

IMPLEMENTAÇÃO DO MÉTODO DOS ELEMENTOS DE CONTORNO PARA
ELASTICIDADE TRIDIMENSIONAL EM AMBIENTE PARALELO DE MEMÓRIA
DISTRIBUÍDA

Leonardo de Souza Miers

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS
PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE
FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM
ENGENHARIA CIVIL.

Aprovada por:

Prof. José Claudio de Faria Telles, Ph.D.

Prof. Luis Paulo da Silva Barra, D.Sc.

Prof. José Antônio Fontes Santiago, D.Sc.

Prof. Elson Magalhães Toledo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 2003

MIERS, LEONARDO DE SOUZA

Implementação do Método dos Elementos
de Contorno para Elasticidade Tridimensional
em Ambiente Paralelo de Memória Distribuída
[Rio de Janeiro] 2003

X, 79 p. 29,7 cm (COPPE/UFRJ, M.Sc.,
Engenharia Civil, 2003)

Tese - Universidade Federal do Rio de
Janeiro, COPPE

1. Elementos de Contorno
2. Elasticidade
3. Computação Paralela

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

IMPLEMENTAÇÃO DO MÉTODO DOS ELEMENTOS DE CONTORNO
PARA ELASTICIDADE TRIDIMENSIONAL EM AMBIENTE PARALELO
DE MEMÓRIA DISTRIBUÍDA

Leonardo de Souza Miers

Abril/2003

Orientadores: José Claudio de Faria Telles

Luis Paulo da Silva Barra

Programa: Engenharia Civil

Este trabalho tem por objetivo o estudo de estratégias alternativas de implementação do Método dos Elementos de Contorno aplicado à elasticidade tridimensional em ambientes paralelos de memória distribuída instalados em redes de PC's. O código computacional utilizado é o CAV3D, em linguagem FORTRAN. São apresentadas diferentes formas de implementação das etapas de montagem do sistema de equações, sua resolução e determinação de resultados no domínio. Os resultados apresentados comprovam a eficiência de cada estratégia, evidenciando qual é a melhor para utilização em redes homogêneas e heterogêneas, bem como de todo o código paralelo.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMPLEMENTATION OF THE BOUNDARY ELEMENT METHOD
FOR THREE-DIMENSIONAL ELASTICITY IN PARALLEL ENVIRONMENT
OF DISTRIBUTED MEMORY

Leonardo de Souza Miers

April/2003

Advisors: José Claudio de Faria Telles

Luis Paulo da Silva Barra

Department: Civil Engineering

This work aims at the study of alternative strategies of implementation of the Boundary Element Method applied to three-dimensional elasticity in parallel environments of distributed memory PC networks. The computational code used is the CAV3D, in FORTRAN language. Different ways of implementing the equation system assemblage, its resolution and the determination of the domain results are presented. The results achieved show the efficiency of each strategy, evidencing which one is the best for use in homogeneous and heterogeneous networks, as well as for the whole parallel code.

Índice Analítico

Capítulo 1 - Introdução	1
1.1. Objetivos	1
1.2. Breve Histórico	1
1.3. Organização do Trabalho	6
Capítulo 2 - Conceitos Básicos	8
2.1. Introdução	8
2.2. Teoria da Elasticidade Tridimensional	8
2.3. Método dos Elementos de Contorno	10
2.4. CAV3D	12
2.5. GMRES	15
2.6. Terminologia	17
2.6.1. SIMD	18
2.6.2. MIMD	18
2.6.3. Memória Compartilhada	18
2.6.4. Memória Distribuída	19
2.6.5. SMP	20
2.6.6. MPP	20
2.6.7. Balanceamento de Carga Computacional	20
2.6.8. <i>Speed Up</i>	21
Capítulo 3 - Ambientes Paralelos de Implementação	22
3.1. Introdução	22
3.2. Hardware Utilizado	22
3.3. Software de Comunicação	24

Capítulo 4 - Metodologia	27
4.1. Introdução	27
4.2. Montagem do Sistema de Equações Lineares	28
4.3. Resolução do Sistema de Equações	31
4.4. Determinação dos Deslocamentos e Tensões no Domínio	33
4.5. Estratégia para Utilização em Redes Heterogêneas de PC's	33
Capítulo 5 - Resultados Numéricos	37
5.1. Introdução	37
5.2. Rede Homogênea – <i>Cluster</i> do NACAD	39
5.2.1. Códigos Sem Balanceamento de Carga Computacional	40
5.2.2. Códigos Com Balanceamento de Carga Computacional	40
5.3. Rede Heterogênea – NUMEC	54
Capítulo 6 - Considerações Finais	61
6.1. Conclusões	61
6.2. Sugestões	63
Referências Bibliográficas	65
Apêndice A - Sub-rotinas MPI Utilizadas	68
Apêndice B - Resultados Utilizados nos Gráficos	72

Índice de Figuras

Figura 1.1 - Arquitetura das 500 máquinas de maior capacidade no mundo entre 1993 e 2002	5
Figura 2.1 - Primeira aplicação do CAV3D – Barragem de Serra da Mesa	13
Figura 2.2 - Geração dos nós funcionais feita pelo CAV3D	15
Figura 2.3 - Esquema de memória compartilhada	19
Figura 2.4 - Esquema de memória distribuída	19
Figura 4.1 – Porcentagens dos tempos relativos às diferentes etapas no código seqüencial	27
Figura 4.2 - Distribuição seqüencial dos nós funcionais	29
Figura 4.3 - Distribuição cíclica dos nós funcionais	30
Figura 4.4 - Multiplicação matriz-vetor na etapa de resolução	32
Figura 4.5 - Reordenação do vetor em cada multiplicação na distribuição cíclica	32
Figura 4.6 - Esquema de posicionamento das linhas adicionais em cada processador	34
Figura 4.7 - Balanceamento de carga computacional na etapa de resolução	36
Figura 5.1 - Malhas dos problemas analisados	38
Figura 5.2 - Código sem balanceamento com montagem sequencial	41
Figura 5.3 - Código sem balanceamento com montagem cíclica	42
Figura 5.4 - Porcentagens dos tempos despendidos na etapa de resolução referentes à multiplicação matriz-vetor	43
Figura 5.5 - Código com balanceamento e montagem sequencial, sem passagem de mensagem	44
Figura 5.6 - Código com balanceamento, montagem seqüencial e passagem de mensagem	45
Figura 5.7 - Código com balanceamento, montagem cíclica e passagem de mensagem	46

Figura 5.8 - Relação entre tempos de comunicação e computação das faixas adicionais	47
Figura 5.9 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 4 processadores	48
Figura 5.10 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 4 processadores (cont.)	49
Figura 5.11 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 8 processadores	50
Figura 5.12 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 8 processadores (cont.)	51
Figura 5.13 - Relação entre os tempos totais de montagem cíclica e seqüencial	52
Figura 5.14 - Tempos de determinação dos resultados em pontos internos para o problema da cavidade	53
Figura 5.15 - Speed Up	53
Figura 5.16 - Tempos por iteração para o problema do prisma refinamento nível 1	55
Figura 5.17 - Tempos por iteração para o problema do prisma refinamento nível 2	56
Figura 5.18 - Tempos por iteração para o problema do prisma refinamento nível 3	57
Figura 5.19 - Tempos por iteração para o problema da cavidade	58
Figura 5.20 - Comparação entre os tempos totais de resolução e multiplicação matriz-vetor	59
Figura 5.21 - Comparação entre os tempos totais de resolução e multiplicação matriz-vetor (cont.)	60

Índice de Tabelas

Tabela 3.1 - Especificações do cluster do NACAD	23
Tabela 3.2 - Especificações da rede instalada no NUMEC	23
Tabela 5.1 - Códigos utilizados na análise e suas respectivas diferenças	37
Tabela 5.2 - Características dos exemplos utilizados	38
Tabela 5.3 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 1	55
Tabela 5.4 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 2	56
Tabela 5.5 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 3	57
Tabela 5.6 - Número de linhas por processador antes e após o balanceamento: cavidade	58
Tabela A.1 - Variáveis da sub-rotina MPI_init	68
Tabela A.2 - Variáveis da sub-rotina MPI_finalize	68
Tabela A.3 - Variáveis da sub-rotina MPI_comm_size	69
Tabela A.4 - Variáveis da sub-rotina MPI_comm_rank	69
Tabela A.5 - Variáveis da sub-rotina MPI_barrier	69
Tabela A.6 - Variáveis da sub-rotina MPI_gatherv	70
Tabela A.7 - Variáveis da sub-rotina MPI_bcast	70
Tabela A.8 - Variáveis da sub-rotina MPI_send	71
Tabela A.9 - Variáveis da sub-rotina MPI_recv	71
Tabela B.1 - Tempos de cada etapa do código sem balanceamento com montagem seqüencial	72
Tabela B.2 - Tempos de cada etapa do código sem balanceamento com montagem cíclica	73

Tabela B.3 - Tempos de cada etapa do código com balanceamento e montagem sequencial, sem passagem de mensagem	73
Tabela B.4 - Tempos de cada etapa do código com balanceamento, montagem sequencial e passagem de mensagem	74
Tabela B.5 - Tempos de cada etapa do código com balanceamento, montagem cíclica e passagem de mensagem	74
Tabela B.6 - Tempo de montagem em cada processador utilizando distribuição sequencial e cíclica dos nós funcionais: 4 processadores	75
Tabela B.7 - Tempo de montagem em cada processador utilizando distribuição sequencial e cíclica dos nós funcionais: 8 processadores	75
Tabela B.8 - Resultados no domínio para o problema da cavidade	75
Tabela B.9 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 1	76
Tabela B.10 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 2	77
Tabela B.11 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 3	78
Tabela B.12 – Tempos de cada iteração utilizando ou não balanceamento: cavidade	79
Tabela B.13 - Tempos da etapa de resolução e operação de multiplicação matriz-vetor utilizando ou não balanceamento de carga	79

Capítulo 1

Introdução

1.1. Objetivos:

Este trabalho tem por objetivo principal o estudo de diversas estratégias de implementação do Método dos Elementos de Contorno (MEC) aplicado à elasticidade tridimensional em ambientes paralelos de memória distribuída instalados em redes de computadores pessoais (PC's).

Obviamente, busca-se aqui também, através da pesquisa bibliográfica realizada, da metodologia utilizada, dos resultados obtidos e das conclusões formadas, fornecer subsídios teóricos e experimentais para futuras análises envolvendo os temas estudados.

1.2. Breve Histórico:

A utilização do MEC vem crescendo significativamente nos últimos anos, tendo sido aplicado na análise dos mais variados tipos de problemas de engenharia. Em vários destes, a capacidade dos recursos computacionais disponíveis é o fator que limita a precisão de sua análise. Até algum tempo atrás, apenas os supercomputadores¹

¹ Um supercomputador é, de forma simplificada, uma máquina de grande capacidade computacional composta de vários processadores de arquiteturas específicas, capazes de executar tarefas em paralelo, e de um esquema de memória compartilhada ou distribuída. Devido ao seu elevado custo, sua utilização ficou restrita a centros de pesquisas e empresas possuidoras de grandes recursos financeiros.

possuíam capacidade computacional que viabilizasse a análise de problemas mais sofisticados.

As primeiras implementações em paralelo do MEC se deram em máquinas de arquitetura SIMD² de memória compartilhada desenvolvidas por SYMM [1] em meados da década de 80 para um problema de potencial em um domínio circular. Nestas implementações, as contribuições de cada elemento eram mapeadas de acordo com o arranjo dos processadores, explorando o paralelismo na montagem da matriz do sistema no nível da integração, o que consistia na distribuição dos pontos de Gauss entre os processadores envolvidos. A solução do sistema de equações lineares resultante se dava por meio de bibliotecas de rotinas de métodos diretos, próprias para a arquitetura utilizada. As implementações para a arquitetura SIMD apresentavam como fator complicador o fato de não conseguirem tratar, com eficiência, situações onde a forma de computação de uma certa grandeza varia de acordo com o caso, tal qual o cálculo das integrais singulares e não-singulares do MEC, onde a primeira é normalmente calculada analiticamente e a segunda numericamente.

Este fator foi um dos que limitaram a evolução das estratégias de implementação em arquiteturas SIMD, além da mudança de rumo do desenvolvimento de *hardware*, que se voltou para arquiteturas MIMD.

Em arquiteturas MIMD, existem basicamente duas formas de se implementar em paralelo a geração da matriz do sistema de equações:

- distribuição dos elementos entre os processadores, fazendo com que cada processador calcule um certo número de colunas da matriz global do sistema;

² O significado desta e de outras terminologias se encontra na seção 2.6.

- distribuição dos pontos de colocação entre os processadores, fazendo com que cada processador calcule um certo número de linhas da matriz global do sistema;

Outra etapa, cuja paralelização se mostra interessante, refere-se à resolução do sistema de equações gerado. Os primeiros desenvolvimentos neste sentido se deram na paralelização de métodos diretos de solução, sendo a fatoração LU [11] uma das mais estudadas. Porém, para que este método seja implementado eficientemente, um esquema adequado de distribuição da matriz entre os processadores é fundamental para que se reduza ao mínimo o tempo ocioso de alguns. Para este caso, foi proposto um esquema de distribuição de linhas (ou colunas), conhecido como cíclico ou *wrapped* [12], porém este se mostrou ineficiente devido ao volume de comunicações envolvido. A evolução deste esquema aplicado à fatoração LU se deu ao combinar uma distribuição em bloco, o que reduz sensivelmente a comunicação requerida, com uma distribuição cíclica que possibilitasse balanceamento de carga, sendo este novo esquema conhecido como distribuição cíclica em blocos ou *torus-wrap*. Mesmo com esta técnica, o ganho de desempenho com o aumento do número de processadores envolvidos (escalabilidade) se mostrou limitado.

Outra estratégia estudada é baseada no conjunto de soluções iterativas que pode ser classificada como método de decomposição de domínio. Neste procedimento, o domínio do problema é artificialmente decomposto em subdomínios (ou sub-regiões) onde a compatibilidade entre eles é alcançada iterativamente através de sucessivas soluções dos problemas definidos em cada região. Nas implementações em paralelo, cada problema é designado a um processador que é responsável por sua solução. Porém, no caso do MEC, ao se introduzir estes contornos artificiais, e junto com eles alguma

forma de aproximação no domínio onde antes não havia, perde-se qualidade na solução final do sistema.

Uma outra estratégia existente é a baseada na paralelização de operações básicas de álgebra linear que compõem o núcleo de *solvers* iterativos e técnicas de pré-condicionamento. Recentemente, esta estratégia foi implementada no método iterativo GMRES [6] e utilizado na análise de problemas de elasticidade bidimensional pelo MEC [4]. Pela própria semelhança entre a implementação citada e este trabalho, utiliza-se neste a mesma estratégia.

Dentre todas as implementações do MEC em arquiteturas MIMD de memória distribuída, uma das mais recentes e de melhor relação custo-benefício são as em redes de PC's, constituindo os chamados *PC clusters* (que fazem parte da classe das NOW, do inglês *network of workstations*, ou COW, *cluster of workstations*).

Este novo recurso computacional teve seu desenvolvimento alavancado pelo aumento da capacidade computacional dos PC's a um baixo custo e pela evolução de técnicas eficientes de conexão destas máquinas, aliando o *hardware* existente com *softwares* de comunicação.

A popularização de sua utilização em aplicações científicas e sua constante evolução vem sendo estimulada pelo desempenho alcançado em diversas aplicações, sendo em várias delas comparável ao obtido em supercomputadores [2], e pelo baixo custo total das instalações, o que fez com que este recurso ganhasse popularidade em centros de pesquisa vinculados a entidades públicas e privadas. Além da instalação original, o baixo custo também se verifica quando se aumenta a capacidade computacional instalada (o que, na pior das hipóteses, corresponde à inclusão de alguns PC's a mais na rede) e quando há a necessidade de reposição de componentes defeituosos. Este recurso vem sendo utilizado em diversos centros de pesquisa

nacionais, como no Laboratório Nacional de Computação Científica – LNCC³ (Petrópolis/RJ) e no Núcleo de Atendimento em Computação de Alto Desempenho – NACAD⁴ (COPPE/UFRJ), e internacionais, como no *Goddard Space Flight Center* – GSFC⁵ (NASA - EUA) e no *Ohio Supercomputer Center* – OSC⁶ (EUA), dentre uma infinidade de outros.

Em função das vantagens apresentadas acima, o número de aplicações científicas implementadas em NOW's vem rapidamente aumentando, permitindo que a capacidade destas máquinas atinja patamares mais altos a cada dia. Abaixo, o gráfico⁷ da figura 1.1 mostra o número de máquinas por tipo de arquitetura que vêm compondo o conjunto das quinhentas máquinas de maior capacidade computacional no mundo desde junho de 1993 até junho de 2002. Atualmente, oitenta destas quinhentas máquinas são NOW's. Este número aumentou consideravelmente nos últimos cinco anos quando se compara com o crescimento do número de máquinas de outras arquiteturas, tendo em vista que até novembro de 1998 não havia sequer uma única máquina deste tipo no conjunto.

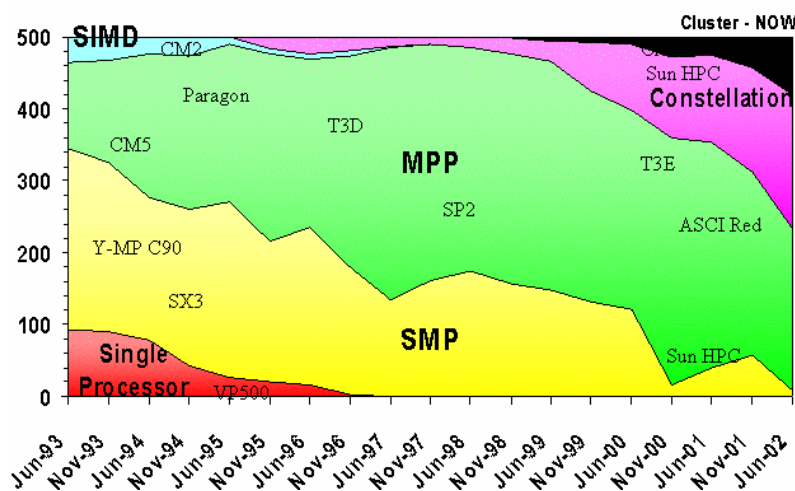


Figura 0.1 - Arquitetura das 500 máquinas de maior capacidade no mundo entre 1993 e 2002

³ www.lncc.br

⁴ www.nacad.ufrj.br

⁵ www.gsfc.nasa.gov

⁶ www.osc.ecu

⁷ Disponível no site www.top500.org; o significado de cada um dos termos se encontra na seção 2.6.

Freqüentemente, os *softwares* utilizados nestas implementações são de domínio público. Como sistema operacional, comumente utilizam-se distribuições do *Linux* em suas diversas versões, *Red Hat*, *SuSe*, *Debian*, entre outras. De forma a prover a troca de mensagens entre os PC's componentes da rede, comumente utilizam-se bibliotecas de sub-rotinas e funções de comunicação, como o PVM⁸ (*Parallel Virtual Machine*) e as de padrão MPI⁹ (*Message Passing Interface*) LAM MPI¹⁰ e MPICH¹¹. Em ambos os casos, os compiladores comumente utilizados podem ser também de domínio público, neste caso acompanhando o sistema operacional, sendo que os usados no *Linux* são o *g77* e o *gcc*, respectivamente para as linguagens FORTRAN 77 e C.

1.3. Organização do Trabalho:

O restante deste trabalho é apresentado da seguinte forma:

No segundo capítulo, serão descritos, de forma resumida, os conceitos envolvidos na elaboração do código, tanto o seqüencial quanto o paralelo. Será apresentada, resumidamente, a caracterização do problema dentro do contexto da Teoria da Elasticidade, a aplicação do MEC, o código computacional utilizado (CAV3D), o método iterativo GMRES, além de terminologias relativas à computação de alto desempenho.

No terceiro capítulo, são descritos os ambientes em que se procederam as executadas neste trabalho, tanto em relação ao *hardware* utilizado quanto ao *software*.

⁸ http://www.csm.ornl.gov/pvm/pvm_home.html

⁹ www.mpi-forum.org

¹⁰ www.lam-mpi.org

¹¹ www.mpich.com

No quarto capítulo, é apresentada a descrição da metodologia utilizada na implementação como um todo, explicitando e detalhando as estratégias utilizadas nas etapas de montagem do sistema de equações, sua resolução e determinação de resultados no domínio, seja para utilização em redes homogêneas ou heterogêneas.

Os exemplos utilizados e os resultados obtidos para eles serão expostos no quinto capítulo, no qual também se determina a eficiência das diversas estratégias utilizadas.

Conclusões e comentários gerais serão feitos no sexto capítulo.

Ao final, as referências bibliográficas utilizadas como base de pesquisa para a realização desta tese são apresentadas.

Na seção apêndice, estão explicadas as sub-rotinas MPI utilizadas (apêndice A), além de tabelas contendo os dados utilizados na confecção dos gráficos expostos no quinto capítulo (apêndice B).

Capítulo 2

Conceitos Básicos

2.1. Introdução:

Neste capítulo serão resumidamente descritos os conceitos sobre os quais este trabalho foi elaborado. Primeiramente, é apresentada a teoria sobre a qual se baseia a implementação computacional do Método dos Elementos de Contorno (MEC) [8], a Teoria da Elasticidade Tridimensional [7]. Posteriormente, é apresentado o MEC, método para a resolução das equações diferenciais apresentadas na teoria da elasticidade, além do CAV3D [16], que é o código computacional utilizado. Em seguida, é introduzida a descrição do GMRES [9], método iterativo para a resolução dos sistemas de equações gerado pelo MEC. Ao final, serão descritas algumas terminologias [9] que aparecem citadas neste trabalho.

2.2. Teoria da Elasticidade Tridimensional:

A equação descrita abaixo representa o equilíbrio estático no interior de um corpo:

$$\sigma_{ij,i} + b_j = 0 \quad (2.1)$$

onde σ_{ij} representa o tensor de tensões e b_j o vetor de forças de volume.

A condição de equilíbrio no contorno de um corpo é dada por

$$p_i = \sigma_{ij} n_j \quad (2.2)$$

onde p_i representa as componentes do vetor de forças de superfície e n_j os co-senos diretores do vetor normal à superfície apontando para fora do corpo.

Para um material elástico linear isotrópico onde não existem mudanças de temperatura, a Lei de Hooke fornece:

$$\sigma_{ij} = 2G\varepsilon_{ij} + \frac{2G\nu}{1-2\nu} \varepsilon_{kk} \delta_{ij} \quad (2.3)$$

onde G é o módulo de elasticidade transversal e ν o coeficiente de Poisson; ε_{ij} são as componentes do tensor de deformações específicas de Cauchy para pequenas deformações

$$\varepsilon_{ij} = \frac{1}{2} (u_{i,j} + u_{j,i}) \quad (2.4)$$

sendo u_i as componentes do vetor de deslocamentos.

Alternativamente, a equação (2.3) pode ser escrita na seguinte forma:

$$\sigma_{ij} = C_{ijkl} \varepsilon_{kl} \quad (2.5)$$

onde C_{ijkl} é o tensor isotrópico de quarta ordem de constantes elásticas:

$$C_{ijkl} = \frac{2G\nu}{1-2\nu} \delta_{ij} \delta_{kl} + G(\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}) \quad (2.6)$$

Se a equação (2.4) for substituída na (2.3), as tensões são agora obtidas em função das derivadas de deslocamentos. Esta equação pode, então, ser substituída em (2.1) e (2.2) para fornecer as equações de equilíbrio também em termos de deslocamentos, que são as chamadas equações de equilíbrio de Navier:

$$G u_{j,kk} + \frac{G}{1-2\nu} u_{k,kj} + b_j = 0 \quad (2.7)$$

cujas forças de superfície no contorno devem satisfazer a seguinte relação:

$$\frac{2G\nu}{1-2\nu} u_{k,k} n_i + G(u_{i,j} + u_{j,i}) n_j = p_i \quad (2.8)$$

2.3. Método dos Elementos de Contorno:

A aplicação do MEC à problemas de elasticidade é baseada na representação integral dos deslocamentos no interior de um corpo de domínio Ω e contorno Γ , conhecida como identidade de Somigliana para deslocamentos. Os deslocamentos em um ponto interno ξ são então calculados em termos de deslocamentos (u_j) e forças de superfície (p_j) no contorno Γ do corpo, como está colocado na equação abaixo onde, por simplicidade, não considerou-se os efeitos de forças de volume:

$$u_j(\xi) = \int_{\Gamma} u_{ij}^*(\xi, x) p_j(x) d\Gamma(x) - \int_{\Gamma} p_{ij}^*(\xi, x) u_j(x) d\Gamma(x) \quad (2.9)$$

Nesta equação, u_{ij}^* e p_{ij}^* representam, respectivamente, deslocamentos e forças de superfície na direção j do ponto campo x , de um problema definido como sendo o de uma carga unitária aplicada na direção i do ponto fonte ξ num domínio infinito. Esta solução, para o caso da aplicação em problemas de elasticidade, é conhecida como solução fundamental de Kelvin, que para o caso tridimensional é a apresentada abaixo:

$$u_{ij}^*(\xi, x) = \frac{1}{16\pi(1-\nu)Gr} [(3-4\nu)\delta_{ij} + r_i r_j] \quad (2.10)$$

$$p_{ij}^*(\xi, x) = \frac{-1}{8\pi(1-\nu)r^2} \left\{ [(1-2\nu)\delta_{ij} + 3r_i r_j] \frac{\partial r}{\partial n} - (1-2\nu)(r_i n_j - r_j n_i) \right\} \quad (2.11)$$

Antes da solução do problema, os valores de u_j e p_j são conhecidos apenas em parte do contorno. Para se determinar os demais valores ao longo de todo o contorno, a equação acima pode ser escrita para um ponto ξ no contorno do problema, através de um processo de limite [8], como:

$$c_{ij}(\xi) u_j(\xi) = \int_{\Gamma} u_{ij}^*(\xi, x) p_j(x) d\Gamma(x) - \int_{\Gamma} p_{ij}^*(\xi, x) u_j(x) d\Gamma(x) \quad (2.12)$$

onde $c_{ij}(\xi)$ é um coeficiente que depende da geometria do contorno no ponto ξ e a

integral à direita indica uma integração no sentido do Valor Principal de Cauchy.

O contorno pode ser discretizado em uma série de NE elementos onde as forças de superfície e os deslocamentos são interpolados entre N pontos nodais (funcionais), gerando um sistema de $3N$ equações:

$$(\hat{H} + C)\mathbf{u} = \mathbf{G}\mathbf{p} \quad (2.13)$$

onde C é uma matriz quase-diagonal de coeficientes que dependem da geometria do contorno no entorno dos pontos fonte. Ao se considerar

$$\mathbf{H} = \hat{H} + C \quad (2.14)$$

o sistema é reescrito na forma

$$\mathbf{H}\mathbf{u} = \mathbf{G}\mathbf{p} \quad (2.15)$$

onde \mathbf{H} é uma matriz não singular para o caso de problemas cujo domínio é infinito.

Como \mathbf{u} e \mathbf{p} são conjuntos de grandezas de diferente natureza (deslocamentos e forças de superfície, respectivamente), os termos em \mathbf{H} e \mathbf{G} podem diferir em ordem de grandeza. A partir do sistema de equações acima, é comum se adotar um único fator multiplicador, em termos do Módulo de Young (E) e do coeficiente de Poisson (ν), levando ao sistema modificado:

$$\mathbf{H}\mathbf{u} = \mathbf{G}'\mathbf{p}' \quad (2.16)$$

onde

$$\mathbf{G}' = \left(\frac{E}{1-\nu^2} \right) \mathbf{G} \quad \text{e} \quad \mathbf{p}' = \left(\frac{1-\nu^2}{E} \right) \mathbf{p} \quad (2.17)$$

Após a introdução das condições de contorno, o sistema de equações modificado pode ser reescrito, isolando-se os valores incógnitos no lado esquerdo (vetor \mathbf{x}) como:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (2.18)$$

Após a resolução do sistema denso e não-simétrico acima, todos os deslocamentos e forças de superfície são conhecidos no contorno e uma versão discreta

da equação (2.9) pode ser utilizada para calcular os valores requeridos em qualquer ponto ξ do domínio Ω .

Como as integrações ao longo dos elementos que não contêm o ponto fonte são usualmente calculadas através de quadratura Gaussiana, a implementação computacional para a geração do sistema final de equações é composta basicamente de três *loops* aninhados: o interno sobre os pontos de Gauss, o intermediário sobre os elementos e o externo sobre os pontos de colocação ξ .

A equação (2.9) é uma representação contínua dos deslocamentos em pontos do domínio. Logo, as componentes de tensões em pontos internos podem ser determinadas se a derivarmos em relação às coordenadas de ξ para obtermos as deformações e substituímos o resultado em (2.3). A expressão final é a seguinte:

$$\sigma_{ij}(\xi) = \int_{\Gamma} u_{ijk}^*(\xi, x) p_k(x) d\Gamma(x) - \int_{\Gamma} p_{ijk}^*(\xi, x) u_k(x) d\Gamma(x) \quad (2.19)$$

onde

$$u_{ijk}^*(\xi, x) = \frac{1}{8\pi(1-\nu)r^2} \left[(1-2\nu)(r_{,k}\delta_{ij} + r_{,j}\delta_{ki} - r_{,i}\delta_{jk}) + 3r_{,i}r_{,j}r_{,k} \right] \quad (2.20)$$

$$p_{ijk}^*(\xi, x) = \frac{G}{4\pi(1-\nu)r^3} \left\{ 3 \frac{\partial r}{\partial n} \left[(1-2\nu)\delta_{ij}r_{,k} + \nu(\delta_{ik}r_{,j} + \delta_{jk}r_{,i}) - 5r_{,i}r_{,j}r_{,k} \right] \right. \\ \left. + 3\nu(n_i r_{,j} r_{,k} + n_j r_{,i} r_{,k}) + (1-2\nu)(3n_k r_{,i} r_{,j} + n_j \delta_{ik} + n_i \delta_{jk}) - (1-4\nu)n_k \delta_{ij} \right\} \quad (2.21)$$

2.4. CAV3D:

O CAV3D é um código computacional em linguagem FORTRAN 77 baseado no MEC para a análise estática de problemas de elasticidade tridimensional. O programa

foi desenvolvido originalmente para Furnas Centrais Elétricas S/A [16], com o objetivo de se analisar problemas envolvendo cavidades em meio infinito, sendo o primeiro o das cavidades do circuito do AHE da Barragem de Serra da Mesa, em Goiás, ilustrada na figura 2.1.

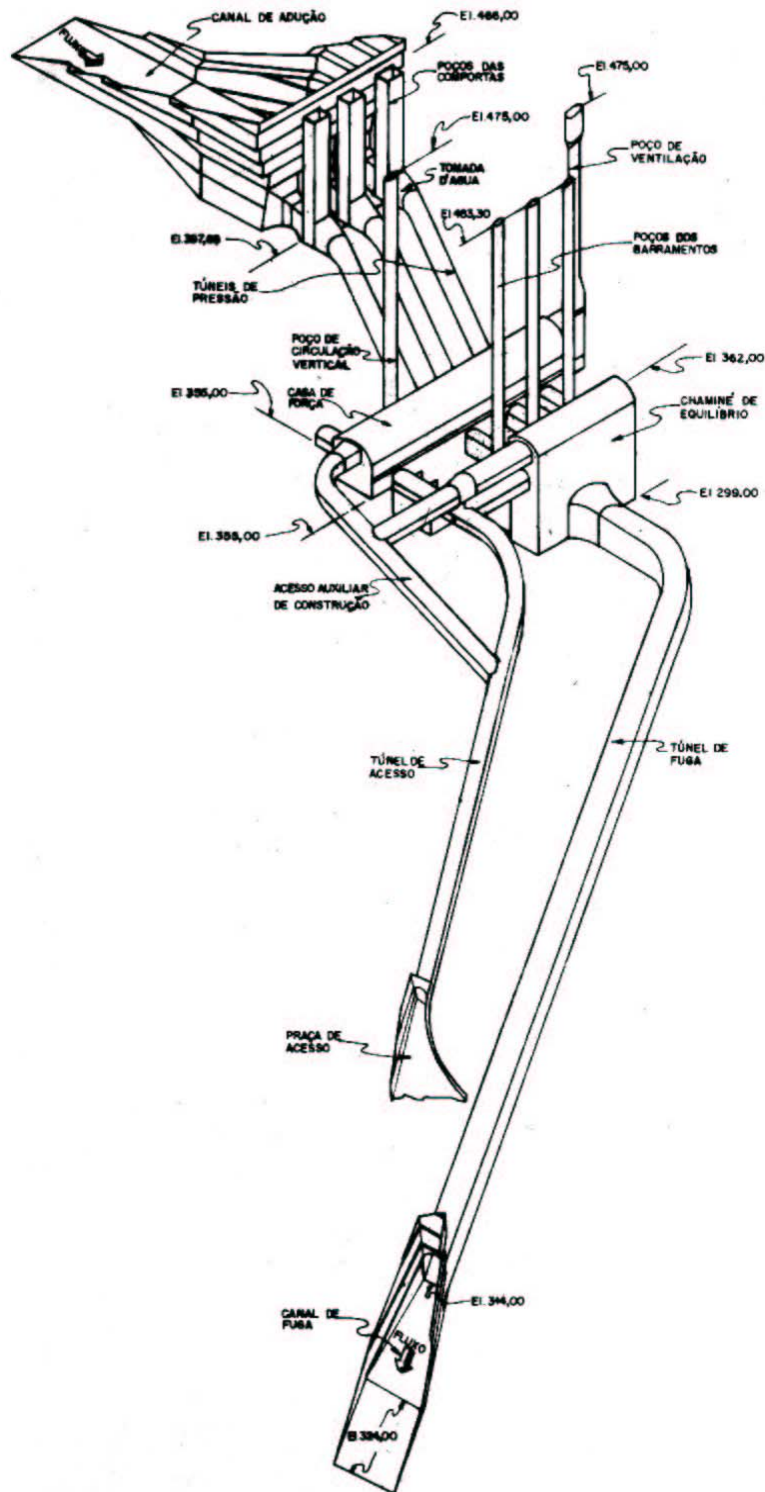


Figura 2.1 – Primeira aplicação do CAV3D – Barragem de Serra da Mesa

Basicamente, o CAV3D é composto das seguintes etapas:

- leitura dos dados de entrada – coordenadas nodais, características e conectividades dos elementos, condições de contorno prescritas e características dos materiais;
- impressão dos dados de entrada para conferência;
- geração dos nós funcionais da malha de elementos de contorno;
- montagem do sistema de equações algébricas lineares;
- resolução do sistema de equações;
- determinação de deslocamentos e tensões em pontos internos;
- impressão dos resultados – tensões e deslocamentos de todos os nós funcionais e pontos internos.

O CAV3D contém uma biblioteca de elementos, sendo eles:

- triangular ou quadrilateral plano constante,
- triangular ou quadrilateral plano linear,
- triangular ou quadrilateral curvo constante,
- triangular ou quadrilateral curvo linear,
- triangular ou quadrilateral curvo quadrático.

Quanto à continuidade, os elementos podem ser contínuos, parcialmente contínuos e descontínuos.

Após a entrada dos dados de entrada, o CAV3D gera todos os nós funcionais, cuja numeração é definida com relação à numeração dos elementos, de acordo com a figura 2.2.

Na etapa de montagem do sistema de equações, as integrais singulares são resolvidas por meio de algoritmos específicos [17] e, para o cálculo das integrais não-singulares, é utilizado o método da Quadratura Gaussiana com transformação de

coordenadas [8], cuja quantidade de pontos de integração utilizados é função da razão entre a área do elemento sobre o qual se está integrando e o quadrado da distância entre ele e o nó funcional fonte. Quanto menor for esta razão, maior o número de pontos a utilizar.

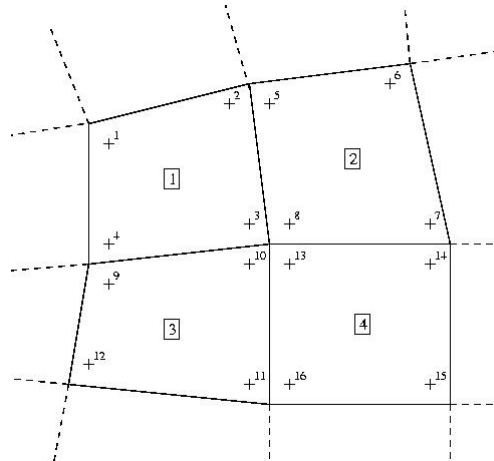


Figura 2.2 – Geração de nós funcionais feita pelo CAV3D

2.5. GMRES:

O algoritmo GMRES, proposto por **SAAD & SCHULTZ** [9], é considerado como sendo um dos mais eficientes e robustos métodos iterativos para a solução de sistemas de equações lineares não-simétricos.

Começando com uma estimativa inicial $\mathbf{x}^{(0)}$ para a solução do sistema de equações lineares, o método busca uma solução $\mathbf{x}^{(m)}$ em um subespaço $\mathbf{x}^{(0)} + K_m$ de dimensão m . Ele é incluído na classe dos métodos do subespaço de Krylov, sendo K_m o subespaço de Krylov, definido como:

$$K_m = \text{span}\{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, A^2\mathbf{r}^{(0)}, \dots, A^{m-1}\mathbf{r}^{(0)}\} \quad (2.22)$$

onde

$$\mathbf{r}^{(0)} = A\mathbf{x}^{(0)} - \mathbf{b} \quad (2.23)$$

Pela imposição da condição de ortogonalidade

$$\mathbf{b} - \mathbf{A}\mathbf{x}^{(m)} \perp \mathbf{A}K_m \quad (2.24)$$

pode-se mostrar que tal técnica minimiza, sobre todos os vetores $\mathbf{x}^{(0)} + K_m$, a norma do resíduo.

A estabilidade do GMRES é confirmada pelo fato da norma residual diminuir em cada iteração e da convergência ser atingida em, no máximo, n iterações, onde n é o número de incógnitas do sistema. Ambas as características foram apresentadas por **SAAD & SCHULTZ** em [9].

Objetivando reduzir a quantidade de memória para armazenamento, uma versão reinicializável do algoritmo foi desenvolvida, de forma que, se após um certo número de iterações a convergência não for obtida, o algoritmo usa a última aproximação calculada como sendo uma nova aproximação inicial $\mathbf{x}^{(0)}$ e reinicia o procedimento.

Esquemas de pré-condicionamento são obtidos quando o sistema original é multiplicado por uma aproximação da inversa da matriz do sistema, \mathbf{A}^{-1} , a qual é chamada matriz de pré-condicionamento. Uma das mais simples matrizes de pré-condicionamento, que vem se mostrando eficiente em outras aplicações [17,4], é a matriz formada pela inversa da matriz formada pela diagonal da matriz do sistema, $\mathbf{D}^{-1} = [\text{diag}\{\mathbf{A}\}]^{-1}$, sendo esta a utilizada neste trabalho. O sistema então se torna:

$$\mathbf{D}^{-1}\mathbf{A}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} \quad (2.25)$$

Abaixo está explicitado o algoritmo do GMRES reinicializável com pré-condicionamento utilizado no presente trabalho:

1. Calcular $\mathbf{r}^{(0)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)})$, $\beta = \|\mathbf{r}^{(0)}\|^2$ e $\mathbf{v}^{(1)} = \mathbf{r}^{(0)}/\beta$
2. Definir a matriz $\mathbf{H}_m = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$
3. Para $j = 1, 2, \dots, m$ faça:

4. Calcular $\mathbf{w}^{(j)} = \mathbf{D}^{-1} \mathbf{A} \mathbf{v}^{(j)}$
5. Para $i = 1, 2, \dots, j$ faça:
 6.
$$h_{ij} = (\mathbf{w}^{(j)})^T \mathbf{v}^{(i)}$$
 7.
$$\mathbf{w}^{(j)} = \mathbf{w}^{(j)} - h_{ij} \mathbf{v}^{(i)}$$
 8. Fim-faça
9.
$$h_{j+1,j} = \|\mathbf{w}^{(j)}\|^2$$
. Se $h_{j+1,j} \leq \text{tolerância}$ então $m = j$ e vá para 12
10.
$$\mathbf{v}^{(j+1)} = \mathbf{w}^{(j)} / h_{j+1,j}$$
11. Fim-faça
12. Calcular \mathbf{y}_m o que minimiza $\|\beta \mathbf{e}_1 - \overline{\mathbf{H}}_m \mathbf{y}\|^2$ e $\mathbf{x}^{(m)} = \mathbf{x}^{(0)} + \mathbf{V}_m \mathbf{y}_m$
13. Se satisfeito então para, senão $\mathbf{x}^{(0)} = \mathbf{x}^{(m)}$ e vá para 1

O aspecto computacional mais importante do algoritmo é que ele apresenta apenas uma multiplicação matriz-vetor por iteração (passo 4) e que, para os problemas analisados, é possível atingir a convergência adotando-se uma pequena dimensão m para o subespaço de Krylov. Porém, ao se diminuir m , pode-se aumentar o número de iterações necessárias para a convergência, além do número de possíveis de reinicializações.

A tolerância que aparece no passo 9 do algoritmo é um valor muito pequeno definido *a priori* pelo usuário (adotou-se 10^{-7} nas implementações deste trabalho).

2.6. Terminologia:

Nesta seção são introduzidos alguns termos relativos a ambientes de

implementação computacional que são mencionados no decorrer deste trabalho.

2.6.1. SIMD (*Single Instruction stream, Multiple Data stream*):

Modelo de execução em paralelo no qual todos os processadores executam a mesma operação no mesmo instante de tempo, porém cada um opera com seu próprio conjunto de dados. Este modelo se ajusta ao conceito de executar as mesmas operações em todos os elementos de um arranjo, logo ele é comumente associado com manipulações em matrizes e vetores.

2.6.2. MIMD (*Multiple Instruction stream, Multiple Data stream*):

Modelo de execução em paralelo no qual todos os processadores agem independentemente uns dos outros. Este modelo se ajusta ao conceito da decomposição de um programa para execução em paralelo. Este modelo é mais flexível que o SIMD, porém as interações entre processadores tendem a ser mais complicadas e menos eficientes.

2.6.3. Memória Compartilhada:

Esquema de arranjo de memória onde todos os processadores estão fisicamente conectados a apenas um grande bloco de memória. O acesso a ela pelos processadores é

feito de maneira direta, não necessitando haver passagem de mensagem. Seu arranjo é apresentado na figura 2.3.

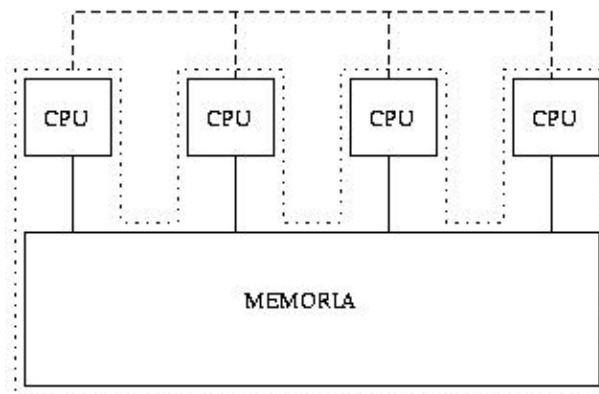


Figura 2.3 – Esquema de memória compartilhada

2.6.4. Memória Distribuída:

Esquema de disposição onde cada processador está fisicamente conectado a apenas uma parte da memória total disponível na máquina, sendo que o acesso a ela por outros processadores é feito mediante passagem de mensagem. Seu arranjo é apresentado na figura 2.4.

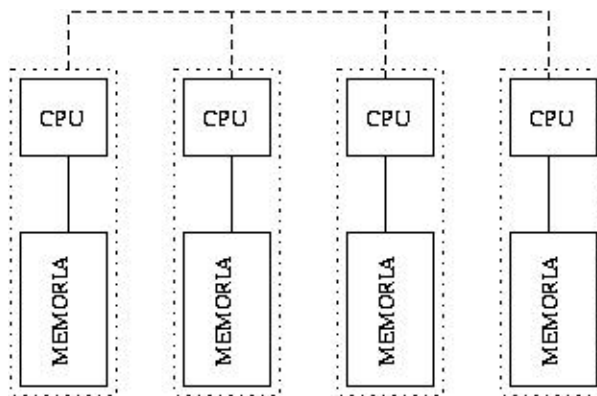


Figura 2.4 – Esquema de memória distribuída

2.6.5. SMP (*Symmetric Multi-Processor*):

A arquitetura SMP é, basicamente, a combinação de uma arquitetura MIMD e um sistema de memória compartilhada. É composta por um grupo de processadores com as mesmas propriedades trabalhando em conjunto, de forma que qualquer trecho de trabalho possa ser executado com o mesmo desempenho por qualquer um deles. No universo dos PC's, SMP geralmente se confunde com o termo MPS (Intel *Multi-Processor Specification*), que faz referência à tecnologia presente nas arquiteturas Intel Pentium Dual. Um conjunto de máquinas de arquitetura SMP é chamado *Constellation Cluster*.

2.6.6. MPP (*Massively Parallel Processor*):

Sistema composto de um grande número de processadores (às vezes até milhares), cada qual com sua respectiva memória, disco rígido e sistema operacional, utilizado para a solução de grandes problemas em computação. Um *PC cluster* pode ser visto como um pequeno sistema MPP.

2.6.7. Balanceamento de Carga Computacional:

Recurso utilizado num ambiente paralelo em situações onde a distribuição original de carga computacional entre os processadores envolvidos faz com que, em certas ocasiões, alguns fiquem ociosos enquanto outros ainda executam tarefas.

Consiste, basicamente, em estratégias para a redistribuição da carga visando equiparar os tempos úteis de cada processador, de forma a fazer com que eles estejam executando tarefas na maior parte do tempo, minimizando assim o tempo ocioso.

2.6.8. Speed Up:

Índice utilizado para calcular a eficiência de uma implementação em paralelo. Consiste na relação entre o tempo de execução do código em um processador (T_{1p}) e o tempo de execução em n processadores (T_{np}), sendo sua fórmula apresentada abaixo:

$$I_{speed-up} = \frac{T_{1p}}{T_{np}} \quad (2.26)$$

Quanto mais próximo o índice for de n , mais eficientemente a implementação.

Capítulo 3

Ambientes Paralelos de Implementação

3.1. Introdução:

Neste capítulo estão sucintamente descritos os ambientes paralelos nos quais foram realizados os estudos e as implementações para problemas de elasticidade tridimensional, tanto em termos de *hardware* utilizado quanto de *software* de comunicação.

3.2. *Hardware* Utilizado:

Como já mencionado anteriormente, o grande aumento de desempenho dos PC's a um custo relativamente baixo e a crescente evolução nas tecnologias de comunicação em redes contribuiu para a popularização dos *PC clusters* em aplicações científicas e de engenharia.

Quanto à arquitetura, as redes podem ser classificadas como homogêneas e heterogêneas. Redes homogêneas, ou *clusters* homogêneos, são aquelas em que todas as máquinas possuem as mesmas características, sejam em processamento ou em capacidade de memória. As que não se enquadram nesta definição são as chamadas redes heterogêneas, ou *clusters* heterogêneos. Neste trabalho, utilizaram-se para as implementações propostas os dois tipos de redes.

A rede homogênea utilizada é o *PC cluster* situado no Núcleo de Atendimento à Computação de Alto Desempenho – NACAD/UFRJ, cujas características são descritas abaixo:

Tabela 3.1 - Especificações do cluster do NACAD

Processadores:	32 processadores Pentium III 1.0 GHz (16 unidades Pentium-Dual)
Memória RAM:	512 Mbytes por unidade
Dispositivos de Comunicação:	Rede <i>Fast Ethernet</i> 100 Mbps
Sistema Operacional:	<i>Linux Red Hat</i>
Software de Comunicação:	MPICH

A rede heterogênea utilizada é a que se encontra no Núcleo de Pesquisa em Métodos Computacionais em Engenharia – NUMEC/UFJF, cujas características são descritas abaixo:

Tabela 3.2 - Especificações da rede instalada no NUMEC

Processadores: (entre parênteses os nomes que os identificam)	2 unidades Pentium IV 1.6 GHz (<i>itapemirim e tocantins</i>) 1 unidade Pentium IV 1.8 GHz (<i>xingu</i>) 1 unidade AMD <i>Athlon</i> XP 1600 1.33 GHz (<i>amazonas</i>)
Memória RAM:	256 Mbytes cada
Dispositivos de Comunicação:	Rede <i>Fast Ethernet</i> 100 Mbps
Sistema Operacional:	<i>Linux Red Hat</i>
Software de Comunicação:	LAM MPI

Um outro critério de classificação das redes é quanto sua utilização. Uma rede é considerada dedicada quando os recursos disponíveis são totalmente direcionados à execução de uma tarefa em particular, não havendo concorrência entre usuários. Em uma rede não dedicada, vários usuários podem estar executando tarefas ao mesmo tempo, sendo que os recursos computacionais disponíveis são divididos entre eles. Este tipo de classificação não é permanente, já que um mesmo cluster pode, em certos momentos, ser de execução dedicada e, em outros, não-dedicada. Um bom exemplo disso é a rede instalada no NUMEC que, ao longo do dia, a utilização em paralelo de

suas máquinas é feita junto à utilização local por parte dos demais usuários, sendo que somente à noite ou em momentos de pouco movimento a rede instalada pode ser utilizada de forma dedicada exclusivamente para a computação distribuída.

3.3. Software de Comunicação:

Em qualquer implementação computacional de aplicações desenvolvidas para ambiente paralelo de memória distribuída, é necessário que haja comunicação entre os processadores, de forma que cada um tenha acesso a informações contidas nas memórias dos demais.

Ao final da década de oitenta, foram criadas as primeiras bibliotecas de funções e sub-rotinas de comunicação de domínio público para uso geral, dentre as quais pode-se citar, entre uma infinidade de opções, o PVM e as de padrão MPI [15].

O PVM foi uma primeira tentativa de padronização das bibliotecas de comunicação, originalmente desenvolvido em 1989 por pesquisadores do *Oak Ridge National Laboratory*¹² (ORNL – EUA) para computação distribuída em ambientes heterogêneos, que tem como principais características sua portabilidade e o fato de trazer ao usuário a noção de se estar utilizando uma “máquina paralela”, isto é, um conjunto de máquinas ligadas por uma rede que simula ser uma única máquina de processamento paralelo.

O padrão MPI foi concebido durante 1993 e 1994 entre profissionais de centros de pesquisa e indústrias com o objetivo de, assim como o PVM, se estabelecer um padrão para as bibliotecas de passagem de mensagem, facilitando a criação de

¹² www.ornl.gov

aplicações paralelas portáteis. Apesar do desenvolvimento do MPI ter acontecido em conjunto com as montadoras de computadores, fato que provoca expectativas de melhor desempenho, o PVM pode apresentar em alguns casos performance similar [3]. O MPI foi o padrão utilizado no ambiente paralelo de memória distribuída onde as implementações descritas neste trabalho se deram. A escolha deste tipo de padrão deveu-se à diversos fatores: primeiro, o MPI é de domínio público e portátil; segundo, os bons resultados na implementação do MEC para elasticidade bidimensional alcançados por **BARRA et al.** [4] e dos resultados obtidos em comparações com outros padrões [3]; terceiro, a grande disponibilidade de recursos humanos e didáticos que subsidiaram o desenvolvimento do trabalho.

A forma de utilização deste padrão se dá por meio de chamadas a sub-rotinas contidas na biblioteca destinada à linguagem utilizada. No FORTRAN, esta se encontra no arquivo `mpif.h`, cuja referência pode ser feita no próprio código ou no próprio comando de compilação (o arquivo é o mesmo tanto para o LAM MPI quanto para o MPICH). Esta biblioteca é composta de cerca de cento e cinquenta sub-rotinas, porém neste trabalho utilizaram-se apenas nove, as quais são descritas no apêndice B.

O compilador utilizado neste trabalho foi o `g77` que acompanha a distribuição do *Linux Red Hat* instalada nas máquinas da rede. Ao se ter instalada na máquina uma biblioteca de padrão MPI, seja o LAM MPI ou o MPICH, a compilação dos códigos paralelos pode ser feita através do comando `mpif77`, este já reconhecendo que o programa a ser compilado é paralelo, não necessitando a inserção de opções especiais de compilação na linha de comando. A chamada básica à compilação na linha de comando é a seguinte:

```
mpif77 <código_paralelo> -o <executável>
```

Com a criação do arquivo executável e inicialização do ambiente paralelo nas máquinas desejadas, o início da execução se dá pela utilização do comando `mpirun`, da forma apresentada abaixo:

```
mpirun -np <número_de_máquinas_à_utilizar> <executável>
```

As formas de inicialização do ambiente paralelo variam de acordo com a biblioteca utilizada e, no caso do MPICH, pode não ser visível ao usuário. Para maiores informações, é sempre possível consultar as publicações disponíveis nos *sites* das respectivas bibliotecas.

Capítulo 4

Metodologia

4.1. Introdução:

A seguir, serão descritas as estratégias de implementação em paralelo do código CAV3D em *clusters* homogêneos. Ao final, serão propostas estratégias de implementação em *clusters* heterogêneos.

A partir da execução do código CAV3D em sua versão seqüencial original, observou-se que cerca de 96% do tempo utilizado por ele concentrava-se em apenas três etapas (vide figura 4.1):

- montagem do sistema de equações lineares,
- resolução do sistema de equações,
- determinação dos deslocamentos e tensões no domínio do problema.

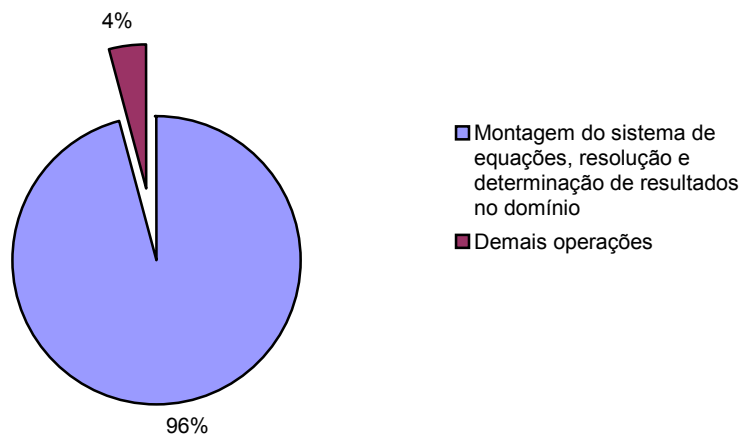


Figura 4.1 - Porcentagens dos tempos relativos às diferentes etapas no código seqüencial

4.2. Montagem do Sistema de Equações Lineares:

Como foi descrito anteriormente, a aplicação do Método dos Elementos de Contorno em problemas de elasticidade linear sem forças de volume nos leva a resolver um sistema de equações lineares da forma

$$Hu = Gp \quad (4.1)$$

que após serem consideradas as condições de contorno, leva o sistema acima à forma

$$Ax = b \quad (4.2)$$

o qual pode ser agora resolvido através de métodos diretos ou iterativos (caso do GMRES, neste trabalho).

A montagem do sistema é feita percorrendo-se os nós funcionais-fonte da malha, gerando-se três equações completas do sistema a partir de cada um, isto é, três linhas da matriz **A** e três termos do vetor **b**. Com base nesta característica, pode-se observar que, uma das possibilidades para a implementação desta etapa em paralelo é apenas dividir os nós funcionais entre os processadores utilizados, fazendo com que cada um opere sobre um sub-conjunto do total de nós, gerando um certo número de linhas do sistema de equações.

Foram estudadas e testadas duas formas de determinar os conjuntos de nós funcionais que seriam utilizados pelos processadores, ambas com base na numeração dos nós. Na primeira forma, chamada seqüencial, os conjuntos são formados por nós de numeração adjacente, de acordo com a figura 4.2.

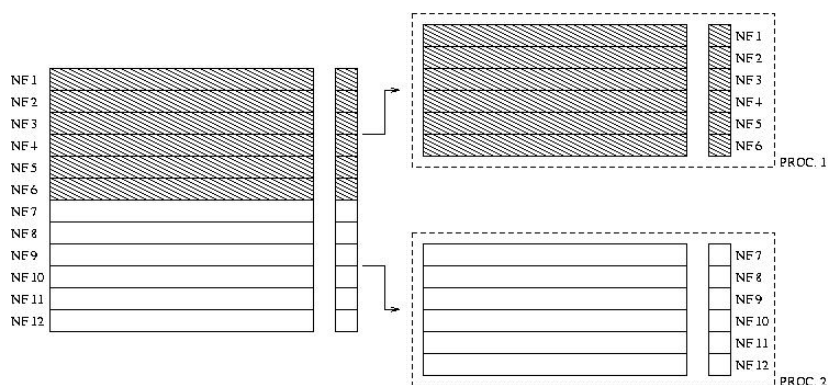


Figura 4.2 – Distribuição sequencial de nós funcionais

Neste caso, cada sub-conjunto possui um número de nós-fonte igual ao quociente inteiro da divisão entre o total de nós funcionais e a quantidade de processadores utilizados na análise. Caso a divisão mencionada não seja exata, incorpora-se um nó a cada processador até a distribuição da totalidade dos nós restantes. Desta forma, cada sub-conjunto possui um total de $\text{int}(NTNF/NP)+r$ nós funcionais com numeração adjacente, tendo início no nó de número $MYID * (\text{int}(NTNF/NP)+r) + 1$ e final no nó de número $(MYID+1) * (\text{int}(NTNF/NP)+r)$, onde $NTNF$ é o total de nós funcionais, NP é a quantidade de processadores utilizados e $MYID$ é o número que identifica o processador (variando de 0 à $NP-1$) e r assume valor 1 do processador 1 ao $\text{mod}(NTNF/NP)$ e 0 para os demais (lembrar que os operadores $\text{int}(_)$ e $\text{mod}(_)$ representam, respectivamente, o quociente inteiro e o resto inteiro da divisão entre parênteses).

Na segunda forma de distribuição, chamada cíclica, os conjuntos são formados por nós com numeração saltada, de acordo com a figura 4.3.

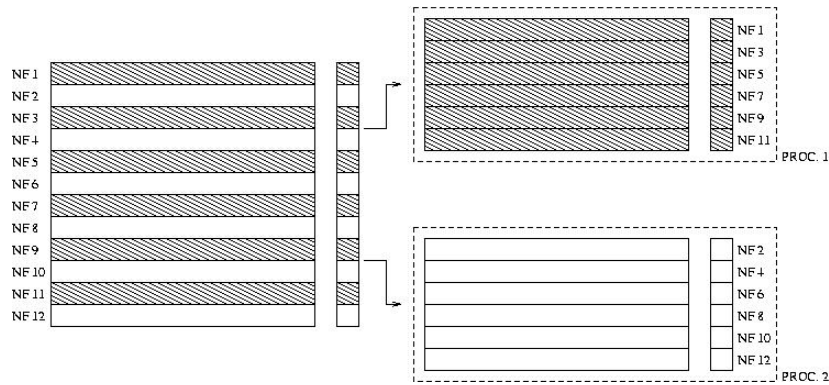


Figura 4.3 – Distribuição cíclica dos nós funcionais

Nesta forma de distribuição, cada sub-conjunto é composto por nós-fonte de numeração não-adjacente saltada de NP posições, isto é, o sub-conjunto do processador 0 é composto pelos nós 1, $1+NP$, $1+2.NP$, ... , o sub-conjunto do processador 1 é composto pelos nós 2, $2+NP$, $2+2.NP$, ... , e assim sucessivamente. Em síntese, cada conjunto possui $int(NTNF/NP)+r$ nós funcionais com numeração saltada em NP , tendo início no nó de número $MYID+1$ e final no nó de número $((int(NTNF/NP)+r)-1)*NP+MYID+1$.

O motivo de se ter estudado duas estratégias de distribuição de nós-fonte é, em princípio, bem simples. Viu-se na seção 2.4 como o CAV3D numera os nós funcionais de acordo com a numeração dos elementos e quais são as características do esquema de cálculo das integrais não-singulares. Em função da geometria e do refinamento do problema, a geração das linhas relativas a alguns nós funcionais demanda mais esforço computacional que as de outros, devido à variação da quantidade de pontos de Gauss utilizados na integração em função da relação entre a área dos elementos sobre os quais se integra e o quadrado de distância entre eles e o nó fonte. Na distribuição seqüencial, há a possibilidade de grande parte destes nós pertencerem a um conjunto pequeno de processadores, enquanto os demais possuam nós de regiões de baixa densidade de elementos. Na distribuição cíclica, tenta-se fazer com que todos os processadores

possuam em seu conjunto nós funcionais de todas as regiões da malha, distribuindo-se melhor o esforço computacional entre eles.

4.3. Resolução do Sistema de Equações:

Como característica, o GMRES executa uma multiplicação entre a matriz global do sistema e um vetor uma vez em cada iteração, operação esta que consome cerca de 95% do esforço computacional necessário para a resolução do sistema. Dentro do conjunto de operações cuja implementação em paralelo é eficiente, a multiplicação matriz – vetor é uma das mais conhecidas e utilizadas. As demais operações executadas no GMRES não foram implementadas em paralelo devido à sua pequena contribuição ao esforço total utilizado, sendo que, na presente implementação, elas só são executadas por apenas um processador (no caso, o processador mestre). Logo, nesta etapa, bastou implementar em paralelo a multiplicação matriz – vetor para se conseguir resultados satisfatórios. Esta multiplicação é executada por todos os processadores entre os trechos da matriz \mathbf{A} e o vetor $\mathbf{v}^{(i)}$ (vide esquema na seção 2.5), sendo este o vetor produto desta mesma operação na iteração anterior. O resultado desta multiplicação é concatenado no processador mestre formando o vetor $\mathbf{w}^{(i)}$ completo. Após a concatenação, verifica-se se com a base de Krylov acrescida deste novo vetor é possível obter uma solução que satisfaça a tolerância especificada. Caso não o seja, a partir de $\mathbf{w}^{(i)}$ calcula-se $\mathbf{v}^{(i+1)}$, que é distribuído aos demais processadores para ser utilizado na próxima iteração. Este processo está esquematizado na figura 4.4.

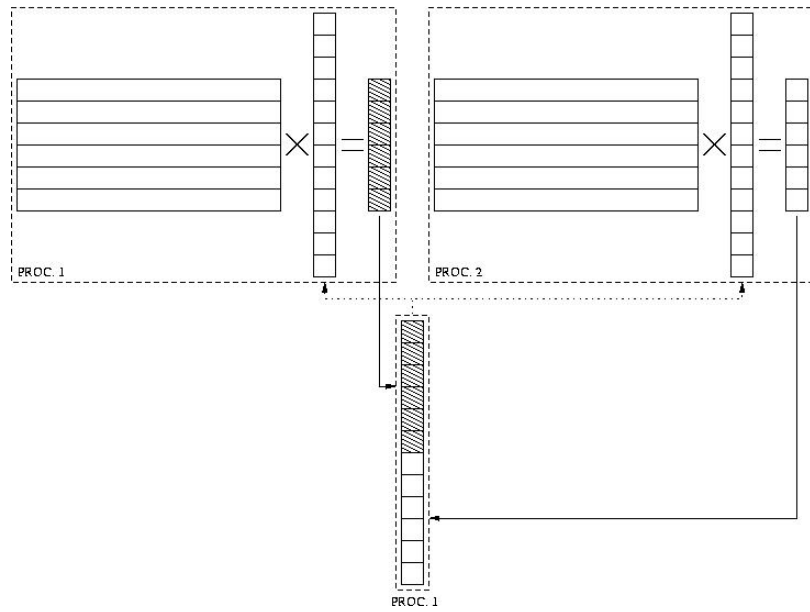


Figura 4.4 – Multiplicação matriz-vetor na etapa de resolução

Quando, na etapa de montagem, a distribuição dos nós funcionais utilizada for cíclica, após a concatenação, é necessário que se reordene os termos do vetor $v^{(i)}$ de acordo com a numeração original das equações, de forma que o sistema de equações não seja alterado durante sua resolução (vide figura 4.5).

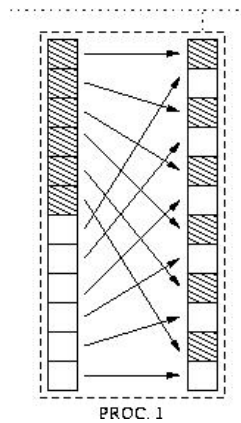


Figura 4.5 – Reordenação do vetor em cada multiplicação na distribuição cíclica

4.4. Determinação dos Deslocamentos e Tensões no Domínio:

Após a determinação dos deslocamentos e forças de superfície ao longo de todos os pontos do contorno, estes são distribuídos a todos os processadores. Com isso, a determinação dos deslocamentos e tensões em pontos no domínio pode ser feita de forma independente por cada processador. Isso faz com que esta etapa seja naturalmente paralelizável, bastando distribuir os pontos internos entre os processadores envolvidos.

Ao final dos cálculos em todos os processadores, os resultados são concatenados no processador mestre para que então sejam escritos no arquivo de saída de dados.

4.5. Estratégia para Utilização em Redes Heterogêneas:

Até agora, na metodologia então proposta, considerou-se apenas a implementação em redes homogêneas, onde todos os processadores envolvidos possuem as mesmas características.

Quando na utilização de redes heterogêneas, a forma que se dá a implementação é definida por fatores inerentes à rede e ao próprio problema a analisar. As características dos processadores da rede que são levadas em consideração são as diferenças de desempenho e memória disponível entre processadores que o compõe.

O balanceamento da carga computacional consiste em fazer com que cada processador envolvido na análise opere, na etapa de resolução do sistema de equações, com uma quantidade de equações proporcional à sua capacidade. Para que isto seja feito com uma quantidade reduzida de comunicações entre os processadores, é necessário que, na etapa de resolução, um certo número de linhas da matriz esteja armazenado em

mais de um processador, compondo um conjunto extra de linhas. O balanceamento consistirá em simplesmente definir qual processador irá operar com cada uma das linhas deste conjunto. Cada processador poderá possuir até dois destes conjuntos, os quais a partir de agora serão referidos como faixas adicionais, de acordo com a figura 4.6.

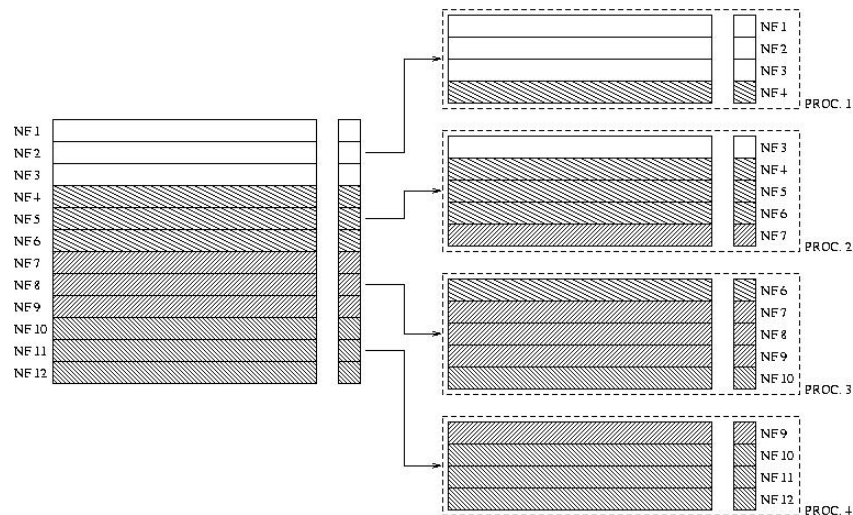


Figura 4.6 – Esquema de posicionamento das linhas adicionais em cada processador

Objetivando que cada processador possua as faixas adicionais que lhe caberia, de acordo com a figura acima, duas estratégias foram estudadas e implementadas. A primeira estratégia consiste simplesmente na montagem, em cada processador, das faixas que lhe são designadas, não havendo comunicação. Neste caso, as linhas que compõem um faixa adicional serão computadas duas vezes em processadores diferentes. A segunda estratégia utiliza comunicação entre os processadores, evitando-se computação redundante. Nesta estratégia, as linhas que compõem as faixas adicionais são computadas apenas uma vez e, ao final da montagem de todas as linhas que compõem os conjuntos de cada processador, cada processador adquire, através de troca de mensagem, as linhas de suas faixas adicionais junto ao processador que as computou. Em ambas as estratégias, o número de equações em cada faixa adicional utilizado neste trabalho é de um quarto do total de equações que cada processador monta, isto é:

$$NEQ_{faixa} = \text{int}\left(\frac{NTEQ}{4NP}\right) \quad (4.3)$$

onde $NTEQ$ é o total de equações do sistema e NP é o número de processadores envolvidos. Este valor foi arbitrado, *a priori*, apenas com objetivo de se determinar as relações entre os tempos de computação e comunicação das linhas da faixa adicional para cada uma das malhas analisadas. Este valor pode ser facilmente alterado de forma a fazer com que o código seja mais eficiente. A fixação deste tamanho é feita tomando-se como base a diferença de capacidade entre as máquinas utilizadas, sendo que quanto maior esta for, maior deverá ser o tamanho.

Na primeira iteração, todos os processadores irão trabalhar com uma mesma quantidade de linhas. A partir do tempo em que cada um necessita para fazer a primeira multiplicação matriz-vetor, será executado o balanceamento da carga, definindo-se a nova quantidade de linhas com que cada processador operará nas iterações seguintes. Partindo-se da aproximação de que o tempo de montagem varia linearmente com o número de linhas em cada processador, estima-se este número para que o tempo gasto na multiplicação em cada processador seja igual ao dos demais. Este é então obtido resolvendo-se o sistema

$$\begin{aligned} N_0T_0 &= N_1T_1 = \dots = N_{NP-1}T_{NP-1} \\ N_0 + N_1 + \dots + N_{NP-1} &= NTEQ \end{aligned} \quad (4.4)$$

onde N_i é o número de linhas com que o processador de *rank* i operará nas próximas iterações e T_i é tempo que o processador i gasta para multiplicar uma linha da matriz por um vetor. A primeira equação define que o tempo em que um processador multiplica seu trecho da matriz pelo vetor é igual ao tempo dos demais. A segunda equação fixa que a soma do número de linhas da matriz que irão ser operadas por cada processador é igual ao número total de linhas da matriz completa do sistema. Este processo está esquematizado na figura 4.7.

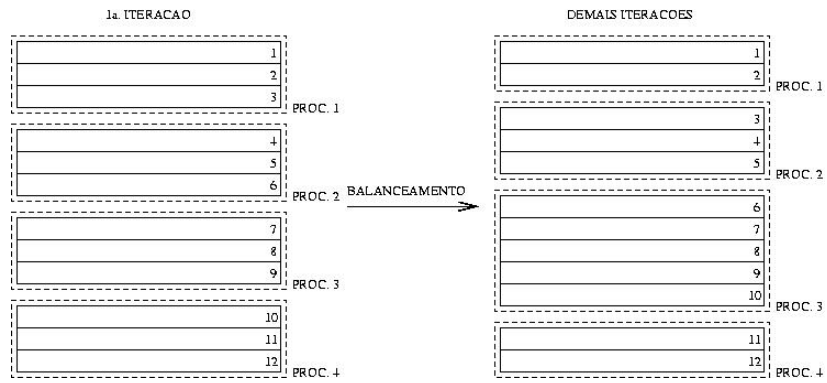


Figura 4.7 – Balanceamento de carga computacional na etapa de resolução

Após a resolução do sistema (4.4), tem-se que a quantidade de linhas que o processador i operará depois da execução do balanceamento é a seguinte:

$$N_i = \frac{NTEQ}{\sum_{j=0}^{NP-1} T_j} \quad (4.5)$$

Capítulo 5

Resultados Numéricos

5.1. Introdução:

Neste capítulo são descritos os resultados alcançados com as várias estratégias utilizadas nas implementações em paralelo do código CAV3D, de forma a se avaliar sua eficiência.

Foram desenvolvidas cinco opções distintas utilizando-se estratégias diferentes para a etapa de montagem e balanceamento de carga na etapa de resolução. Estas diferenças estão explicitadas na tabela 5.1:

Tabela 5.1 – Códigos utilizados na análise e suas respectivas diferenças

Código	Divisão de nós na etapa de montagem	Passagem de mensagem	Balanceamento de carga na etapa de resolução
1	seqüencial	não	não
2	seqüencial	não	sim
3	seqüencial	sim	sim
4	cíclica	sim	não
5	cíclica	sim	sim

Os resultados expostos aqui foram obtidos na análise de dois problemas distintos. O primeiro é o problema de um prisma tracionado, sendo que sua análise foi feita utilizando-se três malhas com níveis de refinamento diferentes. O segundo problema é a cavidade da barragem de Serra da Mesa – GO (apresentada anteriormente na figura 2.1), cujos resultados foram primeiramente obtidos utilizando-se a primeira versão seqüencial do CAV3D [16].

As características das malhas dos exemplos utilizados são as apresentadas na tabela 5.2 e figura 5.1:

Tabela 5.2 - Características das malhas dos exemplos utilizados

Exemplo	Elementos	Nós geométricos	Nós funcionais	Pontos internos
Prisma nível 1	138	140	552	3
Prisma nível 2	198	200	792	3
Prisma nível 3	258	260	1032	3
Cavidade	406	1942	406	32

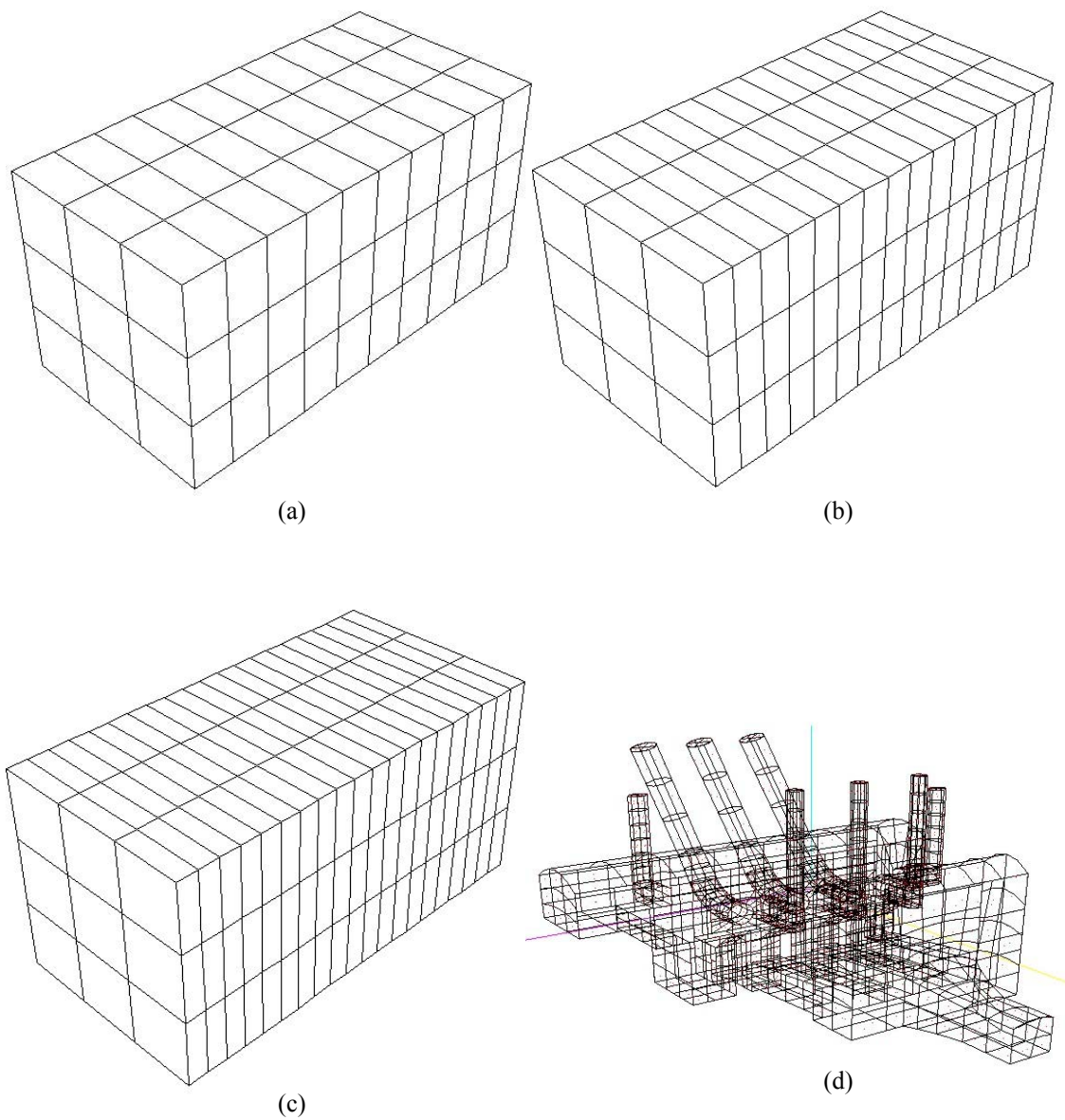


Figura 5.1 - Malhas dos problemas analisados: (a)prisma refinamento nível 1; (b)prisma refinamento nível 2; (c)prisma refinamento nível 3; (d)cavidade da barragem de Serra da Mesa.

As características referentes aos elementos utilizados foram as seguintes:

- Malhas do prisma: elementos quadrilaterais lineares descontínuos;
- Malha da cavidade de Serra da Mesa: elementos quadrilaterais constantes.

Em todos os cinco códigos, as características adotadas no GMRES são as seguintes:

- Dimensão m do subespaço de Krylov: $1/5$ da ordem do sistema de equações;
- Número máximo de ciclos: 10;
- Número máximo de iterações por ciclo: 50.

As redes utilizadas nas análises são as já descritas anteriormente no capítulo 3.

As análises no NACAD foram feitas utilizando-se, por opção, dois, quatro e oito processadores. No NUMEC, utilizou-se quatro processadores. Problemas inerentes a cada uma das redes fizeram com que não se utilizasse um número maior de processadores nas análises.

Para cada análise, foram tomados em cada processador os tempos relativos às etapas de montagem do sistema de equações, resolução, multiplicação matriz-vetor em cada iteração da etapa de resolução, bem como o tempo total da análise. Os tempos relativos à determinação de resultados no domínio foram tomados apenas na análise do problema da cavidade por questão de conveniência.

5.2. Rede Homogênea – *Cluster* do NACAD:

Apesar da rede instalada no NACAD ser homogênea, testou-se nela todos os códigos com balanceamento de carga computacional na etapa de resolução, de forma a verificar a eficiência das estratégias que utilizam ou não comunicação na etapa de

montagem. O balanceamento de carga foi somente testado. Sua eficiência será comprovada apenas na exposição dos resultados obtidos com a rede heterogênea.

5.2.1. Códigos Sem Balanceamento de Carga Computacional:

Os resultados que são expostos nas figuras 5.2 e 5.3 mostram os tempos de cada etapa na análise do exemplo do prisma em seus três níveis de refinamento, utilizando-se dois, quatro e oito processadores. São apresentados três destes gráficos para cada código.

Na figura 5.4 são mostrados gráficos relativos à porcentagem de computação referente à multiplicação matriz-vetor dentro da etapa de resolução do sistema de equações. Estes últimos têm por objetivo mostrar como varia o esforço computacional despendido nesta operação ao se variar o número de processadores utilizados.

5.2.2. Códigos Com Balanceamento de Carga Computacional:

Apresentam-se nas figuras 5.5 à 5.7 os gráficos expondo os resultados encontrados nas análises dos exemplos utilizando os códigos com balanceamento de carga computacional na etapa de resolução do sistema de equações.

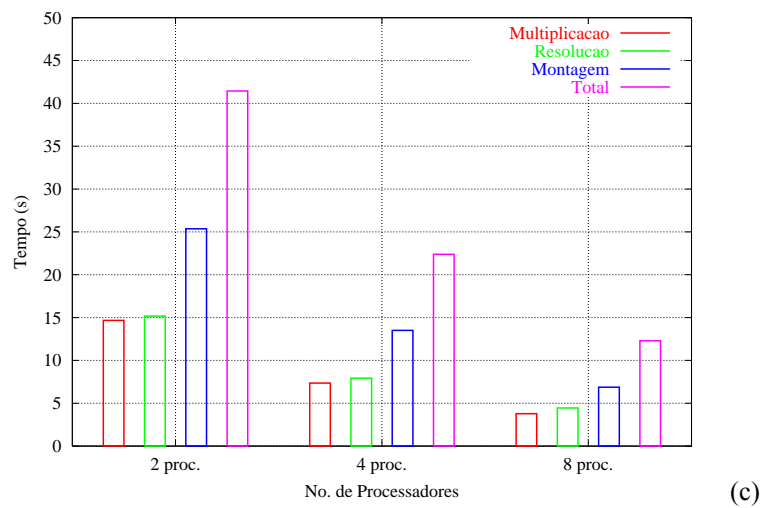
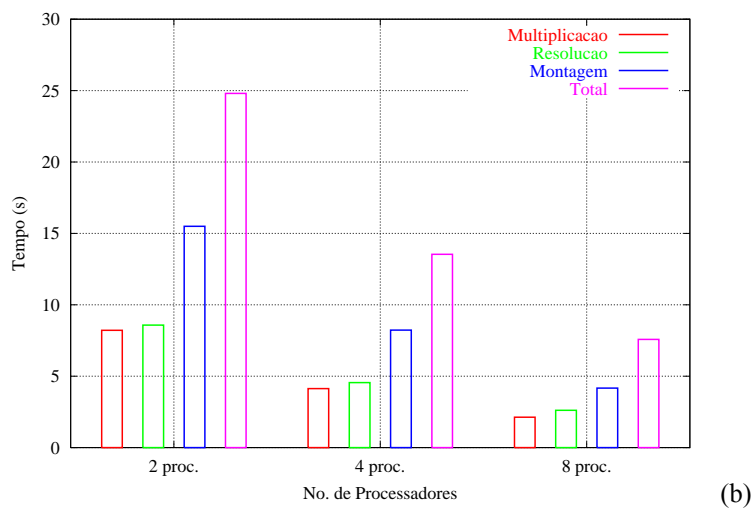
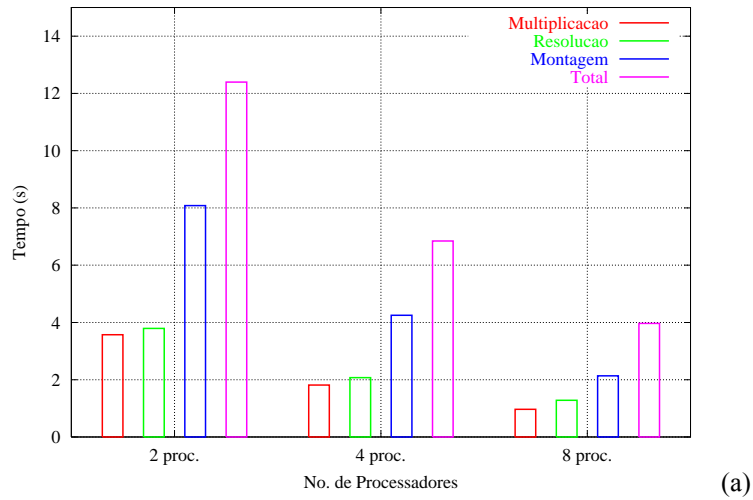


Figura 5.2 - Código sem balanceamento com montagem sequencial: (a) prisma ref. nível 1; (b) prisma ref. nível 2; (c) prisma ref. nível (3).

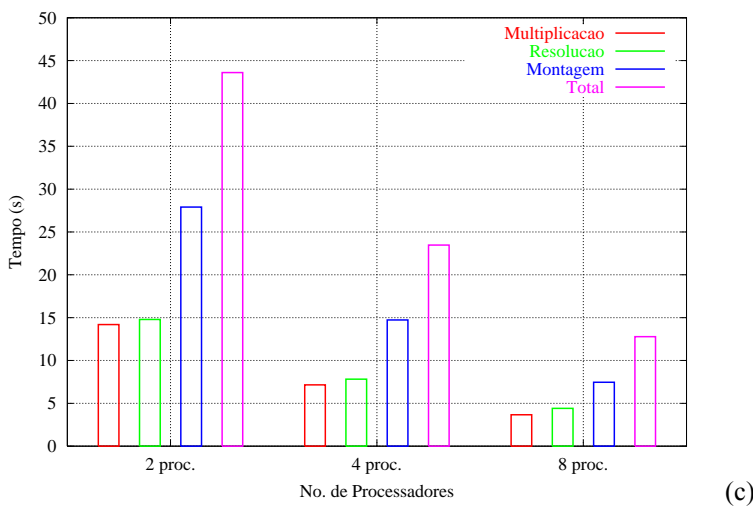
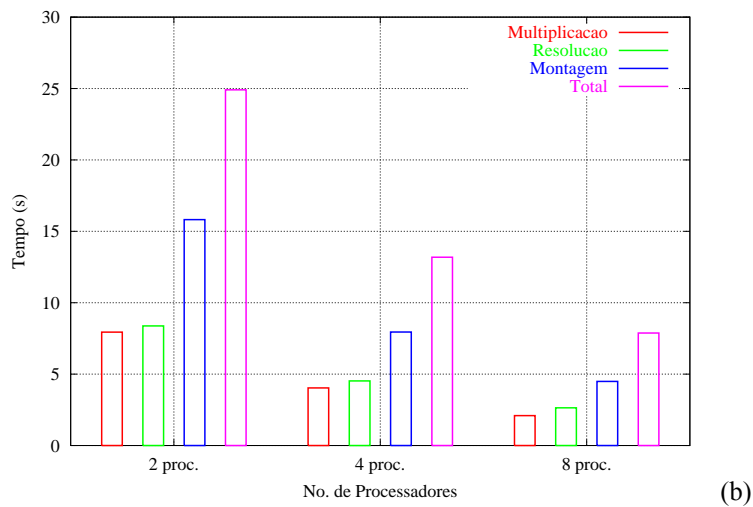
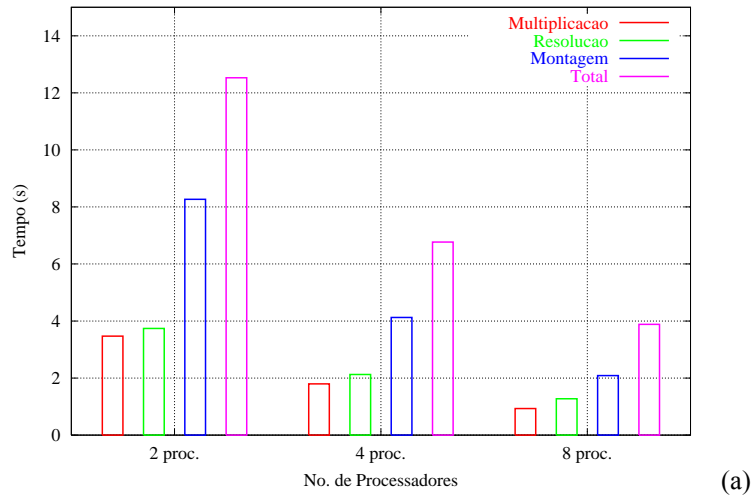
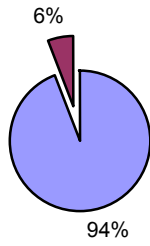
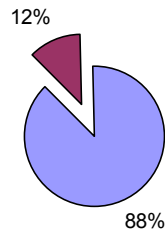


Figura 5.3 - Código sem balanceamento com montagem cíclica: (a)prisma ref. nível 1; (b)prisma ref. nível 2; (c)prisma ref. nível 3.

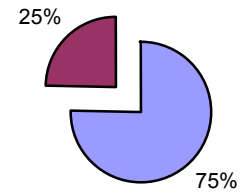
2 Processadores



4 Processadores



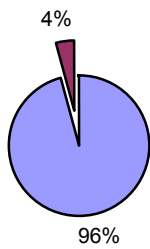
8 Processadores



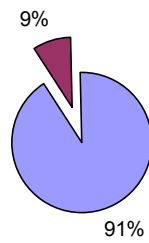
■ Multiplicação ■ Outras Operações

(a)

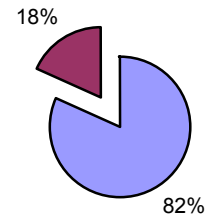
2 Processadores



4 Processadores



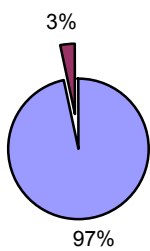
8 Processadores



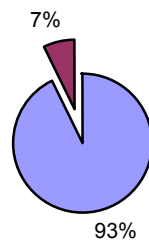
■ Multiplicação ■ Outras Operações

(b)

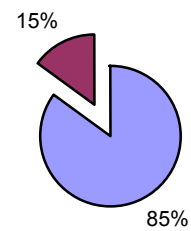
2 Processadores



4 Processadores



8 Processadores



■ Multiplicação ■ Outras Operações

(c)

Figura 5.4 – Porcentagens dos tempos despendidos na etapa de resolução referentes à multiplicação matriz-vetor: (a)prisma ref. nível 1; (b)prisma ref. nível 2; (c)prisma ref. nível 3.

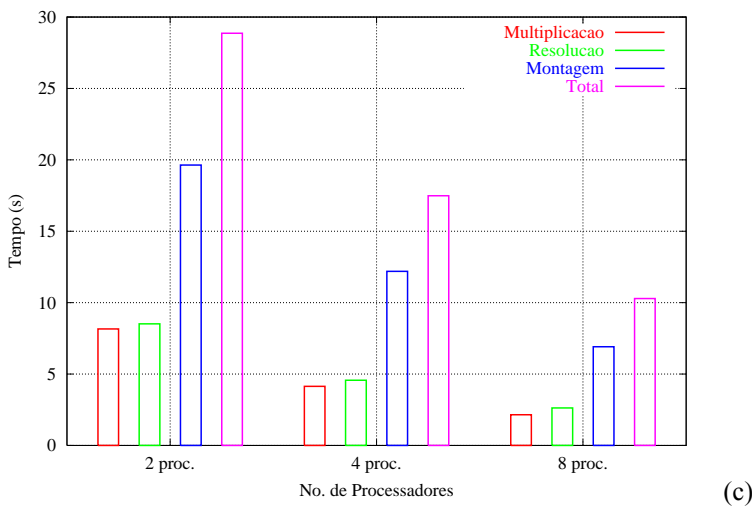
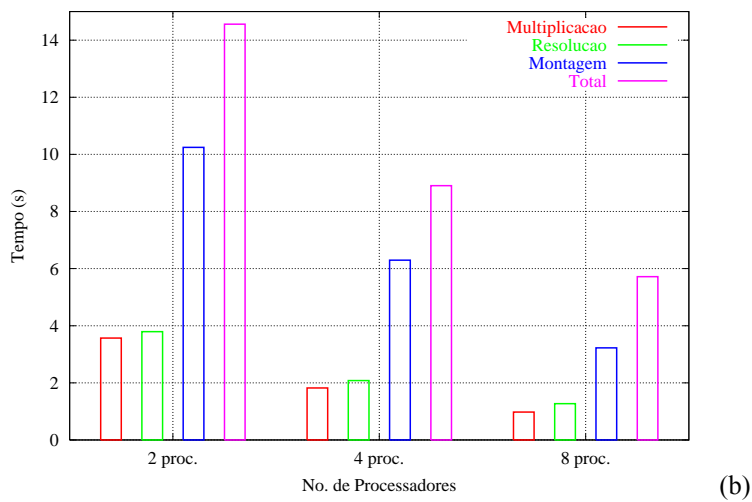
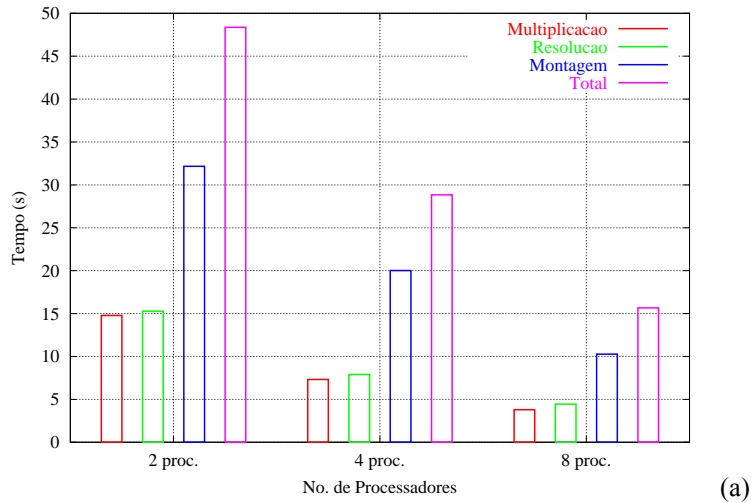


Figura 5.5 - Código com balanceamento e montagem sequencial, sem passagem de mensagem: (a)prisma ref. nível 1; (b)prisma ref. nível 2; (c)prisma ref. nível 3.

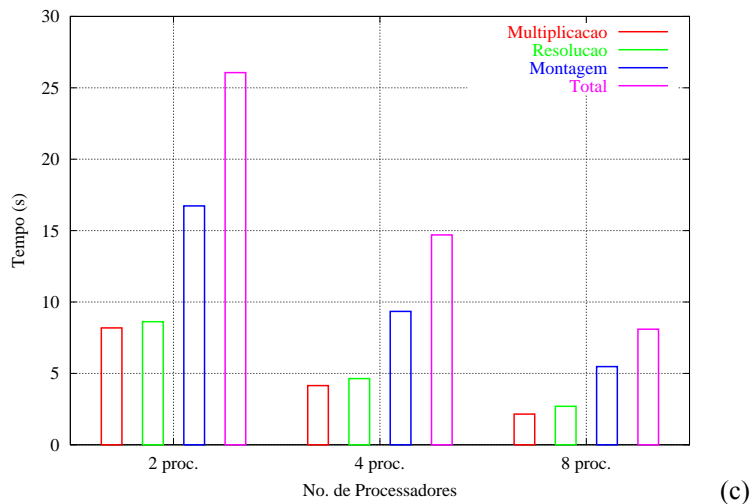
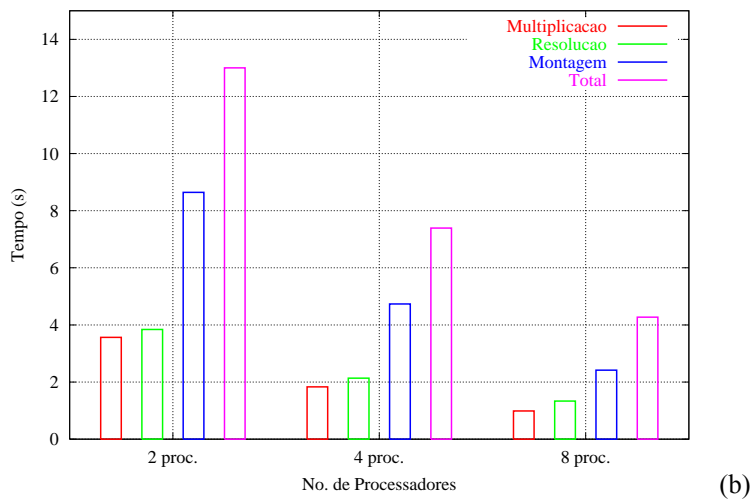
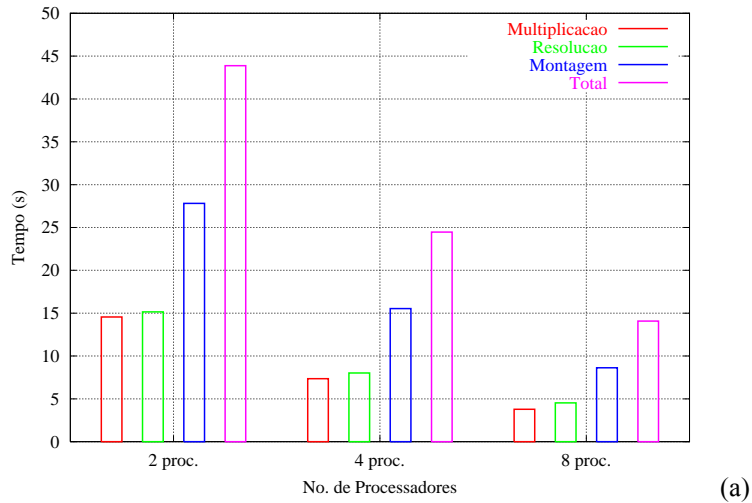


Figura 5.6 - Código com balanceamento, montagem seqüencial e passagem de mensagem: (a)prisma ref. nível 1; (b)prisma ref. nível 2; (c)prisma ref. nível 3.

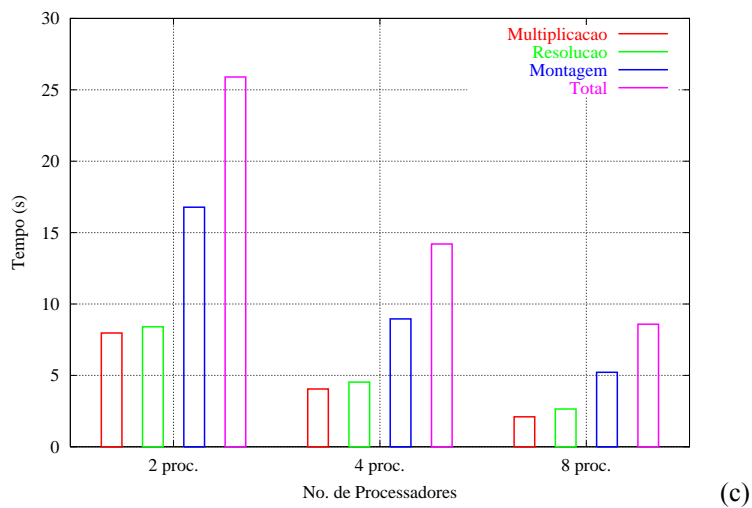
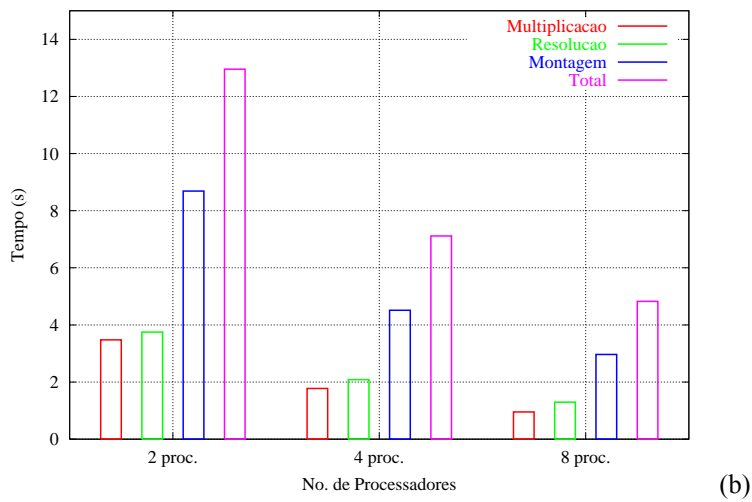
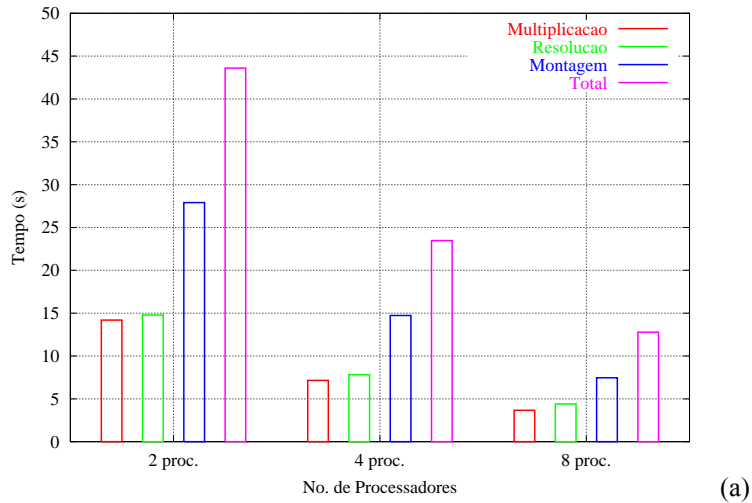


Figura 5.7 - Código com balanceamento, montagem cíclica e passagem de mensagem: (a)prisma ref. nível 1; (b)prisma ref. nível 2; (c)prisma ref. nível 3.

O gráfico apresentado na figura 5.8 ilustra a relação entre os tempos de computação e troca de mensagem relativa às faixas adicionais em cada processador na análise das malhas do exemplo do prisma.

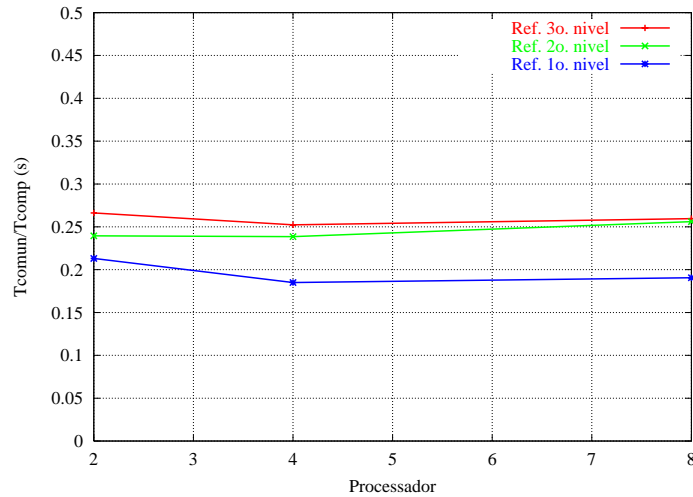


Figura 5.8 - Relação entre tempos de comunicação e computação das faixas adicionais

Como se pode observar para o tamanho da faixa adicional adotado, a estratégia de troca de mensagem se mostrou mais eficiente que a estratégia de computação dos elementos da faixa adicional. Em vista disto, a estratégia de montagem do sistema de equações utilizando distribuição cíclica dos nós funcionais só foi implementada com comunicação entre os processadores.

Nas figuras 5.9 à 5.12, estão apresentadas as comparações dos tempos obtidos por cada processador utilizando-se as duas estratégias de distribuição dos nós funcionais na etapa de montagem para o problema do prisma e da cavidade.

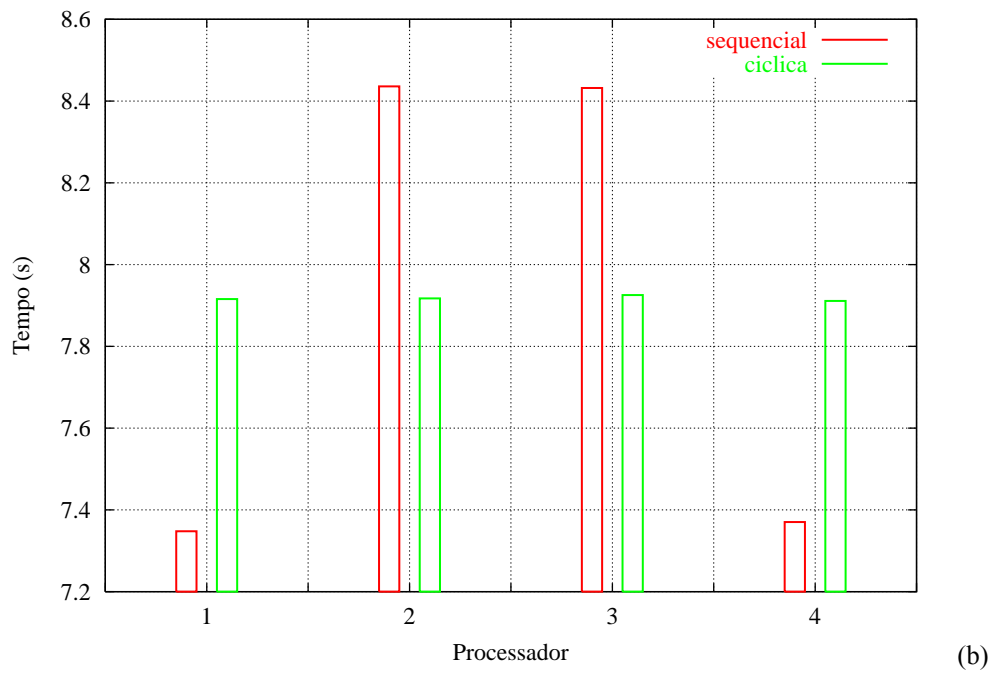
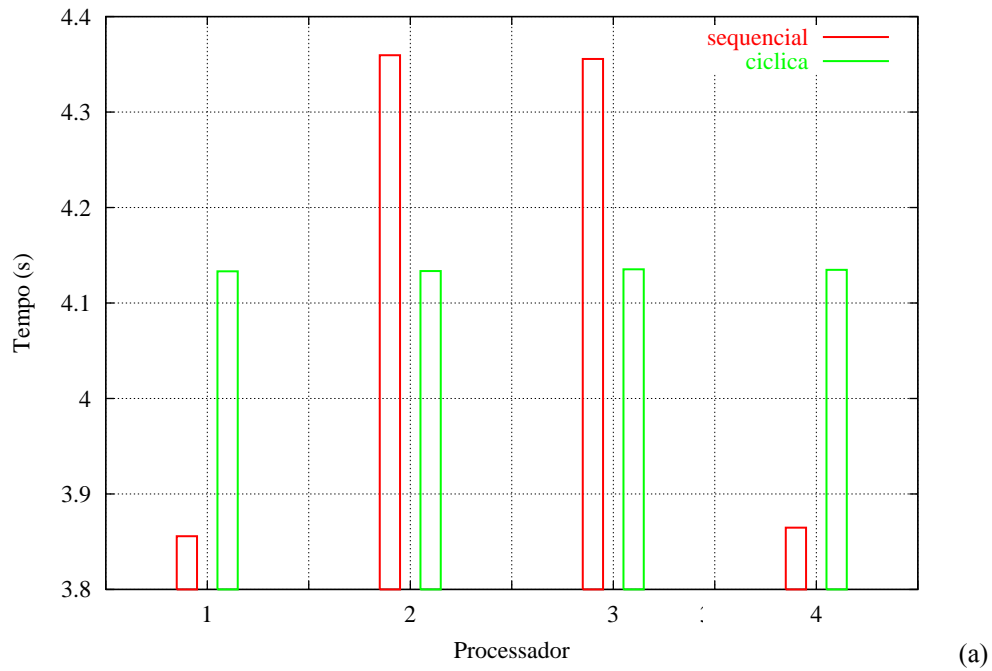
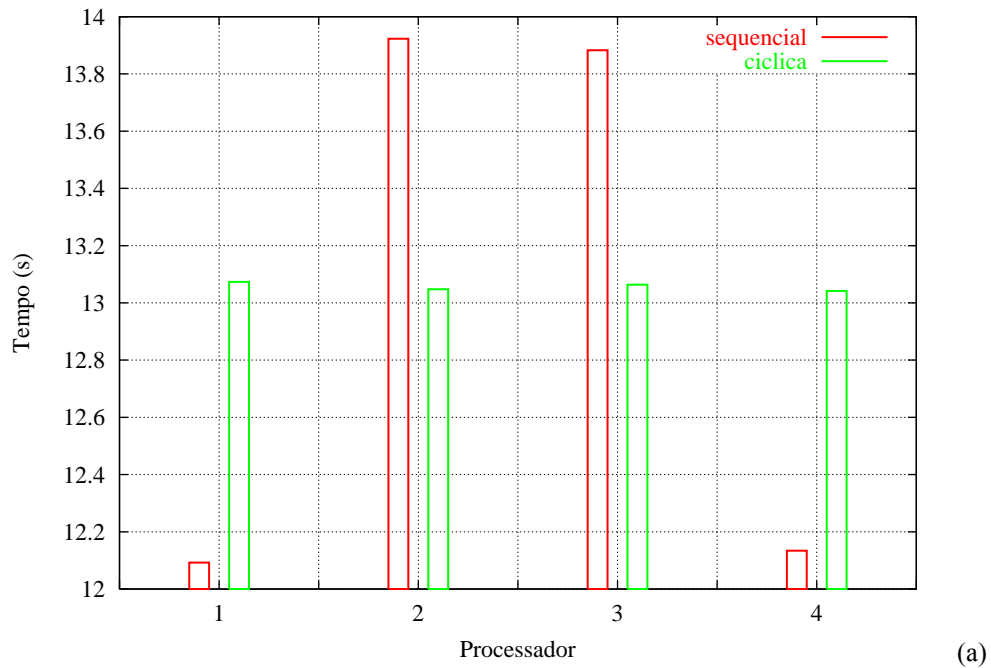
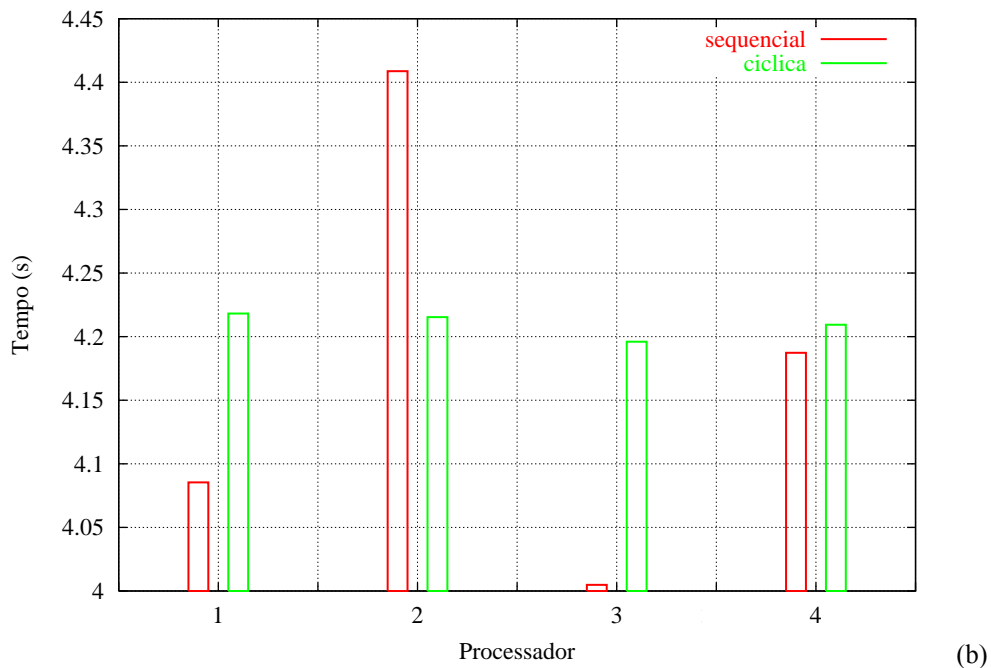


Figura 5.9 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 4 processadores: (a) prisma ref. nível 1; (b) prisma ref. nível 2.

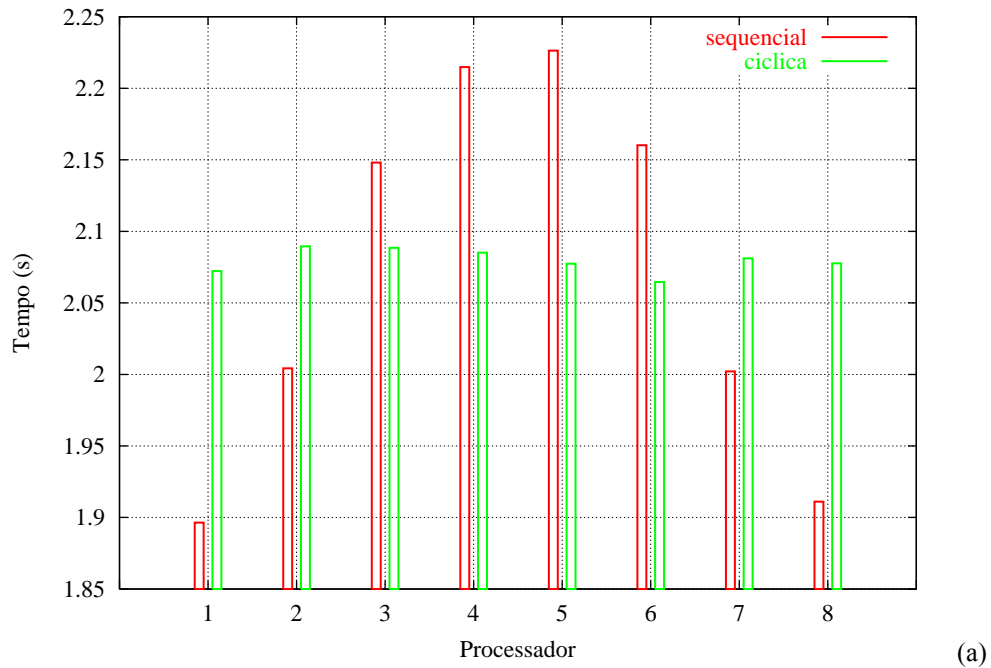


(a)

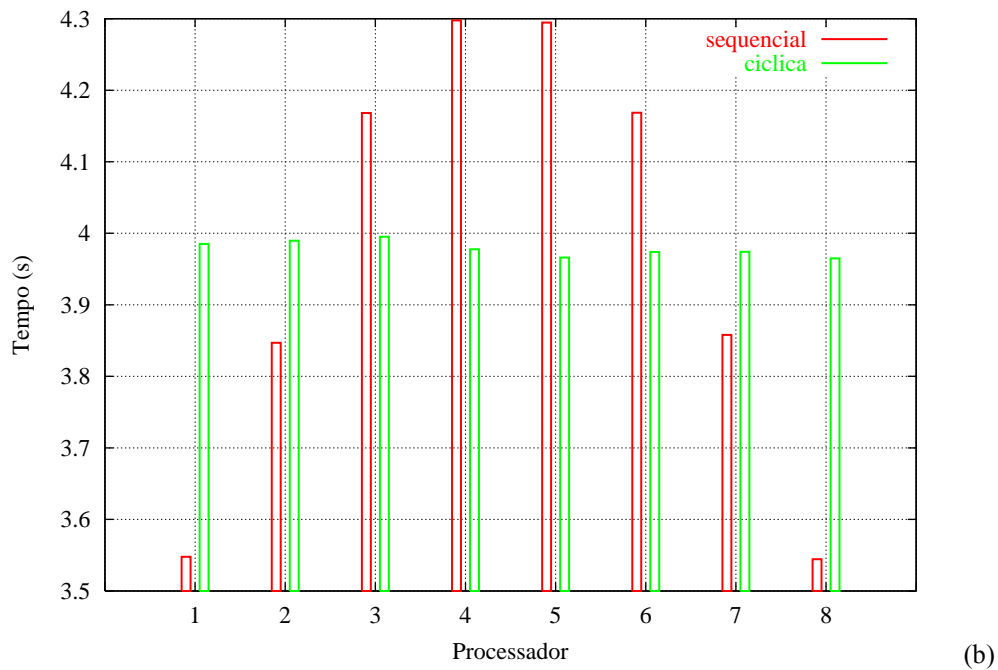


(b)

Figura 5.10 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 4 processadores: (a) prisma ref. nível 3; (b) cavidade.



(a)



(b)

Figura 5.11 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 8 processadores: (a) prisma ref. nível 1; (b) prisma ref. nível 2.

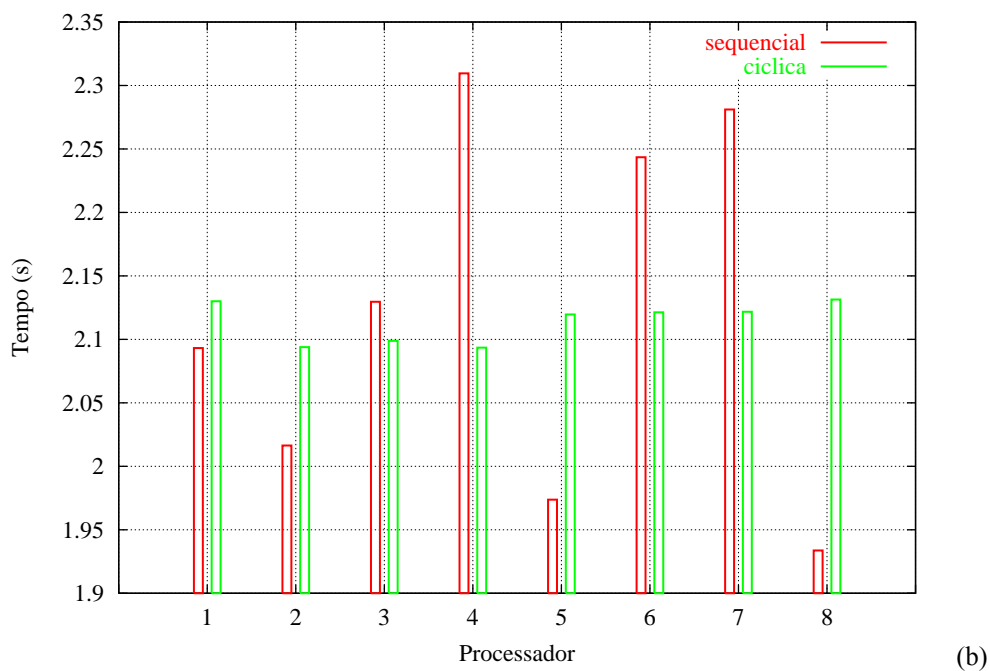
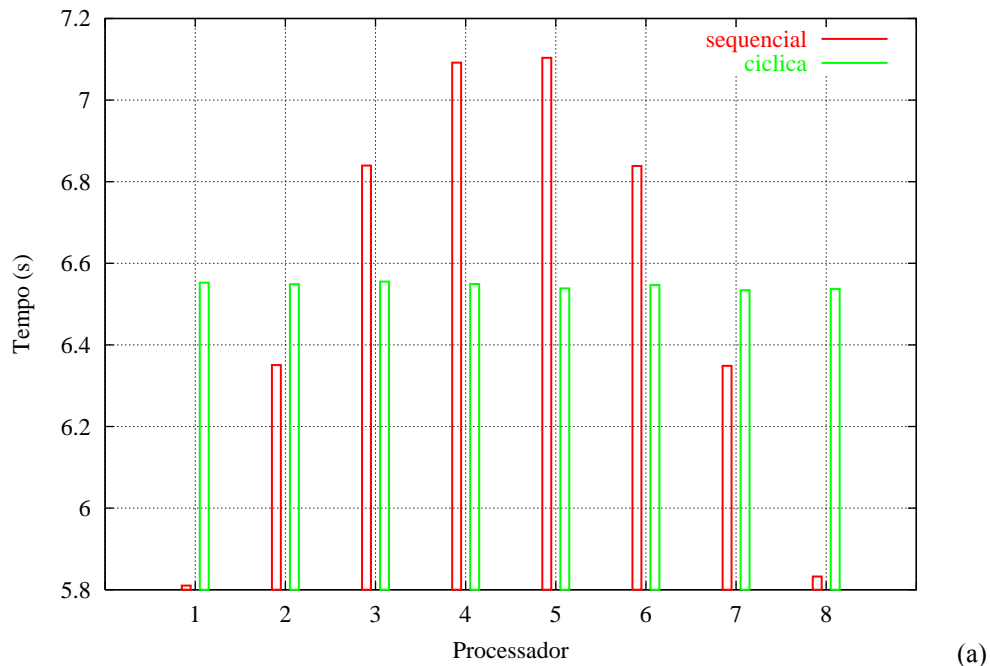


Figura 5.12 - Comparação entre tempos de montagem em cada processador para as duas estratégias de montagem utilizando 8 processadores: (c) prisma ref. nível 3; (d) cavidade.

Observa-se que, na distribuição cíclica, o esforço computacional na etapa de montagem fica mais bem distribuído entre os processadores que na distribuição sequencial. Os motivos pelos quais este fenômeno acontece já foram esclarecidos

anteriormente. Os resultados apresentados na figura 5.13 mostram a relação entre os tempos obtidos com distribuição cíclica e distribuição seqüencial.

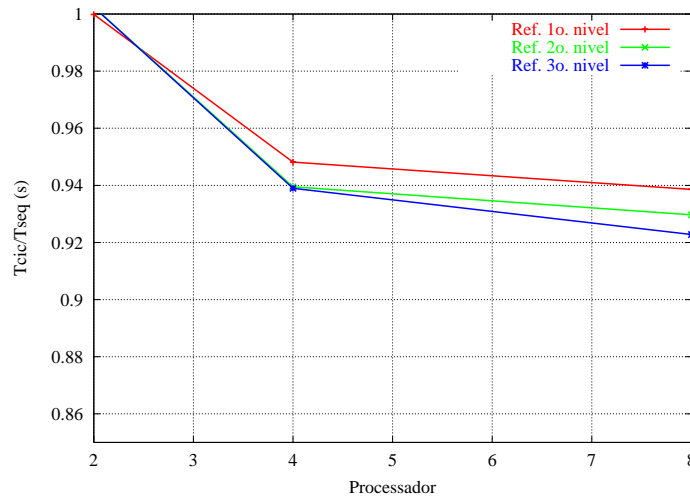


Figura 5.13 - Relação entre os tempos totais de montagem cíclica e seqüencial

Observa-se uma tendência de incremento na eficiência da distribuição cíclica de acordo com o aumento do nível de refinamento e do número de processadores. Os resultados obtidos com dois processadores são próximos de 1.0 devido à simetria das três diferentes malhas do exemplo do prisma.

Os resultados expostos na figura 5.14 são relativos aos tempos obtidos na determinação dos resultados no domínio do problema da cavidade da barragem de Serra da Mesa.

Com isso, mostrou-se o bom desempenho da implementação em paralelo do CAV3D em uma rede homogênea e verificou-se quais são as melhores estratégias de implementação das etapas envolvidas para a utilização nos dois tipos de rede. De forma a ilustrar a eficiência das implementações, estão apresentados na figura 5.15 os *Speed Up's* finais das implementações.

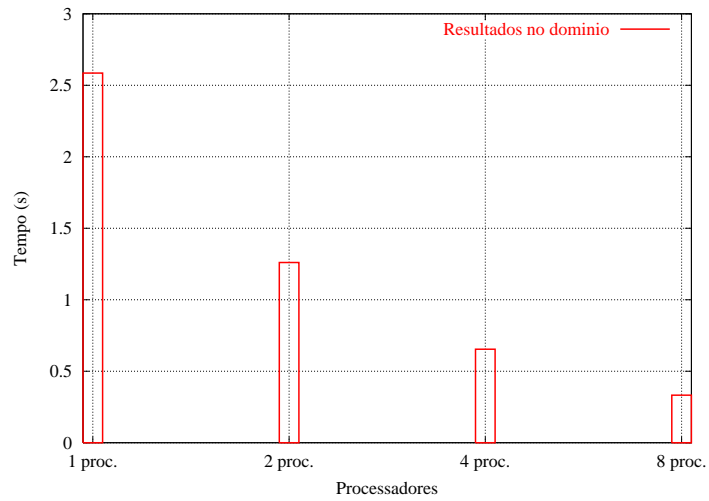


Figura 5.14 - Tempos de determinação dos resultados em pontos internos para o problema da cavidade

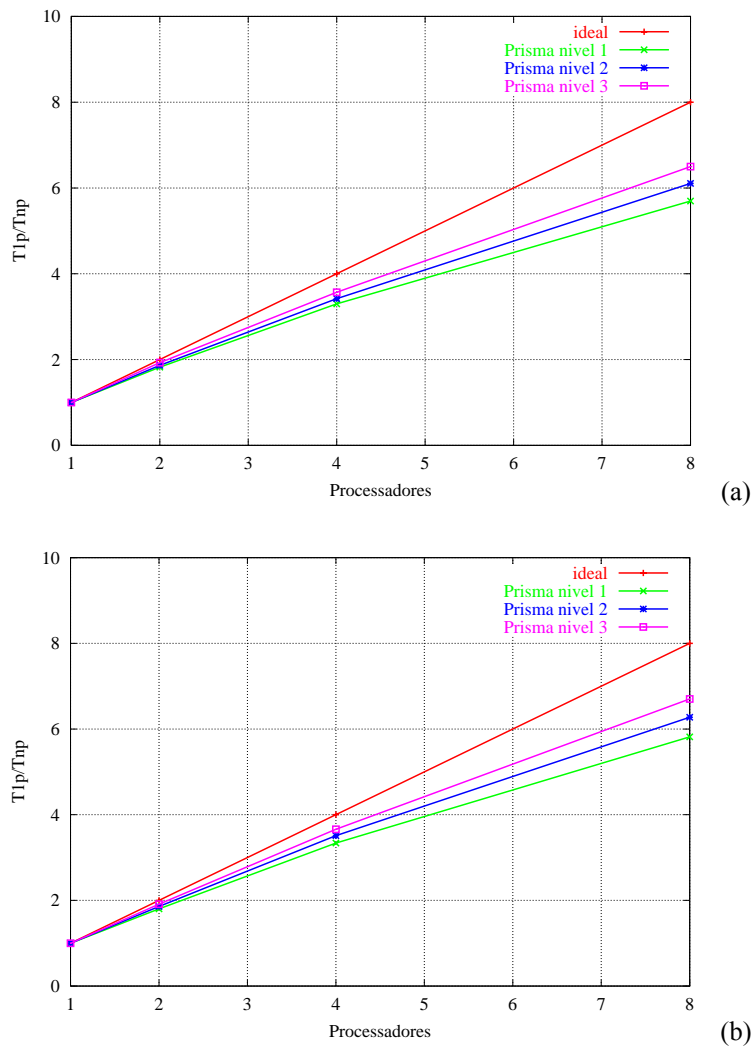


Figura 5.15 - *Speed Up*: (a) código com montagem seqüencial; (b) código com montagem cíclica.

5.3. Rede Heterogênea – NUMEC:

Os resultados apresentados nas figuras 5.16 à 5.19 mostram a eficiência do balanceamento da carga computacional na etapa de resolução do sistema de equações. Os gráficos apresentados nestes mostram o tempo despendido em cada iteração do GMRES ao se utilizar ou não o balanceamento de carga em todos os exemplos analisados. Também são apresentadas, nas tabelas 5.3 à 5.6, a quantidade de linhas em cada processador antes e após o balanceamento.

Conclui-se que o balanceamento de carga foi implementado eficientemente, o que fica claro ao se observar a aproximação dos tempos por iteração em cada processador após o fim do processo de equilíbrio. Os tempos mostrados não estão exatamente iguais porque, em alguns casos, esta diferença é relativa a apenas uma linha que, caso fosse montada por outro processador, faria com que o tempo total por iteração não fosse mais o mínimo e, em outros, o tamanho da faixa adicional se mostrou insuficiente. Fica evidente também que, quando a rede utilizada é dedicada, o balanceamento só necessita ser executado uma vez ao longo de toda a etapa. Quando a capacidade computacional dos processadores do cluster é alterada durante a execução, este balanceamento pode ser necessário mais de uma vez ao longo da resolução (balanceamento dinâmico). O número de vezes em que a repetição pode acontecer depende dos tipos de atividade que estão sendo desenvolvidas concomitantemente com a execução CAV3D. Sua definição, no entanto, pode ser feita pelo usuário.

Os gráficos apresentados nas figuras 5.20 e 5.21 mostram os tempos totais de resolução e da multiplicação matriz-vetor para cada exemplo com e sem balanceamento.

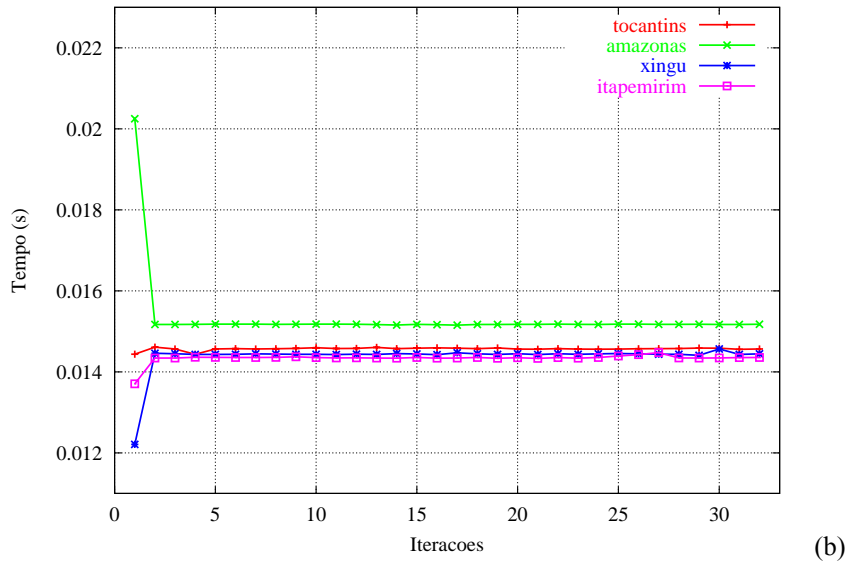
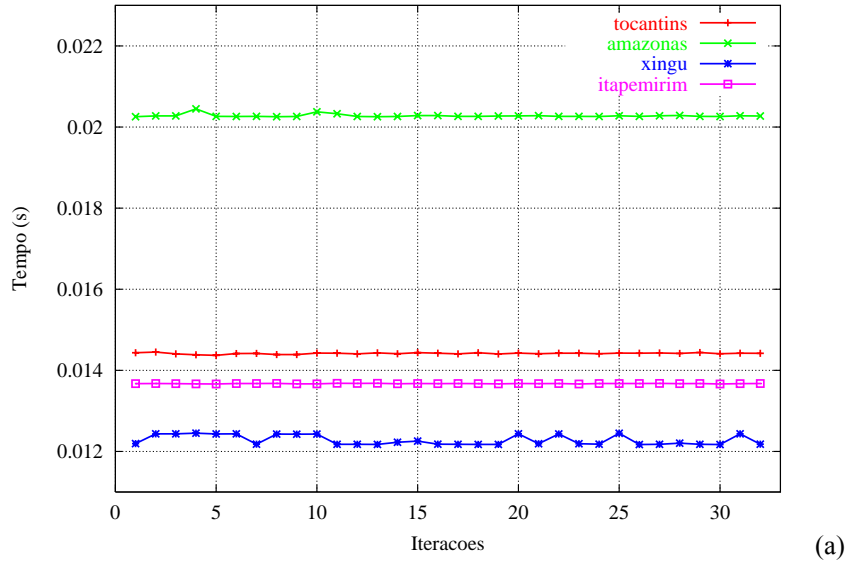


Figura 5.16 - Tempos por iteração para o problema do prisma refinamento nível 1: (a)sem balanceamento; (b)com balanceamento.

Tabela 5.3 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 1

Processador	1. ^a Iteração	Demais Iterações
<i>tocantins</i>	414	419
<i>amazonas</i>	414	307
<i>xingu</i>	414	495
<i>itapemirim</i>	414	435

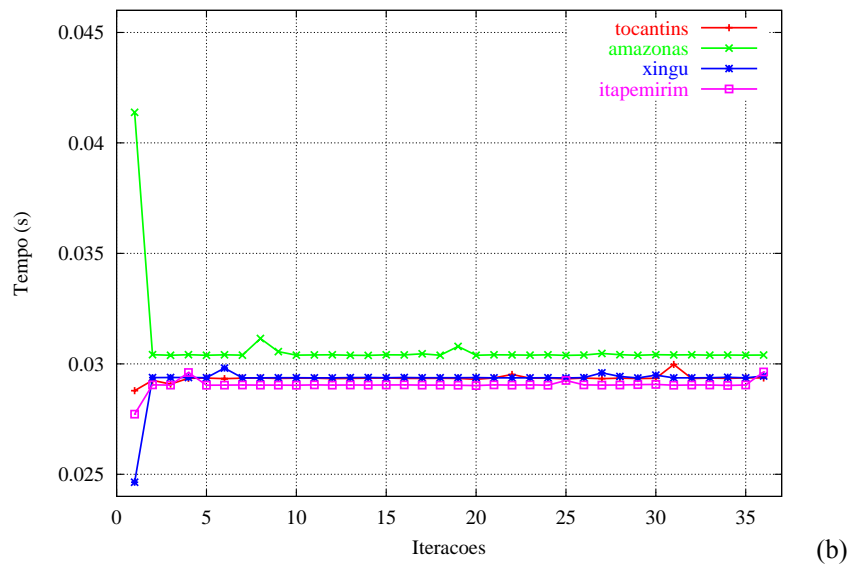
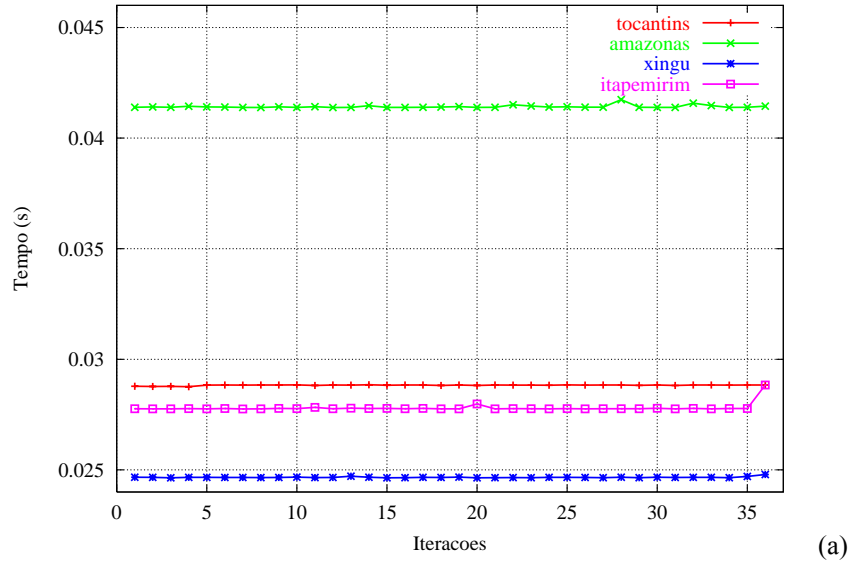


Figura 5.17 - Tempos por iteração para o problema do prisma refinamento nível 2: (a)sem balanceamento; (b)com balanceamento.

Tabela 5.4 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 2

Processador	1. ^a Iteração	Demais Iterações
<i>tocantins</i>	594	608
<i>amazonas</i>	594	433
<i>xingu</i>	594	711
<i>itapemirim</i>	594	624

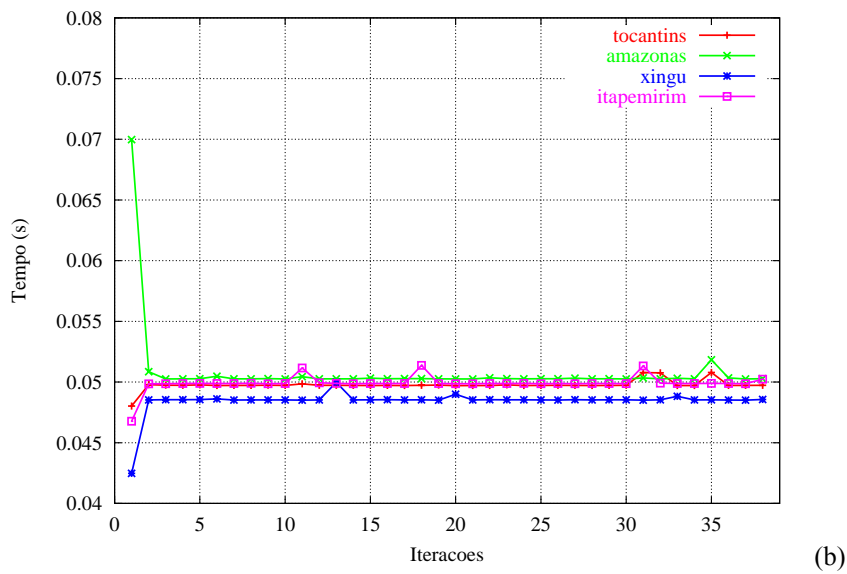
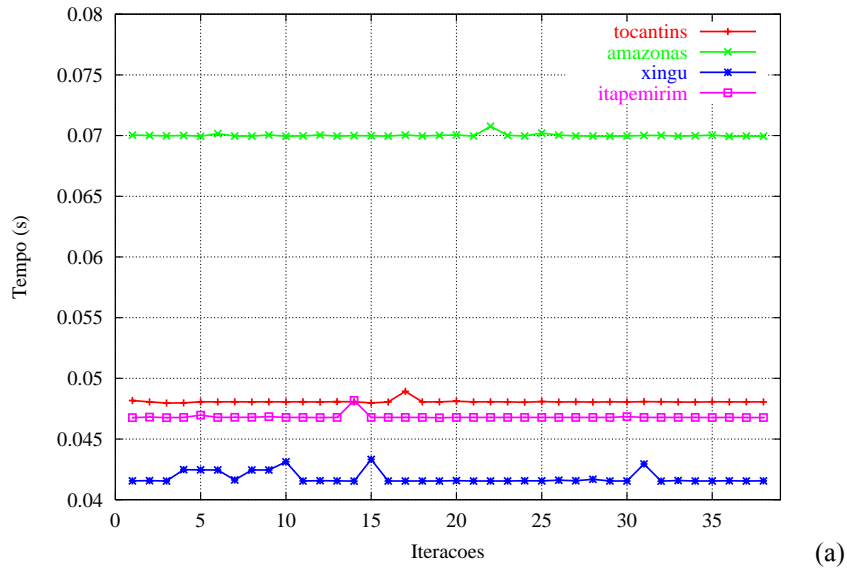


Figura 5.18 - Tempos por iteração para o problema do prisma refinamento nível 3: (a)sem balanceamento; (b)com balanceamento.

Tabela 5.5 - Número de linhas por processador antes e após o balanceamento: prisma ref. nível 3

Processador	1. ^a Iteração	Demais Iterações
<i>tocantins</i>	774	805
<i>amazonas</i>	774	552
<i>xingu</i>	774	910
<i>itapemirim</i>	774	829

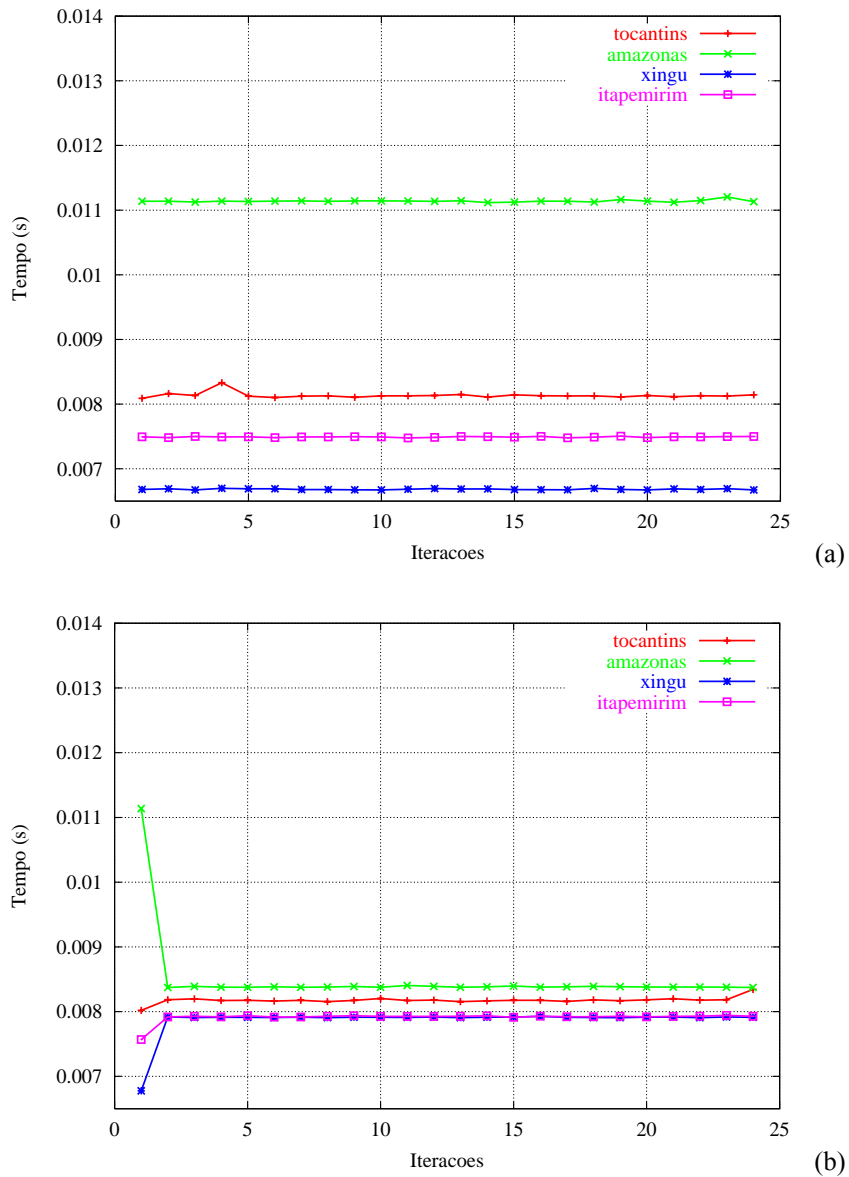


Figura 5.19 - Tempos por iteração para o problema da cavidade de Serra da Mesa: (a)sem balanceamento; (b)com balanceamento.

Tabela 5.6 Número de linhas por processador antes e após o balanceamento: cavidade de Serra da Mesa

Processador	1. ^a Iteração	Demais Iterações
<i>tocantins</i>	306	309
<i>amazonas</i>	306	228
<i>xingu</i>	303	362
<i>itapemirim</i>	303	319

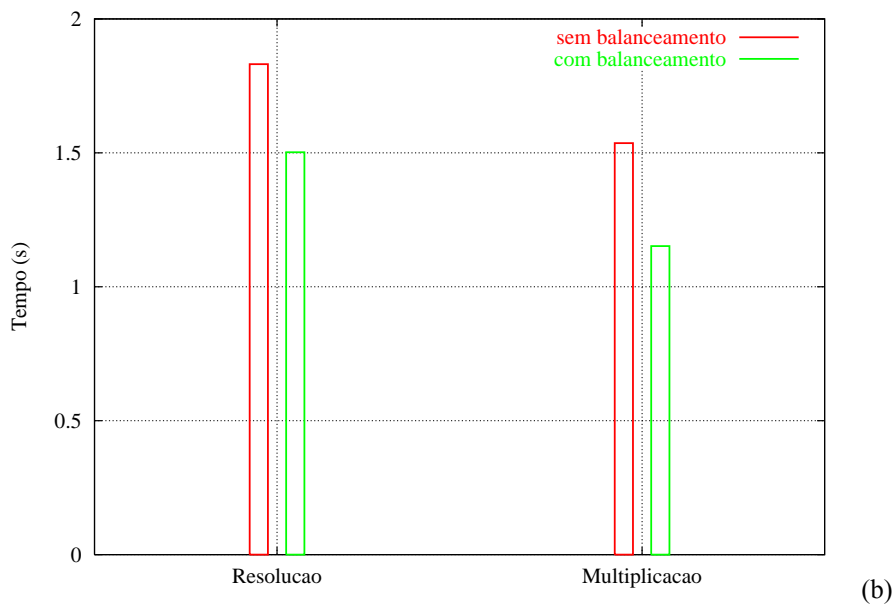
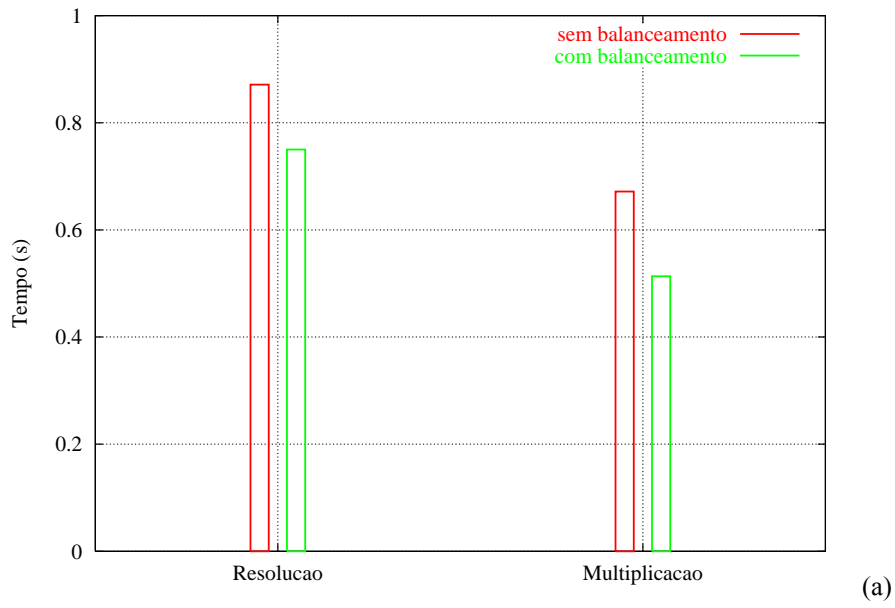


Figura 5.20 - Comparação entre os tempos totais de resolução e multiplicação matriz-vetor: (a) prisma ref. nível 1; (b) prisma ref. nível 2

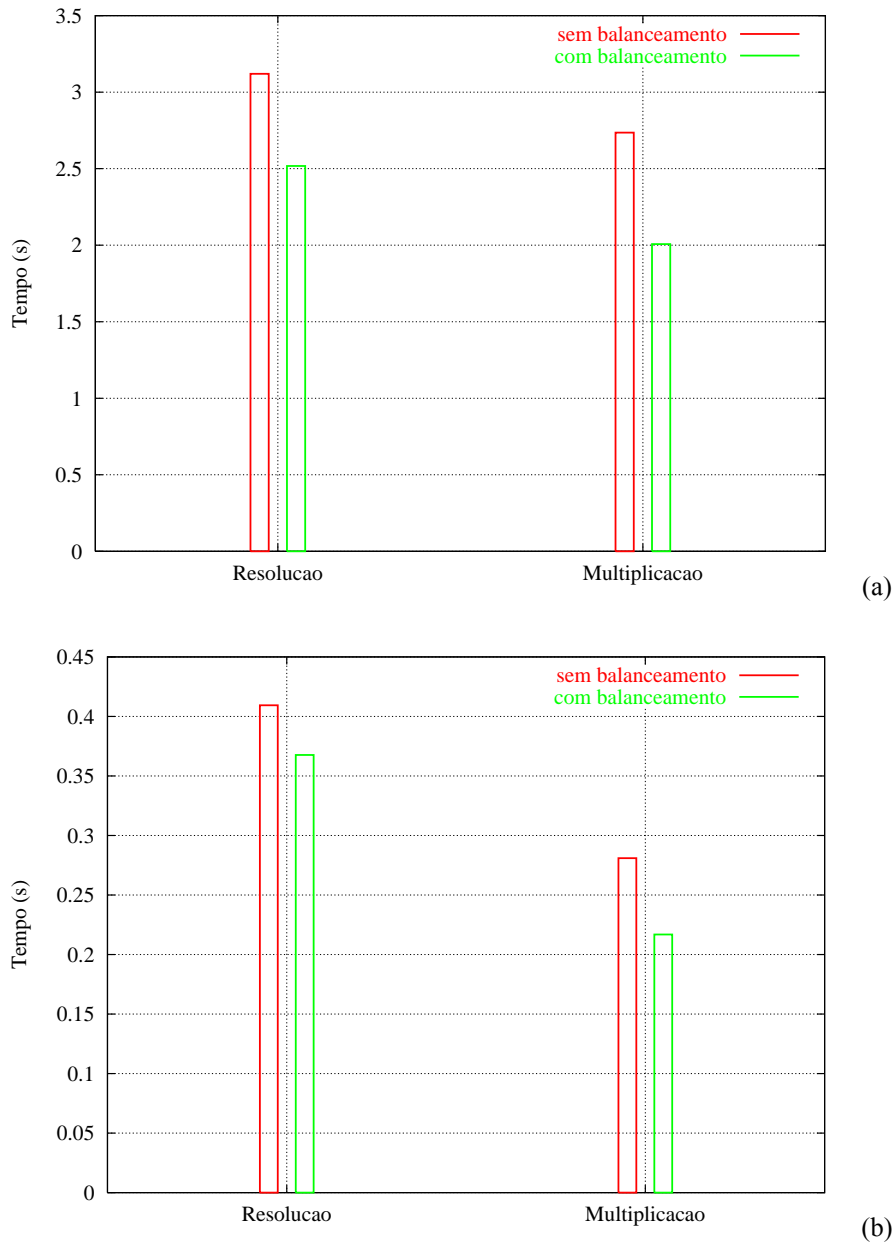


Figura 5.21 - Comparação entre os tempos totais de resolução e multiplicação matriz-vetor: (a) prisma ref. nível 3; (b) cavidade.

A eficiência do balanceamento de carga fica então comprovada com estes resultados, sendo que o ganho final médio no tempo de resolução dos exemplos analisados foi de 15,3% e, no tempo de multiplicação, de 24,5%. Observa-se também que, o ganho de tempo com o balanceamento é proporcional ao tamanho do problema.

Capítulo 6

Considerações Finais

6.1. Conclusões:

Este trabalho teve como objetivo o estudo e o desenvolvimento de estratégias de implementação em paralelo, tanto em redes de PC's homogêneas quanto heterogêneas, do Método dos Elementos de Contorno aplicado à elasticidade tridimensional. Para isso, utilizou-se o código computacional CAV3D, originalmente concebido para utilização seqüencial, além de bibliotecas padrão MPI, de forma a prover capacidade de intercomunicação entre os processadores da rede. As implementações se desenvolveram em ambiente Linux, distribuição *Red Hat*.

Foi traçado aqui um histórico das implementações do MEC em paralelo para diversas aplicações, de forma a mostrar a evolução destas implementações em comparação à evolução das tecnologias empregadas nos computadores. Apresentou-se também os motivos pelos quais se escolheram, como recursos computacionais, as redes ou *clusters* de PC's.

Como foi visto, a escolha de quais etapas seriam implementadas em paralelo deveu-se aos tempos obtidos na análise do código não-paralelizado, no qual verificou-se que cerca de 95% do tempo concentrava-se em três etapas: montagem do sistema de equações lineares, sua resolução e determinação de tensões e deslocamentos no domínio.

Para a etapa de montagem, foram propostas duas formas de distribuição dos nós funcionais fonte entre os processadores envolvidos. Em ambas as formas, cada

processador opera com um conjunto de nós-fonte igual ao quociente inteiro da divisão entre o total de nós-fonte e a quantidade de processadores utilizados (NP) e, caso a divisão não seja exata, um nó é incorporado a cada processador até a distribuição da totalidade dos nós restantes. Na primeira forma de distribuição, chamada seqüencial, cada conjunto é formado por nós-fonte de numeração adjacente. Na outra, chamada cíclica, cada conjunto é formado por nós-fonte de numeração incrementada de NP . De acordo com os resultados apresentados, a segunda forma se mostrou mais eficiente que a primeira, o que ficou evidente quando se mostrou o esforço computacional em cada processador na etapa de montagem. Na forma cíclica, o esforço ficava mais bem distribuído que na forma seqüencial.

Para que se consiga realizar o balanceamento de carga computacional na etapa de resolução do sistema de equações, é necessário que um certo número de linhas da matriz do sistema esteja contido em dois processadores antes do início da resolução do sistema. Para isso, na etapa de montagem, foram propostas duas estratégias: uma utilizando combinação de computação e comunicação e outra não utilizando comunicação. Como apresentado, a estratégia que combina computação e comunicação obteve melhor desempenho que a que não utiliza comunicação. Este resultado depende exclusivamente das características dos processadores e do dispositivo de intercomunicação de rede utilizado. Caso os processadores possuam maior capacidade computacional e o dispositivo de rede apresente uma taxa de transmissão de dados menor, os resultados apresentados indicam que a melhor estratégia é a que não utiliza comunicação.

Para a resolução do sistema de equações lineares, utilizou-se o método GMRES reinicializável com pré-condicionamento diagonal. Como foi mostrado, a única operação desta etapa que foi implementada em paralelo foi a multiplicação entre a

matriz do sistema de equações e um vetor de aproximação, operação esta que consome cerca de 95% do tempo total de resolução não-paralela. De acordo com os resultados apresentados, pode-se observar que a porcentagem do tempo da resolução inerente à multiplicação cresce com o aumento do número de graus de liberdade do problema e diminui com o aumento da quantidade de processadores utilizados. Quanto ao desempenho desta etapa em paralelo, pode-se observar que ela foi eficientemente implementada, apesar de ter-se paralelizado apenas a multiplicação matriz-vetor. Na implementação desta etapa para utilização em redes heterogêneas, observou-se o ganho de tempo de resolução quando se procedeu ao balanceamento da carga computacional. Obviamente, o procedimento de balanceamento passa a ser interessante somente quando o tempo ganho com sua execução é maior que o tempo extra dispendido na execução das etapas deste balanceamento.

Por ser uma etapa naturalmente paralelizável, a determinação de resultados no domínio não possui maiores complicações em sua implementação em paralelo, o que levou a resultados satisfatórios com relação à sua eficiência.

Pode ser observado que, em linhas gerais, a implementação em paralelo do CAV3D foi eficientemente executada. Os resultados poderiam ser um pouco melhores caso a implementação das outras etapas menores, também em paralelo, fosse efetuado.

6.2. Sugestões:

De forma a dar seqüência às atividades realizadas neste trabalho, pode-se propor a análise da viabilidade da implementação das demais etapas e operações em paralelo.

Um outro tópico proposto é o estudo da execução de balanceamento de carga computacional na etapa de montagem, levando em conta as características físicas e funcionais dos computadores e componentes de redes que serão utilizados.

Com relação à resolução do sistema de equações, um outro assunto interessante é o estudo de novas técnicas eficientes de paralelização do próprio GMRES e também de outros métodos iterativos.

Referências Bibliográficas

[1] SYMM, G.T., *Boundary elements on a distributed array processor*. Engineering Analysis with Boundary Elements, v.1, pp. 162-165, 1984.

[2] SCANNELL, E., *IBM demonstrates Linux servers matching supercomputer speeds*. InfoWorld Electric Homepage. Disponível em: <http://www.infoworld.com/cgi-bin/displayStory.pl?99039.ecsuperlinux.htm> . Acesso em 25 fev. 2003.

[3] GEIST, G.A., KOHL, J.A., PAPADOPOULOS, P.M., *PVM and MPI: a comparison of features*. Disponível em: www.wizard.apmath.spbu.ru/~virtlab/PVMvsMPI.pdf. Acesso em: 25 fev. 2003.

[4] BARRA, L.P.S., TOLEDO, E.M., TELLES, J.C.F., *Parallel boundary element method in clusters of PC's*. Relatório de Pesquisa e Desenvolvimento, LNCC, Petrópolis, n.22, 1999.

[5] SAAD, Y., SCHULTZ, M.H., *GMRES – a generalized minimal residual algorithm for solving nonsymmetric linear systems*. SIAM Journal of Scientific Computing, v.7, pp. 869-869, 1986.

[6] BARRA, L.P.S., TELLES, J.C.F., COUTINHO, A.L.G.A. et al., *Iterative solution of BEM equations by GMRES algorithm*. Computer & Structures, v.44, pp. 1249-1253, 1992.

- [7] GARCIA, L.F.T., VILLAÇA, S.F., *Introdução à teoria da elasticidade*. 4.ed., Rio de Janeiro, COPPE/UFRJ, 2000.
- [8] BREBBIA, C.A., TELLES, J.C.F., WROBEL, L.C., *Boundary element techniques: theory and applications in engineering*. 1.ed., Berlin, Springer-Verlag, 1984.
- [9] SAAD, Y., *Iterative methods for sparse linear systems*, 1.ed., Boston, PWS Publishing Company, 1996.
- [10] DIETZ, H., *Linux Parallel Processing HOWTO*. Disponível em: <http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/Parallel-Processing-HOWTO.pdf>. Acesso em 25 fev. 2003.
- [11] GEIST, G.A., ROMINE, C.H., *LU factorization algorithms on distributed-memory multiprocessor architectures*. SIAM Journal of Scientific Computing, v.9, pp. 639-649, 1988.
- [12] DAOUDI, E.M., LOBRY, J., *Implementation of a boundary element method on distributed memory computers*. Parallel Computing, v.18, pp. 1317-1324, 1992.
- [13] HENDRICKSON, B.A., WOMBLE, D.E., *The torus-wrap mapping for dense matrix calculations on massively parallel computers*. SIAM Journal of Scientific Computing, v.15, pp. 1201-1226, 1994.

[14] CARSTENSEN, C., KUHN, M., LANGER, U., *Fast parallel solvers for symmetric boudary element domain decomposition equations*. Numerische Matematik, v.79, pp. 321-347, 1998.

[15] MPI-FORUM, *MPI: a message passing interface standard*. International Journal of Supercomputer Application, v.8, pp. 165-416, 1994.

[16] SANTIAGO, J.A.F., MANSUR, W.J., TELLES, J.C.F. et al., In: *Anais do II Congresso de Engenharia Civil da UFJF*, “Aplicação do método dos elementos de contorno às cavidades do circuito hidráulico do AHE da barragem de Serra da Mesa”, Juiz de Fora, pp. 220-230, agosto, 1996.

[17] SILVA, J.J.R., *MEC3DE – um programa para análise estática tridimensional com o método dos elementos de contorno*. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1989.

Apêndice A

Sub-rotinas MPI Utilizadas

Nesta seção, estão descritas as sub-rotinas padrão MPI utilizadas neste trabalho. Estão definidos os objetivos de cada uma, bem como o significado de cada variável.

- `MPI_init (ierr)` – primeira rotina a ser chamada, inicializando o ambiente MPI no código.

Tabela A.7 – variáveis da sub-rotina `MPI_init`

Variável	Tipo	I/O	Definição
<code>ierr</code>	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- `MPI_finalize(ierr)` – fecha as portas de comunicação dentro do código até outra chamada à rotina `MPI_init`.

Tabela A.8 - variáveis da sub-rotina `MPI_finalize`

Variável	Tipo	I/O	Definição
<code>ierr</code>	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- `MPI_comm_size(comm, size, error)` – retorna a quantidade de processadores utilizados em tempo de execução.

Tabela A.9 - variáveis da sub-rotina `MPI_comm_size`

Variável	Tipo	I/O	Definição
<code>comm</code>	inteiro	entrada	comunicador
<code>size</code>	inteiro	saída	quantidade de processadores inicializados
<code>ierr</code>	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- `MPI_comm_rank(comm, rank, ierr)` – retorna o ranking do processador que a chama em relação aos demais.

Tabela A.10 - variáveis da sub-rotina `MPI_comm_rank`

Variável	Tipo	I/O	Definição
<code>comm</code>	inteiro	entrada	comunicador
<code>rank</code>	inteiro	saída	ranking do processador
<code>ierr</code>	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- `MPI_barrier(comm, ierr)` – bloqueia o processador que faz a chamada até que todos os demais a tenham chamado.

Tabela A.11 - variáveis da sub-rotina `MPI_barrier`

Variável	Tipo	I/O	Definição
<code>comm</code>	inteiro	entrada	comunicador
<code>ierr</code>	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- `MPI_gatherv(sendbuf, sendcount, sendtype, recvbuf, recvcoun-
ts, displs, recvtype, root, comm, ierr)` – junta os trechos `sendbuf` enviados por cada processador que fez a chamada ao processador de rank `root` no array `recvbuf`.

Tabela A.12 - variáveis da sub-rotina MPI_gatherv

Variável	Tipo	I/O	Definição
sendbuf	a definir	entrada	endereço do início do <i>buffer</i> de saída
sendcount	inteiro	entrada	quantidade de elementos no <i>buffer</i> de saída
sendtype	inteiro	entrada	tipo de dados no buffer de saída
recvbuf	a definir	saída	Endereço do início do <i>buffer</i> de entrada no processador de rank <i>root</i>
recvcounts	vetor inteiro	entrada	quantidade de elementos nos <i>buffers</i> de saída de todos os processadores
displs	vetor inteiro	entrada	posição dos dados vindos dos demais processadores dentro do <i>buffer</i> de entrada
recvtype	inteiro	entrada	tipo dos dados do <i>buffer</i> de entrada
root	inteiro	entrada	rank do processador que recebe os dados
comm	inteiro	entrada	comunicador
ierr	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- MPI_bcast (buffer, count, datatype, root, comm, ierr) – distribui a todos os processadores que fizeram a chamada os dados contidos em buffer oriundos do processador de rank root.

Tabela A.13 - variáveis da sub-rotina MPI_bcast

Variável	Tipo	I/O	Definição
buffer	a definir	entrada/saída	endereço do início do <i>buffer</i> de saída
count	inteiro	entrada	quantidade de dados no <i>buffer</i> de saída
datatype	inteiro	entrada	tipo dos dados no <i>buffer</i> de saída
root	inteiro	entrada	rank do processador que envia os dados
comm	inteiro	entrada	comunicador
ierr	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- MPI_send (buf, count, datatype, dest, tag, comm, ierr) – envia os dados contidos em buf do processador que faz a chamada para o processador de rank dest.

Tabela A.14 - variáveis da sub-rotina MPI_send

Variável	Tipo	I/O	Definição
buf	a definir	entrada	endereço do início do <i>buffer</i> de saída
count	inteiro	entrada	quantidade de dados no <i>buffer</i> de saída
datatype	inteiro	entrada	tipo dos dados no <i>buffer</i> de saída
dest	inteiro	entrada	rank do processador que recebe os dados
tag	inteiro	entrada	identificador da mensagem
comm	inteiro	entrada	comunicador
ierr	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

- MPI_recv(buf, count, datatype, source, tag, comm, status, ierr)
 - o processador que faz a chamada recebe em buf o que foi enviado pelo processador de rank source.

Tabela A.15 - variáveis da sub-rotina MPI_recv

Variável	Tipo	I/O	Definição
buf	a definir	saída	endereço do início do <i>buffer</i> de entrada
count	inteiro	entrada	quantidade de dados no <i>buffer</i> de entrada
datatype	inteiro	entrada	tipo dos dados no <i>buffer</i> de saída
source	inteiro	entrada	rank do processador que envia os dados
tag	inteiro	entrada	identificador
comm	inteiro	entrada	comunicador
status	vetor inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha
ierr	inteiro	saída	retorna 0 em caso de sucesso ou o código do erro em caso de falha

Apêndice B

Resultados Utilizados nos Gráficos

Nesta seção, encontram-se os resultados obtidos nas análises executadas, a partir dos quais foram confeccionados os gráficos expostos no quinto capítulo deste trabalho.

Os dados apresentados nas tabelas B.1 à B.5 deram origem aos gráficos apresentados nas figuras 5.2 à 5.8 e 5.15.

Tabela B.1 – Tempos de cada etapa do código sem balanceamento com montagem sequencial

Prisma refinamento nível 1				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	3.571213	3.792918	8.079685	12.393499
4	1.814969	2.074211	4.249132	6.846756
8	0.966374	1.28279	2.136745	3.965945
Prisma refinamento nível 2				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	8.214959	8.577563	15.494823	24.802109
4	4.131363	4.551902	8.225732	13.536587
8	2.129751	2.610816	4.16574	7.57886
Prisma refinamento nível 3				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	14.665229	15.169835	25.36168	41.443347
4	7.354027	7.929945	13.505758	22.38552
8	3.786037	4.447452	6.871079	12.294086

Tabela B.2 - Tempos de cada etapa do código sem balanceamento com montagem cíclica

Prisma refinamento nível 1				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	3.469579	3.739527	8.267011	12.528216
4	1.796918	2.124284	4.1244	6.767662
8	0.930385	1.274693	2.087825	3.882359
Prisma refinamento nível 2				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	7.941766	8.372817	15.817829	24.90567
4	4.03577	4.523197	7.94599	13.184169
8	2.096347	2.641228	4.489498	7.876937
Prisma refinamento nível 3				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	14.172353	14.762666	26.095364	41.768305
4	7.161561	7.827763	13.096085	21.83457
8	3.648221	4.393348	6.580814	11.923065

Tabela B.3 - Tempos de cada etapa do código com balanceamento e montagem sequencial, sem passagem de mensagem

Prisma refinamento nível 1				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	3.56847	3.792348	10.245974	14.559706
4	1.820004	2.0812	6.297003	8.902388
8	0.976812	1.272287	3.22391	5.718518
Prisma refinamento nível 2				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	8.155795	8.518514	19.632372	28.868885
4	4.140799	4.562335	12.190183	17.485697
8	2.14821	2.624197	6.912765	10.288587
Prisma refinamento nível 3				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	14.777679	15.282989	32.165481	48.362434
4	7.321321	7.898146	20.017732	28.842444
8	3.790526	4.444707	10.26796	15.665139

Tabela B.4 - Tempos de cada etapa do código com balanceamento, montagem sequencial e passagem de mensagem

Prisma refinamento nível 1				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	3.56609	3.839347	8.639995	13.002292
4	1.831166	2.137474	4.73377	7.392345
8	0.986516	1.333343	2.415145	4.270205
Prisma refinamento nível 2				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	8.189018	8.619518	16.727952	26.066775
4	4.14724	4.635211	9.343227	14.696657
8	2.151093	2.698819	5.477127	8.093338
Prisma refinamento nível 3				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	14.555262	15.144261	27.815064	43.885298
4	7.349598	8.012429	15.533003	24.467274
8	3.781659	4.529669	8.623228	14.0786

Tabela B.5 - Tempos de cada etapa do código com balanceamento, montagem cíclica e passagem de mensagem

Prisma refinamento nível 1				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	3.477111	3.751229	8.686177	12.955933
4	1.775203	2.084931	4.513639	7.116153
8	0.951075	1.296578	2.963939	4.823958
Prisma refinamento nível 2				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	7.969218	8.40017	16.776973	25.893117
4	4.045709	4.5334	8.955156	14.200814
8	2.098654	2.650423	5.217867	8.581952
Prisma refinamento nível 3				
Nº. de Processadores	Multiplicação	Resolução	Montagem	Total
2	14.194109	14.783042	27.908969	43.601173
4	7.160188	7.823488	14.734573	23.467594
8	3.670343	4.414962	7.456686	12.783012

Nas tabelas B.6 e B.7 estão os dados utilizados nos gráficos das figuras 5.9 à 5.13 e na tabela B.8 estão os utilizados no gráfico da figura 5.14.

Tabela B.6 - Tempo de montagem em cada processador utilizando distribuição sequencial e cíclica de nós funcionais: 4 processadores

Processador	Prisma ref. Nível 1		Prisma ref. Nível 2		Prisma ref. Nível 3		Cavidade	
	Seq.	Cic.	Seq.	Cic.	Seq.	Cic.	Seq.	Cic.
1	3.85568	4.13316	7.34786	7.91576	12.0923	13.0733	4.08543	4.2182
2	4.35959	4.13352	8.43593	7.91731	13.9232	13.0476	4.4087	4.21535
3	4.35558	4.13528	8.43199	7.92557	13.8827	13.0637	4.00482	4.19606
4	3.8647	4.13472	7.37025	7.91108	12.134	13.0417	4.18731	4.20934

Tabela B.7 - Tempo de montagem em cada processador utilizando distribuição sequencial e cíclica de nós funcionais: 8 processadores

Processador	Prisma ref. Nível 1		Prisma ref. Nível 2		Prisma ref. Nível 3		Cavidade	
	Seq.	Cic.	Seq.	Cic.	Seq.	Cic.	Seq.	Cic.
1	1.89639	2.07219	3.54777	3.98509	5.81014	6.55263	2.0932	2.13003
2	2.00425	2.08954	3.847	3.98962	6.35051	6.54857	2.01644	2.09394
3	2.14801	2.08847	4.16809	3.99532	6.83942	6.55533	2.12963	2.09882
4	2.21477	2.08512	4.29752	3.97768	7.09177	6.54936	2.3096	2.09348
5	2.22626	2.07732	4.29454	3.96613	7.10369	6.53854	1.97377	2.11962
6	2.16018	2.06456	4.16858	3.97402	6.83872	6.54666	2.24347	2.1213
7	2.00211	2.08109	3.858	3.97411	6.3487	6.53387	2.2811	2.1217
8	1.91106	2.07766	3.54443	3.96505	5.83233	6.5374	1.93375	2.13144

Tabela B.8 - Resultados no domínio para o problema da cavidade

Nº. de Processadores	Tempo
1	2.58534
2	1.26067
4	0.65431
8	0.33283

As tabelas B.9 à B.12 contêm os dados expostos nos gráficos das figuras 5.16 à

5.19.

Tabela B.9 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 1

Iteração	Com balanceamento				Sem balanceamento			
1	0.01444	0.02025	0.01221	0.01371	0.01443	0.02025	0.01219	0.01367
2	0.01461	0.01517	0.01446	0.01434	0.01445	0.02027	0.01243	0.01368
3	0.01457	0.01517	0.01445	0.01434	0.01441	0.02027	0.01243	0.01367
4	0.01444	0.01517	0.01443	0.01436	0.01438	0.02045	0.01245	0.01367
5	0.01457	0.01518	0.01443	0.01436	0.01437	0.02026	0.01243	0.01366
6	0.01457	0.01518	0.01443	0.01436	0.01441	0.02026	0.01243	0.01367
7	0.01456	0.01518	0.01445	0.01436	0.01442	0.02026	0.01218	0.01368
8	0.01457	0.01517	0.01444	0.01436	0.01439	0.02026	0.01243	0.01368
9	0.01458	0.01518	0.01443	0.01437	0.01439	0.02026	0.01243	0.01367
10	0.01460	0.01518	0.01443	0.01435	0.01443	0.02038	0.01243	0.01367
11	0.01458	0.01518	0.01443	0.01435	0.01442	0.02033	0.01218	0.01368
12	0.01458	0.01518	0.01444	0.01435	0.01441	0.02026	0.01218	0.01368
13	0.01460	0.01517	0.01443	0.01434	0.01443	0.02026	0.01217	0.01368
14	0.01457	0.01516	0.01445	0.01434	0.01441	0.02026	0.01223	0.01367
15	0.01458	0.01517	0.01444	0.01436	0.01444	0.02028	0.01225	0.01367
16	0.01460	0.01517	0.01443	0.01434	0.01442	0.02028	0.01218	0.01367
17	0.01459	0.01515	0.01447	0.01434	0.01441	0.02026	0.01218	0.01368
18	0.01457	0.01517	0.01444	0.01436	0.01443	0.02026	0.01217	0.01367
19	0.01459	0.01517	0.01443	0.01434	0.01440	0.02027	0.01217	0.01367
20	0.01456	0.01517	0.01445	0.01435	0.01443	0.02027	0.01243	0.01368
21	0.01456	0.01517	0.01443	0.01433	0.01441	0.02028	0.01219	0.01367
22	0.01457	0.01518	0.01445	0.01435	0.01442	0.02026	0.01243	0.01367
23	0.01456	0.01517	0.01443	0.01434	0.01442	0.02026	0.01219	0.01366
24	0.01456	0.01517	0.01445	0.01436	0.01441	0.02026	0.01218	0.01367
25	0.01456	0.01518	0.01445	0.01439	0.01443	0.02028	0.01245	0.01367
26	0.01457	0.01518	0.01445	0.01443	0.01442	0.02026	0.01217	0.01367
27	0.01457	0.01517	0.01444	0.01448	0.01443	0.02028	0.01218	0.01368
28	0.01458	0.01517	0.01443	0.01434	0.01442	0.02029	0.01221	0.01367
29	0.01459	0.01518	0.01441	0.01434	0.01444	0.02026	0.01218	0.01367
30	0.01459	0.01517	0.01457	0.01434	0.01441	0.02026	0.01217	0.01366
31	0.01456	0.01517	0.01443	0.01435	0.01442	0.02028	0.01243	0.01367
32	0.01457	0.01518	0.01444	0.01436	0.01442	0.02027	0.01218	0.01367

Tabela B.10 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 2

Iteração	Com balanceamento				Sem balanceamento			
1	0.02879	0.04139	0.02464	0.02772	0.02878	0.04139	0.02467	0.02777
2	0.02926	0.03041	0.02938	0.02905	0.02877	0.04141	0.02466	0.02776
3	0.02909	0.03039	0.02938	0.02904	0.02877	0.04139	0.02464	0.02776
4	0.02935	0.03041	0.02938	0.02961	0.02876	0.04144	0.02466	0.02777
5	0.02936	0.03039	0.02937	0.02903	0.02883	0.04141	0.02466	0.02776
6	0.02933	0.03041	0.02981	0.02904	0.02884	0.04140	0.02466	0.02778
7	0.02935	0.03039	0.02937	0.02904	0.02883	0.04139	0.02465	0.02775
8	0.02936	0.03115	0.02937	0.02904	0.02884	0.04138	0.02465	0.02776
9	0.02934	0.03055	0.02937	0.02904	0.02884	0.04141	0.02466	0.02778
10	0.02935	0.03039	0.02937	0.02904	0.02885	0.04139	0.02467	0.02777
11	0.02935	0.03040	0.02937	0.02905	0.02882	0.04142	0.02465	0.02783
12	0.02933	0.03041	0.02936	0.02904	0.02884	0.04138	0.02466	0.02776
13	0.02934	0.03039	0.02937	0.02904	0.02883	0.04139	0.02471	0.02779
14	0.02935	0.03038	0.02938	0.02904	0.02885	0.04146	0.02467	0.02778
15	0.02936	0.03041	0.02937	0.02904	0.02883	0.04139	0.02464	0.02778
16	0.02934	0.03040	0.02938	0.02905	0.02884	0.04139	0.02464	0.02776
17	0.02934	0.03046	0.02937	0.02904	0.02884	0.04140	0.02466	0.02778
18	0.02935	0.03039	0.02937	0.02904	0.02882	0.04140	0.02466	0.02776
19	0.02934	0.03079	0.02937	0.02903	0.02884	0.04142	0.02467	0.02776
20	0.02930	0.03039	0.02938	0.02902	0.02882	0.04139	0.02464	0.02798
21	0.02936	0.03041	0.02937	0.02905	0.02884	0.04139	0.02465	0.02776
22	0.02952	0.03040	0.02936	0.02904	0.02883	0.04150	0.02465	0.02777
23	0.02935	0.03039	0.02937	0.02905	0.02883	0.04145	0.02465	0.02776
24	0.02935	0.03041	0.02937	0.02904	0.02883	0.04140	0.02466	0.02776
25	0.02932	0.03038	0.02936	0.02925	0.02884	0.04141	0.02466	0.02777
26	0.02936	0.03040	0.02938	0.02905	0.02883	0.04140	0.02466	0.02776
27	0.02933	0.03047	0.02959	0.02904	0.02884	0.04140	0.02465	0.02776
28	0.02935	0.03041	0.02943	0.02904	0.02884	0.04174	0.02467	0.02777
29	0.02933	0.03039	0.02936	0.02906	0.02882	0.04139	0.02465	0.02777
30	0.02935	0.03042	0.02949	0.02907	0.02884	0.04139	0.02467	0.02778
31	0.02997	0.03041	0.02937	0.02903	0.02882	0.04139	0.02466	0.02776
32	0.02934	0.03041	0.02937	0.02904	0.02884	0.04158	0.02466	0.02778
33	0.02935	0.03039	0.02937	0.02904	0.02884	0.04147	0.02466	0.02776
34	0.02934	0.03040	0.02939	0.02902	0.02883	0.04139	0.02465	0.02777
35	0.02937	0.03039	0.02937	0.02904	0.02884	0.04139	0.02470	0.02777
36	0.02936	0.03040	0.02945	0.02963	0.02884	0.04144	0.02479	0.02884

Tabela B.11 - Tempos de cada iteração utilizando ou não balanceamento: prisma refinamento nível 3

Iteração	Com balanceamento				Sem balanceamento			
1	0.04802	0.06997	0.04247	0.04677	0.04817	0.07004	0.04155	0.04676
2	0.04978	0.05085	0.04853	0.04985	0.04806	0.07001	0.04157	0.04680
3	0.04974	0.05025	0.04854	0.04986	0.04797	0.06997	0.04154	0.04676
4	0.04973	0.05026	0.04854	0.04988	0.04797	0.07001	0.04248	0.04678
5	0.04977	0.05027	0.04856	0.04989	0.04807	0.06995	0.04245	0.04697
6	0.04971	0.05047	0.04860	0.04988	0.04806	0.07016	0.04245	0.04678
7	0.04972	0.05026	0.04852	0.04988	0.04807	0.06996	0.04162	0.04679
8	0.04972	0.05026	0.04853	0.04990	0.04806	0.06995	0.04245	0.04678
9	0.04974	0.05028	0.04853	0.04986	0.04806	0.07005	0.04244	0.04684
10	0.04972	0.05025	0.04852	0.04988	0.04806	0.06995	0.04311	0.04677
11	0.04985	0.05044	0.04850	0.05116	0.04807	0.06997	0.04154	0.04677
12	0.04973	0.05026	0.04853	0.04993	0.04805	0.07004	0.04156	0.04677
13	0.04973	0.05027	0.04997	0.04989	0.04808	0.06995	0.04155	0.04678
14	0.04972	0.05026	0.04853	0.04986	0.04807	0.06999	0.04153	0.04818
15	0.04971	0.05032	0.04854	0.04988	0.04798	0.06998	0.04332	0.04678
16	0.04972	0.05027	0.04854	0.04988	0.04805	0.06996	0.04154	0.04678
17	0.04972	0.05028	0.04852	0.04988	0.04893	0.07005	0.04154	0.04678
18	0.04973	0.05028	0.04853	0.05137	0.04806	0.06997	0.04154	0.04678
19	0.04975	0.05026	0.04850	0.04988	0.04806	0.07002	0.04154	0.04675
20	0.04970	0.05025	0.04899	0.04986	0.04812	0.07005	0.04157	0.04677
21	0.04971	0.05025	0.04852	0.04986	0.04806	0.06996	0.04154	0.04678
22	0.04971	0.05034	0.04854	0.04988	0.04807	0.07077	0.04154	0.04678
23	0.04977	0.05028	0.04853	0.04988	0.04806	0.07002	0.04154	0.04678
24	0.04971	0.05026	0.04854	0.04989	0.04803	0.06997	0.04156	0.04678
25	0.04972	0.05027	0.04853	0.04986	0.04809	0.07021	0.04155	0.04678
26	0.04973	0.05027	0.04852	0.04988	0.04805	0.07004	0.04160	0.04678
27	0.04972	0.05032	0.04854	0.04986	0.04807	0.06997	0.04156	0.04678
28	0.04971	0.05026	0.04852	0.04988	0.04804	0.06995	0.04168	0.04678
29	0.04972	0.05027	0.04853	0.04988	0.04806	0.06995	0.04154	0.04678
30	0.04973	0.05025	0.04853	0.04987	0.04805	0.06996	0.04154	0.04684
31	0.05078	0.05034	0.04851	0.05133	0.04807	0.07001	0.04294	0.04678
32	0.05076	0.05027	0.04853	0.04990	0.04807	0.07001	0.04153	0.04678
33	0.04971	0.05031	0.04882	0.04988	0.04805	0.06995	0.04158	0.04678
34	0.04971	0.05027	0.04853	0.04988	0.04804	0.06998	0.04154	0.04677
35	0.05079	0.05183	0.04853	0.04989	0.04806	0.07004	0.04154	0.04677
36	0.04972	0.05034	0.04851	0.04988	0.04805	0.06994	0.04156	0.04677
37	0.04972	0.05025	0.04851	0.04986	0.04806	0.06996	0.04154	0.04677
38	0.04973	0.05027	0.04856	0.05024	0.04806	0.06994	0.04155	0.04677

Tabela B.12 - Tempos de cada iteração utilizando ou não balanceamento: cavidade

Iteração	Com balanceamento				Sem balanceamento			
	1	0.00802	0.01114	0.00678	0.00757	0.00809	0.01114	0.00668
2	0.00818	0.00837	0.00792	0.00792	0.00816	0.01114	0.00669	0.00748
3	0.00820	0.00839	0.00791	0.00793	0.00813	0.01113	0.00667	0.00750
4	0.00817	0.00838	0.00791	0.00792	0.00833	0.01114	0.00670	0.00749
5	0.00818	0.00838	0.00791	0.00794	0.00812	0.01113	0.00669	0.00749
6	0.00816	0.00838	0.00791	0.00792	0.00810	0.01114	0.00669	0.00748
7	0.00818	0.00838	0.00791	0.00792	0.00812	0.01114	0.00668	0.00749
8	0.00815	0.00838	0.00791	0.00793	0.00813	0.01113	0.00668	0.00749
9	0.00817	0.00839	0.00791	0.00794	0.00811	0.01114	0.00668	0.00750
10	0.00820	0.00838	0.00791	0.00793	0.00813	0.01114	0.00667	0.00749
11	0.00817	0.00840	0.00791	0.00793	0.00813	0.01114	0.00668	0.00748
12	0.00818	0.00839	0.00792	0.00793	0.00813	0.01113	0.00669	0.00749
13	0.00815	0.00838	0.00791	0.00793	0.00815	0.01114	0.00669	0.00750
14	0.00817	0.00838	0.00792	0.00793	0.00811	0.01112	0.00669	0.00750
15	0.00818	0.00840	0.00792	0.00791	0.00815	0.01112	0.00668	0.00749
16	0.00818	0.00838	0.00793	0.00793	0.00813	0.01114	0.00668	0.00750
17	0.00816	0.00838	0.00791	0.00792	0.00813	0.01114	0.00668	0.00748
18	0.00818	0.00839	0.00791	0.00792	0.00813	0.01113	0.00670	0.00749
19	0.00817	0.00839	0.00791	0.00793	0.00811	0.01116	0.00668	0.00750
20	0.00818	0.00838	0.00792	0.00792	0.00813	0.01114	0.00667	0.00748
21	0.00820	0.00838	0.00792	0.00793	0.00811	0.01112	0.00669	0.00750
22	0.00818	0.00838	0.00791	0.00793	0.00813	0.01115	0.00668	0.00749
23	0.00818	0.00838	0.00792	0.00794	0.00812	0.01120	0.00669	0.00750
24	0.00834	0.00837	0.00791	0.00793	0.00814	0.01113	0.00667	0.00750

Na tabela B.13 estão os dados expostos nos gráficos das figuras 5.20 e 5.21.

Tabela B.13 – Tempos da etapa de resolução e operação de multiplicação matriz-vetor utilizando ou não balanceamento de carga

Etapa	Prisma ref. Nivel 1		Prisma ref. Nivel 2		Prisma ref. Nivel 3		Cavidade	
	s/b	c/b	s/b	c/b	s/b	c/b	s/b	c/b
Resolução	0.8713	0.75	1.8313	1.5021	3.12	2.5177	0.4094	0.3676
Multiplicação	0.6717	0.5134	1.5363	1.1518	2.7357	2.0073	0.2809	0.2168