

MARCOS FISCHBORN

**COMPUTAÇÃO DE ALTO DESEMPENHO
APLICADA À ANÁLISE DE DISPOSITIVOS
ELETROMAGNÉTICOS**

FLORIANÓPOLIS

2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA**

**COMPUTAÇÃO DE ALTO DESEMPENHO
APLICADA À ANÁLISE DE DISPOSITIVOS
ELETROMAGNÉTICOS**

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

MARCOS FISCHBORN

Florianópolis, Outubro de 2006.

COMPUTAÇÃO DE ALTO DESEMPENHO APLICADA À ANÁLISE DE DISPOSITIVOS ELETROMAGNÉTICOS

Marcos Fischborn

‘Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em *Concepção e Análise de Dispositivos Eletromagnéticos*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.’

Patrick Kuo-Peng, Dr.
Orientador

Nelson Sadowski, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Patrick Kuo-Peng, Dr.
Presidente

Nelson Sadowski, Dr.

João Pedro Assumpção Bastos, Dr.

Renato Cardoso Mesquita, Dr.

Marcelo Grafulha Vanti, Dr.

AGRADECIMENTOS

Esta tese é o resultado de muitos anos de trabalho e dedicação. Não só os anos dedicados exclusivamente a ela, mas muitos outros anos que a precederam numa forma de preparação, onde o universo conspirou a meu favor...

Mas a mesma não teria encontrado fim não fosse o apoio de inúmeras pessoas que direta ou indiretamente colaboraram para que a jornada tivesse êxito. Muitas foram as pessoas que contribuíram com esta pesquisa. Mesmo porque o assunto aqui tratado utiliza muitos recursos de hardware, software e conhecimentos teóricos que extrapolam meus conhecimentos. Foi graças a ajuda de diversas destas pessoas que o trabalho foi realizado. É bem provável que os resultados mostrados nesta tese teriam um alcance menor se contassem apenas com o autor.

Mas um projeto de quatro anos também precisa de um constante tratamento da motivação pessoal. Desta forma, se alguns me auxiliaram tecnicamente outros me auxiliaram com entusiasmo e confiança. Assim, meu muito obrigado a todos.

Gostaria de agradecer especialmente a minha esposa Marci, que levou no ventre, durante esta tese, nossos projetos mais especiais. Além disso, administrou nossa vida à distância durante este tempo e mostrou uma força e uma paciência tão grandes que apenas uma mulher com o amor dela poderia prover. Muito obrigado pelo apoio ilimitado durante esta jornada. Sem você isto realmente não seria possível.

Gostaria de agradecer muito, também, ao meu filho Nicolas, nascido durante esta etapa da vida. Não pude acompanhá-lo em muitos dos seus momentos iniciais, assim faltei com ele nestas ocasiões. Mas mesmo assim esta pequena criança sempre me recebeu com aquele sorriso adorável e com um pedido de colo com seus braços estendidos. Muito obrigado por isto, meu filho.

Ao meu filho Arthur, que agora ainda está a caminho e por isso pouco nos conhecemos, digo que esperamos você com muito amor. Você é muito bem vindo ao nosso lar.

Agradeço também aos meus pais Lotario e Neide, que forneceram um apoio muito importante para este trabalho. Podem ter certeza que mais do que eu e a Marci, o Nicolas sabe da importância das suas presenças nas nossas vidas.

Ao Prof. Patrick Kuo-Peng, uma menção especial pela sua orientação. Começamos um trabalho pioneiro no laboratório, com nenhuma experiência nesta área, mas muita motivação. Contudo, estes elementos não removem as dificuldades nem as decepções com resultados encontrados, que, aliás, somam muitas. Assim, agradeço a confiança depositada e o impulso de perseverança que foram tão fundamentais ao trabalho quanto a disponibilidade de tempo e de recursos financeiros para o projeto da tese.

Agradeço ao Prof. Nelson Sadowski, co-orientador, pela participação em momentos cruciais deste trabalho. Houve muitas etapas onde a experiência deste professor foi decisiva para o acerto dos rumos da pesquisa.

Agradeço ao Prof. João Pedro Assumpção Bastos não só pela colaboração na obtenção das matrizes de grande porte testadas neste trabalho; tenho noção do tempo precioso que a modificação necessária aos programas tomou. Gostaria de agradecer também pela oportunidade de trabalhar com ele desde os tempos de monitoria, no início da década de 1990. Gostaria de deixar claro neste texto que foi esta oportunidade que me permitiu conhecer melhor a área de dispositivos eletromagnéticos e que me conduziu ao GRUCAD.

Também não devo esquecer o apoio do ex-professor do Departamento de Engenharia Elétrica e do GRUCAD, meu orientador de mestrado, o Prof. Sérgio Roberto Arruda. A oportunidade de trabalhar como Engenheiro neste laboratório me levou ao mestrado no GRUCAD e foi uma etapa decisiva, que primeiro me inseriu no mercado de trabalho, e depois permitiu que esta tese tivesse um princípio e um fim.

Agradeço ao Prof. Mauricio Valência Ferreira da Luz pela paciente correção de diversas partes do presente texto e pela agradável troca de idéias, que muito somou ao trabalho.

Agradeço aos Professores Nelson Jhoé Batistela e Walter Carpes Jr. pela amizade e convivência tranqüila.

Este trabalho também deve muito aos meus fiéis escudeiros Julio Trevisan e George Arthur Gavioli. São partes importantes desta tese os resultados obtidos com seus

projetos de pesquisa. Tenho certeza que a dedicação demonstrada por eles ao longo deste tempo de trabalho nada mais é que uma imagem do sucesso que terão em suas carreiras profissionais.

Lembro e agradeço, também, o auxílio fundamental do professor do CEFET/SC, Prof. Daniel Dotta, pelo trabalho de montagem, instalação e manutenção do cluster de computadores montado como parte integrante deste projeto.

Agradeço também a todos os colegas do grupo de pesquisa pela longa e proveitosa convivência. Agradeço especialmente aos colegas Jean Viane Leite, Sérgio Luciano Ávila, Claudenei Simão, Orlando José Antunes, Marconi Januário, Mauricio Rigoni e Celly D. Melo.

Não posso deixar de lembrar a colaboração dos Professores Ildemar Cassana Decker e Jorge Mário Campagnolo, professores dos laboratórios LabPlan e LABSPOT, respectivamente, que muito auxiliaram nas fases iniciais deste trabalho, fornecendo bibliografia e acesso ao cluster do LabPlan.

Além disso, agradeço a ajuda de além-mar fornecida prontamente pelo Prof. Yvonnick Le-Menach, professor do L2EP, na Universidade de Lille, França, pelo acesso ao cluster daquela instituição.

Agradeço a minha instituição de Ensino, a Universidade Tecnológica Federal do Paraná – UTFPR - e especialmente ao Campus de Medianeira, por terem me fornecido a oportunidade de me ausentar da Instituição durante estes anos. Ao amigo, Prof. José Airton Azevedo dos Santos, meus sinceros agradecimentos.

Agradeço também ao Nosso Bom Deus, que sempre me iluminou nesta estrada escura e sinuosa, e sempre me concedeu força para continuar a caminhada. Literalmente, muitos foram os quilômetros percorridos durante estes anos, e durante todo este tempo contei com sua infinita bondade e proteção. Louvado seja.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica.

COMPUTAÇÃO DE ALTO DESEMPENHO APLICADA À ANÁLISE DE DISPOSITIVOS ELETROMAGNÉTICOS

Marcos Fischborn

Out./2006

Orientador: Patrick Kuo-Peng, Dr.

Co-orientador: Nelson Sadowski, Dr.

Área de Concentração: Análise e Concepção de Dispositivos Eletromagnéticos

Palavras-chave: Computação Paralela, Elementos Finitos, Métodos Numéricos, Análise Numérica.

Número de páginas: 206

O presente trabalho aborda um estudo envolvendo técnicas utilizadas em computação de alto desempenho, avaliando métodos numéricos, pré-condicionadores, esquemas de armazenamento de matrizes esparsas e tecnologias de rede de comunicação focando, principalmente, seu uso em sistemas de equações originários da análise de elementos finitos, utilizando clusters de computadores comerciais. Neste trabalho são apresentados diversos resultados com os quais se procurou encontrar, dentre o conjunto de técnicas e ferramentas utilizadas, as opções mais adequadas para o desenvolvimento de programas de computador e ferramentas de auxílio ao projeto em computador utilizando elementos finitos, que venham a se utilizar de clusters de computadores com memória distribuída.

Abstract of Thesis presented as a partial fulfillment of the requirements for the degree of
Doctor in Electrical Engineering.

HIGH PERFORMANCE COMPUTING APPLIED TO THE ANALYSIS OF ELECTROMAGNETIC DEVICES

Marcos Fischborn

Oct./2006

Advisor: Patrick Kuo-Peng, Dr.

Co-advisor: Nelson Sadowski, Dr.

Area of Concentration: Analysis and Conception of Electromagnetic Devices

Keywords: Parallel Computation, Finite Elements, Numerical Methods, Numerical
Analysis.

Number of pages: 206

This work is a study related to techniques applied to high performance computing, evaluating numerical methods, preconditioners, sparse matrix storage schemes e network technologies, focusing its use mainly in systems of equations obtained from finite element analysis, using commercial computer clusters. It presents several results intended to demonstrate, among a large set of used techniques and tools, the more suitable options to be applied on the development of computer programs and in computer aided design tools making use of finite elements method with computer clusters and distributed memory.

SUMÁRIO

LISTA DE FIGURAS.....	x
LISTA DE TABELAS.....	xii
LISTA DE SIGLAS.....	xiv
LISTA DE SÍMBOLOS.....	xvi
1 Introdução	1
1.1 Preliminares: a necessidade de Computação de Alto Desempenho em setores atuais de pesquisa	1
1.2 O estado da arte dos métodos numéricos iterativos, estudos de pré-condicionadores e aplicação de elementos finitos.	4
1.3 As propostas da tese e suas contribuições inéditas	10
1.4 Estrutura do texto	16
1.5 Notação	18
2 Elementos Finitos e suas Matrizes	19
2.1 Introdução	19
2.2 Equações do dispositivo eletromagnético	19
2.3 Análise da estrutura do sistema matricial	26
2.4 Conclusão	29
3 Métodos Iterativos	31
3.1 Introdução	31
3.2 Iteração básica	31
3.3 As abordagens para o subespaço de Krylov	35
3.4 A escolha do método	38
3.5 Os métodos implementados	39
3.5.1 Método do Gradiente Conjugado (CG)	39
3.5.2 Método Gradiente Biconjugado (BiCG)	41
3.5.3 Método Gradiente Biconjugado Estabilizado (BiCGStab)	44
3.5.4 Método Gradiente Conjugado Quadrado (CGS)	46
3.6 Conclusão	48
4 Pré-condicionamento	49
4.1 Introdução	49
4.2 A inserção do pré-condicionamento nos métodos iterativos	49
4.3 As classes de pré-condicionadores	54
4.3.1 Pré-condicionadores baseados em Decomposições Incompletas	55
4.3.2 Decomposições Incompletas convencionais	56
4.3.3 Métodos de Decomposição por blocos	56
4.4 O Pré-condicionador com fatoração LU por blocos com interseção (FLUI)	57
4.5 Conclusão.....	62
5 Plataformas Computacionais de Alto Desempenho	64
5.1 Introdução	64
5.2 Computação de alto desempenho	64
5.3 Conceitos de CAD	67
5.4 Arquitetura de computadores de alto desempenho	69
5.5 Os clusters utilizados	73
5.5.1 O Cluster do L2EP.....	73
5.5.2 O Cluster do GRUCAD	74
5.5.3 O Cluster do LABPLAN	74
5.5.4 Comparações de desempenho de rede	74
5.6 Programação paralela	76
5.7 A Biblioteca MPI	77
5.8 O Framework de teste	77
5.8.1 Estrutura de diretórios	79
5.8.2 Arquivos com os dados das malhas	81
5.8.3 Programa XPLL	81

5.8.4 Script de compilação	84
5.8.5 A geração de matrizes sintéticas	85
5.9 Núcleos de paralelização	86
5.10 Conclusão	91
6 Resultados	92
6.1 Introdução	92
6.2 Os casos testes	93
6.3 Esquemas de armazenamento de matrizes esparsas	94
6.3.1 O Método de Armazenamento Compactado por Linha (Compressed Row Storage)	97
6.3.2 O Método de Armazenamento Compactado por Coluna (Compressed Column Storage)	98
6.3.3 O Método Armazenamento Compactado por Linha Incremental (Incremental Compressed Row Storage)	99
6.3.4 O Método de Armazenamento Compactado por Linha e Coluna (Row and Column Storage)	100
6.4 Resultados para os métodos de armazenamento	101
6.4.1 Resultados parciais de comunicação para o produto matriz por vetor	102
6.4.1.1 Resultados para a Matriz 347829	103
6.4.1.2 Resultados para a Matriz 347832	106
6.4.1.3 Resultados para a Matriz 1009899	107
6.4.1.4 Resultados para a Matriz 2163200	109
6.4.2 Resultados considerando o método iterativo BiCGStab	111
6.4.2.1 Resultados para a Matriz 347829, cluster do GRUCAD	111
6.4.2.2 Resultados para a Matriz 347832, cluster do GRUCAD	113
6.4.2.3 Resultados para a Matriz 2163200, cluster do GRUCAD	114
6.4.2.4 Resultados para a Matriz 347829, cluster do L2EP	115
6.4.2.5 Resultados para a Matriz 347832, cluster do L2EP	116
6.4.2.6 Conclusões para os métodos de compactação e produtos matriz por vetor	117
6.5 Resultados para o pré-condicionamento FLUI	119
6.5.1 Observação do desacoplamento dos sistemas	121
6.5.2 Observação da variação do número de iterações com o erro de convergência	122
6.5.3 Resultados para o pré-condicionador FLUI	123
6.5.3.1 Resultados para a Matriz 219539	123
6.5.3.2 Resultados para a Matriz 347832	126
6.5.3.3 Resultados para a Matriz 2000	128
6.5.3.4 Observações para o pré-condicionador FLUI	130
6.6 Resultados de desempenho no tempo	131
6.6.1 Resultados para a Matriz 219539	131
6.6.2 Resultados para a Matriz 347832	134
6.6.3 Resultados para a Matriz 1009899	136
6.7 Resultados para outros pré-condicionadores	138
6.7.1 Resultado para o pré-condicionador LU incompleto não paralelizado	138
6.7.2 Resultado para o pré-condicionador de Jacobi	139
6.7.3 Comentários	139
6.8 Conclusão	140
7 Conclusão	142
7.1 Notas introdutórias	142
7.2 A síntese dos resultados	143
7.3 Assertivas	146
7.4 Conclusões finais	149
Anexo 1 - Armazenamento Compactado por Linha do EFCAD (ACL).....	151
Anexo 2 - Decomposição LU incompleta por blocos com interseção com compactação RCS.....	160
Referências Bibliográficas.....	182

LISTA DE FIGURAS

Fig. 2.1. Domínio de estudo em 2D.....	21
Fig. 3.1. Árvore de decisão: escolha do método iterativo em função da matriz A	38
Fig. 3.2. Algoritmo do Método Gradiente Conjugado.....	41
Fig. 3.3. Algoritmo do Método Gradiente Biconjugado.....	43
Fig. 3.4. Algoritmo do Método Gradiente Biconjugado Estabilizado.....	45
Fig. 3.5. Algoritmo do Método Gradiente Conjugado Quadrado.....	47
Fig. 4.1. Elementos a serem calculados no j -ésimo passo de cálculo.....	58
Fig. 4.2. Passagens de mensagens para o t -ésimo passo de cálculo.....	58
Fig. 4.3. Pré-condicionamento proposto.....	59
Fig. 4.4. Obtenção da matriz de pré-condicionamento.....	60
Fig. 4.5. Estrutura final de pré-condicionamento.....	61
Fig. 4.6. Estrutura final de pré-condicionamento considerando o armazenamento esparsa RCS.....	62
Fig. 5.1. Categorização baseada na utilização da memória.....	71
Fig. 5.2. Largura de banda para os três clusters utilizados.....	75
Fig. 5.3. Tabela de saída com a variação da interseção e o número de iterações por máquina.....	84
Fig. 5.4. Balanceamento de carga utilizado.....	87
Fig. 5.5. Multiplicação matriz esparsa por vetor.....	89
Fig. 6.1. Comparações para a Matriz 1 - Tempos úteis.....	105
Fig. 6.2. Comparações para a Matriz 2 - Tempos úteis.....	107
Fig. 6.3. Comparações para a Matriz 3 - Tempos úteis.....	108
Fig. 6.4. Comparações para a Matriz 4 - Tempos úteis.....	110
Fig. 6.5. Comparações para a Matriz 347829 - BiCGStab.....	112
Fig. 6.6. Comparações para a Matriz 347832 - BiCGStab.....	113
Fig. 6.7. Comparações para a Matriz 2163200 - BiCGStab.....	114
Fig. 6.8. Comparações para a Matriz 347829 - BiCGStab – L2EP.....	115
Fig. 6.9. Comparações para a Matriz 347832 - BiCGStab – L2EP.....	116
Fig. 6.10. Comparações GRUCAD x L2EP, Matriz 347829, BiCGStab.....	117
Fig. 6.11. Comparações GRUCAD x L2EP, Matriz 347832, BiCGStab.....	118
Fig. 6.12. Matriz 6, 1080 variáveis, pré-condicionador FLUI e desacoplamento.....	121
Fig. 6.13. Variação do número de iterações em função do erro de convergência. Matriz 7, BiCG, pré-condicionador FLUI: (a) erro igual a 1×10^{-4} , (b) erro igual a 1×10^{-5} e (c) erro igual a 1×10^{-6}	122
Fig. 6.14. Variação do número de iterações para a Matriz 219539, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.....	124
Fig. 6.15. Variação do número de iterações para a Matriz 347832, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.....	126
Fig. 6.16. Variação do número de iterações para a Matriz 2000, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG, (c) BiCGStab e (d) CGS.....	129

Fig. 6.17. Variação do tempo de processamento para a Matriz 219539, pré-condicionador FLUI com erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.....	133
Fig. 6.18. Variação do tempo de processamento para a Matriz 347832, pré-condicionador FLUI com erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.....	135
Fig. 6.19. Variação do tempo de processamento para a Matriz 1009899, pré-condicionador FLUI com erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.....	137
Fig. 7.1. Síntese dos resultados obtidos: BiCGStab, FLUI, ICRS e Myrinet numa única plataforma de cálculo.	145
Fig. A1.1. Localização de elementos no vetor compactado.....	155
Fig. A2.1. Divisão e linha de interseção.....	161
Fig. A2.2. Matriz de decomposição e matriz de multiplicação.....	162
Fig. A2.3. Eliminação das linhas e colunas e matrizes L e U de pré-condicionamento.....	162
Fig. A2.4. Algoritmo para divisão do número de linhas.....	163
Fig. A2.5. Divisão da matriz em dois <i>hosts</i>	164
Fig. A2.6. Divisão em três partes e vetores associados.....	165
Fig. A2.7. Porções destinadas a cada processador e vetores associados.....	166
Fig. A2.8. Vetores de armazenamento depois do corte.....	166
Fig. A2.9. Vetores após o corte.....	167
Fig. A2.10. Matriz original e matrizes desmembradas.....	167
Fig. A2.11. Chamada à decomposição.....	168
Fig. A2.12. Matriz ordem 5 e vetor IADRES associado.....	169
Fig. A2.13. Algoritmo para filtragem dos elementos dentro da região válida.....	170
Fig. A2.14. Algoritmo para atualização do vetor NCED	170
Fig. A2.15. Algoritmo para eliminação das linhas em excesso da decomposição.....	171
Fig. A2.16. Vetor NCED resultante.....	172
Fig. A2.17. Algoritmo para eliminação das linhas em excesso para multiplicação.....	173
Fig. A2.18. Eliminação de linhas: para as matrizes fatoradas (em cima) e para a multiplicação (em baixo)....	174
Fig. A2.19. Matrizes parciais e vetores associados.....	175
Fig. A2.20. Vetor NCED Global.....	176
Fig. A2.21. Algoritmo para correção do NCED	176
Fig. A2.22. Algoritmo para correção do IADRES	177
Fig. A2.23a-d – (a) Decomposição L; (b) Decomposição U; (c) Decomposição L desacoplada com 2 hosts; (d) Decomposição U desacoplada com 2 hosts.....	178
Fig. A2.24a-f – Matrizes fatorizadas tipo L: (a) integral, (b) submatrizes, (c) 100% de interseção, (d) 70% de interseção, (e) 34% de interseção, (f) sem interseção.....	179
Fig. A2.25a-f – Matrizes fatorizadas tipo U: (a) integral, (b) submatrizes, (c) 100% de interseção, (d) 70% de interseção, (e) 34% de interseção, (f) sem interseção.....	180

LISTA DE TABELAS

TABELA 6.1. MATRIZES UTILIZADAS.....	94
TABELA 6.2. DISTRIBUIÇÃO DOS VETORES DO MÉTODO CRS EM QUATRO PROCESSADORES.....	98
TABELA 6.3. DISTRIBUIÇÃO DOS VETORES DO MÉTODO CCS EM QUATRO PROCESSADORES.....	99
TABELA 6.4. DISTRIBUIÇÃO DOS VETORES DO MÉTODO ICRS EM QUATRO PROCESSADORES.....	100
TABELA 6.5. DISTRIBUIÇÃO DOS VETORES DO MÉTODO RCS EM QUATRO PROCESSADORES.....	101
TABELA 6.6. DESEMPENHOS PARA A MATRIZ 1, MÉTODO CRS.....	104
TABELA 6.7. DESEMPENHOS PARA A MATRIZ 1, MÉTODO ICRS.....	104
TABELA 6.8. DESEMPENHOS PARA A MATRIZ 1, MÉTODO RCS.....	104
TABELA 6.9. DESEMPENHOS PARA A MATRIZ 1, MÉTODO CCS.....	104
TABELA 6.10. COMPARAÇÕES PARA A MATRIZ 1 - TEMPOS ÚTEIS.....	104
TABELA 6.11. DESEMPENHOS PARA A MATRIZ 2, MÉTODO CRS.....	106
TABELA 6.12. DESEMPENHOS PARA A MATRIZ 2, MÉTODO ICRS.....	106
TABELA 6.13. DESEMPENHOS PARA A MATRIZ 2, MÉTODO RCS.....	106
TABELA 6.14. DESEMPENHOS PARA A MATRIZ 2, MÉTODO CCS.....	106
TABELA 6.15. COMPARAÇÕES PARA A MATRIZ 2 - TEMPOS ÚTEIS.....	107
TABELA 6.16 DESEMPENHOS PARA A MATRIZ 3, MÉTODO CRS.....	107
TABELA 6.17. DESEMPENHOS PARA A MATRIZ 3, MÉTODO ICRS.....	107
TABELA 6.18. DESEMPENHOS PARA A MATRIZ 3, MÉTODO RCS.....	108
TABELA 6.19. DESEMPENHOS PARA A MATRIZ 3, MÉTODO CCS.....	108
TABELA 6.20. COMPARAÇÕES PARA A MATRIZ 3 - TEMPOS ÚTEIS.....	108
TABELA 6.21. DESEMPENHOS PARA A MATRIZ 4, MÉTODO CRS.....	109
TABELA 6.22. DESEMPENHOS PARA A MATRIZ 4, MÉTODO ICRS.....	109
TABELA 6.23. DESEMPENHOS PARA A MATRIZ 4, MÉTODO RCS.....	109
TABELA 6.24. DESEMPENHOS PARA A MATRIZ 4, MÉTODO CCS.....	109
TABELA 6.25 COMPARAÇÕES PARA A MATRIZ 4, TEMPOS ÚTEIS.....	110
TABELA 6.26 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 347829 – GRUCAD.....	111
TABELA 6.27 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 347832 – GRUCAD.....	113
TABELA 6.28 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 2163200 – GRUCAD.....	114
TABELA 6.29 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 347829 – L2EP.....	115
TABELA 6.30 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 347832 – L2EP.....	116
TABELA 6.31. NÚMERO DE ITERAÇÕES PARA O SISTEMA 219539.....	124
TABELA 6.32. NÚMERO DE ITERAÇÕES PARA O SISTEMA 347832.....	126
TABELA 6.33. NÚMERO DE ITERAÇÕES PARA O SISTEMA 2000 CG E BiCG.....	128
TABELA 6.34. NÚMERO DE ITERAÇÕES PARA O SISTEMA 2000 BiCGSTAB E CGS.....	129
TABELA 6.35. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 219539.....	132
TABELA 6.36. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 347832.....	134
TABELA 6.37. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 1009899.....	136

TABELA 6.38. TEMPOS PARA O SISTEMA 1009899 – PRÉ-CONDICIONAMENTO LU INCOMPLETO.....	138
TABELA 6.39. TEMPOS PARA O SISTEMA 347832 – PRÉ-CONDICIONAMENTO LU INCOMPLETO.....	138
TABELA 6.40. TEMPOS PARA O SISTEMA 1009899 – PRÉ-CONDICIONAMENTO DE JACOBI.....	139
TABELA 6.41. TEMPOS PARA O SISTEMA 347832 – PRÉ-CONDICIONAMENTO DE JACOBI.....	139

LISTA DE SIGLAS

ACL – Armazenamento Compactado em Linha
BCRS – Block Compressed Row Storage
BiCG – Biconjugate Gradient
Bi-CGStab – Biconjugate Gradient Stabilized
CAD – Computação de Alto Desempenho
CCS – Compressed Column Storage
CDS – Compressed Diagonal Storage
CG – Conjugate Gradient
CGNE – Conjugate Gradient on the Normal Equations for AA^T
CGNR – Conjugate Gradient on the Normal Equations for $A^T A$
CGS – Conjugate Gradient Squared
CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico
CRS – Compressed Row Storage
EFCAD – Electromagnetic Field Computer Aided Design
FAPESC – Fundação de Apoio à Pesquisa Científica e Tecnológica do Estado de Santa Catarina
Fast Ethernet – Tecnologia de redes de computadores de tráfego de 100 Mbits/s
FEECAD – Finite Edge Elements 3D Calculation of Eletromagnetic Fields
FLUI - Fatoração LU Incompleta por Blocos com Interseção
Gigabit Ethernet – Tecnologia de redes de computadores de tráfego de 1Gbit/s
GMERR – Generalized Minimal Error
GMRES – Generalized Minimum Residual
GRUCAD – Grupo de Concepção e Análise de Dispositivos Eletromagnéticos
ICCG - Incomplete Choleski Conjugate Gradient
ICRS – Incremental Compressed Row Storage
L2EP – Laboratoire d' Electrotechnique et d' Electronique de Puissance
LABPLAN - Laboratório de Planejamento de Sistemas de Energia Elétrica
MILU – Modified Incomplete LU
MIMD – Multiple Instruction Multiple Data
MINRES – Minimal Residual
MISD – Multiple Instruction Multiple Data
MPI – Message Passing Interface
MPICH – Distribuição MPI do Argonne National Laboratory
MPP – Massively Parallel Processing
Myrinet – Tecnologia de redes de computadores de tráfego de 2Gbits/s
NetPIPE – Network Protocol Independent Performance Evaluator
NUMA – Non Uniform Memory Access

PVP – Parallel Vector Processing
RCS – Row and Column Storage
RISC – Reduced Instruction Set Commands
QMR – Quasi-Minimal Residual
SIMD – Single Instruction Multiple Data
SISD – Single Instruction Single Data
SOR – Sucessive Over-Relaxation
SPMD – Single Program Multiple Data
SYMMLQ – Symmetric LQ Decomposition
SMP – Symmetric Multi-Processing
UFSC – Universidade Federal de Santa Catarina

LISTA DE SÍMBOLOS

- μ - permeabilidade magnética (H/m);
- $\overset{1}{A}$ - potencial vetor magnético (Wb/m);
- σ - condutividade elétrica (s/m);
- V - potencial escalar magnético (V);
- $\overset{1}{B}_r$ - indução magnética remanente, a qual é acrescentada para tratar ímãs permanentes (T);
- $\overset{1}{J}_s$ - densidade superficial de corrente de condução (A/m²);
- N_{CO} - número de condutores finos;
- S_f - seção transversal da bobina composta por condutores finos (m²);
- I - a corrente na bobina (A);
- U - tensão na bobina (V);
- R - resistência da bobina (ohm);
- L - profundidade da estrutura (m);
- L_f é a indutância adicional do circuito elétrico (H);
- S – área do domínio de estudo (m²).
- A – matriz do potencial vetor nos nós da malha, matriz de coeficientes;
- SS – matriz de permeabilidade;
- N – matriz de condutividade;
- P' – matriz que relaciona a tensão no elemento aos nós do elemento;
- P – matriz que relaciona a corrente no elemento aos nós do elemento;
- D – vetor de excitações devido aos ímãs permanentes;
- Q' – matriz de enlace de fluxo;
- R' – matriz diagonal contendo as resistências à corrente contínua dos condutores maciços;
- Q – matriz de enlace de fluxo nos enrolamentos;
- R – matriz das resistências à corrente contínua dos enrolamentos;
- L – matriz das indutâncias das cabeças de bobinas;
- I_f – matriz das correntes relativas aos condutores finos;
- U_f – matriz das tensões relativas aos condutores finos;
- I_m – matriz das correntes relativas aos condutores maciços;
- U_m – matriz das tensões relativas aos condutores maciços;
- M - matriz de pré-condicionamento;

\mathbf{b} - vetor do lado direito;
 $\hat{\mathbf{x}}$ - solução exata;
 $\tilde{\mathbf{x}}$ - aproximação da solução;
 \mathbf{r} - vetor resíduo;
 \mathbf{e} - vetor erro;
 $P_i(\mathbf{A})$ - polinômio de grau i em função da matriz \mathbf{A} ;
 K^i - subespaço de Krylov;
 \mathbf{p}_i - vetor com as direções de procura da iteração atual;
 \mathbf{p}_{i-1} - vetor com as direções de procura da iteração anterior;
 \mathbf{q}_i - vetor de direção $\mathbf{A}\mathbf{p}$ -conjugado;
 \mathbf{z}_0 - vetor;
 \mathbf{z}_{i-1} - vetores;
 α_i - escalares: passo de procura;
 β_i - escalar: correção da direção de procura;
 ρ_{i-1} - escalar auxiliar;
 ρ_{i-2} - escalar auxiliar;
 $\tilde{\mathbf{p}}_i^0$ - iterações i e $i-1$, direção de procura conjugado;
 $\tilde{\mathbf{r}}_i^0$ - iterações i e $i-1$, resíduo conjugado;
 $\tilde{\mathbf{z}}_i^0$ - iterações i e $i-1$, resíduo conjugado \mathbf{M} -ortogonal;
 \mathbf{s} - vetor de resíduo atualizado;
 \mathbf{t} - o vetor de resíduo \mathbf{A} conjugado;
 ω_i - parâmetro escalar;
 ω_{i-1} - parâmetro escalar;
 \mathbf{u}_i - vetor de direção auxiliar;
 $\tilde{\mathbf{q}}_i^0$ - vetor de direção $\mathbf{A}\tilde{\mathbf{p}}^0$ -conjugado;
 \mathbf{B} - matriz auxiliar;
 \mathbf{D} - matriz diagonal;
 \mathbf{I} - matriz identidade;
 \mathbf{L} - matriz triangular inferior;
 \mathbf{U} - matriz triangular superior;
 \mathbf{v} - vetor, resultado de $\mathbf{U}^T \mathbf{1}$;

l - vetor;

s - escalar em $s = l^T \mathbf{u} + t$;

t - escalar em $t = s - l^T \mathbf{u}$;

\mathbf{u} - vetor em $\mathbf{r} = \mathbf{L}\mathbf{u}$

nnz - número de elementos não nulos;

$nrow$ - número de linhas da matriz;

$nhost$ - número de processadores, máquinas utilizadas;

ROW_PTR - vetor, armazena os inícios de linhas no método CRS;

COL_IND - vetor, armazena as posições de colunas no método CRS;

VAL - vetor, armazena os valores da matriz no método CRS e CCS;

COL_PTR - vetor, armazena os inícios de coluna no método CCS;

ROW_IND - vetor, armazena as posições das linhas no método CCS;

INC - vetor, armazena incrementalmente os índices de colunas dos elementos no método ICRS;

IVAL - vetor, armazena os elementos não nulos no método ICRS;

NCED - armazena os inícios de linhas/colunas no método RCS;

IADRES - armazena a posição das colunas/linhas no método RCS;

SUP - armazena os elementos da matriz triangular superior no método RCS;

INF - armazena os elementos da matriz triangular inferior no método RCS;

1 Introdução

1.1 Preliminares: a necessidade de Computação de Alto Desempenho em setores atuais de pesquisa

A preocupação com a necessidade de grande capacidade de processamento remonta a meados da década de 60, quando da apresentação do primeiro supercomputador (CDC6000). Desde então, tem-se experimentado uma grande evolução na capacidade de computação, tanto que o termo supercomputador passou a ser substituído gradualmente por Plataforma de Computação de Alto Desempenho.

Até o final da década de 80 os supercomputadores eram facilmente classificados, pois existiam apenas máquinas com poucos recursos destinadas ainda a poucos usuários e grandes dispositivos de cálculo possuindo normalmente poucos processadores, mas com capacidades vetoriais, deixando um vazio entre ambos. A partir da década de 90, os sistemas se tornaram mais escalonáveis, com máquinas possuindo de 1 até 1000 processadores, o que tornou a distinção entre supercomputador e computador “normal” uma questão mais difícil, e levou ao termo Computação de Alto Desempenho. Este tem sido amplamente utilizado para caracterizar o uso de recursos computacionais que permitam obter um desempenho maior que o obtido usando PC's, estações de trabalho ou pequenos servidores.

Esta definição engloba não apenas computadores, mas também a tecnologia de rede, os algoritmos e os ambientes necessários para permitir a utilização destes sistemas, que podem variar desde clusters de PC's a grandes supercomputadores vetoriais. Estas plataformas de Alto Desempenho podem possuir sistemas com memória compartilhada, distribuída, combinação de vários clusters e, invariavelmente, utilizam computação paralela. Isto porque através da computação paralela os limites de uma única CPU podem ser vencidos. Isto quer dizer que a computação paralela

permite, ao menos em tese, que possam ser resolvidos problemas que não se comportam em uma única CPU e que não podem ser resolvidos num tempo razoável. Isto se aplica, por exemplo, aos casos limites atuais como a previsão global do tempo e o seqüenciamento do genoma humano.

A engenharia elétrica também tem feito uso desta evolução dos recursos informáticos na solução de problemas de engenharia cada vez mais complexos. Na área de dispositivos eletromagnéticos dispomos atualmente de modelos físico-matemáticos e sistemas de cálculo em computadores bem estabelecidos com os quais é possível estudar, projetar e aperfeiçoar um grande número destes produtos.

A evolução destes dispositivos, motivada por fatores diversos como diminuição do consumo de energia, de custos de produção, melhoria de materiais construtivos e competitividade entre fabricantes na disputa de novos consumidores, vem acompanhada de uma necessidade crescente de refinamento dos modelos e, conseqüentemente, uma evolução das máquinas capazes de avaliar estes novos modelos num tempo cada vez menor.

Os modelos dos dispositivos eletromagnéticos exigem o estudo e o cálculo criterioso de campos elétricos e/ou magnéticos, realizados numericamente usando técnicas como a de elementos finitos. Esta técnica é mais adaptada à geometria geralmente complexa das máquinas elétricas. Ela consiste na discretização do domínio em um determinado número de regiões elementares, os elementos finitos, possuindo nós e arestas, onde a coleção de nós é conhecida como malha de elementos finitos. A solução do sistema pode ser obtida analisando-se os nós ou as arestas, dando origem a dois modos diferentes de obtenção dos resultados. Analisando-se através dos nós dos elementos, dentro de cada elemento, a equação diferencial que rege o fenômeno é aproximada como uma combinação, geralmente linear ou quadrática, das variáveis definidas em cada nó. A utilização deste método permitiu a construção de ferramentas de cálculo computacionais aplicadas a dispositivos eletromagnéticos que vem se valendo da evolução do hardware para descrever, cada vez com mais precisão, o funcionamento destes dispositivos.

A técnica de elementos finitos é muito versátil e da sua aplicação ao domínio de estudo resulta um sistema matricial cuja dimensão pode, em alguns casos, representar um impedimento na análise do problema. Se o problema sendo estudado não for linear,

deve-se realizar um número de iterações até que a solução encontre um valor razoavelmente correto dentro de um limite estabelecido. Esta procura pela solução pode representar mais uma outra dificuldade a ser vencida. A evolução da análise considerando o movimento [1], o acoplamento de equações do circuito elétrico de acionamento [2], a análise tridimensional com inclinação de partes da estrutura sob estudo [3] e outras variações, elevou a necessidade de processamento.

Assim, dependendo da necessidade do modelo a ser estudado onde as simplificações não são permitidas, teremos alguns casos limites onde os recursos informáticos tradicionais se mostram extremamente limitados. Os limites podem ser físicos, como o tamanho da memória do computador, quando a matriz do sistema tem um tamanho tal que representa uma barreira para uma única máquina. O limite também pode ser o tempo de cálculo ou simulação, que dependendo do problema, da precisão, do número de ciclos de funcionamento desejados, pode consumir dias.

Desde o princípio de utilização destes métodos sempre foram usadas formas de implementação que tentaram contornar as limitações existentes nos sistemas informáticos. Isto porque à medida que os avanços foram chegando à computação, casos mais complexos passaram a ser estudados como um legítimo sinal da evolução científica e tecnológica. Entre as formas de implementação estão aquelas que procuram otimizar o armazenamento de matrizes, técnicas conhecidas como formas de compactação de matrizes. As matrizes de elementos finitos têm como características a simetria, a esparsidade e a aglomeração dos dados na forma de banda. A economia de memória proporcionada pelos métodos de compactação que consideram estes fatores é essencial no desenvolvimento desta classe de programas.

Outra forma de implementação que procura contornar as restrições é a modelagem do dispositivo real através de um modelo equivalente simplificado ao invés da utilização de um modelo próximo da estrutura verdadeira. Isto porque um modelo mais fiel pode significar um domínio de estudo grande e conseqüentemente matrizes maiores, mais memória e maior tempo de cálculo.

A solução dos sistemas lineares gerados pelo método de elementos finitos pode ser feita por métodos diretos ou por métodos iterativos. Tradicionalmente utilizam-se os métodos diretos (Eliminação de Gauss com pivotação parcial ou Fatoração LU) numa versão por banda na solução destes sistemas, necessitando prever os espaços de

enchimento no armazenamento da matriz. Isto porque estes métodos são reconhecidamente eficazes e nos levam à solução exata, se ela existir.

Em contraste, algumas propriedades de esparsidade das matrizes podem ser usadas com vantagem em multiplicações matriz-vetor considerando o número de operações, o menor espaço necessário para o armazenamento de elementos e eliminando o enchimento. Isto nos leva a perguntar se um sistema de equações pode ser resolvido (ou uma solução razoavelmente boa pode ser encontrada) usando multiplicações matriz por vetor. Se isto pode ser executado com um número moderado de multiplicações e um pequeno trabalho adicional, então um procedimento iterativo pode se colocar como alternativa à Eliminação de Gauss.

O termo “método iterativo” se refere a uma variedade de técnicas que usam aproximações sucessivas para obter soluções mais precisas para sistemas lineares a cada passo. Os métodos iterativos possuem núcleos comuns de processamento que independem do método, mas que consomem valioso tempo de processamento. Estes núcleos de consumo de tempo são os produtos internos, as atualizações de vetores, os produtos matriz por vetor e a solução de pré-condicionadores. Entretanto, alguns destes núcleos podem ser facilmente paralelizados tanto em arquiteturas com memória distribuída ou memória compartilhada. Ressalta-se aqui que a expressão facilmente paralelizados não implica melhoria de desempenho.

Isto posto, o texto que se segue descreve um trabalho envolvendo o uso de métodos iterativos modernos pré-condicionados, utilizando computação paralela em máquinas do tipo cluster, aplicada ao método de elementos finitos cujas matrizes são armazenadas através de técnicas de compactação de matrizes esparsas.

1.2 O estado da arte dos métodos numéricos iterativos, estudos de pré-condicionadores e aplicação de elementos finitos.

Dispositivos eletromagnéticos como máquinas elétricas têm geometrias diversas e complexas, tal que o método de discretização do domínio precisa ser flexível e

suficientemente preciso para contemplar as necessidades. CSENDES e SILVESTER [4] em 1970 mostraram as vantagens do método de elementos finitos na análise dos campos e dispersões em guias de onda. Recentemente BASTOS e SADOWSKI [5] elaboraram uma obra completa sobre a aplicação de elementos finitos na análise destes aparatos industriais.

Entretanto, a análise também carece de métodos de solução de sistemas de equações que podem ser rudemente divididos entre os métodos diretos e iterativos. A paralelização dos métodos de resolução de sistemas lineares usada neste trabalho foca operações em métodos iterativos, já que os métodos diretos apresentam elevada necessidade de comunicação [6].

A história dos métodos iterativos é tão antiga quanto a dos métodos diretos e tem em Gauss um de seus precursores. Como encontrado em [7], o próprio Gauss, em 1823, reconheceu que seu método poderia não ser atrativo quando tratando sistemas com número elevado de variáveis. Naquele ano ele propôs um método iterativo para a solução de quatro equações com quatro variáveis. As matrizes por ele estudadas possuíam elevada dominância diagonal e ele percebeu que a principal contribuição do lado direito vinha daqueles elementos diagonais. Isto levou ao método de Gauss-Jacobi, utilizado pelo astrônomo e matemático Jacobi no cálculo das perturbações nas órbitas dos planetas do sistema solar. Uma outra observação do comportamento das matrizes levou Gauss a um segundo método, conhecido como Gauss-Seidel.

Entretanto, das próprias observações de Gauss, a convergência destes métodos é diretamente proporcional à dominância diagonal dos elementos, o que não acontece em muitos dos fenômenos estudados. Neste caso a convergência de métodos como Gauss-Seidel é muito lenta e impraticável, razão pela qual outros métodos mais rápidos começaram a serem estudados, buscando sempre a evolução. Surgiram então os métodos baseados em minimização.

Já no século XX o método das Direções Conjugadas foi provavelmente apresentado primeiro por E. SCHMIDT [8] em 1908, e foi reinventado independentemente por FOX, HUSKEY e WILKINSON [9] em 1948. No início dos anos cinquenta, o método dos Gradientes Conjugados foi proposto independentemente por Magnus R. HESTENES e Eduard STIEFEL e publicado em conjunto em 1952 [10]. O método dos Gradientes Conjugados ficou esquecido por quase duas décadas

quando foi popularizado como método iterativo para matrizes grandes e esparsas por John K. REID [11] em 1971. Logo em seguida, em 1977, J. A. Meijerink e Henk A. van der Vorst [12] apresentaram um trabalho utilizando métodos iterativos aplicados a sistemas de equações onde a matriz de coeficientes é uma M -matriz¹ simétrica. Entretanto, este trabalho permitiu que os métodos iterativos pudessem ser utilizados por vários pesquisadores no mundo, visto que demonstrou que a decomposição incompleta de LL^T poderia ser combinada com o método dos Gradientes Conjugados acelerando a solução, e sua decomposição é tão estável quanto o processo completo de Choleski. Este trabalho deu origem ao termo ICCG (*Incomplete Choleski Conjugate Gradient* – Gradiente Conjugado com decomposição Incompleta de Choleski).

Os estudos sobre o método do Gradiente Conjugado, na tentativa de melhorar seu desempenho e limitações, levaram a métodos como o Gradiente Biconjugado, proposto por FLETCHER em 1976 [13], o método dos Resíduos Mínimos Generalizados, proposto em 1986 por SAAD e SCHULTZ [14], o Gradiente Conjugado Quadrado, proposto em 1989 por SONNEVELD [15] e o método do Gradiente Biconjugado Estabilizado, proposto por VORST em 1992 [16], entre outros.

Os métodos iterativos são normalmente usados em conjunto com pré-condicionadores (que são aproximações da matriz inversa do sistema a ser resolvido) em sistemas esparsos. Os métodos iterativos também podem representar a única forma de solução para alguns sistemas realmente grandes. O campo de estudo de pré-condicionadores é o mais amplo e, atualmente, aquele que concentra o maior esforço de pesquisa [17]. Isto porque a utilização de um pré-condicionador pode significar um conhecimento profundo sobre o problema do qual advém o sistema linear. Entre os pré-condicionadores aplicados a métodos como o Gradiente Conjugado e seus derivados estão aqueles denominados de classe geral, formados pelos próprios métodos de Gauss-Jacobi, Gauss-Seidel, Fatorizações Incompletas de Cholesky e outros. Existem também outras duas categorias de pré-condicionadores: aqueles projetados para matrizes específicas e os mais atuais conhecidos como pré-condicionadores Multigrid e de decomposição do domínio.

As idéias dos Métodos Multigrid são baseadas em ações sobre malhas dos

¹ Uma matriz $A = (a_{ij})$ é uma M -matriz se $a_{ij} \leq 0$ para $i \neq j$, A é não singular e $A^{-1} \geq 0$.

domínios e em propriedades dos métodos iterativos lineares como convergência rápida nas primeiras iterações e uma convergência mais lenta na aproximação do resultado. O método atua sobre diferentes malhas e precisa da elaboração de operadores de interpolação e restrição dependentes das distâncias entre os pontos do *grid*. Os métodos Multigrid têm sido muito utilizados em problemas de incompressibilidade de fluidos. As idéias destes métodos também têm sido aplicadas como pré-condicionadores aos métodos dos subespaços de Krylov [18].

A Decomposição do Domínio trata da decomposição espacial do objeto de cálculo em muitos subdomínios. O problema então é dividido em pequenos problemas de forma tal que cada um possa ser solucionado em um único nó de um sistema multiprocessado. Esta técnica é freqüentemente utilizada na resolução de grandes sistemas de equações gerados pelos métodos de elementos finitos ou volumes finitos em supercomputadores de memória distribuída e em clusters de computadores.

Tanto a técnica Multigrid como a técnica de Decomposição do Domínio não são tratadas neste texto, pois ambos os métodos atuam sobre a malha que forma o sistema. Este trabalho se concentra sobre o sistema matricial final condensado pelos softwares de elementos finitos e, apresenta uma variação de um pré-condicionador da classe geral, mais especificamente um pré-condicionador *LU* por blocos incompleto, adaptado à arquitetura de memória distribuída própria dos clusters de computadores.

A preocupação com a melhoria do desempenho do processamento na resolução dos sistemas provenientes da análise de elementos finitos acompanhou a própria evolução da técnica. No artigo de Alvin WEXLER [19] de 1961, vários métodos usados na solução de fenômenos do eletromagnetismo são revistos, incluindo a solução de sistemas de equações lineares por meios iterativos e uma breve descrição do método de elementos finitos. Em 1967 TINNEY, WALKER e OGBUOBIRI apresentaram trabalhos com a utilização da esparsidade na solução direta através de decomposição e eliminação de Gauss em estudos de redes de sistemas de potência [20,21]. Os engenheiros eletricitistas foram os primeiros a utilizar a idéia da esparsidade no armazenamento de matrizes. Em 1974, KINSNER e DELLA TORRE [22] apresentaram uma abordagem iterativa na solução de diversas formulações de elementos finitos, estudando também métodos para a aceleração da convergência para as seqüências de vetores. Outras evoluções ocorreram no sentido de aumentar a rapidez

de análise numa simplificação do domínio. Em 1974 CHARI [23] aplica o método de elementos finitos no cálculo magnetostático de um gerador de corrente contínua, utilizando a condição de periodicidade como condição de contorno do problema, diminuindo o domínio de estudo e viabilizando a análise.

Na metade da década de 70 e início dos anos 80, com o advento de supercomputadores, iniciaram-se os estudos aplicando computação paralela e vetorial à análise de dispositivos eletromagnéticos. BORDING [24] em 1981 aplicou o processamento paralelo ao método de elementos finitos atuando sobre a eliminação de Gauss. Em 1988, IDA [25-27] et al apresentam trabalhos envolvendo o método de elementos finitos em Computadores Massivamente Paralelos (MPP), além de diversas outras publicações como [28] utilizando *transputers*, MAGNIN [29] et al utilizando memória compartilhada e outras aplicações na área de antenas. Entretanto, aparentemente o acesso a máquinas paralelas e vetoriais de grande porte e a falta de bibliotecas para programação paralela limitou o número de trabalhos na análise de dispositivos magnéticos, com publicações envolvendo elementos finitos em segmentos como a já mencionada área de transmissão de sinais.

Mais recentemente, o advento de arquiteturas paralelas de baixo custo, como *clusters* de computadores, motivou vários centros de pesquisa [30] a investir esforços e estudos nestes tipos de máquinas. Em 2001 MATSUO e SHIMASAKI [31] desenvolveram trabalhos com decomposição do domínio envolvendo eletromagnetismo e dinâmica de gases solucionados por um esquema envolvendo um pré-condicionador *MILU* (*Modified Incomplete LU*). Eles utilizaram como plataformas de alto desempenho um computador HITACHI SR2201 pseudo-vetorial com processadores RISC e um cluster de 16 PC's Intel Pentium III-1GHz conectados por Fast Ethernet. No ano seguinte, 2002, IWASHITA [32] et al escreveram um artigo descrevendo a utilização de um pré-condicionador *LU* incompleto com processamento paralelo, utilizando um método de ordenamento algébrico multi-colorido nas linhas da matriz. A ordenação multi-colorida é uma técnica conhecida no campo da paralelização, entretanto este trabalho e os anteriores aplicam este método na análise de problemas envolvendo diferenças finitas. A análise paralelizada deste trabalho foi desenvolvida num computador FUJITSU GP-7000F modelo 900, um computador paralelo de

memória compartilhada, com biblioteca OpenMP² [33]. A partir de então são encontrados vários artigos de Iwashita e Shimasaki envolvendo renumeração multicolorida e Multigrid aplicados ao eletromagnetismo, envolvendo elementos nodais e de arestas.

Em 2004 KNIGHT [34] apresentou resultados de simulação de motores de indução usando o método TLM para modelar propriedades não lineares dos materiais em conjunto com a técnica de elementos finitos, utilizando um *cluster* de computadores com um total de 40 processadores 2,1 GHz conectados em uma rede Ethernet de 1 Gigabit/s, usando a biblioteca de comunicação MPI [35]. Também em 2004, COSTA [36] et al apresentaram um trabalho de realização do cálculo parametrizado de dispositivos eletromagnéticos analisados pelo método dos elementos finitos, utilizando técnicas de processamento paralelo em cluster de computadores, visando uma redução no tempo de cálculo.

Em 2005 KANAYAMA e SUGIMOTO [37] aplicaram o método da Decomposição do Domínio a problemas 3D com correntes induzidas utilizando o potencial vetor magnético A usando como plataforma de cálculo seis computadores Intel Pentium 4 – 3GHz, usando como pré-condicionador a decomposição incompleta de Cholesky . Já em 2006 SESHIMA [38] et al apresentaram um artigo onde a paralelização do código de elementos finitos hexaédricos utilizados na análise eletromagnética foi realizada utilizando *multi-threads*³ e OpenMP em dois sistemas computacionais. O primeiro um Intel Pentium D 830 (*Dual Processor*) 3 GHz com sistema operacional Fedora Core / Linux e o segundo sistema formado por dois IBM PowerPC 970MPs (*Dual Processor*) 2,5 GHz com sistema operacional MacOS X. Neste trabalho também foi utilizada a decomposição incompleta de Cholesky como pré-condicionador, entretanto, nem a elaboração nem a aplicação do pré-condicionador foram realizadas em paralelo.

Assim, parte da motivação deste trabalho está no estudo da aplicação de *clusters* de computadores na análise de dispositivos eletromagnéticos contribuindo com o desenvolvimento de programas aplicados a estes dispositivos. Outra motivação está no

² OpenMP é uma API (interface de programação) padronizada para programação multi-thread.

³ *Threads*: são seqüências de execução que compartilham uma determinada área da memória (“um endereço de memória”) e onde uma seqüência pode “ver” a memória da outra. *Multi-thread* é o paralelismo através de múltiplas *threads*.

estudo e conhecimento de métodos iterativos modernos e seus pré-condicionadores, utilizados na solução de sistemas lineares esparsos e de grande porte, formando importante conjunto de ferramentas numéricas utilizadas largamente em outras áreas do conhecimento.

1.3 As propostas da tese e suas contribuições inéditas

O desenvolvimento deste trabalho teve início com o estudo dos algoritmos iterativos dos métodos de Krylov. Neste estudo foram levantadas as características computacionais relativas a estes métodos. Características como os núcleos centrais de consumo de tempo e processamento, operações passíveis de paralelização e necessidades de armazenamento. Inicialmente foram estudados os métodos dos Gradientes Conjugados (CG), Gradiente Biconjugado (BiCG) e Resíduos Mínimos Generalizados com reinicialização (GMRESm), todos sem a aplicação de pré-condicionamento. Estes métodos foram selecionados em função de algumas características que se desejava estudar como existência de simetria (CG), necessidade de verificação de robustez dos métodos (GMRESm) e compatibilidade com métodos clássicos (BiCG) em utilização nos softwares desenvolvidos ao longo dos anos no grupo de pesquisa. Estas experiências mostraram que o GMRESm é, de longe, o método mais robusto com o menor número de iterações comparativamente aos outros métodos estudados. Entretanto é um método exigente no que se refere ao armazenamento, principalmente devido à construção interna de um subespaço de reinicialização. Como resultados destas comparações o trabalho prosseguiu substituindo-se o GMRESm por outros dois métodos de implementação mais simples: o Gradiente Biconjugado Estabilizado (BiCGStab) e o Gradiente Conjugado Quadrado (CGS).

Experiências com paralelização implicam a utilização de máquinas com arquitetura afim. Este trabalho utilizou clusters de computadores, uma arquitetura de memória distribuída, cujos nós são conectados por uma rede e onde a comunicação entre os nós acontece com passagens de mensagens. As primeiras experiências com

paralelização foram conduzidas no Cluster de computadores do Laboratório de Planejamento de Sistemas de Energia Elétrica (LABPLAN) da UFSC. Depois, com a obtenção de financiamento através da Fundação de Apoio à Pesquisa Científica e Tecnológica do Estado de Santa Catarina (FAPESC) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) para a montagem de um cluster próprio, os estudos passaram a ser feitos num cluster Gigabit no Laboratório de Concepção e Análise de Dispositivos Eletromagnéticos (GRUCAD). Além disso, realizaram-se comparações num cluster com rede Myrinet no Laboratoire d'Electrotechnique et d'Electronique de Puissance de Lille (L2EP) da Universidade de Lille, França, comparações estas que validaram o esforço de pesquisa desta tese.

O passo seguinte na evolução do trabalho resultou na criação de um *framework* que permitisse a geração de matrizes teste. Os primeiros casos foram obtidos a partir dos programas 2D que formam o pacote de softwares EFCAD – Eletromagnetic Field Computer Aided Design – desenvolvido no GRUCAD. Contudo, as matrizes obtidas a partir desta abordagem têm pequena dimensão, o que não justifica a utilização de métodos iterativos na solução dos sistemas de equações, uma vez que os métodos diretos, notadamente a Eliminação de Gauss, são extremamente rápidos e eficientes nestas condições. Assim, através de um módulo do FEECAD – Finite Edge Elements 3D Calculation of Eletromagnetic Fields, o programa FeeCSE, para cálculo de casos com potencial escalar com elementos nodais, obtiveram-se matrizes com ordens mais elevadas.

Estas matrizes, bem como as matrizes dos casos 2D são muito esparsas. O armazenamento destas matrizes é feito considerando técnicas especiais já citadas anteriormente. O armazenamento esparsa utilizado nas matrizes 2D, o armazenamento ACL⁴ (Armazenamento Compactado em Linha), porém, não pode ser aplicado às matrizes 3D, pois o mesmo mantém um elevado número de elementos nulos, o que compromete a capacidade de armazenamento do sistema computacional. Entretanto, a literatura [39] descreve vários métodos de armazenamento, o que nos levou a estudar outras formas de compactação de matrizes observando seus desempenhos em algoritmos seqüenciais e paralelos.

⁴ Esta forma de armazenamento está exemplificada no Apêndice 1.

Os métodos iterativos da família do Gradiente Conjugado dependem fortemente do pré-condicionamento empregado. Tanto a elaboração como a aplicação do pré-condicionador nos algoritmos de solução de sistemas de equação são operações normalmente executadas em paralelo. À exceção do pré-condicionamento diagonal, outras formas mais elaboradas de pré-condicionamento exigem a resolução de sistemas de equação ou, pelo menos, a aplicação de substituições progressivas ou regressivas. Estas operações têm bom desempenho quando aplicadas seqüencialmente, mas quando aplicadas de forma integral em ambientes paralelos exigem um grande número de passagens de mensagens. Esta condição é justamente o que não se deseja quando a plataforma de alto desempenho é uma máquina de memória distribuída com altos valores de latência⁵, que é o caso dos clusters de computadores.

Então, para evitar a passagem de mensagens este trabalho utiliza uma nova versão do pré-condicionador LU tradicional, onde a matriz de pré-condicionamento é dividida em diversos blocos, de acordo com o número de processadores, mas podendo possuir uma determinada porcentagem de interseção entre os blocos. Este pré-condicionador é uma evolução do trabalho apresentado por VOLLAIRE [40] em 1998. A pesquisa desenvolvida por ele utilizava o pré-condicionamento LL^T dividido por blocos sem o parâmetro de interseção. Como este pré-condicionamento evita a passagem de mensagens, ele se torna adaptado à arquitetura de memória distribuída dos clusters de computadores.

Até aqui temos, então: quatro métodos iterativos paralelizados nesta pesquisa, a possibilidade de variação da porcentagem de interseção do pré-condicionamento, a possibilidade de variação da tolerância na convergência nos métodos iterativos, a possibilidade de variar o número de processadores nos testes, a possibilidade de testar diversas matrizes diferentes, bem como a possibilidade de realizar diversas vezes o mesmo teste a fim de se obter o menor tempo em um conjunto de tomadas. Para realizar estas comparações de forma eficiente e organizada foi necessária a elaboração de um grande *framework*, automatizado através de scripts e programas auxiliares. Neste *framework* é possível escolher todas as variáveis acima coletando ao final do processamento os dados selecionados. Oportunamente serão tecidas descrições mais

⁵ Latência é o tempo para transferência do pacote de dados. Esta e outras definições relativas às arquiteturas de sistemas de alto desempenho estão descritas no capítulo pertinente.

detalhadas sobre este tópico.

Durante o desenvolvimento destas comparações, observou-se a influência da esparsidade dos sistemas em questão, notadamente devido ao baixo desempenho da rede de comunicação Gigabit, cuja latência está na ordem de dezenas de milissegundos e a largura de banda bem abaixo do valor de 1Gb/s. A fim de se verificar o desempenho dos algoritmos, achou-se necessário testar matrizes mais densas tanto nos clusters do GRUCAD e do L2EP como em produtos matriz por vetor, considerando as formas de armazenamento e algoritmos iterativos. Assim, conseguiu-se observar que a forma de armazenamento, a esparsidade e a latência influenciam na paralelização. São características que não devem ser desprezadas na busca de desempenho em sistemas de computação paralelizados através de clusters.

Depois de vencidas estas etapas, elaborou-se um conjunto de testes a partir dos quais se encontraram algumas diretrizes que podem auxiliar futuros trabalhos envolvendo plataformas de alto desempenho e computação paralela.

Assim, algumas das questões que se propõe responder com este trabalho são:

- Em relação ao armazenamento esparsas das matrizes, a forma de armazenamento esparsa da matriz influencia o desempenho da paralelização? Um algoritmo iterativo, utilizando uma determinada forma de armazenamento esparsa de matrizes, tem desempenho diferente executado na forma seqüencial e na forma paralela? Tem um método de armazenamento mais adequado para a paralelização?
- Em relação às características das matrizes de elementos finitos, o padrão de esparsidade das mesmas influencia o paralelismo em cluster de computadores? A partir de qual ordem de sistema se passa a ter ganho com a paralelização?
- Os clusters de computadores fazem uso de redes de computadores para viabilizar a troca de mensagens entre os nós. Foram realizados estudos num cluster conectado por rede Gigabit e num cluster com rede Myrinet. Qual das tecnologias apresentou melhor desempenho?
- Este trabalho propõe uma nova variação do pré-condicionador LU , uma versão modificada por blocos com um parâmetro que permite a interseção entre os mesmos. Assim, este pré-condicionador tem vantagens em relação ao

anteriormente proposto? A porcentagem de interseção entre os blocos precisa ser grande para permitir uma melhoria? A tolerância (erro) de convergência influencia o comportamento do pré-condicionador?

- Ainda, em relação a utilização de clusters de computadores como plataformas de computação de alto desempenho, foi encontrado ganho de desempenho com a paralelização dos algoritmos de resolução de sistemas de equações lineares? Em função do ganho de desempenho encontrado, vale a pena pagar o custo de instalação de um cluster? Qual a perspectiva de utilização de clusters de computadores? Qual seria o próximo passo numa estratégia de pesquisa envolvendo a paralelização com clusters como plataforma de alto desempenho?
- E, encerrando o conjunto de questões envolvendo a paralelização de algoritmos aplicados à resolução de sistemas de equações em sistemas de elementos finitos, a abordagem utilizada neste trabalho na paralelização é válida? É a única forma de paralelização? Que outras abordagens poderiam ser estudadas numa continuação dos esforços de pesquisa no campo da paralelização pelo grupo de pesquisa?

Assim, dadas estas questões e, embora o estudo da paralelização aplicada à resolução de sistemas de equações seja um campo onde os esforços de pesquisa somam já muitos anos de desenvolvimento, com uma grande bibliografia incluindo livros e artigos dedicados ao assunto, a paralelização envolvendo clusters construídos a partir de computadores e redes de comunicação comerciais carece de publicações. A começar pela própria montagem e instalação do hardware. Isto pôde ser observado diretamente quando da instalação do cluster Gigabit do GRUCAD. Foram várias tentativas em busca de um *kernel* recente do Linux adaptado à realidade da computação paralela com *drives* experimentais para a rede Ethernet Gigabit, configuração de parâmetros internos acessados pela biblioteca de comunicação, instalação de recursos voltados à execução remota de procedimentos e inúmeros outros itens de configuração que não seguem exatamente uma receita de instalação, exigindo pessoal capacitado para sua realização.

Também, embora existam vários pacotes que disponibilizam algoritmos de solução de sistemas de equações, tais facilidades não são de uso geral e não são

facilmente aplicáveis a qualquer plataforma de cálculo já existente. Isto acontece principalmente devido à forma de armazenamento das matrizes esparsas. Uma vez alterada a forma de armazenamento, todas as rotinas que executam operações envolvendo armazenamento esparsa precisam ser reescritas, novamente testadas e validadas. Ocorre que isto torna difícil o reaproveitamento de tais resultados e, no que tange desempenho, torna-se incerto, senão inválido.

Assim, uma vez que foi necessário o estudo e implementação de vários recursos adaptados à realidade da computação paralela em clusters de computadores, podem ser citados os seguintes itens como contribuições inéditas descritas neste texto de tese:

- Comparações de métodos de compactação de matrizes esparsas através de produtos matriz por vetor e métodos iterativos em clusters de computadores. A literatura descreve vários métodos de armazenamento e suas características computacionais. Percebeu-se, entretanto, que estes métodos fazem uso de rotinas de comunicação coletiva que interferem, ao menos no estágio atual da tecnologia empregada, no desempenho dos produtos matriz por vetor. Este comportamento foi registrado e comparado [41].
- O desenvolvimento de um pré-condicionamento LU por blocos com interseção adaptado à arquitetura de memória distribuída dos clusters de computadores. Como será descrito com mais detalhes no capítulo específico sobre pré-condicionadores, este pré-condicionador é uma evolução de um trabalho realizado anteriormente, dedicado também à computação paralela, aplicado a máquinas com memória distribuída. Os resultados relativos à aplicação deste pré-condicionador foram registrados e comparados [42].
- Comparações de desempenho de métodos iterativos não estacionários em matrizes de elementos finitos em clusters de computadores. A demonstração do desempenho de um pré-condicionador é feita aplicando-o a um ou mais métodos iterativos de resolução de sistemas de equações lineares. A demonstração do funcionamento do pré-condicionador proposto neste trabalho foi feita inserindo-o em quatro métodos iterativos [42, 43].
- Demonstração da influência da esparsidade no desempenho dos métodos iterativos em clusters de computadores. Como já comentado, o

armazenamento, a esparsidade e a latência influenciam a paralelização. A esparsidade e a latência estão relacionadas entre si, pois os métodos que utilizam multiplicações matriz por vetor dependem do número de operações realizadas por linha da matriz. Assim, quando a latência é maior, os tempos de comunicação para envio de pequenas quantidades de dados acabam representando uma parcela não desprezível no processamento [41. 44].

- E, ainda que não seja um item inédito considerando-se um panorama mais amplo envolvendo os diversos grupos que realizam pesquisa com paralelização, pode-se citar a montagem do primeiro cluster Gigabit da UFSC. Este cluster foi construído mediante financiamento obtido através de dois editais vencidos pelo GRUCAD, o primeiro um Edital proporcionado pela Fundação de Apoio à Pesquisa Científica e Tecnológica do Estado de Santa Catarina (FAPESC) e, o segundo, um Edital Universal proporcionado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

1.4 Estrutura do texto

Considerando-se as propostas deste trabalho, o texto foi dividido em sete capítulos e numa pequena seqüência de apêndices. Neste primeiro capítulo foi apresentada uma descrição geral do assunto sobre o qual versa esta tese e, em seguida, foram descritos os avanços históricos que precederam este estudo, as propostas e as contribuições inéditas resultantes dos esforços de pesquisa aqui formalizados.

No segundo capítulo é feita uma apresentação sucinta do problema numérico: as equações de Maxwell e as características estruturais das matrizes resultantes da aplicação do método de elementos finitos àquelas equações. Também é apresentado o foco no qual o presente trabalho se fixou.

O terceiro capítulo faz uma breve apresentação de diversos métodos iterativos derivados dos espaços de Krylov, com uma ênfase maior nos métodos propostos para a pesquisa (Gradiente Conjugado, Gradiente Biconjugado, Gradiente Biconjugado Estabilizado e Gradiente Conjugado Quadrado). O capítulo também apresenta algumas

generalidades destes métodos e características computacionais que são exploradas nas suas implementações em paralelo.

O quarto capítulo trata de pré-condicionamento. Faz-se uma explanação sobre o porquê do uso de pré-condicionadores, as classes de pré-condicionadores, uma descrição dos pré-condicionadores da classe geral e uma explanação mais detalhada a respeito do pré-condicionador *LU* por blocos proposto.

O quinto capítulo traz conceitos de Computação de Alto Desempenho e arquitetura de computadores com ênfase em clusters de computadores. São descritos alguns aspectos fundamentais dos clusters utilizados, dos dispositivos de rede Gigabit e Myrinet, de programação paralela e do *framework* computacional desenvolvido, com seus scripts e programas.

O capítulo seis mostra os resultados encontrados nos casos estudados. Também são apresentados os diversos esquemas de armazenamento de matrizes esparsas testados neste trabalho. Estes métodos são descritos através de exemplos, formando mais um conjunto de dados importantes que influenciam o desempenho em paralelo dos algoritmos estudados. São mostradas comparações dos produtos matriz por vetor com métodos de compactação diferentes; comparações entre métodos numéricos pré-condicionados: resultados para o *LU* por blocos com variação da interseção de 0% a 100%, resultados com variação do erro de convergência, resultados de tempo na margem de interseção até 15%; comparações com e sem interseção, comparações para matrizes de elementos finitos, matrizes mais densas e explanação da influência da esparsidade.

O sétimo e derradeiro capítulo centra esforços numa síntese dos resultados apresentados no capítulo anterior e numa conclusão onde as questões propostas no início do trabalho são discutidas em função dos resultados encontrados.

Completa este volume um pequeno conjunto de apêndices onde são descritos o método de armazenamento esparsa de matrizes do EFCAD (2D), com exemplo e rotinas de acesso e a exemplificação numérica de um pequeno exemplo do pré-condicionamento *LU* por blocos proposto.

1.5 Notação

Em todo este trabalho de tese, todas as matrizes são escritas em letras maiúsculas formatadas em itálico e negrito (\mathbf{A}), os vetores por letras minúsculas em itálico e negrito (\mathbf{b}), e variáveis escalares por letras comuns (a) ou símbolos (α). Exceto se explicitamente mencionado, as matrizes são quadradas com dimensão n , e os vetores são n -vetores.

Algumas letras têm um sentido predefinido nesta tese. Isto se aplica à matriz de coeficientes \mathbf{A} , à matriz de pré-condicionamento \mathbf{M} , ao vetor do lado direito \mathbf{b} , à solução exata

$$\hat{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b},$$

à qualquer aproximação da solução $\tilde{\mathbf{x}}$, ao vetor resíduo

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}},$$

e ao vetor erro

$$\mathbf{e} = \hat{\mathbf{x}} - \tilde{\mathbf{x}}.$$

Entretanto, a fim de se manter uma homogeneidade com a literatura, nas descrições dos algoritmos foram mantidas as variáveis tradicionalmente utilizadas nos textos correlatos. Nestes casos as descrições destas variáveis são explicitadas nas suas definições.

O símbolo $\langle \mathbf{x}, \mathbf{y} \rangle$ representa o produto escalar entre os dois vetores quaisquer \mathbf{x} e \mathbf{y} , $\|\mathbf{x}\|$ representa a norma de um vetor qualquer ou a norma de uma matriz $\|\mathbf{B}\|$ qualquer, dependendo de quem está sendo operado.

2 Elementos Finitos e suas Matrizes

2.1 Introdução

O objetivo deste capítulo é apresentar a forma estrutural do sistema de matrizes resultantes da discretização das equações de um dispositivo eletromagnético. Deseja-se conhecer a forma das matrizes geradas pela discretização, isto porque os métodos iterativos estudados neste trabalho são aplicados, em quase sua totalidade, apenas a sistemas com matrizes esparsas e com número de elementos de ordem muito elevada [7]. Sistemas com matrizes cheias são mais eficientemente resolvidos com métodos diretos.

Desta forma, a obtenção da equação do dispositivo eletromagnético será apresentada de forma breve com ênfase às matrizes obtidas ao longo do processo e ao sistema matricial geral obtido, para que o problema numérico seja conhecido para, em seguida no próximo capítulo, conhecer-se os métodos adequados à solução do mesmo.

O desenvolvimento aprofundado das equações para o dispositivo é encontrado em [4], podendo-se citar outras fontes na literatura como [2] e [45].

2.2 Equações do dispositivo eletromagnético

O modelo de partida para o dispositivo eletromagnético são as equações de Maxwell, que regem o conjunto dos fenômenos eletromagnéticos clássicos. As hipóteses de trabalho em eletrotécnica permitem extrair um modelo dinâmico que constitui o problema da magnetodinâmica. Sua solução permite caracterizar a evolução temporal dos campos eletromagnéticos no espaço, e em particular, aqueles de correntes

induzidas nos materiais condutores, em presença de certas fontes.

A seguir, apresenta-se a formulação em potencial vetor magnético do problema da magnetodinâmica, equivalente ao modelo de partida, mas que se distingue pela utilização de potenciais escalar e vetor. Estes potenciais são as ferramentas matemáticas e são os intermediários para a determinação dos campos eletromagnéticos. A noção de potencial é muito interessante, pois ela permite tornar implícita uma das equações de Maxwell a resolver.

A formulação em potencial vetor magnético em 3D é dada por:

$$\mathbf{rot} \frac{1}{\mu} \mathbf{rot} \mathbf{A} + \sigma \mathbf{grad} V + \sigma \frac{\partial \mathbf{A}}{\partial t} = \mathbf{J}_s + \mathbf{rot} \frac{1}{\mu} \mathbf{B}_r \quad (2.1)$$

onde:

μ - permeabilidade magnética (H/m);

\mathbf{A} - potencial vetor magnético (Wb/m);

σ - condutividade elétrica (s/m);

V - potencial escalar magnético (Volt);

\mathbf{B}_r - indução magnética remanente, a qual é acrescentada para tratar ímãs permanentes (Tesla);

\mathbf{J}_s - densidade superficial de corrente de condução (A/m^2).

Quando o dispositivo eletromagnético é suficientemente longo, a análise pode ser conduzida sobre um corte que reconduz a um problema bidimensional (2D).

Assim, para exemplificar a estrutura matricial a ser explorada pelos algoritmos de resolução de sistemas de equação em ambiente paralelizado, a equação (2.1) será desenvolvida em duas dimensões, já que isso não representa perda da generalidade.

Deste modo, se as correntes de excitação que criam o campo magnético são ortogonais ao plano de estudo, o potencial vetor só tem um componente dirigido segundo a direção destas correntes. Considerando a figura 2.1, tem-se:

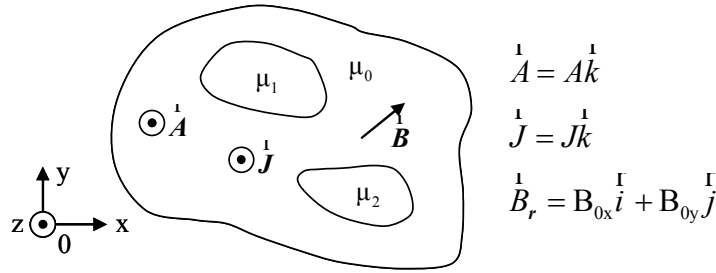


Fig. 2.1 – Domínio de estudo em 2D.

Assim, a equação da magnetodinâmica em 2D considerando os condutores maciços e finos é expressa por:

$$\frac{\partial}{\partial x} \left(\frac{1}{\mu} \frac{\partial A}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\mu} \frac{\partial A}{\partial y} \right) + \sigma \frac{\partial A}{\partial t} + \frac{\sigma}{L} U_m + \frac{N_{co}}{s_f} I_f = \frac{1}{\mu} \frac{\partial B_{0x}}{\partial y} - \frac{1}{\mu} \frac{\partial B_{0y}}{\partial x} \quad (2.2)$$

$$U_m = R_m I_m + R_m \iint_{S_m} \sigma \frac{\partial A}{\partial t} ds \quad (2.3)$$

$$U_f = R_f I_f + L_f \frac{dI_f}{dt} + \frac{N_{co} L}{s_f} \iint_{S_f} \frac{\partial A}{\partial t} ds \quad (2.4)$$

onde N_{co} é o número de condutores finos, s_f é a seção transversal da bobina composta por condutores finos, I é a corrente na bobina, U é a tensão na bobina, R é a resistência da bobina e L é a profundidade da estrutura. L_f é a indutância adicional do circuito elétrico como, por exemplo, a indutância das cabeças de bobina. O índice m é referente aos condutores maciços e o índice f aos condutores finos.

Condutores maciços ou espessos são os condutores que possuem dimensões, tais que, em relação às frequências dos fenômenos, apresentam efeito pelicular.

Condutores ou indutores finos são os condutores que possuem dimensões suficientemente reduzidas de maneira que se pode considerar que a corrente é uniformemente distribuída sobre sua seção transversal.

Nos parágrafos anteriores foi apresentada uma formulação contínua do problema magnetodinâmico sob a forma de equações diferenciais de derivadas parciais. Estas equações regem a distribuição dos campos vetoriais ou das funções escalares.

A resolução de tais equações nem sempre pode ser obtida analiticamente e a utilização de métodos numéricos se torna necessária para se obter uma solução

aproximada do problema. O papel dos métodos numéricos é o de substituir a formulação contínua por uma formulação discreta.

Para discretizar a formulação magnetodinâmica, as equações escritas de uma forma diferencial são conduzidos para uma forma integral, a qual se adapta melhor à discretização pelo método de elementos finitos. Este método consiste em realizar uma malha na estrutura estudada e interpolar as incógnitas sob os elementos desta malha.

A formulação discreta para o potencial vetor é obtida a seguir. Primeiramente se define o resíduo vetorial:

$$\mathbf{R} = \frac{\partial}{\partial x} \left(\frac{1}{\mu} \frac{\partial A}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\mu} \frac{\partial A}{\partial y} \right) + \sigma \frac{\partial A}{\partial t} + \frac{N_{co}}{s_f} I_f + \frac{\sigma}{L} U_m + \frac{1}{\mu} \frac{\partial B_{0y}}{\partial x} - \frac{1}{\mu} \frac{\partial B_{0x}}{\partial y} \quad (2.5)$$

Efetua-se o produto escalar de \mathbf{R} por uma função peso ω , obtendo o chamado método dos resíduos ponderados:

$$\iint_S \mathbf{R} \cdot \omega \, ds = 0 \quad (2.6)$$

Onde:

S – área do domínio de estudo.

Neste caso, a equação (2.2) para o dispositivo se torna:

$$\begin{aligned} & \iint_S \left[\frac{\partial}{\partial x} \left(\frac{1}{\mu} \frac{\partial A}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\mu} \frac{\partial A}{\partial y} \right) \right] \omega \, ds + \iint_S \frac{\sigma}{L} U_m \omega \, ds \\ & + \iint_S \sigma \frac{\partial A}{\partial t} \omega \, ds + \iint_S \frac{N_{co}}{s_f} I_f \omega \, ds = \iint_S \left(\frac{1}{\mu} \frac{\partial B_{0x}}{\partial y} - \frac{1}{\mu} \frac{\partial B_{0y}}{\partial x} \right) \omega \, ds \end{aligned} \quad (2.7)$$

Tomando um elemento triangular como referência, adotando uma aproximação linear para definir os potenciais vetor e escalar no elemento através de funções de interpolação e, considerando as funções de peso iguais às funções de interpolação (Método de Galerkin) pode-se obter o desenvolvimento dos diversos termos da equação 2.7, que, já discretizada e, para um dos elementos da malha de superfície S_i , resulta

para o primeiro termo da equação (2.7):

$$\iint_{S_i} \left[\frac{\partial}{\partial x} \left(\frac{1}{\mu} \frac{\partial A}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\mu} \frac{\partial A}{\partial y} \right) \right] \omega \, ds = -\frac{1}{2\mu D} \begin{bmatrix} q_1 q_1 + r_1 r_1 & q_1 q_2 + r_1 r_2 & q_1 q_3 + r_1 r_3 \\ \text{simétrica} & q_2 q_2 + r_2 r_2 & q_2 q_3 + r_2 r_3 \\ & & q_3 q_3 + r_3 r_3 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \quad (2.8.1)$$

Onde:

$$p_1 = x_2 y_3 - y_2 x_3;$$

$$q_1 = y_2 - y_3;$$

$$r_1 = x_3 - x_2;$$

S_i – área do i -ésimo elemento.

e os outros termos p_2 , p_3 , q_2 , q_3 , r_2 e r_3 são obtidos pela permutação cíclica dos índices. As variáveis x e y são as coordenadas dos pontos e D é igual a duas vezes a superfície do triângulo (elemento).

O segundo termo resulta:

$$\iint_{S_i} \frac{\sigma}{L} U_m \omega \, ds = \frac{\sigma U_m D}{6L} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad (2.8.2)$$

O terceiro termo resulta:

$$\iint_{S_i} \sigma \frac{\partial A}{\partial t} \omega \, ds = -\frac{\sigma D}{24} \frac{\partial A}{\partial t} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad (2.8.3)$$

O quarto termo resulta:

$$\iint_{S_i} \frac{N_{co}}{s_f} I_f \omega \, ds = \frac{1}{6} \frac{N_{co}}{s_f} I_f D \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad (2.8.4)$$

O quinto termo resulta:

$$\iint_{S_i} \left(\frac{1}{\mu} \frac{\partial B_{0x}}{\partial y} - \frac{1}{\mu} \frac{\partial B_{0y}}{\partial x} \right) \omega \, ds = -\frac{1}{2\mu} \begin{bmatrix} q_1 b_{0y} - r_1 b_{0x} \\ q_2 b_{0y} - r_2 b_{0x} \\ q_3 b_{0y} - r_3 b_{0x} \end{bmatrix} \quad (2.8.5)$$

Seguindo o mesmo raciocínio para os demais elementos da malha, as equações (2.2) a (2.4) para todo o domínio, considerando os condutores finos e maciços e considerando a resolução de cada integral (2.8), podem ser organizadas na forma de um sistema matricial de equações, como a seguir [5]:

$$\begin{cases} \mathbf{SS} \, A + N \frac{dA}{dt} - \mathbf{P}' \, U_m - \mathbf{P} \, I_f = \mathbf{D} \\ \mathbf{Q}' \frac{dA}{dt} + \mathbf{R}' \, I_m = U_m \\ \mathbf{Q} \frac{dA}{dt} + \mathbf{L} \frac{dI_f}{dt} + \mathbf{R} \, I_f = U_f \end{cases} \quad (2.9)$$

Onde:

A – matriz do potencial vetor nos nós da malha;

\mathbf{SS} – matriz de permeabilidade;

N – matriz de condutividade;

\mathbf{P}' – matriz que relaciona a tensão no elemento aos nós do elemento;

\mathbf{P} – matriz que relaciona a corrente no elemento aos nós do elemento;

\mathbf{D} – vetor de excitações devido aos ímãs permanentes;

\mathbf{Q}' – matriz de enlace de fluxo;

\mathbf{R}' – matriz diagonal contendo as resistências à corrente contínua dos condutores maciços;

\mathbf{Q} – matriz de enlace de fluxo nos enrolamentos;

\mathbf{R} – matriz das resistências à corrente contínua dos enrolamentos;

\mathbf{L} – matriz das indutâncias das cabeças de bobinas;

I_f – matriz das correntes relativas aos condutores finos;

U_f – matriz das tensões relativas aos condutores finos;

I_m – matriz das correntes relativas aos condutores maciços;

U_m – matriz das tensões relativas aos condutores maciços.

O sistema (2.9) é um sistema que, para ser resolvido numericamente, precisa ser discretizado no tempo. Utilizando o método de Euler sobre as derivadas temporais do vetor potencial magnético e das correntes nos enrolamentos finos, o sistema adquire a seguinte estrutura:

$$\begin{cases} \mathbf{SS} \mathbf{A}(t) + \mathbf{N} \left[\frac{\mathbf{A}(t) - \mathbf{A}(t - \Delta t)}{\Delta t} \right] - \mathbf{P}' \mathbf{U}_m(t) - \mathbf{P} \mathbf{I}_f(t) = \mathbf{D} \\ \mathbf{Q}' \left[\frac{\mathbf{A}(t) - \mathbf{A}(t - \Delta t)}{\Delta t} \right] + \mathbf{R}' \mathbf{I}_m(t) = \mathbf{U}_m(t) \\ \mathbf{Q} \left[\frac{\mathbf{A}(t) - \mathbf{A}(t - \Delta t)}{\Delta t} \right] + \mathbf{L} \left[\frac{\mathbf{I}_f(t) - \mathbf{I}_f(t - \Delta t)}{\Delta t} \right] + \mathbf{R} \mathbf{I}_f(t) = \mathbf{U}_f(t) \end{cases} \quad (2.10)$$

Organizando este sistema tal que os valores das grandezas compreendidos no tempo $(t - \Delta t)$ sejam entendidos como “fontes”, uma vez que seus valores já foram calculados, o sistema de equações se torna:

$$\begin{cases} \mathbf{SS} \mathbf{A}(t) + \frac{\mathbf{N}}{\Delta t} \mathbf{A}(t) - \mathbf{P} \mathbf{I}_f(t) - \mathbf{P}' \mathbf{U}_m(t) = \mathbf{D} + \frac{\mathbf{N}}{\Delta t} \mathbf{A}(t - \Delta t) \\ \frac{\mathbf{Q}'}{\Delta t} \mathbf{A}(t) + \mathbf{R}' \mathbf{I}_m(t) = \mathbf{U}_m(t) + \frac{\mathbf{Q}'}{\Delta t} \mathbf{A}(t - \Delta t) \\ \frac{\mathbf{Q}}{\Delta t} \mathbf{A}(t) + \left(\frac{\mathbf{L}}{\Delta t} + \mathbf{R} \right) \mathbf{I}_f(t) = \mathbf{U}_f(t) + \frac{\mathbf{Q}}{\Delta t} \mathbf{A}(t - \Delta t) + \frac{\mathbf{L}}{\Delta t} \mathbf{I}_f(t - \Delta t) \end{cases} \quad (2.11)$$

Este sistema possui cinco variáveis: o potencial vetor magnético (A), a corrente e a tensão nos enrolamentos finos ($U_f(t)$ e $I_f(t)$) e a corrente e a tensão nos enrolamentos maciços ($U_m(t)$ e $I_m(t)$).

Da forma como está, este sistema não pode ser resolvido, uma vez que o número

de incógnitas é superior ao número de equações obtidas. Em casos onde os condutores maciços são curto-circuitados como, por exemplo, em rotores com gaiola de esquilo, têm-se $U_m = 0$. Se, além disso, as tensões U_f nos enrolamentos finos são fixadas, consegue-se reduzir o número de variáveis e isso permite obter a solução do sistema com as três equações.

Por outro lado, considerando-se que as tensões nos enrolamentos finos seguem leis específicas associadas a circuitos externos e que os enrolamentos maciços podem ser arrançados em conexões série, paralelo ou anel, dá-se origem a outras significativas equações que demandam outras deduções [45].

Assim, para delimitar o problema e ao mesmo tempo obter uma visão da estrutura matricial a ser abordada, considerar-se-á que a tensão nos enrolamentos finos será imposta e conhecida e que os condutores maciços estarão perfeitamente curto-circuitados. Temos, assim, que a tensão sobre estes será igual a zero.

$$U_m = 0 \quad (2.12)$$

2.3 Análise da estrutura do sistema matricial

Levando-se em conta o exposto no item anterior, o sistema de equações passa a ser:

$$\begin{cases} \mathbf{SS} \mathbf{A}(t) + \frac{\mathbf{N}}{\Delta t} \mathbf{A}(t) - \mathbf{P} \mathbf{I}_f(t) = \mathbf{D} + \frac{\mathbf{N}}{\Delta t} \mathbf{A}(t - \Delta t) \\ \frac{\mathbf{Q}}{\Delta t} \mathbf{A}(t) + \left(\frac{\mathbf{L}}{\Delta t} + \mathbf{R} \right) \mathbf{I}_f(t) = \mathbf{U}_f(t) + \frac{\mathbf{Q}}{\Delta t} \mathbf{A}(t - \Delta t) + \frac{\mathbf{L}}{\Delta t} \mathbf{I}_f(t - \Delta t) \end{cases} \quad (2.13)$$

Resulta que o sistema tem apenas duas incógnitas: $\mathbf{A}(t)$ e $\mathbf{I}_f(t)$. Depois de calculado o potencial vetor magnético \mathbf{A} , retorna-se à equação onde \mathbf{I}_m é a variável para encontrar seu valor.

$$\mathbf{I}_m(t) = \mathbf{R}'^{-1} \left(\frac{\mathbf{Q}'}{\Delta t} \mathbf{A}(t - \Delta t) - \frac{\mathbf{Q}'}{\Delta t} \mathbf{A}(t) \right) \quad (2.14)$$

Este mesmo sistema, considerando as matrizes de contribuição de todos os elementos da malha, as matrizes de contribuição de todos os elementos preenchidos por materiais condutores, as matrizes de contribuição dos elementos preenchidos por condutores finos, adquire a seguinte forma para um caso singular de um só elemento finito triangular:

$$\left[\begin{array}{c|c} \frac{1}{2\mu D} \begin{bmatrix} q_1 q_1 + r_1 r_1 & q_1 q_2 + r_1 r_2 & q_1 q_3 + r_1 r_3 \\ \text{simétrica} & q_2 q_2 + r_2 r_2 & q_2 q_3 + r_2 r_3 \\ & & q_3 q_3 + r_3 r_3 \end{bmatrix} + \frac{\sigma D}{24\Delta t} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} & \frac{N_{co} D}{s_t} \frac{1}{6} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ \hline \frac{N_{co} LD}{s_f} \frac{1}{6} \frac{1}{\Delta t} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} & \frac{\mathbf{L}}{\Delta t} + \mathbf{R} \end{array} \right] \begin{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \\ \hline \mathbf{I}_f \end{bmatrix} = \quad (2.15a)$$

$$\left[\begin{array}{c|c} -\frac{1}{2\mu} \begin{bmatrix} q_1 b_{0y} - r_1 b_{0x} \\ q_2 b_{0y} - r_2 b_{0x} \\ q_3 b_{0y} - r_3 b_{0x} \end{bmatrix} + \frac{\sigma D}{24\Delta t} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} A_1^a \\ A_2^a \\ A_3^a \end{bmatrix} \\ \hline \mathbf{U}_f + \frac{\mathbf{L}}{\Delta t} \mathbf{I}_f^a + \frac{N_{co} LD}{s_f} \frac{1}{6\Delta t} \begin{bmatrix} A_1^a & A_2^a & A_3^a \end{bmatrix} \end{array} \right]$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} [\mathbf{x}] = [\mathbf{b}] \quad (2.15b)$$

onde:

o índice a presente na equação 2.15a se refere aos valores das grandezas no tempo anterior.

A equação 2.16 ilustra o sistema matricial final a ser resolvido.

$$\left[\begin{array}{c|c} \mathbf{SS} + \frac{\mathbf{N}}{\Delta t} & -\mathbf{P} \\ \hline \frac{\mathbf{Q}}{\Delta t} & \frac{\mathbf{L}}{\Delta t} + \mathbf{R} \end{array} \right] \begin{bmatrix} \mathbf{A}(t) \\ \hline \mathbf{I}_f(t) \end{bmatrix} = \begin{bmatrix} \mathbf{D} + \frac{\mathbf{N}}{\Delta t} \mathbf{A}(t - \Delta t) \\ \hline \mathbf{U}_f(t) + \frac{\mathbf{Q}}{\Delta t} \mathbf{A}(t - \Delta t) + \frac{\mathbf{L}}{\Delta t} \mathbf{I}_f(t - \Delta t) \end{bmatrix} \quad (2.16)$$

Segue que o sistema da equação 2.15b tem particularidades estruturais, que são:

- O sistema total não é simétrico.
 - Isto é prontamente notado nas equações 2.15 e 2.16;
- O sistema é esparso.
 - A esparsidade é devida a discretização da geometria e posterior condensação dos elementos finitos na matriz de contribuição total (matriz \mathbf{SS}). Ela dependerá da numeração dos nós dos elementos globais. Elementos não conectados por seus nós não possuem contribuição e daí a esparsidade.
- O subsistema \mathbf{a}_{11} é simétrico e do tipo banda.
 - Como pode ser observado nas equações 2.8.1, 2.8.3 e 2.16, a parcela devido à distribuição do potencial vetor e das correntes induzidas na estrutura tem simetria, já que as matrizes de contribuição dos elementos são simétricas. Já a evolução da condensação na matriz global leva a uma distribuição das contribuições concentrada sobre a diagonal principal formando um sistema do tipo banda.
- A largura de banda é influenciada pelo algoritmo da geração da malha.
 - Como já exposto, a condensação dos elementos na matriz global depende da numeração dos nós. A largura de banda será menor quando os nós dos elementos condensados possuírem uma numeração o mais próxima possível. A largura da banda influenciará decisivamente o modo de armazenamento compactado da matriz em memória.
- Diferença acentuada entre os números de elementos que formam a porção \mathbf{a}_{11} e as demais partes da matriz;
 - A porção \mathbf{a}_{11} da matriz se refere à distribuição do potencial vetor em todo o domínio, enquanto que as matrizes \mathbf{Q} , \mathbf{P} e \mathbf{L} se referem às partes da máquina onde estão colocados os condutores finos com suas resistências e indutâncias. Desta forma o número de elementos que compõe o subsistema \mathbf{a}_{11} é consideravelmente maior que o das outras partes. Este subsistema é responsável pelo maior custo de processamento

de todo o dispositivo.

- Diferenças numéricas entre os elementos pertencentes a \mathbf{a}_{11} e os demais;
 - Os elementos pertencentes a \mathbf{a}_{11} dependem do inverso de μ , resultando um número elevado, quando existe a contribuição. Já as contribuições devido aos condutores dependem de parâmetros construtivos e do passo de tempo, podendo ocasionar diferenças numéricas significativas e possivelmente erros numéricos.
- A matriz total tem simetria de forma, mas não de valores.
 - Pode-se extrapolar o modelo das equações 2.15 e 2.16 e observar que os elementos \mathbf{a}_{12} e \mathbf{a}_{21} possuem uma distribuição semelhante na forma, mas não de valores, tal que suas leis de armazenamento sejam parecidas. A simetria final dependerá também do subsistema \mathbf{a}_{22} .

Esta tese restringiu o estudo a alguns métodos numéricos selecionados (tratados no capítulo que se segue) comparando seus desempenhos sobre o mesmo conjunto de matrizes problemas. Para contemplar este requisito os sistemas tratados devem atender todos os métodos estudados. Expondo de uma forma simplificada: a matriz-problema deve ser simétrica na forma e nos valores.

Isto pode ser obtido a partir de um equacionamento que opere apenas o subsistema \mathbf{a}_{11} da equação 2.16. Tal subsistema advém, por exemplo, no caso de um modelo de elementos finitos, onde a corrente elétrica é conhecida e imposta aos enrolamentos. O capítulo de resultados apresentará os problemas numéricos obtidos a partir destas simplificações, expondo as características das matrizes a serem exploradas.

2.4 Conclusão

O presente capítulo apresentou o problema eletromagnético a ser explorado pelos métodos iterativos que serão apresentados na seqüência deste trabalho. Embora o

sistema completo tenha outras características, a matriz-problema a ser resolvida, conforme o tópico anterior, é simétrica, esparsa, a distribuição dos seus elementos forma uma banda e seu custo computacional de resolução é elevado.

Na continuação serão apresentados métodos numéricos iterativos para a solução do sistema de equações obtidos com a discretização do dispositivo eletromagnético, incluindo o método iterativo não-estacionário fundamental, o método Gradiente Conjugado.

Os capítulos seguintes também considerarão a distribuição esparsa da matriz-problema, uma vez que a esparsidade influencia o produto matriz por vetor e, conseqüentemente, a implementação paralela dos algoritmos de resolução de sistemas de equações.

3 Métodos Iterativos

3.1 Introdução

Neste capítulo se pretende mostrar os métodos iterativos implementados para a resolução dos sistemas de equações produzidos pelo método de elementos finitos. Os métodos aqui apresentados geram aproximações para um subespaço de Krylov. Existem várias formas de apresentar o subespaço de Krylov e, neste trabalho, optou-se por apresentá-lo a partir da iteração básica.

Os métodos iterativos implementados nesta tese são o Gradiente Conjugado, o Gradiente Biconjugado, o Gradiente Biconjugado Estabilizado, e o Gradiente Conjugado Quadrado. Estes métodos são utilizados com pré-condicionadores, tal que os algoritmos aqui mostrados incluem a matriz de pré-condicionamento, mas a explanação sobre a utilização do pré-condicionamento é realizada no próximo capítulo, uma vez que neste trabalho foi desenvolvida uma nova variação do pré-condicionador LU , adaptado à arquitetura de memória distribuída, própria dos clusters de computadores.

3.2 Iteração básica

A idéia atrás dos métodos iterativos é trocar o sistema original por um outro sistema próximo que pode ser resolvido mais facilmente. Ao trocarmos o sistema por um outro próximo dele inserimos erros, tal que a solução passa a ser aproximada iterativamente. A aproximação iterativa do resultado é obtida a partir do seguinte sistema linear [7]:

$$A x = b \tag{3.1}$$

Sendo \mathbf{x}_0 a solução inicial, o resíduo inicial será:

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0 \quad (3.2)$$

$$\mathbf{b} = \mathbf{r}_0 + \mathbf{A} \mathbf{x}_0 \quad (3.3)$$

A solução para o sistema é $\hat{\mathbf{x}}$, assim, reescrevendo a expressão (3.1) em função de (3.3), obtemos:

$$\mathbf{A} \hat{\mathbf{x}} = \mathbf{r}_0 + \mathbf{A} \mathbf{x}_0 \quad (3.4)$$

$$\mathbf{A} \hat{\mathbf{x}} - \mathbf{A} \mathbf{x}_0 = \mathbf{r}_0 \quad (3.5)$$

$$\mathbf{A} (\hat{\mathbf{x}} - \mathbf{x}_0) = \mathbf{r}_0 \quad (3.6)$$

A medida verdadeira de quanto \mathbf{x}_0 se aproxima de $\hat{\mathbf{x}}$, a solução do sistema, é o erro. Assim:

$$\mathbf{e}_0 = \hat{\mathbf{x}} - \mathbf{x}_0 \quad (3.7)$$

Desta forma temos um novo sistema a ser resolvido substituindo (3.7) em (3.6):

$$\mathbf{A} \mathbf{e}_0 = \mathbf{r}_0 \quad (3.8)$$

Com a equação (3.8) se pode definir um processo iterativo que leva à solução. A iteração pode ser definida como:

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{e}_0 \quad (3.9)$$

Com o novo valor de \mathbf{x} se calcula um novo resíduo, calcula-se um novo erro e o processo é repetido até que se alcance a convergência. A expressão (3.9), entretanto, pode ser reescrita conforme o método conhecido como *iteração de ponto fixo*. A forma

desta iteração é mostrada na expressão (3.10):

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{K}_i \mathbf{r}_i. \quad (3.10)$$

Se $\mathbf{K}_i = \mathbf{K}$ então a iteração é estacionária. Generalizando a equação (3.2) para qualquer iteração i , o resíduo passa a ser definido por:

$$\mathbf{r}_i = \mathbf{b} - \mathbf{A} \mathbf{x}_i \quad (3.11)$$

Substituindo (3.11) em (3.10), a iteração de ponto fixo adquire a seguinte forma:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{K} (\mathbf{b} - \mathbf{A} \mathbf{x}_i). \quad (3.12)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{K} \mathbf{b} - \mathbf{K} \mathbf{A} \mathbf{x}_i = \mathbf{K} \mathbf{b} + \mathbf{x}_i - \mathbf{K} \mathbf{A} \mathbf{x}_i \quad (3.13)$$

$$\mathbf{x}_{i+1} = \mathbf{K} \mathbf{b} + (\mathbf{I} - \mathbf{K} \mathbf{A}) \mathbf{x}_i. \quad (3.14)$$

A matriz \mathbf{K} pode ser entendida como uma matriz de pré-condicionamento.

Fazendo-se $\mathbf{K} = \mathbf{I}$, o sistema (3.14) transforma-se na iteração básica de Richardson.

$$\mathbf{x}_{i+1} = \mathbf{b} + (\mathbf{I} - \mathbf{A}) \mathbf{x}_i \quad (3.15)$$

Generalizando a expressão para o erro (3.8) para a i -ésima iteração, tal que $\mathbf{e}_i = \hat{\mathbf{x}} - \mathbf{x}_i$, e multiplicando ambos os lados da mesma por \mathbf{A} , obtém-se:

$$\mathbf{A} \mathbf{e}_i = \mathbf{A} (\hat{\mathbf{x}} - \mathbf{x}_i) \quad (3.16)$$

$$\mathbf{A} \mathbf{e}_i = \mathbf{A} \hat{\mathbf{x}} - \mathbf{A} \mathbf{x}_i \quad (3.17)$$

$$\mathbf{A} \mathbf{e}_i = \mathbf{b} - \mathbf{A} \mathbf{x}_i \quad (3.18)$$

$$\mathbf{A} \mathbf{e}_i = \mathbf{r}_i \quad (3.19)$$

A equação (3.19) é conhecida como equação residual. Distribuindo o produto de

(3.15), obtém-se:

$$\mathbf{x}_{i+1} = \mathbf{b} + \mathbf{I} \mathbf{x}_i - \mathbf{A} \mathbf{x}_i \quad (3.20)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{b} - \mathbf{A} \mathbf{x}_i \quad (3.21)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{r}_i \quad (3.22)$$

Multiplicando-se (3.22) por $-\mathbf{A}$ em ambos os lados e somando \mathbf{b} ,

$$-\mathbf{A} \mathbf{x}_{i+1} + \mathbf{b} = -\mathbf{A}(\mathbf{x}_i + \mathbf{r}_i) + \mathbf{b} \quad (3.23)$$

$$\mathbf{b} - \mathbf{A} \mathbf{x}_{i+1} = -\mathbf{A} \mathbf{x}_i - \mathbf{A} \mathbf{r}_i + \mathbf{b} \quad (3.24)$$

Substituindo-se $\mathbf{r}_{i+1} = \mathbf{b} - \mathbf{A} \mathbf{x}_{i+1}$ na equação acima, resulta:

$$\mathbf{r}_{i+1} = \mathbf{b} - \mathbf{A} \mathbf{x}_i - \mathbf{A} \mathbf{r}_i = \mathbf{r}_i - \mathbf{A} \mathbf{r}_i \quad (3.25)$$

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{A})\mathbf{r}_i \quad (3.26)$$

A expressão do resíduo também pode ser expressa na forma de potências, pois:

$$\mathbf{r}_1 = (\mathbf{I} - \mathbf{A})\mathbf{r}_0 \quad (3.27)$$

$$\mathbf{r}_2 = (\mathbf{I} - \mathbf{A})\mathbf{r}_1 = (\mathbf{I} - \mathbf{A})(\mathbf{I} - \mathbf{A})\mathbf{r}_0 = (\mathbf{I} - \mathbf{A})^2 \mathbf{r}_0 \quad (3.28)$$

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{A})^{i+1} \mathbf{r}_0 \quad (3.29)$$

Chamando:

$$\mathbf{P}_{i+1}(\mathbf{A}) = (\mathbf{I} - \mathbf{A})^{i+1} \quad (3.30)$$

um polinômio em função da matriz \mathbf{A} , a expressão (3.29) pode ser reformulada, tal que:

$$\mathbf{r}_{i+1} = \mathbf{P}_{i+1}(\mathbf{A})\mathbf{r}_0 \quad (3.31)$$

Para uma matriz \mathbf{A} linearmente independente, a equação (3.31) pode ser rescrita, depois de algumas passagens algébricas, em termos do erro $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}_i$ e do resíduo $\mathbf{A} \mathbf{e}_i = \mathbf{r}_i$, passando a ser:

$$\hat{\mathbf{x}} - \mathbf{x}_{i+1} = \mathbf{P}_{i+1}(\mathbf{A})(\mathbf{x} - \mathbf{x}_0) \quad (3.32)$$

Nestas equações \mathbf{P} é um polinômio especial de grau $i+1$, onde $\mathbf{P}_{i+1}(0) = 1$.

Destas equações podemos ressaltar as seguintes conclusões:

- 1) Da equação (3.28) observa-se que, se $\|\mathbf{I} - \mathbf{A}\| < 1$, haverá convergência na expressão, pois $\|\mathbf{r}_{i+1}\| \leq \|\mathbf{I} - \mathbf{A}\| \cdot \|\mathbf{r}_i\|$. Se o sistema for pré-condicionado, esta observação impõe restrições ao pré-condicionador que deve atender esta propriedade. Esta propriedade aplicada ao pré-condicionamento será mais bem explanada no próximo capítulo.
- 2) A equação (3.29) também mostra que os resíduos podem ser expressos em termos das potências de \mathbf{A} vezes o resíduo inicial.
- 3) A equação (3.32) mostra que a redução do erro (e do resíduo) depende do polinômio \mathbf{P}_i e de como ele amortece as componentes iniciais do erro. Observando esta terceira propriedade torna-se desejável que este polinômio insira condições de amortecimento melhores que o sistema original.

3.3 As abordagens para o subespaço de Krylov

Pergunta-se se é possível identificar um subespaço no qual as sucessivas aproximações sugeridas pela equação (3.32) estão localizadas. Esta resposta é obtida repetindo-se a iteração simples de Richardson:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{r}_i \quad (3.33)$$

$$\mathbf{x}_{i+1} - \mathbf{x}_0 = \mathbf{r}_0 + \mathbf{r}_1 + \mathbf{r}_2 + \dots + \mathbf{r}_i \quad (3.34)$$

$$\mathbf{x}_{i+1} - \mathbf{x}_0 = \sum_{j=0}^i \mathbf{r}_j \quad (3.35)$$

Generalizando a equação (3.29) tal que $\mathbf{r}_j = (\mathbf{I} - \mathbf{A})^j \mathbf{r}_0$ e substituindo na equação (3.35) obtém-se:

$$\mathbf{x}_{i+1} - \mathbf{x}_0 = \sum_{j=0}^i (\mathbf{I} - \mathbf{A})^j \mathbf{r}_0 \quad (3.36)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_0 + \sum_{j=0}^i (\mathbf{I} - \mathbf{A})^j \mathbf{r}_0 \quad (3.37)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_0 + (\mathbf{I} - \mathbf{A})^0 \mathbf{r}_0 + (\mathbf{I} - \mathbf{A})^1 \mathbf{r}_0 + \dots + (\mathbf{I} - \mathbf{A})^i \mathbf{r}_0 \quad (3.38)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_0 + [\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \mathbf{A}^3\mathbf{r}_0, \dots, \mathbf{A}^i\mathbf{r}_0] \quad (3.39)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_0 + \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \mathbf{A}^3\mathbf{r}_0, \dots, \mathbf{A}^i\mathbf{r}_0\} \quad (3.40)$$

Um subespaço que pode ser escrito na forma:

$$\mathbf{x}_{i+1} \in \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \mathbf{A}^3\mathbf{r}_0, \dots, \mathbf{A}^i\mathbf{r}_0\} \quad (3.41)$$

é conhecido como subespaço de Krylov para a matriz \mathbf{A} e para o vetor \mathbf{r}_0 , escrito como $\mathbf{K}^{i+1}(\mathbf{A}; \mathbf{r}_0)$.

Os métodos que procuram gerar aproximações melhores do subespaço de Krylov são chamados de Métodos do Subespaço de Krylov. Estes métodos podem ser separados, de acordo com [46] nas seguintes classes:

- 1) Métodos de projeção que resolvem um sistema de equações lineares $\mathbf{A}\mathbf{x} = \mathbf{b}$ construindo soluções aproximadas \mathbf{x}_i tal que o resíduo seja ortogonal ao subespaço eleito. Esta abordagem é conhecida como método de projeção ortogonal com condição de *Ritz-Galerkin*: $(\mathbf{b} - \mathbf{A}\mathbf{x}_i) \perp \mathbf{K}^i(\mathbf{A}; \mathbf{r}_0)$.
- 2) Métodos de projeção que resolvem um sistema de equações lineares $\mathbf{A}\mathbf{x} = \mathbf{b}$ construindo soluções aproximadas \mathbf{x}_i tal que o resíduo não mais seja ortogonal ao subespaço eleito, e sim ortogonal a um outro subespaço. O resíduo nestes métodos é oblíquo ao subespaço eleito. Esta abordagem é conhecida como método de projeção

oblíqua com condição de *Petrov-Galerkin*: $(\mathbf{b} - \mathbf{A} \mathbf{x}_i) \perp \mathbf{K}^m(\mathbf{A}; \mathbf{r}_0) \neq \mathbf{K}^i(\mathbf{A}; \mathbf{r}_0)$.

- 3) Métodos de minimização de norma que resolvem sistemas de equações lineares $\mathbf{A} \mathbf{x} = \mathbf{b}$ construindo soluções aproximadas \mathbf{x}_i tal que norma euclidiana do resíduo $\|\mathbf{b} - \mathbf{A} \mathbf{x}_i\|_2$ é mínima sobre $\mathbf{K}^i(\mathbf{A}; \mathbf{r}_0)$. Esta é a abordagem do *resíduo mínimo*.

A primeira abordagem, *Ritz-Galerkin*, conduz ao método dos Gradientes Conjugados entre outros. A abordagem do *resíduo mínimo* produz métodos como o GMRES. A desvantagem destas duas formas reside no fato de que para a maioria dos sistemas não simétricos as relações de recorrência tornam-se longas e computacionalmente custosas. A segunda abordagem, *Petrov-Galerkin*, utilizando um subespaço i -dimensional, leva a métodos como o BiCG. Também foram desenvolvidos métodos híbridos dessas abordagens resultando em métodos como o CGS.

A escolha do método é um problema delicado. Se a matriz \mathbf{A} é simétrica definida positiva a escolha é fácil: Gradientes Conjugados. Para outros tipos de matrizes a situação é menos definida. O método GMRES, proposto em 1986 por SAAD e SCHULTZ [14], é o mais robusto, mas em termos de recursos por iteração é um método exigente. O BiCG, sugerido por FLETCHER em 1977 [13], é uma alternativa que, como o CG, necessita relativamente pouca capacidade de armazenamento. Tem, entretanto, problemas de convergência: as chamadas situações de *breakdown*. Para contornar estes problemas foram, posteriormente, desenvolvidas diversas técnicas conhecidas como *look-ahead*. O desenvolvimento de métodos híbridos começou em 1989 com o CGS proposto por SONNEVELD [15] e foi seguido pelo BiCGStab, proposto por Henk van der VORST em 1992 [16], e depois por outros métodos.

A figura 3.1 a seguir, retirada de DEMMEL [47], ilustra a escolha do método segundo o tipo de matriz. Deve-se levar em consideração que esta é uma sugestão de decisão e não uma regra geral. Este texto descreve apenas os métodos utilizados nas experiências de paralelização realizadas nesta tese. Para estes métodos são descritas as características computacionais relevantes à paralelização, recursos de memória e armazenamento. O apêndice apresenta uma breve descrição da teoria que os regem e uma descrição simplificada dos demais métodos citados na fig. 3.1. Para informações mais detalhadas a respeito dos métodos utilizados neste trabalho e em outros indicam-

se as referências [17], [39] e [48].

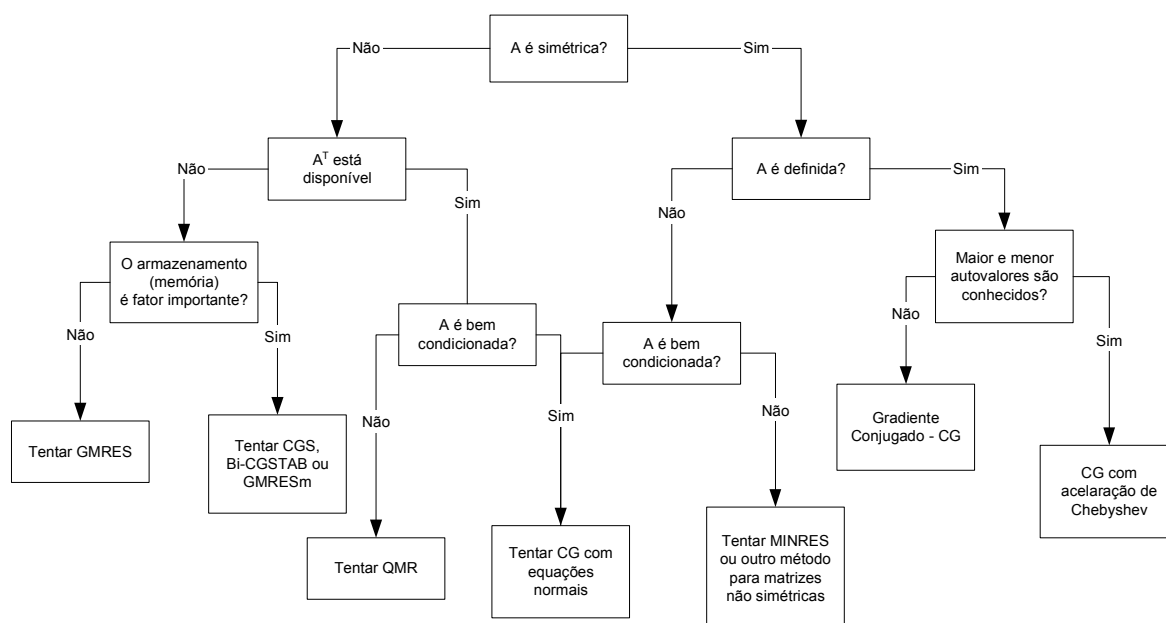


Fig. 3.1 – Árvore de decisão: escolha do método iterativo em função da matriz A .

3.4 A escolha do método

As primeiras experiências realizadas com as matrizes provenientes do método de elementos finitos [49] mostraram que estas possuíam números de condicionamento muito elevados. O método GMRESm (Resíduo Mínimo Generalizado com reinicialização) se mostrou muito robusto e rápido nestas experiências, apresentando o menor número de iterações nos casos estudados. Entretanto, nesta etapa do trabalho as dimensões das matrizes eram reduzidas, o que permitia a geração de um espaço linear interno no algoritmo de tamanho reduzido. Na segunda etapa do trabalho, com matrizes de tamanho substancialmente maior, o tamanho do subespaço interno necessário ao GMRESm tornou-se problemático, consumindo grandes quantidades de memória e, com isso, elevando o tempo de resolução. A partir destes resultados, decidiu-se trocar o método por outros de implementação e necessidades de memória mais facilitadas.

Assim, além do método BiCG, um método tradicionalmente utilizado no GRUCAD e já explorado na sua versão paralela, implementaram-se também as versões

paralelas do Gradiente Biconjugado Estabilizado e do Gradiente Conjugado Quadrado, o primeiro fortemente recomendado pelos resultados conseguidos com o mesmo na área de planejamento de sistemas de energia [50].

Desta forma, com estes dois métodos, experiências com matrizes não simétricas poderiam ser empreendidas sem a necessidade da matriz transposta A^T . Com isso, para a conclusão deste trabalho, foram eleitos os métodos Gradiente Conjugado, Gradiente Biconjugado, Gradiente Biconjugado Estabilizado e Gradiente Conjugado Quadrado.

3.5 Os métodos implementados

3.5.1 Método do Gradiente Conjugado (CG)

Este método foi descoberto independentemente, no início dos anos cinquenta, por Magnus R. HESTENES e Eduard STIEFEL, que logo após reuniram seus trabalhos e publicaram aquela que é considerada a primeira referência sobre o CG [10]. O método é muito eficiente para sistemas com matrizes simétricas definidas positivas. O método funciona gerando seqüências de vetores de soluções aproximadas, de vetores de resíduo e de vetores de direções de procura, usados para atualizar as soluções aproximadas e os resíduos. Embora a seqüência de iterações possa se tornar longa, o método tem poucos vetores que precisam ser armazenados.

O vetor com a solução aproximada é atualizado em cada iteração por um múltiplo (α_i) do vetor que guarda a direção de procura (\mathbf{p}_i):

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i \quad (3.42)$$

Os vetores de resíduo $\mathbf{r}_i = \mathbf{b} - \mathbf{A} \mathbf{x}_i$ são atualizados da seguinte forma:

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{q}_i \quad (3.43)$$

Onde:

$$\mathbf{q}_i = \mathbf{A} \mathbf{p}_i \quad (3.44)$$

O valor de α_i é escolhido tal que:

$$\alpha_i = \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \quad (3.45)$$

E as direções de procura são atualizadas através dos resíduos pela seguinte relação:

$$\mathbf{p}_i = \mathbf{r}_i + \beta_{i-1} \mathbf{p}_{i-1} \quad (3.46)$$

Onde a escolha de $\beta_i = \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{r}_{i-1}^T \mathbf{r}_{i-1}}$ assegura que \mathbf{p}_i e $\mathbf{A} \mathbf{p}_{i-1}$ sejam ortogonais.

O critério de parada é obtido analisando-se a norma do resíduo. O algoritmo deixa de ser executado quando a norma cai abaixo de algum valor especificado, freqüentemente $\|\mathbf{r}_i\| \leq \varepsilon \|\mathbf{r}_0\|$.

Este método utiliza um produto matriz por vetor, três atualizações de vetores, e dois produtos internos por iteração.

O algoritmo do CG, como encontrado em [48], pode ser visto na figura 3.2. Neste algoritmo \mathbf{A} é a matriz de coeficientes, \mathbf{M} é a matriz de pré-condicionamento, \mathbf{b} é o vetor do lado direito, \mathbf{x}_0 é o vetor com a estimativa inicial para a solução, \mathbf{r}_0 é o vetor resíduo inicial, \mathbf{p}_i e \mathbf{p}_{i-1} são vetores com as direções de procura da iteração atual e da anterior, \mathbf{z}_0 e \mathbf{z}_{i-1} são vetores, α_i , β_i , são escalares que guardam o passo de procura e o a correção da direção de procura respectivamente, e ρ_{i-1} e ρ_{i-2} são escalares auxiliares.

Algoritmo para o Gradiente Conjugado

Escolhe uma estimativa inicial \mathbf{x}_0 .

Calcula-se $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$

para $i=1, 2, \dots$

 resolve $\mathbf{M} \mathbf{z}_{i-1} = \mathbf{r}_{i-1}$

$\rho_{i-1} = \mathbf{r}_{i-1}^T \mathbf{z}_{i-1}$

 se $i=1$

$\mathbf{p}_1 = \mathbf{z}_0$

 senão

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$\mathbf{p}_i = \mathbf{z}_{i-1} + \beta_{i-1} \mathbf{p}_{i-1}$

 fim se

$\mathbf{q}_i = \mathbf{A} \mathbf{p}_i$

$\alpha_i = \rho_{i-1} / \mathbf{p}_i^T \mathbf{q}_i$

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \mathbf{p}_i$

$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i \mathbf{q}_i$

 se $\|\mathbf{r}_i\|_2 \leq \varepsilon$, \mathbf{x}_i é a solução

 senão, continua

fim

Fig. 3.2 – Algoritmo do Método Gradiente Conjugado.

3.5.2 Método Gradiente Biconjugado (BiCG)

O Gradiente Conjugado não é aplicado a sistemas não simétricos porque os vetores residuais não podem ser feitos ortogonais com poucas iterações. O método GMRES mantém a ortogonalidade dos resíduos utilizando longas séries de iterações, ao custo de altas cargas de armazenamento. O método do Gradiente Biconjugado usa uma outra abordagem, trocando a seqüência ortogonal dos resíduos por duas seqüências mutuamente ortogonais, ao preço de não mais realizar a minimização.

As relações de atualização dos resíduos no método do Gradiente Conjugado são aumentadas no método Biconjugado por relações que são semelhantes, mas utilizam \mathbf{A}^T ao invés de \mathbf{A} . O método atualiza duas seqüências de resíduos:

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A} \mathbf{p}_i \quad (3.47)$$

$$\tilde{\mathbf{r}}_{i+1} = \tilde{\mathbf{r}}_i - \alpha_i \mathbf{A}^T \tilde{\mathbf{p}}_i \quad (3.48)$$

e duas seqüências de direção de procura:

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta_{i+1} \mathbf{p}_i \quad (3.49)$$

$$\tilde{\mathbf{p}}_{i+1} = \tilde{\mathbf{r}}_{i+1} + \beta_{i+1} \tilde{\mathbf{p}}_i \quad (3.50)$$

com os valores de α_i e β_i iguais a:

$$\alpha_i = \frac{\tilde{\mathbf{r}}_i^T \mathbf{r}_i}{\tilde{\mathbf{p}}_i^T \mathbf{A} \mathbf{p}_i} \quad (3.51)$$

$$\beta_i = \frac{\tilde{\mathbf{r}}_{i+1}^T \mathbf{r}_{i+1}}{\tilde{\mathbf{r}}_i^T \mathbf{r}_i} \quad (3.52)$$

Se a matriz do sistema for simétrica definida positiva a convergência do BiCG será a mesma do método CG, mas com o dobro do custo a cada iteração. Se a matriz for não simétrica, mas a redução da norma do resíduo for significativa, o método é comparável ao GMRES completo. Mas dependendo do sistema a convergência pode ser irregular e, em alguns casos, o método pode não convergir. Existem estratégias para contornar estas deficiências que envolvem a reinicialização do método ou utilizando outra forma de decomposição.

O BiCG requer o cálculo do produto matriz por vetor $\mathbf{A} \mathbf{p}_i$, do produto transposto $\mathbf{A}^T \tilde{\mathbf{p}}_i$, dois produtos internos, e cinco operações de atualização. A matriz transposta, portanto, deve estar disponível. Os produtos matriz por vetor podem ser implementados em ambientes paralelos onde a utilização da cópia da matriz na forma transposta facilita a implementação, mas duplica a necessidade de armazenamento.

O algoritmo do BiCG, como encontrado em [48], pode ser visto na figura 3.3. Além dos vetores e escalares idênticos aos utilizados no algoritmo do Gradiente Conjugado, neste algoritmo aparecem os vetores $\tilde{\mathbf{p}}_i$, $\tilde{\mathbf{r}}_i$ e $\tilde{\mathbf{z}}_i$ nas

iterações i e $i-1$, significando a direção de procura, o resíduo e o resíduo M -ortogonal, conjugados, respectivamente.

Algoritmo para o Gradiente Biconjugado

Escolhe uma estimativa inicial \mathbf{x}_0 .

Calcula-se $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$

Escolhe-se $\tilde{\mathbf{r}} = \mathbf{r}_0$

para $i=1, 2, \dots$

 resolve $\mathbf{M} \mathbf{z}_{i-1} = \mathbf{r}_{i-1}$

 resolve $\mathbf{M}^T \tilde{\mathbf{z}}_{i-1} = \tilde{\mathbf{r}}_{i-1}$

$\rho_{i-1} = \mathbf{z}_{i-1}^T \mathbf{r}_{i-1}$

 se $\rho_{i-1} = 0$ o método falha

 se $i=1$

$\mathbf{p}_{i-1} = \mathbf{z}_{i-1}$

$\tilde{\mathbf{p}}_{i-1} = \tilde{\mathbf{z}}_{i-1}$

 senão

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$\mathbf{p}_i = \mathbf{z}_{i-1} + \beta_{i-1} \mathbf{p}_{i-1}$

$\tilde{\mathbf{p}}_i = \tilde{\mathbf{z}}_{i-1} + \beta_{i-1} \tilde{\mathbf{p}}_{i-1}$

 fim se

$\mathbf{q}_i = \mathbf{A} \mathbf{p}_i$

$\tilde{\mathbf{q}}_i = \mathbf{A}^T \tilde{\mathbf{p}}_i$

$\alpha_i = \rho_{i-1} / \tilde{\mathbf{p}}_i^T \mathbf{q}_i$

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \tilde{\mathbf{p}}$

$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i \mathbf{q}_i$

$\tilde{\mathbf{r}}_i = \tilde{\mathbf{r}}_{i-1} - \alpha_i \tilde{\mathbf{q}}_i$

 se $\|\mathbf{r}_i\|_2 \leq \varepsilon$, \mathbf{x}_i é a solução

 senão, continua

fim

Fig. 3.3 - Algoritmo do Método Gradiente Biconjugado.

3.5.3 Método Gradiente Biconjugado Estabilizado (BiCGStab)

Em algumas aplicações o produto com a matriz transposta pode não estar disponível, inviabilizando o método Gradiente Biconjugado, que então pode ser substituído por métodos como o Gradiente BiConjugado Estabilizado (BiCGStab). O algoritmo do BiCGStab utiliza um parâmetro ω_i escolhido com o objetivo de minimizar o resíduo r_i segundo a norma-2, tal como encontrado no método do gradiente descendente (*steepest descent*) [39][51]. Com isso se consegue uma convergência mais suave [51], embora sejam ainda relatados casos onde a convergência é interrompida [52].

O BiCGStab requer dois produtos matriz por vetor, cinco operações de atualização e quatro produtos internos a cada iteração, quer dizer, dois produtos internos a mais que o BiCG. Em alguns casos o número de iterações é menor para o BiCGStab, o que faz com que nestes casos o BiCGStab seja mais eficiente, apesar de possuir mais produtos internos. Além disso, o método não precisa o armazenamento da transposta da matriz de coeficientes.

Neste algoritmo, além dos vetores e escalares já mencionados anteriormente, são necessários o vetor de resíduo atualizado s , o vetor de resíduo A conjugado t , e os parâmetros escalares ω_i e ω_{i-1} .

O algoritmo para o método Gradiente Biconjugado Estabilizado como encontrado em [48] pode ser visto na figura 3.4.

Algoritmo para o Gradiente Biconjugado Estabilizado

Escolhe uma estimativa inicial \mathbf{x}_0 .

Calcula-se $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$

Escolhe-se $\tilde{\mathbf{r}} = \mathbf{r}_0$

para $i=1, 2, \dots$

$$\rho_{i-1} = \tilde{\mathbf{r}}^T \mathbf{r}_{i-1}$$

se $\rho_{i-1} = 0$ o método falha

se $i=1$

$$\mathbf{p}_{i-1} = \mathbf{r}_{i-1}$$

Senão

$$\beta_{i-1} = (\rho_{i-1} \rho_{i-2}) / (\alpha_{i-1} \omega_{i-1})$$

$$\mathbf{p}_{i-1} = \mathbf{r}_{i-1} + \beta_{i-1} (\mathbf{p}_{i-1} - \omega_{i-1} \mathbf{v}_{i-1})$$

fim se

resolve $\mathbf{M} \tilde{\mathbf{p}} = \mathbf{p}_i$

$$\mathbf{v}_i = \mathbf{A} \tilde{\mathbf{p}}$$

$$\alpha_i = \rho_{i-1} / \tilde{\mathbf{r}}^T \mathbf{v}_i$$

$$\mathbf{s} = \mathbf{r}_{i-1} - \alpha_i \mathbf{v}_i$$

se $\|\mathbf{s}\|_2 \leq \varepsilon$

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \tilde{\mathbf{p}}$$

pára-se o processo.

fim se

resolve $\mathbf{M} \tilde{\mathbf{s}} = \mathbf{s}$

$$\mathbf{t} = \mathbf{A} \tilde{\mathbf{s}}$$

$$\omega_i = \mathbf{t}^T \mathbf{s} / \mathbf{t}^T \mathbf{t}$$

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \tilde{\mathbf{p}} + \omega_i \tilde{\mathbf{s}}$$

$$\mathbf{r}_i = \mathbf{s} + \omega_i \mathbf{t}$$

se $\|\mathbf{r}_i\|_2 \leq \varepsilon$, \mathbf{x}_i é a solução

senão, continua, mas é necessário

que $\omega_i \neq 0$

fim

Fig. 3.4 – Algoritmo do Método Gradiente Biconjugado Estabilizado.

3.5.4 Método Gradiente Conjugado Quadrado (CGS)

Observando a relação 3.31, percebe-se que o vetor de resíduos pode ser escrito em função de um polinômio em A , tal que:

$$\mathbf{r}_j = \mathbf{P}_j(\mathbf{A})\mathbf{r}_0 \quad (3.53)$$

Sonneveld observou que o algoritmo do método BiCG, atualizando seus resíduos pelas equações 3.47 e 3.48, equivalia a aplicar estes polinômios duas vezes, uma para o resíduo e outra vez para o resíduo conjugado. Isto é, além de 3.53 também aplicava sobre 3.54:

$$\tilde{\mathbf{r}}_j = \mathbf{P}_j(\mathbf{A}^T)\tilde{\mathbf{r}}_0 \quad (3.54)$$

Assim, desconsiderando momentaneamente a aplicação do pré-condicionador \mathbf{M} no algoritmo para o BiCG, pode-se escrever que o parâmetro ρ_i , dependendo de \mathbf{r}_i e $\tilde{\mathbf{r}}_i$, fica da seguinte forma:

$$\rho_i = \langle \tilde{\mathbf{r}}_i, \mathbf{r}_i \rangle = \langle \mathbf{P}_i(\mathbf{A}^T)\tilde{\mathbf{r}}_0, \mathbf{P}_i(\mathbf{A})\mathbf{r}_0 \rangle = \langle \tilde{\mathbf{r}}_0, \mathbf{P}_i^2(\mathbf{A})\mathbf{r}_0 \rangle \quad (3.55)$$

Isto sugere que se $\mathbf{P}_i(\mathbf{A})$ reduz¹ \mathbf{r}_0 a um vetor menor \mathbf{r}_i , então pode ser vantajoso aplicar o fator de contração duas vezes, calculando $\mathbf{P}_i^2(\mathbf{A})\mathbf{r}_0$. Estas relações também são aplicadas aos vetores de direção \mathbf{p}_i e $\tilde{\mathbf{p}}_i$, tal que as relações envolvendo as atualizações dos resíduos e das direções de procura são modificadas em relação ao BiCG. Uma explanação com as relações algébricas introduzindo o fator de contração nas equações de recorrência pode ser encontrada em [39].

Neste método são necessárias duas multiplicações matriz por vetor a cada iteração, dois produtos internos e seis atualizações de vetores, mas não é usada a matriz transposta \mathbf{A}^T . Observando a relação de contração ao quadrado, poder-se-ia esperar que

o método convergisse mais rapidamente. Entretanto, o fator de contração ao quadrado influencia também os erros de arredondamento, como mostrado pela equação 3.32, fazendo com que a convergência do método seja altamente irregular.

Neste algoritmo, além dos vetores e escalares comuns a todos os métodos já mencionados, também são encontrados o vetor auxiliar de direção \mathbf{u}_i e os vetores de direção A -conjugados \mathbf{q} e $\tilde{\mathbf{q}}$.

O algoritmo para o método Gradiente Conjugado Quadrado como encontrado em [48] pode ser visto na figura 3.5.

Algoritmo para o Gradiente Conjugado Quadrado

Escolhe uma estimativa inicial \mathbf{x}_0 .

Calcula-se $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$

para $i=1, 2, \dots$

$$\rho_{i-1} = \tilde{\mathbf{r}}^T \mathbf{r}_{i-1}$$

se $\rho_{i-1} = 0$ o método falha

se $i=1$

$$\mathbf{u}_1 = \mathbf{r}_0$$

$$\mathbf{p}_1 = \mathbf{u}_1$$

Senão

$$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$$

$$\mathbf{u}_i = \mathbf{r}_{i-1} + \beta_{i-1} \mathbf{q}_{i-1}$$

$$\mathbf{p}_i = \mathbf{u}_i + \beta_{i-1} (\mathbf{q}_{i-1} + \beta_{i-1} \mathbf{p}_{i-1})$$

fim se

resolve $\mathbf{M} \tilde{\mathbf{p}} = \mathbf{p}_i$

$$\mathbf{v} = \mathbf{A} \tilde{\mathbf{p}}$$

$$\alpha_i = \rho_{i-1} / \tilde{\mathbf{r}}^T \mathbf{v}$$

$$\mathbf{q}_i = \mathbf{u}_i - \alpha_i \tilde{\mathbf{v}}$$

resolve $\mathbf{M} \tilde{\mathbf{u}} = \mathbf{u}_i + \mathbf{q}_i$

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \tilde{\mathbf{u}}$$

$$\tilde{\mathbf{q}} = \mathbf{A} \tilde{\mathbf{u}}$$

$$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i \tilde{\mathbf{q}}$$

se $\|\mathbf{r}_i\|_2 \leq \varepsilon$, \mathbf{x}_i é a solução

senão, continua

fim

Fig. 3.5 – Algoritmo do Método Gradiente Conjugado Quadrado.

¹ Como sugerido na expressão 3.32.

3.6 Conclusão

Este capítulo apresentou, a partir da iteração básica, o subespaço de Krylov. Vários métodos de resolução iterativa de sistemas lineares geram aproximações para um subespaço de Krylov, então sua importância, como pôde ser observado no item onde as abordagens a este subespaço foram mostradas. Também foram apresentadas algumas explicações e características computacionais dos métodos escolhidos para a realização dos testes de paralelização.

O próximo capítulo mostrará a introdução da matriz de pré-condicionamento nestes algoritmos. Será mostrada a técnica com a qual os algoritmos são modificados para a inserção da matriz M e a forma como o pré-condicionamento atua sobre o erro a cada iteração, melhorando a convergência.

Também será apresentada a variação da decomposição incompleta LU proposta por este trabalho, como pré-condicionador aplicado à arquitetura de memória distribuída encontrada nos clusters de computadores.

4 Pré-condicionamento

4.1 Introdução

O uso de pré-condicionadores é fundamental nos métodos iterativos não estacionários. Através do seu emprego se consegue uma convergência mais rápida. O pré-condicionamento não é aplicado diretamente sobre a matriz de coeficientes, mas aplicado através de uma relação envolvendo o resíduo da iteração.

Existem vários tipos de pré-condicionadores descritos na literatura. Este capítulo descreve a forma como os pré-condicionadores são inseridos nos algoritmos dos subespaços de Krylov e apresenta brevemente as classes de pré-condicionadores, atendo-se ao pré-condicionador LU por blocos que representa parte das contribuições inovadoras desta tese.

4.2 A inserção do pré-condicionamento nos métodos iterativos

A idéia de pré-condicionamento tem como objetivo a redução do número de condição do sistema, produzindo uma convergência mais rápida do processo iterativo. Como encontrado em [17], o pré-condicionamento de uma matriz consiste na substituição do sistema (3.1):

$$A \mathbf{x} = \mathbf{b}$$

pelo sistema modificado:

$$M^{-1}A \mathbf{x} = M^{-1}\mathbf{b} \tag{4.1}$$

que consiste num pré-condicionamento à esquerda, ou pelo sistema

$$A M^{-1} \tilde{x} = b \quad (4.2)$$

com $x = M^{-1} \tilde{x}$, que consiste num pré-condicionamento à direita.

A matriz M^{-1} é escolhida com o objetivo de melhorar o condicionamento do sistema original. Na teoria o melhor pré-condicionador é a escolha de $M^{-1} = A^{-1}$ já que o número de condição do sistema é 1. Entretanto, o cálculo da inversa de A é equivalente à resolução do sistema, além de representar um custo computacional muito alto. Desta forma calcula-se uma matriz M^{-1} próxima de A^{-1} e, cujo custo computacional seja pequeno.

No caso do CG, o algoritmo não pode ser aplicado diretamente sobre o sistema $M^{-1} A x = M^{-1} b$, pois mesmo que M^{-1} e A sejam simétricos, $M^{-1} A$ não é necessariamente simétrica [48, 53].

A forma na qual uma matriz de pré-condicionamento se aproxima da identidade também depende do algoritmo usado. Para o método da iteração simples deveria ser como $\rho(I - M^{-1} A) < 1$ (raio espectral < 1) para obter convergência rápida, ou $\|I - M^{-1} A\| < 1$ para o método ter redução rápida do erro.

Esta redução pode ser demonstrada escrevendo-se iterativamente a expressão $A x = b$, quer dizer, fazendo-se a separação da matriz A de acordo com a expressão (4.3) a seguir e desenvolvendo o processo iterativo. Assim, dado o sistema $A x = b$, separa-se a matriz A de acordo com o seguinte procedimento:

$$A = M - N \quad (4.3)$$

$$(M - N) x = b \quad (4.4)$$

$$M x - N x = b \quad (4.5)$$

$$M x = N x + b \quad (4.6)$$

Considerando aproximações, dado x_{k-1} , uma aproximação para x_k pode ser dada

por:

$$\mathbf{M} \mathbf{x}_k = \mathbf{N} \mathbf{x}_{k-1} + \mathbf{b} \quad (4.7)$$

Multiplicando a expressão por \mathbf{M}^{-1} , tem-se:

$$\mathbf{M}^{-1} \mathbf{M} \mathbf{x}_k = \mathbf{M}^{-1} \mathbf{N} \mathbf{x}_{k-1} + \mathbf{M}^{-1} \mathbf{b} \quad (4.8)$$

$$\mathbf{x}_k = \mathbf{M}^{-1} \mathbf{N} \mathbf{x}_{k-1} + \mathbf{M}^{-1} \mathbf{b} \quad (4.9)$$

e, multiplicando ambos os lados da expressão (4.3) por \mathbf{M}^{-1} , temos:

$$\mathbf{M}^{-1} \mathbf{A} = \mathbf{M}^{-1} \mathbf{M} - \mathbf{M}^{-1} \mathbf{N} \quad (4.10)$$

$$\mathbf{M}^{-1} \mathbf{A} = \mathbf{I} - \mathbf{M}^{-1} \mathbf{N} \quad (4.11)$$

$$\mathbf{M}^{-1} \mathbf{N} = \mathbf{I} - \mathbf{M}^{-1} \mathbf{A} \quad (4.12)$$

A expressão (4.9) pode ser reescrita de forma simplificada como:

$$\mathbf{x}_k = \mathbf{B} \mathbf{x}_{k-1} + \mathbf{z} \quad (4.13)$$

com:

$$\mathbf{B} = \mathbf{M}^{-1} \mathbf{N} \quad (4.14)$$

$$\mathbf{z} = \mathbf{M}^{-1} \mathbf{b} \quad (4.15)$$

O erro na k -ésima iteração pode ser escrito como:

$$\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}} \quad (4.16)$$

$$\mathbf{x}_k = \mathbf{e}_k + \hat{\mathbf{x}} \quad (4.17)$$

onde $\hat{\mathbf{x}}$ é a solução do sistema.

Desenvolvendo a expressão (4.17) para a k -ésima -1 iteração ($\mathbf{x}_{k-1} = \mathbf{e}_{k-1} + \hat{\mathbf{x}}$) e aplicando na expressão (4.13), temos:

$$\mathbf{x}_k = \mathbf{B}(\mathbf{e}_{k-1} + \hat{\mathbf{x}}) + \mathbf{z} \quad (4.18)$$

$$\mathbf{x}_k = \mathbf{B} \hat{\mathbf{x}} + \mathbf{z} + \mathbf{B} \mathbf{e}_{k-1} \quad (4.19)$$

$$\therefore \hat{\mathbf{x}} = \mathbf{B} \hat{\mathbf{x}} + \mathbf{z} \quad (4.20)$$

$$\mathbf{x}_k = \hat{\mathbf{x}} + \mathbf{B} \mathbf{e}_{k-1} \quad (4.21)$$

$$\mathbf{x}_k - \hat{\mathbf{x}} = \mathbf{B} \mathbf{e}_{k-1} \quad (4.22)$$

$$\mathbf{e}_k = \mathbf{B} \mathbf{e}_{k-1} \quad (4.23)$$

que, em função da relação (4.13), fica:

$$\mathbf{e}_k = (\mathbf{M}^{-1}\mathbf{N})\mathbf{e}_{k-1} \quad (4.24)$$

que, por sua vez, devido à relação (4.12) se torna:

$$\mathbf{e}_k = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{e}_{k-1} \quad (4.25)$$

Das equações (4.23) e (4.25) se percebe que a matriz $\mathbf{B} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ afeta o ERRO em cada iteração, não afetando a parte correta da resposta, como observado na equação (4.21). Se \mathbf{B} possuir um raio espectral $\rho(\mathbf{B}) < 1$, a cada iteração o erro irá diminuir e o método convergirá, pois a matriz \mathbf{B} aplica um fator de “contração” sobre o erro.

Como descrito anteriormente, quando o algoritmo não pode ser aplicado diretamente sobre o sistema $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$, pois mesmo que \mathbf{M}^{-1} e \mathbf{A} sejam simétricos, $\mathbf{M}^{-1}\mathbf{A}$ não é necessariamente simétrica, faz-se uma modificação para a aplicação do pré-condicionamento no sistema. A modificação é feita como segue, a partir da equação (3.11) do resíduo. Assim, multiplicando o resíduo $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ em ambos os lados por \mathbf{M}^{-1} , tem-se:

$$\mathbf{M}^{-1}\mathbf{r}_k = \mathbf{M}^{-1}\mathbf{b} - \mathbf{M}^{-1}\mathbf{A}\mathbf{x}_k \quad (4.26)$$

$$\mathbf{M}^{-1}\mathbf{r}_k = \mathbf{M}^{-1}\mathbf{b} - \mathbf{M}^{-1}(\mathbf{M} - \mathbf{N})\mathbf{x}_k \quad (4.27)$$

$$\mathbf{M}^{-1}\mathbf{r}_k = \mathbf{M}^{-1}\mathbf{b} - \mathbf{M}^{-1}\mathbf{M}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k \quad (4.28)$$

$$\mathbf{M}^{-1}\mathbf{r}_k = -\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b} \quad (4.29)$$

$$\mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b} = \mathbf{M}^{-1}\mathbf{r}_k + \mathbf{x}_k \quad (4.30)$$

Sabendo que o lado esquerdo da equação (4.30) equivale a \mathbf{x}_k pela relação (4.9), e considerando o passo anterior como $k-1$, encontra-se a seguinte equação:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{M}^{-1}\mathbf{r}_{k-1} \quad (4.31)$$

ou

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{z}_{k-1} \quad (4.32)$$

com:

$$\mathbf{z}_{k-1} = \mathbf{M}^{-1}\mathbf{r}_{k-1} \quad (4.33)$$

ou

$$\mathbf{M}\mathbf{z}_{k-1} = \mathbf{M}\mathbf{M}^{-1}\mathbf{r}_{k-1} \quad (4.34)$$

$$\mathbf{M}\mathbf{z}_{k-1} = \mathbf{r}_{k-1} \quad (4.35)$$

Então, o sistema (4.35) é o sistema a ser resolvido nos algoritmos que utilizam o pré-condicionamento, como quase todos os métodos derivados do CG. A inserção desta relação pode ser observada nas figuras 3.2, 3.3, 3.4 e 3.5. Observe-se que a relação (4.35) não é aplicada diretamente sobre a matriz de coeficientes, mas sim aplicando uma transformação linear sobre o vetor de resíduo. Além disso, a relação sugere a resolução de um sistema de equações, o que parece remeter novamente à relação (3.1) e, num primeiro momento pode parecer não fazer sentido resolver o sistema repetindo várias vezes a solução. Mas a ênfase está na solução aproximada, isto é, só precisamos de uma aproximação rápida, mesmo que seja uma aproximação muito pobre.

O pré-condicionamento mais simples é feito com a diagonal da matriz de coeficientes ($\mathbf{M} = \mathbf{D} = \text{diag}(\mathbf{A})$) conhecido como pré-condicionamento de Jacobi, mas infelizmente esta aproximação não reduz o número de condição o suficiente para ser útil. Como caso oposto ao pré-condicionamento diagonal estão algumas decomposições que tendem a destruir a esparsidade da matriz \mathbf{A} , o que nos leva a uma escolha crítica entre a efetividade do pré-condicionador e a qualidade da aproximação com a matriz de

coeficientes.

4.3 As classes de pré-condicionadores

Partindo do princípio de que qualquer aproximação da matriz A pode ser usada como um pré-condicionador, qualquer método de resolução de sistemas lineares pode ser usado com esta finalidade. Em função da natureza destes métodos, à grosso modo, podemos dividi-los em três categorias [17]:

- pré-condicionadores projetados para uma classe geral de matrizes, isto é, matrizes sem elementos diagonais nulos, matrizes positivas definidas, M -matrizes. Nesta classe geral encontram-se os pré-condicionadores baseados nos próprios métodos iterativos como Gauss-Jacobi, Gauss-Seidel, pré-condicionadores SOR, Cholesky Incompleto, e outras decomposições incompletas.
- pré-condicionadores projetados para uma classe de problemas especiais, onde a origem do problema sugere um tipo particular de pré-condicionador. Nesta classe encontram-se os pré-condicionadores Multigrid e de decomposição do domínio.
- pré-condicionadores projetados para problemas específicos, como os trabalhos que tratam de equações de transporte e incompressibilidade de fluidos.

Os pré-condicionadores da classe geral são normalmente usados por pacotes de software onde, a princípio, não se conhece o problema a ser tratado. A maioria destes pré-condicionadores requer o conhecimento de pelo menos algumas características da matriz de coeficientes, visto que são poucos os teoremas que quantificam a qualidade de um pré-condicionador. Pode-se, por exemplo, calcular o número de condição da matriz pré-condicionada e compará-la ao sistema original, observando em quanto este último foi diminuído.

Entre os pré-condicionadores da classe geral destacam-se os pré-condicionadores construídos a partir de decomposições incompletas. O pré-condicionador desenvolvido neste trabalho se encontra nesta categoria.

A seguir são descritos algumas características importantes deste conjunto de pré-condicionadores, e em seguida são feitas explanações sobre o pré-condicionador proposto.

4.3.1 Pré-condicionadores baseados em Decomposições Incompletas.

Uma larga classe de pré-condicionadores é baseada nas decomposições incompletas da matriz de coeficientes. Uma fatoração é dita incompleta se durante o processo de fatoração certos elementos de enchimento, elementos diferentes de zero na fatoração nas posições onde a matriz original tem um zero, são ignorados. Um pré-condicionador deste tipo é dado na forma $M = LU$, com L tipo triangular inferior e U triangular superior. A eficiência do pré-condicionador depende de quão bem M^{-1} se aproxima de A^{-1} .

As decomposições incompletas são os primeiros pré-condicionadores encontrados para os quais existe um estágio não trivial de criação. As decomposições incompletas podem sofrer erro na decomposição (*break-down* – ocasionado pela divisão por um pivô nulo) ou resultar em matrizes indefinidas (um pivô negativo) mesmo quando a decomposição completa da mesma matriz existe e leva a uma matriz positiva definida.

Uma decomposição incompleta existe para muitas estratégias de fatoração se a matriz original é uma M -matriz [12]. Nos casos onde os pivôs são nulos, algumas estratégias foram propostas com a substituição por um número positivo [54], ou reiniciando a fatoração de $A + \alpha I$ para algum valor positivo de α [55].

Uma importante consideração para os pré-condicionadores baseados em decomposições incompletas é o custo do processo de decomposição. Mesmo quando a decomposição incompleta existe, o número de operações para criar a matriz de pré-condicionamento é no mínimo tão grande quanto para resolver um sistema com a matriz de coeficientes, tal que o custo pode ser igual a uma ou mais iterações do método iterativo. Em computadores paralelos este problema é agravado geralmente pela pequena eficiência do cálculo paralelo das fatorações. Estes custos de fatoração podem ser amortizados se o método iterativo leva muitas iterações para convergir ou se

o mesmo pré-condicionador é usado em vários sistemas lineares.

4.3.2 Decomposições incompletas convencionais

O tipo mais comum de fatoração incompleta é baseado num conjunto \mathcal{S} , de posições da matriz, mantendo todas as posições fora deste conjunto igual a zero durante a fatoração. A fatoração resultante é incompleta no sentido de que não se tem enchimento.

O conjunto \mathcal{S} , é escolhido tal que contenha todas as posições (i,j) para os quais $a(i,j) \neq 0$. Uma posição que é zero em A mas não em uma fatoração exata é chamada de posição de enchimento, e se ela está fora de \mathcal{S} , o enchimento é descartado. Frequentemente \mathcal{S} é escolhido para coincidir com todas as posições diferentes de zero em A , descartando todo o enchimento. Uma fatoração deste tipo é chamada de $ILU(0)$: fatoração LU incompleta de nível zero.

Uma fatoração incompleta pode ser formalmente escrita como:

$$\text{Para cada } k, i, j > k: a_{ij} \leftarrow \begin{cases} a_{ij} - a_{i,k} a_{k,k}^{-1} a_{k,j} & \text{se } (i,j) \in \mathcal{S} \\ a_{ij} & \text{para outros casos} \end{cases} \quad (4.36)$$

MEIJERINK e VORST [12] provaram que, se A é uma M -matriz, então a fatoração existe para qualquer escolha de \mathcal{S} , e resulta uma matriz simétrica positiva definida se A é simétrica positiva definida.

Uma descrição bem detalhada a respeito de estratégias e níveis de enchimento pode ser encontrada em [39].

4.3.3 Métodos de Decomposição por blocos.

Também podem ser consideradas as variantes por blocos dos pré-condicionadores para métodos acelerados. O ponto de partida para uma decomposição incompleta por bloco é o particionamento da matriz. Assim, uma decomposição incompleta é realizada

usando os blocos da matriz como entidades básicas.

Na literatura são encontrados dois tipos de algoritmos classificados como algoritmos de decomposição por blocos: aqueles que precisam da inversão dos blocos pivôs [39] e aqueles que tratam uma submatriz de A como um bloco [40] destinando-a a um determinado processador.

Ambas as abordagens são utilizadas no cálculo paralelizado de pré-condicionadores. O pré-condicionador utilizado neste trabalho pertence ao segundo tipo nesta classificação.

4.4 O Pré-condicionador com fatoração LU por blocos com interseção (FLUI).

A decomposição LU pode ser realizada por eliminação de Gauss, Crout ou Doolittle [56], mas neste trabalho foi utilizada a técnica *bordering* [57], que faz uso de produtos matriz por vetor. Foi utilizada esta abordagem na decomposição paralela, pois o cálculo da matriz de pré-condicionamento por blocos é feito de forma independente em cada processador. A decomposição pode ser escrita na seguinte forma:

$$A = \begin{bmatrix} M & r \\ v^T & s \end{bmatrix} = \begin{bmatrix} L & 0 \\ \ell^T & t \end{bmatrix} \cdot \begin{bmatrix} U & u \\ 0 & 1 \end{bmatrix} \quad (4.37)$$

onde v^T e ℓ^T são vetores linha e r e u são vetores coluna. Do sistema desenvolvem-se as seguintes expressões:

$M = LU$: como é desejada uma matriz triangular superior com diagonal principal unitária, inicialmente temos: $U = 1$ e $L = M$.

$r = Lu$: Sistema com substituição progressiva do tipo $Lx = b$.

$v^T = \ell^T U$: Pode-se escrever este sistema tal que: $v = U^T \ell$, que representa um sistema com substituição progressiva também.

$s = \ell^T u + t$: reescrevendo “t” tal que seja a variável a ser calculada tem-se:

$t = s - \ell^T \mathbf{u}$, um produto entre dois vetores da nova iteração e recém calculados em $\mathbf{r} = \mathbf{L} \mathbf{u}$ e $\mathbf{U}^T \ell = \mathbf{v}$.

Com isso é preciso solucionar dois sistemas, um com substituição progressiva e outro com substituição regressiva, além de um produto escalar. O cálculo dos elementos u_{ij} e l_{ij} é realizado acessando elementos existentes em linhas e colunas diferentes (Fig. 4.1). Se estas linhas e colunas também se encontrarem em processadores diferentes, serão necessárias passagens de mensagens (Fig. 4.2).

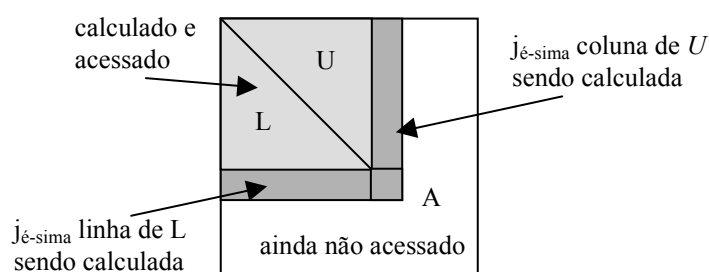


Fig. 4.1 – Elementos a serem calculados no j -ésimo passo de cálculo.

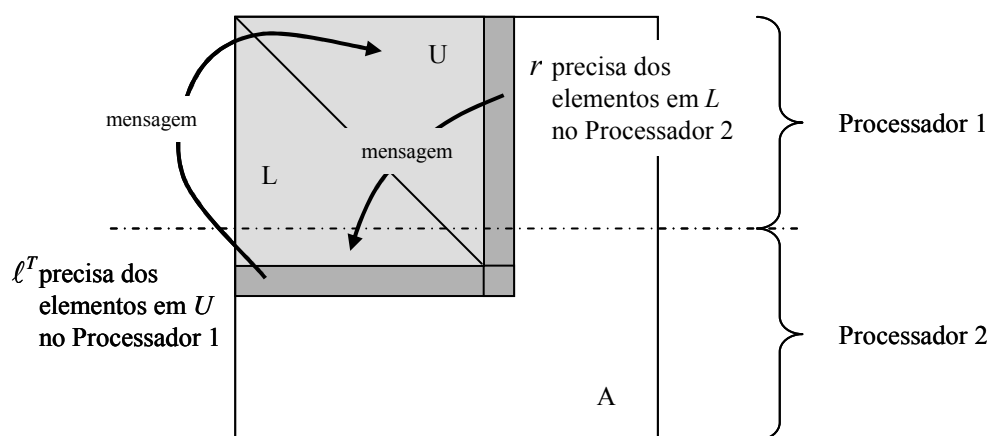


Fig. 4.2 – Passagens de mensagens para o t -ésimo passo de cálculo.

Para evitar estas passagens de mensagens, [40] propôs para um pré-condicionador $\mathbf{L}\mathbf{L}^T$, construir a matriz de pré-condicionamento de forma independente em cada processador, a partir da divisão da matriz de coeficientes de acordo com o número de processadores disponíveis. Assim, cada processador elabora uma matriz de pré-condicionamento desacoplada das outras partes; isto, porém, produz um

enfraquecimento do pré-condicionador. Deve-se lembrar, entretanto, que este pré-condicionamento é baseado numa fatoração incompleta, uma forma que já sofreu um enfraquecimento.

Para diminuir o enfraquecimento causado por este desacoplamento, propôs-se decompor as parcelas destinadas a cada processador com um número a mais de elementos (ou linhas) (Fig. 4.3), determinando um parâmetro de interseção entre os blocos. Assim, em cada processador é construída uma matriz parcial com um número maior de elementos (ou linhas) do que aquele obtido por uma simples quebra de elementos por máquina. Quando o valor de interseção for igual a 0%, temos uma configuração próxima daquela relatada em [40]. Próxima, pois a referida referência trata unicamente de matrizes e pré-condicionamento simétricos.

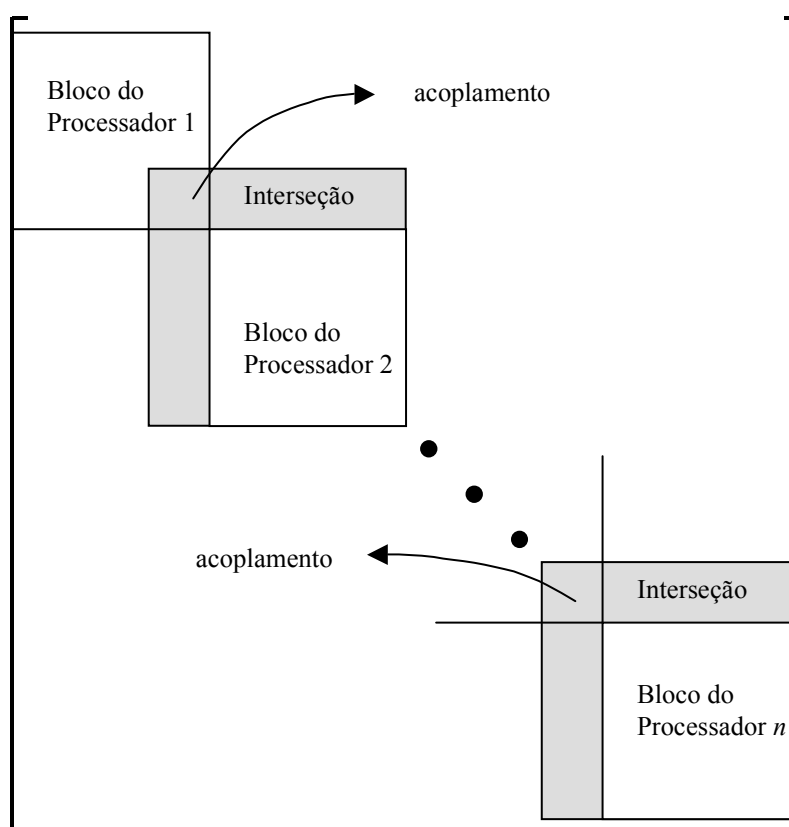


Fig. 4.3 – Pré-condicionamento proposto.

A partir de um valor de porcentagem as parcelas da matriz A destinadas aos processadores com identificador (*rank*) maior que zero passam a receber um número maior de linhas, pertencentes ao bloco anterior, com as quais é realizada a fatoração

LU . Esta Fatoração LU por blocos com Interseção é identificada como FLUI no código desenvolvido para sua implementação. A Fig. 4.4 mostra um exemplo onde, após a divisão da matriz entre dois processadores, faz-se a fatoração LU do segundo processador a partir da interseção mostrada num tom mais claro de cinza. A fatoração utiliza elementos existentes também no primeiro processador, mas alocados na segunda máquina.

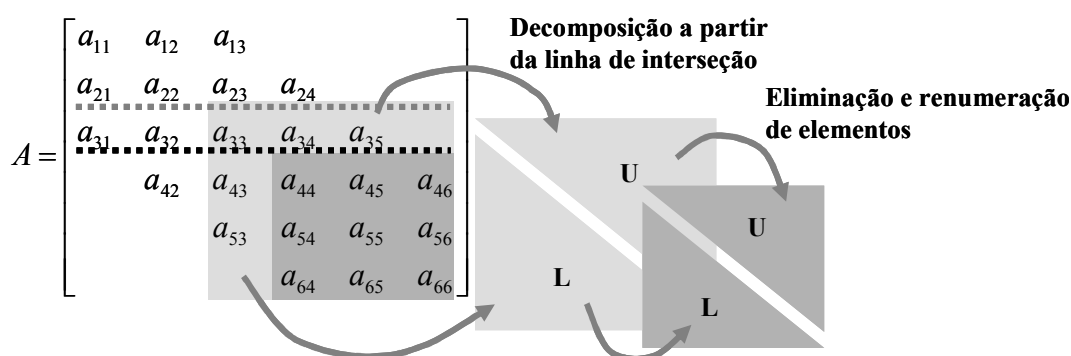


Fig. 4.4 - Obtenção da matriz de pré-condicionamento.

Após a realização desta etapa, eliminam-se os elementos a mais que a matriz de pré-condicionamento possui (os elementos em posições de acoplamento e interseção) resultando a matriz parcial de pré-condicionamento daquele processador, mostrada na Fig. 4.4 num tom mais escuro de cinza.

Além disso, para obter a parcela da matriz A , usada no segundo processador, que deve ser usada nos produtos matriz por vetor, é necessário também eliminar os elementos de acoplamento, mantendo os elementos de interseção. Estes elementos podem ser observados na Fig. 4.3.

Este pré-condicionamento é construído sem passagem de mensagens, com uma melhor utilização dos processadores. Cada processador pode calcular de forma independente sua parcela do vetor resíduo, etapa esta realizada inteiramente em paralelo. O agrupamento dos resultados é feito por passagens de mensagens.

A figura 4.5 ilustra ideograficamente a estrutura final resultante da aplicação deste método de pré-condicionamento a uma matriz do tipo banda. Nesta figura se considera a existência de uma matriz com meias largas de banda idênticas e

constantes. As regiões mostradas em cinza escuro representam os blocos de pré-condicionadores parciais e, a parte em cinza mais claro, os elementos originais da matriz de coeficientes. Ressalta-se que a matriz é esparsa e as regiões mencionadas podem não estar completamente preenchidas.

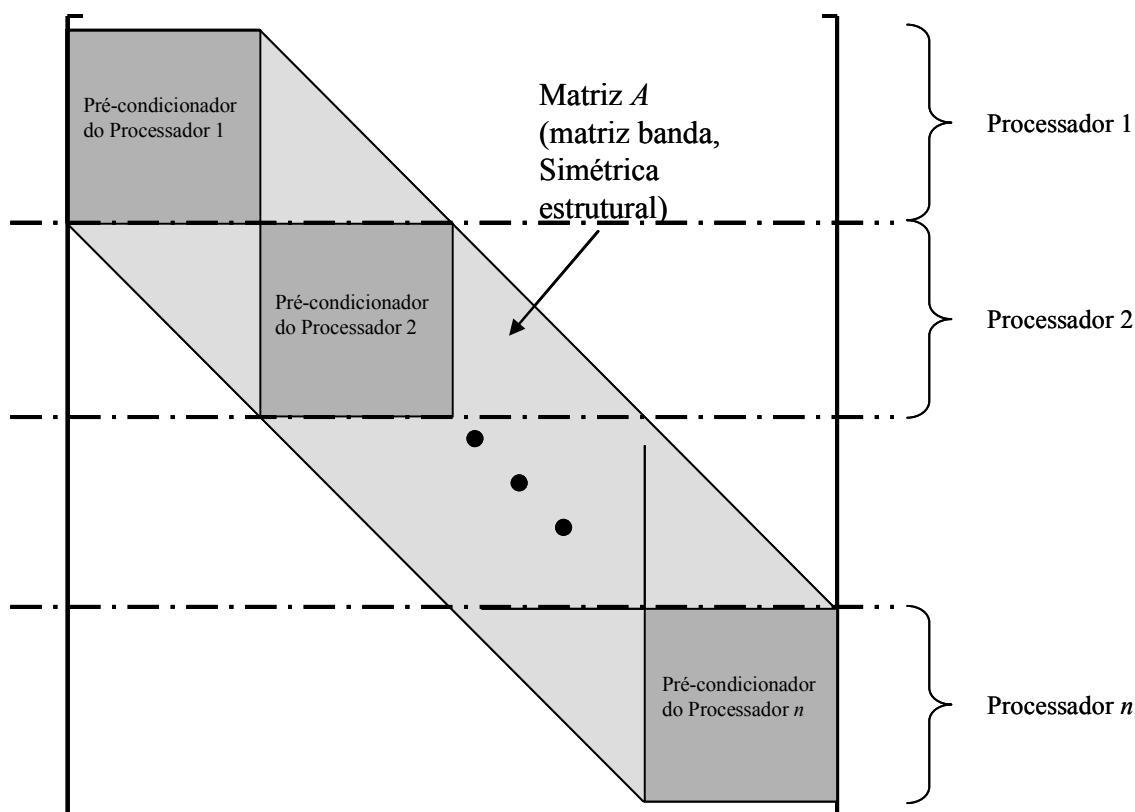


Fig. 4.5 – Estrutura final de pré-condicionamento.

Sendo a matriz esparsa, a divisão acima não corresponde realmente ao código implementado, pois a divisão depende da forma de armazenamento da matriz. Assim, a Fig. 4.6 a seguir ilustra com mais fidelidade a estrutura final de pré-condicionamento em função do método de armazenamento RCS que será descrito detalhadamente no capítulo a seguir.

Note-se que com esta forma de armazenamento um determinado processador mantém para si partes superiores e inferiores de uma matriz de pré-condicionamento, facilitando o cálculo paralelizado dos fatores de decomposição determinados na equação 4.36.

Havendo interseção, bastam primeiramente uma eliminação e posterior

renumeração dos elementos que porventura não pertencem ao conjunto de interseção para a realização da decomposição naquele bloco e, em seguida, uma eliminação para a geração da “matriz quadrada” cinza escuro que forma o bloco do pré-condicionador.

Para executar os produtos matriz por vetor, o processador precisa apenas eliminar os elementos que formam o acoplamento, e devido à forma como o armazenamento RCS constrói seus arranjos, isto é feito rapidamente “esquecendo-se” os elementos que existem entre o início da interseção e início da linha de divisão.

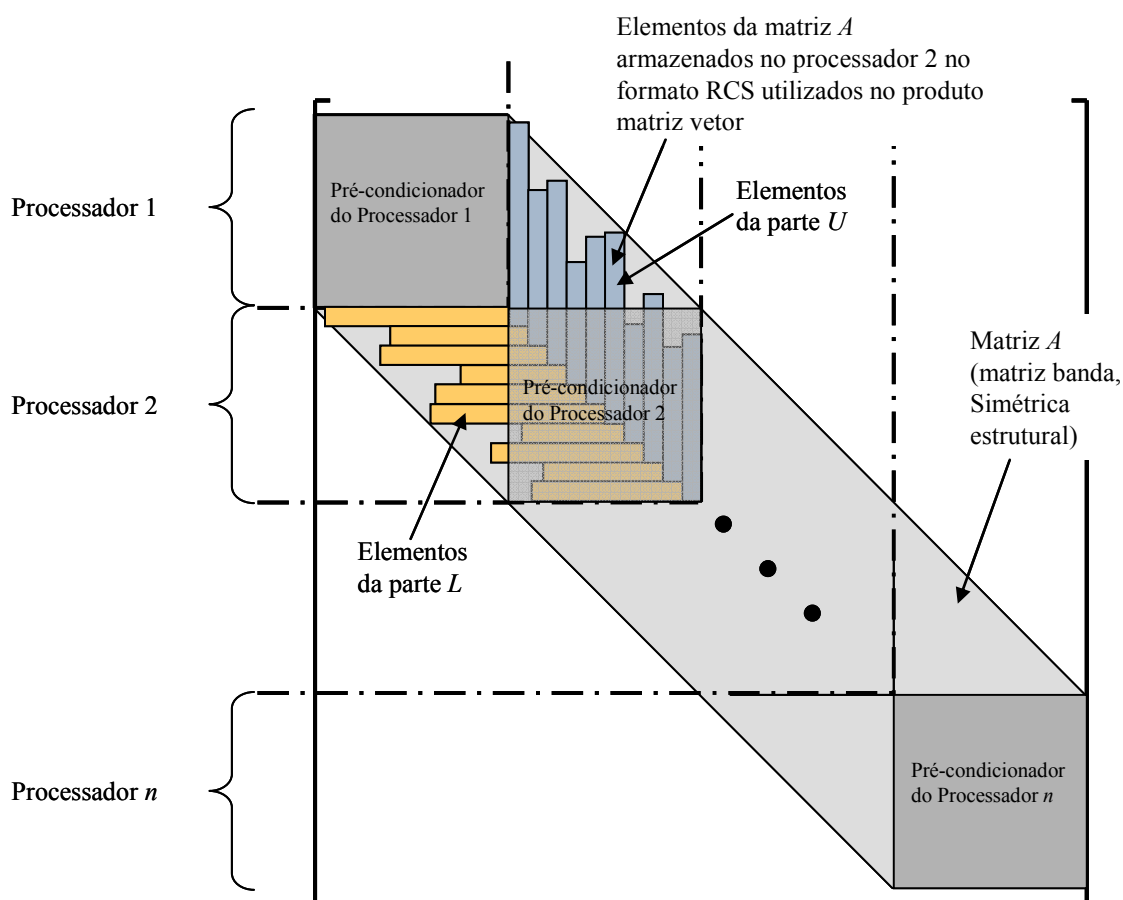


Fig. 4.6 - Estrutura final de pré-condicionamento considerando o armazenamento esparsos RCS.

4.5 Conclusão

A utilização de pré-condicionadores é fundamental nos métodos derivados dos subespaços de Krylov. Este capítulo demonstrou a inserção do pré-condicionador nos

algoritmos apresentados no capítulo 3 e descreveu com mais detalhes o pré-condicionador LU por blocos, objeto principal do presente texto. O anexo 2 traz mais informações, alguns algoritmos e exemplos referentes a este pré-condicionador.

O próximo capítulo apresenta um conjunto de conceitos associados à computação de alto desempenho, necessários à apresentação dos itens deste trabalho que se relacionam à computação em clusters de computadores.

No capítulo que se segue, também são apresentados os clusters utilizados no desenvolvimento deste trabalho e o *framework* com o qual foram obtidos os resultados com o pré-condicionador FLUI aqui desenvolvido.

O capítulo de resultados apresentará sucintamente os diversos métodos de armazenamento de matrizes esparsas, com os quais foram observadas diferenças nos desempenhos da paralelização dos algoritmos de resolução de sistemas de equações lineares.

5 Plataformas Computacionais de Alto Desempenho

5.1 Introdução

Neste capítulo serão apresentados a definição e alguns conceitos de Computação de Alto Desempenho (CAD), tópicos resumidos de arquitetura de computadores e considerações necessárias à programação paralela como a biblioteca MPI [35].

Isto porque este trabalho utiliza clusters de computadores, uma forma de computação de alto desempenho utilizando máquinas e redes de comunicação comerciais e os elementos de sua arquitetura são os mesmos de outras máquinas de alto desempenho. Este capítulo também apresenta e compara os clusters utilizados neste trabalho.

Também serão apresentados os diversos programas construídos para atender os requisitos de pesquisa desta tese. Estes programas formam um *framework* com o qual se obteve uma grande flexibilidade na obtenção de resultados. É feito um detalhamento dos núcleos de processamento encontrados nos algoritmos do capítulo 3 que podem ser paralelizados. São dados detalhes das suas implementações e da distribuição do processamento nas máquinas que formam o cluster.

5.2 Computação de alto desempenho

Uma definição para CAD deve levar em conta a existência de sistemas que

compartilham algumas características de sistemas de alto desempenho como multiprocessamento ou utilização de interconexões de rede de alta velocidade, mas que não representam arquiteturas CAD. Assim, uma definição mais restrita e que ao mesmo tempo ilustre como as técnicas de alto desempenho podem ser aplicadas, pode ser [58]:

“Computação de alto desempenho agrupa uma coleção de sistemas de hardware, ferramentas de software, linguagens e abordagens genéricas de programação, que tornam possíveis, a um preço apropriado, aplicações antes ineficazes.”

A Computação de Alto Desempenho depende de itens como a utilização das opções do compilador para a geração do código mais rápido, otimizações realizadas nas declarações de linhas de código, utilização de autoparalelização realizada pelo próprio compilador, utilização de bibliotecas de comunicação, medidas de tempo e de recursos de hardware disponíveis [59].

As opções do compilador para a geração do código mais rápido dependem de cada compilador, já que cada um tem um conjunto diferente de opções e, algumas vezes, as melhores opções só são encontradas em testes empíricos fazendo tomadas de tempo junto ao executável. As otimizações manuais na declaração das linhas de código incluem: o reordenamento de laços iterativos no acesso de vetores, a redução do esforço em chamadas a funções específicas como utilização de produto ao invés de uma exponencial; a remoção de código sem efeito como funções que nunca são chamadas e variáveis que são calculadas, mas não são usadas novamente, utilizações de funções *inline* (no caso da utilização da linguagem C++), o reordenamento na declaração de variáveis e dezenas de outros cuidados, como encontrado em [60, 61]. Deve-se ressaltar que os programas paralelos escritos para este trabalho fizeram uso de uma biblioteca de comunicação para envio de mensagens entre processadores. Desta forma, muitos dos cuidados empregados na programação seqüencial também foram empregados no código paralelo aqui utilizado.

A contagem de tempo pode ser realizada utilizando comandos do sistema operacional, com chamadas a funções de tempo ou com aplicativos que utilizam registradores específicos para este fim existentes em algumas máquinas. No presente trabalho as tomadas de tempo foram obtidas pelo comando *time* do sistema operacional

Linux. Este comando permite que o programa em teste seja lançado ao mesmo tempo em que é disparado um timer que monitora o tempo de execução do programa. Ao final do programa, o sistema operacional disponibiliza o tempo total de execução do programa, o tempo de CPU utilizado pelo código do usuário e o tempo de CPU utilizado pelo sistema. Como todos os programas testados seguem um mesmo padrão de leitura e gravação de arquivos, optou-se por utilizar como medida o tempo total de execução do programa. Este processo de teste foi automatizado através de programas e scripts, como será descrito mais adiante.

A autoparalelização pode ser usada em máquinas com memória compartilhada. É simples para usar, utilizando *flags* do compilador. Entretanto, ainda existem problemas associados tanto a este tipo de arquitetura como na implementação. Máquinas paralelas utilizam o paradigma da passagem de mensagem. A biblioteca mais usada neste paradigma atualmente é a MPI - *Message Passing Interface*.

A biblioteca MPI é um padrão da indústria, podendo ser usada tanto em sistemas com memória distribuída como os clusters Linux ou os IBM SP2, como em máquinas SMP, sendo executada na maioria dos sistemas Unix e nos sistemas Windows NT/2000/XP. Ela consiste de uma biblioteca de funções que são chamadas pelos usuários de código para passar informações entre processadores. Possui mais de cem funções, mas normalmente somente um pequeno conjunto destas funções é usado no código. A biblioteca é bastante portátil, disponível em Fortran, C e C++, sendo que pode também ser usada em um único processador para emular sistemas paralelos.

Dentre as opções citadas anteriormente, CAD é obtida, em geral, através da paralelização das atividades a serem executadas pelo programa. A paralelização, por sua vez, é uma forma eficiente de processar informações enfatizando a exploração de eventos concorrentes. Consiste na utilização de múltiplos processadores para executar partes diferentes de um mesmo programa simultaneamente, diminuindo o tempo total de processamento.

Então fica claro que uma condição necessária para a obtenção de CAD é a existência de vários elementos de processamento no sistema. A quantidade de elementos vai depender do grau de paralelismo do problema e da disponibilidade de recursos existente. Do ponto de vista de arquitetura essas máquinas podem ser classificadas de várias maneiras, mas, de uma forma rudimentar, podem ser

classificadas entre sistemas fortemente acoplados (com comunicação normalmente feita através de memória compartilhada) e sistemas fracamente acoplados (com comunicação realizada normalmente através de redes).

Pode ser percebido, então, que é possível obter (dentro de certas restrições) CAD a um baixo custo. Para tanto se usam sistemas fracamente acoplados, que possuem um custo menor de implantação. Outras vantagens de tais sistemas são: sua modularidade, que permite que o grau de paralelismo possa ser ampliado máquina a máquina, e sua maior utilização, pois sendo máquinas individuais pode-se aproveitar a capacidade de processamento de cada uma delas, mesmo que a tarefa não exija alto desempenho.

A principal desvantagem da utilização desses sistemas está ligada ao custo alto de comunicação. Essa comunicação faz com que tais sistemas tenham um desempenho bastante inferior ao dos sistemas fortemente acoplados. Apesar disso, o seu uso tem se tornado cada vez mais intenso através da popularização dos sistemas distribuídos, que nada mais são do que redes de computadores especialmente configurados para a paralelização de suas atividades e serviços. É nesta classe de sistemas que se enquadram os *clusters* Beowulf.

5.3 Conceitos de CAD

CAD envolve alguns conceitos que são muitos citados junto às especificações de hardware e software. Entre os principais conceitos estão o *speedup*, a sincronização, o *overhead*, a granulosidade, a escalabilidade, a latência e a largura de banda.

A análise de desempenho de um programa paralelo está associada ao ganho sobre a versão serial do mesmo programa. O *speedup* é o principal meio para se avaliar o ganho de desempenho de programas paralelos e é dado por:

$$S(p) = \frac{\text{tempo de execução sequencial}}{\text{tempo de execução paralela}} = \frac{t_s}{t_p} \quad (5.1)$$

Teoricamente para um programa paralelo executado por N processadores cada

processador é responsável por $1/N$ do processamento, então o *speedup* máximo será N . Entretanto, isto não ocorre na prática devido a fatores como sincronizações e *overheads*, responsáveis por acréscimos no tempo de execução paralelo.

Sincronização é a coordenação temporal de tarefas, que implica a espera pela finalização de duas ou mais tarefas em um determinado local do código (ponto de sincronismo), para continuar com a execução do programa. Um processador pode ficar parado esperando que outro processador termine a sua tarefa. Como exemplo, pode-se supor dois processadores procurando separadamente o projeto ótimo para um dado problema. Para que o melhor projeto possa ser encontrado é preciso comparar os dois resultados. Logo os dois processos precisam estar terminados. Quando um processador terminar, será necessário esperar até que o outro também termine, caso contrário o projeto ótimo não será obtido.

Overhead é o tempo utilizado para coordenar tarefas, transferir dados entre processadores, entre outros consumos de processamento que não são encontrados em programas seriais. Pode-se resumir grosseiramente *overhead* como sendo o custo da paralelização.

Granulosidade é uma medida da razão entre a quantidade de computação realizada em uma tarefa paralela e a quantidade de comunicação necessária. Esta grandeza determina se um *algoritmo* pode ou não ser utilizado na arquitetura paralela pretendida. A granulosidade pode ser grossa ou fina. Se for grossa significa que tarefas grandes podem ser processadas independentemente em paralelo. Do contrário, se for fina indica que tarefas pequenas podem ser processadas em paralelo. Se a arquitetura paralela não for suficientemente rápida (latência alta) para atender a comunicação existente entre as tarefas, um algoritmo com granulosidade fina paralelizado tomará mais tempo que sua versão serial.

Quando o termo escalabilidade é usado para indicar um projeto de sistema de hardware que permite que o mesmo possa ter seu tamanho aumentado tal que o seu desempenho também aumente, significa escalabilidade do hardware ou da arquitetura. Quando o termo é utilizado para indicar um algoritmo paralelo que pode acomodar dados com pequeno aumento nas etapas computacionais, significa escalabilidade do algoritmo. A escalabilidade da arquitetura, porém, é fortemente dependente do projeto de interconexão do sistema, quer dizer, depende muito da forma como ocorrem as

trocas de mensagens entre os diversos processadores. Esta estrutura de hardware é conhecida como Arquitetura de computadores.

Latência (*latency*) é o tempo que uma mensagem demora para atravessar um sistema [62]. Aplicando esta definição aos clusters equivale a dizer que latência é o tempo que um pacote de dados demora em ser transmitido. O tempo de latência determinado pela ferramenta NetPIPE [63], software utilizado para caracterizar as redes dos clusters usados neste trabalho, é calculado dividindo-se o tempo de *round trip*¹ por dois para pequenas mensagens (< 64 Bytes).

Largura de Banda (*bandwidth*) [62] é um termo que designa a quantidade de informação passível de ser transmitida por unidade de tempo, num determinado meio de comunicação (fio, onda radio, fibra óptica, etc.). A largura de banda é normalmente medida em bits por segundo ou em alguma outra unidade múltipla de bits. Aplicada aos clusters esta definição mostra a máxima quantidade de dados que os clusters conseguem transferir em Mbps.

Estes conceitos são importantes quando são abordados tópicos como arquitetura de computadores ou softwares utilizados para verificar o desempenho de determinado sistema. Logo a seguir será mostrado um gráfico obtido através da ferramenta NetPIPE, que pode ser usada em algumas arquiteturas, como a de clusters de computadores.

5.4 Arquitetura de computadores de alto desempenho

Antes de realizar uma descrição das máquinas utilizadas para o processamento de alto desempenho é importante conhecer alguns mecanismos utilizados para aumentar o desempenho. A estrutura do hardware ou, simplesmente, a arquitetura determina quais são as possibilidades ou impossibilidades que uma máquina tem de ir além do desempenho de uma única CPU.

Uma forma de classificar o hardware utiliza a Taxonomia de FLYNN [65] que se baseia nos modos de manipulação das instruções e dos dados, aproximando-se do estilo

de programação. Nesta classificação os dados e instruções podem ser únicos ou múltiplos, de acordo com a seguinte simbologia: S – *single*, M – *multiple*, I – *instruction*, D – *data*.

São quatro as classes arquiteturas produzidas por esta classificação:

- SISD – *single instruction, single data* – que descreve um computador comum, como um PC com um único processador, e acomoda uma única instrução executada serialmente.
- SIMD – *single instruction, multiple data* – estas máquinas têm um grande número de processadores que podem executar a mesma instrução em diferentes dados. Assim, uma única instrução pode operar muitos dados em paralelo. Uma subclasse destes sistemas são os processadores vetoriais que atuam sobre vetores de dados semelhantes.
- MISD – *multiple instruction, single data* – teoricamente neste tipo de máquina múltiplas instruções poderiam ser aplicadas sobre um determinado dado. Entretanto, máquinas comerciais com esta arquitetura ainda não foram construídas.
- MIMD – *multiple instruction, multiple data* – que descreve um sistema que distribui os dados em múltiplos processadores, realizando múltiplas operações em cada um deles. As operações podem ser diferentes programas comunicando-se entre si através de alguma forma de troca de mensagens.

Existe uma grande variedade de sistemas MIMD e, especialmente nesta classe, a taxonomia de Flynn não consegue se adequar totalmente [66]. Sistemas modernos de alto desempenho utilizam uma evolução desta divisão conhecida como SPMD – *single program, multiple data*. Neste modelo os dados são distribuídos, mas operados pelo mesmo programa.

Uma outra forma de categorizar o hardware [67] utilizado em computação de alto desempenho é a divisão dos sistemas de acordo com uma visão prática do mercado da supercomputação. Esta divisão é baseada no acesso à memória e pode ser vista na

¹ *Round Trip Time* é a medida de tempo que um pacote de dados demora a ir de um computador até outro,

figura a seguir.

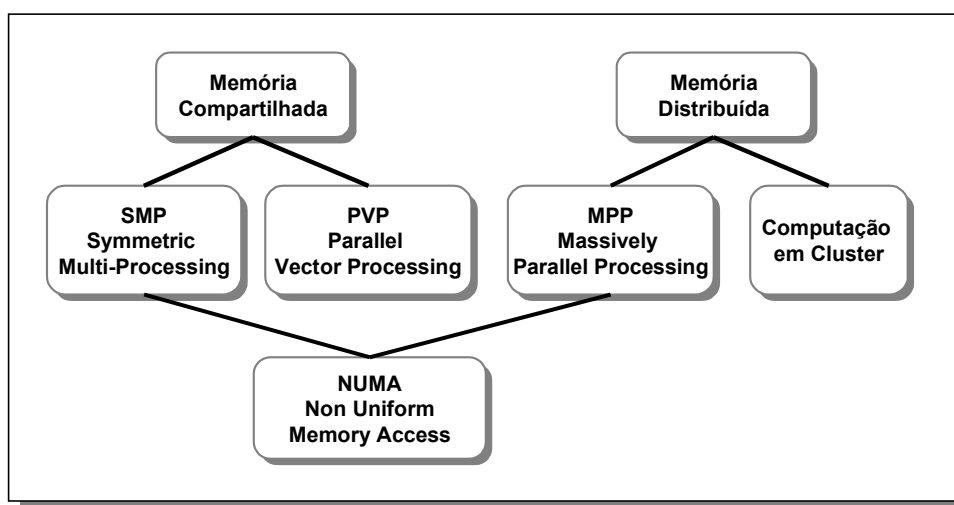


Fig. 5.1 – Categorização baseada na utilização da memória.

Num sistema onde a memória é compartilhada, todos os processadores têm acesso a uma memória comum. Esta memória comum pode ser usada para dados e resultados necessários por mais de processador. Toda a comunicação entre os processadores é feita através da memória comum. A maior vantagem de um sistema com memória compartilhada é a comunicação potencialmente rápida de dados entre os processadores. A desvantagem séria desta configuração é que diferentes processadores podem querer acessar a memória simultaneamente e, nestes casos, haverá um atraso até que a memória esteja livre. Este atraso, chamado de tempo de contenção, pode aumentar com o aumento de processadores.

Uma alternativa aos sistemas de memória compartilhada são os sistemas com memória local, onde cada processador endereça somente sua própria memória. A comunicação entre os processadores é feita através de passagem de mensagens, na qual dados ou outras informações são trocados entre processadores. Exemplos destas diferentes formas de utilização de memória são:

1) arquitetura SMP (Symmetric Multi-Processing) onde uma série de processadores idênticos são fortemente acoplados através de uma memória principal comum. A natureza paralela da máquina fica escondida do usuário e o sistema

operacional multitarefa organiza o compartilhamento do tempo e de processadores entre os programas. Algumas vezes um pequeno grupo de processadores compartilha a memória *cache* e a principal, mas isso reduz a escalabilidade para um pequeno número de processadores. Alguns processadores comuns nesta arquitetura são os processadores MIPS (1984), Sun Sparc (1985), os processadores CISC Intel Pentium (1993) e Intel Pentium Pro (1995) e RISC Dec Alpha (1998).

2) arquitetura NUMA (Non Uniform Memory Access) que é um modelo híbrido possuindo memória distribuída em cada processador, e que através de hardware e software cria uma imagem de memória compartilhada em nível de usuário. Alguns processadores construídos com essa tecnologia são: HP-Convex Exemplar (1994), Cray T3E (1995) e SGI Origin2000 (1998).

3) arquitetura PVP (Parallel Vector Processing) que são máquinas construídas com processadores vetoriais. Com estes processadores uma instrução é aplicada sobre um conjunto de dados (um vetor) de uma única vez. Alguns sistemas construídos com esta arquitetura são: Cray Series C90 (1991)/J90 (1994)/T90 (1995) e NEC SX-6 (2002).

4) arquitetura MPP (Massively Parallel Processing) onde o conjunto de processadores se conecta entre si através de um barramento rápido com auxílio de hardware adicional. Neste modelo cada processador tem seu próprio *cache* e seu *chip* de memória. Alguns processadores construídos com essa tecnologia são: Sistemas Cray T3D (1993), Sistemas Fujitsu VPP (1994), Sistema IBM SP-2 (1994) e os Sistemas Hitachi SR2000 (1996).

5) arquitetura de computação em cluster, que é um sistema de computação construído a partir de máquinas encontradas no mercado e com tecnologia de rede local (LAN – *Local Area Network*). Esta arquitetura é essencialmente de memória distribuída necessitando de software para realizar a conexão entre as máquinas por um sistema de passagem de mensagens. Tanto a MPP como a computação em *Cluster* tem uma arquitetura com memória distribuída. A diferença está na sofisticação da interconexão entre os processadores. É a alta largura de banda e a baixa latência do sistema MPP que permite a escalabilidade de milhares de processadores. A baixa escalabilidade dos *Clusters* é limitada pela sua rede de interconexão. Esta dificuldade, entretanto, tem sido objeto de muita pesquisa com novas tecnologias como Myrinet

[68] e Infiniband [69].

5.5 Os clusters utilizados

Os testes realizados ao longo deste trabalho utilizaram três clusters de computadores. As primeiras experiências foram realizadas numa máquina disponibilizada pelo LABPLAN. Após a obtenção de financiamento os testes passaram a ser feitos no cluster do GRUCAD, e, com a cooperação do L2EP, vários testes foram realizados no cluster deste último centro de pesquisa.

Tanto o cluster do LABPLAN como o cluster do GRUCAD utilizam tecnologia Ethernet, o primeiro conectado por uma rede Fast Ethernet de 100 Mbits/s e o último conectado por uma rede Gigabit a 1Gbit/s. Já o cluster do L2EP conecta suas máquinas através de rede Myrinet a 2Gbits/s, resultando uma máquina com desempenho superior, se comparado às duas anteriores.

A seguir é apresentada uma descrição minimalista das três máquinas, e logo a seguir um gráfico comparativo mostrando o desempenho da rede nos três casos.

5.5.1 O Cluster do L2EP

Este cluster é composto por dois tipos diferentes de máquinas, num total de 8 nós. O primeiro tipo de nó são computadores Intel Xeon bi-processado 2GHz, 1 GB RAM, HD SCSI de 15 GB e sistema operacional GNU/Linux Red Hat 7.2. O segundo tipo de nó são computadores Intel Xeon bi-processador 2.8 GHz, 1 GB RAM, HD ATA de 40 GB e sistema operacional GNU/Linux Red Hat 8.1. Todas as máquinas são conectadas por uma rede de fibra ótica do tipo Myrinet 2Gbits/s e gerenciada por um software CMU/Beowulf. A biblioteca de paralelização utilizada é a MPI versão 1, com implementação para a rede Myrinet.

5.5.2 O Cluster do GRUCAD

O cluster do Grucad é composto por cinco processadores Intel Pentium IV, 3GHz, 2 GB RAM, HD IDE no servidor com 80 GB e sistema operacional GNU/Linux Mandrake 10, com *kernel* 2.6.12. A conexão de rede é Ethernet Gigabit a 1Gbit/s. Este cluster difere do sistema do L2EP também no que concerne à inicialização das máquinas escravo, feita de forma remota. A inicialização remota sem disco usa serviços da rede para carregar o sistema operacional e fazê-lo executar. Isto é feito com uma placa de rede e um disquete que realiza a inicialização. Em seguida o sistema operacional é carregado na RAM do escravo. O *kernel* foi citado, pois na construção do cluster foi preciso encontrar um *kernel* que dispusesse dos *drives* compatíveis com a rede Gigabit. A biblioteca de comunicação usada pelos programas também é MPI, mas a distribuição é MPICH versão 2.

5.5.3 O Cluster do LABPLAN

O cluster do LABPLAN é composto por 16 Pentium IV 2,4 GHz, com 512 MB RAM, HD IDE 20 GB no servidor e sistema operacional Linux Mandrake. A conexão de rede é Fast Ethernet a 100 Mbits/s. Este cluster também utiliza a inicialização remota dos computadores escravos. A comunicação executada pelos softwares é realizada pela biblioteca MPI versão 1 da distribuição MPICH.

5.5.4 Comparações de desempenho de rede

O gráfico a seguir foi elaborado através do software NetPIPE [63]. NetPIPE é uma ferramenta para avaliação de desempenho que mostra visualmente o desempenho da rede considerando diversas situações. O programa realiza diversos testes ping-pong (envio e recepção de mensagens), aumentando o tamanho das mensagens entre dois

processos, através de uma rede de computadores ou dentro de um sistema SMP. Os tamanhos das mensagens são escolhidos a intervalos regulares, e com leves perturbações, para providenciar um teste completo do sistema de comunicação. Cada novo nível de mensagens envolve muitos testes ping-pong a fim de se obter uma medida de tempo acurada.

As curvas mostram largura de banda x tamanho da mensagem. O valor das latências de cada uma das redes é mostrado junto à legenda. Percebe-se a vantagem da tecnologia Myrinet sobre a tecnologia Ethernet não somente no valor da largura de banda, mas também no valor da latência apresentada.

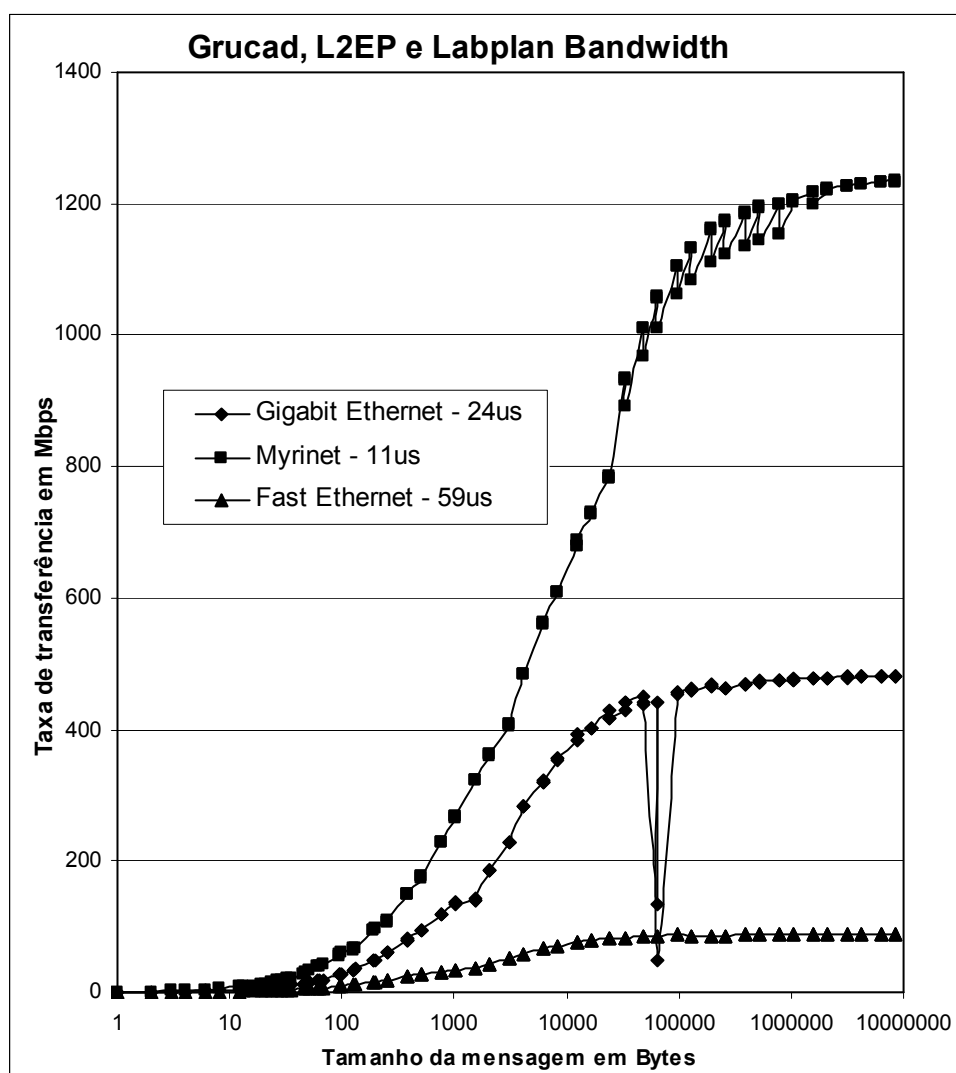


Fig. 5.2 – Largura de banda para os três clusters utilizados.

5.6 Programação paralela

As duas formas mais difundidas de programação paralela estão associadas às arquiteturas paralelas nas quais elas serão implementadas: programação utilizando memória compartilhada e a programação utilizando troca de mensagens.

A programação utilizando memória compartilhada é a mais antiga sendo ainda usada. É considerada como de fácil implementação, utilizando diretivas de compilação. Uma vez que os dados estão disponíveis a todos os processadores o programador preocupa-se apenas com o sincronismo para evitar que os dados sejam alterados indevidamente.

Por outro lado, a programação utilizando a troca de mensagens requer que o programador mova blocos de dados através da memória utilizando comunicações explícitas. Os processos devem se comunicar cooperativamente, ou seja, um processo inicializa a operação através do envio de uma mensagem o que deve corresponder à operação de recebimento em outro processo. Estas operações de envio e recebimento de mensagens podem ser sincronizadas ou podem ser bloqueantes, o que significa que o processo que enviou a mensagem só estará livre para praticar outras tarefas após o recebimento desta mensagem por outro processo.

Os modelos fundamentais de programação são o modelo mestre-escravo e o modelo SPMD – *Single Program, Multiple Data*. No modelo mestre-escravo, os programas para o mestre e para o escravo são diferentes. O processo mestre centraliza o controle, inicializando os escravos. Também distribui o trabalho, sincronizando a comunicação e executando as operações de I/O. O processo mestre pode ou não contribuir para o processamento do problema. Tipicamente, os escravos executam todo o processamento.

No modelo SPMD não existe um controle centralizado e todos os processadores executam o mesmo programa, contribuindo para o processamento. Para tanto necessita que os cálculos e as comunicações entre os processadores sejam coordenados em determinados pontos.

Este trabalho foi realizado em uma arquitetura de memória distribuída. O modelo de programação utilizado foi o SPMD, usando o paradigma da troca de mensagens

através da biblioteca MPI (*Message Passing Interface*) com a versão 2 da distribuição MPICH.

5.7 A Biblioteca MPI

Atualmente, a biblioteca MPI é um padrão da indústria, que começou na forma de um esforço comunitário envolvendo pesquisadores de universidades, laboratórios de governos e indústrias [35].

Esta biblioteca é portátil. Ela pode ser utilizada em máquinas com memória distribuída, com memória compartilhada, redes comuns e até mesmo em um único processador para emular paralelização. Tem aproximadamente 125 funções para programação e ferramentas para se analisar o desempenho utilizando programas em C, C++ e Fortran.

Algumas das rotinas componentes da biblioteca são:

- Rotinas de gerência de processos (inicialização, finalização, determinação do número de processos e identificação dos mesmos).
- Rotinas de comunicação ponto-a-ponto (envio e recebimento de mensagens entre dois processos).
- Rotinas de comunicação por grupos (“*broadcast*”, sincronização de processos).

5.8 O *Framework* de teste

Uma das principais diretrizes que guiou o desenvolvimento desta pesquisa foi a compatibilidade com métodos e programas já desenvolvidos ao longo da existência do GRUCAD, a fim de que o material desenvolvido por este trabalho não contivesse, além da técnica e métodos inerentes ao emprego da paralelização, outros elementos que resultassem em empecilhos a serem vencidos no reaproveitamento do material.

Como ressaltado no item 5.2, um programa escrito com o objetivo de obter desempenho precisa seguir regras como aquelas descritas naquele item. Assim, para se obter desempenho é necessário que o programa esteja bem escrito, fazendo uso de pessoas com grande capacidade de codificação, auxílio este com que, felizmente, o autor deste trabalho pôde contar. Logo, a compatibilidade, neste caso, tem limites. Além disso, visto a grande quantidade de possibilidades de comparação foi necessário estruturar o trabalho de forma a ser expansível e flexível. O *framework* aqui descrito foi a forma natural encontrada para atender a esta exigência.

Um outro objetivo deste trabalho foi documentar certos aspectos computacionais como as técnicas de armazenamento esparsa e de decomposição LU utilizadas nos programas de elementos finitos e explicações a respeito de diversos algoritmos. Estes dados se encontram nos códigos escritos e na literatura de projeto que compõe todo este trabalho, disponíveis sob a forma de relatórios, encontrando-se na posse dos pesquisadores que formam o GRUCAD.

Assim, na medida do possível, procurou-se manter a compatibilidade com:

- a linguagem de programação: os programas que tratam da resolução dos sistemas de equações lineares e rotinas correlatas foram escritos em FORTRAN. Os programas que gerenciam os testes foram escritos na linguagem do *shell* do sistema operacional Linux e também em C++. Isto porque estas linguagens incorporam uma série de facilidades que atendiam às necessidades do conjunto de dados de que se procurava dispor para análise.
- com o método de armazenamento utilizado pelos programas do grupo de pesquisa. Foi mantido o método de armazenamento RCS utilizado pelos programas do FeeCAD, para armazenamento das matrizes dos casos tridimensionais. Como será mostrado nos resultados este método não possui o melhor desempenho nos casos paralelizados, além de não atender necessidades de armazenamento como aquelas exigidas por matrizes que utilizam métodos como o Mortar [70].
- com rotinas utilizadas pelos programas do EFCAD, como a rotina de decomposição LU, testada e validada em inúmeros casos e que utiliza o método de armazenamento RCS.

O *framework* contém vários programas com os quais é possível gerar executáveis com diferentes opções como número de processadores a usar, tipo de pré-condicionador, tolerância, o caso a ser estudado, e porcentagem de interseção.

Os métodos de resolução utilizam a mesma biblioteca compartilhada. Cada arquivo de resolução representa um método diferente de resolução de sistemas de equações e as atividades realizadas por ele são as seguintes:

- Ler o sistema matricial a partir de arquivos onde a matriz A e o vetor b estão armazenados;
- Resolver o sistema;
- Gravar um arquivo de saída com o vetor-solução x .
- Gravar um arquivo de saída com uma tabela com dados de desempenho.

A estrutura geral do *Framework* foi projetada para possibilitar a introdução fácil de novos pré-condicionadores e de novos métodos de resolução matricial, tal que todos os métodos de resolução funcionem com todos os pré-condicionadores.

5.8.1 Estrutura de diretórios

O *framework* está distribuído em uma estrutura padronizada de diretórios. Um diretório **lib** contém as rotinas de leitura dos sistemas matriciais, de multiplicação matriz por vetor, de pré-condicionamento e de gravação do vetor-solução.

Cada método de resolução contém um arquivo “.F” principal armazenado dentro de um diretório exclusivo para aquele método. Por exemplo, o método CGS está no programa **cgsmqtt.F** dentro do diretório **cgsmqtt**.

A seguir está a listagem completa da estrutura de diretórios que compõe o *framework*:

- 📁 **f**: diretório com os arquivos “.F” e os arquivos das malhas
- 📁 **lib**: diretório com arquivos “.F” comuns a todos os métodos de resolução
 - 📄 **comum.F**: declarações “common” (variáveis globais)

- 📄 **distribui.F**: rotina que calcula as frações da matriz e a distribui entre os computadores (rotina utilizada pelo método de armazenamento RCS);
- 📄 **iccg_rotinas.F**: aqui está armazenada a rotina de pré-condicionamento *LU* utilizada pelos programas de elementos finitos do GRUCAD;
- 📄 **inicializacao.F**: procedimentos de inicialização comuns a todos os métodos de resolução, como declaração do ambiente paralelo, exibição de mensagens títulos, leitura da matriz, e etc.;
- 📄 **nia_le_matriz.F**: leitura da matriz para o método de armazenamento RCS;
- 📄 **nia_pll_mult.F**: multiplicação matriz por vetor paralelizada para o método de armazenamento RCS;
- 📄 **nia_pre_cond.F**: rotinas de pré-condicionamento para o método de armazenamento RCS;
- 📄 **saida.F**: grava o vetor-solução em arquivo e exibe mensagens de resultado na tela do computador.

📁 **cgmqt**

- 📄 **cgmqt.F**: código-fonte do método de resolução CG;

📁 **cgsmqt**

- 📄 **cgsmqt.F**: código-fonte do método de resolução CGS.

📁 **bicgmqt**

- 📄 **bicgmqt.F**: código-fonte do método de resolução BiCG.


📁 **bicgestabmqt**

- 📄 **bicgestabmqt.F**: código-fonte do método de resolução BiCG Estabilizado.

📁 **malhas**: diretório contendo as malhas (arquivos com as matrizes armazenadas no formato (i, j, valor)).

📁 **c**: diretório com os programas auxiliares em C++.

📁 **xpll**: diretório com o código-fonte do programa que auxilia a automatização dos testes de desempenho.

 **gealmaeno**: diretório com o código-fonte do programa gerador de matrizes sintéticas.

5.8.2 Arquivos com os dados das malhas

Os arquivos de malhas (.dat) estão no diretório f/malhas. Embora suas extensões sejam “.dat”, os arquivos estão em formato texto. Seguem as explicações sobre os diferentes padrões de arquivos:

- `vdr_malha_<nno>.dat`: são os vetores de fontes ou vetores \mathbf{b} do Sistema $\mathbf{Ax} = \mathbf{b}$.
- `ss_malha_<nno>e_acl.dat`: são os arquivos da matriz \mathbf{A} dos sistemas $\mathbf{Ax} = \mathbf{b}$, codificados especialmente para serem alocados com o método de armazenamento ACL. A matriz é simétrica e somente os elementos não-nulos acima da diagonal e da diagonal, inclusive, constam nesses arquivos.
- `ss_malha_<nno>e_nia.dat`: são os arquivos da matriz \mathbf{A} dos sistemas $\mathbf{Ax} = \mathbf{b}$, codificados especialmente para serem alocados com o método de armazenamento RCS.

Nos nomes de arquivos acima <nno> significa o número de variáveis do sistema.

5.8.3 Programa XPLL

Este programa é utilizado para realizar a avaliação de desempenho dos métodos iterativos paralelizados neste trabalho. É o programa principal do ponto de vista do usuário, pois é através dele que são realizadas as medidas de desempenho dos diversos métodos. As curvas que ilustram o capítulo de resultados foram obtidas através das saídas deste programa.

Os parâmetros aceitos por este programa são:

```
xpll <iSistematica> <sNomeArquivo> <iNpMinimo> <iNpMaximo>
      <iNumAmostras> <iMetodoEstatistico> <iNNO> <dInterMenor>
      <dInterMaior> <dInterPasso> <iOQueGuardar>
```

Segue a descrição dos parâmetros:

<iSistematica>

0 - tradicional: executa um arquivo executável já compilado e gera um arquivo de resultados **np**² versus tempo ou iterações. O nome do arquivo gerado tem o formato:

```
resultado_<sNomeDoArquivo>.wri.
```

1 - sistemática para o método FLUI³: compila o programa antes de executá-lo utilizando o script `nia_script.sh`. O programa precisa ser sempre recompilado para poder variar o percentual de interseção. A recompilação é necessária porque os vetores de armazenamento são sempre redimensionados para se ajustarem exatamente aos tamanhos dos vetores para cada caso. Quando os vetores são dimensionados com tamanhos maiores que os tamanhos necessários aos casos de teste, os tempos medidos são influenciados pelas dimensões destes vetores, mascarando os resultados. O arquivo de resultado gerado será uma tabela de tempos ou iterações (dependendo do parâmetro <iOQueGuardar>) para cada valor de `numero_de_processadores` e `percentual_de_interseccao`. O nome do arquivo gerado tem o formato:

```
resultado_<iNNO>_FLUI_<dInterMenor>_<dInterMaior>_<dInterPasso>.wri.
```

<sNomeArquivo>

Seu significado varia dependendo do valor de <iSistematica>: se <iSistematica>==0, <sNomeArquivo> é o nome do executável já compilado; se <iSistematica>==1, <sNomeArquivo> é o nome do arquivo “.F” principal.

<iNpMinimo>

² **np** significa número de processadores. É um parâmetro passado pelo usuário.

³ FLUI significa: Fatoração LU incompleta por blocos com interseção. O método de pré-condicionamento proposto nesta tese.

Número de processadores (**np**) mínimo.

<iNpMaximo>

Número de processadores (**np**) máximo.

<iNumAmostras>

Número de repetições de cada execução de um programa. As repetições podem ser úteis devido à existência de uma variação no tempo de processamento.

<iMetodoEstatistico>

Este parâmetro só faz sentido se <iNumAmostras> > 1:

0-menor tempo: o menor tempo obtido em todas as repetições de cada execução de um programa será mantido.

1-media dos tempos: será feita a média de todas as repetições de cada execução de um programa.

Os próximos parâmetros só são usados se <iSistemática>==1

<iNNO>

Número de nós.

<dInterMenor>

Menor valor de interseção do pré-condicionador FLUI.

<dInterMaior>

Maior valor de interseção do pré-condicionador FLUI.

<dInterPasso>

Passo de incremento da interseção do pré-condicionador FLUI.

<iOQueGuardar>

Informa o que guardar no arquivo (tabela) de resultados:

0 – tempos;

1 – número de iterações.

Assim, um exemplo da linha de comando do programa XPLL seria:

```
xpll 1 bicgmsqt.F 1 5 3 0 1031 1e-5 0 100 1 1
```

significando que o programa utilizaria o pré-condicionador FLUI com o método de resolução de sistemas de equações BiCG, variando o número de processadores de 1 a 5, fazendo 3 testes com cada arquivo executável gerado, e anotando o menor tempo desses 3 testes. Usaria a malha com 1031 nós, com convergência com erro menor ou igual a 1×10^{-5} , variando a interseção de 0% a 100%, de 1 e 1, guardando no arquivo de saída o número de iterações.

O nome do arquivo (tabela) de saída será `resultado_1031_FLUI_0_100_1.wri`

INTER	1	2	3	4	5
0.0000	31	53	58	69	72
1.0000	31	54	58	70	72
2.0000	31	55	59	70	73
3.0000	31	55	59	71	73
4.0000	31	56	59	72	74
5.0000	31	54	59	72	74
6.0000	31	58	60	72	76
7.0000	31	58	60	72	77
8.0000	31	59	60	71	77

Fig. 5.3 – Tabela de saída com a variação da interseção e o número de iterações por máquina.

5.8.4 Script de compilação

Parte da flexibilidade na produção dos resultados se deve à ação dos scripts que permitem que os programas escritos em Fortran existentes na biblioteca do diretório **lib** se utilizem de variáveis definidas em tempo de pré-compilação passadas por linha de

comando. O script de compilação foi escrito com a linguagem do *shell* do Linux, semelhante à linguagem dos programas em lote do DOS, mas com muito mais recursos.

O pré-compilador (chamado em inglês de *preprocessor*) substitui as variáveis pelos valores passados (ação semelhante ao `#define` do C). Os valores são passados através da opção `-D` da linha de comando do compilador. Assim, as variáveis definidas em tempo de pré-compilação são: `D_NNO` (número de variáveis do sistema); `PRE_COND` (tipo de pré-condicionador); `D_TOLERANCIA` (número real especificando o erro mínimo de convergência); `INTER` (porcentagem de interseção); `D_MAXLIN` (número de elementos a serem lidos no arquivo da malha); `D_FORMAT` (número de caracteres das colunas 1 e 2 do arquivo da malha).

Com estas instruções passadas para o compilador `mpif77` (compilador MPI para a linguagem Fortran 77) na linha de comando, o compilador cria um executável para cada opção enviada. Estas opções são controladas e gerenciadas pelo programa `XPLL` descrito acima e definidas à priori pelo usuário.

5.8.5 A geração de matrizes sintéticas

Embora o objetivo principal deste trabalho seja o de utilizar e avaliar a paralelização na resolução de sistemas matriciais gerados pelo método de elementos finitos, devido à grande esparsidade destes últimos, foram necessários testes com matrizes mais densas.

Assim sendo, foi criado um programa gerador de matrizes sintéticas (`GEALMAENO`). Este programa gera matrizes e vetores independentes a partir de equações matemáticas, com as quais é possível controlar o número de elementos por linha e, conseqüentemente, a esparsidade da matriz.

O programa precisa de quatro parâmetros: a ordem da matriz; a largura de meia-banda, um fator de preenchimento dentro da banda e um fator de atenuação dos elementos fora da diagonal. Com estes parâmetros são criados dois arquivos: o arquivo da matriz e o vetor independente. Na saída do programa também são fornecidos o número de linhas do arquivo da matriz e a média de elementos não nulos na banda.

No capítulo onde serão mostrados os resultados obtidos nos testes, também serão mostrados os resultados da paralelização da resolução de matrizes sintéticas, resultantes da aplicação deste programa.

5.9 Núcleos de paralelização

Como citado no item 5.6, este trabalho fez uso da programação paralela através de troca de mensagens com o modelo SPMD. Assim, todos os processadores executam o mesmo programa e todos contribuem para o processamento, inclusive o processador com identificador (*rank*) igual a zero. Assim, o *mestre* também trabalha. Entretanto, como a utilização de troca de mensagens exige a coordenação de esforços, uma parcela do código existente é executada em apenas um processador, o processador com identificador igual a zero. As tarefas executadas por este processador são, notadamente, aquelas de distribuição de dados e reunião de resultados, leitura e gravação de arquivos.

A distribuição dos dados entre os processadores é uma tarefa de conceito simples, mas quando as máquinas que formam o cluster são heterogêneas, o balanceamento de carga pode ser delicado. Felizmente as máquinas utilizadas no cluster do GRUCAD são semelhantes, tal que o algoritmo de balanceamento de carga não considera fatores de desequilíbrio. Entretanto [71], este tratamento é fundamental em clusters com máquinas heterogêneas, já que o maior tempo será devido ao elemento mais lento da rede.

O primeiro procedimento na divisão de carga é o cálculo do número ideal de elementos em cada processador. Neste caso, cada processador deveria receber um número ideal de elementos (n_{ideal}) calculados como uma função do número total de elementos diferentes de zero (nnz) na matriz e do número de processadores usados (nhost), dado por:

$$n_{\text{ideal}} = \frac{\text{nnz}}{\text{nhost}} \quad (5.2)$$

Caso essa divisão de inteiros resulte resto, este resto será novamente dividido entre os processadores. Esse cálculo, contudo, pode fazer com que elementos localizados em uma determinada linha (ou coluna) da matriz sejam destinados a processadores diferentes, o que é proibitivo, já que um processador precisaria de informações presentes em outra máquina.

Então, de acordo com a figura 5.4, analisa-se a última linha do bloco destinado a um determinado processador e se o número de elementos que esta linha já tem ($iDistancia1$) é menor que o número de elementos que ela precisa para ficar completa ($iDistancia2$), esta linha não é destinada a este processador. Caso contrário, se o número de elementos que ela já tem é maior que o número de elementos que ela precisa para ficar completa, esta linha passa a pertencer a este nó.

A figura a seguir ilustra esta idéia. Embora o armazenamento RCS utilizado armazene os dados na forma de uma cunha, a idéia também pode ser aplicada a outros algoritmos, assim, a figura 5.4 ilustra o conceito considerando blocos inteiros de dados. Os retângulos da figura 5.4 ilustram porções de uma matriz. A variável $iQuantosJahSug$ representa o número de elementos sugeridos para armazenamento no processador. A variável $iSomaTemp$ representa o número de elementos sem a inclusão da última linha. A variável $iAcrescimo$ representa o número de elementos da última linha. Através das decisões ilustradas na figura a linha será ou não destinada ao processador.

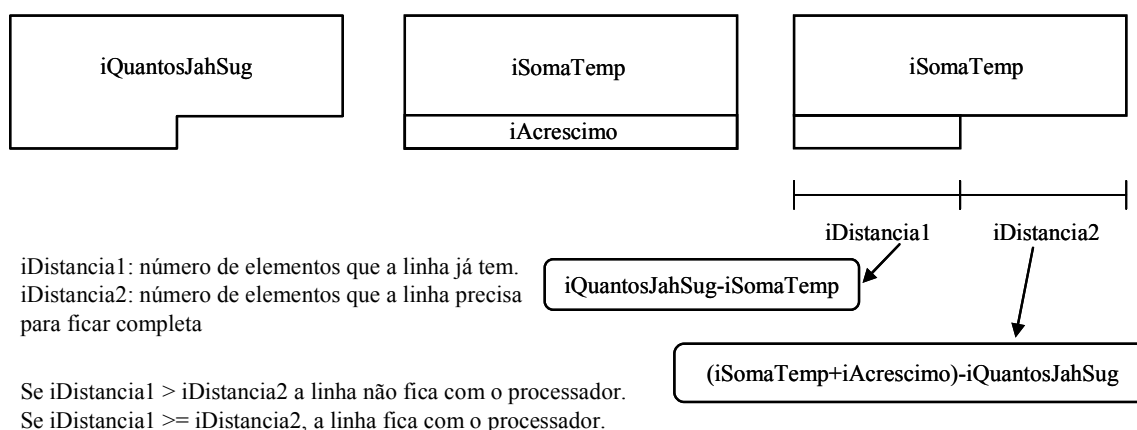


Fig. 5.4 – Balanceamento de carga utilizado.

Depois de adequadamente distribuídos os dados do sistema, podem ser iniciadas

as operações paralelizadas existentes nos diversos algoritmos de resolução de sistemas de equações. Independentemente do método escolhido, os núcleos paralelizáveis são:

- Atualização de vetores, operação também conhecida como *saxpy* ou *linked triad* e que consiste na multiplicação de vetores por um escalar seguido da soma com outro, atualizando ou criando um terceiro vetor;
- Produto escalar entre vetores, também chamado de produto interno;
- Multiplicação matriz por vetor;
- Elaboração da matriz de pré-condicionamento;
- Resolução do sistema de pré-condicionamento.

As operações de atualização e de produto escalar entre vetores são operações envolvendo uma pequena quantidade de dados, normalmente igual a duas vezes a largura de banda da matriz. No caso das matrizes de elementos finitos utilizadas, o número médio de elementos por linha varia entre 46 (Matriz 219519) e 80 (Matriz 347829), considerando a totalidade da banda. Isto quer dizer que nas operações de atualização e de produto escalar envolvendo vetores relacionados às matrizes acima, tem-se menos de 100 operações com resultados diferentes de zero. Isto não é muito diferente dos vetores resíduo e de direção presentes nos algoritmos. Desta forma, devido à inexpressiva quantidade de dados encontrados nos vetores, o tempo de envio e recebimento é muito maior que a própria computação seqüencial do resultado. Experimentalmente se notou que a paralelização destes núcleos com o uso de clusters de computadores tem seu desempenho altamente degradado, tal que os mesmos não foram implementados nos programas de teste descritos neste trabalho.

Já as operações envolvendo matrizes têm uma abordagem diferente. Uma vez que os vetores que compõe o sistema matricial $Ax = b$ são linearmente independentes necessariamente, os produtos linha por coluna podem ser realizados de forma independente também. Tal como descrito no início deste tópico, blocos de linhas da matriz são agrupados e designados a um processador. Basicamente, o início e o fim de cada bloco de linhas são determinados por variáveis como *bloco* e *offset*, como ilustrado na figura 5.5. A variável *bloco* indica o número de linhas pelo qual o processador é responsável. A variável *offset* indica a linha a partir da qual o

processador deve operar. A figura 5.5 abaixo representa uma matriz esparsa dividida entre dois processadores.

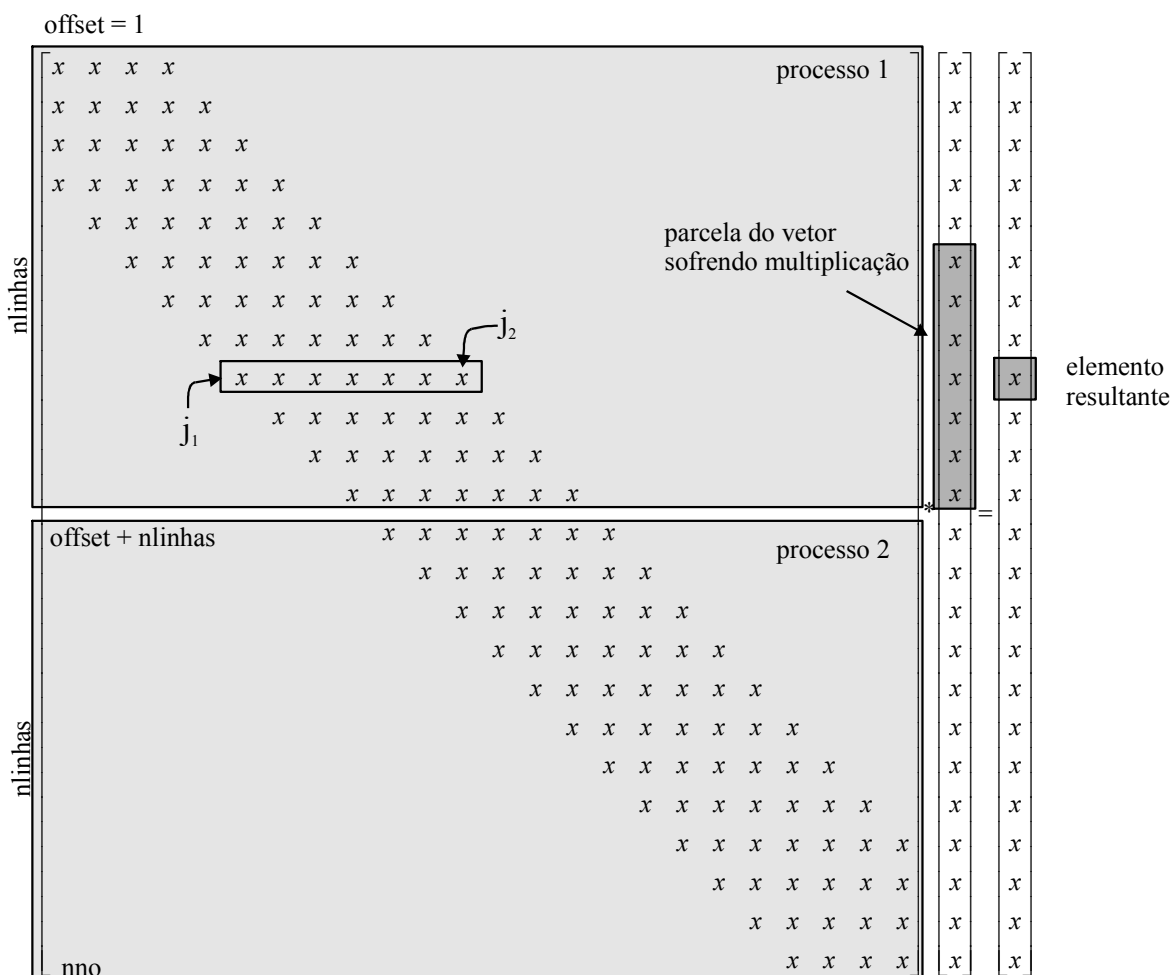


Fig. 5.5 – Multiplicação matriz esparsa por vetor.

Os valores das variáveis *bloco* e *offset* são determinadas na etapa de balanceamento de carga, procedimento este já descrito.

Na figura 5.5 se pode observar que cada processador realiza apenas uma parte do cálculo do produto, sempre entre os limites j_1 e j_2 da linha dentro da banda, pela porção equivalente do vetor, gerando um elemento no vetor de resultados. O particionamento do produto acarreta um particionamento dos resultados. Cada processador retém apenas uma porção do resultado total. Desta forma cabe ao processo definido com $rank=0$ reunir todas as partes do vetor solução, ordenando os resultados.

Outra observação se refere ao fato de que a transferência da matriz de coeficientes ocorre uma única vez, já que a matriz é imutável, variando apenas os vetores pelos quais a mesma é multiplicada. Estes sim são enviados toda vez que ocorre uma atualização, na mesma ou numa nova iteração. Aos processadores utilizados nas computações são informados dados suficientes para realizar os produtos parciais repetidas vezes.

A elaboração do pré-condicionamento é também uma tarefa que pode ser realizada em paralelo. Em máquinas com memória compartilhada se pode construir pré-condicionadores como Cholesky ou LU incompleto da mesma forma como são elaborados no processamento seqüencial, fazendo uso de partes da memória comum a todos os processadores utilizados.

Em máquinas com memória distribuída como clusters de computadores, pré-condicionadores como os acima descritos fariam uso intensivo da rede numa seqüência proibitiva de troca de mensagens. Por isso, para se conseguir um pré-condicionador elaborado em paralelo é necessário se fazer algumas concessões, entre elas permitir uma aproximação mais pobre da decomposição original, estratégia utilizada nesta tese.

Como descrito no capítulo sobre pré-condicionadores, a técnica de pré-condicionamento aqui utilizada faz uma interseção entre blocos contíguos da matriz, procurando diminuir o enfraquecimento causado pelo desacoplamento. Com isso, porém, constrói-se uma matriz de pré-condicionamento em cada processador, paralelamente no tempo.

Uma vez que a matriz de pré-condicionamento se encontra distribuída nos processadores do cluster, a aplicação paralela é imediata. Isto equivale a resolver o sistema $Mz = r$ existente nos diversos algoritmos do capítulo 3 de forma independente. Cada processador recebe um vetor resíduo (r) e calcula sua parcela de resíduo M -ortogonal (z). Naturalmente estas parcelas individuais são devolvidas ao processador com $rank=0$ para a elaboração do vetor resíduo completo.

Este trabalho realiza em paralelo as operações de multiplicação matriz por vetor, a elaboração e a aplicação do pré-condicionamento. Como opção de cálculo a partir da linha de comando também é possível calcular o pré-condicionamento LU incompleto na forma seqüencial, além do pré-condicionamento de Jacobi.

Entretanto, como apontado em [72], a operação crítica é a multiplicação matriz

por vetor. Ainda segundo esta referência, para obter o *speedup* das computações é preciso realizar apenas as operações que mais consomem tempo de processamento.

5.10 Conclusão

Este capítulo procurou mostrar a interdependência entre o problema algorítmico e a arquitetura de hardware disponível. O hardware utilizado neste trabalho é um cluster de computadores mais apropriado para a solução de algoritmos de granulosidade grossa. Entretanto, o texto também abordou que a rede de interconexão entre computadores é motivo de intensas pesquisas com grandes promessas num futuro próximo com as novas tecnologias sendo estudadas. O trabalho fez uso de duas tecnologias: Ethernet Gigabit e Myrinet, padrões de rede utilizados em larga escala atualmente.

O texto também descreveu de forma extensiva o *framework* e os programas desenvolvidos. Também descreveu os cuidados tomados com a adequada distribuição de carga entre os processadores e detalhes dos diversos núcleos de processamento paralelizados nos programas de teste.

O capítulo a seguir trata da apresentação dos resultados obtidos com a paralelização dos métodos estudados. Serão apresentadas primeiramente as quatro formas de armazenamento esparsa de matrizes utilizadas para avaliar a influencia destes esquemas na paralelização dos métodos iterativos na resolução de sistemas de equações lineares. Em seguida serão apresentados os resultados relativos aos esquemas de armazenamento, mostrando as diferenças de desempenho entre eles. Depois será mostrado o desempenho do pré-condicionador FLUI desenvolvido nesta tese. O pré-condicionador FLUI foi aplicado a um conjunto de matrizes de tamanhos variados, com os quais se obteve uma visão geral do seu desempenho em relação ao número de iterações.

Depois serão mostrados os resultados com a paralelização para os tempos de processamento, dentro de uma faixa de variação da porcentagem do pré-condicionador FLUI, bem como a apresentação de resultados relativos a outros pré-condicionadores.

6 Resultados

6.1 Introdução

*E*ste capítulo trata da apresentação dos resultados obtidos com a paralelização dos métodos estudados. Serão apresentados primeiramente os resultados relativos aos esquemas de armazenamento. Dentre as várias formas de armazenamento encontradas na literatura foram estudadas aquelas que mais se aproximavam do método já utilizado para armazenar as matrizes esparsas oriundas da aplicação 3D do método de elementos finitos. Estes métodos têm semelhanças conceituais nas formas de armazenamento, entretanto utilizam abordagens diferentes para a realização da multiplicação e funções diferentes para realizar as operações de soma. Isto faz com que os esquemas de armazenamento apresentem desempenhos diferentes tanto na implementação seqüencial como na implementação paralela.

Em seguida serão mostrados os resultados para o pré-condicionador FLUI desenvolvido nesta tese. O pré-condicionador FLUI foi utilizado nos quatro métodos iterativos de resolução de sistemas de equações discutidos no capítulo 3 e foi aplicado a um conjunto de matrizes de tamanhos variados, com os quais se obteve uma visão geral do seu desempenho em relação ao número de iterações.

O comportamento do pré-condicionador FLUI é explorado numa faixa que vai desde o total desacoplamento dos blocos até a utilização de todo o bloco anterior por um processador. Mas devido aos resultados encontrados, é necessário utilizar apenas uma parcela desta variação para mostrar o desempenho em relação aos tempos de armazenamento.

Além dos resultados para o pré-condicionador proposto, serão mostrados alguns resultados utilizando o pré-condicionador *LU* incompleto calculado integralmente e posteriormente dividido entre os processadores, a utilização do pré-condicionamento de Jacobi, resultados para matrizes menores e mais densas, e comparações de desempenho

envolvendo o cluster do laboratório L2EP da Universidade de Lille.

6.2 Os casos testes

Os resultados apresentados neste capítulo foram obtidos a partir de dois problemas físicos e, também, a partir de expressões matemáticas. Os problemas físicos foram gerados através de sistemas computacionais 2D e 3D. Os casos testes são:

- Um capacitor de placas planas modelado pelo módulo FEECSE do sistema 3D de cálculo FEECAD, usando a formulação em potencial escalar com elementos nodais, obtendo-se como solução a distribuição do campo elétrico na estrutura. Com ele foram geradas as matrizes 1, 3, 4 e 7, descritas com mais detalhes logo a seguir.
- Um problema envolvendo um servomotor com ímãs permanentes em 2D, gerado pelos programas do sistema EFCAD, com formulação em potencial vetor magnético com elementos nodais, obtendo-se como solução a distribuição do fluxo magnético no dispositivo. Com ele foi gerada a matriz 6.
- Expressões matemáticas para controle do número de elementos na banda, com as quais foram geradas as matrizes 2 e 5.

A tabela abaixo mostra algumas das características destas matrizes, como número de variáveis, a esparsidade e em que cluster foram testadas.

TABELA 6.1. MATRIZES UTILIZADAS.

Matriz (número de variáveis)	Denominação	Origem	Esparsidade	GRUCAD	L2EP
347829	Matriz 1	Elem. Finitos 3D	$1,13 \times 10^{-4}$	SIM	SIM
347832	Matriz 2	Expressão	$2,32 \times 10^{-4}$	SIM	SIM
1009899	Matriz 3	Elem. Finitos 3D	$2,03 \times 10^{-5}$	SIM	NÃO
2163200	Matriz 4	Elem. Finitos 3D	$2,40 \times 10^{-5}$	SIM	NÃO
2000	Matriz 5	Expressão	0,499	SIM	NÃO
1080	Matriz 6	Elem. Finitos 2D	$6,60 \times 10^{-3}$	SIM	NÃO
219539	Matriz 7	Elem. Finitos 3D	$1,06 \times 10^{-4}$	SIM	NÃO

Esparsidade é aqui definida como a porcentagem do número de elementos não nulos sobre o número total de elementos para uma mesma matriz cheia.

6.3 Esquemas de armazenamento de matrizes esparsas

Durante o desenvolvimento dos algoritmos paralelos, observando-se a grande diferença de velocidade encontrada entre o método de Armazenamento Compactado em Linha (ACL) utilizado nos programas de elementos finitos em 2D e o formato de Armazenamento por Linha e Coluna (RCS) utilizado nos programas em 3D, perguntou-se qual seria a influência de um formato de compactação de matrizes esparsas na paralelização. O formato ACL utiliza uma grande quantidade de dados, mesmo quando a matriz é muito esparsa. O formato RCS armazena somente os elementos não nulos, mas duplica o armazenamento da diagonal e, na implementação em paralelo utiliza uma função de comunicação coletiva não usada na implementação de alguns outros métodos.

Assim, resolveu-se pesquisar também formas alternativas de armazenamento, focando o desempenho destes métodos nos produtos matriz por vetor, utilizando um método iterativo de resolução de sistemas de equações. Como são vários os esquemas de armazenamento propostos na literatura, decidiu-se por aqueles cuja descrição e utilização são mais conhecidas e próximas do método utilizado nos programas 3D do GRUCAD: o método de Armazenamento Compactado por Linha (CRS) e o método de Armazenamento Compactado por Coluna (CCS). Além desses dois métodos,

encontrou-se uma versão otimizada do método CRS, chamado Armazenamento Compactado por Linha Incremental (ICRS), que também foi incluído na lista de métodos potencialmente úteis.

Os programas de teste desenvolvidos para comparar o desempenho destes métodos de armazenamento também realizam a divisão de carga entre os processadores segundo o procedimento descrito no item 5.9, procurando um número ideal de elementos, calculado pela expressão (5.2).

Entretanto, o valor de n_{ideal} para o método RCS, método que armazena os dados na forma de uma cunha usando as técnicas dos métodos CRS e CCS, especificamente nestas comparações de produtos matriz por vetor envolvendo diversos métodos de armazenamento, foi calculado de uma forma um pouco diferente, de acordo com a expressão (6.1), onde $nrow$ é o número de linhas da matriz.

$$n_{ideal} = \frac{\frac{nnz-nrow}{2} + nrow}{nhost} \quad (6.1)$$

As comparações de desempenho foram realizadas considerando dois aspectos diferentes na implementação. O primeiro analisando o desempenho de acesso do método aos elementos da matriz nas rotinas de multiplicação utilizando o expediente paralelo, com as mensagens de envio e recebimento de dados. Este resultado dá uma noção do desempenho da técnica utilizada pelo método, sua rapidez no processamento, quase de forma seqüencial. Isto é obtido executando-se um determinado número de vezes um cálculo matriz por vetor obtendo-se o menor tempo dentro do conjunto de tomadas. O segundo aspecto verifica a sobrecarga devido à passagem de mensagens exigidas por cada método. Esta sobrecarga não aparece na primeira abordagem. Ela é evidenciada quando a multiplicação matriz por vetor é realizada repetitivamente, como parte de um método iterativo, como os métodos iterativos de resolução de sistemas de equações lineares. A sobrecarga é ainda mais evidente quando a convergência é lenta, seja pela não utilização de um pré-condicionador, seja pela magnitude elevada do número de condição apresentado pelo sistema de equações.

A demonstração do desempenho dos métodos de armazenamento segundo a necessidade repetitiva de comunicação foi realizada com o Gradiente BiConjugado Estabilizado como método iterativo. Como já descrito no capítulo 3, este método utiliza duas multiplicações matriz por vetor sem a necessidade da utilização da transposta da matriz. Para que a convergência do método não seja tão lenta e ao mesmo tempo permitir a observação da influência do método de armazenamento, foi utilizado o pré-condicionamento diagonal nas resoluções dos sistemas de equações.

O uso do pré-condicionamento diagonal também foi escolhido por um segundo motivo. Quando ocorre a troca de um método de armazenamento é necessário que sejam reescritas as rotinas de multiplicação matriz por vetor, de elaboração do pré-condicionamento e da aplicação do pré-condicionador no algoritmo. Isto quer dizer que se o pré-condicionador utilizado utilizasse decomposições incompletas, seria necessário escrever as decomposições e as resoluções *forward* e *backward* em todos os métodos de armazenamento propostos.

Os programas escritos para atender os requisitos de comparação utilizaram o modelo de programação SPMD, com a centralização de resultados na máquina com identificador igual a 0.

Note-se que os métodos CRS e ICRS usam funções de comunicação coletiva diferentes dos métodos CCS e RCS. Os dois primeiros coletam os dados processados nas diversas tarefas com a função `MPI_GATHERV`, enquanto que os dois últimos realizam a mesma tarefa com a função `MPI_REDUCE`. Esta última sofre de erros de arredondamento nos bits menos significativos, tal que podem ser produzidos números de iterações diferentes quando são realizados cálculos com mais ou menos processadores.

Isto pode ser observado, por exemplo, na tabela 6.26 a – d. Observe-se nesta tabela que, quando o processamento é seqüencial ($N_p=1$), o número de iterações é igual em todos os métodos de armazenamento, evidenciando-se a dependência com o número de máquinas utilizadas.

A seguir serão descritos brevemente os métodos de compactação aqui testados e, através do exemplo da matriz A abaixo, serão exemplificadas as formas de armazenamento.

$$\mathbf{ROW_PTR} = [1 \ 4 \ 7 \ 9 \ 12 \ 15 \ 17] \quad (6.5)$$

Abaixo está a distribuição dos vetores em quatro processadores após o balanceamento.

TABELA 6.2. DISTRIBUIÇÃO DOS VETORES DO MÉTODO CRS EM QUATRO PROCESSADORES.

rank	<i>VAL</i>	<i>COL_IND</i>	<i>ROW_PTR</i>
0	[a ₁₁ a ₁₄ a ₁₅]	[1 4 5]	[1 4]
1	[a ₂₂ a ₂₄ a ₂₅ a ₃₃ a ₃₆]	[2 4 5 3 6]	[4 7 9]
2	[a ₄₁ a ₄₂ a ₄₄]	[1 2 4]	[9 12]
3	[a ₅₁ a ₅₂ a ₅₅ a ₆₃ a ₆₆]	[1 2 5 3 6]	[12 15 17]

6.3.2 O Método de Armazenamento Compactado por Coluna (Compressed Column Storage)

O método de Armazenamento Compactado por Coluna (CCS) é idêntico ao método CRS, exceto pelo fato de agora serem armazenadas as linhas de cada elemento não nulo num vetor *ROW_IND* e a posição do elemento inicial de cada coluna no vetor *COL_PTR*. Em outras palavras o método CCS é o método para armazenar a transposta da matriz *A*. A estrutura CCS de dados para matriz *A* é a seguinte:

$$\mathbf{VAL} = [a_{11} \ a_{41} \ a_{51} \ a_{22} \ a_{42} \ a_{52} \ a_{33} \ a_{63} \ a_{14} \ a_{24} \ a_{44} \ a_{15} \ a_{25} \ a_{55} \ a_{36} \ a_{66}] \quad (6.6)$$

$$\mathbf{ROW_IND} = [1 \ 4 \ 5 \ 2 \ 4 \ 5 \ 3 \ 6 \ 1 \ 2 \ 4 \ 1 \ 2 \ 5 \ 3 \ 6] \quad (6.7)$$

$$\mathbf{COL_PTR} = [1 \ 4 \ 7 \ 9 \ 12 \ 15 \ 17] \quad (6.8)$$

Abaixo está a distribuição dos vetores em quatro processadores após o balanceamento.

TABELA 6.3. DISTRIBUIÇÃO DOS VETORES DO MÉTODO CCS EM QUATRO PROCESSADORES.

rank	<i>VAL</i>	<i>ROW_IND</i>	<i>COL_PTR</i>
0	[a ₁₁ a ₄₁ a ₅₁]	[1 4 5]	[1 4]
1	[a ₂₂ a ₄₂ a ₅₂ a ₃₃ a ₆₃]	[2 4 5 3 6]	[4 7 9]
2	[a ₁₄ a ₂₄ a ₄₄]	[1 2 4]	[9 12]
3	[a ₁₅ a ₂₅ a ₅₅ a ₃₆ a ₆₆]	[1 2 5 3 6]	[12 15 17]

6.3.3 O Método Armazenamento Compactado por Linha Incremental (Incremental Compressed Row Storage)

O método de armazenamento ICRS é uma variante do formato CRS. Ele também não faz nenhuma suposição sobre a estrutura esparsa da matriz que será armazenada e também não armazena nenhum elemento nulo. A diferença entre eles é que este formato evita o endereçamento indireto $B(COL_IND(j))$ na multiplicação matriz por vetor, além de utilizar apenas dois vetores: um vetor de elementos reais *IVAL* e vetor de inteiros *INC*. O vetor *IVAL* armazena, como o vetor *VAL* do método CRS, os elementos não nulos da matriz. Já o vetor *INC* armazena o incremento entre um elemento e o seu subsequente do vetor *IVAL*. Assim, a estrutura ICRS de dados da matriz *A* é a seguinte:

$$IVAL = [a_{11} \ a_{14} \ a_{15} \ a_{22} \ a_{24} \ a_{25} \ a_{33} \ a_{36} \ a_{41} \ a_{42} \ a_{44} \ a_{51} \ a_{52} \ a_{55} \ a_{63} \ a_{66}] \quad (6.9)$$

$$INC = [1 \ 3 \ 1 \ 3 \ 2 \ 1 \ 4 \ 3 \ 1 \ 1 \ 2 \ 3 \ 1 \ 3 \ 4 \ 3 \ 1] \quad (6.10)$$

Abaixo está a distribuição dos vetores em quatro processadores após o balanceamento.

TABELA 6.4. DISTRIBUIÇÃO DOS VETORES DO MÉTODO ICRS EM QUATRO PROCESSADORES.

rank	<i>IVAL</i>	<i>INC</i>
0	[a ₁₁ a ₁₄ a ₁₅]	[1 3 1 2]
1	[a ₂₂ a ₂₄ a ₂₅ a ₃₃ a ₃₆]	[2 2 1 4 3 1]
2	[a ₄₁ a ₄₂ a ₄₄]	[1 1 2 3]
3	[a ₅₁ a ₅₂ a ₅₅ a ₆₃ a ₆₆]	[1 1 3 4 3 1]

6.3.4 O Método de Armazenamento Compactado por Linha e Coluna (Row and Column Storage)

O Método de Armazenamento Compactado por Linha e Coluna (RCS) é utilizado para armazenar matrizes simétricas ou estruturalmente simétricas. O método consiste em dividir a matriz A em duas matrizes, uma contendo a parte triangular inferior L e a outra contendo a parte triangular superior U , duplicando os elementos da diagonal.

Para armazenar os elementos não nulos de L e U são utilizados dois vetores, aqui chamados INF e SUP , respectivamente. A condição de armazenamento de matrizes estruturalmente simétricas permite utilizar apenas um vetor para armazenar o índice da linha/coluna dos elementos de INF / SUP . Este vetor de inteiros é denominado $IADRES$. Além disso, utiliza-se um quarto vetor, $NCED$, para armazenar apontadores para as linhas/colunas da matriz. A matriz é armazenada tal que a parte triangular inferior é elaborada por linhas e a parte triangular superior por colunas, unidas pelo elemento diagonal, comum em ambos os vetores, formando uma cunha. Para a matriz exemplo A , os vetores $NCED$, $IADRES$, INF e SUP adquirem a seguinte forma:

$$INF = [a_{11} \ a_{22} \ a_{33} \ a_{41} \ a_{42} \ a_{44} \ a_{51} \ a_{52} \ a_{55} \ a_{63} \ a_{66}] \quad (6.11)$$

$$SUP = [a_{11} \ a_{22} \ a_{33} \ a_{14} \ a_{24} \ a_{44} \ a_{15} \ a_{25} \ a_{55} \ a_{36} \ a_{66}] \quad (6.12)$$

$$IADRES = [1 \ 2 \ 3 \ 1 \ 2 \ 4 \ 1 \ 2 \ 5 \ 3 \ 6] \quad (6.13)$$

$$NCED = [1 \ 2 \ 3 \ 4 \ 7 \ 10 \ 12] \quad (6.14)$$

Abaixo está a distribuição dos vetores em quatro processadores após o balanceamento.

TABELA 6.5. DISTRIBUIÇÃO DOS VETORES DO MÉTODO RCS EM QUATRO PROCESSADORES.

rank	<i>INF</i>	<i>SUP</i>	<i>IADRES</i>	<i>NCED</i>
0	[a ₁₁ a ₂₂ a ₃₃]	[a ₁₁ a ₂₂ a ₃₃]	[1 2 3]	[1 2 3 4]
1	[a ₄₁ a ₄₂ a ₄₄]	[a ₁₄ a ₂₄ a ₄₄]	[1 2 4]	[4 7]
2	[a ₅₁ a ₅₂ a ₅₅]	[a ₁₅ a ₂₅ a ₅₅]	[1 2 5]	[7 10]
3	[a ₆₃ a ₆₆]	[a ₃₆ a ₆₆]	[3 6]	[10 12]

É importante observar que o armazenamento em duplicidade da diagonal não interfere muito na compactação, visto que a utilização de um mesmo vetor para indexar o armazenamento superior e inferior facilita as operações de multiplicação, transposição e solução de sistemas *LU*.

6.4 Resultados para os métodos de armazenamento

A seguir serão mostradas tabelas de valores mostrando os tempos de processamento comparando os diversos métodos de armazenamento.

As primeiras tabelas comparativas se referem aos desempenhos relacionados à comunicação para os vários esquemas de armazenamento. O número de processadores é representado pela variável N_p . Os valores nas tabelas são tempos dados em segundos e representam a soma total do tempo usado para enviar os vetores e dados para os nós, o tempo para receber e elaborar a solução e o tempo para executar a multiplicação matriz por vetor. Note-se que estes resultados foram obtidos para somente *uma* operação de produto matriz por vetor.

Já num processo iterativo, temos duas vezes o tempo de produto mais duas vezes o tempo de envio mais duas vezes o tempo de retorno do vetor de resultados,

dependendo ainda da função de comunicação utilizada, executados muitas vezes. Assim, as tabelas referentes à seção 6.4.2 mostram que, quando os produtos matriz por vetor são realizados repetidamente num processo iterativo de solução de um sistema de equações lineares, o overhead da paralelização, ou seja, o custo da paralelização com os tempos de envio e retorno de dados, chamadas bloqueantes, sincronização, somam tempos que não são explícitos nas tabelas da seção 6.4.1.

Então, torna-se importante a verificação do desempenho do armazenamento em um método iterativo que faça uso intensivo destes produtos, considerando-se a soma de tais valores de tempo. As tabelas da seção 6.4.2 foram elaboradas utilizando o Gradiente BiConjugado Estabilizado com pré-condicionamento diagonal. Os detalhes desta escolha e outros elementos importantes relativos a esta forma de processamento já foram descritos no capítulo 5.

Um outro aspecto importante que deve ser mencionado é o desempenho do hardware utilizado para obtenção destes resultados. Como foram utilizados dois clusters completamente diferentes, tanto nos tipos de processadores como nas redes de interconexão, este tópico também explora e comenta os desempenhos relativos ao processamento numérico oferecidos pelas máquinas.

Com isso foi possível criar um panorama mais amplo sobre a paralelização do produto matriz por vetor: o hardware utilizado no processamento numérico, o hardware de rede usado para obtenção da paralelização e o método de armazenamento.

6.4.1 Resultados parciais de comunicação para o produto matriz por vetor

As tabelas a seguir contêm os tempos obtidos com a realização de produtos matriz por vetor em quatro métodos diferentes de armazenamento: CRS, ICRS, CCS e RCS. As tabelas foram elaboradas a partir de diversas medidas de tempo, registrando diferentes etapas do processamento. Estas comparações foram feitas somente no cluster do GRUCAD. Os tempos medidos foram:

- TempoForm – Tempo necessário para realizar o armazenamento do arquivo de dados nos vetores de cada método. Aqui também é considerado o tempo para formar as variáveis auxiliares e a divisão do número de elementos da matriz entre os processadores.
- Tc1 – Tempo para enviar os vetores e variáveis necessárias à realização do produto matriz vetor do mestre para os processadores;
- Tc2 – Tempo para enviar o vetor solução, calculado pelos processadores para o mestre;
- TempoProd – Tempo gasto na realização do produto matriz-vetor;

Entretanto, na tabela final da avaliação de cada caso matricial, com a qual é construído o gráfico de colunas para uma visualização mais fácil dos resultados, não foi computado o valor de Tempoform, uma vez que, se o método de armazenamento fosse utilizado em um programa dedicado, as matrizes já estariam alocadas na memória, e este valor não mais seria necessário. Assim, a utilização deste valor na elaboração dos gráficos poderia mascarar os resultados envolvendo o processamento do produto matriz por vetor. A soma dos tempos Tc1, Tc2 e Tempoprod é mostrada na coluna TotalUtil em cada tabela.

Reunindo os resultados das colunas TempoUtil, foram elaborados os gráficos que ilustram os desempenhos globais de cada caso. Cada gráfico desta seção contém 5 séries de 4 colunas cada. Uma série significa um conjunto de resultados para um determinado número de processadores. Cada coluna equivale ao tempo de processamento de um método de compactação considerando um determinado número de processadores. Maior a coluna, maior o tempo de processamento.

6.4.1.1 Resultados para a Matriz 347829

TABELA 6.6. DESEMPENHOS PARA A MATRIZ 1, MÉTODO CRS

Método CRS - Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	13,1100	0,1205	0,0038	0,0326	13,2757	0,16
2	13,1600	0,7188	0,0254	0,0165	13,9231	0,76
3	13,1900	0,8985	0,0324	0,0106	14,1375	0,94
4	13,1300	1,0005	0,0417	0,0084	14,1881	1,05
5	13,0800	1,0663	0,0384	0,0065	14,2016	1,11

TABELA 6.7. DESEMPENHOS PARA A MATRIZ 1, MÉTODO ICRS.

Método ICRS - Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	13,1900	0,1205	0,0038	0,0320	13,3419	0,16
2	13,0800	0,6923	0,0251	0,0161	13,8208	0,73
3	13,1400	0,8854	0,0320	0,0108	14,0874	0,93
4	13,2200	0,9853	0,0454	0,0086	14,2840	1,04
5	13,2200	1,0705	0,0380	0,0066	14,3474	1,12

TABELA 6.8. DESEMPENHOS PARA A MATRIZ 1, MÉTODO RCS

Método RCS - Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	6,9100	0,1083	0,0039	0,0362	7,0618	0,15
2	6,9600	0,6170	0,0715	0,0197	7,6777	0,71
3	6,9800	0,7994	0,1236	0,0141	7,9254	0,94
4	6,9800	0,8788	0,1272	0,0113	8,0028	1,02
5	6,9900	0,9391	0,1600	0,0095	8,1147	1,11

TABELA 6.9. DESEMPENHOS PARA A MATRIZ 1, MÉTODO CCS

Método CCS - Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	12,8300	0,1218	0,0041	0,0392	12,9987	0,17
2	12,9200	0,7020	0,0718	0,0214	13,7206	0,80
3	13,0300	0,8985	0,1230	0,0150	14,0752	1,04
4	12,8100	0,9994	0,1155	0,0123	13,9649	1,13
5	12,8900	1,0690	0,1591	0,0100	14,1451	1,24

TABELA 6.10. COMPARAÇÕES PARA A MATRIZ 1 - TEMPOS ÚTEIS

Tempo (s)				
Np	CRS	CCS	ICRS	RCS
1	0,16	0,17	0,16	0,15
2	0,76	0,80	0,73	0,71
3	0,94	1,04	0,93	0,94
4	1,05	1,13	1,04	1,02
5	1,11	1,24	1,12	1,11

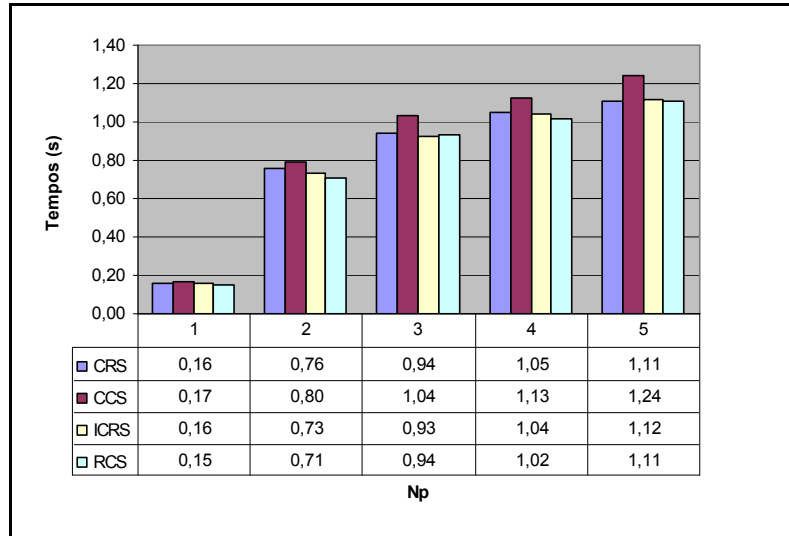


Fig. 6.1 - Comparações para a Matriz 1 - Tempos úteis.

6.4.1.2 Resultados para a Matriz 347832

TABELA 6.11. DESEMPENHOS PARA A MATRIZ 2, MÉTODO CRS

Método CRS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	54,4900	0,5258	0,0043	0,1148	55,1474	0,64
2	53,9700	2,8071	0,0252	0,0559	56,8814	2,89
3	53,5400	3,5428	0,0335	0,0379	57,2182	3,61
4	53,5200	3,9216	0,0362	0,0288	57,4186	3,99
5	53,5700	4,1408	0,0383	0,0023	57,6266	4,18

TABELA 6.12. DESEMPENHOS PARA A MATRIZ 2, MÉTODO ICRS

Método ICRS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	54,5500	0,5234	0,0043	0,1097	55,0967	0,64
2	53,7100	2,8873	0,0252	0,0556	56,7193	2,97
3	53,9500	3,5199	0,0324	0,0382	57,6176	3,59
4	53,6100	3,9077	0,0364	0,0286	57,6773	3,97
5	54,5500	4,1971	0,0384	0,0231	58,7413	4,26

TABELA 6.13. DESEMPENHOS PARA A MATRIZ 2, MÉTODO RCS

Método RCS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	27,4000	0,4302	0,0232	0,1503	28,0094	0,60
2	26,6800	2,3281	0,0721	0,0769	29,1638	2,48
3	25,9800	2,9058	0,1246	0,0518	29,0926	3,08
4	25,9200	3,2790	0,1153	0,0399	29,3990	3,43
5	26,1400	3,4159	0,1607	0,0321	29,7895	3,61

TABELA 6.14. DESEMPENHOS PARA A MATRIZ 2, MÉTODO CCS

Método CCS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	53,2500	0,5443	0,0141	0,1793	54,1486	0,74
2	53,7700	2,7814	0,0721	0,0829	56,9781	2,94
3	54,6600	3,5756	0,1237	0,0564	58,6318	3,76
4	53,0500	3,9340	0,1138	0,0436	57,2200	4,09
5	53,1100	4,1826	0,1748	0,0353	57,5452	4,39

TABELA 6.15. COMPARAÇÕES PARA A MATRIZ 2 - TEMPOS ÚTEIS

Np	Tempo (s)			
	CRS	CCS	ICRS	RCS
1	0,64	0,74	0,64	0,60
2	2,89	2,94	2,97	2,48
3	3,61	3,76	3,59	3,08
4	3,99	4,09	3,97	3,43
5	4,18	4,39	4,26	3,61

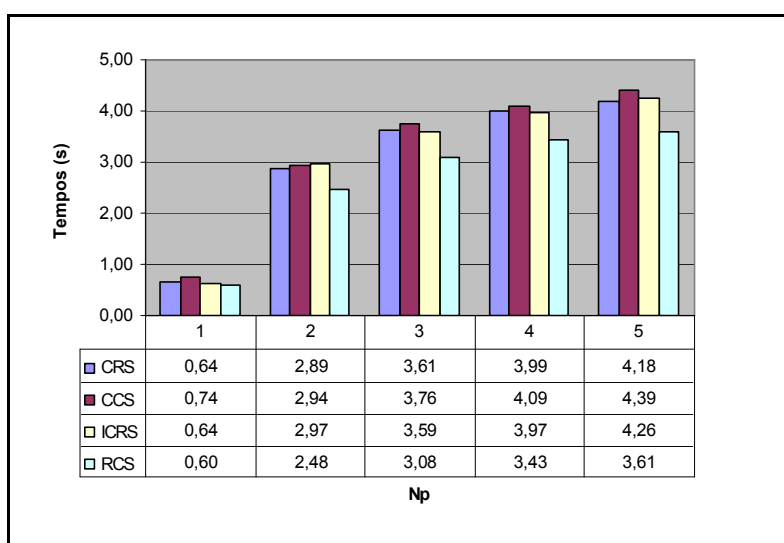


Fig. 6.2 - Comparações para a Matriz 2 - Tempos úteis.

6.4.1.3 Resultados para a Matriz 1009899

TABELA 6.16 DESEMPENHOS PARA A MATRIZ 3, MÉTODO CRS

Np	Tempoform	Método CRS				Total	TotalUtil
		Tc1	Tc2	Tempoprod	Total		
2	40,43	2,2321	7,46E-02	4,75E-02	42,798	2,35	
3	40,27	2,6374	9,35E-02	3,20E-02	43,8387	2,76	
4	40,02	2,9224	0,1319	2,39E-02	43,1363	3,08	
5	40,17	3,12	0,1111	1,91E-02	43,4244	3,25	

TABELA 6.17. DESEMPENHOS PARA A MATRIZ 3, MÉTODO ICRS

Np	Tempoform	Método ICRS				Total	TotalUtil
		Tc1	Tc2	Tempoprod	Total		
2	41,32	2,1052	9,57E-02	4,66E-02	43,5733	2,25	
3	41,13	2,6321	9,36E-02	3,20E-02	43,9264	2,76	
4	40,48	2,9534	0,1094	2,40E-02	43,5708	3,09	
5	41,25	3,1135	0,1106	1,93E-02	44,4955	3,24	

TABELA 6.18. DESEMPENHOS PARA A MATRIZ 3, MÉTODO RCS

Método RCS						
Np	Tempoform	Tc1	Tc2	Tempoprod	Total	TotalUtil
2	20,6	1,8105	0,2045	5,62E-02	22,6772	2,07
3	20,66	2,3356	0,3594	4,04E-02	23,3767	2,74
4	20,44	2,6025	0,3275	3,28E-02	23,3904	2,96
5	20,51	2,7626	0,4662	2,78E-02	23,7713	3,26

TABELA 6.19. DESEMPENHOS PARA A MATRIZ 3, MÉTODO CCS

Método CCS						
Np	Tempoform	Tc1	Tc2	Tempoprod	Total	TotalUtil
2	39,92	2,17	0,23	6,28E-02	42,38	2,46
3	39,84	2,80	0,36	4,49E-02	43,05	3,21
4	39,31	2,96	0,37	3,64E-02	42,68	3,36
5	39,60	3,17	0,47	3,10E-02	43,28	3,67

TABELA 6.20. COMPARAÇÕES PARA A MATRIZ 3 - TEMPOS ÚTEIS

Np	Tempo (s)			
	CRS	CCS	ICRS	RCS
2	2,35	2,46	2,25	2,07
3	2,76	3,21	2,76	2,74
4	3,08	3,36	3,09	2,96
5	3,25	3,67	3,24	3,26

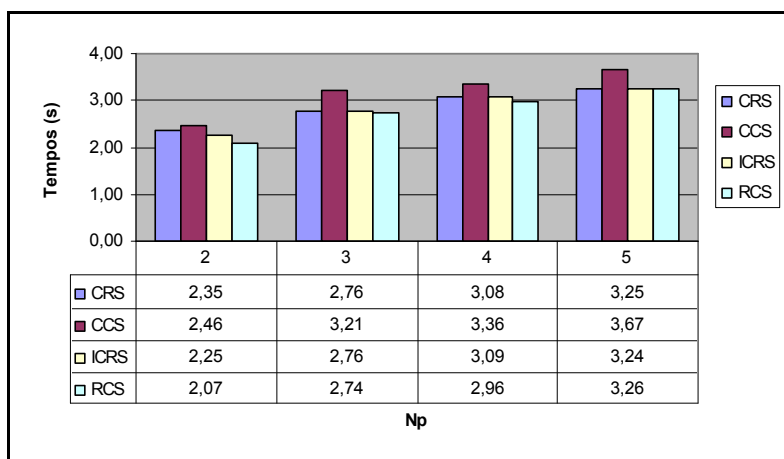


Fig. 6.3 - Comparações para a Matriz 3 - Tempos úteis.

6.4.1.4 Resultados para a Matriz 2163200

TABELA 6.21. DESEMPENHOS PARA A MATRIZ 4, MÉTODO CRS

Método CRS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	111,6300	1,1436	0,0331	0,2668	113,0832	1,44
2	112,5800	5,9182	0,1649	0,1386	118,8458	6,22
3	113,0300	7,4414	0,2537	0,0931	120,8218	7,79
4	111,5800	8,2547	0,2368	0,0706	120,3125	8,56
5	116,0900	8,6788	0,2451	0,0058	125,1681	8,93

TABELA 6.22. DESEMPENHOS PARA A MATRIZ 4, MÉTODO ICRS

Método ICRS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	119,0200	1,6255	0,3098	0,2815	120,8956	2,22
2	113,2600	5,7791	0,1657	0,1378	119,4586	6,08
3	117,1400	7,3700	0,2092	0,0998	125,0780	7,68
4	116,0700	8,1392	0,2266	0,0689	124,8662	8,43
5	113,1600	8,6695	0,2458	0,0552	125,0732	8,97

TABELA 6.23. DESEMPENHOS PARA A MATRIZ 4, MÉTODO RCS

Método RCS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	65,0000	1,0764	0,0304	0,3129	66,4285	1,42
2	62,2200	4,9529	0,4923	0,1750	67,8160	5,62
3	61,3100	6,3182	0,7855	0,1140	68,5898	7,22
4	58,0800	7,0211	0,6947	0,0876	65,8952	7,80
5	57,4600	7,5401	1,0680	0,0740	66,0911	8,68

TABELA 6.24. DESEMPENHOS PARA A MATRIZ 4, MÉTODO CCS

Método CCS – Tempo (s)						
Np	TempoForm	Tc1	Tc2	TempoProd	Total	TotalUtil
1	111,9300	1,1582	0,0460	0,3144	113,4739	1,52
2	109,4300	5,7616	0,4714	0,1708	115,8472	6,40
3	115,6600	7,4130	0,7873	0,1230	123,9818	8,32
4	112,7900	8,1561	0,6725	0,0981	121,8775	8,93
5	110,8500	8,7373	1,0119	0,0814	120,8298	9,83

TABELA 6.25. COMPARAÇÕES PARA A MATRIZ 4 - TEMPOS ÚTEIS

Np	Tempo (s)			
	CRS	CCS	ICRS	RCS
1	1,44	1,52	5,22	1,42
2	2,22	6,40	6,08	5,62
3	7,79	8,32	7,68	7,22
4	8,56	8,93	8,43	7,80
5	8,93	9,83	8,97	8,68

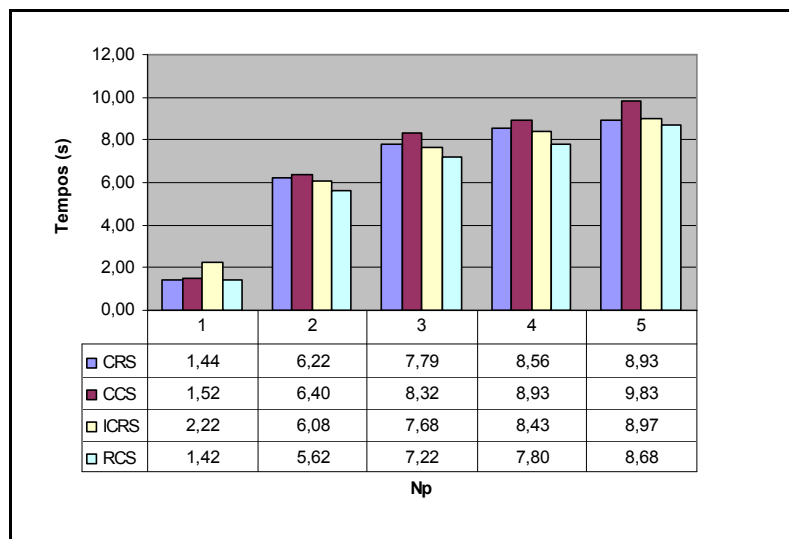


Fig. 6.4 - Comparações para a Matriz 4 - Tempos úteis.

Como pode ser observado nas Tabelas 6.6 a 6.24, o método RCS apresentou os menores tempos para os casos seqüenciais e paralelos na realização de um produto matriz por vetor. Isto ocorre porque o tempo T_{c1} , tempo para envio dos dados é menor para este método em relação às outras técnicas. Esta diferença não é grande, mas comparativamente se percebe que o método CCS apresentou, em todos os casos, o maior overhead.

6.4.2 Resultados considerando o método iterativo BiCGStab

Nas próximas subseções serão mostrados resultados para o cluster do GRUCAD e depois, quando possível, resultados obtidos para o cluster do L2EP. Isto porque não foi possível obter todos os resultados para o cluster da Universidade de Lille.

Os resultados foram organizados tal que, para cada matriz estudada, cada método de compactação recebeu uma tabela para mostrar a variação do número de processadores usado na simulação, o tempo utilizado com aquele número de processadores e o número de iterações que o BiCGStab levou para encontrar a solução. Teoricamente os métodos de compactação não deveriam influenciar no número de iterações do método de solução de equações. Mas devido a erros de arredondamento, inerentes ao uso da biblioteca de comunicação, os métodos que utilizam a função de comunicação *reduce* apresentam variações no número de iterações.

As tabelas para o cluster do GRUCAD mostram resultados para até 5 processadores, enquanto que as tabelas para o cluster do L2EP mostram resultados para até 4 computadores.

6.4.2.1 Resultados para a Matriz 347829, cluster do GRUCAD

TABELA 6.26 (a), (b), (c) e (d). DESEMPENHO DO BICGSTAB PARA A MATRIZ 347829 – GRUCAD.

(a) CRS			(c) ICRS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	22,03	77	1	21,67	77
2	26,30	77	2	26,21	77
3	28,51	77	3	28,26	77
4	29,56	77	4	29,27	77
5	29,71	77	5	29,90	77

(b) CCS			(d) RCS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	22,73	77	1	16,71	77
2	34,53	80	2	28,81	79
3	44,14	81	3	37,09	78
4	29,27	77	4	35,15	79
5	40,76	78	5	42,06	75

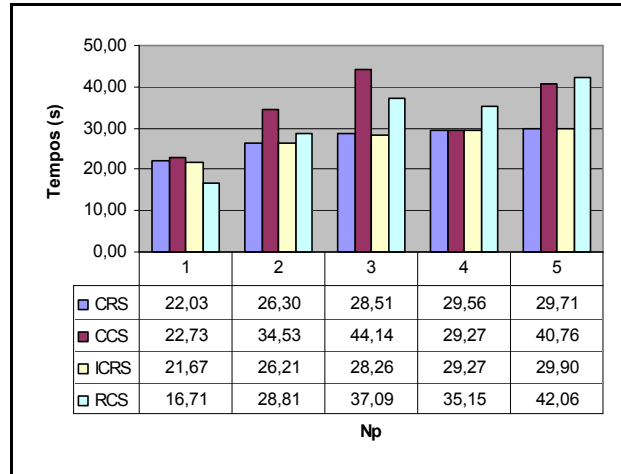


Fig. 6.5 - Comparações para a Matriz 347829 - BiCGStab.

6.4.2.2 Resultados para a Matriz 347832, cluster do GRUCAD

TABELA 6.27 (a), (b), (c) e (d). DESEMPENHO DO BICGSTAB PARA A MATRIZ 347832 – GRUCAD.

(a) CRS			(c) ICRS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	1386,12	5037	1	1354,67	5037
2	1260,15	5037	2	1247,00	5037
3	1218,59	5037	3	1211,63	5037
4	1207,21	5037	4	1195,04	5037
5	1193,90	5037	5	1190,67	5037

(b) CCS			(d) RCS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	1933,41	5037	1	2579,09	7291
2	3024,86	7711	2	2030,84	5225
3	3922,28	8541	3	3239,28	7084
4	3391,77	8296	4	3523,54	8640
5	4021,17	7996	5	2923,48	5785

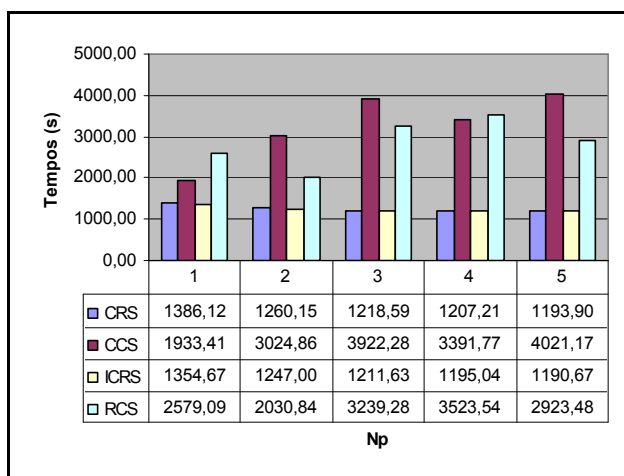


Fig. 6.6 - Comparações para a Matriz 347832 - BiCGStab.

6.4.2.3 Resultados para a Matriz 2163200, cluster do GRUCAD

TABELA 6.28 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 2163200 – GRUCAD.

(a) CRS			(c) ICRS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	338,61	273	1	-----	273
2	422,62	273	2	426,66	273
3	473,74	273	3	476,25	273
4	506,18	273	4	507,44	273
5	537,45	273	5	537,49	273

(b) CCS			(d) RCS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	368,38	273	1	333,54	276
2	613,11	284	2	565,10	275
3	807,90	277	3	757,03	270
4	765,06	272	4	759,46	284
5	1011,74	290	5	898,22	265

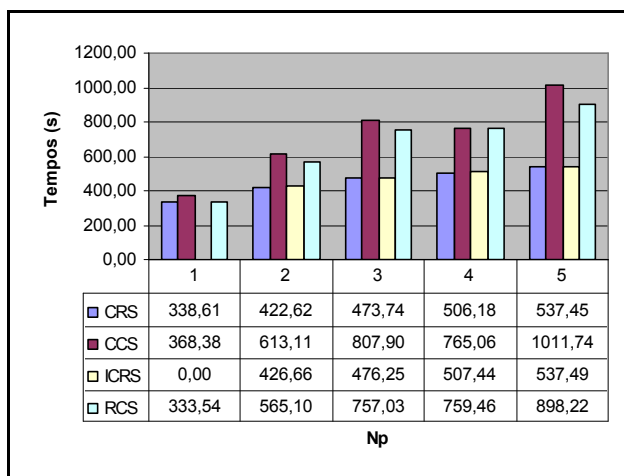


Fig. 6.7 - Comparações para a Matriz 2163200- BiCGStab.

6.4.2.4 Resultados para a Matriz 347829, cluster do L2EP

TABELA 6.29 (a), (b), (c) e (d). DESEMPENHO DO BiCGSTAB PARA A MATRIZ 347829 – L2EP.

(a) CRS			(c) ICRS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	39,46	77	1	50,14	77
2	37,48	77	2	43,73	77
3	37,65	77	3	42,34	77
4	37,72	77	4	41,60	77

(b) CCS			(d) RCS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	41,76	77	1	33,15	77
2	43,98	80	2	34,66	79
3	48,82	81	3	38,33	78
4	48,75	77	4	40,30	79

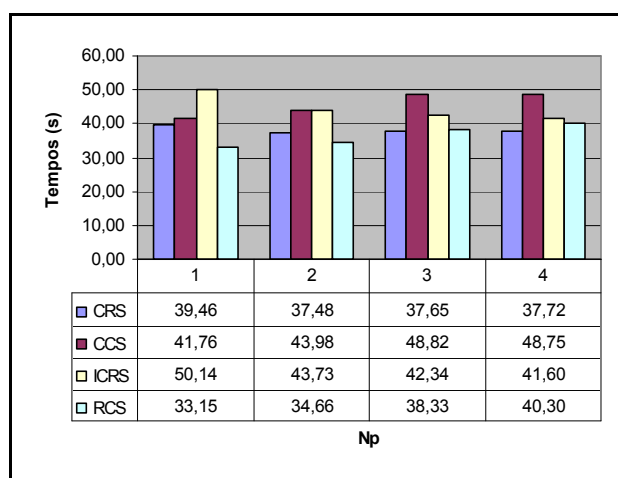


Fig. 6.8 - Comparações para a Matriz 347829- BiCGStab – L2EP.

6.4.2.5 Resultados para a Matriz 347832, cluster do L2EP

TABELA 6.30 (a), (b), (c) e (d). DESEMPENHO DO BICGSTAB PARA A MATRIZ 347832 – L2EP.

(a) CRS			(c) ICRS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	4720,82	5037	1	-----	-----
2	2764,28	5037	2	3003,96	5504
3	2134,04	5037	3	2133,10	5037
4	1835,91	5037	4	1807,22	5037

(b) CCS			(d) RCS		
Np	Tempo (s)	Iterações	Np	Tempo (s)	Iterações
1	6369,90	5037	1	6873,08	7291
2	5925,80	7711	2	3282,79	5225
3	5439,21	8541	3	4004,96	7084
4	4724,73	8296	4	4537,71	8640

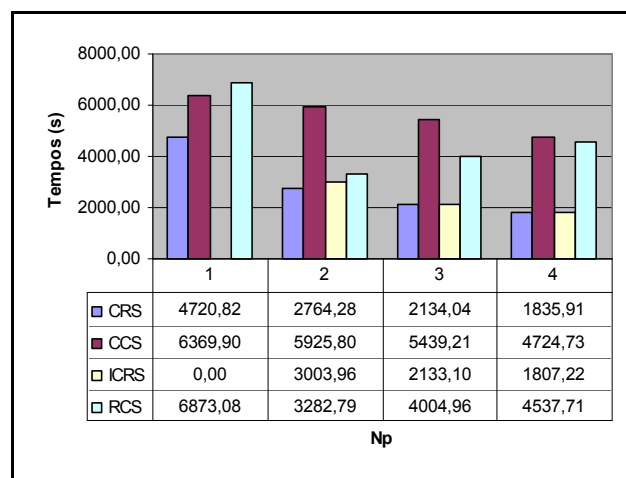


Fig. 6.9 - Comparações para a Matriz 347832- BiCGStab – L2EP.

6.4.2.6 Conclusões para os métodos de compactação e produtos matriz por vetor

Como pode ser observado a partir dos resultados apresentados nas Fig. 6.5 a 6.9, e nas tabelas referentes às mesmas, o método ICRS tem um desempenho global superior aos outros métodos de compactação, seguido pelo formato CRS. Observando também as figuras 6.5, 6.6, 6.7 e 6.9, que tratam de matrizes com grande número de elementos, nota-se que os métodos CRS e ICRS tornam-se equivalentes quando o número de elementos da matriz é muito alto. Colocando em números, a matriz 10098899 tem mais de 20×10^6 de elementos não nulos, enquanto que a matriz 2163200 possui mais de 57×10^6 de elementos não nulos.

Estas comparações são importantes, pois mostram, dentre os métodos de armazenamento estudados, qual método apresenta o melhor desempenho na paralelização do produto matriz por vetor.

As figuras a seguir ilustram comparações entre métodos nos clusters do GRUCAD e do L2EP. Nestas ilustrações é possível observar o desempenho do processamento numérico que cada máquina é capaz de realizar, o desempenho das redes de comunicação que impulsionam cada uma, e também o desempenho dos métodos de armazenamento.

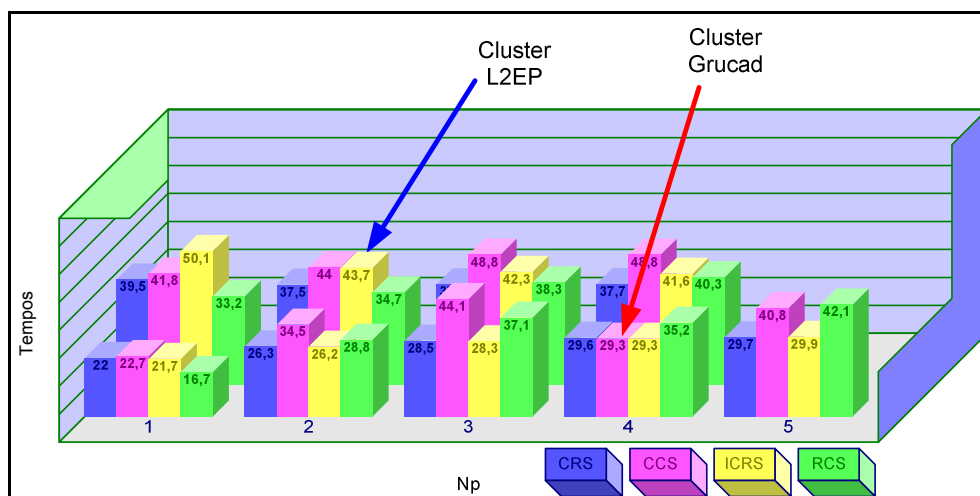


Fig. 6.10 - Comparações GRUCAD x L2EP, Matriz 347829, BiCGStab.

Em relação ao desempenho numérico dos processadores, é flagrante a diferença entre os processadores Pentium 4 3GHz e Dual Xeon 1.8GHz. O gráfico de colunas para $N_p=1$ da Fig. 6.10 mostra que os processadores Pentium 4 chegam a apresentar duas vezes mais capacidade de processamento. Na Fig. 6.11, com uma matriz mais densa, a capacidade de processamento sobe para mais de três vezes.

Em relação ao desempenho da rede, nota-se que a rede Myrinet tem desempenho superior à rede Gigabit. Isto pode ser facilmente observado na Fig. 6.11 abaixo, através das colunas de tempo para o método CCS. Existe um decréscimo no tempo de cálculo do produto matriz por vetor quando o número de processadores aumenta. Quer dizer, a rede Myrinet proporciona ganho com a paralelização do produto matriz por vetor. O mesmo acontece para os métodos CRS e ICRS, mas numa proporção diferente. Já a rede Gigabit não consegue proporcionar ganhos com sua arquitetura.

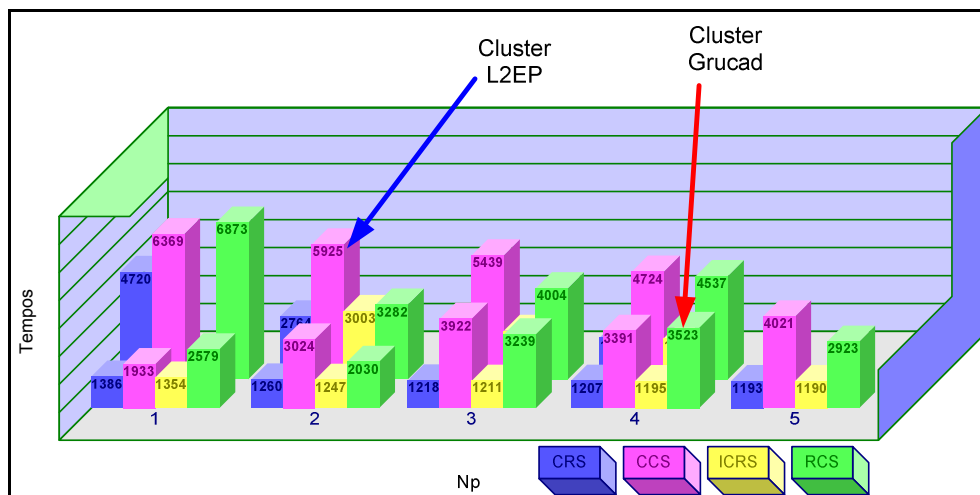


Fig. 6.11 - Comparações GRUCAD x L2EP, Matriz 347832, BiCGStab.

Comparando o último aspecto, os desempenhos dos métodos de armazenamento, nota-se a grande diferença dos métodos ICRS e CRS em relação aos outros dois em ambas as arquiteturas de rede, tanto em matrizes bastante esparsas como em matrizes mais densas. Mesmo considerando as capacidades de processamento proporcionadas pelos processadores diferentes, os métodos CCS e RCS apresentam tanto maiores necessidades de processamento numérico como maiores *overheads* na paralelização.

6.5 Resultados para o pré-condicionamento FLUI

A avaliação do pré-condicionador com fatoração LU por blocos com interseção (FLUI) desenvolvido nesta tese é feito nesta seção. Como comentado anteriormente, o cluster é uma máquina de granulosidade grossa e este trabalho o utiliza para paralelização em pequena escala, tal que resultados com ganhos na paralelização tendem a ser difíceis de serem obtidos. Desta forma, o pré-condicionador FLUI é uma tentativa no sentido de dividir o processamento entre as diversas máquinas do cluster, mantendo a qualidade do pré-condicionamento LU . Entretanto, quando o pré-condicionador é quebrado, dividido entre as máquinas, acontece o desacoplamento do sistema. A interseção procura diminuir este desacoplamento e é dada através de um valor de porcentagem, que significa quantas linhas do bloco imediatamente anterior serão consideradas no cálculo do pré-condicionador destinado a um processador.

Para matrizes pequenas é possível observar a partir de qual número de processadores ocorreu o desacoplamento do sistema. Isto é facilmente percebido através dos gráficos de variação da porcentagem de interseção pelo número de processadores, gráficos que mostram a evolução do número de iterações em função daquelas duas variáveis. Também através destes gráficos é possível observar até qual porcentagem ocorre a variação do número de iterações. Nos casos estudados se observou que a partir de um dado valor de porcentagem o número de iterações permanece constante. Com isso foi possível limitar o valor máximo da porcentagem de interseção nas análises de ganho de tempo com paralelização, criando uma configuração da porcentagem para os testes de tempo.

A seção que descreve os resultados para o pré-condicionador FLUI utilizou apenas os exemplos executados no cluster do GRUCAD, uma vez que se trata de um estudo da observação de ganho em relação ao número de iterações. Além disso, os algoritmos de resolução de sistemas de equações utilizaram algumas rotinas implementadas e correntemente utilizadas nos programas do FEECAD. Tratam-se das rotinas de fatoração LU e da técnica de armazenamento esparsa. Com isso serão mostrados resultados apenas para o método de compactação de matrizes esparsas RCS.

Como uma última observação preliminar, também foram observadas variações do

número de iterações em função do erro de convergência. Como esta variação é esperada uma vez que ocorre a relaxação da norma residual, serão mostrados apenas alguns resultados para demonstrar sua existência também com o pré-condicionador FLUI.

As tabelas que contém os resultados destas observações foram construídas tal que a coluna mais à esquerda mostra a porcentagem de interseção entre os blocos, aqui variando de 0% a 15%, uma vez que experimentalmente se observou que a variação do número de iterações deixa de ocorrer a partir de 15% de interseção em quase todos os casos testados. Em separado é mostrado o número de iterações para o algoritmo executado de forma seqüencial. Nestas tabelas são mostrados em negrito os números de iterações relevantes ao exemplo, quer dizer, quando ocorrem números de iterações menores que no caso seqüencial ou para interseção igual a 0%. A observação da variação do número de iterações com a variação da porcentagem dos blocos de 0% a 100% será sempre mostrada através de gráficos de dispersão, que ilustram com maior objetividade o comportamento estudado.

O pré-condicionamento FLUI foi testado nos quatro métodos iterativos descritos no capítulo 3. São eles: o Gradiente Conjugado (CG), o Gradiente BiConjugado (BiCG), o Gradiente BiConjugado Estabilizado (BiCGStab) e o Gradiente Conjugado Quadrado (CGS).

Para cada um destes métodos foi elaborada uma tabela como descrito acima, tal que os valores de iterações encontrados foram anotados. Entretanto, como descrito nas características dos métodos iterativos no capítulo 3, o método CGS sofre com o seu “operador de contração quadrático”, e, em alguns casos, a solução do sistema não foi encontrada com o pré-condicionador proposto, mesmo com um número extremamente alto de iterações (por exemplo, um número de iterações igual à ordem da matriz). Nestes casos a tabela para o método CGS não foi elaborada.

6.5.1 Observação do desacoplamento dos sistemas

Em sistemas pequenos, na ordem de algumas dezenas de mil variáveis, pode-se acompanhar a influência do pré-condicionador FLUI no desacoplamento dos sistemas. Isto significa que até um determinado número de máquinas, e dependendo também da ordem do sistema (quanto maior o número de variáveis, maior o número de máquinas necessárias para observar o fenômeno) o número de iterações não sofre uma variação muito grande. Entretanto, passando-se este valor, o número de iterações necessárias à convergência aumenta sobremaneira, indicando que houve o desacoplamento.

A figura a seguir, construída para um sistema com somente 1080 variáveis, ilustra a situação de desacoplamento do sistema.

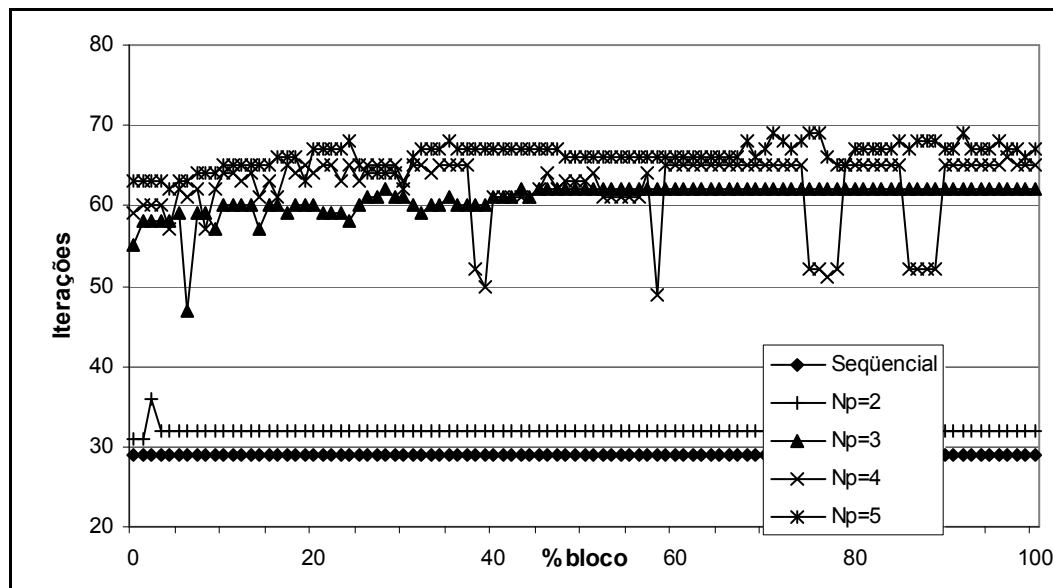


Fig. 6.12 – Matriz 6, 1080 variáveis, pré-condicionador FLUI e desacoplamento.

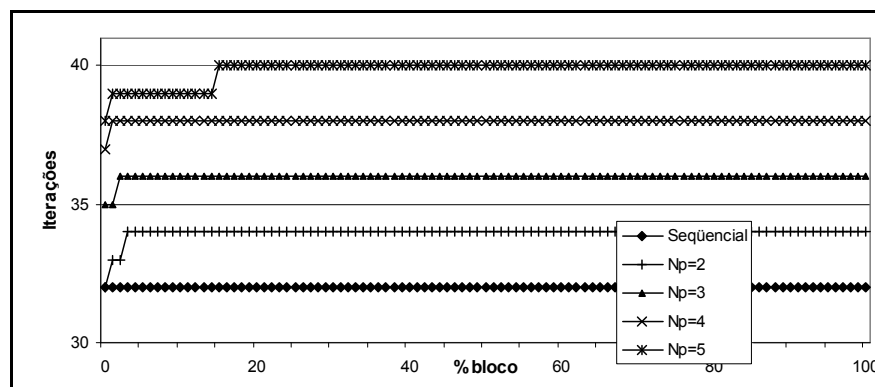
Nesta figura, para a resolução na forma seqüencial, foram necessárias 29 iterações para alcançar a convergência. Com dois processadores o número de iterações ficou próximo, 32, mas quando o número de máquinas subiu para 3 ou mais processadores o número de iterações sofreu uma mudança drástica, ficando em torno de 60 iterações. Neste caso, o desacoplamento ocorreu quando da inclusão da terceira máquina.

6.5.2 Observação da variação do número de iterações com o erro de convergência

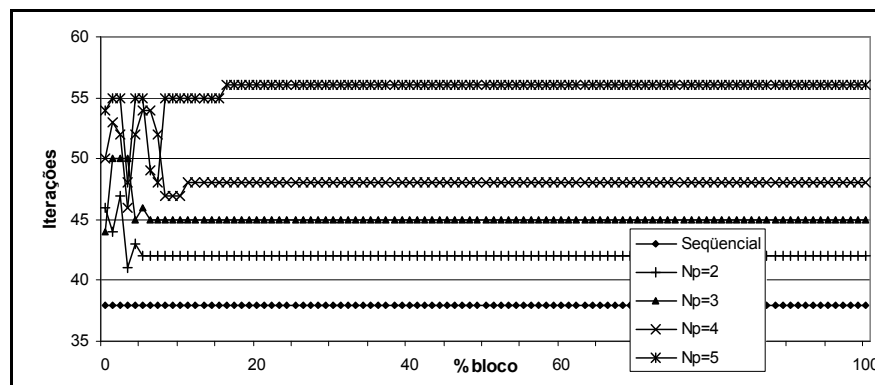
Os gráficos a seguir ilustram a variação do número de iterações em função do erro de convergência para um sistema com 219539 variáveis. O sistema foi resolvido com o BiCG com erros de convergência iguais a 1×10^{-4} , 1×10^{-5} e 1×10^{-6} .

Observe-se que a variação existe para todos os processadores. Executando-se na forma seqüencial para erro igual a 1×10^{-4} o número de iterações é igual a 32, com erro igual a 1×10^{-5} é igual a 38 e com erro igual 1×10^{-6} é igual a 51. Acompanhando os gráficos se pode observar a variação para outros números de processadores.

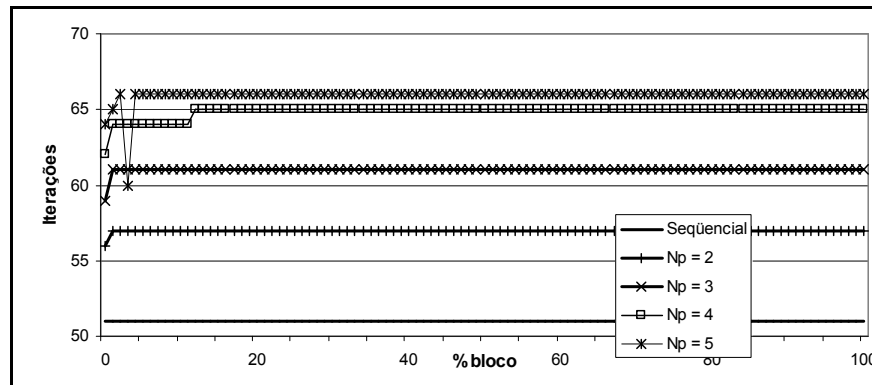
Naturalmente, em se mudando o método de resolução para o mesmo sistema de equações o número de iterações também muda, como será mostrado logo a seguir.



(a)



(b)



(c)

Fig. 6.13 – Variação do número de iterações em função do erro de convergência. Matriz 7, BiCG, pré-condicionador FLUI: (a) erro igual a 1×10^{-4} , (b) erro igual a 1×10^{-5} e (c) erro igual a 1×10^{-6} .

6.5.3 Resultados para o pré-condicionador FLUI

Seguem agora os resultados do comportamento do pré-condicionador FLUI para uma série de sistemas de equações resolvidos pelos métodos de resolução já descritos. As tabelas apresentadas mais adiante foram calculadas para um erro de convergência igual a 1×10^{-5} . Como já foi comentado, para outros valores de erro as tabelas e gráficos têm outros valores.

Neles foram paralelizados os produtos matriz por vetor nos dois métodos de resolução de sistemas de equações, bem como a montagem e aplicação do pré-condicionador FLUI. Quando a interseção é igual a 0% o pré-condicionamento é chamado aqui simplesmente de pré-condicionador *LU* por blocos e equivale ao pré-condicionador proposto em [40].

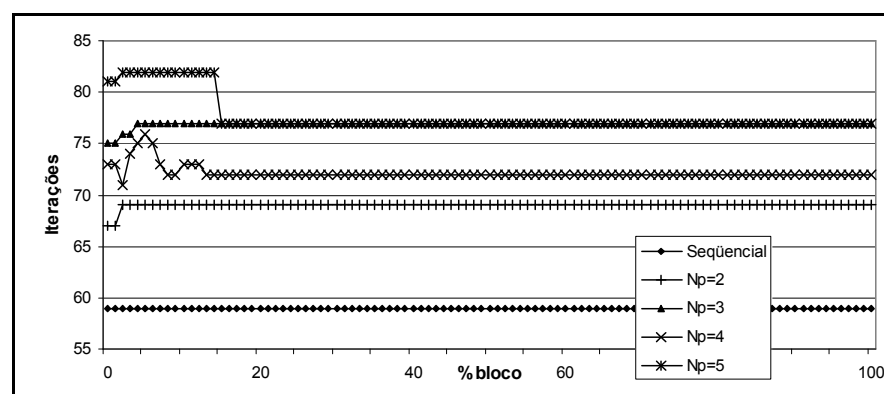
6.5.3.1 Resultados para a Matriz 219539

Para este sistema o CGS não encontrou convergência. Assim, a seguir estão as tabelas com as variações de 0% a 15% de interseção, e na figura 6.14 os gráficos com as variações de 0% a 100%.

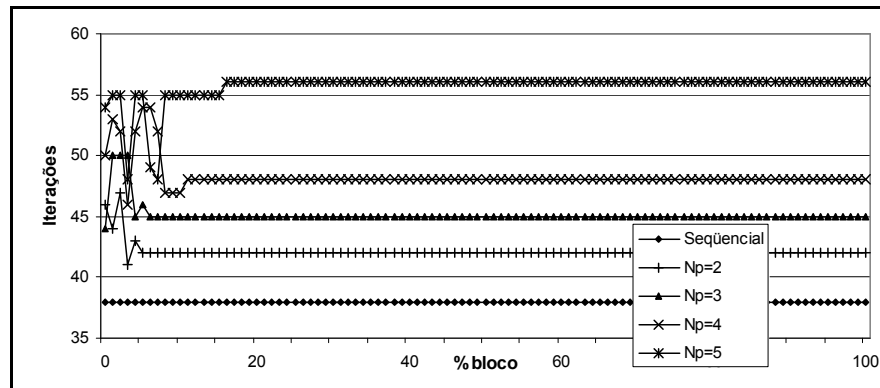
TABELA 6.31. NÚMERO DE ITERAÇÕES PARA O SISTEMA 219539

%	CG				BiCG				BiCGStab			
	Seqüencial = 59				Seqüencial = 38				Seqüencial = 29			
	Np				Np				Np			
	2	3	4	5	2	3	4	5	2	3	4	5
0	67	75	73	81	46	44	50	54	31	28	26	27
1	67	75	73	81	44	50	53	55	29	26	27	28
2	69	76	71	82	47	50	52	55	29	28	26	27
3	69	76	74	82	41	50	46	48	29	28	26	27
4	69	77	75	82	43	45	52	55	31	27	26	27
5	69	77	76	82	42	46	54	55	31	27	26	27
6	69	77	75	82	42	45	54	49	31	27	29	27
7	69	77	73	82	42	45	52	48	29	27	29	27
8	69	77	72	82	42	45	47	55	29	27	29	27
9	69	77	72	82	42	45	47	55	30	27	29	27
10	69	77	73	82	42	45	47	55	29	27	27	27
11	69	77	73	82	42	45	48	55	29	28	27	27
12	69	77	73	82	42	45	48	55	30	27	26	27
13	69	77	72	82	42	45	48	55	29	27	26	27
14	69	77	72	82	42	45	48	55	29	27	26	27
15	69	77	72	77	42	45	48	55	29	27	26	27

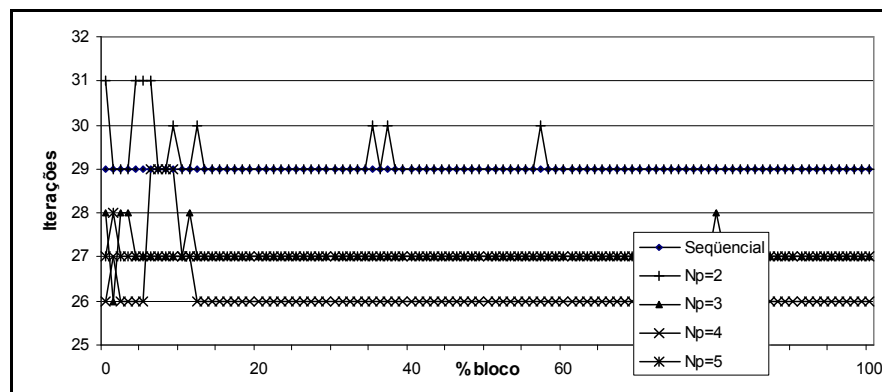
Primeiramente, pode-se observar que os resultados correspondem às informações encontradas na literatura em relação ao número de iterações apresentados pelos métodos iterativos. O CG apresentou o maior número de iterações e o BiCGStab o menor. Comparando o CG e o BiCG encontramos que os números de iterações são maiores para o CG em até duas vezes (82/41) para qualquer número de processadores.



(a)



(b)



(c)

Fig. 6.14 – Variação do número de iterações para a Matriz 219539, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.

Quando o valor de interseção é igual a 0%, temos uma configuração próxima daquela relatada em [73]. Próxima, pois a referida referência trata unicamente de matrizes e pré-condicionamento simétricos. Considerando este valor como referência, nas tabelas são mostrados em **negrito** os valores de iterações menores que aqueles encontrados para 0% de interseção. Isto representa as condições onde o pré-condicionador proposto (FLUI) mostrou ganho em relação ao pré-condicionador desenvolvido em [73].

Assim, numa comparação quantitativa, a utilização do método BiCGStab na resolução até mesmo de sistemas simétricos deve ser considerada preferencial, pois a utilização do pré-condicionador FLUI com o BiCGStab apresenta um desempenho muito superior ao apresentado pela referência, numa razão de no mínimo 27/59 iterações.

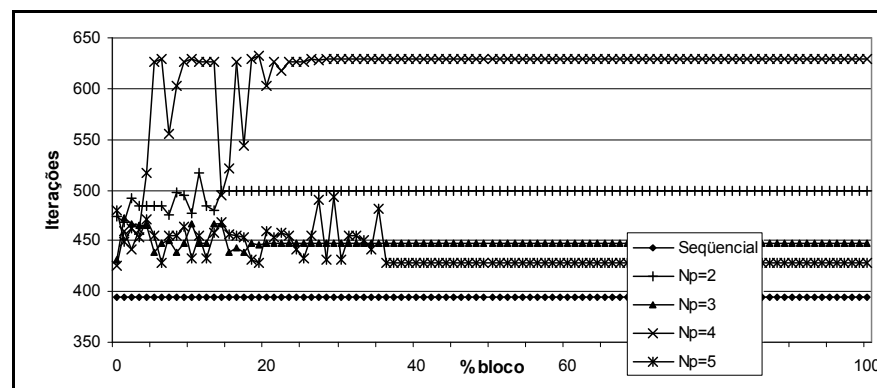
Além disso, na figura 6.14c, pode-se verificar que, para o sistema em questão, a utilização em paralelo levou a um número de iterações menor que o caso seqüencial.

6.5.3.2 Resultados para a Matriz 347832

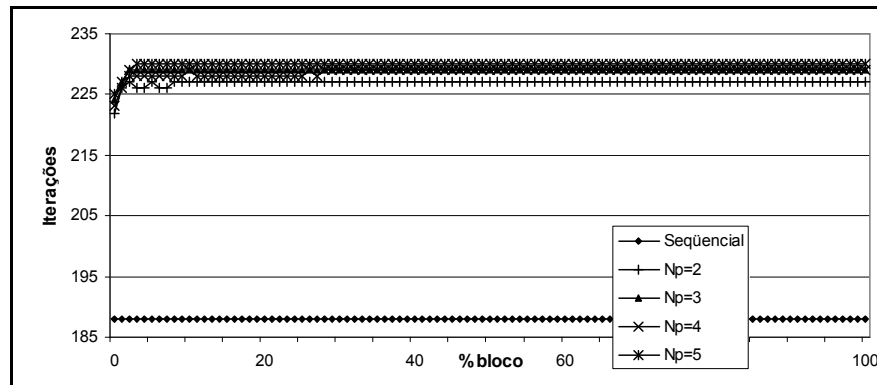
As tabelas e figuras a seguir foram obtidas com a análise do sistema 347832. Novamente neste sistema o método CGS não convergiu.

TABELA 6.32. NÚMERO DE ITERAÇÕES PARA O SISTEMA 347832

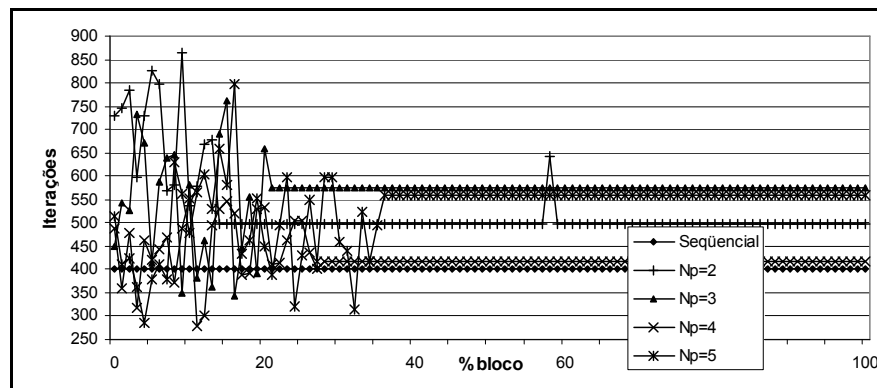
%	CG				BiCG				BiCGStab			
	Seqüencial = 394				Seqüencial = 188				Seqüencial = 401			
	Np				Np				Np			
	2	3	4	5	2	3	4	5	2	3	4	5
0	474	432	426	480	222	224	223	225	731	450	489	513
1	468	472	459	449	226	227	226	227	747	543	361	412
2	492	467	442	462	227	229	228	229	785	528	477	424
3	485	465	461	454	226	229	228	230	599	734	316	362
4	484	465	517	471	226	229	228	230	731	672	462	285
5	485	439	626	455	227	229	228	230	825	415	422	378
6	485	447	629	429	226	229	228	230	798	589	443	410
7	476	451	556	455	226	229	228	230	570	640	470	378
8	498	438	603	455	227	229	228	230	580	646	373	631
9	495	447	627	464	227	229	228	230	866	349	488	563
10	477	467	629	433	227	229	229	230	537	583	554	480
11	517	447	626	455	227	229	228	230	578	383	278	566
12	485	447	627	433	227	229	228	230	668	462	300	605
13	480	467	627	458	227	229	228	230	677	363	496	531
14	500	467	495	468	227	229	228	230	499	691	529	660
15	500	439	521	456	227	229	228	230	499	763	546	583



(a)



(b)



(c)

Fig. 6.15 – Variação do número de iterações para a Matriz 347832, pré -condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.

Como pode ser observado nas tabelas e figuras, o BiCG apresentou um número de iterações menor que aquele encontrado para o CG para qualquer número de processadores. O número de iterações para o BiCG é quase sempre a metade daquele encontrado para o CG em praticamente todos os valores da tabela.

Já o algoritmo BiCGStab, embora tenha obtido mais iterações que o BiCG, mostrou em relação ao seu algoritmo seqüencial, vários valores abaixo daquele encontrado para 0% de interseção e também para a resolução seqüencial.

6.5.3.3 Resultados para a Matriz 2000

Seguem as tabelas e figuras com os resultados para a Matriz 2000. Esta matriz é originária de uma expressão matemática com uma esparsidade muito baixa. Neste caso todos os métodos apresentaram resultados e, como pode ser observado nas tabelas, o CG novamente apresentou o maior número de iterações, o BiCG valores de aproximadamente a metade do número de iterações do CG e os outros dois métodos, BiCGStab e CGS, valores intermediários.

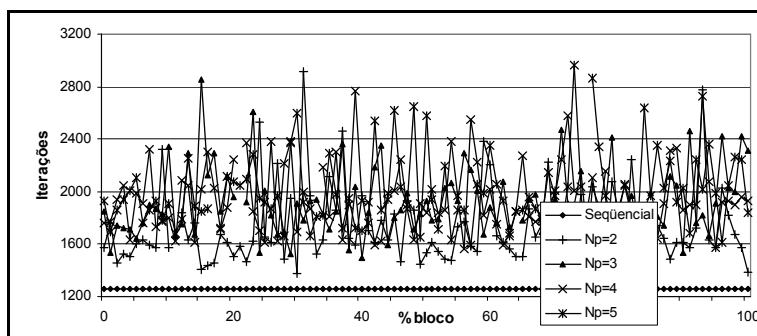
Pode-se observar nestas tabelas diversos valores em negrito onde o número de iterações é menor do que o valor apresentado em 0%. Mas, ao contrário do que pôde ser observado nas matrizes anteriores, nesta matriz não ocorreram para os casos paralelos números de iteração menores do que aquele encontrado para a resolução seqüencial.

TABELA 6.33. NÚMERO DE ITERAÇÕES PARA O SISTEMA 2000 CG E BiCG

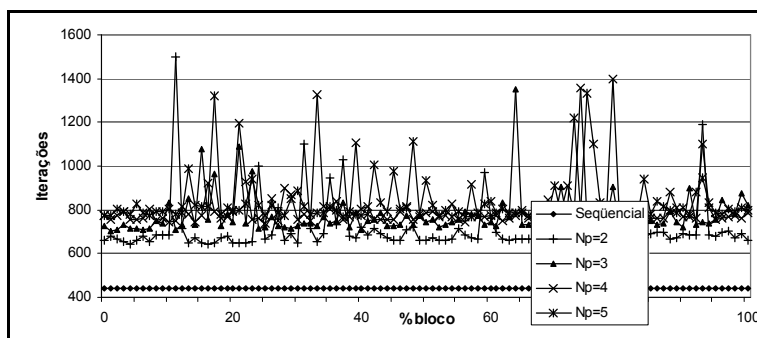
%	CG				BiCG			
	Seqüencial = 1256				Seqüencial = 442			
	Np				Np			
	2	3	4	5	2	3	4	5
0	1576	1850	1766	1927	662	729	774	776
1	1766	1531	1767	1694	681	703	757	777
2	1461	1739	1938	1857	666	710	773	802
3	1530	1718	2050	1939	657	733	795	791
4	1509	1715	1635	2009	646	715	756	781
5	1604	1640	1993	2105	662	715	758	826
6	1632	1759	1911	1763	677	709	760	785
7	1593	1898	2326	1863	653	712	802	774
8	1579	1871	1734	1931	687	749	752	791
9	2320	1826	1770	1784	686	737	761	792
10	1576	2339	1803	1911	683	836	742	802
11	----	1660	1625	1681	1500	708	749	782
12	1836	1752	2084	1791	717	730	813	785
13	1632	2294	2046	2255	647	851	778	986
14	1762	1668	1612	1894	676	731	745	823
15	1409	2853	2018	1848	648	1077	776	813

TABELA 6.34. NÚMERO DE ITERAÇÕES PARA O SISTEMA 2000 BiCGStab E CGS

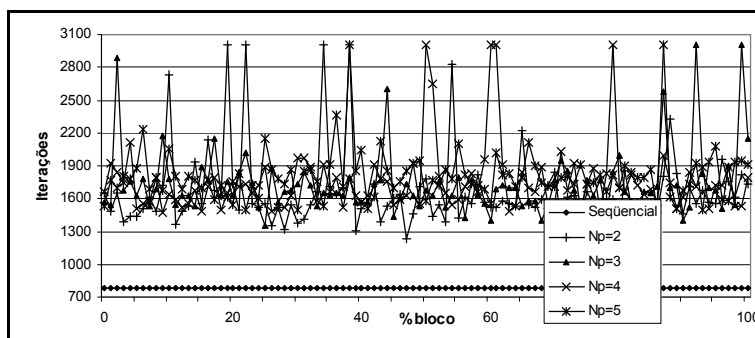
%	BiCGStab				CGS			
	Seqüencial = 782				Seqüencial = 735			
	Np				Np			
	2	3	4	5	2	3	4	5
0	1650	1572	1667	1526	1453	1249	1193	1133
1	1489	1543	1925	1766	1650	1233	1166	1206
2	1646	2887	1836	1693	1471	1169	2018	1323
3	1391	1672	1745	1803	944	1257	1826	1536
4	1441	1757	2116	1778	1102	1064	1125	1138
5	1432	1628	1510	1874	963	1065	1178	1008
6	1509	1780	1561	2230	1046	1203	2006	1169
7	1615	1530	1564	1692	891	1070	1513	1180
8	1487	1687	1795	1795	1021	----	1489	1161
9	1725	2174	1469	1679	1051	987	1231	1241
10	2728	1781	1643	2058	1011	1774	1589	1240
11	1367	1541	1597	1804	948	1105	1156	1259
12	1482	1622	1521	1685	1088	1236	1303	1064
13	1532	1629	1556	1804	993	1093	1191	1386
14	1937	1536	1672	1773	1235	1229	1070	1135
15	1551	1889	1485	1693	903	1085	1137	1145



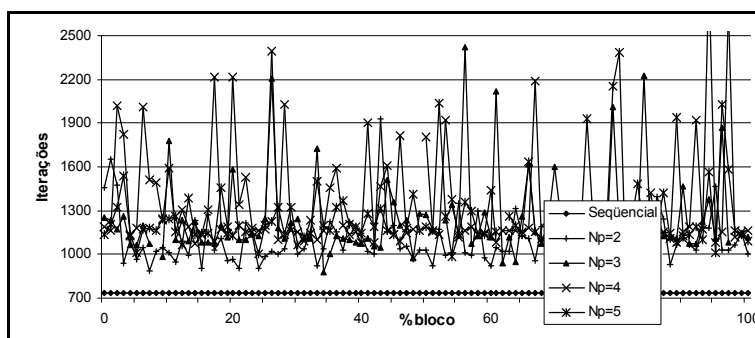
(a)



(b)



(c)



(d)

Fig. 6.16 – Variação do número de iterações para a Matriz 2000, pré -condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG, (c) BiCGStab e (d) CGS.

6.5.3.4 Observações para o pré-condicionador FLUI

Como pode ser observado nas tabelas dos diversos resultados, a utilização da técnica de divisão da matriz de pré-condicionamento, para uma dada porcentagem, leva a um número de iterações maior que o seqüencial. Isto é normal uma vez que quanto maior o número de processadores, maior é o enfraquecimento do pré-condicionador, que é calculado cada vez com uma menor quantidade de linhas da matriz.

Com 0% de interseção temos o mesmo comportamento relatado em [73]. Com o aumento do valor de porcentagem de interseção observou-se uma diminuição do número de iterações em relação ao caso sem interseção. Isto é prontamente observado nas tabelas nos valores em **negrito**. Além disso, também foram observadas porcentagens onde o número de iterações encontrado foi menor que para o caso seqüencial. Isto ocorre particularmente para o BiCGStab. Estes resultados podem ser observados nas tabelas da Matriz 219539 e da Matriz 347832.

Também pode ser relatada uma coerência dos resultados com as informações apresentadas na literatura. Aqui foi verificada a superioridade do BiCG em relação ao CG no que tange o menor número de iterações, a dificuldade de convergência do CGS, e a estabilidade do BiCGStab, encontrando ora com menos iterações que o BiCG, ora com mais, a solução do sistema sem o uso da transposta da matriz, isto é, menos processamento.

Comparando-se o pré-condicionador FLUI com o pré-condicionador encontrado em [73] se observa que o primeiro mostra em várias ocasiões um menor número de iterações, além do que pode ser aplicado a algoritmos utilizados em sistemas não simétricos.

6.6 Resultados de desempenho no tempo

Uma vez que já se mostrou o comportamento do pré-condicionador FLUI em relação ao número de iterações, os resultados que agora serão mostrados refletem os tempos de processamento.

Como pode ser observado nas figuras 6.14 a 6.16, depois de certa porcentagem os números de iterações permanecem constantes. Assim, para a elaboração das tabelas de tempos e as figuras que serão apresentadas logo a seguir, foi considerada a variação de 0% a 15%. Os valores de tempo mostrados em **negrito** agora refletem os tempos menores que aqueles registrados para o processamento sem interseção.

Em **negrito** e *itálico* estão indicados os registros de tempo menores que o tempo para o processamento sequencial.

6.6.1 Resultados para a Matriz 219539

Seguem as tabelas e figuras para a medida dos tempos para a Matriz 219539. Seguindo a mesma tendência encontrada para o número de iterações, não houveram ganhos de tempo com a paralelização neste caso de estudo.

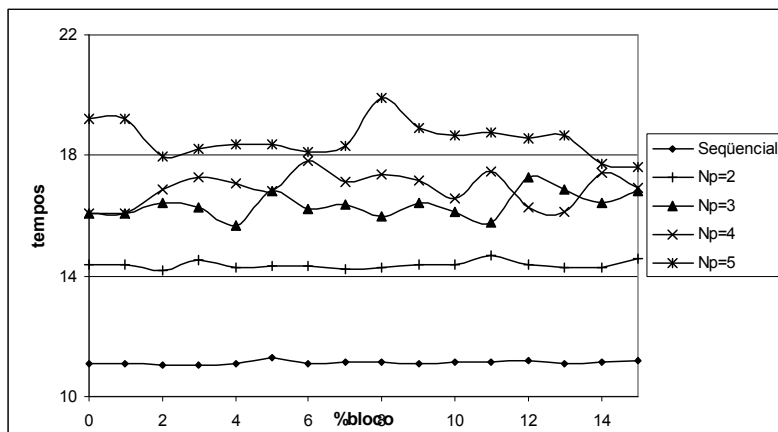
Entretanto podemos notar que o BiCGStab apresentou um desempenho no tempo muito parecido com o método CG. Esta é uma observação importante, pois esta matriz é proveniente do método de Elementos Finitos, e o BiCGStab pode ser utilizado para resolver tanto matrizes simétricas como assimétricas.

Além disso, o processamento com interseção, também acompanhando a tendência do número de iterações, mostrou diversas condições onde o tempo com interseção é menor que o tempo sem interseção.

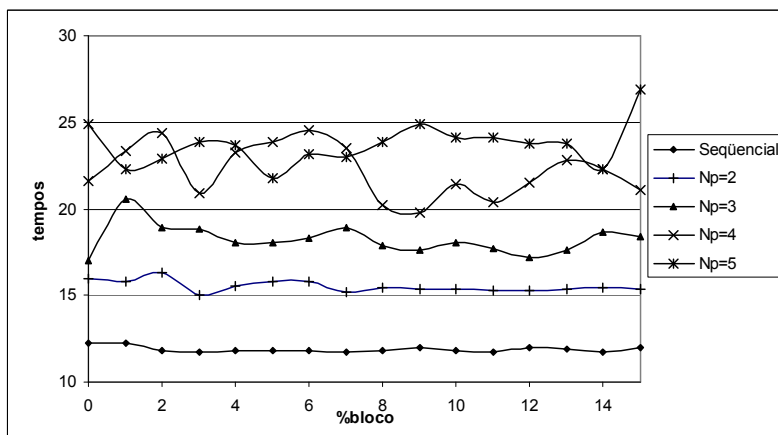
TABELA 6.35. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 219539

%	CG				BiCG				BiCGStab			
	Seqüencial = 11,2 s				Seqüencial = 11,90 s				Seqüencial = 12,2 s			
	Np				Np				Np			
	2	3	4	5	2	3	4	5	2	3	4	5
0	14,39	16,09	16,08	19,23	15,98	17,05	21,58	24,89	14,63	15,14	15,88	15,39
1	14,39	16,09	16,08	19,23	15,77	20,53	23,37	22,31	14,28	14,02	16,52	15,61
2	14,19	16,43	16,88	17,96	16,34	18,90	24,40	22,86	14,46	15,16	15,62	15,79
3	14,52	16,27	17,26	18,23	15,05	18,84	20,95	23,81	14,32	15,34	15,25	15,64
4	14,26	15,69	17,09	18,39	15,50	18,02	23,21	23,65	14,83	14,67	15,33	14,95
5	14,31	16,84	16,83	18,37	15,77	18,03	23,89	21,77	14,78	14,67	16,01	16,25
6	14,32	16,20	17,82	18,14	15,84	18,29	24,53	23,20	14,73	14,41	15,41	14,14
7	14,25	16,36	17,13	18,32	15,23	18,94	23,48	23,01	14,29	15,47	15,83	15,29
8	14,28	15,97	17,38	19,93	15,49	17,91	20,19	23,86	14,42	15,24	15,71	13,88
9	14,37	16,41	17,17	18,89	15,33	17,61	19,77	24,93	14,58	14,72	15,68	13,91
10	14,37	16,14	16,55	18,66	15,39	18,06	21,44	24,10	14,40	14,51	16,19	14,15
11	14,66	15,80	17,48	18,74	15,30	17,74	20,36	24,08	14,47	14,58	15,73	14,32
12	14,36	17,25	16,29	18,54	15,26	17,20	21,49	23,73	14,56	14,55	15,39	14,83
13	14,29	16,89	16,14	18,68	15,37	17,65	22,79	23,80	14,42	14,62	14,95	15,19
14	14,28	16,43	17,40	17,74	15,47	18,69	22,33	22,32	14,45	14,86	15,06	14,49
15	14,59	16,84	16,93	17,63	15,41	18,40	21,05	26,85	14,49	15,66	15,68	14,55

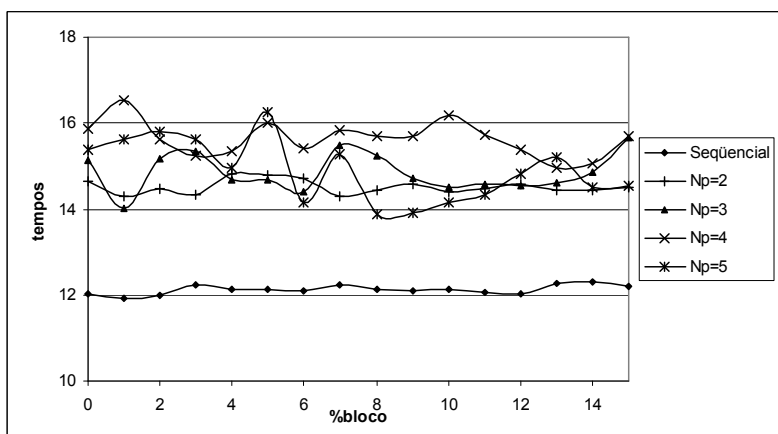
A seguir, os gráficos para a variação do tempo de processamento.



(a)



(b)



(c)

Fig. 6.17 – Variação do tempo de processamento para a Matriz 219539, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab

6.6.2 Resultados para a Matriz 347832

Seguem as tabelas e figuras para a medida dos tempos para a matriz 347832.

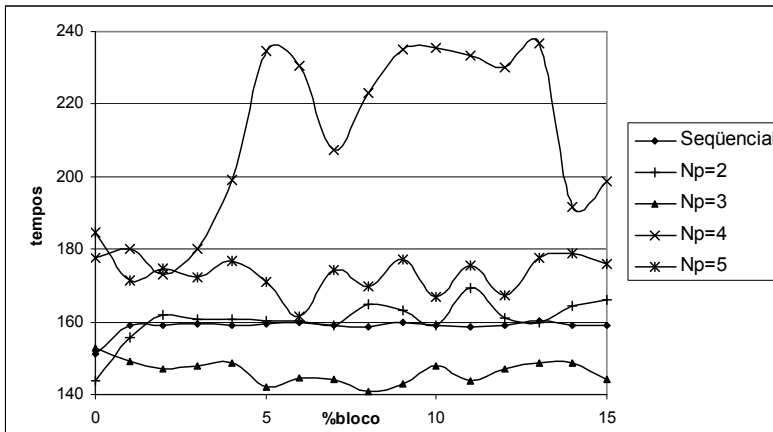
TABELA 6.36. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 347832

%	CG				BiCG				BiCGStab			
	Seqüencial = 158,8 s				Seqüencial = 149,8 s				Seqüencial = 278 s			
	Np				Np				Np			
	2	3	4	5	2	3	4	5	2	3	4	5
0	143,82	152,92	177,67	184,82	143,82	152,92	177,67	184,82	396,62	242,06	328,34	330,46
1	155,54	149,00	180,09	171,59	146,93	157,31	184,50	184,74	406,01	284,30	251,36	267,79
2	162,10	146,82	172,99	174,59	147,03	155,26	183,78	190,48	424,56	283,75	320,79	275,99
3	160,81	147,82	180,21	172,05	146,33	155,59	184,46	188,59	338,32	378,75	219,07	242,42
4	160,74	148,81	199,12	176,75	147,51	155,23	184,45	190,24	400,37	346,56	312,04	199,41
5	160,32	141,94	234,83	170,99	146,85	154,43	184,33	192,59	446,70	231,45	286,35	249,47
6	160,25	144,59	230,55	161,64	146,92	156,86	183,73	189,71	433,92	315,02	303,67	275,74
7	158,95	144,20	207,48	174,17	147,18	156,88	183,01	192,20	325,82	335,23	308,82	253,37
8	164,68	140,97	222,90	169,92	147,02	156,71	181,87	190,68	326,06	337,02	255,83	407,70
9	162,99	142,69	235,17	177,13	146,87	158,86	183,73	190,49	465,44	202,55	327,47	361,85
10	158,92	148,01	235,42	166,90	147,05	156,86	185,76	189,85	303,60	309,96	362,65	311,24
11	169,17	143,84	233,50	175,45	147,14	155,55	184,03	190,19	322,76	217,52	201,91	358,98
12	161,02	146,98	230,07	167,08	147,67	153,40	182,75	187,72	370,05	250,21	214,82	388,01
13	159,64	148,68	236,77	177,56	147,95	156,40	185,17	193,04	373,58	202,40	329,77	341,69
14	164,45	148,56	191,51	178,89	147,05	158,36	181,92	188,68	284,08	355,11	348,28	416,96
15	165,90	143,96	198,70	176,04	147,87	160,08	185,60	191,70	283,65	388,77	365,36	381,89

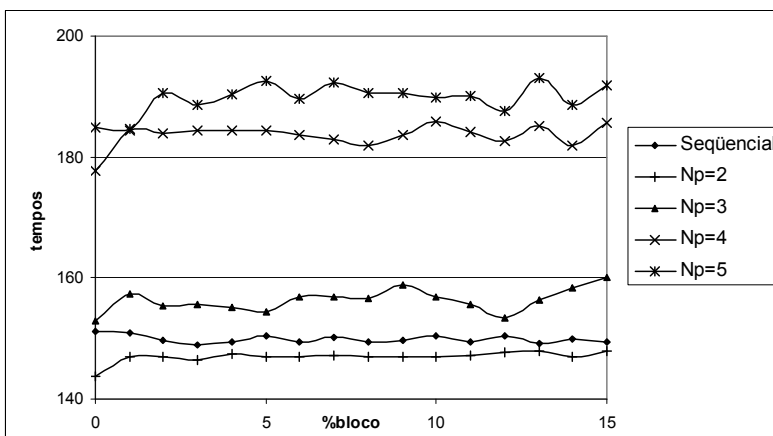
Como pode ser observado na tabela para o CG, bem como na figura 6.17a, o CG com três processadores foi mais rápido que o processamento seqüencial, além de ser mais rápido também para interseção acima de 0%.

O método BiCG com dois processadores também foi mais rápido que o processamento seqüencial, comparando-se ao CG.

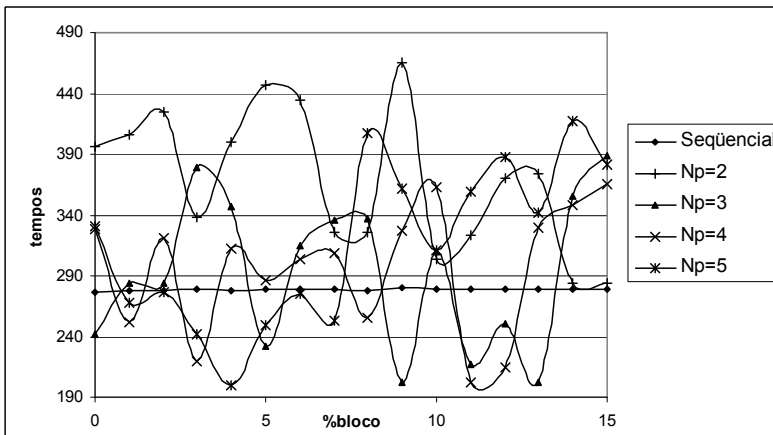
Para o BiCGStab foram encontradas várias condições onde o processamento é mais rápido que o serial com vários números de processadores. Além de vários pontos onde os tempos são menores que o tempo para 0%. Entretanto foi mais lento que o CG e o BiCG.



(a)



(b)



(c)

Fig. 6.18 – Variação do tempo de processamento para a Matriz 347832, pré -condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.

6.6.3 Resultados para a Matriz 1009899

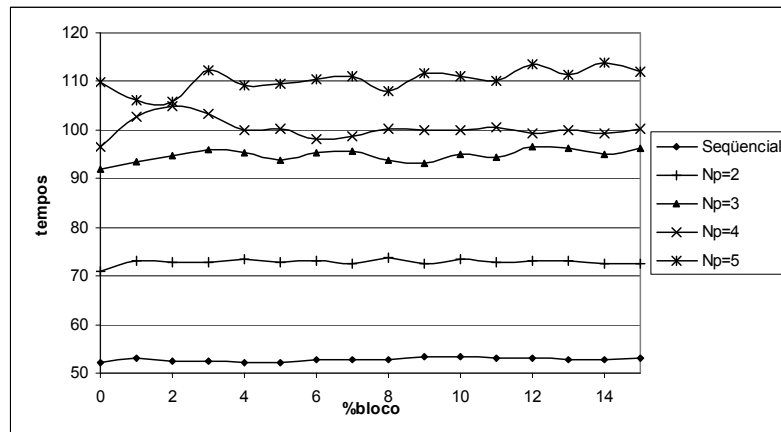
Seguem as tabelas e figuras para a medida dos tempos para a matriz 1009899. Esta matriz foi obtida através da modelagem 3D e, de novo, o algoritmo CGS não convergiu.

TABELA 6.37. TEMPOS DE PROCESSAMENTO PARA O SISTEMA 1009899

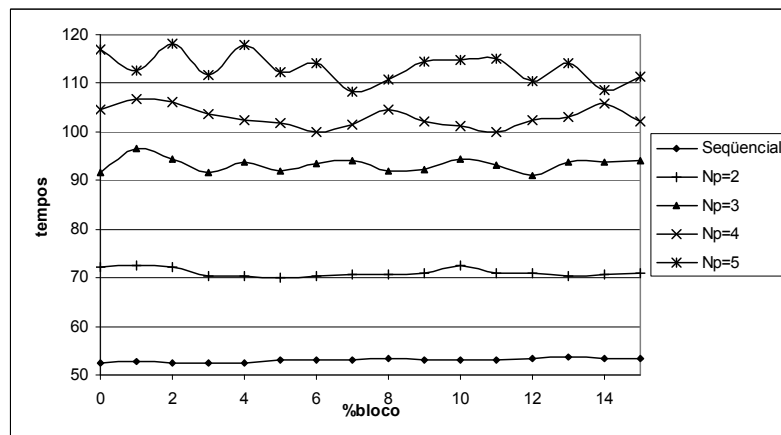
%	CG				BiCG				BiCGStab			
	Seqüencial = 52,8 s				Seqüencial = 53,0 s				Seqüencial = 47,5 s			
	Np				Np				Np			
	2	3	4	5	2	3	4	5	2	3	4	5
0	71,11	91,94	96,66	109,94	72,11	91,70	104,65	116,91	60,79	78,56	84,17	90,27
1	73,15	93,44	102,60	105,98	72,54	96,68	106,75	112,45	61,76	75,22	84,04	90,34
2	72,84	94,78	104,90	105,74	72,31	94,28	106,19	118,03	63,02	74,55	82,52	88,78
3	72,68	96,03	103,35	112,20	70,42	91,59	103,51	111,64	62,22	79,18	85,54	90,74
4	73,39	95,43	99,89	109,35	70,40	93,91	102,47	117,96	62,65	74,74	82,77	92,77
5	72,75	93,82	100,38	109,42	70,12	92,09	101,75	112,18	62,73	75,58	82,68	88,31
6	73,06	95,24	97,98	110,53	70,34	93,47	99,99	114,13	61,32	73,03	82,32	92,00
7	72,44	95,66	98,65	111,16	70,68	94,00	101,50	108,40	61,35	77,49	84,68	94,05
8	73,72	93,91	100,30	107,86	70,78	91,98	104,44	110,72	62,01	76,86	84,51	92,86
9	72,64	93,18	100,05	111,66	70,88	92,27	102,18	114,30	62,17	78,36	83,20	91,72
10	73,42	94,90	100,04	110,94	72,40	94,51	101,11	114,83	61,47	75,88	85,75	89,70
11	72,82	94,37	100,52	110,07	70,95	93,03	99,81	115,15	61,70	79,03	83,05	88,16
12	73,10	96,63	99,47	113,43	71,10	91,09	102,56	110,32	63,49	77,56	85,49	86,78
13	73,25	96,37	99,92	111,48	70,42	93,86	103,01	114,02	62,54	74,96	84,66	89,28
14	72,42	95,09	99,38	113,98	70,76	93,85	105,72	108,59	62,04	76,45	86,21	88,85
15	72,66	96,11	100,35	112,06	70,86	94,18	102,22	111,41	63,30	78,44	85,05	89,37

Na tabela e nos gráficos percebe-se que a paralelização não resultou ganhos de tempo. Entretanto, novamente se nota o desempenho superior do algoritmo BiCGStab, tanto na forma seqüencial como na forma paralela.

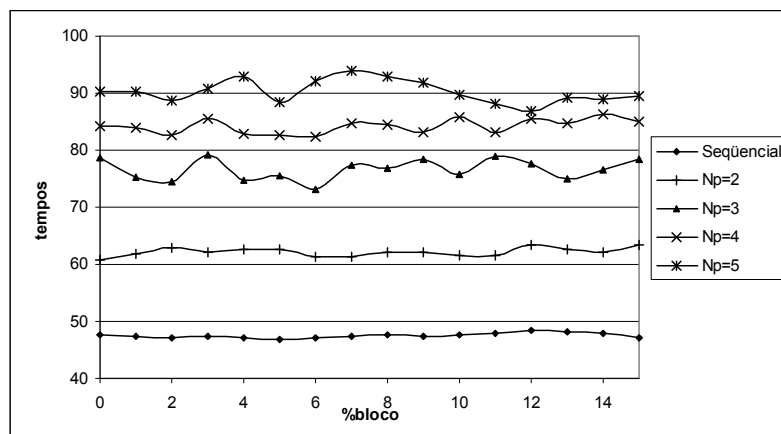
Pode-se observar também nos valores marcados em negrito que em algumas situações de interseção o tempo de processamento encontrado foi menor que aquele determinado para o seqüencial, com exceção para o método CG.



(a)



(b)



(c)

Fig. 6.19 – Variação do tempo de processamento para a Matriz 1009899, pré-condicionador FLUI, erro igual a 1×10^{-5} : (a) CG, (b) BiCG e (c) BiCGStab.

6.7 Resultados para outros pré-condicionadores

Além do pré-condicionador FLUI foram obtidos resultados para outros dois pré-condicionadores: o pré-condicionador *LU* incompleto não paralelizado e o pré-condicionamento de Jacobi.

O pré-condicionador *LU* incompleto não paralelizado calcula a decomposição *LU* antes da paralelização e então divide a matriz e o pré-condicionador entre os processadores realizando em paralelo as multiplicações matriz por vetor e as resoluções dos sistemas pré-condicionados.

O pré-condicionador de Jacobi é extremamente simples, tal que o mesmo é obtido diretamente das partes da matriz, quando estas já se encontram distribuídas nos processadores. Assim, neste caso a montagem e a aplicação do pré-condicionador acontecem em paralelo.

6.7.1 Resultado para o pré-condicionador *LU* incompleto não paralelizado

TABELA 6.38. TEMPOS PARA O SISTEMA 1009899 – PRÉ-CONDICIONAMENTO LU INCOMPLETO

Np	CG		BiCG		BiCGStab		CGS	
	s	iterações	s	iterações	s	iterações	s	iterações
1	51,65	87	50,52	49	45,56	38	47,31	42
2	61,01	87	61,25	49	54,48	38	56,46	42
3	61,94	87	61,99	49	54,86	38	57,68	42
4	64,27	87	64,55	49	57,29	38	60,21	42
5	65,42	87	66,01	49	58,69	38	61,84	42

TABELA 6.39. TEMPOS PARA O SISTEMA 347832 – PRÉ-CONDICIONAMENTO LU INCOMPLETO

Np	CG		BiCG		BiCGStab		CGS	
	s	iterações	s	iterações	s	iterações	s	iterações
1	155,94	394	145,22	188	273,77	401	146,84	190
2	150,05	392	139,92	188	223,47	333	145,02	195
3	144,92	391	137,32	188	232,56	363	138,75	190
4	149,55	392	140,33	188	263,76	408	143,48	195
5	148,42	391	138,21	188	258,61	404	142,95	195

6.7.2 Resultado para o pré-condicionador de Jacobi

TABELA 6.40. TEMPOS PARA O SISTEMA 1009899 – PRÉ-CONDICIONAMENTO DE JACOBI

Np	CG		BiCG		BiCGStab		CGS	
	s	iterações	s	iterações	s	iterações	s	iterações
1	57,93	183	53,46	98	55,27	107	52,63	99
2	76,68	183	72,95	98	76,80	107	72,22	99
3	77,87	183	74,76	98	78,27	106	73,86	99
4	82,88	183	79,01	98	83,82	109	79,35	99
5	86,31	183	82,32	98	87,32	107	82,96	99

TABELA 6.41. TEMPOS PARA O SISTEMA 347832 – PRÉ-CONDICIONAMENTO DE JACOBI

Np	CG		BiCG		BiCGStab		CGS	
	s	iterações	s	iterações	s	iterações	s	iterações
1	205,19	1129	195,83	577	1114,95	3727	190,60	560
2	189,92	1141	177,24	578	1078,44	4123	172,62	557
3	178,47	1129	166,83	577	1111,22	4555	180,33	631
4	194,71	1210	171,06	577	1119,73	4431	179,06	608
5	188,45	1225	163,37	579	836,24	3498	159,31	562

6.7.3 Comentários

Como primeira observação para estes últimos resultados está o maior número de iterações encontrado para o pré-condicionamento de Jacobi. É por este motivo que o mesmo é considerado um pré-condicionamento pobre. Também se conseguiu um ganho com a paralelização para o sistema 347832 para o pré-condicionamento LU incompleto.

Observando as tabelas 6.38 e 6.40, a melhor combinação encontrada para as matrizes de elementos finitos dos casos testados nesta tese une o método BiCGStab e o pré-condicionador LU incompleto com o menor tempo de processamento e também o menor número de iterações.

É possível também comparar, muito claramente, o desempenho do BiCG e do CG em ambas as situações de pré-condicionamento. O BiCG apresentou aproximadamente a metade do número de iterações do CG, mas também o mesmo tempo de processamento. Isto é consequência do produto matriz transposta por vetor exigido pelo BiCG. Mais uma vantagem para o BiCGStab.

Comparando estes desempenhos ao LU por blocos pode ser constatada uma elevada diferença entre os tempos apontados para cada método e o cálculo realizado com LU incompleto. Isto pode ser observado através dos resultados das tabelas 6.37 e 6.38 na linha de 0% para esta última. Isto se deve principalmente ao maior número de iterações necessárias ao LU por blocos.

6.8 Conclusão

Este capítulo apresentou um conjunto de resultados envolvendo processamento paralelo, seqüencial, métodos numéricos de resolução de sistemas de equações lineares e pré-condicionadores.

Aqui também foram apresentados os quatro métodos de armazenamento utilizados nos estudos da influência do método de compactação de matrizes esparsas no desempenho da multiplicação matriz por vetor em paralelo. Estes resultados são importantes, pois a operação de multiplicação matriz por vetor é a operação que mais consome tempo de processamento nos métodos iterativos de resolução de sistemas de equações lineares.

A resolução do sistema pré-condicionado também é uma operação de alto custo computacional, entretanto o tipo de pré-condicionador é uma escolha que o usuário pode fazer, enquanto que a multiplicação ocorre sempre, independente do método iterativo escolhido. Quando o método de armazenamento sofre sobrecarga devido à troca de mensagens, o método influencia a paralelização degradando o desempenho. Desta forma, um método de armazenamento pode ter um alto desempenho na forma seqüencial, mas quando implementado em paralelo pode apresentar um resultado que o torne menos indicado para uso na paralelização.

A primeira idéia que se faz do processamento paralelo é a redução do tempo de processamento, mas para que isso aconteça é necessário que exista um casamento entre a granulosidade do problema e a granulosidade do hardware empregado.

Em se tratando de clusters, o hardware tem uma granulosidade muito grossa comparada à granulosidade do problema numérico em questão. Desta forma é

necessário estudar formas de aliviar o acoplamento do problema numérico a fim de se obter ganho com o hardware.

Assim, este conjunto de resultados é parte de um dos objetivos da tese que trata da caracterização do hardware, comparando-o com outros, avaliando a capacidade de processamento do mesmo utilizando diversos algoritmos e formas de programação, procurando os limites onde o mesmo pode ser usado como plataforma de computação no tratamento matemático dos problemas envolvendo o método de elementos finitos.

Neste conjunto de resultados foi possível observar que o pré-condicionamento *LU* por blocos com interseção tem algumas vantagens sobre o pré-condicionador proposto em [73], sendo possível usá-lo em sistemas e algoritmos assimétricos, permitindo em alguns casos obter número de iterações e tempos menores que os encontrados para os casos sem interseção e também para o processamento seqüencial.

Além disso, foi possível mostrar o comportamento dos diversos algoritmos de resolução de sistemas de equações em relação aos números de iterações, tempo de processamento e dificuldades de convergência.

O próximo capítulo encerra este texto com uma análise mais completa deste estudo através de uma síntese dos resultados encontrados, respondendo, também, às questões relativas à tese levantadas na introdução deste trabalho.

7 Conclusão

7.1 Notas introdutórias

A questão principal que vem à mente de um leitor, quando este se depara com um texto sobre computação paralela ou de alto desempenho é quanto de ganho de processamento se obteve com a experimentação. Este trabalho teve início com a mesma preocupação e com as mesmas expectativas, para não dizer folclores, a respeito do inevitável ganho com a divisão de tarefas entre processadores.

No início do trabalho se acreditava que dada qualquer máquina que pudesse executar um programa paralelo, se um algoritmo como eliminação de Gauss fosse paralelizado nesta máquina, obter-se-iam ganhos e logo todos os problemas com longas horas de cálculo de campos estariam resolvidos. Mas a experimentação logo mostrou que o entendimento do problema e o conhecimento do hardware limitariam as expectativas.

O ganho de processamento com paralelização implica um problema numérico resolvido através de um hardware computacional com uma granulosidade idêntica ou mais fina que o problema numérico. Um cluster de computadores construído com máquinas e redes comerciais é um hardware de memória distribuída e de granulosidade grossa, tal que para resolver problemas como a solução de sistemas de equações lineares são necessárias algumas considerações, como o abandono da resolução através de métodos essencialmente seqüenciais como eliminação de Gauss, em favor de técnicas iterativas que utilizam multiplicações matriz por vetor e a utilização de outros expedientes como o desacoplamento do cálculo de pré-condicionadores [40], ao preço de se perder a eficácia desta última ferramenta.

Em vista destas necessidades o trabalho descrito neste texto procurou relacionar algumas das muitas variáveis que cercam este assunto e realizou um estudo das suas

interdependências. Assim foi possível observar o comportamento de diversos métodos iterativos de resolução de sistemas de equações, técnicas de compactação de matrizes esparsas, pré-condicionadores e hardwares dos chamados sistemas de alto desempenho aplicados às matrizes de elementos finitos.

As conclusões que se seguem procuram formar um panorama destas variáveis de processamento e, assim, esclarecer vários dos aspectos que as cercam. O item abaixo trata da síntese dos resultados mostrados no capítulo anterior e acima mencionados.

7.2 A síntese dos resultados

Ao longo deste texto foram mostradas características computacionais e resultados numéricos de quatro métodos de resolução iterativa de sistemas de equações lineares. O método iterativo fundamental Gradiente Conjugado, a versão para matrizes não simétricas Gradiente BiConjugado, e as versões para matrizes não simétricas, que não utilizam a transposta da matriz, Gradiente Conjugado Quadrado e Gradiente BiConjugado Estabilizado. Podemos observar o desempenho destes métodos sob dois pontos de vista: o primeiro no que diz respeito a sua capacidade de convergência em número de iterações e o segundo que diz respeito ao tempo de processamento, caso haja convergência.

Estes dois aspectos estão interligados, pois um menor número de iterações num método está relacionado a alguma técnica numérica auxiliar que consome processamento e, conseqüentemente, aumenta o tempo de cálculo. Desta forma, o algoritmo mais simples, mas talvez com maior número de iterações, pode apresentar o menor tempo. Isto acontece freqüentemente com o CG. Mas comparando-o com o BiCGStab, embora este último possua núcleos de processamento mais pesados computacionalmente, apresentou em diversas oportunidades tempos de processamento inferiores ao CG. Isto pode ser observado nas tabelas 6.37, 6.38 e 6.40.

Dentre os métodos que tratam matrizes não simétricas, as tabelas mostram a clara dificuldade de convergência do método Gradiente Conjugado Quadrado, tanto para matrizes pequenas como para matrizes maiores. Experimentalmente também se observou que os erros de arredondamento provocados pelas funções da biblioteca de comunicação

potencializaram os efeitos do operador de contração deste método, aumentando ainda mais a dificuldade de convergência deste método.

O Gradiente BiConjugado mostrou a eficiência já conhecida. Comparando os números de iterações entre o CG e o BiCG encontramos que o último tende a apresentar a metade do número de iterações encontrado pelo CG. Entretanto, por realizar pelo menos um produto matriz por vetor a mais, seu tempo de processamento é idêntico ou maior que para o CG. Isto pode ser visto nas tabelas 6.36 a 6.40.

Por último, o Gradiente BiConjugado Estabilizado se mostrou tão eficiente quanto o BiCG e, para as matrizes originárias do método de elementos finitos nos casos testados neste trabalho, menor número de iterações e menor tempo de processamento. Comparando os métodos CGS, BiCG e BiCGStab, pode-se afirmar a partir do desempenho registrado nas tabelas 6.35, 6.37 e 6.38 para matrizes originárias de elementos finitos, que o BiCGStab obteve o melhor desempenho.

Os resultados colhidos para os métodos de compactação mostraram que, em se tratando de clusters de computadores, a forma de armazenamento de matrizes esparsas também deve concordar com métodos que facilitem a distribuição dos dados numa filosofia que não entre em conflito com a arquitetura de memória distribuída. Isto se estende também às rotinas de comunicação utilizadas pelos métodos de armazenamento.

Os métodos de compactação ICRS e CRS mostraram pouca diferença de desempenho entre si. Para matrizes com grandes números de elementos o desempenho foi idêntico. Assim, com base nos números de iterações e tempos registrados nas tabelas 6.26 a 6.30, o armazenamento CCS e o armazenamento RCS se mostraram pouco adequados à arquitetura de computação paralela em clusters.

Este trabalho propôs uma versão do pré-condicionador LU por blocos com o acréscimo de um parâmetro de interseção para reduzir o enfraquecimento causado pelo desacoplamento do matriz de pré-condicionamento. Este pré-condicionador foi testado nos quatro métodos iterativos de resolução de sistemas de equação já descritos e seu desempenho comparado ao caso sem interseção e também aos pré-condicionadores LU incompleto não paralelizado e de Jacobi.

Observando as tabelas 6.31 a 6.37, notou-se em muitos casos a diminuição do número de iterações e também do tempo de processamento, especialmente no método BiCGStab, comparados aos casos sem interseção. Nos exemplos que tratam de matrizes de

elementos finitos a variação da porcentagem do bloco mostrou que os números de iterações variam até algo em torno de 15% da variação do número de linhas. Depois destes valores os números de iteração passam a variar muito pouco. Com isso e através dos valores colhidos, pode-se dizer que os clusters de computadores podem ser usados com sucesso na paralelização da resolução de sistemas de equações lineares com valores de interseção de até 10%.

Comparando com os pré-condicionadores LU incompleto e Jacobi encontramos novamente que a paralelização só é bem sucedida em relação ao tempo de processamento quando as matrizes são um pouco mais densas (Matriz 347832). Observando especificamente o número de iterações resultantes para o LU incompleto e o LU por blocos (sem interseção), percebe-se que este é rigorosamente o mesmo.

Por último foram utilizadas duas plataformas de alto desempenho para os testes nesta etapa final do trabalho: o cluster do GRUCAD e o cluster do L2EP. Embora ambos sejam clusters, máquinas com arquitetura de memória distribuída executando o sistema operacional Linux/GNU e biblioteca de comunicação MPI, estas máquinas são conectadas por hardware de rede com tecnologias muito diferentes.

A diferença pode ser observada através da figura 5.2 que ilustra a largura de banda e a latência destas máquinas. O cluster do L2EP tem uma latência inferior a duas vezes a latência do cluster do GRUCAD, o que permite a obtenção de ganhos em alguns exemplos como os casos mostrados na figuras 6.8 e 6.9.

Esta diferença de desempenho de hardware fica mais acentuada quando são comparados os resultados levando-se em conta a esparsidade das matrizes estudadas.

A figura 7.1 ilustra de forma simplificada a síntese dos resultados aqui apresentados.

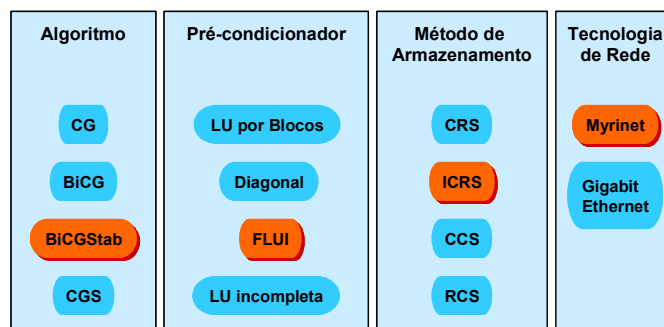


Fig. 7.1 – Síntese dos resultados obtidos: BiCGStab, FLUI, ICRS e Myrinet numa única plataforma de cálculo.

Entre os métodos iterativos, o BiCGStab apresentou o melhor desempenho, tanto na forma seqüencial como paralelizado, podendo substituir com vantagens o método BiCG. O pré-condicionamento LU incompleto por blocos com interseção mostrou-se adaptado às necessidades da computação paralela com memória distribuída. O método de armazenamento ICRS apresentou vantagens sobre os outros três métodos testados e, o método atualmente utilizado nos programas 3D (RCS) se mostrou pouco adaptado às necessidades da arquitetura de memória distribuída. A tecnologia de rede Gigabit Ethernet, embora seja mais facilmente encontrada no mercado, apresentou altos valores de latência e baixa largura de banda nos testes realizados, prejudicando o desempenho dos programas, que fazem uso de muito tráfego de mensagens.

7.3 Assertivas

Na introdução deste trabalho foram propostas questões e um conjunto de testes com os quais foram delineados objetivos de pesquisa, cuja intenção fundamental é auxiliar trabalhos futuros envolvendo computação de alto desempenho e cálculo de campos.

As respostas aqui escritas são baseadas nos experimentos realizados ao longo deste trabalho, através da comparação do conjunto de resultados obtidos.

- Como levantado nos diversos comentários sobre os resultados, o armazenamento influencia o desempenho na paralelização utilizando clusters de computadores quando a forma de armazenamento não concorda com a arquitetura de memória distribuída do hardware. Como pode ser observado nas tabelas de resultados do capítulo 6, os métodos CRS e ICRS distribuem os dados mantendo-se o desacoplamento exigido pela arquitetura, ao contrário dos métodos CCS e RCS cujas exigências de implementação incluem a utilização de funções de comunicação que exigem mais tempo de comunicação devido à arquitetura. Assim, um mesmo algoritmo iterativo (no caso testado o BiCGStab) utilizando formas de armazenamento diferentes, apresenta

desempenhos diferentes. Um outro detalhe a ser considerado é o número de iterações encontrado quando da utilização de outras formas de compactação. Ao menos na forma atual da implementação, a biblioteca de comunicação utiliza funções de comunicação que não realizam somas globais associativas [74]. Isto faz com que apareçam números de iterações diferentes devido aos erros de arredondamento num mesmo método iterativo funcionando com número diferente de processadores. Isto pode ser observado nas tabelas do capítulo anterior quando processando os dados para um processador comparando os métodos de compactação. Quando se tem apenas um processador o número de iterações é igual para todos os métodos, mostrando que isto é devido às funções de comunicação. Dentre os métodos de compactação utilizados nos testes e considerando matrizes com grande número de elementos, os métodos ICRS e CRS tornaram-se equivalentes, tal que podem ser utilizados em algoritmos paralelizados nos clusters de computadores com melhor desempenho que com os métodos CCS e o RCS.

- A esparsidade de um sistema matricial é um dado fundamental em se tratando de paralelização com clusters de computadores, tal que segundo os resultados já mostrados a torna mais importante que a ordem da matriz. Isto é prontamente observado através dos resultados da matriz com 2000 variáveis, mas com 50% de enchimento (ou esparsidade). Neste caso, apesar do pequeno número de variáveis a paralelização obteve ganho. Este não é o caso das matrizes de elementos finitos testadas cuja esparsidade se encontra na ordem de 1×10^{-5} . Estas matrizes apresentam algo em torno de 20 elementos por linhas, tal que mesmo com matrizes com ordens muito altas, não foram encontrados ganhos de paralelização no cluster do GRUCAD. Mas em sistemas com latências menores e com esparsidade em torno de 1×10^{-4} já é possível se obter ganhos na paralelização.
- Comparando-se as tecnologias de rede utilizadas para conectar os processadores dos clusters, observou-se que a tecnologia Myrinet apresenta uma latência menor que a metade do valor encontrado no *switch* com tecnologia Ethernet Gigabit. Este valor de latência se torna muito importante quando a esparsidade dos sistemas matriciais é muito pequena, caso dos

sistemas de elementos finitos. Observando estes valores de latência e os resultados comparativos entre as duas tecnologias para os mesmos problemas, como mostrado nas tabelas 6.6 a 6.30, nota-se que a tecnologia Myrinet é mais apropriada para a computação em cluster, ao menos no estágio atual.

- Neste trabalho é encontrada a proposta de uma nova variação do pré-condicionador LU , uma versão modificada por blocos com um parâmetro que permite a interseção entre os mesmos. A divisão da matriz de pré-condicionamento entre os processadores é uma forma de diminuir o tráfego de mensagens entre as máquinas do cluster. Entretanto, quando o tráfego é diminuído, adaptando o cálculo do pré-condicionamento à arquitetura de memória distribuída, enfraquece-se o pré-condicionador. Isto é sentido através do aumento do número de iterações em relação à execução seqüencial. A versão por blocos proposta apresentou uma leve vantagem sobre o método sem acoplamento anteriormente proposto [40] no que tange o número de iterações e o tempo de processamento. A vantagem é a utilização desta variação em métodos mais robustos como o BiCGStab, cujo tempo de processamento é menor que o encontrado para o CG. A interseção necessária para obter esta vantagem é pequena, algo em torno de 10%, o que não acarreta grandes custos computacionais a mais. Também, através da figura 6.15, é possível observar que o erro de convergência também afeta o comportamento desta variação de pré-condicionamento.
- Respondendo, então, “à pergunta que não quer calar”: foi encontrado ganho de desempenho com a paralelização dos algoritmos de resolução de sistemas de equações lineares nos cluster de computadores? A resposta é afirmativa, desde que a esparsidade do sistema não seja tão pequena e o switch de rede não possua uma latência muito alta. Mas, em função do ganho de desempenho encontrado, o custo de montagem, da programação e da utilização do cluster ainda é alto demais para ser largamente utilizado como plataforma de cálculo neste tipo de problema numérico. Assim, a perspectiva de utilização de clusters de computadores ainda fica limitada ao tratamento de problemas onde o desacoplamento acontece em um nível superior da análise como, por exemplo, o cálculo paralelizado a partir de uma divisão do dispositivo (domínio de

estudo). Numa estratégia de pesquisa que continue a utilizar clusters de computadores, também se deve levar em conta a tecnologia de rede e seus valores de latência em função do tipo de problema que se pretende resolver. Problemas com troca de mensagens mais pronunciadas devem optar por tecnologias com menores tempos de comunicação.

- Assim, encerrando o conjunto de questões envolvendo a paralelização de algoritmos aplicados à resolução de sistemas de equações em sistemas de elementos finitos, a abordagem de paralelização utilizada neste trabalho é apenas uma parte do esforço que deve ser empregado na paralelização destes sistemas de cálculo. São necessários estudos e desenvolvimentos com outras técnicas de paralelização, como por exemplo, a utilização da decomposição do domínio na formação do problema numérico, realizando etapas como a geração da malha e condensação dos sistemas de equação, com suas variáveis de interface de forma concorrente. Se pensarmos que a médio prazo estarão disponíveis redes muito mais rápidas permitindo processamento distribuído e paralelo, os esforços da pesquisa utilizando os recursos da paralelização são promissores, mas envolvem investimentos também no conhecimento das tecnologias de rede, também se dispendo a pagar por elas.

7.4 Conclusões finais

O trabalho apresentado testou diversas técnicas e algoritmos utilizados na análise numérica de sistemas de equações lineares e apresentou um subconjunto destas ferramentas que se mostraram mais adequadas à computação paralela utilizando clusters de computadores. Como já bastante frisado, arquitetura de memória distribuída destas máquinas tem associada uma elevada latência na comunicação, tal que os problemas a serem tratados nas mesmas também devem ter granulosidades mais grossas. Ou, devem ser utilizadas técnicas de relaxamento do acoplamento destes problemas.

Como conclusões importantes, a forma de armazenamento da matriz esparsa não

deve penalizar o processo de distribuição de dados entre os nós da máquina, seja através da distribuição dos elementos matriciais, seja pela utilização de rotinas que sofram mais fortemente com a latência do sistema. O pré-condicionador LU por blocos com interseção proposto pode ser usado com sucesso na resolução dos sistemas de equações com valores de interseção de até 10%, não afetando de forma significativa o armazenamento de dados.

Um programa completo utilizando as técnicas aqui estudadas deveria contemplar um conjunto de fatores que vai da escolha do algoritmo de resolução com menor custo iterativo, a forma de armazenamento esparsa que colabore com a arquitetura de memória distribuída, um pré-condicionador adaptado à arquitetura do cluster e uma rede de baixa latência. Além disso, como mencionado, sugere-se o estudo de outras técnicas de paralelização a fim de se explorar com mais intensidade a paralelização do cálculo de dispositivos eletromagnéticos.

ANEXO 1

Armazenamento Compactado por Linha do EFCAD (ACL)

Introdução

Dada uma matriz *simétrica* A , na forma de banda, o formato ACL permite armazenar a matriz em um único vetor. Isto é obtido mantendo-se apenas a medida da meia-banda, que neste caso tem valor igual para os lados esquerdo e direito, pois a matriz é simétrica.

Nós dizemos que a matriz $A = a_{ij}$ é banda se existirem constantes não negativas p e q chamadas de meia-banda direita e esquerda, tal que seus valores se mantenham constantes e caracterizando a existência de uma banda, tal que a matriz não seja necessariamente simétrica. Se a matriz é simétrica p e q são iguais e a meia-banda passará a ser chamada mb .

O armazenamento na forma ACL envolve o armazenamento de alguns zeros já que as diagonais mais exteriores podem ser mais esparsas. Assim, uma meia-banda esparsa pode causar um enchimento e pode tornar este método ineficiente.

Considere-se a matriz simétrica A definida por:

$$A = \begin{pmatrix} 10 & 3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 7 & 0 & 0 & 0 \\ 0 & 7 & 8 & 8 & 0 & 0 \\ 0 & 0 & 8 & 7 & 9 & 0 \\ 0 & 0 & 0 & 9 & 9 & 2 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{pmatrix}$$

Usando o formato ACL, armazenaremos esta matriz A num vetor de dimensão $(nno \times mbb)$ onde nno é a ordem da matriz A , e mbb é a meia-banda mais o elemento da diagonal.

Assim, a matriz A compactada na forma de um vetor $ssF(:)$ fica:

$$ss\mathbf{F} = (10 \ 3 \ 9 \ 7 \ 8 \ 8 \ 7 \ 9 \ 9 \ 2 \ -1 \ 0)$$

Note-se que o último zero do vetor corresponde a um elemento que *não existe* na matriz A .

Armazenamento na forma de vetor

Um outro exemplo de matriz simétrica é mostrado a seguir, onde se pode observar a presença de vários zeros na banda. A matriz será compactada no formato ACL e onde aparecerão elementos no vetor $ss\mathbf{F}$ que não existem na matriz original.

Esta matriz será utilizada no desenvolvimento dos algoritmos de compactação e tratamento ACL.

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 & 4 \\ 0 & 0 & 0 & 4 & 5 & 5 \\ 0 & 0 & 0 & 4 & 5 & 6 \end{pmatrix}$$

Dados fundamentais para o processo são a ordem da matriz n e a meia largura de banda mb . Obtidos estes valores determina-se o número de elementos válidos na linha da matriz, ou a largura válida de meia-banda definida por mbb , considerando que as meias larguras de banda sejam iguais e a matriz, simétrica.

$$mbb = mb + 1$$

A largura válida de meia-banda mbb considera a meia-banda mais o elemento diagonal. Além disso, apenas a meia-banda superior da matriz é armazenada. Se a

banda for esparsa serão armazenados zeros dentro das linhas que não estão totalmente preenchidas.

O vetor ssF representando a matriz A no formato ACL é:

$$ssF = (1 \ 1 \ 1 \ 2 \ 0 \ 0 \ 3 \ 0 \ 0 \ 4 \ 4 \ 4 \ 5 \ 5 \ X \ 6 \ X \ X)$$

O comprimento do vetor é igual a $(nno \times mbb)$ onde nno é a ordem da matriz A , e mbb é a largura válida de meia-banda. Deve-se observar a presença dos elementos “X” no vetor, indicando que estes elementos são alocados na memória, mas não representam elementos válidos no cálculo e **não** devem ser considerados.

Os elementos diagonais do vetor são acessados da seguinte forma:

```
for i=1:nno
    pos=(i-1)*mbb+1
```

Para a matriz exemplo os valores de nno , mb e mbb são:

```
mb = 2; mbb = mb + 1 = 3; nno = 6;
```

```
i = 1          pos=((1-1)*3+1=1
               ssF(pos)=ssF(1)=1
```

```
i = 2          pos=((2-1)*3+1=4
               ssF(pos)=ssF(4)=2
```

```
i = 6          pos=((6-1)*3+1=16
               ssF(pos)=ssF(16)=6
```

Fazendo uma ligeira modificação na expressão da variável pos , podemos acessar qualquer elemento do vetor considerando a largura válida de meia-banda mbb . Aqui i indicará a posição da linha e j indicará a *posição dentro da banda*.

```
for i=1:nno
    for j=1:mbb
        pos=(i-1)*mbb+j
```

Para a mesma matriz exemplo, com os valores de $nno = 6$, $mb = 2$ e $mbb = 3$:

```

for i=1:6
    for j=1:3
        pos=(i-1)*3+j

i = 1; j = 1; pos=((1-1)*3+1=1
        ssF(pos)=ssF(1)=1

i = 2; j = 3; pos=((2-1)*3+3=6
        ssF(pos)=ssF(6)=0

i = 4; j = 2; pos=((4-1)*3+2=11
        ssF(pos)=ssF(11)=4

i = 5; j = 3; pos=((5-1)*3+3=15
        ssF(pos)=ssF(15)='X'

i = 6; j = 2; pos=((6-1)*3+2=17
        ssF(pos)=ssF(17)='X'

```

A figura a seguir mostra a localização destes elementos no vetor ssF .

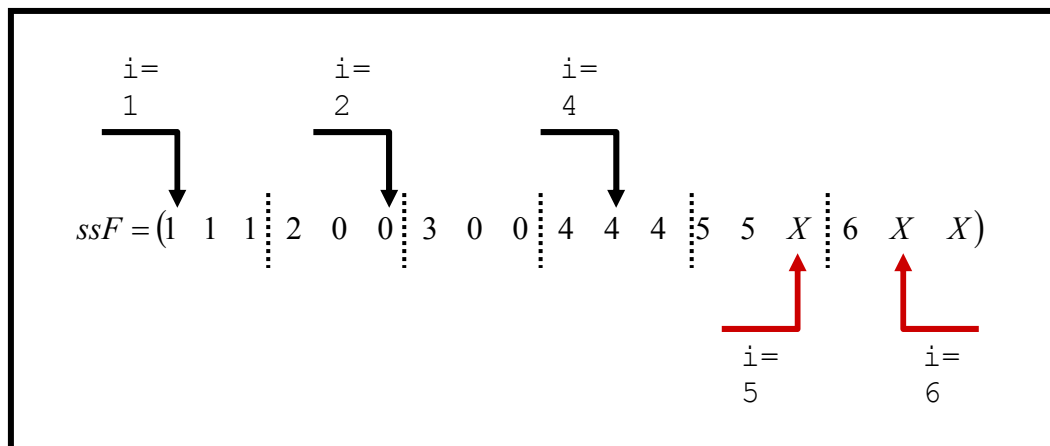


Fig. A1.1 – Localização de elementos no vetor compactado.

Os dois últimos exemplos acessaram elementos válidos de memória, mas valores *inválidos* no sistema. Por esse motivo o acesso deve ser prevenido considerando e corrigindo a máxima largura válida de meia-banda m_{bb} .

Então, vejamos:

Para $i=5$, obtemos a linha número 5. O índice também indica o elemento diagonal.

Se $(i + mbb > nno)$, então a máxima largura válida de meia-banda para a linha 5 é a diferença entre o limite direito (nno) e a diagonal (i) mais a própria diagonal.

```
i = 5; mb = 2; mbb = 3; nno = 6;
5 + 2 = 7 > 6
mbmax = nno - i + 1 = 6 - 5 + 1 = 2
```

O valor obtido para $mbmax$ é correto, como pode ser observado na figura logo acima.

Deste modo:

```
for i=1:nno
    if ((i+mb)>nno) mbmax=nno-i+1
    else mbmax = mbb
    for j=1:mbmax
        pos=(i-1)*mbb+j
```

Exemplos para o procedimento:

```
i = 5; mb = 2; mbb = 3; nno = 6;
5 + 2 = 7 > 6
mbmax = nno - i + 1 = 6 - 5 + 1 = 2
for j=1:2
    pos=(5-1)*3+j → pos=4*3+1=13  ssF(13)=5
                  → pos=4*3+2=14  ssF(14)=5
```

```
i = 6; mb = 2; mbb = 3; nno = 6;
6 + 2 = 8 > 6
mbmax = nno - i + 1 = 6 - 6 + 1 = 1
for j=1:1
    pos=(6-1)*3+j → pos=5*3+1=16  ssF(16)=6
```

Mapeamento matriz-ACL

O mapeamento da matriz A para armazenamento na forma de um vetor ACL é feito considerando apenas a parcela triangular superior da matriz, examinando as linhas e colunas e usando a transformação a seguir.

$$J = j - i + 1 \quad \begin{cases} i = 1 : nno \\ j = i : mbmax \end{cases}$$

$$pos = (i - 1) * mbb + J$$

O valor de J vem da definição da distância entre o elemento e a diagonal, mais a própria diagonal, onde i e j percorrem todos os elementos da matriz. Os limites de procura para o índice j têm sua razão nos seguintes fatores:

- $j = i$: a busca numa matriz simétrica começa a partir da diagonal;
- $j = mbmax$: j não deve ultrapassar o limite da banda, nem o limite de colunas nno .

```
for i=1:nno
    if((i+mb)>nno) mbmax=nno;
    else mbmax=i+mb;
    end;
    for j=i:mbmax
        J=j-i+1;
        pos=(i-1)*mbb+J
        ssF(pos)=A(i,j);
    end;
end;
```

Exemplos:

```
A(4,5)=4  i = 4; j = 5; mb = 2; mbb = 3; nno = 6;
           J = j - i + 1 = 5 - 4 + 1 = 2
           pos = (4 - 1) * 3 + 2 = 11
           ssF(pos) = ssF(11) = A(4,5) = 4
```

```
A(1,2)=1  i = 1; j = 2; mb = 2; mbb = 3; nno = 6;
           J = j - i + 1 = 2 - 1 + 1 = 2
           pos = (1 - 1) * 3 + 2 = 2
           ssF(pos) = ssF(2) = A(1,2) = 1
```

```
A(5,6)=5  i = 5; j = 6; mb = 2; mbb = 3; nno = 6;
           J = j - i + 1 = 6 - 5 + 1 = 2
           pos = (5 - 1) * 3 + 2 = 14
           ssF(pos) = ssF(14) = A(5,6) = 5
```

```
A(6,6)=6  i = 6; j = 6; mb = 2; mbb = 3; nno = 6;
           J = j - i + 1 = 6 - 6 + 1 = 1
           pos = (6 - 1) * 3 + 1 = 16
           ssF(pos) = ssF(16) = A(6,6) = 6
```


Conversão vetor compactado ACL para Matriz

A transformação inversa consiste na procura do índice de coluna (j) de um determinado elemento, visto que são conhecidas a sua linha (i) e sua posição dentro da banda (J). A transformação inversa é realizada utilizando a mesma relação da formação do vetor, tomando-se o cuidado de verificar se não foi gerado um valor inválido no índice j da matriz A , pois o valor calculado pode acessar índices fora do limite da banda nas regiões no final da matriz, como descrito anteriormente na formação do vetor ACL.

$$\begin{array}{l}
 J = j - i + 1 \\
 \\
 j = J + i - 1
 \end{array}
 \left\{ \begin{array}{l}
 i = 1 : nno \\
 j = 1 : nno \\
 J = 1 : mbmax
 \end{array} \right.
 \left\{ \begin{array}{l}
 nno - i + 1 \\
 mbb
 \end{array} \right.$$

O formato ACL contém apenas a banda superior da matriz A . Desta forma na reconstrução do vetor deve-se também endereçar o elemento simétrico.

```

for i=1:nno
  if((i+mb)>nno) mbmax=nno-i+1;
  else mbmax=mbb;
  end;
  for J=1:mbmax
    j=J+i-1;
    pos=(i-1)*mbb+J;
    A(i,j)=ssF(pos);
    A(j,i)=A(i,j);
  end;
end;

```

Exemplos:

```

i = 6; mb = 2; nno = 6;
6 + 2 = 8 > 6 mbmax = 6 - 6 + 1 = 1
J = 1:1
J = 1                j = 1 + 6 - 1 = 6
pos = (6 - 1) * 3 + 1 = 16

```

$A(6,6) = \text{ssF}(16)$

i = 5; mb = 2; nno = 6;
 $5 + 2 = 7 > 6$ $\text{mbmax} = 6 - 5 + 1 = 2$
 $J = 1:2$
 $J = 1$ $j = 1 + 5 - 1 = 5$
 $\text{pos} = (5 - 1) * 3 + 1 = 13$
 $A(5,5) = \text{ssF}(13)$

$J = 2$ $j = 2 + 5 - 1 = 6$
 $\text{pos} = (5 - 1) * 3 + 2 = 14$
 $A(5,6) = \text{ssF}(14)$
 $A(6,5) = A(5,6)$

i = 4; mb = 2; nno = 6;
 $4 + 2 = 6$ $\text{mbmax} = \text{mbb} = 2 + 1 = 3$
 $J = 1:3$
 $J = 1$ $j = 1 + 4 - 1 = 4$
 $\text{pos} = (4 - 1) * 3 + 1 = 10$
 $A(4,4) = \text{ssF}(10)$

$J = 2$ $j = 2 + 4 - 1 = 5$
 $\text{pos} = (4 - 1) * 3 + 2 = 11$
 $A(4,5) = \text{ssF}(11)$
 $A(5,4) = A(4,5)$

$J = 3$ $j = 3 + 4 - 1 = 6$
 $\text{pos} = (4 - 1) * 3 + 3 = 12$
 $A(4,6) = \text{ssF}(12)$
 $A(6,4) = A(4,6)$

ANEXO 2

Decomposição LU incompleta por blocos com interseção com compactação RCS

Introdução

Este anexo descreve uma forma de interseção entre partes de uma matriz, mantendo, porém, o desacoplamento entre elas.

A divisão entre os *hosts*

No cálculo paralelizado a matriz A deve ser dividida em submatrizes, onde cada computador do cluster (*host*) recebe uma parte da mesma. No caso do cálculo do pré-condicionamento com interseção, envia-se aos *hosts* com $ID \geq 1$, um maior número de linhas, mais precisamente, linhas pertencentes à submatriz imediatamente anterior, definindo-se assim um intervalo de interseção. A figura a seguir ilustra a divisão proposta.

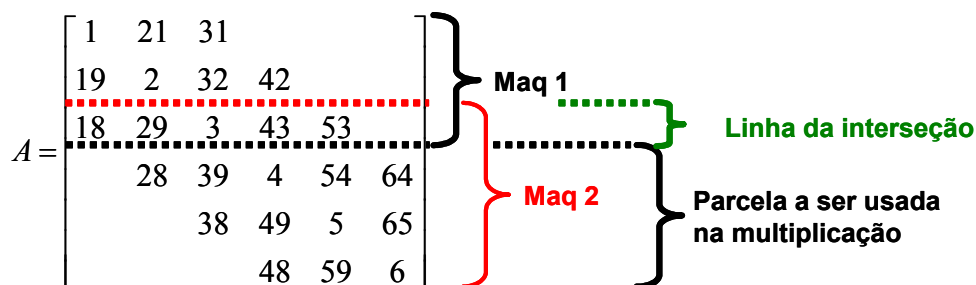


Fig. A2.1 – Divisão e linha de interseção.

Divisão do número de linhas da matriz para quebra do vetor `nced`.
 Criação de tabelas com informações do conteúdo em cada máquina.

`nhost`: número de máquinas para dividir a matriz
`bloco`: número inteiro de linhas que um host vai possuir.
`extra`: "sobra" de elementos da matriz na divisão (que será destinado ao host número 1).

`maq`: vetor com o número do host. Exemplo: `maq=[1 2 3]`
`nlinhas`: vetor com o número de linhas de matriz que cada host vai ter.
`offset`: distância da origem em número de linhas que um bloco tem.
 exemplo: se `nno=6` com `nhost=2`

```
maq =    1    2
nlinhas = 3    3
offset =  0    3
```

`bini`: linha em que cada host tem seu inicio
`bend`: linha em que cada host tem seu fim
 exemplo: se `nno=6` com `nhost=2`

```
bini =    1    4 --> host=1 começa em 1, host=2 começa em 4
bend =    3    6 --> host=1 termina em 3, host=2 termina em 6
```

A divisão de `nno/nhost` pode gerar um número fracionário, então com a função `floor` obtém-se o inteiro arredondado mais próximo em direção ao menos infinito.

```
nhost = 2;
bloco = floor(nno/nhost);
extra = nno-bloco*nhost;
```

Vetores com as informações de cada máquina.

```
maq=zeros(1,nhost);
nlinhas=zeros(1,nhost);
offset=zeros(1,nhost);
bini=zeros(1,nhost);
bend=zeros(1,nhost);
```

Nesta versão, o primeiro host fica com um número de linhas que é igual à quantidade inteira de linhas denominada *bloco*, cujo valor é destinado a todos os *hosts* igualmente, mais a parcela *extra* encontrada.

```
for dest = 1:nhost
    maq(dest) = dest;
Para o primeiro vai a quantidade extra, se ela existir.
    if(extra~=0 && dest==1)
        nlinhas(dest) = bloco + extra;
    else
        nlinhas(dest) = bloco;
    end;
    bini(dest) = offset(dest) + 1;
    bend(dest) = bini(dest) + nlinhas(dest) - 1;
    disp(sprintf('bloco %d, tamanho %d, offset %d, inicio %d, fim %d',
        maq(dest), nlinhas(dest), offset(dest), bini(dest),
        bend(dest)));

    if ((dest+1) <= nhost) offset(dest+1) = offset(dest) + nlinhas(dest);
end;
end;
```

Fig. A2.4 – Algoritmo para divisão do número de linhas.

Considerando a matriz exemplo e o número de hosts igual a 2 este processamento resulta a seguinte divisão, mostrada na figura a seguir:

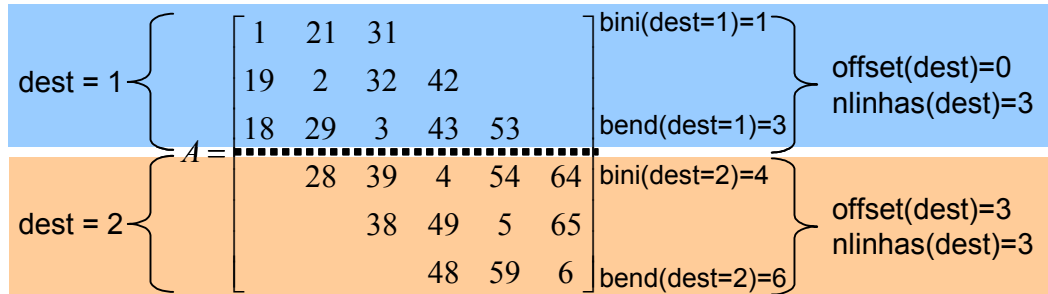


Fig A2.5 – Divisão da matriz em dois *hosts*.

Para o novo processamento, além da informação do número de *hosts*, deve aparecer o valor de interseção, que indica quanto de interseção se deseja com a região anterior. O valor de interseção é um número de 1 a 100 indicando a porcentagem de interseção que se deseja obter. Naturalmente, dependendo do número de linhas reais existentes a interseção total pode ser tornar inviável ou, se o número de linhas for muito pequeno o valor de interseção deverá ser grande, para que se obtenha um resultado útil. A passagem de um valor em porcentagem para um número simples de linhas é feita com uma conversão simples, como mostrado na expressão abaixo:

$$itr\text{linhas}[dest] = \text{floor}\left(\frac{n\text{linhas}[dest-1] * \text{inter}}{100}\right)$$

Nesta expressão o número atual (*dest*) de linhas de interseção é tomado como função do número de linhas destinado ao processador do destino anterior (*dest-1*). No caso do primeiro host o número de linhas de interseção é zero, naturalmente.

Considerando que as matrizes serão primeiramente decompostas e depois multiplicadas, são, necessários dois conjuntos de dados como tamanhos, *offsets* e linhas iniciais e finais para o processamento. Estes conjuntos são:

```
maq          | vetor, as máquinas destino
nlinhas     | vetor, as quantidades de linhas destinadas à multiplicação
offset      | vetor, os inícios de cada bloco para multiplicação
bini        | vetor, os inícios de cada bloco
```

```

bend      | vetor, os fins de cada bloco

inter     | inteiro, a porcentagem destinada à interseção de blocos
itrlinhas| vetor, as quantidades de linhas de interseção dos blocos
FTlinhas  | vetor, as quantidades totais de linhas usadas na fatoração
FToffset  | vetor, os offsets utilizados na decomposição
FTbini    | vetor, os inícios de cada bloco quando da decomposição

```

A idéia básica de divisão pode ser entendida a partir da ilustração a seguir, onde é desejada uma interseção de 34%, ou igual a 1 linha, conforme a expressão.

```

inter = 34
itrlinhas[1] = 0
itrlinhas[2] = floor(nlinhas[1]*34/100) = floor(3*34/100) = floor(1,02) = 1
itrlinhas[3] = floor(nlinhas[2]*34/100) = floor(3*34/100) = floor(1,02) = 1

```

A figura mostra uma matriz, onde se desejam partições de 3 linhas em cada processador com interseções de 1 linha (34%). São ilustrados os vetores *nlinhas*, *FTlinhas* e *itrlinhas*.

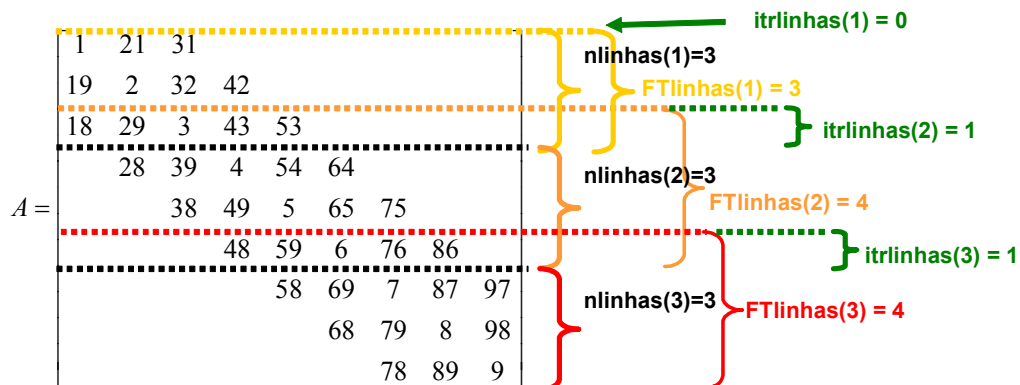


Fig. A2.6 – Divisão em três partes e vetores associados.

Depois que ocorrem as procuras pelos limites das matrizes para as partes destinadas à decomposição e à multiplicação, quebra-se a matriz nas suas devidas parcelas. A divisão para a decomposição deve ser feita com as dimensões *FTbini* – *bend*, com *FTlinhas* e *FToffset*.

Continuando o exemplo, a matriz sofrerá um corte, desejando-se separá-la com 70% de interseção (gerando duas linhas a mais no vetor do segundo processador) considerando seus vetores *baspk* e *haupk*, conforme a figura a seguir. Observe que existe interseção nas porções abraçadas por cada vetor. A cor mais clara mostra a

porção da matriz destinada ao primeiro processador enquanto que a cor mais escura mostra a porção destinada ao segundo.

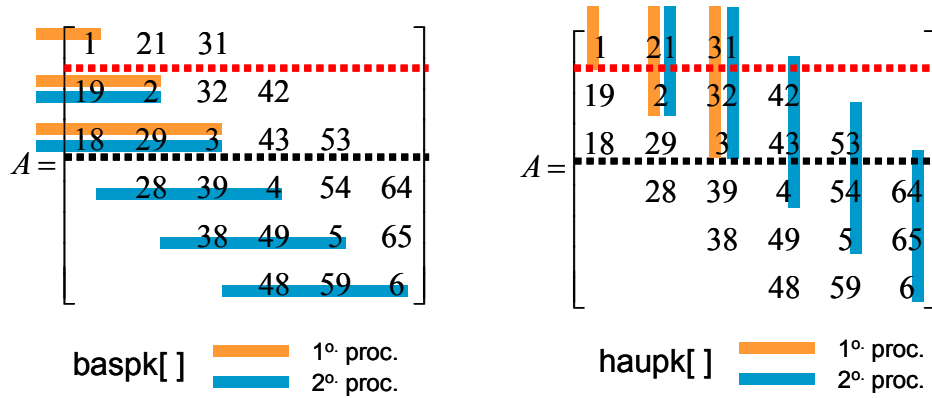


Fig. A2.7 – Porções destinadas a cada processador e vetores associados.

O resultado da segmentação é ilustrado na figura a seguir, onde são mostrados os vetores resultantes.

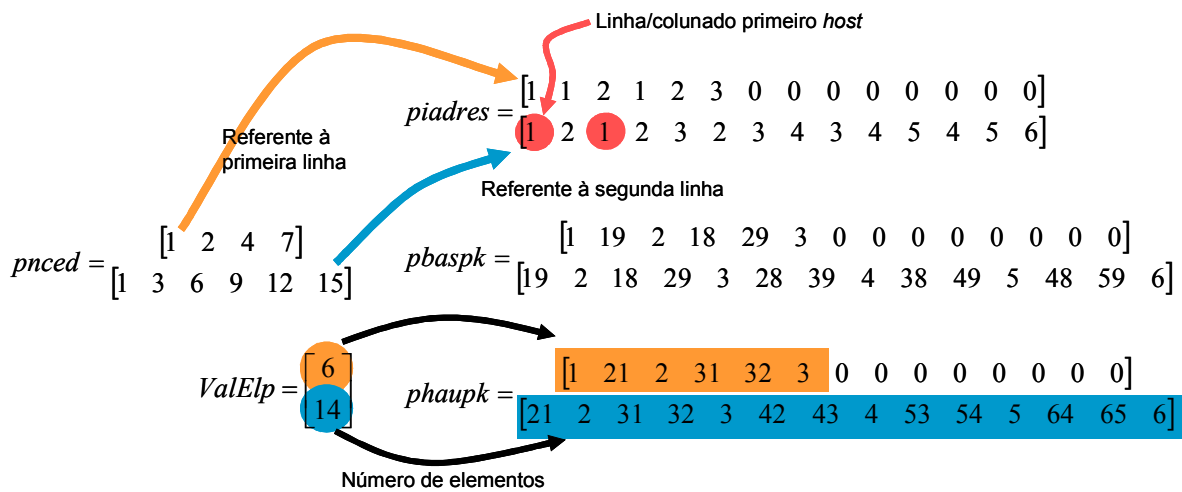


Fig. A2.8 – Vetores de armazenamento depois do corte.

Pode-se observar nos pequenos círculos no vetor *piadres*, na parte destinada ao segundo processador, dois elementos pertencentes ao processador 1. Tais elementos devem ser retirados por um deslocamento para a esquerda do vetor. Depois do deslocamento os vetores são reenumerados, percebendo-se o encurtamento dos vetores

```

Chamada à decomposição
A: vetor para receber soupk
B: vetor para receber suppk
C: vetor para receber itr
for dest = 1:nhost
    clear A;
    clear B;
    clear C;
    [A,B,C,rapeff(dest),ier(dest)] =
        nced_lu_fact(FTlinhas(dest),pnced(dest,:),piadres(dest,:),
        pbaspk(dest,:),phaupk(dest,:),ValElp(dest))
    for ii=1:ValElp(dest)
        psoupk(dest,ii) = A(ii);
        psuppk(dest,ii) = B(ii);
    end;
    for ii=1:FTlinhas(dest)
        itr(dest,ii) = C(ii);
    end;
end;

```

Fig. A2.11 – Chamada à decomposição.

Eliminação das linhas em excesso para as matrizes fatoradas

Depois que as decomposições foram realizadas considerando o número de linhas a partir da linha de interseção, estas matrizes L e U devem ter as linhas em excesso eliminadas, como sugerido pela figura A2.3.

Utilizando o exemplo da figura A2.10, se a decomposição calculou uma matriz com 5 linhas (da linha 2 até a 6), mas são desejadas apenas as linhas que vão da 4 até a 6, isto é, somente as 3 últimas linhas, então as linhas 2 e 3 devem ser excluídas.

Mas nos vetores, estes endereçamentos estão organizados na forma padrão RCS, indo de 1 até o número de linhas destinado àquela parte, valor este armazenado no vetor ***FTlinhas***. Assim, cada matriz fatorada, a partir do segundo processador, deve ter seus dados cortados, excluindo-se a parte que não deve ser armazenada.

O defasamento novo será dado por:

Esta filtragem tem uma idéia simples: se o índice encontrado em *IADRES* for maior que *dpl* então o índice de *IADRES* é corrigido.

```

Algoritmo para filtragem dos elementos dentro da região válida
  if(piadres(dest,jj) > dpl)
    ...
    piadres(dest,elm) = piadres(dest,jj) - dpl;
    ...
    elm = elm + 1 ;

```

Fig. A2.13 – Algoritmo para filtragem dos elementos dentro da região válida.

O vetor é encurtado através das duas contagens: uma com *jj*, que mostra o incremento natural com o qual são percorridos os vetores entre *kmin* e *kmax*; e outra com *elm*, que controla o incremento dos elementos que permanecem, pois se encontram dentro da região maior que *dpl*.

Uma operação mais crítica é a determinação de quando o *NCED* muda. Isto é o controlado através das variáveis *col*, *ii* e *cont*.

ii é a variável que indica em qual linha/coluna o *NCED* está localizado no momento.

col é a variável usada para controlar quando o *NCED* deve ser atualizado. De uma forma simbólica indica a coluna de inserção do elemento. Se a coluna for a mesma que a anterior, o IF não é verdadeiro e a atualização não acontece.

cont é a variável utilizada para controlar o avanço do indexador de *NCED*.

```

Algoritmo para atualização do vetor NCED
for ii=1:FTlinhas(dest)
  ...
  for jj = kmin:kmax
    if(piadres(dest,jj) > dpl)
      if (col ~= ii) % se a coluna/linha for diferente da atual
                    % significa que houve uma mudança
        col = ii; % então atualiza para a coluna/linha atual
        pnced(dest,cont) = elm; % atualiza o vetor de apontadores
                               % com a linha inicial
        cont = cont+1; % incrementa o número de elementos de NCED
      end;

```

Fig. A2.14 – Algoritmo para atualização do vetor *NCED*.

Depois que os elementos são eliminados com correção dos índices de colunas/linhas é necessário eliminar os elementos que antes preenchiam os vetores *NCED*, *IADRES* e de valores.

O algoritmo a seguir desenvolve todas as operações aqui descritas.

```

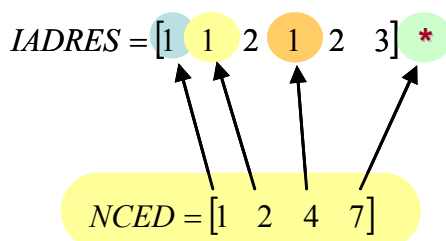
Algoritmo para eliminação das linhas em excesso.
for dest = 1:nhost
    dpl = FTlinhas(dest) - nlinhas(dest);
    elm = 1;
    cont = 1;
    col = 0;

    for ii=1:FTlinhas(dest)
        kmin = pnced(dest,ii);
        kmax = pnced(dest,ii+1)-1;

        for jj = kmin:kmax
            if(piadres(dest,jj) > dpl)
                if (col ~= ii)
                    col = ii;
                    pnced(dest,cont) = elm
                    cont = cont+1;
                end;
                piadres(dest,elm) = piadres(dest,jj) - dpl;
                psoupk(dest,elm) = psoupk(dest,jj);
                psuppk(dest,elm) = psuppk(dest,jj);
                elm = elm + 1;
            end;
        end;
    end;
    pnced(dest,cont) = elm; % vetor de apontadores corrigido na última
                          % posição com o valor do último elemento +1
    for ii=cont+1:FTlinhas(dest)+1 %zera os elementos restantes do pnced
        pnced(dest,ii) = 0;
    end;
    for ii=elm:ValElp(dest) %zera os elementos restantes dos outros vetores
        piadres(dest,ii) = 0;
        psoupk(dest,ii) = 0;
        psuppk(dest,ii) = 0;
    end;
    ValElp(dest) = elm -1; % número de elementos corrigidos de iadres/p_haupk
end;

```

Fig. A2.15 – Algoritmo para eliminação das linhas em excesso da decomposição.

Fig. A2.16 – Vetor $NCED$ resultante.

Eliminação das linhas em excesso para a multiplicação matriz-por-vetor

Diferente da eliminação de linhas destinada à decomposição, que precisa eliminar elementos para formar uma matriz quadrada, na qual é aplicado o algoritmo de decomposição, a eliminação de linhas para a matriz A não deve eliminar os outros elementos existentes e que pertenceriam a um outro processador. Isto porque, se eliminarmos estes elementos, não possuiríamos mais a matriz de rigidez (matriz SS) e a solução encontrada não seria a solução do sistema em questão, e sim a solução de um outro problema.

Desta forma, a eliminação das linhas de SS deve simplesmente eliminar o bloco "quadrado" que forma o conjunto de dados no processador anterior. Com isso ficarão os "rabos das setas". Assim, a integridade da matriz será mantida. Assim o único vetor a ser reescrito será o vetor $NCED$, que deve marcar o início e fim de cada seqüência de linhas/colunas.

De novo o defasamento (dpl) será:

$$dpl = FTlinhas - nlinhas$$

As primeiras " dpl " entradas do vetor $NCED$ serão eliminadas bem como os respectivos elementos indicados por estas nos outros vetores, sem a correção do $iadres/baspk/haupk$ (quando o índice for $< dpl$) como ocorreu no caso da decomposição.

Os endereços de *IADRES* são os endereços originais, não sendo reescritos. Este procedimento precisa da matriz inicialmente encontrada logo após o corte e antes de qualquer deslocamento.

Por isso o algoritmo demonstrativo usa uma cópia nas variáveis *Anced*, *Apiadres*, *Apbaspk*, *Aphaupk*, *AValElp*.

```

Algoritmo para eliminação das linhas em excesso
for dest = 1:nhost
    dpl = FTlinhas(dest) - nlinhas(dest);
    elm = 1; %controla o numero de elementos que existe no vetor piadres
    cont = 1; %usado para controlar o avanço do indexador de pnced
    col = 0; %usado para controlar quando pnced deve ser atualizado. De uma
            %forma simbólica indica a coluna de inserção do elemento. Se a
            %coluna for a mesma que a anterior, o IF não é verdadeiro e a
            %atualização não acontece.
    for ii=dpl+1:FTlinhas(dest),
        kmin = Apnced(dest,ii); %o inicio da linha original
        kmax = Apnced(dest,ii+1)-1; %o fim da linha

        if (col ~= ii)
            col = ii;
            Apnced(dest,cont) = elm; %atualiza o vetor de apontadores
            cont = cont+1;
        end;
        for jj = kmin:kmax
            Apiadres(dest,elm) = Apiadres(dest,jj);
            Apbaspk(dest,elm) = Apbaspk(dest,jj);
            Aphaupk(dest,elm) = Aphaupk(dest,jj);
            elm = elm + 1;
        end;
    end;

    Apnced(dest,cont) = elm; % vetor de apontadores corrigido na ultima
                            % posição com o valor do último elemento+1
    for ii=cont+1:FTlinhas(dest)+1 %zera os elementos restantes do pnced
        Apnced(dest,ii) = 0;
    end;
    for ii=elm:AValElp(dest) % zera os elementos restantes
        Apiadres(dest,ii) = 0;
        Apbaspk(dest,ii) = 0;
        Aphaupk(dest,ii) = 0;
    end;
    AValElp(dest) = elm -1; %numero de elementos corrigidos de iadress/p_haupk
end;

```

Fig. A2.17 – Algoritmo para eliminação das linhas em excesso para multiplicação.

Observe que a grande diferença entre o algoritmo da figura A2.15 e o da figura A2.17 é a ausência da declaração condicional **if(piadres(dest,jj) > dpl)** que restringe os elementos que são copiados.

Como exemplo, observe a figura a seguir onde a matriz é cortada mantendo-se o “rabo da seta”.

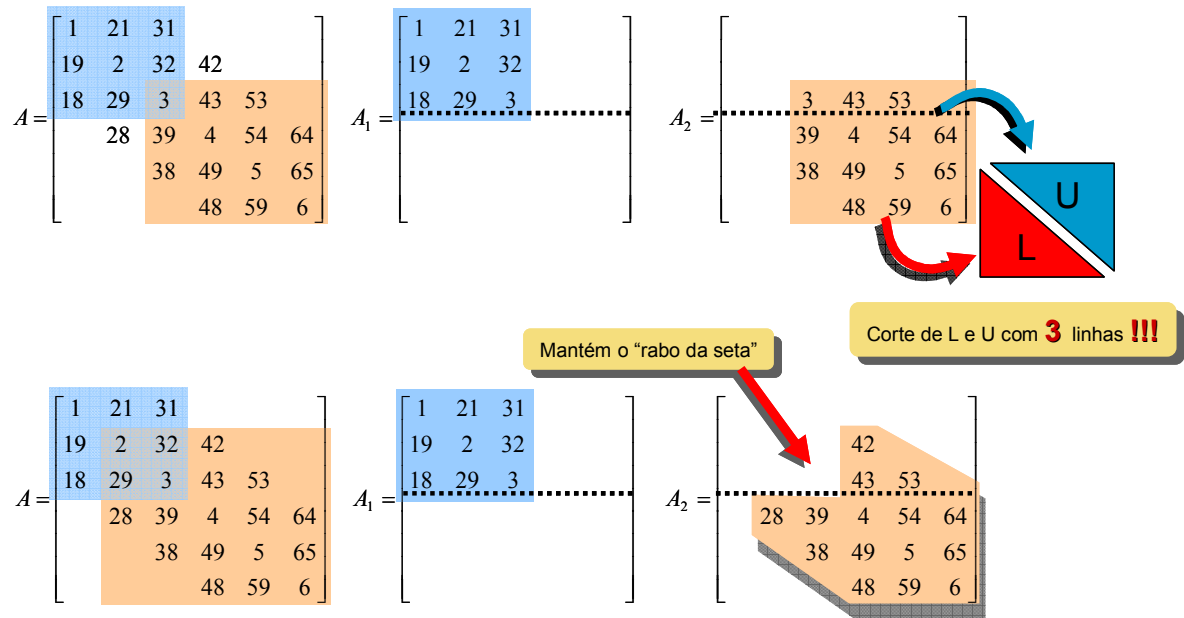


Fig. A2.18 – Eliminação de linhas: para as matrizes fatorizadas (em cima) e para a multiplicação (em baixo).

Correção dos índices para multiplicação em paralelo

Para complementar o anexo, o texto a seguir elabora algoritmos para a reunião dos diversos vetores *NCED*/*IADRES* encontrados em diversos processadores de volta na forma de uma única seqüência. Com esta preparação os vetores passam ao estado original antes do corte. É necessário corrigir a posição indicada pelo *NCED* e a coluna/linha indicada pelo *IADRES*. Estes algoritmos são executados no “mestre” depois de receberem os diversos vetores vindos dos “escravos”.

NCED: Este vetor é uma composição seqüencial que aponta para o início de cada coluna/linha de *IADRES*. Este vetor é quebrado e posteriormente rearranjado de forma a descrever matrizes parciais completas que são fatorizadas no método LU.

Para remontar esta seqüência, partiu-se de uma verificação de todos os vetores *NCED* parciais e através deles reescreveu-se a seqüência. As etapas do processamento são:

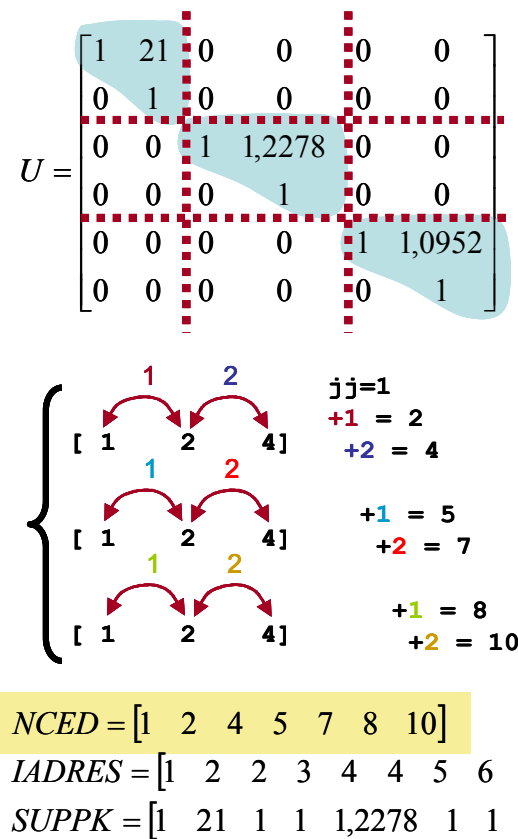
- Num procedimento no MESTRE, reúnem-se os vetores parciais e monta-se em um único vetor a seqüência completa. O vetor chamado “*somaced*” é inicializado com 1 (como padrão no método RCS).
- Depois, calcula-se a distância entre os inícios de cada nova linha/coluna, calculando-se a variável *dist* e somando-se ao *somaced* anterior, de acordo com a figura a seguir.

A figura a seguir mostra 3 matrizes parciais tipo *U* que devem ser reunidas em uma única matriz tipo *U* global para elaborar o exemplo de reunião do *NCED*. A figura também mostra os vetores *NCED* e *IADRES* para cada uma delas.

$$\begin{array}{l}
 U_1 = \begin{bmatrix} 1 & 21 \\ & 1 \end{bmatrix} \quad \begin{array}{l} NCED = [1 \quad 2 \quad 4] \\ IADRES = [1 \quad 1 \quad 2] \\ SUPPK = [1 \quad 21 \quad 1] \end{array} \\
 \\
 U_2 = \begin{bmatrix} 1 & 1,2278 \\ & 1 \end{bmatrix} \quad \begin{array}{l} NCED = [1 \quad 2 \quad 4] \\ IADRES = [1 \quad 1 \quad 2] \\ SUPPK = [1 \quad 1,2278 \quad 1] \end{array} \\
 \\
 U_3 = \begin{bmatrix} 1 & 1,0952 \\ & 1 \end{bmatrix} \quad \begin{array}{l} NCED = [1 \quad 2 \quad 4] \\ IADRES = [1 \quad 1 \quad 2] \\ SUPPK = [1 \quad 1,0952 \quad 1] \end{array}
 \end{array}$$

Fig. A2.19 – Matrizes parciais e vetores associados.

Calculando-se a distância entre os elementos de cada um dos *NCED* parciais e fazendo a somatória, encontramos o *NCED* global da matriz total, tal como mostrado na figura que se segue.

Fig. A2.20 – Vetor *NCED* Global.

```

Algoritmo para correção do NCED.
jj = 1;
somanced(jj)=1;
for dest=1:nhost
    disp('-----');
    disp(sprintf('Processador %d',dest));
    for ii=1:nlinhas(dest)
        dist = pnced(dest,ii+1)-pnced(dest,ii);
        somanced(jj+1)=somanced(jj)+dist;
        jj = jj+1;
    end;
end;

```

Fig. A2.21 – Algoritmo para correção do *NCED*.

IADRES: A correção dos índices de *IADRES* é mais simples e consiste simplesmente na soma dos *offsets* referentes a cada escravo, como mostra a parcela de código a seguir.

```

Algoritmo para correção do IADRES.
elm = 1;
for dest=1:nhost
    disp('-----');
    disp(sprintf('Processador %d',dest));
    for ii=1:nlinhas(dest)
        kmin = pnced(dest,ii);      % o inicio da linha
        kmax = pnced(dest,ii+1)-1; % o fim da linha

        % dentro dos limites kmin ate kmax aplica o offset
        for jj = kmin:kmax
            piadres(dest,jj) = piadres(dest,jj) + offset(dest);
            elm = elm + 1;
        end;
    end;
end;

```

Fig. A2.22 – Algoritmo para correção do *IADRES*.

Resultados para a decomposição sem interseção

A figura a seguir traz os resultados da fatoração sobre toda a matriz problema, com as matrizes L e U mostradas em separado, e também os resultados da fatoração sem interseção através da quebra simples da matriz em duas submatrizes quadradas, fatoradas de forma independente por dois processadores. A figura mostra estas duas matrizes reorganizadas na forma de uma matriz ampliada, onde é possível observar a ausência de diversos termos em relação à matriz fatorada em um processador. Em ambos os casos a decomposição é feita somente com os elementos não nulos (incompleta).

Pode-se observar que o primeiro processador calcula matrizes iguais àquelas que seriam encontradas com um cálculo realizado de forma única. A diferença ocorre a partir do segundo *host*, quando o algoritmo não mais possui informações das outras linhas. Um comportamento semelhante foi relatado na referência [40].

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 28 & -0,2846 & 6,9357 & 0 & 0 \\ 0 & 0 & 38 & 52,5346 & -371,4518 & 0 \\ 0 & 0 & 0 & 48 & -313,1186 & -83,0774 \end{bmatrix}$$

(a)

$$U = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & -0,1058 & 0 & 0 \\ 0 & 0 & 1 & -0,093 & -0,8111 & 0 \\ 0 & 0 & 0 & 1 & 7,7525 & 9,2276 \\ 0 & 0 & 0 & 0 & 1 & 1,1301 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)

$$L_{\text{des}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 49 & -656,5 & 0 \\ 0 & 0 & 0 & 48 & -589 & -116,9261 \end{bmatrix}$$

(c)

$$U_{\text{des}} = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 13,5 & 16 \\ 0 & 0 & 0 & 0 & 1 & 1,0952 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(d)

Fig. A2.23a-d – (a) Decomposição L; (b) Decomposição U; (c) Decomposição L desacoplada com 2 hosts; (d) Decomposição U desacoplada com 2 hosts.

Resultados para a decomposição com interseção

Os resultados a seguir têm por objetivo mostrar as diferenças nos resultados conseguidos com e sem interseção. O último resultado mostra que o algoritmo sem interseção é um subconjunto da técnica aqui apresentada quando a interseção desejada é igual a 0%.

Os resultados são apresentados na seguinte forma: a decomposição L incompleta integral da matriz, com interseção de 100%, com 70%, com 34% e com 0%; depois a decomposição de U nas mesmas condições. Observe-se que importa no exemplo os resultados obtidos para a segunda submatriz, visto que a primeira tem sempre o mesmo resultado, já que não sofre interseção.

A figura a seguir mostra o resultado da decomposição incompleta feita de forma integral para a parte L da matriz, para uma interseção de 100% (as 3 linhas da parte destinada ao processador 1), com 70% (2 linhas), com 34% (1 linha) e com 0% (nenhuma linha de interseção).

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 28 & -0,2846 & 6,9357 & 0 & 0 \\ 0 & 0 & 38 & 52,5346 & -371,4518 & 0 \\ 0 & 0 & 0 & 48 & -313,1186 & -83,0774 \end{bmatrix} \quad L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \quad (b)$$

$$L_{100\%} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6,9357 & 0 & 0 \\ 0 & 0 & 0 & 52,5346 & -371,4518 & 0 \\ 0 & 0 & 0 & 48 & -313,1186 & -83,0774 \end{bmatrix} \quad (c)$$

$$L_{70\%} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 0 & 0 & -81,8438 & 0 & 0 \\ 0 & 0 & 0 & 2,3449 & 9,5687 & 0 \\ 0 & 0 & 0 & 48 & 63,0927 & -397,1429 \end{bmatrix} \quad (d)$$

$$L_{34\%} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 0 & 0 & -555 & 0 & 0 \\ 0 & 0 & 0 & -495,6667 & -99,2192 & 0 \\ 0 & 0 & 0 & 48 & 4,0811 & 11,8577 \end{bmatrix} \quad (e)$$

$$L_{0\%} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 19 & -397 & 0 & 0 & 0 & 0 \\ 18 & -349 & -65,3451 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 49 & -656,5 & 0 \\ 0 & 0 & 0 & 48 & -589 & -116,9261 \end{bmatrix}$$

(f)

Fig. A2.24a-f – Matrizes fatorizadas tipo L: (a) integral, (b) submatrizes, (c) 100% de interseção, (d) 70% de interseção, (e) 34% de interseção, (f) sem interseção.

Observe-se que no primeiro caso (figura A2.24c, $L_{100\%}$) a decomposição levou a uma submatriz L_{22} idêntica à submatriz da decomposição integral incompleta, como era de se esperar. Quando a interseção foi de 0% a fatoração gerou uma matriz L_{22} igual ao resultado apresentado sem interseção (fig. A2.14c). Nos casos intermediários as matrizes resultantes apresentaram valores diferentes.

A figura A2.25 a seguir realiza a mesma comparação, agora com as matrizes triangulares superiores resultantes da decomposição. Novamente para a matriz $U_{100\%}$ temos uma semelhança da submatriz U_{22} com a fatoração num único processador e, no caso sem interseção, o resultado levou a uma matriz idêntica ao resultado obtido no item anterior. Isto demonstra que a decomposição sem interseção é um resultado particular para o algoritmo de decomposição por blocos com interseção de 0%.

$$U = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & -0,1058 & 0 & 0 \\ 0 & 0 & 1 & -0,093 & -0,8111 & 0 \\ 0 & 0 & 0 & 1 & 7,7525 & 9,2276 \\ 0 & 0 & 0 & 0 & 1 & 1,1301 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a)

$$U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

(b)

$$U_{100\%} = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 7,7525 & 9,2276 \\ 0 & 0 & 0 & 0 & 1 & 1,1301 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(c)

$$U_{70\%} = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -0,0853 & -0,782 \\ 0 & 0 & 0 & 0 & 1 & 6,9846 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(d)

$$U_{30\%} = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1,1441 & -0,1153 \\ 0 & 0 & 0 & 0 & 1 & -0,079 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(e)

$$U_{0\%} = \begin{bmatrix} 1 & 21 & 31 & 0 & 0 & 0 \\ 0 & 1 & 1,403 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 13,5 & 16 \\ 0 & 0 & 0 & 0 & 1 & 1,0952 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(f)

Fig. A2.25a-f – Matrizes fatorizadas tipo U: (a) integral, (b) submatrizes, (c) 100% de interseção, (d) 70% de interseção, (e) 34% de interseção, (f) sem interseção.

Referências Bibliográficas

- [1] SADOWSKI, N.; LEFEVRE, Y.; LAJOIE-MAZENC, M.; CROS, J. Finite Element Torque Calculation in Electrical Machines while Considering the Movement. *IEEE Transactions on Magnetics*, v. 28, n. 2, p. 1410-1413, Mar.
- [2] KUO-PENG, P.; SADOWSKI, N.; BASTOS, J. P. A.; CARLSON, R.; BATISTELA, N. J. A General Method for Coupling Static Converters with Electromagnetic Structures. *IEEE Transactions on Magnetics*, v. 33, n. 2, p. 2004-2009, Mar. 1997.
- [3] OLIVEIRA, A. M.; ANTUNES, R.; KUO-PENG, P.; SADOWSKI, N.; DULAR, P. Electrical Machine Analysis Considering Field – Circuit – Movement and Skewing Effects. In: 10th BIENNIAL IEEE CONFERENCE ON ELECTROMAGNETIC FIELD COMPUTATION (Jun. 2002: Perugia, Itália). *Proceedings*. Perugia, 2002. p. 274.
- [4] CSENDES, Z. J.; SILVESTER, P. Numerical Solutions of Dielectric Loaded Waveguides: I – Finite Element Analysis. *IEEE Transactions On Microwave Theory and Techniques*, v. MTT-8, n. 12, p. 1124-1131, Dec. 1970.
- [5] BASTOS, J.P.A., SADOSWSKI, N. *Electromagnetic Modeling by Finite Elements*. New York: Marcel Dekker, 2003.
- [6] KWON, H.-D. Efficient parallel implementations of finite element methods based on the conjugate gradient method. *Appl. Math. Comput*, v. 145, p. 869-880, 2003.
- [7] H. A. van der Vorst, “Iterative Methods for Large Linear Systems,” <http://www.math.uu.nl/people/vorst/lecture.html> (acesso Ago. 07, 2004).
- [8] SCHMIDT, E. Título desconhecido. *Rendiconti del Circolo Matematico di Palermo*. 25, p. 53-57, 1908.
- [9] FOX, L; HUSKEY, H. D.; WILKINSON, J. H. Notes on the Solution of Algebraic Linear Simultaneous Equations. *Quarterly Journal of Mechanics and Applied Mathematics*, v. 1, p. 149-173, 1948.
- [10] HESTENES, M. R.; STIEFEL E. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, v. 49, p.

- 409-436, 1952.
- [11] REID, J. K. On the Method of Conjugate Gradients for Solution of Large Sparse Systems of Linear Equations. In: *Large Sparse Sets of Linear Equation*. John K. Reid, ed. London and New York: Academic Press, 1971, p. 231-254.
- [12] MEIJERINK, J. A.; VORST, H. A. An iterative solution method for linear systems of with the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, v. 31, p. 148-162, 1977.
- [13] FLETCHER, R. Conjugate gradient methods for indefinite systems. In: G. A. Watson, editor. *Proceedings of the Dundee Biennial Conference on Numerical Analysis 1974*. New York: Springer Verlag, 1975. p. 73-89.
- [14] SAAD, Y.; SCHULTZ, M. H. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, v. 7, p. 856-869, 1986.
- [15] SONNEVELD, P. CGS, a fast Lanczos-type solver for non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, v. 10(1), p. 36-52, 1989.
- [16] VORST, H. A. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, v. 12, p. 631-644, 1992.
- [17] GREENBAUM, A. *Iterative Methods for Solving Linear Systems, Frontiers in Applied Mathematics*. Vol. 17. Philadelphia: SIAM, 1997.
- [18] PIQUET, J.; VASSEUR, X. Multigrid Preconditioned Krylov Subspace Methods for Three-dimensional Numerical Solutions of the Incompressible Navier-Stokes Equations. *Numerical Algorithms*, v. 17, 1-2, p. 1-32, 1998.
- [19] WEXLER, A. Computation of Electromagnetic Fields. *IEEE Transactions on Microwave Theory and Techniques*, v. 17, p. 416-439, Aug. 1969.
- [20] TINNEY, W. F.; WALKER, J. W. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, v. 55, p. 1801-1809, Nov. 1967.
- [21] OGBUOBIRI, E. C.; TINNEY, W. F.; WALKER, J. W. Sparsity directed decomposition for Gaussian elimination on matrices. In: *POWER INDUSTRY COMPUTER APPL. CONF. REC. (1967)*. p. 215-225.

- [22] KISNER, W.; DELLA TORRE, E. An Iterative Approach to the Finite-Element Method in Field Problems. *IEEE Transactions on Microwave Theory and Techniques*, v. 22, n. 3, p. 221–228, Mar. 1974.
- [23] CHARI, M. V. K. Nonlinear finite element solution of electrical machines under no-load and full-load conditions. *IEEE Transactions on Magnetics*, v. 10, n. 3, p. 686–689, Sep. 1974.
- [24] BORDING, R. P. Parallel Processing And Finite Element Methods. In: SOUTHEASTCON '81 (5-8 April 1981). *Conference Proceedings*. p. 189–193.
- [25] WANG, J.-S.; IDA, N. Parallel algorithms for direct solution of large systems of equations. In: 2ND SYMPOSIUM ON THE FRONTIERS OF MASSIVELY PARALLEL COMPUTATION (10-12 Oct. 1988). *Proceedings*. p. 231 – 234.
- [26] IDA, N.; WANG, J.-S. Parallel implementation of field solution algorithms. *IEEE Transactions on Magnetics*, v. 24, n. 1, p. 291–294, Jan. 1988.
- [27] SUBRAMANIAN, P.; IDA, N. A parallel algorithm for finite element computation. In: 2ND SYMPOSIUM ON THE FRONTIERS OF MASSIVELY PARALLEL COMPUTATION (10-12 Oct. 1988). *Proceedings*. p. 219–222.
- [28] WAIT, R. Parallel finite elements. In: INTERNATIONAL SPECIALIST SEMINAR ON THE DESIGN AND APPLICATION OF PARALLEL DIGITAL PROCESSORS (11-15 Apr. 1988). *Proceedings*. p. 11–14.
- [29] MAGNIN, H.; COULOMB, J. L.; PERRIN-BIT, R. Parallel and vectorial solving of finite element problems on a shared-memory multiprocessor. *IEEE Transactions on Magnetics*, v. 28, n. 2, p. 1712–1715, Mar. 1992.
- [30] ALBUQUERQUE, Marcelo P.; ALBUQUERQUE, Márcio. P.; ALVES, N.; RIBEIRO, D. P. Ambiente de Computação de Alto Desempenho do CBPF: Projeto SSOLAR. In: WGC 2004 (Fev. 2004: Petrópolis).
- [31] MATSUO, T.; SHIMASAKI, M. Parallel Computation for High Speed Duct Flow of Weakly Ionized Plasma. In: THE FOURTH INTERNATIONAL CONFERENCE/ EXHIBITION ON HIGH PERFORMANCE COMPUTING IN THE ASIA-PACIFIC REGION, (May 2000). *Proceedings*. v. 2, 14-17, p. 1075–1080.
- [32] IWASHITA, T; et al. Parallel Processing of ILU Preconditioned BiCGSTAB Solver Using Algebraic Multi-Color Ordering Method. In: INTERNATIONAL

- CONFERENCE ON PARALLEL COMPUTING IN ELECTRICAL ENGINEERING PARELEC '02 (22-25 Sept. 2002). Proceedings. p. 293–298.
- [33] “The OpenMP Application Program Interface,” <http://www.openmp.org/> (acesso Maio 10, 2006).
- [34] KNIGHT, A. M. Efficient parallel solution of time-stepped multislice eddy-current induction motor models. *IEEE Transactions on Magnetics*, v. 40, n. 2, p. 1282–1285, Mar. 2004.
- [35] M. Snir et al, “MPI: The Complete Reference,” <http://www.netlib.org/utk/papers/mpi-book/mpi-book.html> (acesso Mar. 11, 2004).
- [36] COSTA, M. C.; PALIN, M. F.; CARDOSO, J. R. Processamento Paralelo Aplicado no Cálculo Parametrizado por Elementos Finitos. In: MOMAG 2004 (1. : 16–19 Ago. 2004: São Paulo). CD-ROM. p. 1–4.
- [37] KANAYAMA, H.; SUGIMOTO S. Effectiveness of A-O method in a Parallel Computing with an Iterative Domain Decomposition Method, *IEEE Transactions on Magnetics*, v. 42, n. 4, p. 539-542, April 2006.
- [38] SESHIMA, N.; et al. Electromagnetic Field Analysis by using Parallel Processing Based on OpenMP. In: THE TWELFTH BIENNIAL IEEE CONFERENCE ON ELECTROMAGNETIC FIELD COMPUTATION - CEFC 2006 (12. : 30 April – 3 May 2006: Miami, Florida, USA). CD-ROM (CEFC – 2006). v. 1, p. 113.
- [39] SAAD, Y. *Iterative Methods for Sparse Linear Systems*. Boston: PWS, 1996.
- [40] VOLLAIRE, C.; NICOLAS, L. Preconditioning techniques for the conjugate gradient solver on a parallel distributed memory computer, *IEEE Transactions on Magnetics*, v. 34, n. 5, p. 3347-3350, Sep. 1998.
- [41] FISCHBORN, M.; KUO-PENG, P.; SADOWSKY, N.; BASTOS, J. P. A.; GAVIOLI, G. A.; TREVISAN, J.; DOTTA, D. LE-MENACH, Y.; PIRIOU, F. Sparse Matrix Storage Formats and Matrix-Vector Products Compared in Parallel Cluster Computing. In: 7th International Symposium on Electric and Magnetic Fields - EMF'2006 (7. : 19-22 Junho, 2006: Aussois, França). CD-ROM (EMF'2006).
- [42] FISCHBORN, M.; KUO-PENG, P.; SADOWSKY, N.; BASTOS, J. P. A.; TREVISAN, J. LU Parallel Preconditioning with Block Intersection Applied to FEM on Computer Clusters. In: THE TWELFTH BIENNIAL IEEE CONFERENCE

- ON ELECTROMAGNETIC FIELD COMPUTATION - CEFC 2006 (12. : 30 April – 3 May 2006: Miami, Florida, USA). CD-ROM (CEFC – 2006). v. 1, p. 401.
- [43] FISCHBORN, M.; KUO-PENG, P.; SADOWSKY, N.; BASTOS, J. P. A.; DULAR, P.; LUZ, M. V. F.; GAVIOLI, G. A. TREVISAN, J. Avaliação de Esquemas de Armazenamento de Matrizes Esparsas em Cluster de Computadores. : MOMAG 2006 (2. : 7-10 Ago. 2006: Belo Horizonte). CD-ROM.
- [44] FISCHBORN, M.; KUO-PENG, P.; SADOWSKY, N.; BASTOS, J. P. A.; TREVISAN, J.; GAVIOLI, G. A. Comparação de Desempenho do BiCG e BiCGStab em Clusters de Computadores. In: MOMAG 2006 (2. : 7-10 Ago. 2006: Belo Horizonte). CD-ROM.
- [45] OLIVEIRA, Ana Margarida de. *Modelagem de Máquinas Elétricas e seus Circuitos Elétricos Associados Utilizando o Método de Elementos Finitos 2D*. Florianópolis, 2004. Tese (Doutorado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.
- [46] FACIUS, Axel. *Iterative Solution of Linear Systems with Improved Arithmetic and Result Verification*. Karlsruhe, 2000. Tese (PhD), Universität Karlsruhe, Alemanha.
- [47] DEMMEL, J. W. *Applied Numerical Linear Algebra*. Philadelphia: SIAM, 1997.
- [48] BARRETT, R. et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: SIAM, 1994.
- [49] FISCHBORN, Marcos. *Computação de Alto Desempenho Aplicada à Análise de Dispositivos Eletromagnéticos*. Florianópolis, 2004. Qualificação (Doutorado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.
- [50] AZEVEDO, J. L. L. *Um Estudo da Simulação da Dinâmica de Sistemas de Energia Elétrica Usando Computadores Paralelos de Alto Desempenho*. Florianópolis, 1996. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.
- [51] V. Eijkhout, Computational Variants of the CGS and BiCGStab Methods, Tech. Rep. CS-94-241, Computer Science Department, The University of Tennessee Knoxville.
- [52] SLEIJPEN, G. L. G. BiCGStab(l) for Linear Equations Involving Unsymmetric Matrices with Complex Spectrum, *IEEE Electronic Transactions on Numerical Analysis*, v. 1, p. 11-32, Sep. 1993.

- [53] MESQUITA, R. C. *Cálculo de campos eletromagnéticos tridimensionais utilizando elementos finitos: magnetostática, quase-estática e aquecimento indutivo*. Florianópolis, 1990. Tese (Doutorado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.
- [54] KERSHAW, D. The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations. *J. Comput. Phys.*, v. 26, p. 43-65, 1978.
- [55] MANTEUFEL, T. A. An incomplete factorization technique for positive definite linear systems. *Numer. Math.*, v. 28, p. 307-327, 1977.
- [56] G. Dahlquist, Å. Björck, “Numerical Methods in Scientific Computing,” <http://www.mai.liu.se/~akbjo/NMbook.html> (acesso Abril 3, 2006).
- [57] ORTEGA, J. M. *Introduction to Parallel and Vector Solution of Linear Systems*. New York: Plenum, 1988.
- [58] “HPCinfo: What is HPC?,” EPCC – The University of Edinburgh, <http://www.epcc.ed.ac.uk/HPCinfo/intro.html> (acesso Ago. 21, 2004).
- [59] A. Manacero Jr, “Fundamentos em Sistemas de Computação,” <http://beast.dcce.ibilce.unesp.br/~aleardo/cursos/cursos.html> (acesso Ago. 9, 2004).
- [60] “Code Optimization,” UVa Information Technology and Communication, <http://www.itc.virginia.edu/research/Optim.html> (acesso Ago. 12, 2004).
- [61] Stephen Booth et al, “Introduction to the T3D,” EPCC – The University of Edinburgh, http://www.epcc.ed.ac.uk/computing/training/document_archive/intro-course/Intro.book_1.html (acesso Ago. 12, 2004).
- [62] P. D. Medeiros, *Introdução ao Processamento Paralelo*, apostila, 1998.
- [63] “NetPIPE - A Network Protocol Independent Performance Evaluator,” <http://www.scl.ameslab.gov/netpipe/> (acesso Maio 25, 2006).
- [64] Definição obtida em, <http://www.netbook.cs.purdue.edu/othrpags/qanda20.htm> (acesso Maio 25, 2006).
- [65] FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computing*, C-21, p. 948-960, 1972.
- [66] A. van der Steen, “The Main Architectural Classes,” TOP 500, <http://www.top500.org/ORSC/2003/architecture.html> (acesso Ago. 22, 2004).
- [67] “HPCinfo: HPC Hardware Technology,” EPCC – The University of Edinburgh,

- [http:// www.epcc.ed.ac.uk/HPCinfo/hardware.html](http://www.epcc.ed.ac.uk/HPCinfo/hardware.html) (acesso Ago. 21, 2004).
- [68] “Myrinet,” TOP 500, <http://www.top500.org/ORSC/2003/myrinet.html> (acesso Ago. 20, 2004).
- [69] “Infiniband,” TOP 500, <http://www.top500.org/ORSC/2003/infiniband.html> (acesso Ago. 20, 2004).
- [70] ANTUNES, Orlando José. *Formulações Conformes e Não-Conformes com Interpolação de ordem Elevada para a Modelagem do Movimento em Máquinas Elétricas*. Florianópolis, 2005. Tese (Doutorado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.
- [71] BASSERMANN A. Conjugate Gradient and Lanczos Methods for Sparse Matrices on Distributed Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, v. 45, p. 46–52, 1997.
- [72] BYCUL, P.; JORDAN, A.; CICHOMSKI, M. A New Version of Conjugate Gradient Method Parallel Implementation. *Proceedings of the International Conference on Parallel Computing in Electrical Engineering*, 2002.
- [73] VOLLAIRE, Christian. *Modélisation de phénomènes électromagnétiques hyperfréquences sur calculateurs parallèles*. Lille, 1997. Tese (Doutorado), Universidade de Lille, França.
- [74] T. Kielman et al, “MAGPIE: MPI’s Collective Communication Operations for Clustered Wide Area Systems,” PPOPP’99, <http://home.tiscali.nl/askeplaat/pubs.html> (acesso Mar. 31, 2006).