento deste tipo de abordagem se referem à dificuldade em poder-se obter uma partição da malha de elementos de modo que se tenha o mínimo de acoplamento entre os processos, diminuindo-se assim a quantidade de comunicação entre os mesmos.

O método dos gradientes conjugados se revelou robusto, tanto na adequação ao esquema de paralelização adotado, quanto na solução do sistema de equações. Ressalta-se no entanto, que outros esquemas de precondicionamento poderiam ser utilizados a fim de melhorar a taxa de convergência do MGC.

Como sugestões para trabalhos futuros, poderia-se empregar este esquema solução no método multimalha, ou aplicá-lo a problemas adaptativos.

Outra sugestão se refere à investigação de algoritmos de divisão de domínio automática, de modo a diminuir a contenção no programa paralelo.

Poderia-se também, investigar a implementação de outros métodos de solução de sistemas de equações, como o método frontal, utilizando-se este tipo de arquitetura.

A fim de procurar obter o máximo de ganho nas aplicações distribuídas da maneira proposta, seria extremamente útil a utilização de alocadores de carga dinâmicos, uma vez, como já citado no final do capítulo anterior, que se tem variações da carga em cada componente da máquina paralela virtual.

Bibliografia

[Almeida e Árabe, 1991] V. A. F. Almeida e J. N. C. Árabe. *Introdução à Supercomputação.* Livros Técnicos e Científicos Editora Ltda., 1ª edição, 1991.

[Alquati e Groehs, 1991] E. L. G. Alquati e A. G. Groehs. Estudo Comparativo de Precondicionadores na Solução de Sistemas de Equações por Gradientes Conjugados numa Formulação Elemento-por-Elemento. Em *Anais da XXV Jornadas Sul-Americanas de Engenharia Estrutural*, páginas 45–56, novembro 1991.

[Alquati, 1991] E. L. G. Alquati. Precondicionamento do Método dos Gradientes Conjugados numa Formulação Elemento-por-Elemento. Dissertação, Universidade Federal do Rio Grande do Sul, dezembro 1991.

[Arruda *et al.*, 1990] L. S. Arruda, A. L. G. A. Coutinho, e L. Landau. Um Estudo sobre Precondicionadores Elemento-por-Elemento para a Solução de Sistemas de Equações do Método dos Elementos Finitos. Em *Anais do XI Congresso Ibero Latino-Americano sobre Métodos Computacionais para Engenharia*, páginas 487–500, outubro 1990.

[Barr e Hickman, 1993] R. S. Barr e B. L. Hickman. Reporting Computacional Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. *Operations Research Society of America Journal on Computing*, 5:2–32, 1993.

[Barragy e Carey, 1988] E. Barragy e G. F. Carey. A Parallel Element-by-Element Solution Scheme. *International Journal for Numerical Methods in Engineering*, 26:2367–2382, 1988.

[Bathe, 1982] K. J. Bathe. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, 1982.

[Callioli *et al.*, 1983] C. A. Callioli, H. H. Domingues, e R. C. F. Costa. *Álgebra Linear e Aplicações.* Atual Editora Ltda, 4ª edição, 1983.

[Carey e Jiang, 1986] G. F. Carey e B.-N. Jiang. Element-by-Element Linear and Nonlinear Solution Schemes. *Communications in Applied Numerical Methods*, 2:145–153, 1986.

[Carnahan *et al.*, 1969] B. Carnahan, H. A. Luther, e J. O. Wilkes. *Applied Numerical Methods.* John Wiley & Sons, 1ª edição, 1969.

[Coutinho *et al.*, 1986] A. L. G. A. Coutinho, J. L. D. Alves, L. Landau, e N. F. F. Ebecken. Um Procedimento Elemento-por-Elemento para a Solução de Grandes Sistemas de Equações do MEF pelo Método dos Gradientes Conjugados. Em *Anais do VII Congresso Latino-Americano sobre Métodos Computacionais para Engenharia*, páginas 243–257, novembro 1986.

[Coutinho *et al.*, 1988] A. L. G. A. Coutinho, J. L. D. Alves, L. Landau, e N. F. F. Ebecken. Solução Iterativa de Sistemas de Equações do Método dos Elementos Finitos no Computador IBM 3090. Em *Anais do II Simpósio Brasileiro de Arquitetura de Computadores*, 1988.

[Coutinho *et al.*, 1991] A. L. G. A. Coutinho, J. L. D. Alves, L. Landau, N. F. F. ebecken, e L. M. Troina. Comparison of Lanczos and Conjugate Gradients for the Element-by-Element Solution of Finite Element Equations on the IBM 3090 Vector Computer. *Computer & Structures*, 39:47–55, 1991.

[Crisfield, 1986] M. A. Crisfield. *Finite Elements and Solution Procedures for Structural Analysis*, volume 1. Pineridge Press, 1986.

[Dongarra *et al.*, 1991] J. J. Dongarra, I. S. Duff, D. C. Sorensen, e H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM - Society for Industrial and Applied Mathematics, 1ª edição, 1991.

[Duff *et al.*, 1986] I. S. Duff, A. M. Erisman, e J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford Science Publication, 1986.

[Figueiredo, 1992] A. M. G. Figueiredo. Processo Adaptativo R-H Global Conjugado com Método Multimalha para Elementos Finitos. Dissertação, Universidade Federal de Minas Gerais, fevereiro 1992.

[Fox *et al.*, 1988] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, e D. W. Walker. *Solving Problems on Concurrent Processors*, volume 1, General Techniques and Regular Problems. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[Geist *et al.*, 1993] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, e V. Sunderam. *PVM 3.0 User's Guide and Reference Manual.* Oak Ridge National Laboratory, February 1993.

[George e Liu, 1981] A. George e J. W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Inc., 1ª edição, 1981.

[Golub e van Loan, 1989] G. H. Golub e C. F. van Loan. *Matrix Computations*. The John Hopkins University Press, 2ª edição, 1989.

[Gouveia, 1987] J. P. Gouveia. SDP: Sistema para Desenvolvimento de Programas Baseados no Método dos Elementos Finitos. Em *Anais do IX COBEM*, páginas 561–564, 1987.

[Hughes e Ferencz, 1987] T. J. R. Hughes e R. M. Ferencz. Large-Scale Vectorized Implicit Calculations in Solid Mechanics on a Cray X-MP/48 Utilizing EBE Preconditioned Conjugate gradients. *Computer Methods in Applied Mechanics and Engineering*, 61:215–248, 1987.

[Hughes *et al.*, 1983a] T. J. R. Hughes, I. Levit, e J. Winget. An Element-by-Element Solution Algorithm for Problems of Structural and Solid Mechanics. *Computer Methods in Applied Mechanics and Engineering*, 36:241–254, 1983.

[Hughes *et al.*, 1983b] T. J. R. Hughes, I. Levit, e J. Winget. Element-by-Element Implicit Algorithms for Heat Conduction. *Journal of Engineering Mechanics*, 109:576–585, 1983.

[Johnsson e Mathur, 1989] S. L. Johnsson e K. K. Mathur. Experience with the Conjugate Gradient Method for Stress Analysis on a Data Parallel Supercomputer. *International Journal for Numerical Methods in Engineering*, 27:523–546, 1989.

[Justino Filho e Ebecken, 1993] M. R. Justino Filho e N. F. F. Ebecken. Decomposição de Domínio Automática de Malhas do Método dos Elementos Finitos. Em *Anais do V Simpósio Brasileiro de Arquitetura de Computadores*, COPPE-UFRJ, setembro 1993.

[Loures *et al.*, 1992] E. F. Loures, G. E. P. Silva Júnior, e V. Almeida. Análise de Desempenho de Programas Paralelos em Redes de Workstations. Em *Anais do 4o Simpósio Brasileiro de Arquitetura de Computadores*, 1992.

[Loures, 1993] E. F. Loures. Implementação de políticas de balanceamento em ambientes distribuídos. Dissertação, Universidade Federal de Minas Gerais, outubro 1993.

[Mesquita, 1992] R. C. Mesquita. Método de Gradientes Conjugados com Pré-Condicionamento na Solução de Sistemas de Equações Geradas por Elementos Finitos. Em *Anais do Congresso Brasileiro de Eletromagnetismo Aplicado*, março 1992.

[Muller e Hughes, 1986] A. Muller e T. J. R. Hughes. Precondicionadores Elemento-por-Elemento y Globales. Una Perspectiva. *Revista Internacional de Métodos para Cálculo y Diseño en Ingeniería*, 2:27–41, 1986.

[Reddy, 1984] J. N. Reddy. *An Introduction to the Finite Element Method.* McGraw-Hill, 1984.

[Reddy, 1986] J. N. Reddy. *Applied Functional Analysis and Variational Methods in Enginneering.* McGraw-Hill, 1986.

[Stasa, 1985] F. L. Stasa. *Applied Finite Element Analysis for Engineers.* Holt, Rinehart and Winston, 1985.

[Sunderan, 1990] V. S. Sunderan. PVM: A Framework for Parallel Distributed Computing. Concurrency: Practice and Experience. 4(2):315–339, December 1990.

[Zienkiewicz e Morgan, 1983] O. C. Zienkiewicz e K. Morgan. *Finite Elements and Approximation.* John Wiley & Sons, 1983.

[Zienkiewicz e Taylor, 1989] O. C. Zienkiewicz e R. L. Taylor. *The Finite Element Method*, volume 1-2. McGraw-Hill, 4ª edição, 1989.