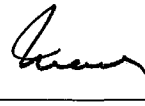
CODIFICAÇÕES ESTRUTURAIS PARA A RESOLUÇÃO ESCALAR DE SISTEMAS LINEARES ESPARSOS SIMÉTRICOS DEFINIDOS POSITIVOS

Ricardo Duarte Arantes

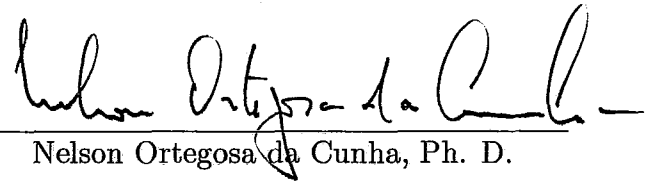
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

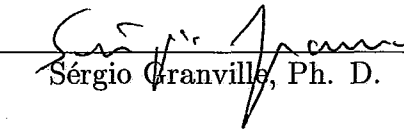


Nelson Maculan Filho, D. Sc.
(presidente)

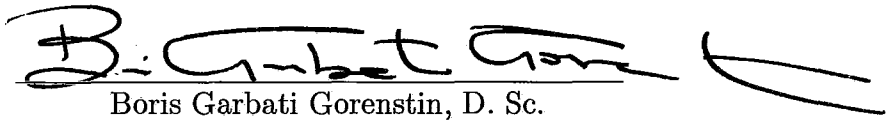
Clovis Caesar Gonzaga, D. Sc.



Nelson Ortegosa da Cunha, Ph. D.



Sérgio Granville, Ph. D.



Boris Garbati Gorenstin, D. Sc.

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 1996

ARANTES, RICARDO DUARTE

Codificações Estruturais para a Resolução Escalar de Sistemas Lineares Esparsos Simétricos Definidos Positivos [Rio de Janeiro] 1996

XV, 243 p., 29.7 cm, (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1996)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Esparsidade 2. Métodos Diretos 3. Álgebra Linear Computacional

4. Computação de Alto Desempenho

I. COPPE/UFRJ II. Título(série).

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

CODIFICAÇÕES ESTRUTURAIS PARA A RESOLUÇÃO ESCALAR DE SISTEMAS LINEARES ESPARSOS SIMÉTRICOS DEFINIDOS POSITIVOS

Ricardo Duarte Arantes

ABRIL, 1996

Orientador: Clovis Caesar Gonzaga
Co-orientador: Nelson Maculan Filho
Programa: Engenharia de Sistemas e Computação

Este trabalho considera a solução computacional por métodos diretos, de sistemas lineares esparsos simétricos, definidos positivos, em arquiteturas monoprocessadas.

As metodologias apresentadas, são aplicáveis à solução de sucessivos sistemas, em que a disposição original de elementos não nulos se mantenha *estruturalmente* constante, ao longo de todo o processo de soluções.

Mediante novas técnicas, baseadas no uso de *codificações estruturais*, para a representação dos padrões de contribuições por *colunas*, é possível obter-se procedimentos do tipo Crout, com um menor percentual de operações auxiliares, na fase numérica do processo de eliminação.

Os algoritmos propostos viabilizam uma significativa redução de *overheads* (como os de indexação indireta), por acessar diretamente os elementos básicos a cada etapa, dipensando (em muitas das variantes) o uso de vetores auxiliares de trabalho.

Uma vantagem da metodologia proposta, é a de permitir um melhor aproveitamento de meios *rápidos* de armazenamento secundário (como memórias do tipo *cache*), de uma forma generalizada, tanto para o caso de contribuições *esparsas* de poucas linhas isoladas, como para o caso de contribuições *supernodais* (unicamente considerado até então pela literatura).

Por mais de duas décadas, métodos esparsos baseados no *desenrolamento completo de loops*, deixaram de ser considerados na prática, em face ao dispêndio excessivo de recursos (proporcionais ao número de operações de ponto flutuante, efetuadas ao longo do processo de solução).

Algumas variantes das técnicas de *codificação estrutural* resultantes deste trabalho, apresentam espaço auxiliar proporcional apenas aos dados de entrada, conseguindo sustentar no entanto, um desempenho computacional similar ao de métodos completamente livres de *loops*, mostrando pois ser plenamente viável, a extensão e o aprimoramento de novas técnicas baseadas nestes conceitos.

Como resultado da presente pesquisa, obteve-se não apenas novas metodologias de cunho *simbólico*, como também introduziu-se melhoramentos em algumas das metodologias convencionais, baseadas no tratamento por *supernodes*. Em particular, conseguiu-se incorporar técnicas *supernodais* em métodos voltados para a geração de fatores por *colunas*, aumentando-se sua eficiência, sem onerar-se o espaço de trabalho, além das estruturas auxiliares usualmente adotadas para a representação *supernodal*.

O desempenho computacional dos algoritmos apresentados, foi mensurado em duas classes distintas de arquiteturas monoprocessadas, uma do tipo CISC *escalar* (IBM PC-486), e outra do tipo RISC *superescalar* (IBM RS/6000-370). Em ambos os casos, comparou-se o desempenho das metodologias propostas, com procedimentos para a resolução escalar direta de sistemas lineares esparsos, convencionalmente adotados pela literatura.

Palavras-Chave: Esparsidade, Métodos Diretos, Álgebra Linear Computacional, Computação de Alto Desempenho.

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

STRUCTURAL CODIFICATIONS FOR THE SCALAR SOLUTION OF SPARSE SYMMETRIC POSITIVE DEFINITE LINEAR SYSTEMS

Ricardo Duarte Arantes

APRIL, 1996

Thesis Supervisor: Clovis Caesar Gonzaga
Co-supervisor: Nelson Maculan Filho
Department: Systems and Computation Engineering

This work consider the computational solution by direct methods, of sparse symmetric, positive definite linear systems, in uniprocessor architectures.

The presented methodologies are applicable to the solution of successive systems, where the original placement of nonzero elements is kept *structurally* constant, abroad the hole solution process.

By new techniques, based on the use of *structural codifications* for the representation of column contributing patterns, it is possible to obtain Crout procedures, with a lower percentage of auxiliary operations, in the numerical phase of the elimination process.

The proposed algorithms allow a significant reduction of *overheads* (like the indirect indexing ones), by directly accessing the basic elements at each stage, turning unnecessary (in most of the variants) the use of auxiliary work vectors.

A point in favor of the proposed methodology, is allowing a better exploitation

of *fast* secondary storage means (like *cache* memories), in a generalized way, for the case of *sparse* contributions of a few isolated rows, and for the case of *supernodal* contributions (solely considered up to now by the literature).

For more than two decades, sparse methods based on the *complete unrolling of loops*, were left over in practice, due to the excessive amount of resources required (proportional to the number of floating point operations, to be effectuated during the solution process).

Some variants of the *structural codification* techniques resulting from this work, exhibits an auxiliary space proportional only to the input data, being able to sustain however, a computational performance similar to the completely *loop-free* methods, showing thus, that the extension and improvement of new techniques based on these concepts, is perfectly viable.

As the result of the current research, not only new *symbolic* methodologies were obtained, but also improvements were introduced in some of the conventional methodologies, based on the treatment of *supernodes*. Particularly it was possible to incorporate *supernodal* techniques in methods directed to the generation of factors by *columns*, increasing their efficiency, without burdening the work space by more than the auxiliary structures commonly adopted for the *supernodal* representation.

The computational performance of the presented algorithms, was measured in two distinct classes of uniprocessor architectures, one of CISC type (IBM PC-486), and the other of RISC *superscalar* type (IBM RS/6000-370). In both cases, the performance of the proposed methodologies was compared against procedures for the direct scalar solution of sparse linear systems, conventionally adopted by the literature.

Key-Words: Sparsity, Direct Methods, Computational Linear Algebra, High Performance Computing.

Índice

1 Introdução **1**
1.1 Considerações Gerais e Notações 2
1.2 Histórico dos Métodos Diretos de Solução 10

2 Conceituação Básica **15**
2.1 Problema Fundamental 15
2.2 Conceituação de Esparsidade 16
2.3 Propriedades Elementares 19
2.4 Métodos de Eliminação Direta 20
2.5 Aspectos Computacionais 37

3 Abordagem Clássica **39**
3.1 Estruturas Elementares de Armazenamento 39
3.2 Ordenamento Visando a Redução de "Fill-In" 47
3.3 Obtenção da Estrutura de Fatores 56
3.4 Resolução Numérica de Sistemas com Mesma Estrutura 62

4 Alternativas para o Aprimoramento da Fase Numérica **76**
4.1 Uso de Listas Simbólicas de Endereços 76
4.2 Exploração da Estrutura de Supernodes 86
4.3 Metodologias Híbridas por Janelas de Eliminação 110

5 Nova Abordagem Proposta **116**
5.1 Princípios Básicos e Motivação Computacional 116
5.2 Codificações Explorando Padrões por Envelope 123
5.3 Codificações Mais Elaboradas 139
5.4 Acumulações via Múltiplos Vetores de Trabalho 157

6 Resultados Computacionais **171**

7 Conclusões **179**

APÊNDICES **184**

A Estruturas de Dados para Fatorações Esparsas **185**

B Métodos Diretos de Eliminação **194**

C Dispêndio de Recursos Computacionais **208**

D Glossário **214**

Referências Bibliográficas **237**

Lista de Figuras

2.1 Cancelamentos por Colunas 22
2.2 Cancelamentos por Linhas 24
2.3 Representação Compacta para Cancelamentos por Colunas 27
2.4 Representação Compacta para Cancelamentos por Linhas 28

3.1 Grafo Associado a uma matriz Esparsa (Simétrica) 40
3.2 Representação Seqüencial Contígua (Adjacências) 41
3.3 Representação Seqüencial Segmentada (Adjacências) 43
3.4 Representação Seqüencial Alternativa (Adjacências) 44
3.5 Representação fornecida para as fases Numéricas 45
3.6 Matriz Original e de Fatores e seus Grafos Associados 47
3.7 Matriz Reordenada e de Fatores e seus Grafos Associados 48
3.8 Primeira Etapa de Transformações sobre o Grafo de Eliminação . . . 51
3.9 Segunda Etapa de Transformações sobre o Grafo de Eliminação . . . 52
3.10 Terceira Etapa de Transformações sobre o Grafo de Eliminação . . . 53
3.11 Grafos de Fatores Resultantes a cada Etapa 56
3.12 Concatenação dos índices de Colunas 57
3.13 Adição de Vetores Esparsos 64
3.14 Eliminação $U^T D U$ gerando fatores por Linhas 67
3.15 Eliminação $U^T D U$ gerando fatores por Colunas 71

4.1 Matriz a ser fatorada por código *Loop-Free* 77
4.2 Particionamento por Supernodes 87
4.3 Árvore de Eliminação 89
4.4 Acumulação de Vetores Supernodais 92
4.5 Janelas de Eliminação 111

5.1 Reordenamento e Codificação por Envelopes 125
5.2 Matriz de Fatores 127
5.3 Árvore de Eliminação 127
5.4 Estrutura Ancestral (*Reduzida*) 128
5.5 Codificação Binária Múltipla 147
5.6 Codificação Múltipla por Envelopes 150
5.7 Reaproveitamento de Códigos Binários 151
5.8 Primeira Etapa de Cancelamentos Explorando Cache 157
5.9 Segunda Etapa de Cancelamentos Explorando Cache 157
5.10 Terceira Etapa de Cancelamentos Explorando Cache 158
5.11 Quarta Etapa de Cancelamentos Explorando Cache 158

5.12 Uso de Múltiplos Vetores de Trabalho 159
5.13 Primeira Etapa de Inicializações 160
5.14 Segunda Etapa de Inicializações 160
5.15 Terceira Etapa de Inicializações 161
5.16 Quarta Etapa de Inicializações 161
5.17 Múltiplos Vetores Reduzidos 163

Lista de Tabelas

4.1 Particionamento por Janelas de Eliminação 112
4.2 Particionamento Alternativo por Janelas de Eliminação 114

5.1 Remapeamento de Códigos Binários via Multiplexação 152
5.2 Codificação via Tabela de Índices 166
5.3 Codificação via Tabela de Huffman 166
5.4 Reordenamento por Colunas 167
5.5 Codificação por Colunas (RLE) e Freqüências (Huffman) 167

6.1 Dados Característicos dos Problemas 172
6.2 Tempo (seg) Várias Fases (PC-486) 173
6.3 Tempo (seg) Geração de Codificações (PC-486) 173
6.4 Tempo (seg) Rotinas Convencionais de Fatoração (PC-486) 174
6.5 (Mflops/seg) Rotinas Convencionais de Fatoração (PC-486) 174
6.6 Tempo (seg) Rotinas Estruturais de Fatoração (PC-486) 175
6.7 (Mflops/seg) Rotinas Estruturais de Fatoração (PC-486) 176
6.8 Tempo (seg) Rotinas Convencionais de Fatoração (RS-6000) 176
6.9 (Mflops/seg) Rotinas Convencionais de Fatoração (RS-6000) 177
6.10 Tempo (seg) Rotinas Estruturais de Fatoração (RS-6000) 177
6.11 (Mflops/seg) Rotinas Estruturais de Fatoração (RS-6000) 178
6.12 Dispêndio (Memória) Estruturas de Rotinas de Fatoração 178

C.1 Dispêndio de Recursos em função de Padrões de Esparsidade 213

Lista de Procedimentos

2.4(1) *Substituição Forward* 30
2.4(2) *Normalização Diagonal* 30
2.4(3) *Substituição Backward* 31
2.4(4) *Processo Unificado de Substituições* 31
3.4(1) *Eliminação via Vetor de Trabalho (Linhas)* 66
3.4(2) *Eliminação via Vetor de Trabalho (Colunas)* 70
3.4(3) *Substituições de Variáveis* 71
3.4(4) *Eliminação e Substituição (Linhas)* 73
4.1(1) *Listas de Endereços (Linhas)* 80
4.1(2) *Listas de Endereços (Colunas)* 82
4.1(3) *Eliminação via Listas de Endereços (Linhas)* 84
4.1(4) *Eliminação via Listas de Endereços (Colunas)* 85
4.2(1) *Eliminação Supernodal (Linhas)* 98
4.2(2) *Eliminação Supernodal (Colunas)* 102
4.2(3) *Substituições Supernodais* 107
5.1(1) *Eliminação via Códigos Binários (Crout)* 122
5.1(2) *Geração da Codificação Binária* 124
5.2(1) *Reordenamento Ancestral* 131
5.2(2) *Eliminação via Códigos por Envelopes (Crout)* 133
5.2(3) *Geração da Codificação por Envelopes* 135
5.2(4) *Eliminação via Códigos Supernodais (Crout)* 138
5.3(1) *Eliminação Binária Recorrente via Multiplexação* 154

Lista de Algoritmos

3.1 *Fatoração Simbólica (Versão Preliminar)* 60
3.2 *Fatoração Simbólica* 61
4.1 *Fatoração Supernodal (Linhas)* 94
4.2 *Fatoração Supernodal (Colunas)* 95
4.3 *Eliminação Híbrida (Janelas)* 113

Lista de Codificações

4.1-1 *Código Loop-Free no estilo proposto por Gustavson* 78
5.1-1 *Produto Escalar via Codificação Binária* 120
5.3-1 *Acumulação Dedicada de Contribuições sobre Fill-In's* 140
5.3-2 *Acumulação Supernodal Múltipla (Linhas)* 142
5.3-3 *Acumulação Supernodal Múltipla (Colunas)* 144
5.3-4 *Acumulação Supernodal Múltipla (Linhas × Colunas)* 145
5.3-5 *Eliminação Supernodal Múltipla via Multiplexação* 146
5.3-6 *Acumulação Binária Múltipla (Colunas)* 149
5.4-1 *Acumulação via Múltiplos Vetores de Trabalho* 162
5.4-2 *Acumulação via Múltiplos Vetores Reduzidos* 165

Lista de Alternativas

2.4(a) *Fatoração $U^T D U$ (Gauss) por Sub-Matrizes* 27
2.4(b) *Fatoração $U^T D U$ (Gauss) por Linhas* 28
2.4(c) *Fatoração $U^T D U$ (Gauss) por Colunas* 29
2.4(d) *Fatoração $U_*^T U_*$ (Cholesky) por Sub-Matrizes* 32
2.4(e) *Fatoração $U_*^T U_*$ (Cholesky) por Linhas* 32
2.4(f) *Fatoração $U_*^T U_*$ (Cholesky) por Colunas* 33
3.2(a) *Ordenamento pelo Critério T1* 54
3.2(b) *Ordenamento pelo Critério T2* 55
3.2(c) *Ordenamento pelo Critério T3* 55
3.4(a) *Adição de Vetores Esparsos via Vetor de Trabalho* 65
4.1(a) *Adição de Vetores Esparsos via Listas de Endereços* 79
4.2(a) *Adição de Vetores Supernodais* 97
4.2(b) *Contribuições Supernodais via Listas Recorrentes* 105
5.1(a) *Fatoração $U^T D U$ (Crout) por Linhas* 117
5.2(a) *Adição de Vetores Esparsos via Listas Restritas* 132

Capítulo 1

Introdução

Neste trabalho são apresentadas novas metodologias de resolução direta de sistemas lineares esparsos simétricos definidos positivos, baseadas na plena exploração de suas características de natureza estrutural, identificadas numa fase preliminar de codificação simbólica de informações.

Em uma vasta gama de arquiteturas seqüenciais (escalares ou super-escalares), os esquemas propostos para a caracterização estrutural de cada sistema, viabilizam a utilização de rotinas de fatoração numérica mais eficientes, por permitir uma redução de *overheads* intrínsecos de processamento, até então presentes nas demais abordagens da literatura.

Estes novos métodos se mostram particularmente atraentes, quando adotados para a resolução de sub-problemas, em áreas de aplicação, cujas metodologias fundamentais de solução, requeiram como etapas auxiliares, a resolução sucessiva de sistemas lineares.

Em face ao ônus adicional da fase de codificação simbólica, as técnicas apresentadas neste trabalho, se mostram vantajosas apenas para aplicações onde a disposição estrutural de elementos não nulos da matriz do sistema se mantenha constante ao longo de todo o processo de fatorações (o que ocorre com freqüência nas áreas de otimização e engenharia).

No final deste capítulo, é apresentado um breve histórico dos principais aprimoramentos incorporados às técnicas de esparsidade, desde os trabalhos fundamentais de Markowitz [79], Tinney [98], Sherman [94] e Gustavson [60], até as metodologias atuais, particularmente voltadas para as novas classes de arquiteturas paralelas e vetoriais.

No segundo capítulo é conceituado o problema básico da solução de sistemas lineares por métodos diretos, sendo apresentadas propriedades elementares da resolução de sistemas definidos positivos, bem como alguns dos aspectos de natureza computacional passíveis de exploração, em função da presença de esparsidade.

No terceiro capítulo são apresentadas estruturas elementares de armazenamento e as etapas fundamentais da abordagem clássica para a resolução de sistemas esparsos, que se mantém igualmente aplicáveis nas demais metodologias consideradas ao longo deste trabalho.

No quarto capítulo são apresentados aprimoramentos, como a exploração do conceito de supernodes e da árvore de caminhos de eliminação, e a utilização de esquemas híbridos por janelas e listas simbólicas de endereços.

No quinto capítulo são apresentadas as novas abordagens propostas, baseadas em técnicas como o reordenamento ancestral de contribuições e a plena exploração das características de cada problema, mediante rotinas de fatoração numérica eficientemente implementadas, a partir das informações de cunho simbólico obtidas numa fase preliminar de codificação estrutural.

No sexto capítulo apresentam-se os desempenhos computacionais obtidos com a implementação de algumas das metodologias consideradas, em duas classes de arquiteturas escalares tipicamente encontráveis nos dias atuais.

Finalmente, no sétimo capítulo apresentam-se as conclusões obtidas e as futuras direções de pesquisa, em função das novas tecnologias introduzidas no corrente trabalho.

1.1 Considerações Gerais e Notações

Ao longo do texto introduzir-se-ão as notações aqui resumidas, complementando-se o material desta seção nos Apêndices A e D (com uma descrição das estruturas de dados adotadas e um glossário das principais terminologias empregadas).

Algumas considerações iniciais e hipóteses básicas são:

O problema fundamental a ser abordado será o da solução computacional (por métodos *diretos* de *eliminação*) de sistemas (algébricos) *irredutíveis* de equações lineares, com a matriz de coeficientes real, *simétrica* e *definida positiva.*

A dimensão dos sistemas a serem tratados, será sempre assumida como n, com todas as matrizes envolvidas em $\mathcal{R}^{n \times n}$ e vetores em $\mathcal{R}^n$ (adotando-se respectivamente letras maiúsculas e minúsculas em itálico para suas representações).

A especificação particular de um elemento isolado, situado em uma linha i e coluna j de uma matriz A, ou linha k de um vetor b, se dará respectivamente por referências ao coeficiente $a_{i,j}$ ou a componente b_k de tais entidades.

A linha i e a coluna j de uma matriz A, serão implicitamente denotadas respectivamente por $a_{i,*}$ e $a_{*,j}$.

O símbolo ^{T} será utilizado para a representação da *Transposta* de matrizes ou vetores.

Algumas formas de matrizes consideradas serão:

- *Positivo Definida:* tal que $x^T A\, x > 0$ para qualquer $x \neq 0,\ x \in \mathcal{R}^n$
- *Simétrica:* tal que $a_{i,j} = a_{j,i}$ para $j < i$
- *Triangular Superior:* tal que $a_{i,j} = 0$ para $j < i$
- *Triangular Inferior:* tal que $a_{i,j} = 0$ para $j > i$

- *Diagonal:* tal que $a_{i,j} = 0$ para $j \neq i$
- *Diagonal Dominante:* tal que $\mid a_{i,i} \mid > \sum_{j \neq i} \mid a_{i,j} \mid$ para todo i
- *Permutação:* obtida a partir do intercâmbio entre linhas ou colunas de uma matriz diagonal, com elementos não nulos unitários

Dentre as matrizes consideradas, destacam-se particularmente as matrizes:

- *Original:* de forma simétrica, contendo os coeficientes de um sistema linear (definido positivo) da forma $A\ x = b$ a ser originalmente solucionado
- *Fatores:* quando não explicitamente citado em contrário, assumida como uma matriz triangular superior U com diagonal unitária (com os elementos $u_{i,j}$ para $j > i$ denominados seus "fatores triangulares")

No caso das matrizes Triangulares, se fará particular distinção entre aquelas com elementos diagonais estritamente positivos, daquelas com elementos diagonais estritamente unitários e das com elementos diagonais não nulos.

Referências a elementos situados na "porção" triangular superior (ou inferior) de uma matriz, estarão restritas apenas aos coeficientes $a_{i,j}$ com $j > i$ (ou $j < i$), excluindo-se em ambos os casos, os elementos situados sobre a diagonal.

Quando não explicitamente indicado, assumir-se-á que as referências aos elementos de uma matriz estejam limitadas particularmente apenas à sua porção Triangular Superior.

Assumir-se-ão como esparsas, matrizes com um percentual relativamente pequeno de elementos não nulos (em comparação com o número total de seus elementos).

Ao longo de todo o texto, a menção a elementos *não nulos* de uma matriz, assumirá sempre a referência a seus elementos *estruturalmente* não nulos, e cuja determinação pode ser efetuada estáticamente a priori (partindo-se apenas das estruturas de dados fornecidas originalmente como entrada).

Eventualmente, com a aplicação de operações elementares sobre linhas ou colunas de uma matriz, alguns destes elementos *potencialmente* não nulos, poderão vir a ser numericamente anulados. O percentual de elementos nesta condição é na maior parte das vezes nulo ou insignificante, razão pela qual não se adotará um tratamento diferenciado a tais elementos. Assumir-se-á pois para fins computacionais, o seu armazenamento físico (como os demais não nulos), efetuando-se explicitamente sobre os mesmos, todas as operações de ponto flutuante associadas.

(No processamento da solução de sistemas por métodos diretos, invàriavelmente efetuam-se *cancelamentos* de elementos, visando conduzir o sistema original a uma forma triangular. Neste caso particular, tais cancelamentos são inteiramente previsíveis e necessários à conclusão dos métodos, não se incorrendo deste modo, em *overheads* computacionais, como no caso dos cancelamentos de potenciais elementos não nulos, descrito no parágrafo anterior).

Em face a modularidade das estratégias para a resolução de sistemas esparsos, ao longo do capítulo 3 será adotada sua sub-divisão por *"fases"* distintas de processamento (como as de *ordenamento*, *fatoração simbólica*, *fatoração numérica* e *substituições de variáveis*), constituídas cada uma, na aplicação de seqüências de *"etapas"* correlatas (implementando localmente os objetivos da fase associada).

Os processos de *Eliminação Direta* (considerados a partir da seção 2.4), podem deste modo ser sub-divididos fundamentalmente em n etapas (denominadas "básicas"), associando-se a cada uma, um índice (correspondente a etapa corrente), especificando uma linha ou coluna *"base"* (da matriz de fatores) a ser particularmente tratada.

Referências a uma dada linha ou coluna base, estarão sempre restritas ao particular conjunto de seus elementos (não nulos), confinados à porção Triangular da matriz (de fatores) associada.

O processo de solução adotado nos métodos de Eliminação, compõe-se basicamente de etapas efetuando um *Cancelamento* de elementos triangulares *inferiores* (por *linhas* ou *colunas*), tendo como objetivo a *Geração de Fatores* triangulares *superiores* (segundo algum padrão associado).

Três formas básicas de *Eliminação* podem ser destacadas:

- *Cancelamentos* por *linhas,* com *Geração de Fatores* por *Linhas*
- *Cancelamentos* por *linhas,* com *Geração de Fatores* por *Colunas*
- *Cancelamentos* por *colunas*, com *Geração de Fatores* por *Sub-Matrizes*

Serão fundamentalmente considerados neste trabalho, procedimentos baseados em *Cancelamentos* inferiores por *linhas*, com *Geração de Fatores* por *Linhas* ou por *Colunas*, respectivamente denominados Métodos de Eliminação por Linhas ou por Colunas.

Eliminações baseadas em cancelamentos inferiores por *colunas*, e geração de fatores por *sub-matrizes*, não serão explicitamente consideradas, exceto por ocasião de sua aplicação à métodos de ordenamento (na seção 3.2) e no tratamento de submatrizes densas (em etapas finais do processo de eliminação) como se verá na seção 3.3.

Ao longo do texto, menções a processos de *Eliminação, Decomposição, Atualização* ou de *Fatoração* serão utilizadas como sinônimo, referindo-se (quando não especificado) particularmente a processos de *Fatoração* (*Triangular*) da matriz Original na forma $U^T D\ U$ (apresentada na seção 2.4), mediante a aplicação de algum método de *Eliminação Direta* (com o cancelamento dos elementos triangulares inferiores por *linhas*).

Os grafos considerados neste trabalho serão *não orientados* (e, via de regra conexos). Referências a representações por grafos de uma matriz, quando não explicitamente especificadas, corresponderão a uma representação por "Grafos Associados" à estrutura de seus elementos não nulos (introduzida na seção 3.1). Serão consideradas também representações por Grafos de Eliminação e por Grafos de Fatores

Resultantes. No caso das matrizes triangulares, suas representações por grafos corresponderão ao de estruturas simetricamente obtidas a partir das matrizes originais.

Baseando-se nas garantias de natureza teórica [41], [55], assumiu-se a estabilidade numérica de todos os processos de fatoração na forma $U^T D\ U$, considerados. Tratamentos especiais para se contornar problemas decorrentes da representação e operação aritmética em ponto flutuante, como erros de arredondamento e de cancelamento, não foram adotados particularmente neste trabalho.

Eventualmente, em função de mal condicionamento nos dados de entrada ou da propagação de erros, as matrizes obtidas como resultado do processo de fatoração, podem vir a se tornar *numericamente indefinidas*, acarretando em alguns casos elementos diagonais não positivos (ou nulos). Nenhum procedimento como os adotados em [53], ou [92], [40] foi empregado no entanto, tendo em vista que o objetivo central do presente trabalho foi o de demonstrar a viabilidade de esquemas mais eficientes sob o ponto de vista de menores tempos de execução. A extensão de algumas das técnicas disponíveis, de modo a se tratar elementarmente as questões de natureza numérica comentadas (sem lançar-se mão de procedimentos para fatoração de matrizes *indefinidas*, como [16], [35]) é possível (sem acarretar sensíveis modificações nos códigos e procedimentos ora apresentados).

Embora conceitualmente similares e representados ao longo do texto sob uma mesma estrutura de dados, na forma de vetores auxiliares **W** (introduzidos na seção 3.4), uma particular distinção faz-se notar entre Vetores de Trabalho (Expandidos) e Vetores de Contribuições. A diferença básica se dá no fato de que no primeiro caso (aplicado tipicamente na geração de fatores por *Linhas*), processam-se as acumulações, assumindo-se o Vetor de Trabalho contendo inicialmente os valores Originais dos elementos sobre a linha base a ser operada. No segundo caso (aplicado normalmente na geração de fatores por *Colunas*), assume-se que o Vetor de Contribuições esteja originalmente zerado em todas as suas posições, processando-se apenas o acumulo das contribuições relativas aos elementos básicos, a serem posteriormente incorporadas.

Menções a *overheads* de processamento assumirão fundamentalmente os *overheads* a nível de *tempo de execução* (em detrimento a eventuais *overheads* em espaço de armazenamento decorrentes da adoção de estruturas de dados auxiliares).

As estruturas de dados (na forma de vetores) introduzidas ao longo do texto e computacionalmente implementadas nos procedimentos propostos, virão sempre representadas por identificadores maiúsculos em negrito, com o uso de expressões entre colchetes para se denotar especificamente alguma de suas componentes isoladas em particular. Variáveis escalares por sua vez, serão sempre expressas por identificadores minúsculos em itálico.

Nas esquematizações de estruturas como as de vetores de apontadores para posições randômicas de memória, as localizações apontadas estarão sempre representadas por valores em itálico entre colchetes. No caso de vetores de inteiros, contendo índices estruturais (como os de colunas de coeficientes não nulos de cada linha de uma matriz), optou-se por sua representação entre parênteses. Valores numéricos em ponto flutuante, serão por sua vez indicados pela presença do ponto decimal como terminador (de modo a diferenciá-los de índices inteiros).

Como regra geral, nas matrizes apresentadas, adotou-se por convenção a representação das posições correspondentes a seus elementos não nulos por símbolos como "•", "×", "□" ou "△". A indicação de elementos originalmente não nulos, cancelados em alguma etapa do processo de eliminação se dará por sua vez, mediante o símbolo "o". No caso dos elementos denominados de *fill-in*, a serem conceituados na seção 2.4 (possuindo valor nulo na matriz original de coeficientes e que venham a assumir valores não nulos na matriz de fatores resultante), adotou-se para sua representação o símbolo "⊗".

Na ilustração de matrizes de fatores envolvidas, de modo a se facilitar a percepção das linhas efetivamente contribuintes para a formação de cada linha base, optou-se por representar em cada figura as estruturas de U^T, D e U "conjugadas" numa mesma matriz, a partir da representação de uma estrutura simétrica (idêntica a de $U^T + U$).

Com relação aos processos computacionais listados, optou-se por classificá-los segundo o grau de detalhamento utilizado em sua apresentação, denotando-se por *Algoritmos* os processos (a nível de pseudo-código) com um padrão de abstração mais elevado, e por *Procedimentos* os processos completamente detalhados (e de fácil transporte para alguma linguagem computacional).

Após a listagem de cada processo, uma descrição detalhada de alguns seus passos (alfabeticamente destacados, e numerados em itálico), é normalmente apresentada, adotando-se o símbolo □ de modo a indicar o término do detalhamento de seus principais passos.

Na presença de variantes para determinados Procedimentos ou Algoritmos, ou para trechos específicos de suas sub-tarefas, optou-se por sua classificação sob a forma de *Alternativas*.

Finalmente nos casos da listagem de trechos de programas em alguma linguagem computacional específica (como FORTRAN ou C/C^{++}) ou de algumas sub-tarefas (não detalhadas por completo) na forma de pseudo-código, optou-se por sua apresentação na forma de *Codificações*.

Enumera-se a seguir uma compilação das principais notações e estruturas de dados utilizadas (agrupadas pelas respectivas seções onde estarão sendo introduzidas ou referenciadas).

Seção 2.1:

A: *Matriz de um Sistema de Equações Lineares*

x: *Vetor Solução de um Sistema de Equações Lineares*

b: *Vetor Lado Direito de um Sistema de Equações Lineares*

D: *Matriz Diagonal (com Elementos Diagonais Estritamente Positivos)*

U: *Matriz Triangular Superior (com Diagonal Unitária)*

Seção 2.2:

$S(A)$: *Estrutura de Elementos Não Nulos de uma Matriz (excluindo-se sua diagonal)*

$S_{\text{sup}}(A)$: *Estrutura de Elementos Não Nulos da Porção Triangular Superior de uma Matriz*

$S^{(i)}(A)$: *Estrutura de Elementos Não Nulos da i'ezima Linha de uma Matriz (excluindo-se o elemento diagonal)*

$S^{(i)}_{\text{sup}}(A)$: *Estrutura de Elementos Não Nulos da i'ezima Linha, situados na Porção Triangular Superior de uma Matriz*

$\eta(A)$: *Número de Elementos Não Nulos de uma Matriz (excluindo-se sua diagonal)*

$\eta_{\text{sup}}(A)$: *Número de Elementos Não Nulos da Porção Triangular Superior de uma Matriz*

$\eta^{(i)}(A)$: *Número de Elementos Não Nulos da i'ezima Linha de uma Matriz (excluindo-se o elemento diagonal)*

$\eta^{(i)}_{\text{sup}}(A)$: *Número de Elementos Não Nulos da i'ezima Linha, situados na Porção Triangular Superior de uma Matriz*

$\rho(A)$: *Densidade de Elementos Não Nulos de uma Matriz*

Seção 2.4:

L: *Matriz Triangular Inferior (com Diagonal Unitária)*

D: *Matriz Diagonal (com elementos Estritamente Positivos)*

U: *Matriz Triangular Superior (com Diagonal Unitária)*

$D_{\prime}$: *Matriz Diagonal (com elementos Não Nulos)*

$U_{\prime}$: *Matriz Triangular Superior (com Diagonal Não Nula)*

L_*: *Matriz Triangular Inferior (com Diagonal Estritamente Positiva)*

U_*: *Matriz Triangular Superior (com Diagonal Estritamente Positiva)*

$L^{(k)}$: *Matriz Elementar Inferior da k'ezima Etapa do Processo de Eliminação*

$A^{(k)}$: *Matriz Transformada após a k'ezima Etapa do Processo de Eliminação*

P: *Matriz de Permutações*

Seção 3.1:

G_A: *Grafo Associado a Estrutura de Elementos Não Nulos de uma Matriz A*

IA: *Apontadores para o Início da representação dos elementos não nulos de cada linha na porção triangular superior da matriz original A*

IAF: *Apontadores para o Final da representação dos elementos não nulos de cada linha na porção triangular superior da matriz original A*

NA: *Número de elementos não nulos da representação de cada linha na porção triangular superior da matriz original A*

JA: *Índices das Colunas dos elementos não nulos de cada linha na porção triangular superior da matriz original A*

AN: *Valores Numéricos dos elementos não nulos de cada linha na porção triangular superior da matriz original A*

DN: *Valores Numéricos dos elementos Diagonais da matriz original A*

IU: *Apontadores para o Início da representação dos elementos não nulos de cada linha da matriz de fatores U*

IUF: *Apontadores para o Final da representação dos elementos não nulos de cada linha da matriz de fatores U*

NU: *Número de elementos não nulos da representação de cada linha da matriz de fatores U*

JU: *Índices das Colunas dos elementos não nulos de cada linha da matriz de fatores U*

UN: *Valores Numéricos dos elementos não nulos de cada linha da matriz de fatores U*

DI: *Valores numéricos Inversos dos elementos Diagonais da matriz de fatores D*

IUT: *Apontadores para o Início da representação dos elementos não nulos de cada linha da matriz de fatores Transposta* U^T

IUTF: *Apontadores para o Final da representação dos elementos não nulos de cada linha da matriz de fatores Transposta* U^T

JUT: *Índices das Colunas dos elementos não nulos de cada linha da matriz de fatores Transposta* U^T

Seção 3.2:

G_U^*: *Grafo de Fatores (associado a estrutura de* $U^T + U$)

δ_i: *Grau (de conexões) de um dado nó i de um grafo não orientado*

v_i: *Valência de um dado nó i no grafo associado a representação de fatores da i'ezima etapa de um processo de cancelamento por Colunas*

$\mathcal{V}_U$: *Valência Total do grafo associado a representação de fatores U resultantes ao final de um processo de cancelamento*

$G_A^{(i)}$: *Grafo de Eliminação associado a i'ezima etapa de um processo de cancelamento por Colunas (partindo-se da estrutura* $G_A^{(0)} = G_A$ *do Grafo Associado da matriz Original A)*

$G_U^{*(i)}$: *Grafo da estrutura (parcial) de Fatores U Resultantes, gerados ao se concluir as primeiras i etapas básicas de um processo de cancelamento por Colunas*

IORD: *Vetor contendo a Ordem de permutação simétrica de linhas e colunas, visando a redução do número de fill-in's da matriz de fatores U*

Seção 3.3:

$p(i)$: *Função indicadora da Coluna associada ao Primeiro elemento Não Nulo da i'ezima linha da matriz de fatores U*

$S_{\text{sup}}^{(i)}(A)$: *Estrutura de Elementos Não Nulos da i'ezima linha de A, situados sobre sua porção Triangular Superior.*

$S^{(i)}(U)$: *Estrutura de Fatores Não Nulos da i'ezima linha de U (gerada a partir da concatenação das estruturas de suas linhas Características contribuintes)*

$C_*^{(i)}$: *Conjunto de linhas Características da matriz de fatores U (geradoras do espectro de colunas de todos os elementos não nulos contribuintes sobre a linha base) na i'ezima etapa de um processo de cancelamento por Linhas*

Seção 3.4:

piv: *Variável a conter temporariamente o valor do elemento diagonal corrente, e a acumular as contribuições por ele sofridas ao longo de cada etapa*

W: *Vetor de trabalho "expandido" ou de contribuições, contendo temporariamente no primeiro caso os valores numéricos dos elementos não nulos originais da linha base, descompactados segundo o índice de suas colunas associadas (a serem posteriormente operadas), e no segundo caso diretamente os valores das contribuições sofridas a serem incorporadas na coluna base*

IUP: *Apontadores (dinâmicos) para a posição do Primeiro elemento na representação de cada linha da matriz de fatores U, a partir da qual se efetuarão as contribuições sobre a linha básica corrente (em processos de geração dos fatores por Linhas)*

IUPF: *Apontadores (dinâmicos) para a posição do Último elemento na representação de cada linha da matriz de fatores U, até onde se efetuarão as contribuições sobre a linha básica corrente (em processos de geração retardada dos fatores por Colunas)*

Seção 4.1:

IW: *Vetor de apontadores "expandido", contendo temporariamente a cada etapa, as Posições de armazenamento dos fatores não nulos da linha base corrente, descompactadas segundo o índice de suas colunas associadas*

PLIN: *Lista "simbólica" com uma seqüência de apontadores para as Posições de armazenamento dos elementos não nulos da Linha Base corrente, a serem acessados durante o recebimento de contribuições oriundas de linhas antecessoras (a cada etapa do processo de eliminação)*

PCOL: *Lista "simbólica" com uma seqüência de apontadores para as Posições de armazenamento dos elementos não nulos da Coluna Base corrente, a serem acessados durante o recebimento de contribuições oriundas de linhas antecessoras (a cada etapa do processo de eliminação)*

Seção 4.3:

nwind: *Número de "Janelas" de Eliminação (constituídas por porções de linhas básicas contíguas e de característica estrutural similar), adotadas na sub-divisão do processo de eliminação triangular*

WINTYP: *Vetor especificando o Tipo de processamento a ser adotado para cada uma das Janelas de Eliminação associadas*

WINROW: *Vetor contendo a linha Base Inicial (de cada Janela)*

WINROWF: *Vetor contendo a linha Base Final (de cada Janela)*

CONTRB: *Vetor contendo a linha Contribuinte Inicial (a ser considerada no processamento de cada Janela)*

CONTRBF: *Vetor contendo a linha Contribuinte Final (a ser considerada no processamento de cada Janela)*

Seção 5.1:

LBIN: *Lista simbólica de Códigos associados ao padrão de elementos não nulos de cada coluna de contribuições sobre a linha base da matriz de fatores U, a serem posteriormente decodificados, fornecendo as seqüências de operações de acumulo por produto escalar (para fases numéricas projetadas de modo a explorar tal informação)*

Seção 5.2:

LENV: *Lista simbólica de Códigos espelhando padrões de contribuição por colunas, na forma por Envelopes*

LSUP: *Lista simbólica de Códigos espelhando padrões de contribuição (comuns a todo um grupo de colunas contribuintes) de forma Supernodal sobre a linha base*

PJ: *Lista simbólica Restrita, com o endereço Inicial de um grupo de posições contíguas sobre a linha base, a serem consecutivamente acessadas e operadas, numa mesma etapa do processo de contribuições*

PJF: *Lista simbólica Restrita, com o endereço Final de um grupo de posições contíguas sobre a linha base, a serem consecutivamente acessadas e operadas, numa mesma etapa do processo de contribuições*

Seção 5.3:

LMPLX: *Lista simbólica de Códigos de Multiplexação, possibilitando uma translação para os trechos definitivos de programa a serem executados, a partir de um remapeamento encadeado de desvios condicionais computados*

LEXCL: *Lista simbólica de Padrões de Exclusão, utilizados em processos de remapeamento de desvios, de modo a se tratar cada um dos possíveis sub-casos de padrões de contribuição a serem excluídos no reprocessamento de um dado código*

Seção 5.4:

W_k: *Vetores de Trabalho Expandidos, associados a cada uma das linhas contribuintes a serem processadas simultaneamente por ocasião do acumulo de produtos escalares sobre a linha base (em função dos padrões de codificação adotados para cada coluna)*

UW_k: *Vetores de Trabalho Reduzidos, associados a cada uma das linhas contribuintes a serem processadas simultaneamente por ocasião do acumulo de produtos escalares sobre a linha base (em função dos padrões de codificação adotados para cada coluna)*

1.2 Histórico dos Métodos Diretos de Solução

Nesta seção, apresenta-se um breve histórico dos aprimoramentos introduzidos ao longo das últimas duas décadas, nos métodos para solução direta de sistemas espar-

sos de equações lineares, em função da introdução de novas arquiteturas, descoberta de novos algoritmos ou o estabelecimento de técnicas de implementação mais eficazes.

A resolução de sistemas lineares esparsos de porte elevado como nos mostra Duff [27], [34] é um problema frequentemente encontrado em uma extensa gama de ramos da engenharia, tais como: controle de tráfego aéreo, astrofísica, engenharia química, simulação de circuitos, demografia, modelagem econômica, projeto de reatores nucleares, fluxo de potência ótimo, modelagem estocástica, espalhamento acústico, modelagem de reservatórios de petróleo, solução de equações diferenciais parciais e ordinárias, problemas de Navier-Stokes, oceanografia, problemas estruturais de engenharia civil e de modelagem por malhas, elementos finitos, redes de sistemas de potência, problemas estruturais decorrentes de projetos da indústria naval, aeroespacial e automobilística, programação matemática, linear e não linear, resolução de problemas de mínimos quadrados em otimização e estatística, entre outras áreas.

O caso particular de sistemas esparsos simétricos, definidos positivos é notado em muitas destas áreas, destacando-se particularmente as de equações diferenciais, cálculo estrutural, modelagem por malhas, elementos finitos, redes de sistemas de potência, fluxo de potência ótimo, mínimos quadrados, otimização não linear, e mais recentemente programação linear (com o advento de abordagens baseadas em algoritmos de pontos interiores [67]).

O acompanhamento histórico a seguir não se manterá restrito apenas ao caso simétrico positivo definido, abrangendo também algumas das evoluções incorporadas no caso geral para o tratamento de sistemas quadrados assimétricos.

A primeira aplicação das técnicas de esparsidade para a solução de grandes sistemas lineares, remonta à década de 50, devendo-se aos esforços de pesquisa nas áreas de elementos finitos e programação linear. Técnicas de redução de banda e o critério de Markowitz [79] para o pivoteamento de equações pertencem a esta fase em particular.

Para os problemas tipicamente encontrados na área de potência, as técnicas de redução de banda não se mostraram adequadas. A solução de sistemas lineares esparsos nesta época invariavelmente implicava na utilização dos métodos iterativos clássicos, [99], [100], [102] com uma taxa de convergência na maioria das vezes baixa.

No final da década de 60, após a publicação do revolucionário artigo de Tinney e Walker [98], a solução de problemas esparsos mediante o uso de métodos diretos de solução passou a ser uma realidade, graças ao bom desempenho alcançado pela heurística de menor grau, conhecida como o critério #2 de Tinney (ou simplesmente T2).

Outros critérios mais onerosos que o segundo proposto por Tinney foram experimentados, como por exemplo o terceiro, baseado na minimização local do número de *fill-in's* (eficientemente implementado em [19]).

O critério T2 consagrou-se inicialmente para as aplicações encontradas na área de potência, podendo ser encarado como uma particularização do critério de Markowitz, para o caso de matrizes simétricas.

O segundo critério acabou sendo paulatinamente adotado na maioria das áreas de aplicação e aprimorado principalmente no que diz respeito ao desempate na escolha de candidatos com o mesmo grau. (Uma exposição de alguns dos mais bem sucedidos aprimoramentos introduzidos ao longo das últimas duas décadas, é apresentada em [49]).

Paralelamente à evolução das técnicas de ordenamento para a esparsidade, os trabalhos pioneiros de Tinney [98] e de Gustavson [60], [59], viabilizaram a solução direta de grandes sistemas (na época, com dimensões em torno de alguns milhares de equações).

Do início dos anos 70 remontam dois conceitos fundamentais e aplicados desde então ao processamento esparso: a utilização de vetores de trabalho [59] e de esquemas de pré-processamento simbólico [60]. O primeiro acarretando *overheads* em termos de tempo de execução, enquanto que o segundo invariavelmente introduzindo *overheads* nem sempre toleráveis no espaço de armazenamento.

Em esparsidade a dicotomia tempo $\times$ espaço é uma presença constante, e que depende fundamentalmente do tipo de estratégia de solução adotada (métodos diretos, iterativos ou híbridos) e dos recursos disponíveis para a solução (memória, tempo de CPU e tipo de arquitetura). Em muitas situações um ponto de balanceamento ótimo encontra-se numa adaptação de várias técnicas, ajustadas especificamente para cada problema em particular.

Assim, a utilização de esquemas híbridos, aproveitando-se de técnicas densas e esparsas, resultou em abordagens nas quais dois ou mais níveis de processamento são considerados de acordo com a densidade de cada sub-matriz encontrada durante o processo de eliminação, como em [20], [5] e [28].

Conceitos como os de tipos de variabilidade também remontam a esta época em particular, enquanto as idéias de processamento por blocos introduzidas por Hachtel [62], faziam sua primeira aparição no cenário das técnicas de esparsidade.

No início dos anos 80, implementações esparsas extensivamente testadas como as de Waterloo [44], [48], Yale [38] e Harwell [61], [30] consolidaram-se na área de solução por métodos diretos, tomando como base estruturas de dados já então padronizadas [39], [31], [43] (fundamentadas no uso de vetores de trabalho/acumulação) e critérios de ordenamento como o de Tinney #2 ou *nested dissection* (no caso de problemas na área de cálculo estrutural).

Com a implantação das primeiras arquiteturas vetoriais [65] em meados desta década, as estratégias para a solução eficiente de sistemas esparsos de grande porte, acabaram tendo de ser completamente reformuladas, visto ser na época, substancialmente inferior, o desempenho de códigos baseados em acessos indiretos a memória, nos quais a quase totalidade dos métodos esparsos desenvolvidos até então se baseavam.

A partir desta restrição computacional, os métodos multi-frontais [36], [33], [76], acabaram por se firmar definitivamente como a melhor alternativa para implementação nos supercomputadores desta geração, por viablizarem uma maior eficiência vetorial dos códigos de eliminação. Tal sendo obtido, mediante a montagem e eliminação em *frontes* simultâneas de trabalho, onde as sub-matrizes *frontais*

associadas, por possuírem características eminentemente densas, possibilitarem a exploração de técnicas mais avançadas de eliminação (além de se mostrarem dominantes em esforço computacional, sobre o restante das operações efetuadas para se chegar à solução).

A necessidade de códigos de fatoração ainda mais eficientes continuava a existir em muitas áreas de aplicação, especialmente naquelas em que apenas poucos elementos do sistema sofriam alterações em valor numérico entre sucessivas soluções de sistemas com matrizes de mesma estrutura (um caso típico na área de otimização).

Desta necessidade surgiram os métodos de refatoração parcial baseados em *sparse vectors* [14], [58] e [57], [17] que continuam a despertar interesse até hoje, especialmente nas fases de ordenamento e mapeamento, em arquiteturas do tipo paralelo.

Assim, critérios baseados na minimização da altura da árvore de caminhos de eliminação [58], [57], passaram a ser considerados como novos critérios de desempate para a heurística de menor grau.

Para o final dos anos 80 ainda estavam reservados melhoramentos, explorando características de *hardware* encontradas nas arquiteturas mais avançadas disponíveis.

O redescobrimento da abordagem via *supernodes* [8], [63] (onde blocos de elementos não nulos contíguos são tratados como uma única entidade na estrutura de representação esparsa) veio a ser merecedor de um importante prêmio de supercomputação em 1988 [15] (com uma implementação vetorial / paralela eficiente, baseada na exploração desse conceito).

Finalmente uma abordagem importante proposta já no início da década de 90, é a da representação particionada para a inversa da matriz de fatores [2], [3].

A característica inovadora deste método reside no fato de permitir a solução de múltiplos sistemas (com a mesma matriz de coeficientes e diferentes vetores "lado direito") mediante a utilização de produtos do tipo matriz $\times$ vetor (no lugar do processo de retro-substituição usual, e que em arquiteturas paralelas não pode ser explorado de forma tão eficiente).

Paralelamente aos avanços na fase numérica, a exploração de informações de natureza simbólica também vem sendo alvo de recente investigação, com a aplicação de técnicas de pré-processamento, especialmente em implementações na área de programação linear baseadas nos métodos de pontos interiores [1], [80].

Cabe pois mencionar o substancial avanço obtido ao se passar de uma abordagem clássica baseada em listas encadeadas e criação dinâmica de *fill-in's*, para a abordagem tradicional dos dias atuais, baseada em fases distintas de processamento, com a geração prévia da estrutura de fatores resultantes e a adoção de uma representação supernodal seqüencial durante a fase numérica de solução.

Recentes conquistas na fase de fatoração simbólica apontam cada vez mais para o uso de estruturas quase ótimas [45], tendo por base a exploração da árvore de caminhos de eliminação [74]. Da mesma forma, não se deve deixar de mencionar a descoberta de esquemas compactos de armazenamento [69] que em alguns casos superam em pelo menos uma ordem de grandeza a economia obtida por esquemas de compressão tradicionais como o de Shermann [94], [32], amplamente consagrado

nas últimas duas décadas.

A minimização de *overheads* como paginações [72], utilização eficiente de memórias *cache* [13], e o uso de políticas eficazes de gerenciamento das informações armazenadas em meio secundário [50], [71], [70], também se mostra cada vez mais determinante no sucesso das novas implementações.

Em face à considerável dimensão dos problemas em algumas das áreas de aplicação, alternativas para a redução do espaço total de armazenamento via o uso de estruturas dinâmicas como em [11], [10] vem merecendo a atenção da literatura ao longo dos últimos anos.

A fatoração esparsa eficiente nas arquiteturas vetoriais ou paralelas existentes [7], [63], constitui um dos desafios desta década, servindo de base para a implementação eficaz de métodos de otimização não linear em geral [21], [104], [103].

Até mesmo a aplicação de antigas técnicas como a de desenrolamento de *loops* [25] encontra terreno fértil na busca por menores *overheads* intrínsecos de processamento.

O problema da fatoração eficiente nas novas arquiteturas vem sendo merecedor por si só, de uma atenção redobrada desde meados da década de 80. "Clássicos" da análise numérica como os de Golub e Van Loan [55] dedicam agora capítulos inteiros à fatoração em diversos tipos de arquiteturas, como em [23] e [83].

Para finalizar esta seção, como colocou Duff em [29], reafirma-se:

> *O futuro aponta cada vez mais para o constante desenvolvimento e a plena utilização de novas técnicas de esparsidade, sendo extremamente promissor não apenas nos dias atuais, como num número significativo de gerações ainda por vir.*

Capítulo 2

Conceituação Básica

Neste capítulo conceituará-se o problema fundamental da resolução de sistemas lineares esparsos, enumerando-se algumas das propriedades particulares aos sistemas simétricos, definidos positivos. Serão também apresentadas caracterizações do grau de esparsidade de uma matriz, e aspectos de natureza computacional, de vital importância para a exploração dos padrões de esparsidade presentes nos sistemas a serem solucionados.

2.1 Problema Fundamental

O problema a ser abordado neste trabalho é o da solução computacional em arquiteturas *seqüenciais* (CISC escalares ou RISC super-escalares), de sistemas *irredutíveis* de equações algébricas lineares da forma:

$$A\,x = b \tag{2.1}$$

com a matriz de coeficientes A *esparsa*, *simétrica* e *definida positiva*, e os respectivos vetores solução e lado direito x, b *densos*.

A solução para este problema pode ser obtida mediante duas famílias distintas de métodos:

- *diretos* (ou de eliminação)
- *iterativos* (clássicos ou pré-condicionados)

No presente trabalho, apenas métodos *diretos*, baseados em fatorações da matriz de coeficientes serão considerados. (Para maiores referências na área de resolução por métodos iterativos, reporta-se a [1], [12], [23], [54], [56], [68], [83], [84], [85], [99], [100], [102]).

A solução por métodos diretos fundamenta-se no fato de ser possível encontrar-se univocamente uma fatoração da matriz A do sistema original, na forma de matrizes U, D (respectivamente triangular superior com diagonal unitária e diagonal com elementos estritamente positivos) tais que:

$$U^T D\,U = A \tag{2.2}$$

Uma vez concluído este processo de eliminação, a solução para o problema original (2.1) pode ser trivialmente obtida mediante esquemas de "substituição" *forward*, *diagonal* e *backward* (a serem vistos um pouco mais detalhadamente na seção 2.4) solucionando-se os seguintes sub-sistemas:

$$U^T x'' = b \tag{2.3}$$

$$D\ x' = x'' \tag{2.4}$$

$$U\ x = x' \tag{2.5}$$

A atenção ao longo do texto, estará voltada basicamente para a obtenção eficiente dos fatores em (2.2), pelo fato dos esquemas subseqüentes de substituição se mostrarem de fácil implementação, nas classes de arquiteturas particularmente consideradas.

2.2 Conceituação de Esparsidade

Apresenta-se a seguir, um conjunto de medidas e caracterizações para o grau de esparsidade de uma matriz (especificamente para o caso simétrico, positivo definido).

Tal conceituação é importante para que se possa posteriormente observar alguns aspectos de caráter computacional passíveis de exploração, em face a natureza estruturalmente esparsa das matrizes envolvidas no processo de eliminação.

Em todas as definições a seguir, assume-se (como notado na seção 1.1) que a referência aos elementos de uma matriz, exclua particularmente aqueles situados sobre sua diagonal.

Menções a elementos não nulos, na verdade corresponderão a todos os elementos "potencialmente" não nulos, cuja localização (estrutural) pode ser estáticamente conhecida "a priori" (como resultado da aplicação de operações elementares), não se descartando porém a hipótese de que no decorrer de eventuais alterações, os valores de alguns coeficientes venham a ser coincidentalmente anulados.

Introduzem-se inicialmente noções relativas a *estrutura* (de elementos não nulos) de uma matriz, a partir de:

Definição 2.2.1 (Estrutura de Elementos Não Nulos) *Define-se por* $S(A)$, *para uma matriz* A, *o conjunto* $\{\ (i,j)\ |\ a_{i,j} \neq 0,\ j \neq i\ \}$ *dos pares (linha,coluna) associados a seus coeficientes não nulos.*

Definição 2.2.2 (Estrutura Superior de Elementos Não Nulos) *Define-se por* $S_{\text{sup}}(A)$, *para uma matriz* A, *o conjunto* $\{\ (i,j)\ |\ a_{i,j} \neq 0,\ j > i\ \}$ *dos pares (linha,coluna) associados aos coeficientes não nulos de sua porção triangular superior.*

Definição 2.2.3 (Estrutura de Elementos de uma Linha) *Define-se por* $S^{(i)}(A)$, *para uma matriz* A, *o conjunto* $\{\ j \neq i\ |\ a_{i,j} \neq 0\ \}$ *das colunas associadas aos coeficientes não nulos de sua i'ézima linha.*

Definição 2.2.4 (Estrutura Superior de Elementos de uma Linha) *Define-se por* $S^{(i)}_{\text{sup}}(A)$, *para uma matriz* A, *o conjunto* $\{ j > i \mid a_{i,j} \neq 0 \}$ *de colunas associadas aos coeficientes não nulos de sua* i*'ézima linha, situados em sua porção triangular superior.*

O conceito de *esparsidade* não se encontra rigidamente definido na literatura, sendo na maioria das vezes intuitivamente associado a matrizes com um número reduzido de elementos não nulos, em comparação com o número total de seus elementos.

Em função disto, mostra-se conveniente a definição de algumas funções indicadoras como:

Definição 2.2.5 (Número de Elementos Não Nulos) *Define-se por* $\eta(A)$, *para uma matriz* A, *a cardinalidade* $\mid S(A) \mid$, *correspondendo ao número de seus elementos não nulos.*

Definição 2.2.6 (Número de Elementos Superiores) *Define-se por* $\eta_{\text{sup}}(A)$, *para uma matriz* A, *a cardinalidade* $\mid S_{\text{sup}}(A) \mid$, *correspondendo ao número de elementos não nulos de sua porção triangular superior.*

Definição 2.2.7 (Número de Elementos por Linha) *Define-se por* $\eta^{(i)}(A)$, *para uma matriz* A, *a cardinalidade* $\mid S^{(i)}(A) \mid$, *correspondendo ao número de elementos não nulos de sua* i*'ézima linha.*

Definição 2.2.8 (Número de Elementos Superiores por Linha) *Define-se por* $\eta^{(i)}_{\text{sup}}(A)$, *para uma matriz* A, *a cardinalidade* $\mid S^{(i)}_{\text{sup}}(A) \mid$, *correspondendo ao número de elementos não nulos de sua* i*'ézima linha, situados em sua porção triangular superior.*

Estas medidas isoladamente, não podem ser usadas para uma caracterização (independente de escala) do grau de esparsidade de uma matriz, pois expressam apenas um número absoluto de seus elementos não nulos, sem levar-se em conta sua dimensão ou número total de elementos.

Uma quantificação para a noção de um percentual reduzido de elementos não nulos pode ser formalizada por uma medida característica do grau de *densidade* matricial, através de:

Definição 2.2.9 (Densidade de uma Matriz) *Define-se como densidade* $\rho(A)$, *para uma matriz* A, *a razão* $(\eta(A) + n)$ / n^2, *entre o número de seus elementos não nulos e o número total de seus elementos.*

Em função desta medida, pode-se enumerar algumas classes particulares de matrizes como por exemplo:

Definição 2.2.10 (Matriz Completamente Esparsa) *Define-se uma matriz como "completamente esparsa" ou "diagonal", caso apenas os elementos de sua diagonal possuam valor numérico não nulo.*

Definição 2.2.11 (Matriz Esparsa) *Define-se uma matriz como "esparsa", quando sua densidade for inferior a um certo percentual, aceito como relativamente reduzido (na maioria dos casos inferior a 1%, podendo em função da sua dimensão, adotar-se percentuais de densidade mais elevados ou reduzidos).*

Definição 2.2.12 (Matriz Quase Densa) *Define-se uma matriz como "quase densa", quando sua densidade for superior a um certo percentual aceito como relativamente elevado (na maioria dos casos superior a 90%, podendo em função da sua dimensão, adotar-se percentuais de densidade mais reduzidos ou elevados).*

Definição 2.2.13 (Matriz Completamente Densa) *Define-se uma matriz como "completamente densa" ou "cheia", caso todos os seus elementos possuam valor numérico não nulo.*

Estas definições porém, (com a exceção dos casos completamente densos ou esparsos) não são suficientes para um nítido delineamento entre esparsidade e não esparsidade.

Na prática, o conceito de esparsidade não pode ser caracterizado unicamente por medidas como as de densidade, pois deve-se levar em conta aspectos relevantes de natureza computacional como a forma de representação a ser adotada para o armazenamento dos elementos matriciais, e o modo de tratamento para as operações aritméticas sobre elementos originalmente nulos.

Em vista disso, o que se pode dizer é que uma matriz estará sendo tratada de forma esparsa, quando de algum modo se estiver explorando alguma das particularidades de sua estrutura, de modo a se obter algum benefício computacional, quer na redução de seu espaço de armazenamento, e/ou no número de operações aritméticas efetuadas durante o processo de sua decomposição.

Em face à não unicidade de caracterizações para o conceito de esparsidade, enumeram-se a seguir, algumas definições complementares apresentadas em [87]:

Definição 2.2.14 (Matriz Esparsa) *Define-se uma matriz de dimensão n como "esparsa", se o número total de seus elementos não nulos for limitado superiormente por $O(n)$.*

Definição 2.2.15 (Matriz Esparsa) *Define-se uma matriz de dimensão n como "esparsa", se o número de seus elementos não nulos em cada linha for igual ou inferior a um dado valor κ (relativamente pequeno em comparação a n).*

Definição 2.2.16 (Matriz Esparsa) *Define-se uma matriz de dimensão n como "esparsa", se o número total de seus elementos não nulos for limitado superiormente por $n^{1+\gamma}$ com $0 \leq \gamma \ll 1$.*

Estas definições são mais restringentes do que a 2.2.11 e sob certa forma delineiam com maior fidelidade a noção de matrizes esparsas computacionalmente *tratáveis*, por implicitamente associar graus de densidade decrescentes em função do aumento da dimensão do problema.

O que se perceberá ao longo do texto, é que o conceito de esparsidade está pois intimamente ligado ao de *tratabilidade computacional* para o problema de solução de sistemas lineares, com uma medida proporcional desta grandeza, advindo diretamente do grau de eficiência alcançada na exploração dos aspectos estruturais das matrizes envolvidas.

2.3 Propriedades Elementares

A base para a construção dos métodos diretos vem da exploração de algumas propriedades elementares dos sistemas de equações lineares, de amplo conhecimento na literatura ([55], [37], [41], [22], [48]).

Enumeram-se a seguir, sem qualquer comprovação formal, algumas destas propriedades, assumindo-se uma matriz de coeficientes simétrica e definida positiva.

Propriedade 2.3.1 *A solução de um sistema de equações lineares não se altera quando se multiplica todos os coeficientes de uma dada linha por uma constante real não nula.*

Propriedade 2.3.2 *A solução de um sistema de equações lineares não se altera quando se adicionam ou subtraem duas equações (coeficiente a coeficiente), substituindo-se uma delas pela nova equação assim obtida.*

Propriedade 2.3.3 *A solução de um sistema de equações lineares não se altera quando se permutam duas ou mais linhas de um mesmo sistema entre si.*

Propriedade 2.3.4 *A solução de um sistema de equações lineares altera-se apenas a nível de uma permutação de índices das variáveis da solução, quando se permutam duas ou mais colunas associadas ao mesmo sistema entre si.*

Propriedade 2.3.5 *Um sistema de equações lineares cuja matriz de coeficientes é real, simétrica e definida positiva, admite uma única solução real.*

Propriedade 2.3.6 *Uma matriz simétrica, definida positiva admite uma decomposição única na forma $U^T D\, U$, com matrizes $U \in \mathcal{R}^{n\,x\,n}$ da forma triangular superior com diagonal unitária e $D \in \mathcal{R}^{n\,x\,n}$ da forma diagonal.*

As propriedades 2.3.1, 2.3.2, 2.3.3 e 2.3.4 são exploradas por todos os métodos de eliminação direta, mediante a combinação linear e permutação de equações visando levar a matriz A do sistema original, à forma triangular superior com diagonal unitária U.

A propriedade 2.3.5 garante a existência e unicidade da solução no corpo real, nos casos em que a matriz do sistema é simétrica e definida positiva.

A propriedade 2.3.6 garante a existência e unicidade da fatoração da matriz A na forma $U^T D\, U$, viabilizando a aplicação de métodos diretos para a solução do sistema (2.1).

Apenas estas propriedades, não são suficientes para a caracterização completa de um método de eliminação, uma vez que não fixam uma ordem para a aplicação das operações elementares sobre os coeficientes do sistema, durante o processo de sua triangularização.

Como será visto na próxima seção, distintas estratégias de cancelamento, oferecem um variado leque de opções para a composição dos métodos diretos.

2.4 Métodos de Eliminação Direta

Serão apresentadas nesta seção algumas das alternativas clássicas para a solução direta de sistemas lineares, não se concentrando fundamentalmente nos aspectos de natureza esparsa, a serem abordados na seção 2.5 e mais detalhadamente ao longo do capítulo 3.

O material desta seção é complementado no apêndice B, onde se apresenta passo a passo o desenvolvimento de todas as etapas da eliminação direta, para um exemplo concreto de pequena dimensão, permitindo a constatação de alguns dos pontos salientados sem maior comprovação no presente trabalho (reportando-se a [55] para maiores detalhes).

Os métodos diretos de eliminação tiveram sua utilização sistematizada após os trabalhos de Gauss, Cholesky, Crout e Doolittle entre outros, e que viabilizaram a posterior aplicação destas metodologias nas mais diversas áreas da engenharia, como se pode notar na seção 1.2.

Especificamente para o tratamento de sistemas simétricos definidos positivos, duas metodologias de eliminação matricial mostram-se particularmente aplicáveis:

- *Eliminação de Gauss Simétrica*
- *Eliminação de Cholesky*

Ambas se baseiam na exploração das propriedades apresentadas na seção 2.3, procurando levar o sistema original a uma forma triangular, quando a partir deste ponto, a aplicação repetida de processos de substituições de variáveis permite trivialmente alcançar-se a sua solução.

A etapa inicial de combinações lineares entre linhas, visando levar o sistema à forma triangular, pode ser expressa na forma de produtos de matrizes *elementares* (a serem apresentadas mais adiante), e que ao final do processo, se multiplicadas termo a termo, permitem que se expresse compactamente o processo de eliminação na forma de uma decomposição de matrizes de fatores triangulares.

No caso de matrizes quadradas *genéricas* com *rank* máximo, duas possíveis fatorações conhecidas como *Eliminações de Gauss* se expressam por:

$$A = L\,U, \tag{2.6}$$

ou

$$A = L\,(D,U) \tag{2.7}$$

com L e $U, = D,U$ respectivamente da forma triangular inferior com diagonal unitária e triangular superior com diagonal não nula, e $D,$ e U respectivamente da forma diagonal com elementos não nulos e triangular superior com diagonal unitária.

No caso simétrico, definido positivo, a fatoração de *Gauss Simétrica*, análoga a eliminação (2.7), pode ser expressa equivalentemente por:

$$A = L\,D\,L^T \tag{2.8}$$

ou

$$A = U^T D\,U \tag{2.9}$$

com $L = U^T$ e U da forma triangular correspondente com diagonal unitária e D da forma diagonal, com valores estritamente positivos (diferindo unicamente na opção por L ou U como base para a expressão dos fatores triangulares resultantes).

Outra possibilidade de fatoração, análoga a (2.6) no caso simétrico, definido positivo, é a *Eliminação de Cholesky*, expressa equivalentemente por:

$$A = (L\,D^{1/2})\,(D^{1/2}L^T) = L_*\,L_*^T \tag{2.10}$$

ou

$$A = (U^T D^{1/2})\,(D^{1/2}U) = U_*^T\,U_* \tag{2.11}$$

com $L_* = L\,D^{1/2} = U_*^T$ e $U_* = D^{1/2}U$ da forma triangular correspondente, com diagonal estritamente positiva (oferecendo-se novamente a liberdade de opção por L_* ou U_* como base para a representação dos fatores).

Cancelamentos Visando a Transformação à Forma Triangular

Para cada uma das alternativas de decomposição matricial, se pode associar um processo de eliminação direta de variáveis, fundamentado basicamente na aplicação sucessiva das propriedades 2.3.1 e 2.3.2, visando conduzir a matriz do sistema linear original a uma forma triangular.

Por simplicidade, se apresentarão inicialmente processos *genéricos* de eliminação, assumindo-se matrizes de *rank* completo (sem levar-se em conta particularmente o caso simétrico, definido positivo, abordado complementarmente no apêndice B).

Todos os processos considerados neste trabalho, consistem na aplicação de n etapas denominadas *básicas*, onde se efetuam cancelamentos dos elementos de uma dada linha ou coluna situados na porção triangular inferior da matriz sendo transformada, gerando-se em contrapartida um grupo de fatores numa porção triangular superior associada.

A opção entre cancelar-se os elementos por linhas ou colunas, oferece várias possibilidades de se implementar o processo de eliminação, uma vez que a seqüência de operações, como salientado na seção anterior, não se encontra rigidamente fixada "a priori".

Duas alternativas clássicas são a eliminação dos elementos inferiores por *Colunas* e a eliminação por *Linhas*, cujas formas de atualização se encontram exemplificadas antes e após a conclusão de uma etapa básica, respectivamente nas Figuras 2.1 e 2.2 (com $a_{k,j}^{(i)}$ e $b_k^{(i)}$ denotando os valores atualizados após a i'ézima etapa).

Cancelamentos por Colunas

Durante o processo de transformação do sistema original à forma triangular, associam-se matrizes $A^{(i)}$ a cada uma de suas etapas básicas, assumindo-se inicialmente $A^{(0)} = A$.

A cada nova etapa básica i, transforma-se a matriz resultante $A^{(i)}$ a partir de $A^{(i-i)}$ (obtida na etapa anterior), anulando-se seus elementos subdiagonais situados na i'ézima coluna (mediante combinações escalares da linha base i, com cada uma das linhas subseqüentes).

O processo é levado adiante até se alcançar a n'ézima etapa, onde é encerrado, por não haverem mais elementos a se cancelar (tendo-se concluído a transformação da matriz original à forma triangular superior).

Apresenta-se formalmente a seguir, a seqüência de operações efetuadas em cada uma das etapas básicas deste processo.

Após a Etapa $i-1$

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{i,i}^{(i-1)} & a_{i,i+1}^{(i-1)} & \cdots & a_{i,n}^{(i-1)} \\ 0 & \cdots & a_{i+1,i}^{(i-1)} & a_{i+1,i+1}^{(i-1)} & \cdots & a_{i+1,n}^{(i-1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{n,i}^{(i-1)} & a_{n,i+1}^{(i-1)} & \cdots & a_{n,n}^{(i-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_i^{(i-1)} \\ b_{i+1}^{(i-1)} \\ \vdots \\ b_n^{(i-1)} \end{pmatrix}$$

Após a Etapa i

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{a_{i,i}^{(i)}} & \mathbf{a_{i,i+1}^{(i)}} & \cdots & \mathbf{a_{i,n}^{(i)}} \\ 0 & \cdots & \mathbf{0} & a_{i+1,i+1}^{(i)} & \cdots & a_{i+1,n}^{(i)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{0} & a_{n,i+1}^{(i)} & \cdots & a_{n,n}^{(i)} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ \mathbf{b_i^{(i)}} \\ b_{i+1}^{(i)} \\ \vdots \\ b_n^{(i)} \end{pmatrix}$$

Fig 2.1: Cancelamentos por Colunas

Cancelamento por Colunas com Atualização por Sub-Matrizes

Etapa i **:**

para k **de** $i+1$ **até** n **faça**

para j **de** $i+1$ **até** n **faça**

$$a_{k,j}^{(i)} \leftarrow a_{k,j}^{(i-1)} - (\, a_{k,i}^{(i-1)} \, / \, a_{i,i}^{(i-1)} \,) * a_{i,j}^{(i-1)}$$

$$b_k^{(i)} \leftarrow b_k^{(i)} - (\, a_{k,i}^{(i-1)} \, / \, a_{i,i}^{(i-1)} \,) * b_i^{(i)}$$

Um fato a se ressaltar é que uma vez obtidos os fatores de uma dada linha base a cada etapa, estes não mais sofrerão alteração ao longo de todo o restante do processo.

Por esta razão, não se mostra necessário armazenar separadamente cada uma das matrizes $A^{(i)}$, bastando se reescrever os novos valores obtidos a cada etapa, por sobre as posições correspondentes da matriz associada a etapa anterior.

Cancelamentos por Linhas

Análogamente ao processo descrito anteriormente, assume-se $A^{(0)} = A$.

A cada nova etapa básica i, transforma-se a matriz resultante $A^{(i)}$ a partir de $A^{(i-i)}$, anulando-se os elementos à esquerda da diagonal situados na i'ézima linha (mediante combinações escalares de linhas prévias, com a linha base sendo gerada).

O processo é levado adiante até se alcançar a n'ézima etapa, e se concluir o cancelamento de todos os elementos triangulares inferiores associados.

A seqüência de operações efetuadas em cada uma das etapas básicas é apresentada a seguir.

Cancelamento por Linhas com Atualização por Linhas

Etapa i **:**

para k **de** 1 **até** $i-1$ **faça**

para j **de** 1 **até** n **faça**

$$a_{i,j}^{(i)} \leftarrow a_{i,j}^{(i)} - (\, a_{i,k}^{(i)} \, / \, a_{k,k}^{(k)} \,) * a_{k,j}^{(k)}$$

$$b_i^{(i)} \leftarrow b_i^{(i)} - (\, a_{i,k}^{(i)} \, / \, a_{k,k}^{(k)} \,) * b_k^{(k)}$$

Operações Redundantes em Face a Presença de Simetria

Considerando-se a forma de atualização de elementos a cada etapa básica i no processo de eliminação por colunas, nos casos em que a matriz A do sistema original é simétrica, pode-se mostrar que todas as submatrizes $a_{k,j}^{(i)}$ para $k, j > i$, $i = 1, ..., n-1$

Após a Etapa $i-1$

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i-1}^{(1)} & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{i-1,i-1}^{(i-1)} & a_{i-1,i}^{(i-1)} & a_{i-1,i+1}^{(i-1)} & \cdots & a_{i-1,n}^{(i-1)} \\ a_{i,1} & \cdots & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \cdots & a_{i,n} \\ a_{i+1,1} & \cdots & a_{i+1,i-1} & a_{i+1,i} & a_{i+1,i+1} & \cdots & a_{i+1,n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & a_{n,i} & a_{n,i+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_{i-1}^{(i-1)} \\ b_i \\ b_{i+1} \\ \vdots \\ b_n \end{pmatrix}$$

Após a Etapa i

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i-1}^{(1)} & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{i-1,i-1}^{(i-1)} & a_{i-1,i}^{(i-1)} & a_{i-1,i+1}^{(i-1)} & \cdots & a_{i-1,n}^{(i-1)} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{a_{i,i}^{(i)}} & \mathbf{a_{i,i+1}^{(i)}} & \cdots & \mathbf{a_{i,n}^{(i)}} \\ a_{i+1,1} & \cdots & a_{i+1,i-1} & a_{i+1,i} & a_{i+1,i+1} & \cdots & a_{i+1,n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & a_{n,i} & a_{n,i+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_{i-1}^{(i-1)} \\ \mathbf{b_i^{(i)}} \\ b_{i+1} \\ \vdots \\ b_n \end{pmatrix}$$

Fig 2.2: Cancelamentos por Linhas

são simétricas.

Para tal, suponha que para algum $i = 2, ..., n-1$ a submatriz $a_{k,j}^{(i)}$ para $k, j > i$ seja simétrica (visto que para $i = 1$ este fato já se verifica por construção).

Considere-se para $k, j > i$ as atualizações da forma:

$$a_{k,j}^{(i)} \leftarrow a_{k,j}^{(i-1)} - (\, a_{k,i}^{(i-1)} \, / \, a_{i,i}^{(i-1)} \,) * a_{i,j}^{(i-1)}$$

onde portanto

$$a_{j,k}^{(i)} \leftarrow a_{j,k}^{(i-1)} - (\, a_{j,i}^{(i-1)} \, / \, a_{i,i}^{(i-1)} \,) * a_{i,k}^{(i-1)}$$

o que lançando-se mão da hipótese, permite facilmente se notar por recursão que $a_{k,j}^{(i)} = a_{j,k}^{(i)}$.

Esta propriedade pode ser demonstrada análogamente no caso de eliminações por linhas, ou das demais formas de eliminação para o caso simétrico (como as apresentadas no apêndice B), e em sua decorrência, permite a economia de recursos computacionais tanto do ponto de vista do espaço de armazenamento efetivamente necessário, como do ponto de vista do esforço de cálculo, visto tornar redundante e desnecessário o cálculo explícito dos valores temporários de elementos simétricos situados na porção triangular inferior dos sistemas transformados a cada etapa.

Representação na Forma de Produtos de Matrizes Elementares

As etapas básicas do processo de eliminação podem ser representadas compactamente na forma de produtos de matrizes *elementares*.

Tais matrizes assumem uma forma triangular, contendo diagonal unitária, e apenas elementos (multiplicadores) não nulos nas posições além da diagonal em uma dada linha ou coluna básica.

Um exemplo de uma matriz *elementar* inferior associada a k'ézima etapa é apresentado em (2.12) .

$$L^{(k)} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\frac{a_{k+1,k}^{(k)}}{a_{k,k}} & 1 & \\ & & \vdots & & \ddots \\ & & -\frac{a_{n,k}^{(k)}}{a_{k,k}} & & 1 \end{pmatrix} \tag{2.12}$$

Mediante a representação por matrizes *elementares*, o processo de combinações lineares visando anular-se os elementos da porção inferior a k'ézima coluna pode ser expresso compactamente na forma:

$$A^{(k+1)} = L^{(k)} A^{(k)} \tag{2.13}$$

A partir desta relação, estendendo-se o processo até a conclusão de todas as etapas da eliminação direta, teremos:

$$U_{\prime} = A^{(n)} = L^{(n-1)} \ldots L^{(1)} A \tag{2.14}$$

De (2.12), não fica difícil de se comprovar (por multiplicação) que:

$$(L^{(k)})^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \frac{a_{k+1,k}^{(k)}}{a_{k,k}} & 1 & \\ & & \vdots & & \ddots \\ & & \frac{a_{n,k}^{(k)}}{a_{k,k}} & & 1 \end{pmatrix} \tag{2.15}$$

A partir de (2.14), multiplicando-se esta relação pela seqüência de matrizes *elementares* inversas, pode se expressar:

$$A = (L^{(1)})^{-1} \ldots (L^{(n-1)})^{-1} U_{\prime} \tag{2.16}$$

e que a partir de (2.15) fornece:

$$L = (L^{(1)})^{-1} \ldots (L^{(n-1)})^{-1} = \begin{pmatrix} 1 & & & & & \\ \frac{a^{(1)}_{2,1}}{a^{(1)}_{1,1}} & \ddots & & & & \\ \vdots & \ddots & 1 & & & \\ \frac{a^{(1)}_{k,1}}{a^{(1)}_{1,1}} & \cdots & \frac{a^{(k)}_{k+1,k}}{a^{(k)}_{k,k}} & 1 & & \\ \vdots & \ddots & \vdots & \ddots & \ddots & \\ \frac{a^{(1)}_{n,1}}{a^{(1)}_{1,1}} & \cdots & \frac{a^{(k)}_{n,k}}{a^{(k)}_{k,k}} & \cdots & \frac{a^{(n-1)}_{n,n-1}}{a^{(n-1)}_{n-1,n-1}} & 1 \end{pmatrix} \tag{2.17}$$

mostrando que A pode ser expressa na forma:

$$A = L\,U, \tag{2.18}$$

Eliminações de Gauss Simétricas

A partir deste ponto, formalizam-se alguns procedimentos de eliminação, apresentando-se codificações para o caso denso, simétrico, definido positivo, das alternativas conhecidas como fatorações de *Gauss Simétricas.*

Em todos os casos, assume-se que apenas a porção diagonal e triangular superior da matriz original A esteja explicitamente disponível e armazenada.

Ao longo de cada etapa i do processo de eliminações, os fatores triangulares $u_{i,j}$ associados e os elementos diagonais $d_{i,i}$, vão sendo gerados (podendo nas implementações computacionais, serem escritos de volta em memória, por sobre os valores originais $a_{i,j}$ correspondentes).

Na prática o que se efetivamente armazena ao longo da seqüência de eliminações, é uma representação *compacta* da porção diagonal (de D), e triangular superior excluindo-se a diagonal unitária (de U), bem como o vetor lado direito atualizado, como se apresenta nas Figuras 2.3 e 2.4 (comparando-as com os sistemas transformados a cada etapa).

Inicialmente apresenta-se na alternativa **2.4(a)** o algoritmo de fatoração $U^T D\,U$ (*Gauss*) com atualização dos fatores por *Sub-Matrizes.*

Em todos os procedimentos considerados, a normalização dos elementos triangulares superiores $u_{k,i}$ (dividindo-se os elementos $a_{k,i}$ correspondentes pelo elemento diagonal associado) é efetuada paulatinamente, explorando-se propriedades decorrentes da simetria do sistema (conforme notado anteriormente e no apêndice B).

Esta ordem de normalização, não obedece necessariamente a mesma em que os elementos $a_{k,i}$ originais vão sendo atualizados ao longo do processo de eliminações. Isso é feito de modo a se aproveitar resultados parciais sendo gerados, efetuando-se oportunamente a normalização necessária a cada etapa (garantindo-se que ao final do processo de eliminações, todos os elementos supradiagonais de U terão sido normalizados).

Sistema Transformado Após a Etapa i

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{a_{i,i}^{(i)}} & \mathbf{a_{i,i+1}^{(i)}} & \cdots & \mathbf{a_{i,n}^{(i)}} \\ 0 & \cdots & \mathbf{0} & a_{i+1,i+1}^{(i)} & \cdots & a_{i+1,n}^{(i)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{0} & a_{n,i+1}^{(i)} & \cdots & a_{n,n}^{(i)} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ \mathbf{b_i^{(i)}} \\ b_{i+1}^{(i)} \\ \vdots \\ b_n^{(i)} \end{pmatrix}$$

Representação Compacta Após a Etapa i

$$\begin{pmatrix} d_{1,1} & \cdots & u_{1,i} & u_{1,i+1} & \cdots & u_{1,n} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \mathbf{d_{i,i}} & \mathbf{u_{i,i+1}} & \cdots & \mathbf{u_{i,n}} & \mathbf{b_i^{(i)}} \\ 0 & \cdots & \mathbf{0} & a_{i+1,i+1}^{(i)} & \cdots & a_{i+1,n}^{(i)} & b_{i+1}^{(i)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \mathbf{0} & a_{n,i+1}^{(i)} & \cdots & a_{n,n}^{(i)} & b_n^{(i)} \end{pmatrix}$$

Fig 2.3: Representação Compacta para Cancelamentos por Colunas

Alternativa 2.4(a) *Fatoração $U^T D U$ (Gauss) por Sub-Matrizes*

para i **de** 1 **até** n **faça**
  $d_{i,i} \leftarrow a_{i,i}$
  para k **de** $i+1$ **até** n **faça**
    $u_{i,k} \leftarrow a_{i,k} \ / \ d_{i,i}$
    para j **de** k **até** n **faça**
      $a_{k,j} \leftarrow a_{k,j} - u_{i,k} * a_{i,j}$

Alt 2.4(a): *Fatoração $U^T D U$ (Gauss) por Sub-Matrizes*

Apresenta-se na alternativa **2.4(b)** o algoritmo de fatoração $U^T D U$ (*Gauss*) com geração dos fatores por *Linhas*.

Na alternativa **2.4(c)** se apresenta uma última variante do processo de eliminação para o caso simétrico, com a geração dos fatores triangulares superiores por *Colunas*, mediante o cancelamento por Linhas, à esquerda da diagonal de cada etapa básica.

Esta alternativa não foi considerada previamente nesta seção em particular, referindo-se ao apêndice B para uma exposição mais detalhada.

Em face a simetria do problema original, a geração por *Colunas* pode ser concreti-

Sistema Transformado Após a Etapa i

$$\begin{pmatrix} a_{1,1}^{(1)} & \cdots & a_{1,i-1}^{(1)} & a_{1,i}^{(1)} & a_{1,i+1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{i-1,i-1}^{(i-1)} & a_{i-1,i}^{(i-1)} & a_{i-1,i+1}^{(i-1)} & \cdots & a_{i-1,n}^{(i-1)} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{a_{i,i}^{(i)}} & \mathbf{a_{i,i+1}^{(i)}} & \cdots & \mathbf{a_{i,n}^{(i)}} \\ a_{i+1,1} & \cdots & a_{i+1,i-1} & a_{i+1,i} & a_{i+1,i+1} & \cdots & a_{i+1,n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & a_{n,i} & a_{n,i+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_{i-1}^{(i-1)} \\ \mathbf{b_i^{(i)}} \\ b_{i+1} \\ \vdots \\ b_n \end{pmatrix}$$

Representação Compacta Após a Etapa i

$$\begin{pmatrix} d_{1,1} & \cdots & u_{1,i-1} & u_{1,i} & u_{1,i+1} & \cdots & u_{1,n} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & d_{i-1,i-1} & u_{i-1,i} & u_{i-1,i+1} & \cdots & u_{i-1,n} & b_{i-1}^{(i-1)} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{d_{i,i}} & \mathbf{u_{i,i+1}} & \cdots & \mathbf{u_{i,n}} & \mathbf{b_i^{(i)}} \\ a_{i+1,1} & \cdots & a_{i+1,i-1} & a_{i+1,i} & a_{i+1,i+1} & \cdots & a_{i+1,n} & b_{i+1} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & a_{n,i} & a_{n,i+1} & \cdots & a_{n,n} & b_n \end{pmatrix}$$

Fig 2.4: Representação Compacta para Cancelamentos por Linhas

Alternativa 2.4(b) *Fatoração $U^T D U$ (Gauss) por Linhas*

para i **de** 1 **até** n **faça**
  para k **de** 1 **até** $i-1$ **faça**
    $u_{k,i} \leftarrow a_{k,i} \ / \ d_{k,k}$
    para j **de** i **até** n **faça**
      $a_{i,j} \leftarrow a_{i,j} - u_{k,i} * a_{k,j}$
  $d_{i,i} \leftarrow a_{i,i}$

Alt 2.4(b): *Fatoração $U^T D U$ (Gauss) por Linhas*

zada, mediante uma alteração da ordem natural de geração dos fatores triangulares.

Ao invés de se efetuar o cálculo para os elementos da porção triangular *superior* da *Linha* básica, como nos processos anteriores, o que se efetua é justamente o cálculo dos fatores situados na porção *inferior*, e que em função de existência de fatores simétricos de mesmo valor na porção superior, podem ser lançados nas posições correspondentes da *Coluna* básica a medida que vão sendo gerados a cada etapa.

O que se pode perceber é que neste processo, o cálculo dos fatores triangulares

superiores é na verdade "retardado" até que se efetue efetivamente o cálculo dos valores inferiores correspondentes.

Por esta razão, este processo recebeu em [83] a designação de método de "*Atualizações Retardadas*", sendo apresentado na alternativa **2.4(c)** a codificação do algoritmo de fatoração $U^T D\,U$ (*Gauss*) com geração dos fatores por *Colunas*.

Alternativa 2.4(c) *Fatoração* $U^T D\,U$ (*Gauss*) *por Colunas*

para i **de** 1 **até** n **faça**
  para k **de** 1 **até** $i-1$ **faça**
    $u_{k,i} \leftarrow a_{k,i} \;/\; d_{k,k}$
    para j **de** $k+1$ **até** i **faça**
      $a_{j,i} \leftarrow a_{j,i} - u_{k,i} * a_{k,j}$
  $d_{i,i} \leftarrow a_{i,i}$

Alt 2.4(c): *Fatoração* $U^T D\,U$ (*Gauss*) *por Colunas*

Substituições Triangulares e Diagonais

Uma vez obtida uma decomposição triangular para a matriz do sistema linear a se solucionar, passa-se finalmente as fases (2.3), (2.4) e (2.5) correspondentes ao processo de substituições de variáveis, como notado na seção 2.1.

A obtenção da solução é obtida, partindo-se de

$$U^T\,(D\,U\,x) \;=\; b$$

o que permite se expressar a fase *Forward* em termos de um vetor incógnita x'' mediante

$$U^T\,x'' \;=\; b \tag{2.19}$$

Das duas relações anteriores obtem-se

$$D\,(U\,x) \;=\; x''$$

o que permite que se expresse a fase *Diagonal* em termos de um vetor incógnita x' como

$$D\,x' \;=\; x'' \tag{2.20}$$

Finalmente, das duas últimas relações, pode-se expressar o vetor solução x a partir de

$$U\,x \;=\; x' \tag{2.21}$$

Inicialmente apresenta-se no procedimento **2.4(1)** o processo de substituições *Forward.*

Procedimento 2.4(1) *Substituição Forward*

$$U^T\ x'' \ = \ b$$

para i **de** 1 **até** n **faça**
$x''_i \leftarrow b_i$

para i **de** 1 **até** $n-1$ **faça**
para j **de** $i+1$ **até** n **faça**
$x''_j \leftarrow x''_j - u_{i,j} * x''_i$

Proc 2.4(1): *Substituição Forward*

Apresenta-se no procedimento **2.4(2)** o processo de normalização *Diagonal.*

Procedimento 2.4(2) *Normalização Diagonal*

$$D\ x' \ = \ x''$$

para i **de** 1 **até** n **faça**
$x'_i \leftarrow x''_i \ / \ d_{i,i}$

Proc 2.4(2): *Normalização Diagonal*

Finalmente é apresentado no procedimento **2.4(3)** o processo de retro-substituições.

A fase de substituição *Forward* pode ser combinada com a de normalização *Diagonal* em um único procedimento, tornando mais eficiente este processo. No procedimento **2.4(4)** se apresenta esta alternativa, incorporando-se também a fase de substituições *Backward*, e unificando-se no vetor x, toda a seqüência intermediária de operações das fases anteriores.

Eliminações de Cholesky

Na prática, os procedimentos diretos empregados na resolução de sistemas simétricos, definidos positivos, tomam por base alguma das eliminações de *Gauss Simétricas* apresentadas, ou suas variantes, encontradas na literatura.

As eliminações de *Cholesky* podem ser teoricamente empregadas para a solução do mesmo problema, porém algumas desvantagens e inconvenientes sob o ponto de

Procedimento 2.4(3) *Substituição Backward*

$$U\ x\ =\ x'$$

para i **de** 1 **até** n **faça**
$x_i \leftarrow x'_i$

para i **de** $n-1$ **de volta até** 1 **faça**
para j **de** n **de volta até** $i+1$ **faça**
$x_i \leftarrow x_i - u_{i,j} * x_j$

Proc 2.4(3): *Substituição Backward*

Procedimento 2.4(4) *Processo Unificado de Substituições*

para i **de** 1 **até** n **faça**
$x_i \leftarrow b_i$

para i **de** 1 **até** n **faça**
para j **de** $i+1$ **até** n **faça**
$x_j \leftarrow x_j - u_{i,j} * x_i$
$x_i \leftarrow x_i \ / \ d_{i,i}$

para i **de** $n-1$ **de volta até** 1 **faça**
para j **de** n **de volta até** $i+1$ **faça**
$x_i \leftarrow x_i - u_{i,j} * x_j$

Proc 2.4(4): *Processo Unificado de Substituições*

vista numérico a serem comentadas um pouco mais adiante, tornam a opção por tais métodos na maior parte dos casos menos segura do que a aplicação dos métodos de eliminação *Gaussiana Simétrica*, ou de métodos mais robustos, especificamente projetados para o tratamento de sistemas *numericamente indefinidos* como a abordagem *Multifrontal* adotada por Duff em [30].

A analogia entre os processos de fatoração $U_*^T\,U_*$ (*Cholesky*) com seus equivalentes processos $U^T D\,U$ (*Gauss*) faz-se notar nos procedimentos apresentados a seguir, iniciando-se com a apresentação na alternativa **2.4(d)** do algoritmo de fatoração de *Cholesky* por *Sub-Matrizes.*

O único fato a se ressaltar é que no caso das eliminações de *Cholesky*, a normalização é feita em separado, para todos as componentes da linha ou coluna de fatores sendo gerada (o que não ocorre nas eliminações de *Gauss Simétricas*, onde

o processo de normalização é efetuado de forma paulatina e concomitante com as operações do *loop* intermediário).

Alternativa 2.4(d) *Fatoração $U_*^T U_*$ (Cholesky) por Sub-Matrizes*

para i **de** 1 **até** n **faça**
  $u^*_{i,i} \leftarrow \sqrt{a_{i,i}}$
  para k **de** $i+1$ **até** n **faça**
    $u^*_{i,k} \leftarrow a_{i,k} \;/\; u^*_{i,i}$
  para k **de** $i+1$ **até** n **faça**
    para j **de** k **até** n **faça**
      $a_{k,j} \leftarrow a_{k,j} - u^*_{i,k} * u^*_{i,j}$

Alt 2.4(d): *Fatoração $U_*^T U_*$ (Cholesky) por Sub-Matrizes*

Apresenta-se na alternativa **2.4(e)** o algoritmo de fatoração $U_*^T U_*$ (*Cholesky*) com geração dos fatores por *Linhas.*

Alternativa 2.4(e) *Fatoração $U_*^T U_*$ (Cholesky) por Linhas*

para i **de** 1 **até** n **faça**
  para k **de** 1 **até** $i-1$ **faça**
    para j **de** i **até** n **faça**
      $a_{i,j} \leftarrow a_{i,j} - u^*_{k,i} * u^*_{k,j}$
  $u^*_{i,i} \leftarrow \sqrt{a_{i,i}}$
  para j **de** $i+1$ **até** n **faça**
    $u^*_{i,j} \leftarrow a_{i,j} \;/\; u^*_{i,i}$

Alt 2.4(e): *Fatoração $U_*^T U_*$ (Cholesky) por Linhas*

Finalmente, é apresentado na alternativa **2.4(f)** o algoritmo de fatoração $U_*^T U_*$ (*Cholesky*) com geração dos fatores por *Colunas.*

Os métodos de eliminação de *Cholesky*, embora teoricamente *estáveis* sob o ponto de vista numérico (dispensando o uso de técnicas de pivoteamento a serem vistas mais adiante), apresentam como principal inconveniente, a necessidade de se efetuar operações de raiz quadrada sobre os elementos diagonais a cada etapa.

Tal problema reside praticamente ao nível das implementações computacionais, por não se poder garantir que por erros intrínsecos ao processamento numérico em ponto flutuante (como os de arredondamento), tais elementos diagonais venham a de fato estar computacionalmente representados com valores positivos (obrigando-se a artifícios numéricos de modo a se contornar situações dessa natureza).

Alternativa 2.4(f) *Fatoração $U_*^T U_*$ (Cholesky) por Colunas*

para i **de** 1 **até** n **faça**
  para k **de** 1 **até** $i-1$ **faça**
    $u^*_{k,i} \leftarrow a_{k,i} \ / \ u^*_{k,k}$
  para k **de** 1 **até** $i-1$ **faça**
    para j **de** $k+1$ **até** i **faça**
      $a_{j,i} \leftarrow a_{j,i} - u^*_{k,i} * u^*_{k,j}$
  $u^*_{i,i} \leftarrow \sqrt{a_{i,i}}$

Alt 2.4(f): *Fatoração $U_*^T U_*$ (Cholesky) por Colunas*

Este entrave não se encontra presente nos métodos de eliminação de *Gauss Simétricos*, pois neste caso, o que os erros numéricos podem normalmente introduzir, são elementos diagonais negativos, e que embora não garantam mais que as sub-matrizes subseqüentes no processo de cálculo sejam numericamente definidas positivas, ao menos permite se chegar a uma solução na maioria das vezes "próxima" a do sistema original.

Pivoteamentos em Valor Numérico no Caso Geral

No caso da eliminação de *Gauss* para o caso de matrizes genéricas (com *rank* máximo), pode ser que elementos diagonais de valor nulo sejam encontrados, durante as etapas do processo, o que inviabilizaria a aplicação das metodologias até então apresentadas.

Pelo fato de se estar assumindo matrizes com *rank* completo, a existência de uma fatoração (bem como a existência de inversa) é garantida, como se pode notar em [9] (página 397), [87] (página 42) entre outras referências clássicas da literatura como [55] e [41].

A solução para se conseguir levar adiante o processo de eliminações neste caso, é lançar mão das propriedades 2.3.3 e 2.3.4 ainda não exploradas até então, e que garantem que permutações entre linhas ou colunas do sistema podem ser efetuadas, alterando-se sua solução no máximo ao nível de uma permutação dos índices de suas variáveis.

Desta forma, o que os procedimentos de eliminação passam a solucionar são problemas permutados de modo a se excluir elementos diagonais nulos.

Este processo de intercâmbio de linhas ou colunas a cada etapa, recebeu a designação de "*pivoteamento* em valor numérico", e os métodos de eliminação *Gaussiana* para o caso de matrizes genéricas, na prática são todos implementados com alguma forma de pivoteamento como salvaguarda.

A cada etapa, associa-se um elemento denominado "pivô", a ser escolhido entre os coeficientes das linhas e colunas a serem processadas, e que determinará uma

permutação a ser efetuada com o elemento diagonal corrente.

Diferentes técnicas de pivoteamento numérico acabaram sendo consagradas, como as de pivoteamento *Parcial* ou *Total*. No primeiro caso procura-se permutar a cada etapa k do processo, a linha corrente k com a linha i abaixo desta contendo o coeficiente $a_{i,k}$ de maior valor absoluto. No caso do pivoteamento Total, procura-se permutar tanto a linha como a coluna k pela linha i e coluna j do elemento $a_{i,j}$ de maior valor absoluto em toda a sub-matriz definida a partir de $a_{k,k}$ até $a_{n,n}$.

A escolha específica pelo elemento de maior valor absoluto, deve-se a questões de *estabilidade numérica*, visando melhorar a precisão dos resultados, evitando-se um crescimento explosivo no valor numérico dos fatores multiplicativos da decomposição matricial. Estas questões não serão consideradas em maior detalhe neste trabalho, pois no caso simétrico, definido positivo, como se verá um pouco mais adiante, uma propriedade intrínseca a estes sistemas, garante a *estabilidade* numérica do processo, dispensando o uso de pivoteamentos em valor para esta finalidade.

No caso geral, o que as técnicas de pivoteamento em valor numérico introduzem é a solução de sistemas transformados na forma:

$$(P_l\, A\, P_u)\,(P_u^{-1} x) = (P_l\, b) \tag{2.22}$$

com as matrizes de permutação P_l e P_u correspondendo ao que se obteria caso se aplicasse a mesma seqüência de permutações de linhas e de colunas, sobre matrizes originalmente diagonais, com elementos unitários.

Condicionamento e Estabilidade Numérica de Sistemas Lineares

Apresenta-se nesta subseção, sem demonstrações, alguns resultados e propriedades estensivamente estudadas na álgebra linear clássica, reportando-se o leitor a referências como [55] (capítulo 5), [41] (capítulo 23), [87] (capítulos 2 e 3), [9] (capítulo 7) e [32] (capítulo 4).

A noção de *estabilidade* está associada a da precisão computacional com que se obtém os resultados durante a solução de um problema linear. É uma propriedade característica de cada método, e em função de alterações na ordem com que se efetuam tais operações, pode-se obter implementações mais ou menos *estáveis*. Todas as técnicas de pivoteamento em valor numérico, procuram deste modo minimizar o crescimento em valor absoluto dos fatores, mediante a escolha de candidatos a pivô devidamente selecionados de modo a manter a taxa de crescimento dos fatores baixa.

Já a noção de *condicionamento* encontra-se associada a natureza particular de cada problema, independentemente do método de solução a ser empregado. Nestes casos, problemas em que ao se efetuar pequenas perturbações em seus coeficientes acabem por determinar vetores solução completamente distantes da solução obtida para o problema original, são denominados *mal-condicionados*, pois independente da forma como se processe as operações de cancelamento, a própria natureza do problema acabará ditando um comportamento *instável*, em função dos erros de truncamento ou arredondamento cometidos ao longo do processo de sua solução.

Matemáticamente, caracteriza-se o *condicionamento* de cada problema através de um coeficiente (denominado *número de condicionamento*), e que pode ser determinado (ou estimado) em função de grandezas características de cada problema.

Suponha-se que a matriz de coeficientes A de um sistema linear é tal que exista vetores v e w satisfazendo as relações

$$||v||_2 = ||w||_2 \tag{2.23}$$

e

$$||Av||_2 \gg ||Aw||_2 \tag{2.24}$$

No caso do vetor lado direito b ser igual Av, nota-se que a solução para o sistema $Ax = b$ original é dada por $x = v$.

Caso em um novo problema se substitua o vetor b por Aw, nota-se que a solução deste novo sistema será alterada pela grandeza w, correspondendo a uma variação significativa entre a solução do problema original.

Nota-se pois neste caso, que o problema é mal condicionado, e uma forma de se expressar este fato é através da razão entre ambos os termos da equação 2.24, e que pode ser expressa na forma

$$\frac{||Av||_2}{||Aw||_2}\frac{||w||_2}{||v||_2} = \frac{||Av||_2}{||v||_2}\frac{||A^{-1}y||_2}{||y||_2} \tag{2.25}$$

supondo-se $y = Aw$.

O primeiro termo de 2.25 pode assumir no máximo o valor $||A||_2$ e o segundo termo no máximo $||A^{-1}||_2$ de modo que a razão $||Av||_2/||Aw||_2$ pode ser no máximo tão grande quanto $||A||\ ||A^{-1}||$.

Esta última grandeza é justamente a que se convencionou denominar pelo número de condicionamento associado a matriz A, $\kappa(A) = ||A||\ ||A^{-1}||$.

Outra noção importante para a caracterização da *dinâmica* do comportamento e da *estabilidade* do processo de eliminações, é a de *fator de crescimento*, expressa a partir da relação

$$\rho = \max_{i,j,k} |\ a_{i,j}^{(k)}| \tag{2.26}$$

No caso de matrizes simétricas definidas positivas, mostra-se que $\rho \leq \max_{i,j} |\ a_{i,j}|$, e no caso de matrizes diagonais dominantes $\rho \leq 2\ \max_{i,j} |\ a_{i,j}|$, o que em ambos os casos garante que o crescimento dos valores intermediários dos fatores sendo gerados, mantém-se *estável* e "sob controle" ao longo de todo o processo de eliminação.

Estabilidade Numérica no Caso Simétrico, Definido Positivo

No caso simétrico, definido positivo, pivoteamentos em valor numérico não são necessários, em face a algumas propriedades intrínsecas a matrizes simétricas definidas positivas, apresentadas a seguir.

Propriedade 2.4.1 *Uma matriz que possua todos os elementos de sua diagonal positivos, não nulos e "dominantes" (superiores a soma em valor absoluto dos demais elementos da mesma linha ou coluna associada) dispensa pivotementos em valor numérico ao longo do processo de eliminação visando sua fatoração, pelo fato de seu fator de crescimento ρ ser limitado a no máximo o dobro do maior valor absoluto dentre os coeficientes da matriz A do sistema original.*

Propriedade 2.4.2 *Uma matriz simétrica, definida positiva admite uma decomposição $U^T D U$ numericamente estável, dispensando desta forma pivoteamentos em valor numérico ao longo do processo de eliminação visando sua fatoração, pelo fato de seu fator de crescimento ser limitado a no máximo o maior valor absoluto dentre todos os coeficientes da matriz A do sistema original.*

A propriedade 2.4.2 é a que viabiliza as metodologias de solução consideradas, pois garante que a menos de permutações de caráter meramente estrutural introduzidas com objetivo de se manter a esparsidade da matriz de fatores U resultante (como será visto na próxima sub-seção), pivoteamentos em valor numérico visando manter a estabilidade numérica do processo de eliminação, não são necessários no caso particular das matrizes simétricas, definidas positivas.

Introdução de Novos Elementos Não Nulos no Caso Esparso

A aplicação repetida da propriedade 2.3.2 ao longo do processo de combinações lineares de equações visando a eliminação de variáveis, acaba conduzindo a um efeito secundário, com a introdução de fatores não nulos em algumas das posições originalmente correspondentes a coeficientes nulos na matriz A.

Este efeito, propagado a medida que se leva adiante o processo de combinações, é de importância central na área de resolução de sistemas esparsos, razão pela qual, apresenta-se a definição de um conceito que será detalhadamente abordado ao longo das seções subseqüentes.

Definição 2.4.3 (Fill-In) *Define-se por "elemento de fill-in" ou doravante ao longo do texto simplesmente por "fill-in", a um elemento $a_{i,j} = 0$ na matriz original, e que após o processo de combinação linear de vetores esparsos, explorando-se a propriedade 2.3.2 venha a adquirir um valor $u_{i,j} \neq 0$ na matriz de fatores associada.*

Permutações Estruturais Indispensáveis no Caso Esparso

A possibilidade de existirem elementos de *fill-in* na matriz de fatores U resultante, leva a um problema inexistente no caso denso, e que será abordado em detalhes na seção 3.2.

Tal problema consiste em se buscar uma minimização do número de *fill-in's*, mediante permutações adequadas nas linhas e colunas da matriz do sistema original.

Estas permutações, movidas por razões meramente "estruturais", visam retardar ao máximo, um crescimento explosivo no número de fatores não nulos, nas etapas iniciais do processo de eliminação.

Tais permutações mostram-se indispensáveis, uma vez que caso ocorra um aumento expressivo no número de fatores não nulos nas etapas iniciais, suas contribuições futuras por ocasião do cancelamento de outras linhas, tenderão a levar precocemente a matriz de fatores a uma forma quase densa, inviabilizando a solução por métodos diretos.

Visando preservar-se este caráter esparso, o que se estará efetivamente solucionando após a determinação de uma permutação ideal de linhas e colunas, serão sistemas da forma:

$$(P\,A\,P^T)\,(P^{T^{-1}}x) = (P\,b) \tag{2.27}$$

com matrizes de permutação P e P^T correspondendo a aplicação de pivoteamentos de natureza estrutural, com os candidatos a cada etapa restritos apenas aos elementos *diagonais*, por imposição da simetria original do problema.

2.5 Aspectos Computacionais

A solução de sistemas lineares de grande porte, mediante a aplicação das técnicas de esparsidade, está baseada na exploração eficiente de suas características de natureza estrutural.

Para alcançar este objetivo, as metodologias de solução devem se ater a alguns aspectos de vital importância, para a preservação da "tratabilidade" computacional do problema.

Os aspectos centrais a serem observados são:

- *espaço de armazenamento*
- *volume de operações aritméticas*

No primeiro caso, a meta principal é se evitar o armazenamento de informações triviais com relação aos coeficientes dos sistemas (original e triangular) envolvidos.

Para tal, a solução normalmente adotada é não se representar explicitamente coeficientes cujos valores se mantenham nulos, com o decorrer do processo de eliminações.

No segundo caso, deve-se levar em conta, apenas as operações de combinação linear realmente necessárias, uma vez que as operações envolvendo coeficientes de valor nulo, podem ser trivialmente evitadas, por manterem inalterados os valores originalmente envolvidos.

Qualquer projeto de métodos de solução esparsa, deve satisfazer portanto a estas restrições básicas, quanto ao espaço total de armazenamento e volume de operações efetuadas, de modo a se manter computacionalmente tratável, em função da disponibilidade de recursos e da dimensão de cada problema.

Caso uma abordagem convencional não explorando as características esparsas de cada problema fosse adotada, chegaria-se rapidamente a limites práticos, ditados pela escassez de recursos quer de espaço, quer de tempo.

Ilustra-se tal fato a seguir, supondo-se como limites hipoteticamente aceitáveis (para uma abordagem densa convencional), a utilização de até algumas centenas de *MegaBytes* de espaço de armazenamento, e o consumo da ordem de até algumas horas de *CPU* exclusivamente dedicadas ao processo de fatoração.

Em termos de armazenamento, metodologias que viessem a utilizar estruturas da ordem de n^2 posições de memória, mostrariam-se pois inadequadas para a solução de problemas de dimensão superior a casa de algumas dezenas de milhar de variáveis nas máquinas mais modernas atuais (com armazenamento principal na faixa de centenas de *MegaBytes*), sendo portanto impraticáveis para o tratamento de sistemas esparsos de médio ou grande porte.

De um modo geral, um espaço aceitável para estruturas de dados auxiliares adotadas no processo eliminação, deve situar-se entre valores proporcionais à dimensão do problema ou no máximo ao número de fatores não nulos da matriz triangular U resultante.

Com relação ao volume total de operações aritméticas, metodologias que venham a requerer da ordem de n^3 operações, mostram-se computacionalmente inadequadas para a resolução de problemas de dimensão acima de algumas centenas de milhar de variáveis, mesmo nas arquiteturas de maior desempenho atuais (capazes de sustentar taxas máximas de execução, em torno de algumas dezenas de *GigaFlops*).

Como regra prática, nota-se que um volume aceitável para uma solução computacional eficiente, deve se situar no máximo numa faixa proporcional a $\sum_{i=1}^{n} \eta^{(i)}(U^T)\,\eta^{(i)}(U)$ operações (correspondendo a um valor, ditado apenas pelo número de elementos não nulos de U e de linhas contribuintes a cada etapa, equivalente ao número de elementos não nulos de U^T, para cada linha base i).

Essa restrição, impõe ao projeto de novas metodologias esparsas, a utilização de sub-tarefas com um volume de operações limitado normalmente apenas a $O(\eta^{(\,\cdot\,)}(U))$ em cada uma das n etapas do processo de solução, descartando-se desta forma o uso de boa parte das ferramentas de amplo conhecimento em áreas correlatas da computação, com complexidade na maioria dos casos igual ou superior a $O(n)$.

Percebe-se pois, que o tratamento eficiente das características esparsas de cada problema, é uma tarefa que exige um balanceamento criterioso de recursos e estratégias, de modo a se viabilizar a solução de sistemas de porte elevado.

Ao longo do texto, especialmente nos capítulos 4 e 5 apresentar-se-ão alternativas consagradas na literatura e novas abordagens propostas neste trabalho, visando a redução de *overheads* computacionais, mantendo-se no entanto fiéis as *restrições básicas* salientadas nesta seção.

Capítulo 3

Abordagem Clássica

Neste capítulo serão apresentadas abordagens tradicionalmente adotadas para a resolução de sistemas lineares esparsos, por métodos diretos de eliminação.

O ferramental necessário, a ser introduzido nas seções subseqüentes, abrange: estruturas de dados elementares, reordenamentos visando a minimização do número de fatores não nulos, determinação simbólica da estrutura de fatores resultantes, e resolução de novos sistemas com mesma disposição estrutural de elementos.

3.1 Estruturas Elementares de Armazenamento

A exploração da natureza esparsa de sistemas lineares de equações, se dá mediante a adoção de estruturas de dados, visando evitar o armazenamento de informações de fácil determinação implícita, durante as sucessivas etapas do processo de solução.

A principal característica explorada nestas abordagens, é a do não armazenamento dos coeficientes cujo valor numérico permaneça nulo, ao longo de toda a seqüência de eliminações triangulares.

Grafo Associado à Estrutura de uma Matriz Esparsa Simétrica

Dada uma matriz simétrica e irredutível $A \in \mathcal{R}^{n \times n}$, uma forma de representação esparsa de sua *estrutura* pode ser naturalmente definida (como nos mostra a Figura 3.1) mediante sua associação a um grafo conexo e não direcionado G_A com nós $\{ 1, ..., n \}$ correspondendo a cada uma de suas linhas (ou colunas), e arestas (i, j) para todo $i, j = 1, ..., n$ com $i \neq j$ tais que $a_{i,j} \neq 0$.

Representações Adotadas para Cada Fase de Processamento

Em princípio, qualquer das estruturas de dados normalmente adotadas para a representação de grafos não direcionados, como listas de arestas ou de adjacências, pode ser aplicada para o armazenamento de matrizes simétricas na forma esparsa.

Algumas estruturas porém, mostram-se particularmente mais adequadas para determinadas fases do processo de solução de sistemas, como as de:

$$A = \begin{array}{c} \begin{matrix} (1) & (2) & (3) & (4) & (5) \end{matrix} \\ \begin{pmatrix} 2. & \mathbf{9.} & 0 & 0 & \mathbf{1.} \\ \mathbf{9.} & 3. & 0 & \mathbf{7.} & 0 \\ 0 & 0 & 5. & \mathbf{8.} & \mathbf{4.} \\ 0 & \mathbf{7.} & \mathbf{8.} & 1. & \mathbf{2.} \\ \mathbf{1.} & 0 & \mathbf{4.} & \mathbf{2.} & 6. \end{pmatrix} \end{array} \qquad G_A = \quad$$

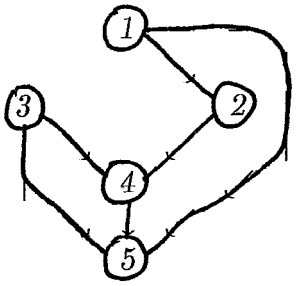


Fig 3.1: Grafo Associado a uma matriz Esparsa (Simétrica)

- Entrada de Dados
- Reordenamento de Equações e Variáveis
- Obtenção da Estrutura de Fatores
- Fatoração Numérica
- Substituições Triangulares

sendo desta forma, consideradas em maior detalhe oportunamente ao longo deste capítulo (e complementarmente no apêndice A, onde se apresenta um sumário de todas as estruturas utilizadas neste trabalho).

Do ponto de vista da simplicidade de especificação, a utilização de listas de arestas, contendo as informações (*linha, coluna, valor*) de cada coeficiente não nulo, é a que se mostra mais adequada para a *interface* com o usuário, durante a fase de *entrada de dados.*

Este formato não é o mais econômico ou prático para o tratamento nas fases subseqüentes, de modo que a solução adotada pelos pacotes de resolução esparsa, é converter-se a entrada fornecida para um formato interno por listas de adjacências.

A representação computacional de listas de adjacências pode ser implementada por um amplo leque de estruturas de dados [6], destacando-se: listas encadeadas mediante apontadores, listas lineares seqüencialmente armazenadas, listas encadeadas por blocos seqüencialmente armazenados, vetores de *bits*, entre outras.

Para a fase de *reordenamento das equações*, a utilização de listas encadeadas se mostra a mais indicada na maioria das implementações, pois nesta fase as estruturas de representação matricial estarão sendo dinâmicamente manipuladas, até se alcançar a estrutura final da matriz de fatores resultante.

Uma vez obtida a *estrutura de fatores*, a representação matricial mais indicada para as *fases numéricas* de eliminação e *substituições triangulares*, passa a ser baseada em listas de adjacências seqüencialmente armazenadas, por permitir uma maior eficiência no acesso aos elementos não nulos contíguamente armazenados.

Representações para Matrizes Esparsas Simétricas

Visando aproveitar-se da simetria e esparsidade das matrizes envolvidas no processo de solução, mostra-se conveniente o armazenamento apenas de seus elementos

diagonais e daqueles situados em porções triangulares superiores (acima da diagonal) que venham a assumir valores não nulos com o decorrer das etapas de eliminação.

Uma forma de armazenamento tipicamente utilizada para a representação de matrizes da forma triangular, é justapor-se cada uma de suas linhas, concatenando-as seqüencialmente em um único vetor estendido.

Em face à natureza esparsa das matrizes envolvidas, o que se utiliza na prática é uma representação unidimensional como a indicada, porém armazenando-se apenas os elementos não nulos de cada uma das linhas.

Mediante o uso de apontadores auxiliares, como será visto mais adiante, torna-se possível delimitar as posições iniciais e finais da representação de cada uma das linhas isoladamente, viabilizando esquemas de acesso seqüencial a seus elementos.

Para todos os esquemas de representação adotados, o armazenamento dos elementos diagonais se fará sempre em um vetor à parte da estrutura de representação triangular esparsa.

Assim, a menção à estrutura de elementos de cada uma das linhas ou colunas de uma matriz simétrica armazenada, assumirá sempre apenas a porção triangular correspondente, excluindo-se seu elemento diagonal.

Apresenta-se a partir deste ponto, algumas das principais estruturas de representação esparsa a serem adotadas (especialmente na implementação das fases numéricas do processo de solução).

Representação Seqüencial Contígua

Tomemos como exemplo a matriz da Figura 3.1, com uma representação seqüencial por listas de adjacências apresentada na Figura 3.2.

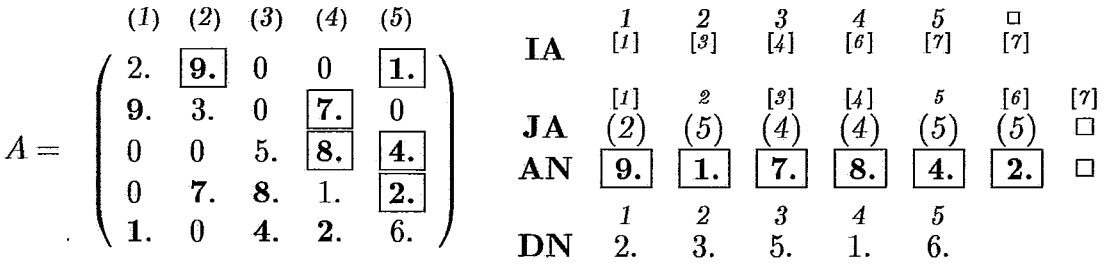


Fig 3.2: Representação Seqüencial Contígua (Adjacências)

Esta representação faz uso de dois vetores auxiliares de índices **IA** e **JA**, além dos vetores **AN** e **DN** contendo respectivamente os valores numéricos não nulos da porção triangular superior de A e de sua diagonal.

Para se compreender o esquema proposto, tomemos duas linhas consecutivas i e $i+1$ da matriz representada e suas respectivas estruturas $S^{(i)}_{\text{sup}}(A)$ e $S^{(i+1)}_{\text{sup}}(A)$. Assumindo-se que seus coeficientes não nulos na porção triangular superior (excluindo-se a diagonal) sejam respectivamente a_{i,j_1}, a_{i,j_2}, ..., a_{i,j_m} e a_{i+1,k_1}, a_{i+1,k_2}, ..., a_{i+1,k_p}

então o que se armazena em posições consecutivas nos vetores **JA** e **AN** é respectivamente j_1, j_2, ..., j_m, k_1, k_2, ..., k_p e a_{i,j_1}, a_{i,j_2}, ..., a_{i,j_m}, a_{i+1,k_1}, a_{i+1,k_2}, ..., a_{i+1,k_p}.

O vetor **IA** de dimensão $n+1$, contém em seus n primeiros elementos, apontadores para as posições iniciais nos vetores **JA** e **AN** onde se armazenam seqüencialmente as informações relativas aos coeficientes não nulos de cada uma das linhas associadas de A. Em seu último elemento, o vetor **IA** contém um apontador para uma posição consecutiva a última efetivamente utilizada na representação dos vetores **JA** e **AN**, visando a delimitação terminal destas estruturas.

O vetor **JA** por sua vez, contém os índices das colunas em A, associadas a cada um dos elementos não nulos representados em **AN**. Este vetor possui dimensão idêntica ao de coeficientes **AN**, correspondendo ao número total de elementos triangulares não nulos da matriz associada.

O armazenamento dos elementos diagonais é feito através do vetor denso **DN** e de dimensão igual a n (pelo fato de serem consideradas matrizes definidas positivas, possuindo portanto todos os seus elementos diagonais distintos de zero, como notado na seção 2.4).

Na forma particular de representação considerada, o armazenamento dos elementos não nulos em cada linha é assumido de forma seqüencial *contígua*, implicando que o armazenamento dos elementos não nulos de uma dada linha, inicie a partir da posição de memória consecutiva a do término do armazenamento dos elementos da linha anterior.

Deste modo, o acesso aos coeficientes triangulares não nulos de cada linha $1 \leq i < n$, ocorre partindo-se dos elementos **AN[IA**$[i]$**]** até **AN[IA**$[i+1]-1$**]**, excluindo-se particularmente o caso da linha n (por não possuir elementos triangulares além de sua diagonal).

Representação Seqüencial Segmentada

Uma forma alternativa de representação, apresentada na Figura 3.3, e que é mais flexível do que a anterior, pode ser obtida lançando-se mão de um novo vetor auxiliar de índices **IAF**, contendo apontadores para as posições finais na representação de cada linha (correspondendo a informação **IA**$[i+i]-1$ na representação contígua da Figura 3.2).

Neste caso, os acessos aos coeficientes de cada linha $1 \leq i < n$ passam a ser efetuados partindo-se dos elementos **AN[IA**$[i]$**]** até **AN[IAF**$[i]$**]**.

A flexibilidade que se tem com esta representação *segmentada* como notado em [6] é que o armazenamento de blocos de elementos não nulos (correspondentes a linhas consecutivas entre si) não necessita ser contíguo, viabilizando desta forma alterações estruturais como a adição ou remoção de linhas, sem requerer o remanejamento dinâmico de toda a estrutura de elementos em **JA** e **AN**.

Outra vantagem está na possibilidade de se permitir o mapeamento da estrutura de elementos não nulos da matriz original, utilizando-se posições correspondentes dentro da estrutura alocada para a representação da matriz de fatores. Isto só é

possível em representações de forma *segmentada*, pelo fato da estrutura de elementos original ser um subconjunto da estrutura de fatores (como será visto na seção 3.3), o que mediante a redefinição adequada dos ponteiros **IAF**, permite que o mesmo espaço de armazenamento possa ser compartilhado em fases distintas do processo de solução.

Por estas facilidades adicionais, representações *seqüenciais segmentadas* como as da Figura 3.3 serão preferencialmente adotadas como base neste trabalho.

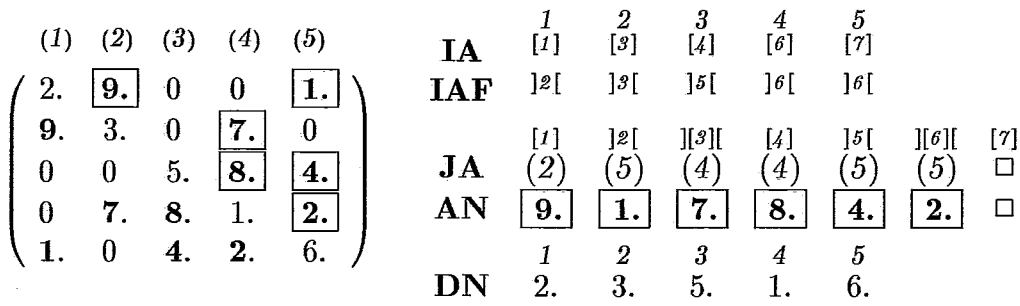


Fig 3.3: Representação Seqüencial Segmentada (Adjacências)

Representação Seqüencial Alternativa

Uma variante da representação segmentada utiliza um vetor de índices **NA** no lugar do vetor **IAF**. Este vetor contém o número de elementos não nulos de cada linha i, de modo que o acesso aos seus coeficientes passa a ser processado partindo-se de **AN[IA**[i]] até **AN[IA**[i]+**NA**[i]−1], para $1 \leq i < n$.

A vantagem nesta representação, ilustrada na Figura 3.4 é que o vetor **NA** pode ser armazenado como um vetor de inteiros de palavra reduzida, nas linguagens computacionais onde estruturas ocupando um número de bits inferior ao tamanho de palavra podem ser alocadas, como por exemplo **INTEGER*2** em FORTRAN, ou **short int** em C.

Em contrapartida esta representação tem como ônus um *overhead* adicional de cálculo e endereçamento inteiro, ao se determinar explicitamente as posições finais **IA**[i]+**NA**[i]−1, e que já se encontram previamente armazenadas nas representações utilizando-se o vetor **IAF** no lugar de **NA**.

Estrutura Adotada para a Fase Numérica

Uma vez determinada a estrutura da matriz de fatores (como será detalhado na seção 3.3), no lugar de uma representação para a matriz A original, o que se passa a fornecer como entrada para as fases subseqüentes de eliminação e substituições de variáveis, são representações para as matrizes U e D, mapeando-se previamente os coeficientes do sistema original em suas posições correspondentes nestas novas estruturas.

$$A = \begin{array}{c} \begin{array}{ccccc} (1) & (2) & (3) & (4) & (5) \end{array} \\ \left(\begin{array}{ccccc} 2. & \boxed{9.} & 0 & 0 & \boxed{1.} \\ 9. & 3. & 0 & \boxed{7.} & 0 \\ 0 & 0 & 5. & \boxed{8.} & \boxed{4.} \\ 0 & 7. & 8. & 1. & \boxed{2.} \\ 1. & 0 & 4. & 2. & 6. \end{array} \right) \end{array}$$

	1	2	3	4	5		
IA	[1]	[3]	[4]	[6]	[7]		
NA	2	1	2	1	0		
	[1]	2	[3]	[4]	5	[6]	[7]
JA	(2)	(5)	(4)	(4)	(5)	(5)	□
AN	**9.**	**1.**	**7.**	**8.**	**4.**	**2.**	□
	1	2	3	4	5		
DN	2.	3.	5.	1.	6.		

Fig 3.4: Representação Seqüencial Alternativa (Adjacências)

Como se notará mais adiante, o processo de eliminação tende a introduzir elementos potencialmente não nulos em novas posições (inicialmente contendo coeficientes nulos na matriz original). Estes novos elementos, denominados *fill-in's* (conceituados ao final da seção 2.4) passam a ser portanto incorporados à estrutura de elementos não nulos da matriz de fatores resultante, tendo sua localização estrutural determinada previamente em alguma fase de pré-processamento simbólico.

Exceto por conter um número maior de posições à cada linha, de modo a se incluir os *fill-in's* (zerados inicialmente), as estruturas de representação para a fase numérica se mantém análogas as anteriormente apresentadas para o armazenamento *seqüencial segmentado* da matriz original, simplesmente tomando por base agora uma estrutura para o sistema triangular superior resultante.

Nesta representação, lança-se mão de vetores **IU, IUF, JU, UN** e **DI**, com interpretação análoga as estruturas utilizadas para o armazenamento esparso de A, como nos mostra a Figura 3.5.

O vetor **UN** neste caso, deve conter também posições reservadas para os elementos de *fill-in*, (a serem operados na fase numérica). Nota-se deste modo, que a representação da Figura 3.5, assume que a estrutura da matriz original encontre-se mapeada na de fatores resultantes, incluindo pois particularmente a posição correspondente ao *fill-in* $u_{2,5}$ (inicialmente zerada).

Ao final do processo de eliminações, os valores resultantes em **DI** acabarão contendo um valor inverso ao dos elementos diagonais de D (visto serem por construção unitários os elementos diagonais de U, e necessitar-se apenas dos inversos dos valores $d_{i,i}$ na fase de normalização de variáveis).

Observa-se também que por se adotar uma representação em separado para os elementos diagonais, nas implementações com espaço de armazenamento restrito (onde normalmente o vetor **UN** é representado em precisão simples), o vetor **DI** pode ser opcionalmente armazenado em precisão dupla, nos casos em que se opte por privilegiar uma melhor precisão, em detrimento a um pequeno *overhead* em termos de espaço adicional (e que não seria aceitável no caso da representação dos demais fatores, armazenados em **UN**).

Mostra-se conveniente dispor-se também da estrutura por linhas dos elementos não nulos de U^T (visto nela estarem contidas as informações de quais linhas ante-

$$A = \begin{array}{c} \begin{matrix} (1) & (2) & (3) & (4) & (5) \end{matrix} \\ \begin{pmatrix} 2. & \boxed{9.} & 0 & 0 & \boxed{1.} \\ 9. & 3. & 0 & \boxed{7.} & \boxed{0.} \\ 0 & 0 & 5. & \boxed{8.} & \boxed{4.} \\ 0 & 7. & 8. & 1. & \boxed{2.} \\ 1. & 0. & 4. & 2. & 6. \end{pmatrix} \end{array} \qquad b = \begin{pmatrix} \boxed{5.} \\ \boxed{4.} \\ \boxed{9.} \\ \boxed{6.} \\ \boxed{7.} \end{pmatrix}$$

	1	2	3	4	5			
IU	[1]	[3]	[5]	[7]	[8]			
IUF	]2[	]4[	]6[	]7[	]7[			
	1	2	3	4	5	6	7	8
JU	(2)	(5)	(4)	(5)	(4)	(5)	(5)	□
UN	**9.**	**1.**	**7.**	**0.**	**8.**	**4.**	**2.**	□
	1	2	3	4	5			
IUT	[1]	[1]	[2]	[2]	[4]			
IUTF	]0[	]1[	]1[	]3[	]7[			
	][1][	[2]	]3[	[4]	5	6	]7[	
JUT	(1)	(2)	(3)	(1)	(2)	(3)	(4)	
	1	2	3	4	5			
DI	2.	3.	5.	1.	6.			
	1	2	3	4	5			
B	5.	4.	9.	6.	7.			

Fig 3.5: Representação fornecida para as fases Numéricas

riores efetivamente contribuem sobre a linha básica corrente a cada etapa).

Neste caso, apenas a representação da *estrutura* de elementos não nulos da transposta necessita ser armazenada nos vetores **IUT, IUTF** e **JUT** (uma vez que somente os valores numéricos da porção triangular superior são de fato necessários nos processos de cálculo).

No caso de linhas onde não existam elementos não nulos na representação da transposta, toma-se por convenção armazenar os valores $\mathbf{IUT}[i] = \mathbf{IUTF}[i-1]+1$ e $\mathbf{IUTF}[i] = \mathbf{IUT}[i]-1$, de modo a sinalizar a não existência de elementos a se processar, nestas linhas em particular.

Utilização de Estruturas Compostas

Em linguagens computacionais onde tipos estruturados de dados na forma de *records* ou vetores de n'uplas não homogêneas podem ser definidos como em C/C++ ou FORTRAN 90, algumas das estruturas previamente apresentadas na forma de vetores isolados, podem ser mais eficientemente implementadas, se declaradas em estruturas compostas.

Como exemplo temos os pares de vetores **(IU, IUF)**, **(IUT, IUTF)** e **(JU, UN)**, que podem ser declarados em estruturas compostas, uma vez que ao se acessar um de seus vetores componentes, normalmente se acessa em instruções seguintes,

uma posição correspondente no vetor par associado.

Outros Esquemas de Armazenamento

Visando meramente complementar o material desta seção, apresenta-se uma breve descrição de esquemas alternativos para representação de matrizes esparsas (não considerados neste trabalho), reportando-se o leitor as referências indicadas a seguir.

Dentre as formas alternativas de armazenamento para a estrutura da matriz de fatores, a que obteve maior aceitação até hoje, foi introduzida originalmente por Sherman [94] em meados da década de 70.

O esquema proposto visa compactar o tamanho do vetor de índices de colunas **JU**, tomando por base a observação (empírica), de que as linhas finais da matriz de fatores U tendem muitas das vezes a apresentar sub-conjuntos idênticos de índices de seus elementos não nulos (como o caso de sub-matrizes completamente densas, entre outros padrões de significativa freqüência).

No caso das linhas passíveis de compactação, o que Sherman propôs foi, lançar-se mão de um novo vetor auxiliar (**IJU** por exemplo), apontando para a posição em **JU**, a partir da qual a estrutura de índices já armazenados (de alguma linha prévia) possa ser igualmente utilizada na representação das atuais linhas sendo consideradas.

Este esquema propicia economias significativas em termos de espaço de armazenamento em muitos dos casos, introduzindo porém *overheads* em termos de tempo de execução, em função do nível adicional de "indirecionamento" necessário para se obter o acesso aos índices das linhas compactadas, razão pelo qual não foi adotado no presente trabalho.

Duas outras formas de representação visando igualmente a redução no espaço de armazenamento dos índices de elementos não nulos da matriz de fatores, devem-se aos trabalhos de Bank e Smith [11] e de Liu [75].

No primeiro caso, adota-se um esquema de representação (voltado para eliminações via *bordering* [10]) no qual a estrutura de índices da matriz de fatores é completamente dispensável, lançando-se mão apenas dos índices de elementos não nulos da matriz original, e de funções indexadoras, permitindo o acesso direto sobre a estrutura numérica de fatores resultantes.

Novamente este esquema apresenta como inconveniente, a introdução de ineficiências a nível de tempo de execução, em função das operações adicionais de indexação inevitavelmente empregadas em favor de um menor espaço de armazenamento.

O trabalho de Liu por sua vez, consegue estabelecer novos padrões para a compactação das informações necessárias à caracterização da estrutura de fatores, explorando-se uma representação implícita a partir de sua *árvore de eliminação* [74] (fundamentada em técnicas eficientes para união de conjuntos como [96], [97]). A estrutura utilizada porém, supõe um armazenamento por *colunas* de U (linhas de L no trabalho original), o que inviabilizaria sua aplicação nas técnicas de geração e

armazenamento por *linhas*, consideradas e introduzidas no corrente trabalho.

3.2 Ordenamento Visando a Redução de "Fill-In"

Como foi notado na seção 2.4, o processo de eliminação esparsa tende a introduzir novos elementos não nulos com o decorrer de suas etapas.

O modo de se tentar amenizar o esforço computacional introduzido com a criação de novos fatores, se dá mediante um reordenamento prévio das linhas e colunas da matriz original, visando uma minimização do número destes elementos, denominados *fill-in's*.

Como notado na seção 2.4, no caso das matrizes simétricas, a única forma de permutação possível de modo a se preservar a simetria do problema, se dá mediante pivoteamentos estruturais, tomando-se como candidatos, elementos diagonais a cada etapa.

Tomemos como exemplo na Figura 3.6 uma matriz A e a estrutura de seus elementos não nulos, representada em seu Grafo Associado G_A, assim como a matriz de fatores U resultante e seu Grafo de Fatores $G_U^* = G_{U^T+U}$ (com "•" denotando as posições de elementos originalmente não nulos, "⊗" os elementos de *fill-in* introduzidos com o processo de eliminação, e "o" os elementos cancelados).

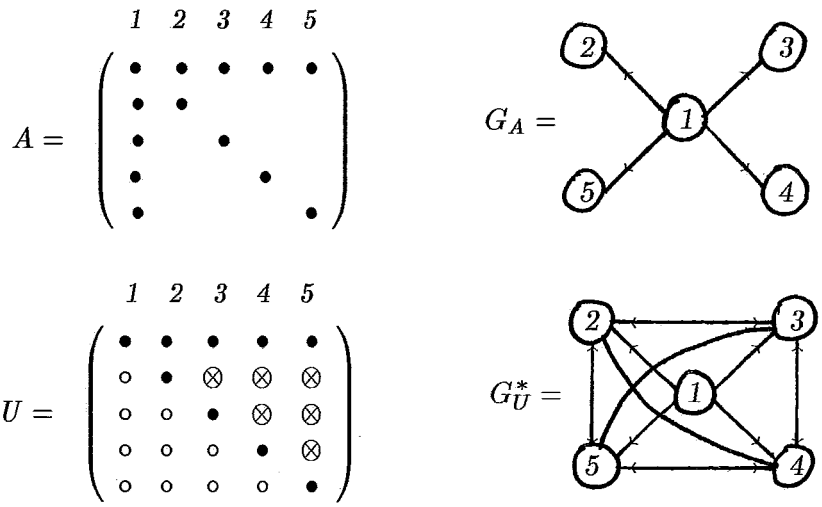


Fig 3.6: Matriz Original e de Fatores e seus Grafos Associados

Se efetuarmos o processo de eliminação Gaussiana (cancelando-se os elementos inferiores por colunas) na ordem natural (de $a_{1,1}$ até $a_{5,5}$) em que foram representadas as linhas e colunas desta matriz, percebe-se que a matriz de fatores U resultante será completamente densa, visto que ao se eliminar os elementos $a_{2,1}$, $a_{3,1}$, $a_{4,1}$ e $a_{5,1}$ subtraindo-se múltiplos escalares da primeira linha de cada uma das demais, novos elementos acabarão sendo introduzidos nas posições $u_{2,3}$, $u_{2,4}$, $u_{2,5}$, $u_{3,4}$, $u_{3,5}$ e $u_{4,5}$.

Para o exemplo apresentado na Figura 3.6, percebe-se que mediante permutações

entre suas primeira e quinta linhas e colunas, levando-se a matriz original a forma reordenada A_l como na Figura 3.7, o número de elementos de *fill-in* introduzidos na nova matriz de fatores U_l é consideravelmente reduzido em relação ao caso anterior (e particularmente nulo neste exemplo em questão).

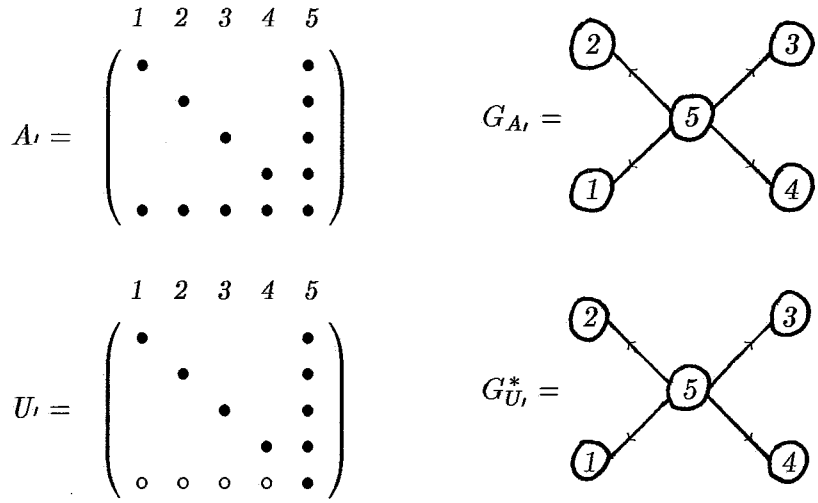


Fig 3.7: Matriz Reordenada e de Fatores e seus Grafos Associados

Para uma melhor compreensão do problema apontado no exemplo original e da estratégia de solução adotada (no exemplo reordenado), mostra-se interessante, uma re-interpretação do ocorrido, a luz de noções como a de *grau* e *valência* associadas a cada nó, conceituadas (análogamente à [81]) a seguir.

Em todos os casos, assumem-se processos de eliminação com *cancelamentos* por *colunas* (e *geração dos fatores* por *sub-matrizes*), e representações por Grafos Associados (à estrutura de elementos não nulos gerada) a cada etapa do processo.

Definição 3.2.1 (Grau de um Nó) *Define-se por grau δ_i de um dado nó i de um grafo não orientado, como o número de arcos a ele conectados.*

Definição 3.2.2 (Valência de um Nó) *Define-se por valência v_i de um dado nó i de uma representação por grafos (associada a i'ezima etapa de um processo de eliminação), como o número de novos arcos introduzidos após a conclusão da etapa corrente de cancelamentos (em comparação com a representação por grafos da etapa anterior).*

Definição 3.2.3 (Valência Total) *Define-se por valência total $\mathcal{V}_U$ associada a representação por grafos de um processo de eliminação, como o número total de novos arcos introduzidos no grafo de fatores resultante ao final de todo o processo (em comparação com o número de arcos da representação por grafos da matriz original).*

A luz destas definições, no exemplo considerado na Figura 3.6 tem-se $\delta_1 = 4$, $v_1 = 6$ e $\mathcal{V}_U = 6$, e no caso do reordenamento proposto no exemplo seguinte $\delta_1 = 1$, $v_1 = 0$ e $\mathcal{V}_{U_l} = 0$.

O que intuitivamente se observou, sendo posteriormente comprovado de forma empírica, é que escolhendo-se a cada etapa, nós com o *menor grau* ou *menor valência*

como candidatos a pivô, a matriz de fatores resultante tende normalmente a apresentar uma *valência total* inferior a que seria obtida a partir da matriz original (caso não se permitisse a aplicação de pivoteamentos, visando a minimização de algum critério).

Serão considerados a seguir, alguns destes critérios passíveis de adoção na escolha dos candidatos a pivô (a cada etapa), tendo por objetivo a minimização do número de elementos de *fill-in* introduzidos ao longo de todo o processo de eliminação.

Critérios de Ordenamento

A determinação de uma seqüência de permutações capazes de minimizar a criação de elementos de *fill-in* no caso geral (de matrizes simétricas de qualquer dimensão), é um problema NP-Completo [101], de modo que o que se pode aplicar como solução prática, são métodos heurísticos, baseados em algum critério visando uma redução local do número de novos elementos introduzidos a cada etapa do processo de eliminações.

Ao final desta seção, apresentam-se algumas alternativas (denominadas *T1*, *T2* e *T3*), baseadas em critérios desenvolvidos originalmente por Tinney no final da década de 60.

Na literatura encontram-se muitos critérios e variantes baseadas no esquema *T2* [98], e que ficou conhecido como critério de permutação segundo o "menor grau" (*minimum degree ordering*) [49] (podendo ser encarado como uma particularização do critério de Markowitz [79] para o caso simétrico).

Critérios mais simples do que este, como o esquema *T1*, levando em conta apenas o número de elementos não nulos de cada linha da matriz original (e determinando a seqüência de permutações estáticamente em função apenas de um ordenamento crescente destes valores), não chegam a competir com o critério do menor grau, em termos de percentual de redução no número de *fill-in's*.

Critérios mais complexos, como o esquema *T3* [19], conhecido como o de "menor *fill-in* local" (escolhendo-se como pivôs, os nós de menor valência a cada etapa), embora em alguns casos conseguindo reduções significativas no percentual de *fill-in's* em comparação com o critério *T2*, na maioria das vezes mostram-se computacionalmente mais onerosos do que o segundo critério, não justificando a sua utilização, exceto nos casos onde o dispêndio de tempo adicional, possa ser compensado ao longo de várias re-fatorações mais eficientes (de matrizes com mesma estrutura).

Em alguns ramos da engenharia (como notado na seção 1.2) outros critérios de ordenamento (como os baseados em *nested dissection* [48]), são particularmente mais adaptados para as matrizes tipicamente encontradas em áreas como as de Elementos Finitos e Cálculo Estrutural.

Atualmente, em função da introdução de novas arquiteturas paralelas, critérios alternativos de ordenamento, vem merecendo a atenção da literatura, visando concomitantemente a minimização de *fill-in*, juntamente com o aumento da independência das linhas passíveis de eliminação em paralelo a cada etapa. O que se adota muitas das vezes nestes casos, são variantes do critério de *menor grau*, baseadas na ex-

ploração de estruturas associadas (como a de *árvore de eliminação*) [58], [57], [3], [73], ou de critérios adicionais para seleção de pivôs (entre candidatos com o mesmo grau) [86] e [51].

Processamento mediante Grafos de Eliminação

Alguns dos métodos de ordenamento, requerem para sua implementação, a simulação estrutural de toda a seqüência de cancelamentos (por *colunas*) efetuada durante o processo de eliminação (visando levar a matriz original a forma triangular).

Para tal, utiliza-se normalmente uma estrutura de representação baseada em *Grafos de Eliminação* (a ser definida a seguir), e que será considerada um pouco mais atentamente num exemplo particular, ilustrado nas Figuras 3.8 à 3.10 (com "•" denotando os elementos originalmente não nulos, "⊗" os elementos de *fill-in* introduzidos, "o" os elementos inferiores cancelados e "□" os fatores superiores definitivamente gerados).

Nas definições a seguir, lança-se mão da noção de Matrizes de Transformação $A^{(\cdot)}$ (introduzida na seção 2.4), considerando-se particularmente as representações à cada etapa, para o caso de processos de *cancelamentos* por colunas, com respectiva *geração dos fatores* por sub-matrizes.

Definição 3.2.4 (Grafo de Eliminação) *Para cada etapa $i = 0, ..., n$ de um processo de cancelamento por Colunas, define-se como o Grafo de Eliminação $G_A^{(i)}$, o Grafo Associado à estrutura de elementos não nulos da submatriz $A_i^{(i)}$, formada a partir dos elementos $a_{j,k}^{(i)}$ da Matriz Transformada $A^{(i)}$ para $j, k = i+1, ..., n$.*

Complementarmente, pode-se definir:

Definição 3.2.5 (Grafo de Fatores Resultantes) *Para cada etapa $i = 0, ..., n$ de um processo de cancelamento por Colunas, define-se como o Grafo de Fatores Resultantes $G_U^{*(i)}$, o Grafo Associado à estrutura de elementos não nulos da matriz $A_u^{(i)}$, formada anulando-se os elementos $a_{j,k}^{(i)}$ de $A^{(i)} + A^{T(i)}$ para $j, k = i+1, ..., n$.*

A partir destas definições, observa-se particularmente que $G_A^{(0)} = G_A$, $G_U^{*(n)} = G_U^* = G_{U^T+U}$ e $G_A^{(n)} = G_U^{(0)} = \emptyset$. Nota-se também, que os Grafos de Eliminações são sempre conexos, ao passo que os Grafos de Fatores Resultantes só atingem a conectividade total em sua última instância $G_U^{*(n)}$.

No exemplo ilustrado nas Figuras 3.8 à 3.10, assume-se que o grafo $G_A^{(0)} = G_A$ esteja disponível inicialmente, exibindo-se apenas a seqüência $G_A^{(1)}$, $G_A^{(2)}$ e $G_A^{(3)}$ de transformações correspondentes as três primeiras etapas do processo de eliminação. Para a conclusão deste processo, ainda seriam necessárias as transformações de $G_A^{(4)}$ à $G_A^{(n-1)} = G_A^{(10)}$ (não exibidas), até ser alcançada a estrutura final da matriz de fatores.

A criação de *fill-in's* se dá quando após a remoção das arestas associadas ao nó a ser eliminado na etapa corrente, introduzem-se novas conexões entre todos os

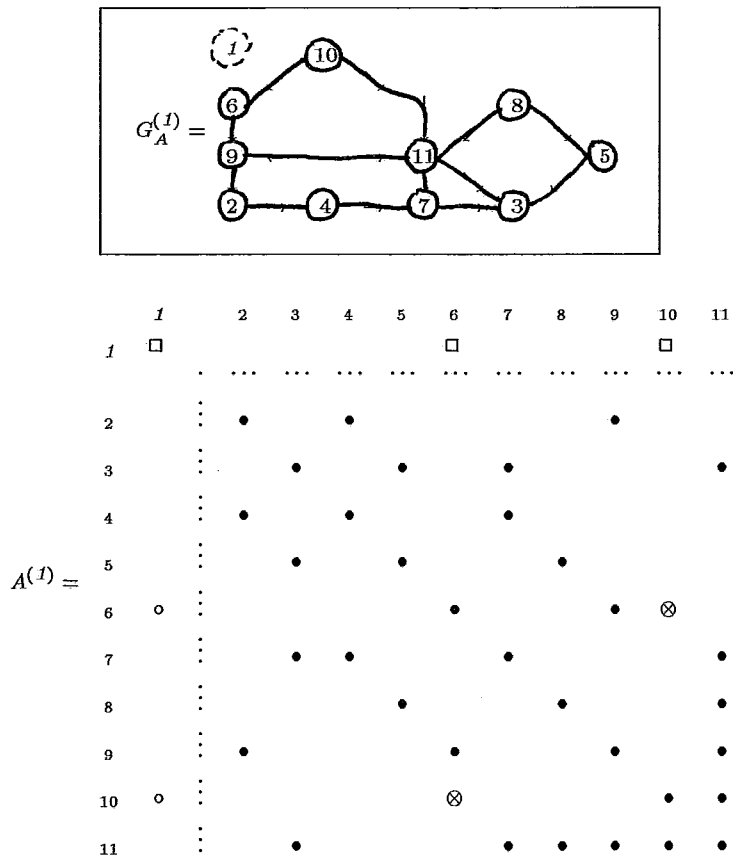


Fig 3.8: Primeira Etapa de Transformações sobre o Grafo de Eliminação

nós originalmente a ele conectados (formando-se um *clique*), e que previamente não existiam, na etapa anterior.

Nas figuras apresentadas convencionou-se representar as novas conexões introduzidas no Grafo de Eliminação associado, por arcos pontilhados, circundando-se desta forma também o nó *pivô* (em itálico), removido da representação ao final de cada etapa.

Assume-se originalmente na Figura 3.8, que com a remoção do primeiro nó, uma nova conexão entre os nós 6 e 10 tenha sido introduzida, gerando-se particularmente o elemento de *fill-in* $a_{6,10}^{(1)}$ (e seu simétrico correspondente).

Percebe-se na Figura 3.9 que a remoção do segundo nó, acaba por criar uma conexão entre os nós 4 e 9, correspondendo ao elemento de *fill-in* $a_{4,9}^{(2)}$ (e seu simétrico) introduzidos neste caso.

Com a eliminação do terceiro nó, exemplificada na Figura 3.10, criam-se novas conexões entre os nós 5 e 7 e entre 5 e 11, que não existiam previamente, gerando-se os elementos $a_{5,7}^{(3)}$ e $a_{5,11}^{(3)}$ (e seus simétricos na porção triangular inferior).

O processo completo acabaria por se estender até a décima etapa, quando o Grafo de Eliminação associado, passaria a conter apenas o nó 11 (particularmente no caso do exemplo considerado).

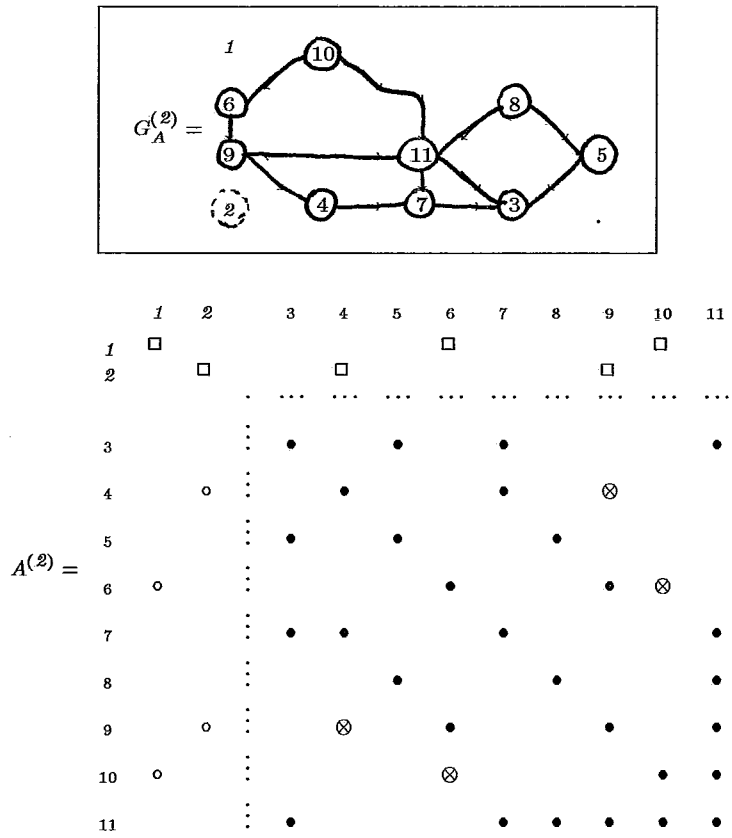


Fig 3.9: Segunda Etapa de Transformações sobre o Grafo de Eliminação

Ordenamento segundo o Critério do Menor Grau

O ordenamento segundo o critério do *menor grau*, pode ser implementado mediante uma simulação *estrutural* do processo de *eliminação direta por colunas* (sem que se efetue qualquer operação aritmética de ponto flutuante).

Para tal, assume-se que uma representação por Grafos de Eliminação $G_A^{(0)} = G_A$ seja fornecida como entrada, executando-se os seguintes passos a cada etapa básica i do processo:

(a) tomar como critério para seleção do elemento *pivô* (a ter os elementos sub-diagonais da coluna básica, estruturalmente cancelados), o nó na representação por Grafos de Eliminação $G_A^{(i-1)}$ possuindo o *menor grau* (de conexões aos demais nós)

(b) remove-lo (juntamente com as conexões a ele associadas) na nova representação por grafos $G_A^{(i)}$ (como notado nas Figuras 3.8 à 3.10)

(c) estabelecer conexões mútuas entre todos nós originalmente a ele conectados (incorporando-as em $G_A^{(i)}$), estabelecendo-se um clique, correspondente a criação de novos elementos de *fill-in* (que porventura tenham sido introduzidos com o cancelamento estrutural dos elementos sub-diagonais da coluna *pivô*)

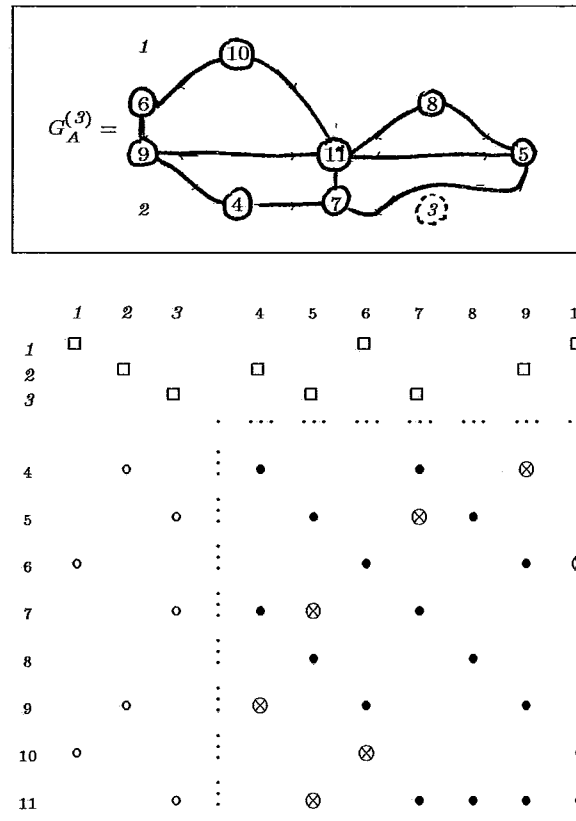


Fig 3.10: Terceira Etapa de Transformações sobre o Grafo de Eliminação

A cada etapa, pode ser encontrado mais de um nó, satisfazendo ao critério do *menor grau*, e o que se tem neste caso é uma liberdade na escolha do elemento *pivô* a ser eliminado.

Diferentes critérios de "desempate" na escolha do candidato de menor grau, conduziram a um grande leque de variações sobre o critério básico *T2*.

O critério mais simples de todos e utilizado particularmente em muitas das implementações escalares, é simplesmente tomar-se o primeiro nó encontrado, dentre o conjunto de nós satisfazendo ao menor grau a cada etapa básica do processo.

Como nos mostra Duff [32], [26], muitos outros critérios de desempate foram testados ao longo da década de 70, e a conclusão obtida após um longo período de experimentações, foi de que nenhum dos critérios alternativos se sobressaiu significativamente sobre os demais no tocante apenas a redução do número de *fill-in's*.

Estruturas de Dados para a fase de Ordenamento

Para a fase de ordenamento, é aconselhável a utilização de estruturas de representação baseadas em listas "encadeadas" de adjacências, pois as informações a serem manipuladas vão sendo dinâmicamente alteradas com a evolução do processo.

Representações alternativas mais eficientes para o critério de *menor grau* [49], [47], [46], baseiam-se em conceitos como os de *"reachable sets"*, e eliminação em

"massa" de nós, explorando-se a estrutura de *cliques* introduzidos durante o ordenamento.

A vantagem na utilização destas estruturas mais sofisticadas, é que o processo de ordenamento pode ser implementado com espaço de trabalho proporcional apenas aos dados de entrada, efetuando-se toda a seqüência de operações apenas sobre a representação mais elaborada, adotada para a matriz original.

Isso garante a estes métodos, além de uma maior eficiência, a total previsibilidade do espaço de armazenamento para as fases posteriores, incluindo-se o número de *fill-in's* gerados, sem que se precise simular todo o processo de eliminações, como nas implementações convencionais.

A única desvantagem é não poderem fornecer como sub-produto, a estrutura da matriz de fatores resultante, por operarem apenas sobre representações compactas a partir da matriz original.

Como referência a estas e outras abordagens, reporta-se a [49], para um *survey* dos mais recentes aprimoramentos incorporados nas atuais implementações do critério de *menor grau.*

Alternativas de Ordenamento baseadas nos Critérios de Tinney

Apresentam-se agora, a nível de pseudo-código, as três Alternativas de ordenamento, propostas originalmente por Tinney [98].

Assume-se que uma representação por Grafos Associados G_A da matriz original, seja fornecida como entrada, atualizando-se (no caso dos critérios *T2* e *T3*) a estrutura dos Grafos de Eliminação $G_A^{(i)}$ correspondentes a cada etapa, fornecendo-se como resultado final em todos os casos, o vetor IORD de índices (ditando a seqüência de linhas a serem eliminadas durante a fase numérica de eliminações).

Alternativa 3.2(a) *Ordenamento pelo Critério T1*

para i **de** 1 **até** n **faça:**

(a) Dentre o conjunto de todos os nós não eliminados de G_A, escolha o nó j com o menor de todos os graus

(b) IORD $[i] \leftarrow j$

(c) Marque o nó j como "já eliminado"

Alt 3.2(a): *Ordenamento pelo Critério T1*

Uma vez obtido o vetor de ordenamento IORD, nos casos em que múltiplas fatorações de matrizes com a mesma estrutura venham a ser realizadas, mostra-se mais eficiente para a fase numérica, efetuar-se uma permutação física entre as linhas e colunas da matriz original, aplicando-se o ordenamento determinado (com a primeira linha/coluna da matriz permutada, correspondendo a IORD [1] da matriz

Alternativa 3.2(b) *Ordenamento pelo Critério T2*

$G_A^{(0)} \leftarrow G_A$

para i **de** 1 **até** n **faça:**

(a) Dentre o conjunto de todos os nós não eliminados de G_A, escolha o nó j com o *menor grau*

(b) IORD $[i] \leftarrow j$

(c) Marque o nó j como "já eliminado" e processe todas as alterações correspondentes a sua remoção no Grafo de Eliminação $G_A^{(i)}$ associado, estabelecendo-se um *clique* a partir de todos os nós originalmente a ele conectados (opcionalmente atualizando-se a estrutura $G_U^{*(i)}$ do Grafo de Fatores Resultantes)

Alt 3.2(b): *Ordenamento pelo Critério T2*

Alternativa 3.2(c) *Ordenamento pelo Critério T3*

$G_A^{(0)} \leftarrow G_A$

para i **de** 1 **até** n **faça:**

(a) Dentre o conjunto de todos os nós não eliminados de G_A, escolha o nó j com a menor *valência*, tal que sua eliminação venha a resultar na criação do menor número possível de *fill-in's* (simulando-se o processo de eliminação para cada um dos candidatos a *pivô* da etapa corrente)

(b) IORD $[i] \leftarrow j$

(c) Marque o nó j como "já eliminado" e processe todas as alterações correspondentes a sua remoção no Grafo de Eliminação $G_A^{(i)}$ associado, estabelecendo-se um *clique* a partir de todos os nós originalmente a ele conectados (opcionalmente atualizando-se a estrutura $G_U^{*(i)}$ do Grafo de Fatores Resultantes)

Alt 3.2(c): *Ordenamento pelo Critério T3*

original, e assim sucessivamente, até se alcançar a n'ezima linha/coluna da matriz reordenada, correspondendo a IORD $[n]$ na representação original).

No caso das Alternativas *T2* e *T3*, um sub-produto passível de ser obtido com a fase de ordenamento (em algumas implementações), é a estrutura final $G_U^{*(n)} = G_U^* = G_{U^T + U}$ do Grafo de Fatores Resultantes (incluindo-se todos os elementos de *fill-in* introduzidos ao longo do processo de eliminações).

Tal estrutura pode ser obtida, incorporando-se a cada etapa i, as informações removidas do Grafo de Eliminações $G_A^{(i)}$ (ou seja o nó *pivô* e suas conexões aos nós originalmente a ele conectados), introduzindo-se estas informações, na representação

$G_U^{*(i)}$ do Grafo de Fatores Resultantes (como na Figura 3.11, para o exemplo particularmente considerado nesta seção).

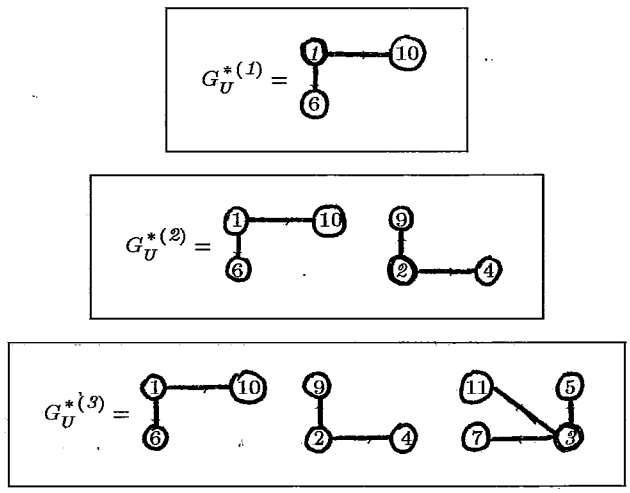


Fig 3.11: Grafos de Fatores Resultantes a cada Etapa

Alguns critérios de ordenamento não considerados, como os do tipo *nested dissection*, ou implementações mais eficientes do critério de menor grau, baseadas em *reachable sets* [49], não fornecem explicitamente a estrutura de fatores resultantes, assim, uma fase auxiliar de geração desta informação adicional, torna-se nestes casos indispensável, passando a ser o objeto de estudo da próxima seção.

3.3 Obtenção da Estrutura de Fatores

Em alguns procedimentos visando a redução de *fill-in*, a estrutura da matriz de fatores U resultante, não pode ser fornecida como um sub-produto do processo de ordenamento.

Nestes casos, uma etapa auxiliar de determinação das posições estruturais dos fatores triangulares, conhecida como "fatoração simbólica" [45], mostra-se adequada, uma vez que de posse desta informação, pode-se implementar de forma mais eficiente, os procedimentos da fase numérica, a serem vistos na próxima seção.

A fase "simbólica" recebe esta denominação, por fundamentar-se essencialmente numa simulação estrutural (sem efetuar atualizações em valor numérico) das operações de combinação linear entre as linhas envolvidas visando a triangularização matricial. Para tal, lança-se mão unicamente de operações de concatenação de índices de colunas, produzindo-se como resultado, a estrutura de elementos não nulos de cada uma das linhas de U (incluindo-se os *fill-in's* introduzidos ao longo do processo).

Uma seqüência de concatenações é exemplificada na Figura 3.12 (a partir da sétima etapa de um processo de eliminação por *linhas*), visando-se neste caso, a obtenção da estrutura resultante da sétima linha da matriz apresentada (a sofrer contribuições estruturais das linhas 3, 4 e 5 em particular). Na Figura em questão "•" denota os elementos não nulos originais, "⊗" os *fill-in's* introduzidos, e "o" os elementos (originalmente não nulos) cancelados.

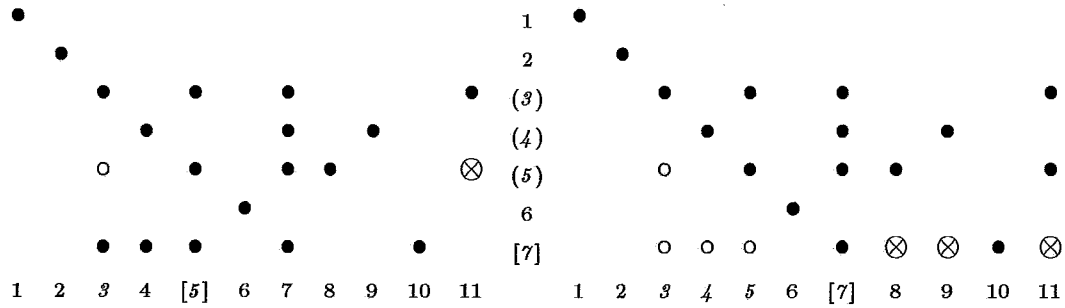


Fig 3.12: Concatenação dos índices de Colunas

No processo de concatenação anterior, ao gerar-se a estrutura da quinta linha, incorporou-se o elemento de *fill-in* $u_{5,11}$, oriundo da combinação linear com a terceira linha (por ocasião do cancelamento do elemento $a_{5,3}$).

Para a geração da estrutura (superior) de fatores da linha 7, devem se concatenar as estruturas (a partir da sétima coluna), das linhas 3, 4 e 5 (correspondentes ao cancelamento dos elementos $a_{7,3}$, $a_{7,4}$ e $a_{7,5}$) e que acabarão por criar novos elementos nas colunas 8, 9 e 11 como se verá a seguir.

Inicialmente apresenta-se a representação original, dos sub-conjuntos de índices de colunas associadas (a partir da sétima) para cada uma das linhas envolvidas:

linha	*colunas*
(*3*)	7 *11*
(*4*)	7 *9*
(*5*)	7 *8* *11*
[*7*]	7 *10*

Concatenando-se o conjunto de índices da linha 7 com os das linhas 3, 4 e 5 (nesta ordem), a estrutura de fatores resultante da sétima linha passará a ser:

linha	*colunas*
[*7*]	7 *10* *11* *9* *8*

Uma propriedade dos métodos de eliminação Gaussiana estudada por Rose [88] e Sherman [94] em meados da década de 70, viabiliza implementações mais eficientes do processo de concatenações estruturais, merecendo particularmente a atenção nesta seção.

O que se consegue é reduzir-se o número de linhas contribuintes, a serem estru-

turalmente concatenadas a linha básica de cada etapa, explorando-se a possibilidade da estrutura de algumas destas linhas, estarem integralmente contidas na estrutura de outras linhas a contribuírem na mesma etapa básica, minimizando-se deste modo, a concatenação de informações redundantes.

O critério para a exclusão das linhas contribuintes, a terem suas estruturas concatenadas (a partir da coluna básica corrente), é descartar-se aquelas contendo elementos não nulos a partir de suas posições diagonais, em colunas anteriores a coluna básica de cada etapa.

No exemplo em questão, a linha 3 em particular, não necessita ser considerada no processo de concatenação, por ocasião da determinação da estrutura de fatores da linha básica 7.

Para o exemplo apresentado, as únicas estruturas originalmente necessárias, para fins de concatenação, seriam pois:

linha	*colunas*
(*4*)	7 *9*
(*5*)	7 *8* *11*
[*7*]	7 *10*

Produzindo como resultado final deste processo:

linha	*colunas*
[*7*]	7 *10* *9* *8* *11*

Tal ocorre, pois a existência do elemento $u_{3,5}$ garante que por ocasião do cancelamento de seu simétrico $a_{5,3}$, a estrutura de colunas dos elementos não nulos da porção triangular superior da terceira linha acabará por ser integralmente propagada na estrutura de elementos da quinta linha.

Este fato dispensa assim a concatenação explícita da estrutura da linha 3 para a geração das estruturas de linhas subseqüentes, uma vez processada sua contribuição sobre a estrutura de sua descendente imediata, visto que a partir deste ponto, contribuições de suas descendentes (como a de número 5) sobre outras linhas, acabarão por implicitamente propagar consigo a estrutura de elementos da terceira linha.

De modo a formalizar-se a propriedade garantindo o descarte de linhas, mostra-se conveniente introduzir-se a seguinte definição (tomando por base a notação $S^{(i)}(U)$ para o conjunto dos índices das colunas dos elementos não nulos da i'ezima linha da matriz de fatores U, introduzida na seção 2.2):

Definição 3.3.1 (Coluna do Primeiro Fator Não Nulo)

$$p(i) = \begin{cases} \min\ S^{(i)}(U) & \text{caso } S^{(i)}(U) \neq \emptyset \\ 0 & \text{caso contrário} \end{cases}$$

Lançando mão desta função indicadora, pode-se restringir o grupo de linhas da matriz de fatores, a serem efetivamente empregadas durante o processo de concatenações, mediante:

Definição 3.3.2 (Conjunto de Linhas Características) *Define-se o conjunto $C_*^{(i)}$ de linhas "características", associadas a i'ezima linha de uma matriz triangular de fatores U, como aquele formado por suas antecessoras* $\{\ j < i \mid\ p(j) = i\ \}$.

Tal grupo de linhas corresponde particularmente a um sub-conjunto de linhas antecessoras a linha base corrente i, cuja concatenação das estruturas de seus elementos não nulos a partir da i'ezima coluna, mostra-se suficiente para abranger integralmente o espectro de colunas dos elementos contribuintes sobre a mesma, na i'ezima etapa do processo de eliminação.

Com base nestas definições, a propriedade observada por Rose e Sherman, viabilizando a redução do esforço computacional na determinação da estrutura de fatores de cada linha, pode ser expressa por:

Propriedade 3.3.3 (Concatenação de Linhas Características)

$$S^{(i)}(U) \;=\; S^{(i)}_{\text{sup}}(A) \;\cup\; \left(\bigcup_{j\ \in\ C_*^{(i)}} \left\{\ S^{(j)}(U)\ \right\} - \{\ i\ \} \right) \tag{3.1}$$

Apresenta-se inicialmente no Algoritmo **3.1** uma versão "preliminar" do processo de fatoração simbólica, a ser posteriormente refinado (explorando-se a propriedade de descarte considerada).

Neste algoritmo, assume-se como entrada apenas $S_{\text{sup}}(A)$ (contendo a estrutura de colunas associadas aos elementos não nulos por cada linha da porção triangular superior da matriz original), produzindo como resultado a estrutura $S(U)$ da matriz de fatores resultante (incluindo-se a posição estrutural de todos os elementos de *fill-in* introduzidos ao longo do processo).

A cada etapa básica i, atualiza-se a estrutura de fatores $S^{(i)}(U)$ da linha base, concatenando-se à mesma, as estruturas $S^{(j)}(U)$ das linhas j que contribuem sobre esta no método de eliminações (por linhas), determinando-se desta forma a posição estrutural de todos os elementos de *fill-in* introduzidos com o decorrer da etapa corrente.

A cada etapa i o algoritmo apresentado efetua o seguinte:

(a) Atualiza a estrutura da linha i, concatenando a ela as estruturas das linhas com índices em $C^{(i)}$

Algoritmo 3.1 *Fatoração Simbólica* (*Versão Preliminar*)

para i **de** 1 **até** n **faça**
$C^{(i)} \leftarrow \emptyset$
para i **de** 1 **até** n **faça**
$S^{(i)}(U) \leftarrow S^{(i)}_{\text{sup}}(A)$
para $j \in C^{(i)}$ **faça**
$S^{(i)}(U) \leftarrow S^{(i)}(U) \cup (S^{(j)}(U) - \{ j+1, ..., i \})$ (a)
para $j \in S^{(i)}(U)$ **faça**
$C^{(j)} \leftarrow C^{(j)} \cup \{ i \}$ (b)

Alg 3.1: *Fatoração Simbólica* (*Versão Preliminar*)

(b) Adiciona o índice i aos conjuntos $C^{(j)}$ de todas as linhas tais que $j > i$ e $u_{i,j}$ é não nulo. (Devido à simetria, essas serão as linhas que no futuro serão afetadas pela linha i no método de eliminações)

Este algoritmo pode ser refinado como mostraremos a seguir.

Refinamentos no processo de Fatoração Simbólica

Lançando-se mão da função indicadora $p(i)$, mostra-se agora que no algoritmo original, a atualização dos conjuntos $C^{(j)}$ pode ser restrita a:

$$C^{p(i)} \leftarrow C^{p(i)} \cup \{ i \}$$

Considere o algoritmo em uma etapa k, e sejam $j_1 = p(k)$, j_2, j_3, ... elementos consecutivos de $S^{(k)}(U)$. Então, $k \in C^{(j_1)}$, $k \in C^{(j_2)}$, $k \in C^{(j_3)}$, ...

Na etapa j_1 ocorre o seguinte:

(1) $S^{(k)}(U)$ é concatenado a $S^{(j_1)}(U)$

(2) O índice j_1 é adicionado a $C^{(j_2)}$, $C^{(j_3)}$, ... devido a (1)

Na etapa j_2 concatenam-se a $S^{(j_2)}(U)$ as estruturas $S^{(k)}(U)$ e $S^{(j_1)}(U)$.

Mas $S^{(j_1)}(U)$ já havia sido concatenado a $S^{(k)}(U)$, e portanto

$$S^{(j_1)}(U) \supset S^{(k)}(U) - \{ j_1 \}$$

Conclui-se: é desnecessário concatenar $S^{(k)}(U)$ a $S^{(j_2)}(U)$.

Pelo mesmo raciocínio, também se mostra desnecessário concatenar $S^{(k)}(U)$ a $S^{(j_3)}(U)$, $S^{(j_4)}(U)$, ... Basta portanto no algoritmo refinado, definir os conjuntos $C^{(j)}$ como acima, lançando-se mão de $C^{f(i)}_*$ em seu lugar (visto corresponderem estes

novos conjuntos exatamente aos das linhas "características" envolvidas no processo de geração da estrutura de contribuições sobre a linha básica a cada etapa).

Fundamentando-se na Propriedade 3.3.3, apresenta-se a seguir, no Algoritmo **3.2**, um processo de fatoração simbólica mais eficiente, proposto originalmente por George e Liu [45] (e padronizado por Eisenstat [39]).

Algoritmo 3.2 *Fatoração Simbólica*

para i **de** 1 **até** n **faça**
    $C_*^{(i)} \leftarrow \emptyset$
para i **de** 1 **até** n **faça**
    $S^{(i)}(U) \leftarrow S_{\sup}^{(i)}(A)$
    para $j \in C_*^{(i)}$ **faça**
        $S^{(i)}(U) \leftarrow S^{(i)}(U) \cup (S^{(j)}(U) - \{ i \})$ (a)
    se $S^{(i)}(U) \neq \emptyset$ **então**
        $C_*^{p(i)} \leftarrow C_*^{p(i)} \cup \{ i \}$ (b)

Alg 3.2: *Fatoração Simbólica*

Observa-se que o passo (a) pode ser implementado por sempre se ter por construção $i = p(j)$ (isto é, i ser o primeiro elemento de $S^{(j)}(U)$).

No exemplo apresentado na Figura 3.12, nota-se ao final das concatenações à linha básica 7, um fato passível de ser encontrado com relativa freqüência durante a determinação estrutural de fatores, e que é o completo "enchimento" da estrutura de colunas, a partir de uma dada linha do processo de eliminações (inevitavelmente acarretando consigo, o total enchimento de toda a sub-matriz de fatores restante, cuja estrutura ainda se encontraria por determinar).

Em face ao enchimento completo, torna-se inteiramente dispensável, levar-se adiante a seqüência de concatenações estruturais, uma vez que a estrutura final da matriz de fatores será inteiramente previsível a partir deste ponto. Uma solução prática neste caso, é abortar-se o processo de fatoração simbólica, uma vez detectada esta condição (logo após se efetuar as concatenações explorando-se a Propriedade 3.3.3), o que no pseudo-código apresentado no Algoritmo **3.2**, corresponderia a inclusão da cláusula a seguir (após o passo (a)):

se $\eta^{(i)}(U) = n - i$ **então**
    para j **de** $i+1$ **até** n **faça**
        $S^{(j)}(U) \leftarrow \{ j+1, \ldots, n \}$
    termine processamento

O que se pode observar, é que em função da natureza estruturalmente "aditiva" do processo de eliminações, a contribuição da estrutura de elementos não nulos de uma linha base acabará por se "propagar" na estrutura de suas "descendentes" (como se verá no próximo capítulo). Tal propagação ocorrendo, quando estas linhas

vierem a sofrer a contribuição estrutural de suas "ancestrais", em alguma etapa subseqüente do processo (durante as operações de combinação linear, visando o cancelamento de seus elementos triangulares inferiores).

Esta estrutura aditivamente propagada, da contribuição das linhas sobre suas descendentes, continuará a ser incorporada nas descendentes de suas descendentes, e assim sucessivamente, até o final do processo de eliminação.

Em função disto, as linhas terminais da matriz de fatores, tendem naturalmente a sofrer um maior acréscimo de *fill-in's*, visto receberem contribuições de linhas que por sua vez receberam contribuições relativas ao processo de agregação estrutural de todas as suas ancestrais. Este fato, é o que faz com que os critérios heurísticos de ordenamento, como o de *menor grau*, sejam bem sucedidos na prática, por visarem retardar ao máximo o "enchimento" da estrutura das linhas a contribuírem sobre suas descendentes (selecionando-se como linha candidata à eliminação a cada etapa, aquela contendo o menor número de elementos em sua porção triangular superior).

Durante o processo de obtenção da estrutura de fatores resultantes, as concatenações efetuadas a cada etapa, podem ser implementadas de forma desordenada segundo o índice de colunas (simplesmente incorporando-se os novos elementos na ordem natural de concatenação aos conjuntos associados).

Este fato pode ser explorado, uma vez que o objetivo da fatoração simbólica concentra-se unicamente na determinação do conjunto de índices de colunas dos elementos não nulos da matriz de fatores resultante, não se importando nesta fase particular do processamento, com o ordenamento particular de cada uma das estruturas de colunas isoladamente determinadas.

Para a fase de resolução numérica, onde assume-se (como descrito na seção 3.1) que a matriz original a ser fornecida para o processo de fatoração, encontre-se mapeada sobre a estrutura de fatores, com os elementos de *fill-in* inicialmente zerados, torna-se indispensável o ordenamento dos conjuntos de índices de colunas associadas a cada linha.

3.4 Resolução Numérica de Sistemas com Mesma Estrutura

Nesta seção se apresentarão procedimentos convencionalmente adotados para a implementação numérica de eliminações $U^T D\, U$ (esparsas) e das fases de substituições triangulares e diagonais subseqüentes.

Serão considerados ambos os processos de cancelamentos (inferiores) por linhas, com geração dos fatores U respectivamente por *Linhas* e por *Colunas*, a cada etapa.

É assumido, também, que a estrutura de elementos não nulos da matriz de fatores tenha sido determinada em alguma fase preliminar do processamento, e os valores numéricos da matriz original mapeados sobre esta estrutura (com as posições relativas aos elementos de *fill-in* inicialmente zeradas).

Os procedimentos de eliminação numérica aqui apresentados, poderão ser indefinidamente re-aplicados para a solução de novos sistemas, estruturalmente idênticos

ao originalmente considerado, permitindo a solução de sucessivos sub-problemas em ramos da engenharia ou otimização (onde tal ocorre com relativa freqüência).

Adição de Vetores Esparsos via Vetor de Trabalho Expandido

A principal operação a ser efetuada em cada etapa básica do processo de eliminações é a combinação linear de vetores esparsos (armazenados de forma seqüencial contígua).

Uma solução normalmente adotada é obtida lançando-se mão de um vetor auxiliar de trabalho **W** de dimensão n, utilizado para a descompactação e acumulação temporária de contribuições, como veremos a seguir.

Considere como exemplo a adição de vetores u^4 e u^5 ao vetor u^7, armazenados de forma seqüencial contígua como nos mostra a Figura 3.13.

Inicialmente, descompacta-se a estrutura esparsa de elementos do vetor u^7 (incluindo-se os *fill-in's* inicialmente zerados), nas colunas correspondentes do vetor de trabalho w (correspondendo a operação $w \leftarrow u^7$ na Figura considerada).

A seguir, efetuam-se (descompactadamente nas colunas correspondentes de w) as operações relativas a $w \leftarrow w + u^4$, armazenando-se em w temporariamente este resultado.

Operam-se posteriormente (de forma descompactada) as operações referentes a $w \leftarrow w + u^5$, armazenando-se no vetor de trabalho o resultado deste processo de cálculo.

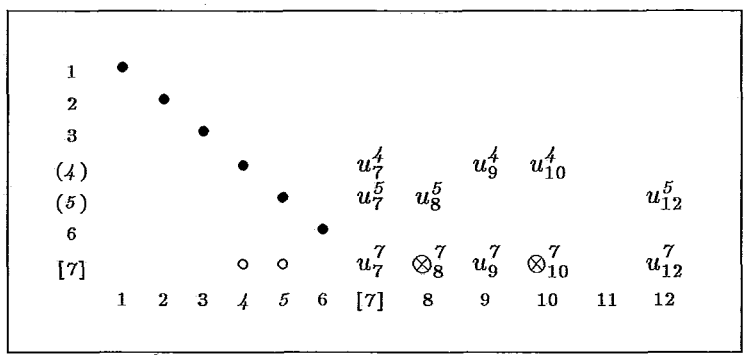
O processo de adição esparsa $u^7 \leftarrow u^4 + u^5$ é completado, armazenando-se compactadamente de volta na estrutura de u^7, os elementos não nulos associados do vetor de trabalho (correspondendo a operação $u^7 \leftarrow w$ do exemplo em questão).

Este processo pode ser formalizado, para uma adição genérica de vetores esparsos da forma $u \leftarrow u + v$ como apresentado na Alternativa **3.4(a)**, assumindo-se um armazenamento seqüencial segmentado para ambos os vetores (como na seção 3.1), com (iu, iuf) e (iv, ivf) delimitando as posições associadas respectivamente a representação de cada vetor.

Esta metodologia para a adição de vetores esparsos foi introduzida originalmente por Tinney [98] e Gustavson [59] e desde então vem sendo adotada como base para a implementação dos procedimentos da fase numérica de eliminações (em pacotes como YSMP [38], SPARSPAK [48], [18] e HARWELL [32], [64]).

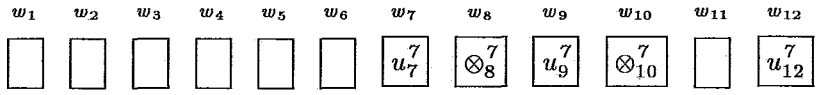
Uma vantagem adicional que pode ser explorada em alguns casos, é o fato de se poder acumular temporariamente em **W**, valores em precisão superior a utilizada no armazenamento dos vetores originais (no caso de se utilizar uma representação em precisão simples para os vetores **U** e **V**, ou no caso de linguagens onde precisão quádrupla possa ser adotada para a representação temporária em **W**).

Nos capítulos 4 e 5 se apresentarão formas mais eficientes de implementação, baseadas na exploração de informações de natureza simbólica, extraídas particularmente da estrutura de cada sistema a ser solucionado.

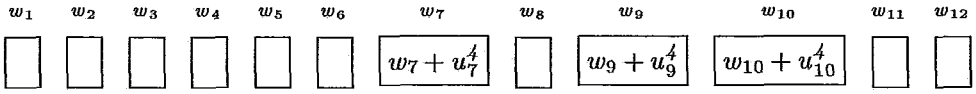


vetor	*elementos*
u^4	u_7^4 u_9^4 u_{10}^4
u^5	u_7^5 u_8^5 u_{12}^5
u^7	u_7^7 $\otimes_8^7$ u_9^7 $\otimes_{10}^7$ u_{12}^7

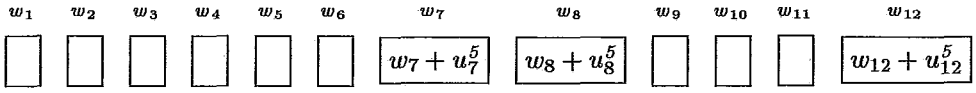
$w \leftarrow u^7$



$w \leftarrow w + u^4$



$w \leftarrow w + u^5$



$u^7 \leftarrow w$

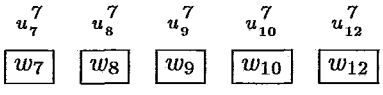


Fig 3.13: Adição de Vetores Esparsos

Alternativa 3.4(a) *Adição de Vetores Esparsos via Vetor de Trabalho*

(*descompactação* do vetor u no vetor de trabalho "expandido" w)
para j **de** iu **até** iuf **faça**
$\quad$ **W[JU**$[j]] \leftarrow$ **U**$[j]$ (a)

(processamento de $w \leftarrow w + v$ de forma "descompactada" em w)
para j **de** iv **até** ivf **faça**
$\quad$ **W[JV**$[j]] \leftarrow$ **W[JV**$[j]] +$ **V**$[j]$ (b)

(*compactação* do resultado em w seqüencialmente em u)
para j **de** iu **até** iuf **faça**
$\quad$ **U**$[j] \leftarrow$ **W[JU**$[j]]$ (c)

Alt 3.4(a): *Adição de Vetores Esparsos via Vetor de Trabalho*

Cancelamentos por linhas com Geração dos fatores por Linhas

Apresenta-se a seguir, no Procedimento **3.4(1)** (complementado esquematicamente na Figura 3.14), uma alternativa para a implementação numérica de eliminações $U^T D\, U$ (cancelando-se os elementos inferiores por linhas) com geração dos fatores triangulares superiores por *Linhas*, análoga a Alternativa **2.4(b)**.

Além das estruturas de armazenamento seqüencial segmentado para a matriz de fatores, e do vetor de trabalho expandido **W**, lança-se mão de um vetor auxiliar **IUP** de apontadores, indicando a cada etapa, a posição correntemente normalizada na representação de U.

A normalização deste modo, processa-se paulatinamente por colunas, e a cada etapa i, apenas os elementos $u_{l_1,i}$, $u_{l_2,i}$, ..., $u_{l_f,i}$ (da coluna base) apontados por **IUP**$[l_1]$, **IUP**$[l_2]$, ..., **IUP**$[l_f]$ para $1 \leq l_1, l_2, ..., l_f < i$ são efetivamente normalizados.

Descrição de Passos do Procedimento **3.4(1)**

A cada etapa básica i do procedimento **3.4(1)**, efetuam-se as seguintes operações:

No passo (a) armazena-se inicialmente em *piv* o valor original correspondente do elemento pivô $a_{i,i}$, a ser atualizado durante o processo de contribuições. (A opção pela acumulação temporária de valores em *piv* ao invés de acessar-se diretamente as posições em **DI** correspondentes, tem como benefício o fato de se poder utilizar um armazenamento temporário em precisão dupla ou quádrupla, especialmente nos casos onde os coeficientes **DI** encontram-se armazenados com precisão inferior).

No passo (b) o apontador **IUP**$[i]$ é inicializado para a posição correspondente ao primeiro elemento não nulo u_{i,j_1} da linha base (excluindo-se a diagonal). Este apontador não será necessário na etapa básica corrente, e é apenas inicializado de modo

Procedimento 3.4(1) *Eliminação via Vetor de Trabalho* (*Linhas*)

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

(*inicialização* do elemento *diagonal*)
$piv \leftarrow \mathbf{DI}[i]$ (a)

(inicialização do *apontador dinamico* de *posições iniciais*)
$\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$ (b)

(*expansão* da *linha base* no *vetor de trabalho*)
para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça**
$\mathbf{W}[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$ (c)

(para cada linha l a ser subtraida da *linha base*)
para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça** (d)

(indice da atual *linha contribuinte*)
$l \leftarrow \mathbf{JUT}[k]$ (e)

(*posição inicial* em l)
$iupl \leftarrow \mathbf{IUP}[l]$ (f)

(*proxima* posição inicial)
$\mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1$ (g)

(*fator multiplicativo*)
$um \leftarrow \mathbf{UN}[iupl] * \mathbf{DI}[l]$ (h)

(atualização da *diagonal*)
$piv \leftarrow piv - \mathbf{UN}[iupl] * um$ (i)

(*normalização*)
$\mathbf{UN}[iupl] \leftarrow um$ (j)

(acumulação das contribuições da linha l sobre a *linha base*)

---

para j **de** $iupl + 1$ **até** $\mathbf{IUF}[l]$ **faça** (k)
$\mathbf{W}[\mathbf{JU}[j]] \leftarrow \mathbf{W}[\mathbf{JU}[j]] - \mathbf{UN}[j] * um$ (l)

---

fim para k

(*inversão* do elemento *diagonal*)
$\mathbf{DI}[i] \leftarrow 1. \; / \; piv$ (m)

(*compactação* dos valores definitivos da *linha base*)
para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{W}[\mathbf{JU}[j]]$ (n)

fim para i

Proc 3.4(1): *Eliminação via Vetor de Trabalho* (*Linhas*)

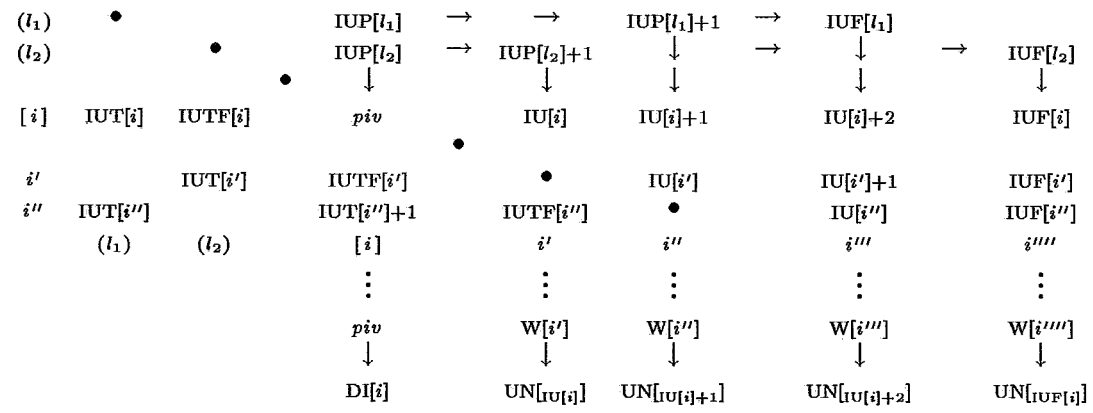


Fig 3.14: Eliminação $U^T D\, U$ gerando fatores por Linhas

a que quando futuramente acessado (e incrementado) em etapas j_1, j_2, ..., venha a apontar para os elementos u_{i,j_1}, u_{i,j_2}, ... a serem respectivamente normalizados.

No passo (c) expande-se os elementos não nulos da linha base i, no vetor de trabalho **W**.

Como opção em alguns casos (objetivando melhorar a qualidade numérica dos resultados), pode-se adotar para **W**, uma representação numérica em precisão superior a adotada para **UN**. (O preço por tal opção contudo, pode vir a ser pago num maior tempo de execução, decorrente da operação em ponto flutuante com dados temporariamente acumulados em precisão mais elevada).

O passo (d) corresponde ao *loop* intermediário, em que se irão processar as contribuições sobre a linha base, de cada uma das linhas especificadas na representação transposta (correspondendo particularmente aos índices dos elementos não nulos a esquerda da diagonal corrente, a serem cancelados).

No passo (e) armazena-se em l o índice da atual linha contribuinte a ser subtraída da linha base corrente.

No passo (f) armazena-se em *iupl* a posição (correntemente apontada por **IUP**$[l]$) do elemento $u_{l,i}$, a ser normalizado (na coluna base) e utilizado como fator multiplicativo no processo de contribuição dos elementos da linha l sobre a linha base.

No passo (g) o apontador auxiliar **IUP** é dinâmicamente incrementado, apontando para a posição u_{l,i_l} correspondente ao próximo elemento não nulo da linha l (a ser futuramente normalizado na etapa i_l).

No passo (h) armazena-se em *um* o fator multiplicativo correspondente a $u_{l,i} \;/\; d_{l,l}$.

No passo (i) processa-se a contribuição correspondente a $u_{i,i} \leftarrow u_{i,i} - u_{l,i} * (u_{l,i} \;/\; d_{l,l})$ (atualizando-se o elemento pivô da linha base corrente).

No passo (j) o valor normalizado $(u_{l,i} \;/\; d_{l,l})$ é armazenado definitivamente na posição (da coluna base) correspondente a $u_{l,i}$.

O passo (k) corresponde ao *loop* mais interno do processo de eliminação, e onde efetivamente se processam as contribuições de cada um dos elementos não nulos da

linha l sobre a linha base. Exclui-se particularmente a contribuição sobre a diagonal, processada em separado no passo (i), e por esta razão o *loop* é iniciado a partir da posição apontada por $iupl + 1$.

O passo (l) implementa (mediante o uso do vetor expandido **W**) as contribuições da forma $u_{i,\mathrm{JU}[j]} \leftarrow u_{i,\mathrm{JU}[j]} - u_{l,\mathrm{JU}[j]} * (u_{l,i} \ / \ d_{l,l})$.

No passo (m) armazena-se definitivamente em **DI**$[i]$ o inverso do fator diagonal $d_{i,i}$ acumulado em *piv*. Nos casos em que se opte por uma maior precisão nos resultados, o vetor **DI** pode ser representado em precisão superior a utilizada para o armazenamento em **UN** (visto introduzir um *overhead* proporcional à apenas n posições adicionais de memória). Em todos os casos, a acumulação temporária dos valores mediante a variável escalar *piv* pode ser sempre efetuada em precisão superior, o que já garante por si só uma maior qualidade numérica nos resultados (sem onerar-se significativamente o espaço adicional de armazenamento).

Finalmente no passo (n), compacta-se de volta (na representação **UN** da linha corrente), os valores correspondentemente operados de forma expandida no vetor de trabalho **W**.

□

Cancelamentos por linhas com Geração dos fatores por Colunas

Apresenta-se agora no Procedimento **3.4(2)** (complementado esquematicamente na Figura 3.15), um processo de eliminação $U^T D\, U$ com geração dos fatores U por *Colunas*, análogo a Alternativa **2.4(c)**.

As estruturas de dados utilizadas são as mesmas adotadas no procedimento anterior, e apenas os principais pontos de distinção serão comentados mais detalhadamente a seguir.

O processo de geração por *colunas* corresponde a um processo de atualizações "retardadas", como comentado na seção 2.4, e assim opera-se na verdade com os valores situados a esquerda da diagonal na linha básica corrente (e que em função da simetria também podem ser lançados nas posições supra-diagonais correspondentes na coluna base sendo gerada a cada etapa).

Por esta razão, uma das principais distinções entre os dois processos de eliminação considerados, é o fato de na geração por *colunas*, os acessos aos elementos das linhas l contribuintes sobre a linha base, serem processados a partir de **IU**$[l]$ até **IUPF**$[l] - 1$, com o vetor auxiliar **IUPF** (análogo a **IUP**) contendo neste caso, apontadores para as posições finais de acesso (dinâmicamente incrementados a cada etapa).

Outra distinção importante é que no processo de geração por colunas, o vetor de trabalho expandido **W** é utilizado apenas para a acumulação das contribuições sofridas a cada etapa, sem requerer deste modo a descompactação inicial dos valores originalmente armazenados na linha ou coluna básica associada.

Neste caso, a atualização dos valores definitivos incorporando-se as contribuições sofridas, é efetuada paulatinamente no *loop* intermediário, por ocasião da norma-

lização do fator associado. Este fato, confere ao procedimento **3.4(2)** menores *overheads* de processamento, visto não requerer explicitamente as fases de descompactação inicial e re-compactação final, presentes no procedimento anterior.

No caso da geração dos fatores por *colunas*, faz-se imperativo ao se iniciar o processamento, que todas as posições de memória do vetor de contribuições expandido **W** tenham sido préviamente zeradas. Tal ocorre, pois neste procedimento não se efetua uma descompactação dos valores originalmente armazenados na linha base corrente, como no procedimento anterior (onde em função da estrutura de dados fornecida para a fase numérica, as posições relativas aos *fill-in's* encontram-se inicialmente zeradas).

Descrição de Passos do Procedimento **3.4(2)**

Descreve-se a seguir apenas os passos distintos no Procedimento **3.4(2)**, com relação ao procedimento anteriormente apresentado.

No passo (a) anulam-se inicialmente todas as posições de **W**.

No passo (b) armazena-se em $iupfl$ a posição do atual elemento $u_{l,i}$ (na coluna base) a sofrer a contribuição retardada de **W**$[l]$ e que será utilizado como fator multiplicativo e posteriormente normalizado.

No passo (c) incrementa-se **IUPF**$[l]$, com a finalidade de apontar-se para a posição do próximo elemento não nulo $u_{l,i'}$ (da linha l) a ser posteriormente normalizado na etapa i'.

No passo (d) as contribuições acumuladas no vetor de trabalho **W** são incorporadas ao fator multiplicativo um.

No passo (e) a posição **W**$[l]$ (recém utilizada do vetor de trabalho) é zerada, de modo a permitir o acumulo futuro de novas contribuições.

No passo (f) processa-se a normalização do fator $u_{l,i}$, lançando-se o resultado na posição associada da coluna base (apontada por $iupfl$).

No passo (g) o processamento das contribuições é restrito apenas as posições compreendidas entre **IU**$[l]$ e $iupfl - 1$, em virtude de apenas os elementos das linhas contribuintes situados a esquerda da diagonal básica corrente, serem necessários para a geração (retardada) dos fatores triangulares superiores da coluna base.

Finalmente, no passo (h) processa-se o acumulo temporário das contribuições nas posições correspondentes do vetor de trabalho **W**. Como o processo de geração por colunas é um processo de atualizações retardadas, tais contribuições só serão subtraídas dos fatores associados, por ocasião de instâncias futuras do passo (c), quando os valores temporariamente acumulados em **W** se mostrarem necessários para a geração definitiva de novos fatores.

□

Substituições de Variáveis

Uma vez obtida a fatoração numérica, conclui-se o processo de solução para

Procedimento 3.4(2) *Eliminação via Vetor de Trabalho* (*Colunas*)

("zeragem" inicial do *vetor de trabalho*)
para i **de** 1 **até** n **faça**
    $\mathbf{W}[i] \leftarrow 0.$ (a)

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

    (*inicialização* do elemento *diagonal*)
    $piv \leftarrow \mathbf{DI}[i]$

    (inicialização do *apontador dinamico* de *posições finais*)
    $\mathbf{IUPF}[i] \leftarrow \mathbf{IU}[i]$

    (para cada linha l a ser subtraida da *linha base*)
    para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça**

        (indice da atual *linha contribuinte*)
        $l \leftarrow \mathbf{JUT}[k]$

        (*posição final* em l)
        $iupfl \leftarrow \mathbf{IUPF}[l]$ (b)

        (*proxima* posição final)
        $\mathbf{IUPF}[l] \leftarrow \mathbf{IUPF}[l] + 1$ (c)

        (*fator multiplicativo*)
        $um \leftarrow \mathbf{UN}[iupfl] - \mathbf{W}[l]$ (d)

        ("zeragem" do *vetor de trabalho*)
        $\mathbf{W}[l] \leftarrow 0.$ (e)

        (*normalização*)
        $\mathbf{UN}[iupfl] \leftarrow um * \mathbf{DI}[l]$ (f)

        (atualização da *diagonal*)
        $piv \leftarrow piv - \mathbf{UN}[iupfl] * um$

        (acumulação das contribuições da linha l sobre a *coluna base*)

        ---

        para j **de** $\mathbf{IU}[l]$ **até** $iupfl - 1$ **faça** (g)
            $\mathbf{W}[\mathbf{JU}[j]] \leftarrow \mathbf{W}[\mathbf{JU}[j]] + \mathbf{UN}[j] * um$ (h)

        ---

    fim para k

    (*inversão* do elemento *diagonal*)
    $\mathbf{DI}[i] \leftarrow 1. \; / \; piv$
fim para i

Proc 3.4(2): *Eliminação via Vetor de Trabalho* (*Colunas*)

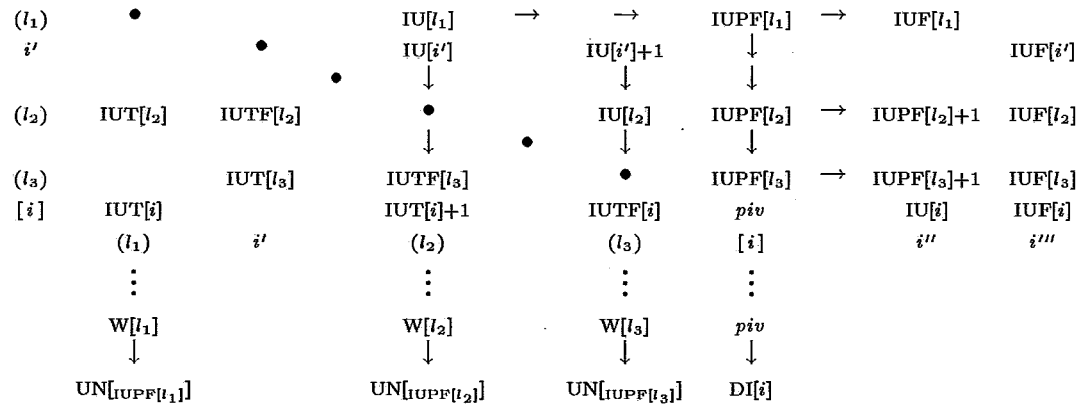


Fig 3.15: Eliminação $U^T D\, U$ gerando fatores por Colunas

o sistema linear esparso original, através das fases de *substituições de variáveis* (*Forward*, *Diagonal* e *Backward*) definidas em (2.3), (2.4) e (2.5). Tais fases serão apresentadas unificadamente no Procedimento **3.4(3)** em termos do vetor solução x (como notado na seção 2.4), de forma análoga ao Procedimento **2.4(4)** (para o caso denso).

Procedimento 3.4(3) *Substituições de Variáveis*

(*Forward e Diagonal*)

para i **de** 1 **até** n **faça**
    $\mathbf{X}[i] \leftarrow \mathbf{B}[i]$ (a)

para i **de** 1 **até** n **faça**
    $xi \leftarrow \mathbf{X}[i]$
    para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça**
        $\mathbf{X}[\mathbf{JU}[j]] \leftarrow \mathbf{X}[\mathbf{JU}[j]] - \mathbf{UN}[j] * xi$ (b)
    $\mathbf{X}[i] \leftarrow xi * \mathbf{DI}[i]$ (c)

(*Backward*)

para i **de** $n-1$ **de volta até** 1 **faça**
    $xi \leftarrow \mathbf{X}[i]$
    para j **de** $\mathbf{IUF}[i]$ **de volta até** $\mathbf{IU}[i]$ **faça**
        $xi \leftarrow xi - \mathbf{UN}[j] * \mathbf{X}[\mathbf{JU}[j]]$ (d)
    $\mathbf{X}[i] \leftarrow xi$

Proc 3.4(3): *Substituições de Variáveis*

Descrição de Passos do Procedimento **3.4(3)**

No passo (a) inicializa-se o vetor solução x com as componentes do vetor lado direito b.

No passo (b) processa-se a atualização *forward* das componentes de x, subtraindo-se as parcelas $u_{i,\mathrm{JU}[j]} * x_i$ das componentes $x_{\mathrm{JU}[j]}$ associadas. Em face a simetria, este processo corresponde a subtração de múltiplos escalares da componente x_i (ditados pelos coeficientes não nulos da i'ezima coluna de U^T) de todas as componentes $x_{\mathrm{JU}[j]}$ afetadas na etapa corrente. Ao se chegar a i'ezima etapa do processo, a atualização *forward* da componente x_i, (por construção) já terá sido concluída (em alguma etapa prévia), permitindo que o valor correntemente obtido venha a contribuir no processo de atualização de componentes subseqüentes.

No passo (c) processa-se a fase de substituição diagonal, normalizando-se a componente x_i.

No passo (d) processa-se a fase de substituição *backward,* subtraindo-se de x_i múltiplos escalares (ditados pelos coeficientes não nulos $u_{i,\mathrm{JU}[j]}$ da i'ezima linha de U) das variáveis $x_{\mathrm{JU}[j]}$ anteriormente geradas.

□

Geração por linhas Agregada a substituições Forward e Diagonal

A solução de sistemas, nos casos em que se requer apenas uma única solução para uma dada matriz de coeficientes e um dado vetor lado direito associado, pode ser simplificada, agregando-se num mesmo processo as fases de:

- eliminação por *linhas*
- substituições *forward*
- normalização *diagonal*

como se apresentará no Procedimento **3.4(4)**.

A base para tal se origina no fato do vetor lado direito poder ser encarado para fins práticos, como uma coluna $n+1$ na representação do sistema sendo triangularizado (num esquema similar ao da representação *compacta* apresentada na Figura 2.4).

Assim, as fases de substituição *forward* e *diagonal* podem ser implementadas concomitantemente com a eliminação, simplesmente aplicando-se sobre o vetor x, as mesmas operações de combinação linear efetuadas para o processamento das contribuições incidentes sobre a linha básica.

Neste caso, no primeiro *loop* principal do processo, efetuam-se tanto as eliminações triangulares, como as atualizações e normalizações correspondentes no vetor lado direito. Uma vez concluída esta fase de eliminações e substituições *forward* e *diagonal*, passa-se finalmente a uma fase de substituição *backward* idêntica a apresentada no Procedimento **3.4(3)**.

Procedimento 3.4(4) *Eliminação e Substituição (Linhas)*

para i **de** 1 **até** n **faça**
$\mathbf{W}[i] \leftarrow 0.$ (a)

(*Eliminação e Substituições Forward e Diagonal*)
para i **de** 1 **até** n **faça**

(*inicialização* do elemento *diagonal*)
$piv \leftarrow \mathbf{DI}[i]$

(inicialização da i'ezima componente de x)
$xi \leftarrow \mathbf{B}[i]$ (b)

(inicialização do *apontador dinamico* de *posições iniciais*)
$\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$

(para cada linha l a ser subtraida da *linha base*)
para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça**

(indice da atual *linha contribuinte*)
$l \leftarrow \mathbf{JUT}[k]$

(*posição inicial* em l)
$iupl \leftarrow \mathbf{IUP}[l]$

(*proxima* posição inicial)
$\mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1$

(*fator multiplicativo*)
$um \leftarrow \mathbf{UN}[iupl] * \mathbf{DI}[l]$

(atualização da *diagonal*)
$piv \leftarrow piv - \mathbf{UN}[iupl] * um$

(*normalização*)
$\mathbf{UN}[iupl] \leftarrow um$

(atualização *forward* da i'ezima componente de x)
$xi \leftarrow xi - \mathbf{X}[l] * um$ (c)

(acumulação das contribuições da linha l sobre a *linha base*)
para j **de** $\mathbf{IUP}[l]$ **até** $\mathbf{IUF}[l]$ **faça**
$\mathbf{W}[\mathbf{JU}[j]] \leftarrow \mathbf{W}[\mathbf{JU}[j]] + \mathbf{UN}[j] * um$ (d)

(*inversão* do elemento *diagonal*)
$\mathbf{DI}[i] \leftarrow 1. \ / \ piv$

(*normalização* diagonal da i'ezima componente de x)
$\mathbf{X}[i] \leftarrow xi * \mathbf{DI}[i]$ (e)

(*compactação* definitiva da *linha base*)
para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - \mathbf{W}[\mathbf{JU}[j]], \mathbf{W}[\mathbf{JU}[j]] \leftarrow 0.$ (f)

(*Substituição Backward*)
para i **de** $n-1$ **de volta até** 1 **faça**
para j **de** $\mathbf{IUF}[i]$ **de volta até** $\mathbf{IU}[i]$ **faça**
$\mathbf{X}[i] \leftarrow \mathbf{X}[i] - \mathbf{UN}[j] * \mathbf{X}[\mathbf{JU}[j]]$ (g)

Proc 3.4(4): *Eliminação e Substituição (Linhas)*

No procedimento de Eliminação, introduziu-se a título de complementação uma pequena variação no modo de operação sobre o vetor de trabalho expandido **W**.

Ao invés de ser utilizado como um *vetor de trabalho* (para eliminações por linhas tradicionais), adotou-se para o mesmo a uma estratégia similar a empregada nos métodos de geração por colunas, ou seja na forma de um *vetor de contribuições.* Desta forma inicialmente assume-se que **W** esteja zerado em todas as suas posições, para durante o *loop* mais interno, processar-se o acumulo das contribuições a serem subtraídas da linha base ao final do processo.

A vantagem com esta forma de operação sobre **W** é que apenas um *loop* de compactação definitiva de valores mostra-se necessário, em comparação com os procedimentos usuais, onde os elementos da linha base são originalmente descompactados no *vetor de trabalho* antes do processamento das contribuições.

Esta variante mostra-se assim mais eficiente, por possuir um volume de *overhead* menor, e por permitir que os acessos expandidos ao vetor **W** sejam explorados tanto para compactação definitiva dos valores sobre a linha base, como para o imediato cancelamento de cada posição em **W** recém utilizada.

Descrição de Passos do Procedimento **3.4(4)**

Uma descrição dos novos passos incorporados ao Procedimento **3.4(4)** visando a geração dos fatores por *linhas* agregada as fases de substituições *forward* e *diagonal*, é apresentada a seguir:

No passo (a) inicializam-se todas as posições do vetor de contribuições **W**.

No passo (b) inicializa-se x_i com a componente do vetor lado direito associada.

No passo (c) processa-se a atualização *forward* sobre a componente x_i. Este passo difere fundamentalmente do passo equivalente no Procedimento **3.4(3)**, uma vez que embora a geração dos fatores U seja efetuada por linhas, sua normalização processa-se por colunas, permitindo que a substituição *forward* seja efetuada tomando por base os coeficientes definitivamente gerados na linha corrente de U^T, subtraindo-se de x_i múltiplos escalares destes valores (multiplicados pelas componentes x_l anteriormente já determinadas).

No passo (d) processa-se o acumulo das contribuições sobre **W** (a serem posteriormente subtraídas da linha base, no passo (f)).

No passo (e) processa-se a normalização *diagonal* da componente x_i correntemente gerada.

No passo (f) processa-se a compactação definitiva das contribuições acumuladas em **W**, subtraindo-as das posições correspondentes na linha base, e anulando-se a posição recém acessada de **W**, de modo a manter este vetor completamente zerado para a próxima etapa.

Finalmente, no passo (g) processa-se separadamente a fase de substituição *backward,* subtraindo-se de x_i múltiplos escalares correspondentes das componentes de x anteriormente determinadas.

□

Seqüência completa de Fases envolvidas na resolução de sistemas

Ao se concluir este capítulo, mostra-se conveniente indicar a seqüência completa de fases envolvidas na resolução de sistemas lineares esparsos, apresentadas nas seções anteriores.

Inicialmente processam-se fases de pré-processamento, visando preparar-se a estrutura de dados original da forma mais adequada para as subseqüentes fases de resolução numérica.

Isto permite agrupar-se as várias etapas em duas categorias distintas:

Pré-Processamento Estrutural

Executadas apenas uma vez (para matrizes com mesma estrutura)

- *Ordenamento* das equações visando a *redução* de *fill-in's*
- *Permutação física* da estrutura original (segundo o ordenamento obtido)
- *Fatoração Simbólica* (determinando a estrutura de fatores resultante)
- *Ordenação* (crescente) da estrutura de *índices de colunas* a cada linha

Processamento Numérico

Executadas repetidas vezes (para matrizes numericamente distintas)

- *Fatoração Numérica* (conduzindo o sistema original à forma triangular)
- *Substituições de Variáveis* (através das fases *forward*, *diagonal* e *backward*)

A partir dos próximos capítulos, a atenção estará concentrada na apresentação de técnicas adicionais de *pré-processamento*, visando aumentar-se especificamente o desempenho da fase *numérica* de fatoração.

Capítulo 4

Alternativas para o Aprimoramento da Fase Numérica

Neste capítulo apresentam-se aprimoramentos para a fase numérica do processo de eliminações, baseados no aproveitamento de características estruturais e informações de natureza simbólica não exploradas nos procedimentos convencionais até então considerados.

Conceitos consagrados, como os de *supernode* e estruturas auxiliares como a "árvore de caminhos de eliminação" também serão objeto de atenção deste capítulo, por fornecerem o ferramental básico para novos aprimoramentos.

Inicialmente serão considerados procedimentos de eliminação, baseados na utilização de "listas simbólicas de endereços", objetivando especificamente a eliminação de *overheads* oriundos do uso de estruturas intermediárias de armazenamento (como os vetores de trabalho expandidos).

Posteriormente serão apresentados os conceitos de *supernode* e o de árvore de caminhos de eliminação, incorporando-se a exploração destas informações em processos de fatoração numérica aprimorados.

Finalmente será introduzida a noção de "janelas" de eliminação, viabilizando a construção de metodologias híbridas de fatoração direta, combinando técnicas convencionais (como as apresentadas no capítulo anterior), com o uso de listas simbólicas de endereços e de técnicas voltadas para o processamento de classes especiais de submatrizes (como as do tipo *supernodal* e *denso* entre outras).

4.1 Uso de Listas Simbólicas de Endereços

Uma forma de se viabilizar a redução de *overheads* (no tempo de execução) da fase numérica do processo de eliminação esparsa, é a utilização de estruturas de dados auxiliares, como por exemplo a de Listas Simbólicas de Endereços (a serem conceituadas um pouco mais adiante), no mesmo estilo das aplicadas originalmente com sucesso na resolução de problemas de Programação Linear via Algoritmos de Pontos Interiores por Mauricio Resende, Geraldo Veiga, Ilan Adler e Narendra Karmarkar

[1].

Mostra-se conveniente no entanto, apresentar-se inicialmente um breve histórico da evolução das técnicas baseadas na exploração de informações de natureza simbólica, visando o aprimoramento específico da fase *numérica* do processo de fatoração.

Breve Histórico das Abordagens Simbólicas para a Fase Numérica

As abordagens para a fase numérica centradas na exploração das informações de cunho simbólico, como as propostas em [1], [32], [20], [5] e [6] podem ser consideradas como alternativas "computacionalmente mais econômicas" para a implementação do esquema proposto originalmente por Gustavson em [60], no início da década de 70.

A idéia original de Gustavson foi a de lançar mão de sub-programas *"loop-free"* análogos ao da Codificação **4.1-1** (correspondente ao exemplo da Figura 4.1) contendo explicitamente todas as operações aritméticas necessárias para a fatoração específica da matriz de cada sistema fornecida como entrada.

Mediante recursos de pré-processamento, tal código era automaticamente gerado em alguma linguagem computacional como FORTRAN (ou diretamente em Assembler), consistindo basicamente apenas nas linhas dedicadas à fatoração numérica para cada caso particularmente considerado.

$$\begin{pmatrix} DI(1) & o & UN(1) & o \\ o & DI(2) & UN(2) & UN(3) \\ UN(1) & UN(2) & DI(3) & UN(4) \\ o & UN(3) & UN(4) & DI(4) \end{pmatrix}$$

Fig 4.1: Matriz a ser fatorada por código *Loop-Free*

O único sério entrave na abordagem de Gustavson (além do tempo adicional gasto nas fases de geração e *linkedição* dos sub-programas à aplicação principal) é o da dimensão do código gerado, que por ser proporcional ao número de operações aritméticas efetuadas, torna-o proibitivo para problemas de médio a grande porte, mesmo nas arquiteturas atuais (e acabou por invalidar a idéia original na época).

Assim, alternativas menos onerosas em termos de recursos, mas visando alcançar o mesmo objetivo de redução de *overheads* passaram a ser consideradas, abrindo-se mão da utilização de trechos de código completamente "desenrolados" no estilo *"loop-free"*, e adotando-se em seu lugar estruturas de dados auxiliares, como as de listas contendo todos os endereços de memória a serem efetivamente acessados durante o processo de fatoração.

Tais listas receberam a denominação de "Listas Simbólicas de Endereços" por basearem-se nas localizações derivadas a partir da estrutura da Matriz de Fatores Resultante (obtida preliminarmente em alguma fase, como a de Fatoração Simbólica, apresentada na seção 3.3).

Como se mostrará no capítulo 5, a noção de Listas Simbólicas de Endereços pode ser estendida, e ao invés do simples armazenamento das posições de memória a serem referenciadas durante a fatoração, pode-se lançar mão de estruturas baseadas

Codificação 4.1-1 *Código Loop-Free no estilo proposto por Gustavson*

```
SUBROUTINE LFFACT (UN, DI)
DOUBLE PRECISION UN(*), DI(*)
DI(3) = DI(3) - UN(1) * UN(1) / DI(1)
UN(1) = UN(1) / DI(1)
DI(1) = 1. / DI(1)
DI(3) = DI(3) - UN(2) * UN(2) / DI(2)
UN(4) = UN(4) - UN(2) * UN(3) / DI(2)
DI(4) = DI(4) - UN(3) * UN(3) / DI(2)
UN(2) = UN(2) / DI(2)
UN(3) = UN(3) / DI(2)
DI(2) = 1. / DI(2)
DI(4) = DI(4) - UN(4) * UN(4) / DI(3)
UN(4) = UN(4) / DI(3)
DI(3) = 1. / DI(3)
DI(4) = 1. / DI(4)
END
```

Cod 4.1-1: *Código Loop-Free no estilo proposto por Gustavson*

em listas de "Códigos" (capazes de implicitamente expressar seqüências completas de operações a serem efetuadas nas sucessivas etapas da eliminação triangular).

Serão nestas estruturas que se basearão os novos procedimentos propostos neste trabalho, por possuírem como principal característica, um desempenho (em tempo de execução) similar aos obtidos por procedimentos completamente *loop-free*, viabilizando no entanto a preservação dos *overheads* de armazenamento a níveis proporcionais apenas ao volume de dados (e não ao de operações aritméticas, como nos demais procedimentos simbólicos até então considerados pela literatura).

Conceituação e Geração das Listas Simbólicas de Endereços

A informação adicional explorada nos procedimentos numéricos aprimorados pelo uso de Listas Simbólicas, consiste numa enumeração da seqüência de todos os endereços de memória, dos elementos acessados na linha (ou coluna) Base sendo gerada, em cada etapa do processo de eliminação.

Tal informação permite dispensar-se o uso de estruturas auxiliares como a de vetores de trabalho expandidos, reduzindo-se assim um volume desnecessário de acessos e transferências temporárias de dados em memória durante a fase numérica.

As posições de memória sobre as linhas Contribuintes não necessitam ser armazenadas, uma vez que em ambos os procedimentos de eliminação por linhas (com geração dos fatores por Linhas ou por Colunas), o acesso aos elementos contribuintes, se dá de forma seqüencial, em virtude do armazenamento contíguo dos elementos não nulos por cada linha na matriz de fatores.

Conceitua-se a seguir a noção de Listas Simbólicas de Endereços, efetuando-se a distinção para o caso da geração de fatores por Linhas e para o da geração por Colunas.

Definição 4.1.1 (Lista Simbólica por Linhas) *Define-se por Listas Simbólicas de Endereços para geração de fatores por Linhas, a seqüência de todas as posições de memória dos elementos acessados na Linha base a cada etapa do processo de eliminação (durante as operações de combinação linear com elementos de linhas contribuintes anteriores, visando sua geração).*

Definição 4.1.2 (Lista Simbólica por Colunas) *Define-se por Listas Simbólicas de Endereços para geração de fatores por Colunas, a seqüência de todas as posições de memória dos elementos acessados na Coluna base a cada etapa do processo de eliminação (durante as operações de combinação linear com elementos de linhas contribuintes anteriores, visando sua geração).*

Mediante a adoção de Listas de Endereços, como ilustrado na Alternativa **4.1(a)**, o processo de adição de vetores esparsos seqüencialmente armazenados, originalmente apresentado na Alternativa **3.4(a)**, pode ser agora formulado a partir da utilização de uma lista de apontadores **PU[.]** para o acesso as posições no vetor base U afetadas pela contribuição dos elementos correspondentes de V armazenados desde iv até ivf.

Alternativa 4.1(a) *Adição de Vetores Esparsos via Listas de Endereços*

(posição anterior a inicial na *lista de endereços*)
$ipu \leftarrow 0$

(processamento de $u \leftarrow u + v$ *diretamente* sobre as posições de u)
para j **de** iv **até** ivf **faça**
    $ipu \leftarrow ipu + 1$
    $\mathbf{U}[\mathbf{PU}[ipu]] \leftarrow \mathbf{U}[\mathbf{PU}[ipu]] + \mathbf{V}[j]$

Alt 4.1(a): *Adição de Vetores Esparsos via Listas de Endereços*

Para a adoção de esquemas baseados em Listas Simbólicas de Endereços mostra-se necessário inicialmente a determinação prévia de todas as posições de memória a serem acessadas na fase numérica.

Apresenta-se, inicialmente, no Procedimento **4.1(1)** um esquema voltado para geração de fatores por Linhas, determinando a Lista Simbólica de Endereços, a ser posteriormente utilizada no Procedimento **4.1(3)** durante sucessivas fatorações numéricas de matrizes com mesma estrutura.

Para a determinação da lista de endereços, pode-se tomar como base o Procedimento **3.4(1)** voltado originalmente para a fatoração numérica, sem que se efetue no entanto qualquer operação aritmética em ponto flutuante (dispensando-se deste modo a utilização do vetor de trabalho expandido **W**).

Em contrapartida, lança-se mão de um vetor auxiliar **IW** (de dimensão n) de modo a conter temporariamente a localização de todos os fatores não nulos situados sobre a linha base a cada etapa.

Além do novo vetor auxiliar, lança-se mão de uma variável escalar *iplin* indicando a posição atual na lista simbólica de endereços, a serem armazenados no vetor **PLIN**.

Procedimento 4.1(1) *Listas de Endereços* (*Linhas*)

(posição anterior a inicial na *lista de endereços*)
$iplin \leftarrow 0$ (a)

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

(inicialização do *apontador dinamico* de *posições iniciais*)
$\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$

(montagem do *vetor auxiliar de endereços* relativos a *linha base*)
para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça**
$\mathbf{IW}[\mathbf{JU}[j]] \leftarrow j$ (b)

(para cada linha l a ser subtraida da *linha base*)
para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça**

(indice da atual *linha contribuinte*)
$l \leftarrow \mathbf{JUT}[k]$

(*posição inicial* em l)
$iupl \leftarrow \mathbf{IUP}[l]$

(*proxima* posição inicial)
$\mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1$

(geração da *lista de endereços* relativos a contribuição da linha l sobre a *linha base*)

---

para j **de** $iupl + 1$ **até** $\mathbf{IUF}[l]$ **faça**
$iplin \leftarrow iplin + 1$ (c)
$\mathbf{PLIN}[iplin] \leftarrow \mathbf{IW}[\mathbf{JU}[j]]$ (d)

---

fim para k
fim para i

Proc 4.1(1): *Listas de Endereços* (*Linhas*)

Descrição de Passos do Procedimento **4.1(1)**

No passo (a) do Procedimento **4.1(1)** inicializa-se o indicador *iplin* de posições na Lista Simbólica de Endereços.

No passo (b) o vetor auxiliar **IW** é inicializado, indicando as localizações de memória (em **UN**) onde se encontram armazenados cada um dos elementos não nulos da Linha Base.

No passo (c) *iplin* é atualizado de modo a indicar a posição de armazenamento do novo endereço a ser incorporado na Lista Simbólica.

No passo (d) a posição do elemento corrente sobre a Linha Base (préviamente armazenada em **IW**[**JU**[j]]), é incorporada à Lista Simbólica de Endereços.

□

O procedimento considerado simplesmente "simula" todas as etapas do processo de fatoração, e em lugar de efetuar as operações de ponto flutuante sobre os coeficientes da matriz, registra as respectivas posições de memória onde se encontram armazenados os elementos da Linha Base a serem posteriormente acessados na fase numérica.

Em vista do que foi apresentado na seção 3.2, é possível cogitar-se uma simplificação do esforço computacional na geração da lista simbólica de endereços, em casos particulares como os de sub-matrizes completamente densas (tipicamente encontradas ao final do processo de eliminação). Estes e outros casos no entanto, e que permitem uma sensível "compactação" do tamanho da lista de endereços, serão considerados particularmente na seção 4.2, quando se formalizar a noção de *supernode.*

Apresenta-se a seguir no Procedimento **4.1(2)**, uma metodologia análoga à anterior (derivada a partir do Procedimento **3.4(2)**) só que desta vez voltada para o caso de eliminações por linhas, com geração dos fatores por Colunas (objetivando determinar a lista de endereços a ser explorada posteriormente no Procedimento **4.1(4)**).

Neste caso, a utilização do vetor auxiliar **IW** mostra-se desnecessária, em função da particular localização dos elementos sendo gerados sobre a Coluna básica, e que já se encontram apontados pelo vetor dinâmico auxiliar **IUPF**.

No lugar de **PLIN**, lança-se mão do vetor análogo **PCOL** (de modo a conter a Lista Simbólica de Endereços para a geração dos fatores por Colunas).

Descrição de Passos do Procedimento **4.1(2)**

No passo (a) do Procedimento **4.1(2)**, inicializa-se o indicador *ipcol* de posições na Lista Simbólica de Endereços.

No passo (b) atualiza-se tal indicador (por ocasião da incorporação de um novo endereço a lista).

No passo (c) armazena-se em **PCOL** a localização (em **UN**) do elemento pertencente a Coluna Base, a ser correntemente atualizado durante a fase numérica.

□

Ao final de ambos os processos considerados, percebe-se que o tamanho das Listas Simbólicas de Endereços obtidas é o mesmo, tanto para a geração de fatores por Linhas, como para a geração por Colunas, pelo fato de espelharem o número de operações de ponto flutuante efetuadas sobre os elementos **UN** ao longo de todo o processo de eliminação.

A nível de desempenho computacional, estruturas baseadas em Listas de Endereços (como notado em [78]) podem ser implementadas mais eficientemente em

Procedimento 4.1(2) *Listas de Endereços (Colunas)*

(posição anterior a inicial na *lista de endereços*)
$ipcol \leftarrow 0$ (a)

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

(inicialização do *apontador dinamico* de *posições finais*)
IUPF$[i] \leftarrow$ **IU**$[i]$

(para cada linha l a ser subtraida da *linha base*)
para k **de IUT**$[i]$ **até IUTF**$[i]$ **faça**

(indice da atual *linha contribuinte*)
$l \leftarrow$ **JUT**$[k]$

(*posição final* em l)
$iupfl \leftarrow$ **IUPF**$[l]$

(*proxima* posição final)
IUPF$[l] \leftarrow$ **IUPF**$[l] + 1$

(geração da *lista de endereços* relativos a contribuição da linha l sobre a *coluna base*)

---

para j **de IU**$[l]$ **até** $iupfl - 1$ **faça**
$ipcol \leftarrow ipcol + 1$ (b)
PCOL$[ipcol] \leftarrow$ **IUPF**$[$**JU**$[j]]$ (c)

---

fim para k
fim para i

Proc 4.1(2): *Listas de Endereços (Colunas)*

linguagens como C/C^{++}, FORTRAN 90, ADA, PASCAL ou PL/I, onde facilidades de acesso direto a posições de memória (como "apontadores") encontram-se disponíveis.

Nas linguagens como FORTRAN 77 onde não se dispõe de tal recurso, a solução é lançar mão de estruturas auxiliares baseadas em vetores de Inteiros, adotando-se esquemas de indexação relativos a posição inicial do armazenamento de tais vetores, como forma de se determinar a localização dos dados a serem acessados (como por exemplo no caso do vetor **IW**, no Procedimento **4.1(1)**).

Fatorações Numéricas utilizando Listas Simbólicas de Endereços

Enumeram-se agora procedimentos para a fatoração Numérica, explorando as estruturas de Listas Simbólicas de Endereços (respectivamente consideradas na subseção anterior).

Inicialmente apresenta-se no Procedimento **4.1(3)** um processo de eliminação por linhas com geração dos fatores por Linhas (análogo ao Procedimento **3.4(1)**).

Descrição de Passos do Procedimento **4.1(3)**

No passo (a) deste procedimento, o indicador da posição corrente na Lista Simbólica de Endereços é inicializado.

No passo (b) seu valor é incrementado de modo a especificar a nova posição na lista de endereços, a ser utilizada para acesso ao atual elemento situado sobre a Linha Base.

No passo (c) o elemento sobre a Linha Base é atualizado, sofrendo a contribuição do elemento de mesma coluna, proveniente da linha contribuinte l.

□

Apresenta-se a seguir, no Procedimento **4.1(4)** um esquema complementar de fatoração numérica (análogo ao Procedimento **3.4(2)**), fundamentado em eliminações por linhas com geração dos fatores por Colunas.

As estruturas de dados utilizadas são as mesmas do caso anterior, apenas substituindo-se *iplin* e **PLIN** respectivamente por *ipcol* e **PCOL**.

Descrição de Passos do Procedimento **4.1(4)**

No passo (a) deste novo esquema, cabe ressaltar uma diferença com relação ao Procedimento **3.4(2)** (em que se processava uma subtração do valor correntemente acumulado no vetor de trabalho **W**, durante a atualização de *um*). Neste novo caso, a subtração é dispensável, por não se lançar mão de vetores auxiliares (processando-se as atualizações dos valores diretamente sobre a Coluna Base a cada contribuição).

No passo (b) a atualização do elemento na Coluna Base é efetuada diretamente sobre a posição de memória correntemente apontada por **PCOL**.

Procedimento 4.1(3) *Eliminação via de Listas de Endereços (Linhas)*

(posição anterior a inicial na *lista de endereços*)
$iplin \leftarrow 0$ (a)

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

(*inicialização* do elemento *diagonal*)
$piv \leftarrow \mathbf{DI}[i]$

(inicialização do *apontador dinamico* de *posições iniciais*)
$\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$

(para cada linha l a ser subtraida da *linha base*)
para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça**

(indice da atual *linha contribuinte*)
$l \leftarrow \mathbf{JUT}[k]$

(*posição inicial* em l)
$iupl \leftarrow \mathbf{IUP}[l]$

(*proxima* posição inicial)
$\mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1$

(*fator multiplicativo*)
$um \leftarrow \mathbf{UN}[iupl] * \mathbf{DI}[l]$

(atualização da *diagonal*)
$piv \leftarrow piv - \mathbf{UN}[iupl] * um$

(*normalização*)
$\mathbf{UN}[iupl] \leftarrow um$

(acumulação das contribuições da linha l sobre a *linha base*)

---

para j **de** $iupl + 1$ **até** $\mathbf{IUF}[l]$ **faça**
$iplin \leftarrow iplin + 1$ (b)
$\mathbf{UN}[\mathbf{PLIN}[iplin]] \leftarrow \mathbf{UN}[\mathbf{PLIN}[iplin]] - \mathbf{UN}[j] * um$ (c)

---

fim para k

(*inversão* do elemento *diagonal*)
$\mathbf{DI}[i] \leftarrow 1. \; / \; piv$
fim para i

Proc 4.1(3): *Eliminação via Listas de Endereços (Linhas)*

Procedimento 4.1(4) *Eliminação via Listas de Endereços (Colunas)*

(posição anterior a inicial na *lista de endereços*)
$ipcol \leftarrow 0$

(para cada *linha base* i a eliminar)
para i **de** 1 **até** n **faça**

(*inicialização* do elemento *diagonal*)
$piv \leftarrow \mathbf{DI}[i]$

(inicialização do *apontador dinamico* de *posições finais*)
$\mathbf{IUPF}[i] \leftarrow \mathbf{IU}[i]$

(para cada linha l a ser subtraida da *linha base*)
para k **de** $\mathbf{IUT}[i]$ **até** $\mathbf{IUTF}[i]$ **faça**

(indice da atual *linha contribuinte*)
$l \leftarrow \mathbf{JUT}[k]$

(*posição final* em l)
$iupfl \leftarrow \mathbf{IUPF}[l]$

(*proxima* posição final)
$\mathbf{IUPF}[l] \leftarrow \mathbf{IUPF}[l] + 1$

(*fator multiplicativo*)
$um \leftarrow \mathbf{UN}[iupfl]$ (a)

(*normalização*)
$\mathbf{UN}[iupfl] \leftarrow um * \mathbf{DI}[l]$

(atualização da *diagonal*)
$piv \leftarrow piv - \mathbf{UN}[iupfl] * um$

(acumulação das contribuições da linha l sobre a *coluna base*)

---

para j **de** $\mathbf{IU}[l]$ **até** $iupfl - 1$ **faça**
$ipcol \leftarrow ipcol + 1$
$\mathbf{UN}[\mathbf{PCOL}[ipcol]] \leftarrow \mathbf{UN}[\mathbf{PCOL}[ipcol]] - \mathbf{UN}[j] * um$ (b)

---

fim para k

(*inversão* do elemento *diagonal*)
$\mathbf{DI}[i] \leftarrow 1. \ / \ piv$

fim para i

Proc 4.1(4): *Eliminação via Listas de Endereços (Colunas)*

□

O desempenho de ambos os processos de fatoração numérica propostos nesta seção é muito próximo, em face a considerável similaridade de seus códigos (diferindo apenas na ordem com que os acessos e operações sobre os elementos Base vão sendo paulatinamente efetuados ao longo de cada uma das etapas da eliminação).

Convém notar no entanto, que sem alguma forma de "truncamento" ou compactação do tamanho final das listas de endereços, os métodos apresentados nesta seção, embora alcançando o objetivo de redução dos *overheads* em termos de tempo de execução (por efetuar os acessos a memória diretamente sobre os elementos da matriz de fatores), podem na verdade se tornar computacionalmente inviáveis em decorrência de *overheads* em espaço de armazenamento, oriundos da enumeração de todos os endereços de memória acessados (por sua proporcionalidade ao volume total de operações aritméticas efetuadas).

Este ponto foi levantado em trabalhos como [42], e soluções para se viabilizar a utilização destes e outros esquemas de cunho simbólico serão vistas na seção 4.2 e especialmente ao longo do capítulo 5.

4.2 Exploração da Estrutura de Supernodes

Como notado nas seções 3.1, 3.2 e 3.3, é comum observar-se padrões similares na estrutura de elementos não nulos de determinadas linhas de matrizes de fatores resultantes (especialmente nas linhas terminais de matrizes ordenadas por critérios como o de menor *fill-in*, ou menor *fill-in* local).

Esta seção tratará especificamente da exploração de tais estruturas, a serem conceituadas sob a designação de *supernodes.*

Como exemplo a ser adotado ao longo desta seção, tomemos uma matriz, representada (conjuntamente) por seus fatores U^T e U na Figura 4.2.

Especificamente neste caso, convencionou-se representar pelo símbolo "×" os elementos não nulos das linhas de U (e respectivas colunas de U^T) que não apresentam qualquer padrão estrutural similar comum entre si, em contrapartida aos elementos denotados por "□" e "•" onde a presença de similaridades entre suas respectivas linhas (e colunas) constituintes pode ser trivialmente notada.

Os elementos denotados por "•" (especificamente confinados à porções bloco-diagonais da matriz de fatores), por razões a serem vistas um pouco mais adiante, foram representados com seus índices correspondentes, particularmente destacados em negrito na figura em questão.

Caracterizações Conceituais para a noção de Supernodes

Apresenta-se a seguir algumas caracterizações estruturais para a noção de *supernode*, a ser posteriormente complementada nas definições 4.2.2 e 4.2.3 ao se introduzir o conceito de *Árvore de Eliminação.*

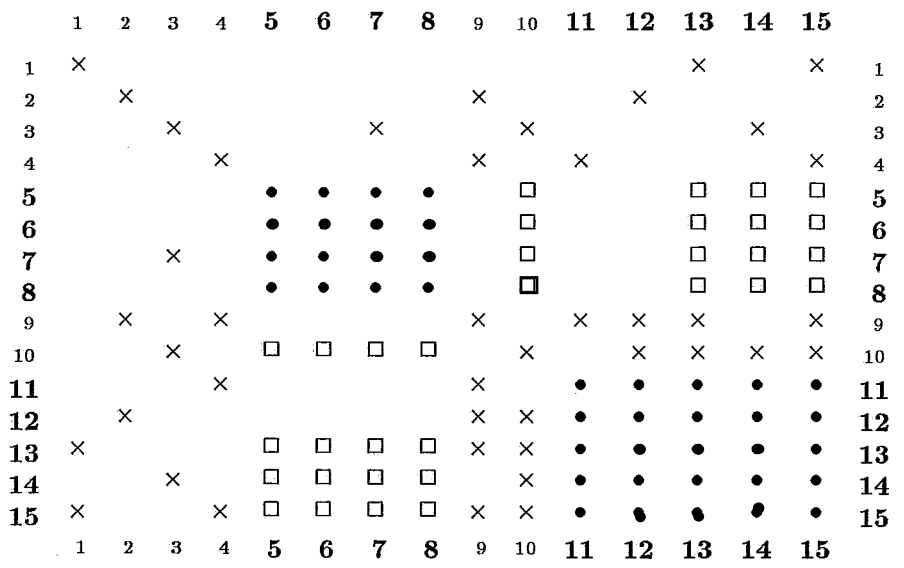


Fig 4.2: Particionamento por Supernodes

Definição 4.2.1 (Supernode Maximal) *Define-se por "Supernode Maximal" o maior conjunto de linhas contíguas* $i, i+1, ..., i+m$ *de uma matriz de fatores* U, *tais que* $S^{(i)}(U) = S^{(i+j)}(U) \cup \{\ i, i+1, ..., i+j-1\ \}$ *para* $j = 1, ..., m$.

Tal definição especifica como integrantes de um mesmo um *supernode*, todas as linhas componentes de uma mesma estrutura bloco-diagonal densa na matriz triangular superior de fatores, e estruturalmente idênticas no padrão de colunas de seus elementos não nulos restantes.

No exemplo da Figura 4.2 os conjuntos $\{\ \mathbf{5,6,7,8}\ \}$ e $\{\ \mathbf{11,12,13,14,15}\ \}$ constituem dois *supernodes Maximais*, sendo o restante desta matriz formado por linhas de padrão estruturalmente distinto entre si (correspondentes a *nós elementares* individuais).

O que a exploração de padrões *supernodais* basicamente viabiliza, é um *particionamento* ao nível de linhas da matriz de fatores, agrupando-as segundo similaridades estruturais (a serem particularmente processadas).

• Nos processos de *ordenamento* baseados em representações por grafos, a noção de *supernode* corresponde a de um *clique* formado por nós *consecutivamente* numerados, no Grafo de Fatores Resultante.

Tais conjuntos de nós em algumas abordagens como [49], foram denominados como *indistinguíveis*, em virtude de poderem ser tratados para fins de armazenamento (e processamento a nível simbólico), como uma única entidade no grafo de representação associado.

• O processo de fatoração *simbólica* apresentado na seção 3.3 também explora implicitamente a estrutura de *supernodes* ao se considerar apenas uma fração de linhas contribuintes para fins de concatenação (à estrutura de cada linha base sendo gerada).

• O esquema de compressão proposto por Sherman [95] por exemplo, aproveita-se das similaridades estruturais para reduzir o número de índices de colunas explicita-

mente armazenados na representação de cada linha (simplesmente apontado-se para localizações comuns na estrutura de colunas, a serem compartilhadas por todas as linhas de um mesmo *supernode*).

O particionamento por *supernodes* traz consigo não apenas vantagens do ponto de vista estrutural, propiciando também uma significativa redução de *overheads* em tempo de execução (derivados principalmente de acessos indiretos ou temporários a memória).

Isso ocorre, porque considerando-se apenas sub-conjuntos de linhas restritas a um *mesmo supernode*, o processo de eliminação esparsa poder ser implementado unicamente por acessos *diretos* e *seqüenciais* a cada uma das linhas envolvidas (pelo fato de estarem seqüencialmente armazenados em posições "estruturalmente correspondentes", tanto os elementos das linhas contribuintes, como os de cada linha base a receber as contribuições de suas respectivas antecessoras).

Árvore de Eliminação Associada à Matriz de Fatores

De modo a se complementar a noção de *supernode* mostra-se conveniente a caracterização de uma estrutura auxiliar conhecida como *Árvore de Eliminação* (introduzida originalmente por Schreiber [93], e desde então amplamente explorada sob vários aspectos, nas abordagens diretas para a resolução de sistemas simétricos [74]).

O que a estrutura da árvore de eliminação visa fornecer é uma relação de *dependência* entre as linhas a serem eliminadas, estabelecendo-se desta forma níveis de precedência a serem obedecidos na seqüência de cancelamentos visando a geração dos fatores.

Tomando como exemplo a matriz da Figura 4.2, percebe-se que a linha de menor índice a depender da contribuição da linha 1 é a linha 13. Por sua vez, a primeira linha a sofrer a contribuição da linha 2 é a linha 9. Do mesmo modo, a primeira contribuição da linha 3 ocorre sobre a linha 7, assim como a da linha 4 ocorre sobre a linha 9. Isso significa por exemplo que a linha 9 só poderá contribuir sobre as linhas 11, 12, 13 ou 15 após ter recebido as contribuições das linhas 2 e 4, e que a linha 11 só poderá contribuir sobre as demais, após ter recebido as contribuições das linhas 2 e 9.

Levando-se esse processo adiante, e estabelecendo-se uma relação de "parentesco" (direto) entre cada linha e a *primeira* a sofrer sua contribuição (introduzindo-se uma conexão entre os nós a elas correspondentes, na estrutura de um grafo), pode-se definir uma árvore (como a ilustrada na Figura 4.3 para o exemplo em questão).

Esta estrutura, que é única para cada matriz de fatores, é conhecida como sua Árvore de Eliminação associada.

Pelo fato de se assumir como hipótese básica, que os sistemas lineares a serem solucionados sejam *irredutíveis*, a estrutura associada a representação de fatores corresponderá sempre a de uma árvore *conexa*. (Caso a hipótese de irredutibilidade fosse relaxada, tal estrutura corresponderia a de uma *floresta*, e uma generalização de modo a estender o ferramental apresentado para este caso seria possível, não

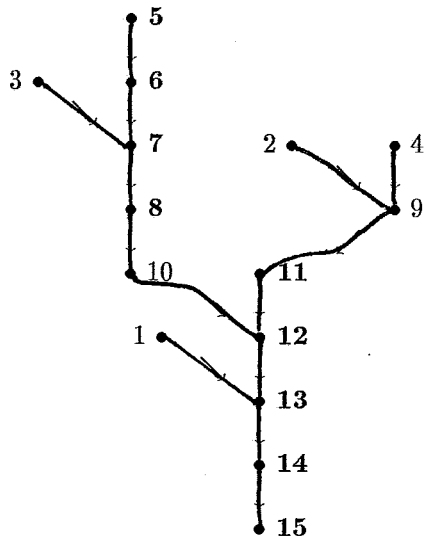


Fig 4.3: Árvore de Eliminação

sendo porém particularmente considerada neste trabalho).

Passaremos a apresentar algumas definições e propriedades da árvore de eliminação (visando uma melhor compreensão de alguns aspectos do processo de decomposição matricial).

Convenciona-se denominar as linhas a exercerem sua primeira contribuição sobre uma dada outra, como suas *antecessoras diretas*, bem como considera-se como *sucessora direta* de uma dada linha, a primeira a depender explícitamente de sua contribuição.

As antecessoras *diretas* de uma dada linha, podem não ser as únicas linhas a exercerem contribuição sobre a mesma, mostrando-se conveniente deste modo introduzir-se a noção de suas antecessoras *indiretas* (ou seja antecessoras diretas de alguma de suas antecessoras diretas, ou assim sucessivamente).

Essas relações de parentesco podem ser também formalizadas seguindo notações como a de *ancestrais* e *descendentes* (usualmente empregada nas representações de árvores de um modo geral), e onde se assume que um *descendente* possa ter um número não unitário de *ancestrais*, e cada *ancestral* apenas um único *descendente*.

Convém notar que a árvore de eliminação espelha originalmente apenas as relações de parentesco *direto* entre as linhas da matriz de fatores, e que o conjunto de linhas contribuintes sobre uma dada outra, alem de incluir necessariamente suas ancestrais diretas, pode em muitos dos casos incluir também as contribuições provenientes de algumas de suas ancestrais *indiretas* (como no caso da contribuição da linha 2 sobre a linha 11 no exemplo da Figura 4.2).

Formalmente, a árvore de eliminação pode ser completamente caracterizada mediante um único vetor auxiliar de dimensão n, a ser denominado **PARENT**, exibindo para cada uma de suas componentes, a *sucessora direta* de cada linha da matriz de fatores.

Assumindo-se o conhecimento da estrutura de fatores (obtida em alguma fase

prévia como a de fatoração simbólica por exemplo), pode-se trivialmente expressar o vetor **PARENT** a partir de:

$$\mathbf{PARENT}[i] = \mathbf{JU}[\mathbf{IU}[i]] \qquad \text{para } i = 1, ..., n-1$$

$$\mathbf{PARENT}[n] = 0 \qquad \text{por convenção}$$

Tal sendo possível, pelo fato da hipótese de irredutibilidade da matriz original, garantir obrigatóriamente a existência de ao menos um elemento não nulo além da diagonal em cada uma das linhas de sua matriz triangular superior de fatores (excetuando-se a última).

Relembrando-se a definição 3.3.1 da função $p(.)$ (indicadora da coluna do *primeiro fator não nulo* de cada linha), introduzida na seção 3.3, percebe-se que o vetor **PARENT** pode ser igualmente expresso a partir de:

$$\mathbf{PARENT}[i] = p(i) \qquad \text{para } i = 1, ..., n$$

A partir do conhecimento da estrutura da árvore de eliminação, podem ser formuladas novas caracterizações para o conceito de *Supernode*, apresentadas a seguir.

Definição 4.2.2 (Supernode Fundamental) *Define-se por "Supernode Fundamental" o maior conjunto de linhas* $i, i+1, ..., i+m$ *de uma matriz de fatores* U, *tais que* $S^{(i)}(U) = S^{(i+j)}(U) \cup \{\ i, i+1, ..., i+j-1\ \}$ *para* $j = 1, ..., m$ *e com a exceção de sua primeira linha constituinte, todas as demais possuam um só ancestral direto (de índice consecutivamente anterior) na árvore de eliminação associada.*

Particularmente para o exemplo da Figura 4.2, os *supernodes Fundamentais* são compostos pelos seguintes conjuntos $\{\ \mathbf{5,6}\ \}$, $\{\ \mathbf{7,8}\ \}$, $\{\ \mathbf{11}\ \}$, $\{\ \mathbf{12}\ \}$ e $\{\ \mathbf{13,14,15}\ \}$.

A definição 4.2.2 é mais restringente do que a 4.2.1 onde linhas com *mais de um* ancestral direto (porém com estrutura *similar* a de alguma de suas antecessoras diretas), são permitidas.

Ambos os conceitos no entanto, abrangem apenas conjuntos de *maior dimensão possível* (satisfazendo as condições especificadas).

Uma generalização com relação a este dimensionamento pode ser obtida mediante a definição de um conceito como o de *"Supernode Parcial"* (introduzido neste trabalho).

Definição 4.2.3 (Supernode Parcial) *Define-se por "Supernode Parcial" (ou simplesmente Supernode) um conjunto de linhas* $i, i+1, ..., i+m$ *de uma matriz de fatores* U, *tais que* $S^{(i)}(U) = S^{(i+j)}(U) \cup \{\ i, i+1, ..., i+j-1\ \}$ *para* $j = 1, ..., m$.

A razão para se lançar mão de tal caracterização, advém do fato de que do ponto de vista computacional, nem sempre *supernodes* de dimensão *plena* (como os Maximais ou Fundamentais) poderem ser eficientemente explorados por características

de *hardware* como memórias *cache* por exemplo. A solução nestes casos, consiste em se particionar os *Supernodes* de modo a conter um número de linhas capazes de serem integralmente armazenadas e exploradas por tais recursos.

Do ponto de vista teórico, os *supernodes Maximais* são os que melhor expressam as similaridades estruturais de uma matriz, sob o ponto de vista da menor quantidade de informação necessária para sua caracterização.

Sob o ponto de vista algorítmico, porém, o que se consegue obter de forma mais econômica e eficiente, é a representação por *supernodes Fundamentais*, explorando-se como informação básica o número de elementos não nulos de cada linha de U e a estrutura de sua árvore de eliminação, a partir do Teorema 4.1 apresentado sem demonstração, a seguir.

Teorema 4.1 (Determinação de Supernodes Fundamentais) *O sub-conjunto de linhas* $L = \{ i, i+1, ..., i+m \}$ *de uma matriz de fatores* U*, compõe um Supernode Fundamental se e somente se* L *for o maior conjunto de linhas contíguas tais que* $i+j-1$ *é um ancestral direto de* $i+j$ *na árvore de eliminação para* $j = 1, ..., m$ *e* $\eta^{(i)}(U) = \eta^{(i+m)}(U) + m$.

De posse de qualquer das representações apresentadas no entanto, é possível obter-se as demais, mediante amalgamentos ou particionamentos de seus *supernodes* e *nós elementares*.

Por sua generalidade e maior flexibilidade, a noção de *Supernodes Parciais* (doravante denominados apenas *Supernodes*) será fundamentalmente considerada nas conceituações sub-seqüentes deste trabalho.

Sob o ponto de vista computacional, assumirá-se sempre que uma representação por *supernodes Fundamentais* (única para cada matriz) tinha sido préviamente obtida, e tomada como base para posteriores particionamentos de *supernodes* em função de grandezas ou limitações algorítmicas ou de *hardware*.

Formulação Conceitual para Fatorações Supernodais

A base para a exploração de características *supernodais* nos métodos de eliminação numérica, consiste no fato de que a contribuição entre linhas constituintes de um mesmo *supernode* poder ser inteiramente previsível e efetuada *seqüencialmente* sobre as posições de memória envolvidas.

No caso da contribuição do conjunto de linhas de um dado *supernode* sobre outras linhas, mesmo assim ainda se pode explorar a seqüencialidade de acessos, lançando-se mão de um vetor auxiliar *denso* $\mathbf{V}$, acumulando temporáriamente as contribuições de todas as linhas relativas a um mesmo *supernode*, sendo tais resultados posteriormente incorporados ao vetor de trabalho expandido convencional $\mathbf{W}$ (adotado para o tratamento genérico das linhas sem padrão estrutural comum).

Este fato é ilustrado esquemáticamente na Figura 4.4, onde se considera a contribuição do *supernode* constituído pelas linhas $\{ 5, 6, 7, 8 \}$ sobre a linha base 10, mediante o uso da acumulação temporária sobre o vetor auxiliar denso $\mathbf{V}$ (de forma seqüencial), para a seguir se incorporar o resultado de tais contribuições ao vetor

de trabalho expandido convencional **W** (a ser posteriormente incorporado de forma compacta, definitivamente sobre as posições correspondentes da linha base gerada em U).

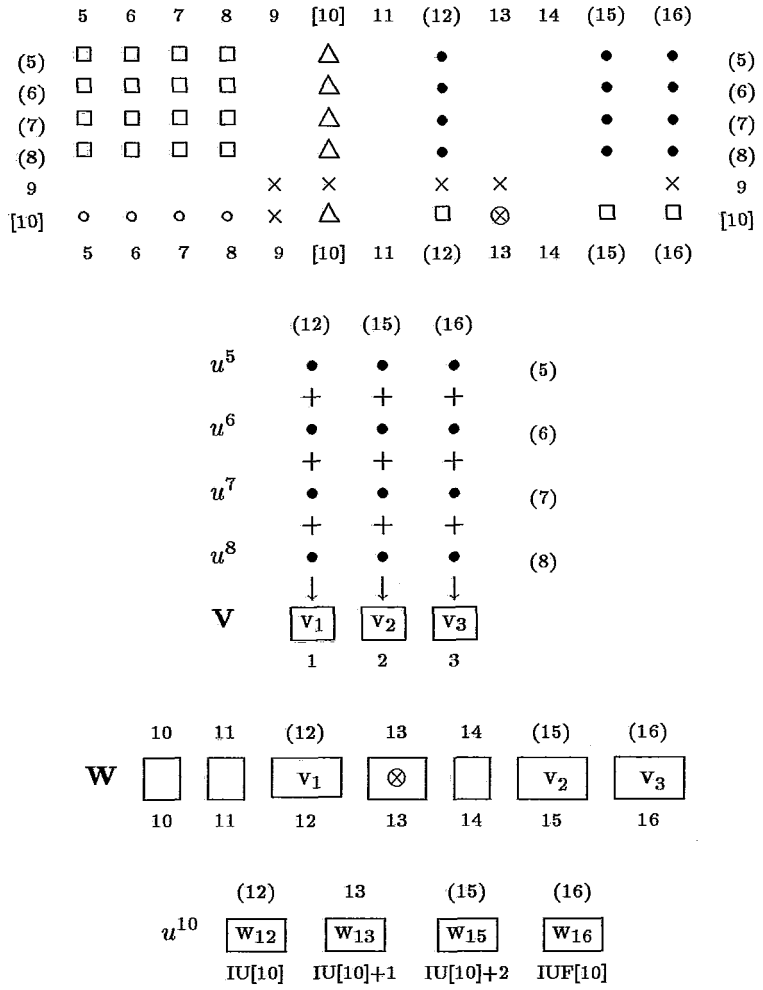


Fig 4.4: Acumulação de Vetores Supernodais

Nos algoritmos a serem apresentados a seguir, faz-se pois a distinção de 3 formas de processamento do *loop* mais interno, correspondentes ao acúmulo das contribuições de cada uma das linhas l, sobre a linha básica i de cada etapa.

Inicialmente apresenta-se no Algoritmo **4.1** um esquema voltado para a geração de fatores por Linhas. Posteriormente apresenta-se um esquema equivalente para a geração por Colunas.

Ambos os algoritmos, inicialmente apresentados num nível mais abstrato nesta subseção, serão oportunamente refinados e detalhados na forma de Procedimentos completos, um pouco mais adiante.

Como convenção, adotou-se a representação de determinadas grandezas escalares específicamente relacionadas a estruturas de *supernodes*, mediante identificadores "replicados" (como *ii* e *nn*) de modo a se distinguí-las de suas equivalentes na estrutura de *nós elementares* usualmente adotada em todos os procedimentos até então considerados.

Descrição de Passos do Algoritmo **4.1**

No passo (a) processa-se a eliminação, para cada um dos *supernodes* na qual a matriz de fatores encontra-se particionada.

A partir do passo (b) processa-se a contribuição das linhas constituintes de cada *supernode* a incidir sobre a linha base corrente. (Assume-se que o conjunto $\mathbf{C}_*^{(i)}$ contendo o índice de todos os *supernodes* incidentes na etapa i tenha sido determinado a priori, durante alguma fase de pré-processamento, como a de fatoração simbólica).

No passo (c) explora-se o Teorema 4.1, identificando-se se o número de elementos da linha primeira linha do *supernode* corrente a efetivamente contribuir sobre a linha base é idêntico ao número de elementos não nulos da mesma. Em caso afirmativo, sabe-se por construção, que as contribuições subseqüentes deste particular *supernode*, serão estruturalmente idênticas, viabilizando o acesso *direto* as componentes da linha base no passo (d).

No passo (d) processam-se as contribuições de todas as linhas pertinentes do *supernode*, *diretamente* sobre a linha base (pelo fato de se estar considerando apenas o conjunto de linhas pertencentes ao *mesmo supernode*).

No passo (e) identifica-se se o atual *supernode* é formado por apenas um único nó. Em caso afirmativo, não se mostra vantajoso a utilização das técnicas mais aprimoradas (adotadas no passo (g)).

No passo (f) deste modo, processa-se a contribuição isolada da única linha constituinte do atual *supernode*, de forma descompactada mediante a técnica de vetores de trabalho expandidos usual.

No passo (g) processam-se as todas as contribuições das linhas constituintes do *supernode*, sobre um vetor auxiliar denso $\mathbf{V}$, a ser posteriormente incorporado ao vetor de trabalho expandido $\mathbf{W}$ (no passo (h)).

No passo (h) incorporam-se as contribuições acumuladas de forma convencional no vetor de trabalho expandido, anulando-se suas posições a medida que os valores definitivos forem sendo transferidos para a estrutura de fatores da linha base. (Esta forma alternativa de processamento de contribuições mediante vetores de trabalho expandidos, não havia sido considerada até então, mostrando-se no entanto igualmente aplicável, desde que assuma-se que o vetor de trabalho tenha sido préviamente anulado em todas as suas posições ao se iniciar o processamento, vindo a ser paulatinamente anulado em todas as posições utilizadas para a geração da linha base a cada etapa).

□

Apresenta-se a seguir no Algoritmo **4.2** uma versão análoga a do Algoritmo anterior, só que voltada para eliminações *Supernodais* com geração dos fatores por *Colunas*.

Este Algoritmo difere básicamente do anterior, com relação a forma de processamento das contribuições (retardadas) das linhas de um *mesmo supernode*, sobre a coluna base sendo gerada a cada etapa. (Para uma melhor compreensão deste fato

Algoritmo 4.1 *Fatoração Supernodal (Linhas)*

para i **de** 1 **até** n **faça** $\mathbf{W}[i] \leftarrow 0$.

para cada *supernode ii* **de** 1 **até** *nn* **faça** (a)

para cada *linha base* $i \in$ ao *supernode ii* a eliminar **faça**

piv $\leftarrow \mathbf{DI}[i]$, $\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$
ilen $\leftarrow \mathbf{IUF}[i] - \mathbf{IU}[i]$

para cada *supernode kk* $\in \mathbf{C}_*^{(i)}$ a contribuir sobre a *linha base* i **faça** (b)

lbeg $\leftarrow$ indice da *primeira* linha do *supernode kk* corrente
lend $\leftarrow$ min { indice da *ultima* linha do *supernode kk*, $i-1$ }

se ($\mathbf{IUF}[lbeg] - \mathbf{IUP}[lbeg] =$ *ilen*) **então** (c)

para cada linha l **de** *lbeg* **até** *lend* **faça**
atualize $\mathbf{IUP}[l]$, *piv* e *normalize* $\mathbf{UN}[\mathbf{IUP}[l]]$

---

adicione contribuições da linha l *diretamente* sobre a *linha base* (d)

---

fim para l
senão

$l \leftarrow$ *lbeg* (*indice da atual linha contribuinte*)
atualize $\mathbf{IUP}[l]$, *piv* e *normalize* $\mathbf{UN}[\mathbf{IUP}[l]]$

se (*lend* = *lbeg*) **então** (e)

---

adicione contribuições da linha l *descompactadamente* no vetor $\mathbf{W}$ (f)

---

senão
incorpore contribuições da linha l no *vetor supernodal* $\mathbf{V}$
para cada linha l **de** *lbeg* $+1$ **até** *lend* **faça**
atualize $\mathbf{IUP}[l]$, *piv* e *normalize* $\mathbf{UN}[\mathbf{IUP}[l]]$

---

adicione contribuições da linha l *seqüencialmente* no vetor $\mathbf{V}$ (g)

---

fim para l
incorpore contribuições do vetor $\mathbf{V}$ *descompactadamente* no vetor $\mathbf{W}$
fim se (*lend* = *lbeg*)

fim se ($\mathbf{IUF}[lbeg] - \mathbf{IUP}[lbeg] =$ *ilen*)
fim para *kk*

$\mathbf{DI}[i] \leftarrow 1.$ / *piv*

incorpore contribuições de $\mathbf{W}$ na *linha base*, *anulando-as* respectivamente a seguir (h)

fim para i
fim para *ii*

Alg 4.1: *Fatoração Supernodal (Linhas)*

Algoritmo 4.2 *Fatoração Supernodal (Colunas)*

para i **de** 1 **até** n **faça** $\mathbf{W}[i] \leftarrow 0$.

para cada *supernode ii* **de** 1 **até** *nn* **faça**

para cada *linha base* $i \in$ ao *supernode ii* a processar **faça**

$piv \leftarrow \mathbf{DI}[i]$, $\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$

para cada *supernode kk* $\in \mathbf{C}_*^{(i)}$ a contribuir sobre a *coluna base* i **faça**

lbeg $\leftarrow$ indice da *primeira* linha do *supernode kk* corrente
lend $\leftarrow$ min { indice da *ultima* linha do *supernode kk*, $i-1$ }

se ($lend = i-1$) **então** (*a*)

para cada linha l **de** *lbeg* **até** *lend* **faça**
atualize **IUPF**[l] e *piv*, *normalizando* **UN**[**IUPF**[l]]

---

adicione contribuições "retardadas" de l *seqüencialmente* em **W** (*b*)

---

fim para l

senão

$l \leftarrow$ *lbeg* (*indice da atual linha contribuinte*)
atualize **IUPF**[l] e *piv*, *normalizando* **UN**[**IUPF**[l]]

se ($lend = lbeg$) **então**

---

adicione contribuições "retardadas" de l *descompactadamente* em **W** (*c*)

---

senão
incorpore contribuições da linha l no *vetor supernodal* **V**
para cada linha l **de** $lbeg + 1$ **até** *lend* **faça**
atualize **IUPF**[l] e *piv*, *normalizando* **UN**[**IUPF**[l]]

---

adicione contribuições "retardadas" de l *seqüencialmente* em **V** (*d*)

---

fim para l
incorpore contribuições do vetor **V** *descompactadamente* no vetor **W**
fim se ($lend = lbeg$)

fim se ($lend = i-1$)

fim para *kk*

$\mathbf{DI}[i] \leftarrow 1.$ / *piv*

fim para i

fim para *ii*

Alg 4.2: *Fatoração Supernodal (Colunas)*

reporta-se ao Procedimento **4.2(2)** a ser posteriormente apresentado).

Descrição de Passos do Algoritmo **4.2**

No passo (*a*) do Algoritmo **4.2** identifica-se se o *supernode* corrente é o constituído pelas linhas da estrutura bloco-diagonal englobando a coluna base. Em caso afirmativo, a partir deste ponto, processa-se no passo (*b*) o acúmulo das contribuições *supernodais* sobre as posições correspondentes no vetor de trabalho expandido **W**.

No passo (*b*) ao invés de se processar a contribuição das linhas do *supernode* diretamente sobre as posições da coluna base sendo gerada, processa-se o acúmulo de tais contribuições de forma *seqüencial* sobre o vetor de trabalho **W**. Esta forma de acesso, difere básicamente da forma até então adotada pelos demais procedimentos propostos ou pela literatura, no sentido de se explorar o padrão seqüencial de acessos em uma estrutrura usualmente adotada para acúmulo descompactado de contribuições em posições *randômicas* (como no caso do vetor de trabalho expandido). Tal é possível por se poder garantir por construção, o fato de que as contribuições retardadas sobre o vetor **W** a serem posteriormente incorporadas a coluna base, ocuparem posições contíguas deste vetor, a partir do processamento das contribuições das linhas constituintes da estrutura bloco-diagonal remanescente.

No passo (*c*) processam-se convencionalmente as contribuições retardadas relativas a *supernodes* contendo apenas uma única linha, lançando-se mão da técnica de vetores de trabalho expandidos usual.

No passo (*d*) processam-se mediante o uso do vetor *Supernodal* auxiliar **V**, as contribuições das linhas de um mesmo *supernode* nos casos não cobertos anteriormente, ou seja, de *supernodes* compostos por mais de uma linha, e que nao correspondam a estrutura bloco-diagonal englobando a coluna base corrente. Posteriormente tais contribuições serão incorporadas de forma descompactada sobre o vetor expandido **W** contendo o valor consolidado de todas as contribuições retardadas a serem incorporadas oportunamente sobre a coluna base sendo gerada.

□

Adição de Vetores Supernodais

Mediante a exploração da *similaridade estrutural* e *seqüencialidade* de armazenamento de Vetores Esparsos constituintes de um *mesmo supernode*, o processo de sua adição pode ser formulado na Alternativa **4.2(a)**, permitindo um acesso *seqüencial* as posições no vetor base **U** *diretamente* afetadas pela contribuição dos elementos correspondentes em **V**.

Tal alternativa pode ser utilizada para se implementar por exemplo o passo (*c*) do Algoritmo **4.1** como se verá mais detalhadamente na próxima sub-seção.

Percebe-se neste caso, que no tratamento das linhas componentes de um mesmo *supernode*, todo o processamento pode ser efetuado como se estivesse operando de forma *densa*, sobre os vetores *esparsos* seqüencialmente armazenados, o que confere a esta alternativa o melhor desempenho computacional entre todas as formas de adição

Alternativa 4.2(a) *Adição de Vetores Supernodais*

(*offset base* para o endereçamento das posições de u correspondentes as de v)
$ib \leftarrow iu - iv$

(processamento de $u \leftarrow u + v$ *diretamente* sobre as posições de u)
para j **de** iv **até** ivf **faça**
$\mathbf{U}[ib+j] \leftarrow \mathbf{U}[ib+j] + \mathbf{V}[j]$

Alt 4.2(a): *Adição de Vetores Supernodais*

até então consideradas (por se poder acessar e operar *diretamente* as posições do vetor base).

Eliminações Supernodais com Geração dos Fatores por Linhas

Apresenta-se inicialmente no Procedimento **4.2(1)** uma versão refinada do Algoritmo **4.1** para o processo de eliminação *Supernodal* com geração dos fatores por *Linhas*, combinando-se o uso vetores de trabalho expandidos convencionais (análogamente a seção 3.4), com um tratamento diferenciado e eficiente das contribuições *supernodais*.

Descrição de Passos do Procedimento **4.2(1)**

No passo (a) do Procedimento **4.2(1)** anulam-se inicialmente todas as posições do vetor de trabalho expandido **W**, e que no caso da eliminação Supernodal por Linhas, será utilizado na forma de um vetor de Contribuições.

No passo (b) processa-se a eliminação para cada um dos *supernodes* ii no qual foi particionada a matriz de fatores.

No passo (c) para cada uma das linhas i constituintes do *supernode* corrente ii efetua-se o processo de eliminação.

No passo (d) armazena-se em $ilen$ o número de elementos não nulos (excluindo-se a diagonal) da linha corrente i da matriz de fatores. Tal valor será utilizado posteriormente no passo (h), de modo a identificar os *supernodes* com estrutura idêntica a da linha corrente, permitindo uma implementação mais eficiente do processo de combinação linear entre as linhas envolvidas, no passo (k).

No passo (e) inicia-se o processamento para cada um dos *supernodes* contribuintes sobre a linha base.

No passo (f) armazena-se em kk o índice do atual *supernode* contribuinte.

No passo (g) armazenam-se em $lbeg$ e $lend$ os índices da primeira e última linha do *supernode* kk a contribuírem sobre a linha base i. Nota-se que no caso da linha i pertencer ao *supernode* kk, o índice da última linha contribuinte sobre a mesma

Procedimento 4.2(1) *Eliminação Supernodal (Linhas)*

```
para i de 1 até n faça W[i] ← 0.                                          (a)
( para cada supernode ii a processar )
para ii de 1 até nn faça                                                  (b)
    ( para cada linha i do supernode ii a eliminar )
    para i de INOD[ii] até INODF[ii] faça                                 (c)
        piv ← DI[i], IUP[i] ← IU[i]
        ilen ← IUF[i] − IU[i]                                             (d)
        ( para cada supernode kk a contribuir sobre a linha base )
        para k de IKNOD[i] até IKNODF[i] faça                             (e)
            kk ← KNOD[k]      ( indice do atual supernode )               (f)
            lbeg ← INOD[kk], lend ← min { INODF[kk], i − 1 }              (g)
            se ( IUF[lbeg] − IUP[lbeg] = ilen ) então                     (h)
                para l de lbeg até lend faça                              (i)
                    iupl ← IUP[l], IUP[l] ← IUP[l] + 1
                    um ← UN[iupl] * DI[l]
                    piv ← piv − UN[iupl] * um
                    UN[iupl] ← um
                    jl ← IUP[l] − IU[i]                                   (j)
                    ( acumulação das contribuições da linha l sobre a linha base )
                ---------------------------------------------------------
                    para j de IU[i] até IUF[i] faça                       (k)
                        UN[j] ← UN[j] − UN[jl + j] * um
                ---------------------------------------------------------
                fim para l
            senão
                l ← lbeg          ( indice da atual linha contribuinte )
                iupl ← IUP[l], IUP[l] ← IUP[l] + 1
                um ← UN[iupl] * DI[l]
                piv ← piv − UN[iupl] * um
                UN[iupl] ← um
                se ( lend = lbeg ) então                                  (l)
                    ( acumulação das contribuições da linha l no vetor expandido )
                ---------------------------------------------------------
                    para j de IUP[l] até IUF[l] faça                      (m)
                        W[JU[j]] ← W[JU[j]] + UN[j] * um
                ---------------------------------------------------------
                senão
                    jlen ← IUF[l] − iupl                                  (n)
                    ( incorporação da primeira contribuição ao vetor supernodal )
                    para j de 1 até jlen faça                             (o)
                        V[j] ← UN[iupl + j] * um
                    para l de lbeg + 1 até lend faça                      (p)
                        iupl ← IUP[l], IUP[l] ← IUP[l] + 1
                        um ← UN[iupl] * DI[l]
                        piv ← piv − UN[iupl] * um
                        UN[iupl] ← um
                        ( acumulação das contribuições da linha l no vetor supernodal )
                    ---------------------------------------------------------
                        para j de 1 até jlen faça                         (q)
                            V[j] ← V[j] + UN[iupl + j] * um
                    ---------------------------------------------------------
                    fim para l
                    jl ← 1 − IUP[lbeg]                                    (r)
                    ( incorporação das contribuições supernodais ao vetor expandido )
                    para j de IUP[lbeg] até IUF[lbeg] faça                (s)
                        W[JU[j]] ← W[JU[j]] + V[jl + j]
                fim se ( lend = lbeg )
            fim se ( IUF[lbeg] − IUP[lbeg] = ilen )
        fim para k
        DI[i] ← 1. / piv
        ( incorporação das contribuições do vetor expandido na linha base )
        para j de IU[i] até IUF[i] faça                                   (t)
            UN[j] ← UN[j] − W[JU[j]], W[JU[j]] ← 0.
    fim para i
fim para ii
```

corresponderá a $i-1$, processando-se deste modo uma contribuição "parcial" do *supernode* corrente.

No passo (h) identifica-se se as linhas do *supernode* contribuinte possuem estrutura de elementos não nulos idêntica a da linha básica.

Em caso afirmativo, no passo (i) processam-se as contribuições para cada uma das linhas l contribuintes.

Neste caso, no passo (j) armazena-se temporariamente em jl um *offset* relativo a posição inicial da linha l a contribuir sobre a linha corrente, de modo a se aproveitar a indexação seqüencial do índice j (utilizado na varredura dos elementos base no passo (k)) para o acesso aos elementos $jl+j$ da linha contribuinte. (Nota-se particularmente que para $j = \mathbf{IU}[i]$, $jl+j = \mathbf{IUP}[l]$, correspondendo a posição do primeiro elemento não nulo de l contribuinte sobre a linha i).

Efetivamente processa-se no passo (k) a contribuição de cada um dos elementos de l, diretamente sobre as posições correspondentes da linha base, em função de sua estrutura de elementos não nulos (a partir da coluna i) ser idêntica. Nota-se que este *loop* mais interno não contém qualquer forma de acesso *indireto* a posições de memória (como em $\mathbf{W}[\mathbf{JU}[j]]$ ou $\mathbf{UN}[\mathbf{PLIN}[iplin]]$), tornando-o desta forma mais eficientemente implementável (por viabilizar a plena exploração da seqüencialidade de acessos à linha base e à linha contribuinte). Pelo fato de se operar diretamente sobre os elementos da linha base, os *overheads* de utilização de estruturas auxiliares (como as de vetores de trabalho) são também completamente eliminados.

No caso do número de elementos contribuintes das linhas do *supernode* corrente ser inferior ao de elementos da linha base, duas outras opções de processamento são consideradas no passo (l).

No caso do *supernode* corrente possuir apenas uma única linha, mostra-se mais vantajoso processar-se o acumulo das contribuições de l diretamente sobre o vetor de trabalho expandido $\mathbf{W}$, do que lançar mão de um vetor auxiliar como $\mathbf{V}$ (utilizado no passo (q)).

Assim, processa-se no passo (m) uma contribuição convencional mediante a utilização de vetor de trabalho expandido (como adotado nos procedimentos da seção 3.4).

No caso do número de linhas contribuintes do *supernode* corrente ser superior a unidade, pode-se implementar o acumulo de suas contribuições de forma mais eficiente, explorando-se o fato da estrutura de seus elementos não nulos (a partir da i'ezima coluna) ser idêntica.

Para tal, armazena-se inicialmente em $jlen$ no passo (n), o número de elementos não nulos das linhas do *supernode* corrente a contribuir sobre a linha base.

No trecho compreendido entre os passos (o) e (s), processa-se uma técnica similar a adotada para vetores de trabalho expandidos. Neste caso porém o acumulo de contribuições será efetuado num vetor auxiliar $\mathbf{V}$, distinguindo-se do caso anterior, pela total seqüencialidade nos acessos a memória, e pelo fato da estrutura *base* a receber as contribuições ser justamente o vetor expandido $\mathbf{W}$ (a ser posteriormente incorporado a linha básica corrente).

Assim, no passo (o) processa-se inicialmente o carregamento dos valores da primeira linha do *supernode* contribuinte no vetor de acumulo *supernodal* **V**.

No passo (p) inicia-se o processamento do acumulo das contribuições das demais linhas do *supernode*, efetivamente implementadas no *loop* mais interno (q).

Nota-se que similarmente ao passo (k), os acessos a ambas as estruturas de armazenamento no passo (q) processa-se de forma puramente seqüencial (evitando-se acessos randômicos do tipo indireto como os do passo (m)).

Em função do mesmo padrão estrutural de elementos não nulos das linhas constituintes do *supernode* corrente, as posições $iupl + j$ da atual linha contribuinte no passo (q), podem ser diretamente acumuladas sobre posições correspondentemente indexadas no vetor **V**.

Uma vez concluído o acumulo das contribuições, no passo (r) armazena-se temporariamente em jl o *offset* a ser adotado de modo a permitir a incorporação dos valores seqüencialmente armazenados em **V**, sobre a posições correspondentes no vetor expandido **W**. (Nota-se particularmente que para $j = \mathbf{IUP}[lbeg]$, a posição $jl + j$ corresponderá a primeira posição do vetor de acumulo *supernodal* **V**).

No passo (s) processa-se a transferência dos valores acumulados em **V**, para o vetor expandido **W**, de modo a serem posteriormente incorporados a linha base (no passo (t)). Em função da identicidade das estruturas de elementos (a partir da i'ezima coluna) do *supernode* corrente, toma-se arbitrariamente o índice *lbeg* da primeira de suas linhas, de modo a servir de base para a indexação em **W** e descompactação dos elementos acumulados em **V**.

Finalmente, no passo (t) processa-se a incorporação de todas as contribuições consolidadas no vetor de trabalho expandido, de volta as posições correspondentes da linha base, anulando-se concomitantemente as posições de **W** já incorporadas (de modo a se garantir que para a próxima etapa do *loop* principal, o vetor de contribuições esteja inicialmente zerado).

□

Eliminações Supernodais com Geração dos Fatores por Colunas

Apresenta-se a seguir no Procedimento **4.2(2)** uma versão refinada do Algoritmo **4.2** para o processo de eliminação *Supernodal* com geração dos fatores por *Colunas*.

Tal procedimento difere basicamente do anterior, em função de considerar-se como caso principal para tratamento diferenciado sob a forma *supernodal* "plena", as contribuições relativas apenas ao *supernode* de índice mais elevado, abrangendo exclusivamente a estrutura de elementos bloco-diagonais a ele associada.

Neste caso particular, ao invés de se processar o acumulo das contribuições diretamente sob as posições da estrutura de fatores base sendo gerada, diferentemente do procedimento anterior, acumulam-se tais contribuições diretamente sobre o vetor expandido **W**, e que naturalmente será gradualmente incorporado aos elementos sendo gerados (e normalizados) na coluna base a cada etapa.

O tratamento das contribuições dos demais *supernodes*, análogamente ao proce-

dimento anterior, pode ser eficientemente explorado, adotando-se para tal a mesma estrutura do vetor de acumulações *supernodais* **V**.

Uma última distinção básica com relação a geração dos fatores por *Linhas*, é que no caso da geração por *Colunas*, pelo fato de se estar considerando apenas as contribuições à esquerda da diagonal corrente, as contribuições das linhas associadas a cada *supernode* não possuem o mesmo número de elementos não nulos (entre si), distinguindo-se consecutivamente em uma unidade. Tal inevitavelmente ocorre, em função da gradual exclusão das colunas inicialmente consideradas no processamento das contribuições da linha prévia, com índice anterior ao da nova coluna em **W** a ser atualizada.

Descrição de Passos do Procedimento **4.2(2)**

No passo (a) do Procedimento **4.2(2)** é feita a identificação do último *supernode* contribuinte, e em caso de ser constituído por uma estrutura bloco diagonal (englobando necessariamente a coluna corrente i) processa-se de forma diferenciada no passo (b), o acumulo das contribuições correspondentes.

Como notado anteriormente, neste caso, no passo (b), processa-se um acumulo das contribuições diretamente sobre as colunas seqüencialmente correspondentes do vetor **W** (que será posteriormente incorporado a coluna base, por ocasião da normalização de cada um de seus elementos). Tal forma de acesso à **W** é possível por construção, por estar se considerando apenas as acumulações a esquerda da diagonal, relativas as linhas do *supernode* bloco-diagonal (envolvendo a coluna corrente a ser "retardadamente" atualizada).

No caso dos demais *supernodes*, no passo (c) é feita a distinção entre aqueles possuindo apenas uma linha, dos possuidores de um número superior.

Nos casos de *supernodes* compostos por apenas uma linha, processa-se no passo (d) uma acumulação tradicional de forma descompactada sobre o vetor de contribuições (análogamente ao apresentado na seção 3.4).

Nos demais casos, processa-se no passo (f) uma acumulação de forma *supernodal* análoga a utilizada no passo (q) do procedimento anterior.

Como notado inicialmente no entanto, no caso da geração dos fatores por *Colunas*, o número de elementos não nulos de cada uma das linhas do *supernode* corrente a efetivamente contribuir sobre a coluna base é distinto entre si, razão pela qual é decrementado de uma unidade o valor de $jlen$ cada instância do passo (e).

Outro ponto a se observar é que justamente em virtude desta alteração dinâmica do número de contribuições, o vetor *supernodal* **V** é montado e acessado em ordem reversa a do procedimento anterior, com **V**[1] neste caso correspondendo à **UN**[$iupfl-1$] e **V**[$jlen$] à **UN**[**IU**[l]].

Uma última distinção a se notar no passo (g) do Procedimento **4.2(2)** é que em virtude do incremento das posições delimitadoras do vetor **IUPF** (durante o processamento das contribuições de cada uma das linhas l associadas), o acesso

Procedimento 4.2(2) *Eliminação Supernodal (Colunas)*

```
para i de 1 até n faça W[i] ← 0.
( para cada supernode ii a processar )
para ii de 1 até nn faça
    ( para cada linha i do supernode ii a eliminar )
    para i de INOD[ii] até INODF[ii] faça
        piv ← DI[i], IUPF[i] ← IU[i]
        ( para cada supernode kk a contribuir sobre a coluna base )
        para k de IKNOD[i] até IKNODF[i] faça
            kk ← KNOD[k]      ( indice do atual supernode )
            lbeg ← INOD[kk], lend ← min { INODF[kk], i − 1 }
            se ( lend = i − 1 ) então                                   (a)
                para l de lbeg até lend faça
                    iupfl ← IUPF[l], IUPF[l] ← IUPF[l] + 1
                    um ← UN[iupfl] − W[l], W[l] ← 0.
                    UN[iupfl] ← um * DI[l]
                    piv ← piv − UN[iupfl] * um
                    jl ← i − iupfl
                    ( acumulação seqüencial das contribuições de l no vetor expandido )
                ---------------------------------------------------------
                    para j de IU[l] até iupfl − 1 faça                  (b)
                        W[jl + j] ← W[jl + j] + UN[j] * um
                ---------------------------------------------------------
                fim para l
            senão
                l ← lbeg          ( indice da atual linha contribuinte )
                iupfl ← IUPF[l], IUPF[l] ← IUPF[l] + 1
                um ← UN[iupfl] − W[l], W[l] ← 0.
                UN[iupfl] ← um * DI[l]
                piv ← piv − UN[iupfl] * um
                se ( lend = lbeg ) então                                (c)
                    ( acumulação das contribuições da linha l no vetor expandido )
                ---------------------------------------------------------
                    para j de IU[l] até iupfl − 1 faça                  (d)
                        W[JU[j]] ← W[JU[j]] + UN[j] * um
                ---------------------------------------------------------
                senão
                    jlen ← iupfl − IU[l]
                    ( incorporação da primeira contribuição ao vetor supernodal )
                    para j de 1 até jlen faça
                        V[j] ← UN[iupfl − j] * um
                    para l de lbeg + 1 até lend faça
                        iupfl ← IUPF[l], IUPF[l] ← IUPF[l] + 1
                        um ← UN[iupfl] − W[l], W[l] ← 0.
                        UN[iupfl] ← um * DI[l]
                        piv ← piv − UN[iupfl] * um
                        jlen ← jlen − 1                                 (e)
                        ( acumulação das contribuições da linha l no vetor supernodal )
                    ---------------------------------------------------------
                        para j de 1 até jlen faça                       (f)
                            V[j] ← V[j] + UN[iupfl − j] * um
                    ---------------------------------------------------------
                    fim para l
                    jl ← IUPF[lbeg] −1
                    ( incorporação das contribuições supernodais ao vetor expandido )
                    para j de IU[lbeg] até IUPF[lbeg] −2 faça           (g)
                        W[JU[j]] ← W[JU[j]] + V[jl − j]
                fim se ( lend = lbeg )
            fim se ( lend = i − 1 )
        fim para k
        DI[i] ← 1. / piv
    fim para i
fim para ii
```

Proc 4.2(2): *Eliminação Supernodal (Colunas)*

as posições de **W** a receberem a contribuição do vetor *supernodal* **V**, é delimitado terminalmente por **IUPF**[*lbeg*] − 2 (e não por **IUPF**[*lbeg*] − 1 como seria esperado).

Finalmente cabe mencionar que em função da própria natureza intrínseca do processo de geração por *Colunas*, com a incorporação "retardada" e normalização concomitante das contribuições no *loop* intermediário (em termos da variável *l*) a cada etapa, torna-se desnecessário um processamento em separado das incorporações, como no passo (*t*) do Procedimento anterior.

□

Contribuições Supernodais via Listas Recorrentes de Endereços

Uma forma de se minimizar uma das ineficiências presentes em ambos os algoritmos *supernodais* até então apresentados, será considerada a seguir.

Em face a similaridade da nova técnica para ambas as formas de geração de fatores, optou-se particularmente por detalhar o seu desenvolvimento apenas para o caso da geração por Linhas.

Analisando-se os passos de (*o*) até (*s*) do Procedimento **4.2(1)** (voltados para o processamento *supernodal* mediante o vetor auxiliar **V**), o que se percebe é que nos casos em que o número de linhas constituintes do *supernode* sendo tratado for relativamente pequeno (da ordem de 2 ou 3 por exemplo), os *overheads* gastos com a transferência da primeira linha para o vetor **V** (no passo (*o*)), seguidos do acúmulo das contribuições da segunda linha neste vetor (no passo (*q*)), para posteriormente serem incorporadas todas as contribuições consolidadas em **V** de forma descompactada em **W** (no passo (*s*)), para finalmente serem compactadas de volta as posições definitivas na linha base (no passo (*t*)), implicam num número excessivo de transferências em memória, e que poderiam ser evitadas, caso se lançasse mão de recursos como o de listas simbólicas de endereços por exemplo.

Uma das soluções para se minimizar tais acessos, seria relaxar a condição do passo (*l*) do Procedimento de eliminação *Supernodal por Linhas* permitindo que *supernodes* com 2 ou 3 linhas fossem operados convencionalmente mediante o uso de vetores de trabalho expandidos.

Outra solução, proposta nesta sub-seção, é substituir-se o passo (*m*) deste mesmo Procedimento por um processamento como o do passo (*d*) da Alternativa **4.2(b)** a ser apresentada a seguir.

A idéia básica consiste em se lançar mão de Listas Simbólicas de Endereços para se tratar o caso particular de *supernodes* de *pequena dimensão.*

Tal fundamenta-se no fato de que em um número significativo de aplicações, o percentual de *supernodes* de pequena dimensão não se mostrar dominante sobre o número total de nós.

Isso implica que um tratamento por listas simbólicas deste caso em particular, não tende a apresentar um comportamento de forma tão "explosiva" com relação ao tamanho das listas (por estar se considerando justamente apenas casos de *pequena dimensão*).

Nota-se porém que a aplicação de listas simbólicas para *supernodes* com dimensão acima de 2 ou 3 linhas, não se mostra indicada ou vantajosa por uma razão muito mais forte, qual seja a de se deixar de explorar justamente os acessos *seqüencias* do tipo *direto*, presentes apenas em *loops* como o do passo (q) (no Procedimento **4.2(2)**).

A principal motivação para a técnica apresentada a seguir é pois a de minizar-se os *overheads* decorrentes de acessos e transferências temporárias de dados em memória, e que nos casos de *pequena dimensão*, mostraram-se significativamente presentes, nos procedimentos até então considerados.

A propriedade básica explorada pela Alternativa **4.2(b)**, é o fato de que durante o processamento das contribuições das linhas de um *mesmo supernode*, as posições acessadas na linha base (e que estariam especificadas individualmente para cada caso numa lista de endereços convencional), apresentarem no entanto um padrão nítidamente *recorrente*, no sentido de que os mesmos endereços acessados durante a contribuição da primeira linha, serão posteriormente acessados, durante o processamento das demais linhas constituintes (em face a similaridade estrutural de todas as linhas do *mesmo supernode*).

Assim, a alternativa apresentada a seguir mostra-se muito mais econômica com relação ao tamanho da lista efetivamente necessário para a representação de todos os endereços acessados (e que não mais se mostra diretamente proporcional ao número de operações em ponto flutuante, como no caso convencional, passando a ser função básicamente do número total de *supernodes* operados, neste novo caso).

Em face a similaridades do código a ser apresentado e os procedimentos anteriormente detalhados, optou-se por lista-lo na forma de uma Alternativa, destacando-se apenas os passos relevantes para sua compreensão.

Em função de possíveis diferenças de desempenho em distintas classes de arquiteturas, não se especificou explícitamente um valor para a dimensão máxima dos *supernodes* a serem tratados pela técnica de Listas Recorrentes, deixando-se a variável *nlmax* como um parâmetro ajustável a cada caso.

Descrição de Passos da Alternativa **4.2(b)**

No passo (a) da Alternativa **4.2(b)** inicializa-se o indicador da posição inicial na lista simbólica recorrente a ser utilizada no processamento do primeiro *supernode* a satisfazer a condição especificada no passo (b).

No passo (b) testa-se a dimensão do *supernode* corrente, e no caso de possuir um número de linhas suficientemente pequeno, processam-se nos passos subseqüentes as contribuições relativas as suas linhas constituintes (mediante a exploração das recorrências no acesso as posições da lista simbólica de endereços). Caso o número de linhas constituintes do *supernode* seja superior ao limite *nlmax*, processam-se as contribuições de forma usual, mediante o vetor *supernodal* auxiliar **V** (não listado explícitamente na Alternativa) de modo análogo aos passos desde (n) até (s) do Procedimento **4.2(1)**.

Alternativa 4.2(b) *Contribuições Supernodais via Listas Recorrentes*

(*posição inicial* a ser utilizada na lista *recorrente* de endereços)
$ipbeg \leftarrow 1$ (a)

(para cada linha base a processar)
para i **de** 1 **até** n **faça**
  para cada *supernode contribuinte* sobre a linha base i **faça**
    ...
    (caso *numero de linhas* contribuintes justifique a utilização de *listas de endereços*)
    se ($lend - lbeg < nlmax$) **então** (b)
      (para cada *linha contribuinte* do *supernode corrente*)
      para l **de** $lbeg$ **até** $lend$ **faça**
        (posição *recorrente* inicial na lista de endereços sobre a *linha base*)
        $iplin \leftarrow ipbeg$ (c)
        (acumulo das contribuições da linha l *diretamente* sobre a linha base)
        para j **de IUP**[l] **até IUF**[l] **faça**
          UN[**PLIN**[$iplin$]] $\leftarrow$ **UN**[**PLIN**[$iplin$]] $-$ **UN**[j] $*$ um (d)
          $iplin \leftarrow iplin + 1$
        fim para l
        (*proxima* posição inicial na lista *recorrente* para o *supernode subseqüente*)
        $ipbeg \leftarrow iplin$ (e)
    fim se ($lend - lbeg < nlmax$)
    ...
  fim para cada *supernode contribuinte*
fim para i

Alt 4.2(b): *Contribuições Supernodais via Listas Recorrentes*

No passo (c) inicializa-se o indicador da posição recorrente inicial na lista de endereços (a ser re-utilizada no processamento das contribuições de cada uma das linhas do *mesmo supernode*).

No passo (d) processam-se as contribuições, de forma análoga aos procedimentos simbólicos apresentados no início deste capítulo, incrementando-se o indexador *iplin* a cada acesso a lista simbólica **PLIN**.

Finalmente no passo (e) aproveita-se do fato de *iplin* já ter sido incrementado para uma posição *além* da última utilizada no processamento do *supernode* corrente, re-inicializando-se desta forma o indicador *ipbeg* para a nova posicão inicial (a ser utilizada no processamento do próximo *supernode* a satisfazer a condição do passo (b)).

□

A técnica apresentada, por explorar padrões *recorrentes* de acesso, viabiliza uma utilização "controlada" de listas de endereços para *pequenos casos*, consistindo portanto num critério alternativo de "truncamento", além dos sugeridos por Resende e

Veiga [1] ou por [42] e que não se mostram efetivos na presença de padrões predominantemente *supernodais* de modo mais geral (além do caso particular de sub-matrizes *quase densas* ao final do processo, amplamente considerado pela literatura).

No próximo capítulo, técnicas mais eficazes explorando novas *recorrências* e a compactação de informações simbólicas serão apresentadas.

Substituições Supernodais

Do mesmo modo como se mostra vantajoso sob o ponto de vista computacional, a exploração de padrões *supernodais* durante a fase numérica do processo de eliminações, uma extensão de tais técnicas para a fase de *substituições de variáveis* também é merecedora de atenção.

Apresenta-se pois a seguir, no Procedimento **4.2(3)** uma versão *supernodal* para os processos de substituições *forward, diagonal* e *backward.*

Descrição de Passos do Procedimento **4.2(3)**

No passo (a) do Procedimento **4.2(3)** inicializa-se o vetor solução x com as componentes do vetor lado direito b a serem transformadas (nas subseqüentes fases do processo de substituições).

No passo (b) inicia-se o processamento das substituições *forward* e *diagonal* para cada um dos supernodes no qual foi particionada a matriz de fatores.

No passo (c) associa-se a variável i o índice da primeira linha do *supernode* corrente.

No passo (d) verifica-se se o *supernode* corrente é constituído por apenas uma única linha.

Em caso afirmativo, no passo (e) processa-se uma acumulação tradicional (análoga a do Procedimento **3.4(3)**) subtraindo-se múltiplos escalares de x_i de cada uma das variáveis $x_{\mathrm{JU}[.]}$ associadas as linhas de U^T possuindo coeficiente não nulo na coluna i.

Caso o *supernode* corrente possua mais de uma linha, processa-se (a partir do passo (f) até o passo (m)) uma acumulação *supernodal*, explorando-se a seqüencialidade dos índices das variáveis x envolvidas.

Assim, inicialmente no passo (f) armazena-se em ji um *offset* relativo a posição do último elemento não nulo da linha i, de modo a que o mesmo seja mapeado na primeira posição do vetor *supernodal* $\mathbf{V}$.

No passo (g) inicializa-se o vetor *supernodal*, armazenando-se contíguamente as componentes de x correspondentes aos índices das linhas com elementos não nulos da i'ezima coluna de U^T. O mapeamento adotado neste passo (e ao longo de todo o restante do processo), é da forma "reversa", levando a última componente de x considerada, para a primeira posição do vetor *supernodal.*

Procedimento 4.2(3) *Substituições Supernodais*

```
( inicialização do vetor solução com as componentes do lado direito )
para i de 1 até n faça                                                    (a)
    X[i] ← B[i]

( Substituições Forward e Diagonal )

( para cada supernode ii a processar )
para ii de 1 até nn faça                                                  (b)
    i ← INOD[ii]                                                          (c)
    se ( INODF[ii] = INOD[ii] ) então                                     (d)
        xi ← X[i]
        para j de IU[i] até IUF[i] faça                                   (e)
            X[JU[j]] ← X[JU[j]] − UN[j] * xi
        X[i] ← xi * DI[i]
    senão
        ji ← 1+IUF[i]                                                     (f)
        para j de IU[i] até IUF[i] faça                                   (g)
            V[ji − j] ← X[JU[j]]
        para i de INOD[ii] até INODF[ii] − 1 faça                         (h)
            ji ← 1+IUF[i]
            xi ← XI[i]
            para j de IU[i] até IUF[i] faça                               (i)
                V[ji − j] ← V[ji − j] − UN[j] * xi
            X[i] ← xi * DI[i]                                             (j)
            X[i + 1] ← V[ji − IU[i]]                                      (k)
        fim para i
        i ← INODF[ii]                                                     (l)
        ji ← 1+IUF[i]
        xi ← XI[i]
        para j de IU[i] até IUF[i] faça                                   (m)
            X[JU[j]] ← V[ji − j] − UN[j] * xi
        X[i] ← xi * DI[i]
    fim se ( INODF[ii] = INOD[ii] )
fim para ii

( Substituição Backward )

( para cada supernode ii a processar )
para ii de nn de volta até 1 faça                                         (n)
    i ← INOD[ii]
    se ( INODF[ii] = INOD[ii] ) então                                     (o)
        xi ← X[i]
        para j de IUF[i] de volta até IU[i] faça                          (p)
            xi ← xi − UN[j] * X[JU[j]]
        X[i] ← xi
    senão
        ji ← 1+IUF[i]
        para j de IUF[i] de volta até IU[i] faça                          (q)
            V[ji − j] ← X[JU[j]]
        para i de INODF[ii] de volta até INOD[ii] faça                    (r)
            ji ← 1+IUF[i]
            xi ← XI[i]
            para j de IUF[i] de volta até IU[i] faça                      (s)
                xi ← xi − UN[j] * V[ji − j]
            X[i] ← xi
            V[ji − (IU[i] − 1)] ← X[i]                                    (t)
        fim para i
    fim se ( INODF[ii] = INOD[ii] )
fim para ii
```

Proc 4.2(3): *Substituições Supernodais*

No passo (h) inicia-se o processo de substituição *supernodal forward* propriamente dita, excluindo-se do tratamento neste *loop*, a última linha do *supernode* corrente (a ser especialmente tratada nos passos (l) e (m)).

No passo (i) processa-se o acumulo dos múltiplos escalares da variável x_i, efetuados de forma compacta, diretamente sobre as posições do vetor *supernodal* **V**, em função da identicidade do padrão estrutural de elementos não nulos, de cada uma das colunas de U^T associadas ao *supernode* corrente.

No passo (j) normaliza-se a componente x_i, implementando-se a fase de *substituição diagonal.*

No passo (k) por estar se operando com a estrutura de um mesmo *supernode*, sabe-se por construção, da inevitável presença de um padrão bloco-diagonal na matriz de fatores. Deste modo, ao se completar o processamento relativo a i'ezima coluna de U^T, transfere-se para a componente de x com índice imediatamente consecutivo, o valor correspondente a última posição correntemente operada no vetor *supernodal* (e que não mais será acessada de forma contígua a partir da próxima instância do passo (i)). Na verdade, o valor transferido para a componente x_{i+1} corresponderá particularmente ao fator multiplicativo xi a ser utilizado na próxima iteração do *loop* do passo (h).

A partir do passo (l) até o passo (m), opera-se isoladamente com a última linha do *supernode* corrente.

No passo (l) o índice da última linha do *supernode* é associado a variável i.

No passo (m) processa-se o acumulo dos múltiplos escalares relativos a última coluna de elementos não nulos de U^T associada a *supernode* corrente. Este processamento é feito de forma contígua como nas demais instâncias do passo (i), porém por se tratar da linha terminal do *supernode*, os valores anteriormente agregados a **V** são deste modo transferidos para as componentes do vetor x correspondentes, explicando-se desta forma a razão para um tratamento isolado das demais componentes do mesmo *supernode.*

A partir do passo (n) inicia-se o processamento da fase de *substituições backward*, reversamente segundo o índice de cada um dos *supernodes* no qual foi particionada a matriz de fatores.

No passo (o) identifica-se se o *supernode* corrente é constituído por apenas uma única linha.

Em caso afirmativo, no passo (p) processam-se as retro-substituições de forma tradicional (como no Procedimento **3.4(3)**) subtraindo-se múltiplos escalares das variáveis $x_{\mathbf{JU}[.]}$ associadas a cada um dos coeficientes não nulos da i'ezima linha. Por já se ter processado préviamente a fase de normalização *diagonal*, os coeficientes $u_{i,i}$ serão desta forma unitários por construção, e uma vez completada a subtração de todos os múltiplos escalares envolvidos, o valor obtido corresponderá definitivamente ao da componente x_i associada.

No caso do *supernode* corrente possuir mais de uma linha, processa-se desde o passo (q) até o passo (t), uma fase *supernodal* de *substituições backward.*

No passo (q) (análogamente ao passo (g)) inicializa-se o vetor *supernodal* (de forma reversa) com os valores das componentes de x associadas aos coeficientes não nulos da i'ezima linha da matriz de fatores U.

No passo (r) inicia-se o processamento para cada uma das linhas constituintes do *supernode* corrente.

No passo (s) processa-se o acumulo dos múltiplos escalares das variáveis correspondentes aos elementos não nulos da linha corrente, armazenados contíguamente no vetor *supernodal.*

Finalmente no passo (t) processa-se a incorporação ao vetor *supernodal*, do valor correntemente gerado da componente do vetor solução x, a ser futuramente acessado de forma seqüencial contígua, juntamente com as demais componentes do mesmo *supernode* geradas, em instâncias futuras do passo (s).

□

Considerações Finais sobre a Exploração de Padrões Supernodais

Dentre as formas de aprimoramento consideradas até hoje para a resolução de sistemas esparsos, a exploração de características *supernodais* é seguramente uma das que obteve maior sucesso, por justamente preencher *simultâneamente* lacunas vitais como a de redução no espaço de armazenamento e propiciar aumentos no desempenho em tempo de execução (pela redução significativa de *overheads* de endereçamento nos *loops* mais internos).

Algumas extensões e refinamentos das técnicas apresentadas nesta seção são possíveis, como por exemplo a adoção de uma representação *"Compacta"* para a Árvore de Eliminação, baseada na noção de *supernodes Fundamentais* [8]. Uma forma de se aumentar a eficiência dos procedimentos *supernodais* é lançar mão de códigos específicamente otimizados para o processamento de operações matriciais *densas*, transformando-se as operações de contribuições sobre vetores *supernodais* em chamadas a sub-rotinas dedicadas como as do tipo *BLAS* [82], [66]. A plena exploração de meios secundários de armazenamento como memórias *cache* também responde por boa parte dos aprimoramentos passíveis de incorporação e exploração (especialmente nas *workstations superescalares* atuais), como em [91], [89],[90], [82] e [66].

A determinação da estrutura de *supernodes* e da *árvore de eliminação* de forma modularizada e mais eficiente do que a apresentada nesta seção, vem merecendo também especial atenção a partir de trabalhos como [74], [77], [52] e [8].

Um último comentário a se fazer sobre a exploração de padrões *Supernodais* é que sob certos aspectos tal abordagem permite se extrair uma real medida do *grau de esparsidade* de uma matriz, quantificável não apenas por sua densidade ou número de elementos não nulos, mas também pela *quantidade de informação* necessária para a *caracterização de sua estrutura.*

Percebe-se desse modo que uma matriz *esparsa* nada mais é do que um *agregado* de *sub-matrizes densas* (correspondentes a cada um de seus *supernodes* constituin-

tes).

Sob este aspecto, matrizes completamente densas ou completamente esparsas (na forma diagonal) requerem uma mesma e única quantidade de informação, qual seja, apenas a de sua dimensão, para sua completa caracterização. Matrizes quase densas ou quase esparsas por sua vez, requerem uma quantidade reduzida de informação, em comparação com matrizes de densidade intermediária e que não possuam significativos padrões estruturais similares em comum.

O que se encontra na maior parte das vezes na prática, são matrizes em algum dos dois extremos, como as de Potência (extremamente esparsas, com baixa densidade e quase nenhuma similaridade estrutural), ou matrizes com porções significativamente densas, e elevada similaridade estrtutural, decorrente da presença maçica de padrões *supernodais.*

O caso intermediário é o de mais difícil tratamento, e onde até então apenas as técnicas baseadas em vetores de trabalho expandidos foram consideradas pela literatura.

Os novos métodos propostos no capítulo 5, vem preencher esta lacuna, por serem indistintamente aplicáveis (de forma unificada) a matrizes de qualquer padrão estrutural, sem para tal necessitar recorrer ao uso de estruturas intermediárias como as de vetores de trabalho, além de permitir a elaboração de estratégias mais eficientes de implementação da fase numérica do processo de fatoração (e de substituições).

4.3 Metodologias Híbridas por Janelas de Eliminação

Uma outra forma de se aumentar o desempenho de procedimentos de fatoração esparsa, sem no entanto elevar-se consideravelmente o dispêndio de recursos auxiliares de memória, pode ser obtido mediante a utilização de esquemas "híbridos" como [20], [1], [32] e [5], adotando-se a noção de *Janelas de Eliminação* como em [1], [5] e [6].

Estes esquemas baseiam-se no fato de que determinadas porções da matriz de fatores tendem normalmente a apresentar padrões tipicamente reconhecíveis (e similares), como no caso de sub-matrizes densas ao final do processo de eliminação.

Outros padrões de fácil reconhecimento incluem os de forma completamente diagonal, presente muitas das vezes na porção inicial superior (especialmente em matrizes ordenadas pelos critérios de menor grau ou menor *fill-in* local).

Na Figura 4.5 apresenta-se uma matriz de fatores como exemplo, onde cada um dos símbolos utilizados denota a localização de coeficientes não nulos, para cada um dos respectivos tipos de janela considerados.

Na porção representada por "$\triangle$" e delimitada desde $u_{1,1}$ até $u_{4,4}$, percebe-se a existência de uma janela "diagonal".

Na porção representada por "$\square$" e delimitada desde $u_{5,5}$ até $u_{8,15}$, encontra-se uma janela "supernodal" (como pode ser notado pela semelhança estrutural de suas

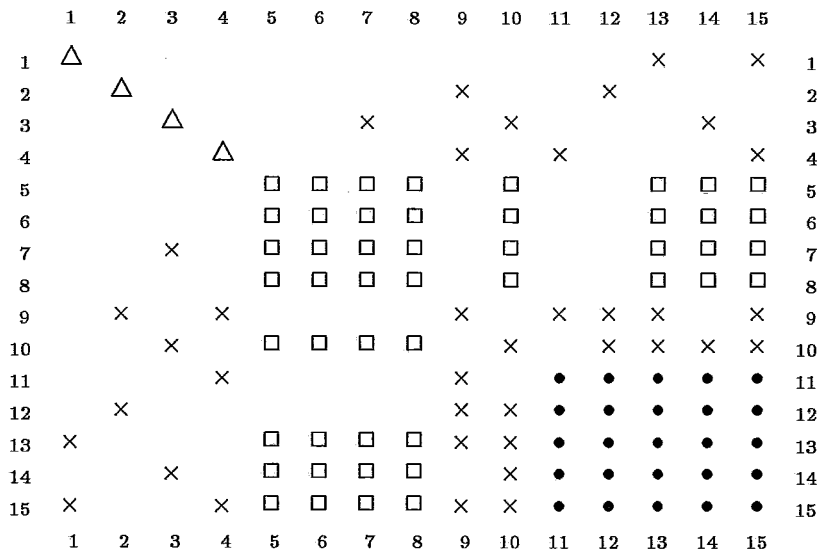


Fig 4.5: Janelas de Eliminação

linhas componentes).

Na porções representadas por "×" e delimitadas desde $u_{1,7}$ até $u_{4,15}$ e de $u_{9,9}$ até $u_{10,15}$ associam-se janelas do tipo "convencional" (por não possuírem nenhum padrão característico comum entre seus elementos).

Finalmente na porção representada por "•" e delimitada desde $u_{11,11}$ até $u_{15,15}$ percebe-se a existência de uma janela do tipo "completamente densa".

A noção de *Janelas de Eliminação* (doravante denominadas simplesmente por *Janelas*) pode ser formalizada a partir de:

Definição 4.3.1 (Janela de Eliminação) *Define-se por Janela de Eliminação (para procedimentos de fatoração esparsa por linhas), como a região delimitada por um conjunto de linhas e colunas contíguas (da matriz de fatores), a ser tratada de forma unificada por uma mesma estratégia de implementação, e a sofrer contribuição de um selecionado grupo de linhas incidentes sobre seus elementos.*

Esta noção é mais ampla que as de [1] e [5], por especificar não apenas uma determinada porção física de elementos da matriz de fatores, como também um particular grupo de linhas incidentes, a contribuírem sobre os elementos confinados à região selecionada.

Na verdade, a caracterização de *Janela* (introduzida em [6]) adotada no presente trabalho, corresponde a um particionamento ao nível da seqüência de operações de cancelamento (a serem aplicadas selecionadamente sobre particulares etapas do processo de eliminação).

Para a completa especificação do escopo de abrangência de uma *Janela* necessita-se pois das seguintes informações:

- *Linha Base Inicial* de U (a ser parcial ou totalmente gerada)
- *Linha Base Final* de U (a ser parcial ou totalmente gerada)

- *Linha Contribuinte Inicial* (a ser considerada na geração das linhas *base*)
- *Linha Contribuinte Final* (a ser considerada na geração das linhas *base*)

A outra informação relevante e específica à cada janela, é o seu *Tipo* de processamento associado, e que pode ser sub-dividido nas seguintes classes:

- *Diagonal*
- *Convencional* (via Vetor de Trabalho) com geração de fatores por *Linhas.*
- *Convencional* (via Vetor de Trabalho) com geração de fatores por *Colunas.*
- *Simbólico* (via Lista de Endereços) com geração de fatores por *Linhas.*
- *Simbólico* (via Lista de Endereços) com geração de fatores por *Colunas.*
- *Supernodal* com geração dos fatores por *Linhas*
- *Supernodal* com geração dos fatores por *Colunas*
- *Completamente Denso* com geração dos fatores por *Sub-Matrizes*

Deste modo o processo de eliminação como um todo, pode ser sub-dividido em seqüências de operações envolvendo cada uma das classes de janelas, como nos mostra o Algoritmo **4.3**, onde *iwind* denota o *Índice* da janela a ser correntemente tratada, **WINTYP** o seu *Tipo* de processamento, **WINROW**, **WINROWF**, **CONTRB** e **CONTRBF** delimitam respectivamente do escopo de abrangência das linhas *Base* e *Contribuintes* associadas, e DIAGONAL, WORKROW, WORKCOL, SIMBROW, SIMBCOL, SNODROW, SNODCOL e DENSMAT correspondem a *Procedimentos de Fatoração Numérica* particularmente voltados para cada tipo de janela.

Para a matriz de fatores utilizada como exemplo na Figura 4.5 apresenta-se a seguir (na Tabela 4.1), um possível particionamento por *Janelas*, fornecendo-se a seqüência de índices delimitadores $(ibeg, iend)$, $(lbeg, lend)$ das linhas *Base* e *Contribuintes* a serem consideradas.

iwind	*tipo*	*ibeg*	*iend*	*lbeg*	*lend*
1	△	1	4	1	4
2	×	7	15	1	4
3	□	5	15	5	8
4	×	11	15	9	10
5	•	11	15	11	15

Tab 4.1: Particionamento por Janelas de Eliminação

Algoritmo 4.3 *Eliminação Híbrida (Janelas)*

(para cada uma das *Janelas de Eliminação* consideradas)

para *iwind* **de** 1 **até** *nwind* **faça**

ibeg ← **WINROW**[*iwind*]
iend ← **WINROWF**[*iwind*]
lbeg ← **CONTRB**[*iwind*]
lend ← **CONTRBF**[*iwind*]

(em funçao do *Tipo* da janela corrente processe a eliminação apropriada)

caso WINTYP[*iwind*] **seja**

1: **processe** DIAGONAL (*ibeg*, *iend*, *lbeg*, *lend*)
2: **processe** WORKROW (*ibeg*, *iend*, *lbeg*, *lend*)
3: **processe** WORKCOL (*ibeg*, *iend*, *lbeg*, *lend*)
4: **processe** SIMBROW (*ibeg*, *iend*, *lbeg*, *lend*)
5: **processe** SIMBCOL (*ibeg*, *iend*, *lbeg*, *lend*)
6: **processe** SNODROW (*ibeg*, *iend*, *lbeg*, *lend*)
7: **processe** SNODCOL (*ibeg*, *iend*, *lbeg*, *lend*)
8: **processe** DENSMAT (*ibeg*, *iend*, *lbeg*, *lend*)

fim caso WINTYP[*iwind*]

fim para *iwind*

Alg 4.3: *Eliminação Híbrida (Janelas)*

O critério adotado para este particionamento, foi obtido simplesmente tomando-se os *supernodes* da matriz de fatores resultante, como delimitadores entre os sucessivos grupos de linhas Contribuintes de cada janela. No exemplo em questão, tal correspondeu particularmente aos intervalos $(1,4)$, $(5,8)$, $(9,10)$, $(11,15)$.

Para cada grupo de linhas Contribuintes (confinadas a um dado intervalo de colunas de U^T) tomou-se como o escopo de abrangência de cada janela, o maior intervalo de linhas Base afetadas por tais contribuições.

Observa-se no entanto, que os esquemas de eliminação por janelas oferecem um amplo espectro de opções no que tange aos critérios de particionamento a serem adotados.

O exemplo apresentado foi apenas uma destas formas, a ser considerada mais detalhadamente no próximo capítulo (ao se apresentar as novas metodologias propostas neste trabalho).

Outra forma de se particionar o mesmo problema, correspondendo a um processamento análogo ao efetuado no Procedimento **4.2(1)** de eliminação *Supernodal* por *Linhas* (apresentado na seção anterior), é enumerada a título de ilustração, na Tabela 4.2 a seguir.

iwind	*tipo*	*ibeg*	*iend*	*lbeg*	*lend*
1	△	1	4	1	4
2	□	5	6	5	6
3	×	7	7	3	3
4	■	7	8	5	8
5	×	9	10	2	4
6	□	10	10	5	8
7	×	11	12	2	10
8	□	12	12	11	12
9	×	13	13	1	1
10	□	13	13	5	8
11	×	13	13	9	10
12	□	13	13	11	13
13	×	14	14	3	3
14	□	14	14	5	8
15	×	14	14	10	10
16	□	14	14	11	14
17	×	15	15	1	4
18	□	15	15	5	8
19	×	15	15	9	10
20	□	15	15	11	15

Tab 4.2: Particionamento Alternativo por Janelas de Eliminação

Neste caso percebe-se que o número de janelas foi consideravelmente maior que o do particionamento anteriormente apresentado (superando inclusive a dimensão do problema), em função de estar-se "simulando" toda a seqüência de cancelamentos

efetuadas no procedimento de geração Supernodal por Linhas (onde cada linha Base da matriz de fatores resultante é tratada e gerada isoladamente das demais). A razão para o número de janelas exceder o número de linhas do problema, deve-se a presença de mais de um tipo janela de contribuição em boa parte das linhas Base sendo geradas.

Na prática, esta forma de particionamento não é adotada, por poder-se lançar mão de estratégias bem mais eficientes e econômicas, como as apresentadas no exemplo anterior.

Em todos os métodos considerados até o momento, adotou-se sempre como base, procedimentos de eliminação numérica em que o processamento das contribuições de linhas antecessoras (quer na geração de fatores por Colunas, quer na geração por Linhas) foi sempre efetuado sistemática e isoladamente, linha por linha (contribuição à contribuição), aproveitando-se das seqüencialidades de armazenamento, para o acesso aos elementos contribuintes.

A partir do próximo capítulo, considerar-se-á uma outra forma de processamento e acúmulo de contribuições, gerando-se os fatores seqüencialmente a cada linha, porém acessando a estrutura de elementos contribuintes por *Colunas* (lançando mão de esquemas de codificação, espelhando os padrões estruturais a serem operados a cada etapa).

Capítulo 5

Nova Abordagem Proposta

Neste capítulo apresenta-se uma extensão das metodologias propostas pelo autor em [6].

Tais métodos baseiam-se na utilização de *Listas de Códigos* (a serem introduzidas mais adiante), no lugar de Vetores de Trabalho ou *Listas de Endereços*, usualmente adotadas para o acúmulo de contribuições sobre a estrutura base a cada etapa.

A metodologia proposta, pode ser caracterizada como uma alternativa viável em comparação com métodos *simbólicos* Completamente *"Loop-Free"* como os de Gustavson [60].

Os novos métodos apresentados possuem desempenho em tempo de execução equiparável ao destas metodologias, mantendo no entanto (em algumas das variantes apresentadas), um espaço de trabalho proporcional apenas ao volume de dados (no lugar de um volume proporcional ao número de operações de ponto flutuante, como no caso de Gustavson).

5.1 Princípios Básicos e Motivação Computacional

A base para a formulação das novas metodologias foi obtida a partir de uma variante dos métodos de eliminação direta, conhecida na literatura como *Método de Crout* (originalmente desenvolvido para o caso assimétrico e trivialmente extensível ao caso simétrico).

O método de Crout Simétrico, apresentado na Alternativa **5.1(a)** (e exemplificado ao final do apêndice B) nada mais é do que uma forma alternativa de implementação de um processo de eliminação (triangular inferior) por linhas, com geração (triangular superior) por *linhas*, como 2.4(b), simplesmente revertendo-se a ordem de execução dos dois *loops* mais internos.

Uma diferença básica com relação a todos os métodos até então considerados, faz-se notar, visto que no método de Crout o *loop* mais interno é baseado em acumulações na forma de produtos escalares (no lugar de combinações lineares sobre vetores esparsos da forma $\mathbf{u} \leftarrow \mathbf{u} + \alpha\mathbf{v}$, como nos demais métodos de geração por linhas).

O método de Crout baseia-se numa geração por *linhas*, porém acumulando-se

Alternativa 5.1(a) *Fatoração $U^T D U$ (Crout) por Linhas*

para i **de** 1 **até** n **faça**
  para k **de** 1 **até** $i-1$ **faça**
    $u_{k,i} \leftarrow a_{k,i} \ / \ d_{k,k}$
  para j **de** i **até** n **faça**
    $a_{i,j} \leftarrow a_{i,j} - \sum_{k=1}^{i-1} u_{k,i} * a_{k,j}$
  $d_{i,i} \leftarrow a_{i,i}$

Alt 5.1(a): *Fatoração $U^T D U$ (Crout) por Linhas*

as contribuições sobre cada elemento da linha base, *coluna* a *coluna*. Tal acúmulo é processado na forma do produto escalar dos elementos sobre a coluna base de U, pelos da coluna do elemento sendo correntemente gerado na linha base.

Em função de requerer acessos por *colunas*, a estratégia de Crout, praticamente não veio sendo empregada na solução de sistemas esparsos (com o armazenamento dos fatores superiores por *linhas*), visto introduzir *overheads* adicionais na varrerdura de cada linha (de modo a se determinar a contribuição relativa à coluna do elemento correntemente sendo gerado na linha base).

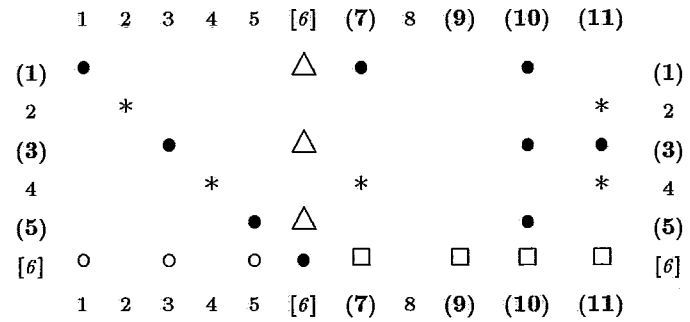
A única outra abordagem relativamente bem sucedida, baseada num método do tipo Crout para o caso esparso, foi a de Gustavson [60]. (Tal abordagem porém, apresentou como inconveniente o tamanho excessivo dos subprogramas gerados, inviabilizando-a para problemas de médio a grande porte, mesmo nas arquiteturas atuais).

Os novos métodos a serem considerados, conseguem contornar tais dificuldades, mediante a introdução do conceito de *Listas de Códigos* a ser explorado ao longo de todo este capítulo.

Codificação Binária

Apresenta-se a seguir uma técnica básica, a ser adotada em alguns dos procedimentos, para a implementação de métodos do tipo Crout (esparsos).

Tomemos como exemplo o seguinte padrão estrutural de contribuições sobre a linha base 6 (onde cada um dos símbolos denota a presença de um elemento não nulo na posição triangular superior associada).



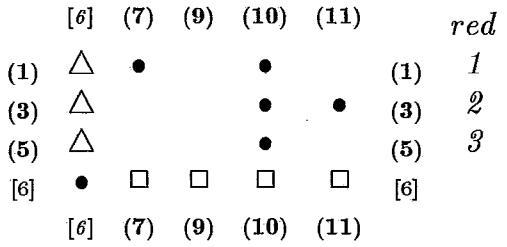
Percebe-se que as contribuições sobre a linha 6, (excluindo-se as sobre a diagonal) correspondem particularmente a:

$$\begin{array}{lllllllll} u_{6,7} & \leftarrow & u_{6,7} & - & u_{1,7} * um_1 & & & & \\ u_{6,10} & \leftarrow & u_{6,10} & - & u_{1,10} * um_1 & - & u_{3,10} * um_3 & - & u_{5,10} * um_5 \\ u_{6,11} & \leftarrow & u_{6,11} & & & - & u_{3,11} * um_3 & & \end{array} \tag{5.1}$$

Nota-se que por estar se processando operações envolvendo múltiplas linhas contribuintes sobre o elemento base de cada coluna, torna-se necessário a especificação de constantes multiplicativas (normalizadoras) um_1, um_2, um_3 no lugar de apenas uma única constante um como ocorria no caso do processamento tracional mediante varreduras linha por linha.

A solução para o problema dos acessos por coluna, consiste em notar que caso se disponha de alguma informação que caracterize completamente o padrão estrutural para cada coluna de elementos contribuintes, é possível construir-se um procedimento dedicado, de modo a tratar particularmente o caso de cada coluna.

Para o exemplo em questão, tomemos particularmente apenas um sub-conjunto *reduzido* das linhas contribuintes sobre a linha 6 (e das colunas correspondentes a elementos não nulos sobre a mesma), como apresentado a seguir.



Nota-se que a única informação adicional necessária para a caracterização das operações 5.1 é o padrão de elementos não nulos de cada coluna.

Uma forma de se processar tal caracterização, é lançar-se mão de uma representação binária, como a apresentada a seguir.

	[6]	(7)	(9)	(10)	(11)		*red*
(1)	△	1	0	1	0	(1)	*1*
(3)	△	0	0	1	1	(3)	*2*
(5)	△	0	0	1	0	(5)	*3*
[6]	•	□	□	□	□	[6]	
	[6]	(7)	(9)	(10)	(11)		
bin		001	000	111	010		
cod		1	0	7	2		

Nota-se para o exemplo em questão, que o padrão de contribuições sobre cada coluna da linha base, pode ser completamente especificado, mediante o conjunto de *códigos* binários $\{ 001, 000, 111, 010 \}$, onde tomou-se por convenção, uma representação binária crescente em função do índice da linha (*reduzida*) contribuinte (correspondendo à de menor índice, ao bit menos significativo da representação).

Uma vez de posse da codificação característica do padrão de contribuições sobre cada coluna, o que se mostra necessário para a implementação da eliminação numérica é um procedimento de decodificação, capaz de efetuar as operações 5.1.

Tal é apresentado na Codificação 5.1-1, voltada particularmente para a acumulação de produtos escalares de até 3 linhas simultâneas.

Nesta codificação, assume-se que um procedimento principal (a ser apresentado mais adiante), venha a se utilizar de desvios (computados segundo cada código), para o trecho de programa correspondente a operação de produto escalar esparso, a ser efetuada a cada coluna.

É assumido também, que ao se iniciar o processamento de cada etapa básica, os apontadores p_1, p_2, p_3 tenham sido originalmente inicializados de modo a apontar para a posição correspondente a primeira contribuição de cada uma das linhas (de forma análoga a adotada para os apontadores dinâmicos **IUP**[.] considerados nos capítulos anteriores deste trabalho).

Uma vez acessadas as posições correntemente apontadas por p_1, p_2, p_3, incrementam-se tais apontadores, de modo a que por ocasião das próximas acumulações, o elemento não nulo seqüencialmente consecutivo (de cada linha) possa ser acessado durante o processamento dedicado a sua coluna correspondente.

Nota-se que o índice associado a cada um destes apontadores, é na verdade um *índice reduzido*, ou seja, correspondendo a posição relativa de cada linha no subconjunto de linhas efetivamente contribuintes sobre a linha base. Para o exemplo em questão, (correspondendo a sexta etapa do processamento), os apontadores p_1, p_2, p_3 encontram-se pois físicamente associados as linhas 1, 3, 5 da matriz de fatores original.

Um outro ponto a se considerar é o caso de uma linha base depender de um número de linhas contribuintes superior ao número máximo de *bits* da codificação empregada. Neste caso, a solução consiste em se aplicar mais de uma varredura sobre a linha base, com cada grupo de códigos correspondendo a um processamento de até n_{bits} linhas contribuintes, simultâneamente de cada vez.

Codificação 5.1-1 *Produto Escalar via Codificação Binária*

0 : **retorne ao processamento principal**

1 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1$
retorne ao processamento principal

2 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2]$
$p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

3 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

4 : $s \leftarrow s + um_3 * \mathbf{UN}[p_3]$
$p_3 \leftarrow p_3 + 1$
retorne ao processamento principal

5 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_3 * \mathbf{UN}[p_3]$
$p_1 \leftarrow p_1 + 1,\ p_3 \leftarrow p_3 + 1$
retorne ao processamento principal

6 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
$p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
retorne ao processamento principal

7 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
retorne ao processamento principal

Cod 5.1-1: *Produto Escalar via Codificação Binária*

Apresenta-se a seguir no Procedimento **5.1(1)** uma versão completa de um método de eliminação do tipo Crout, por acumulações mediante a técnica de *Codificação Binária.*

Para fins de exemplificação, adotou-se uma versão dedicada para códigos de até apenas 3 bits. (Na prática, versões com 8 ou mais bits devem ser efetivamente aplicadas para que se possa explorar plenamente a capacidade de armazenamento por cada *byte* ou *palavra* de código).

Descrição de Passos do Procedimento **5.1(1)**

No passo (a) do Procedimento **5.1(1)**, inicializa-se o indexador de posições para a *lista de códigos binários.*

No passo (b) inicializam-se os delimitadores *kbeg* e kf dos valores extremos a serem assumidos pela variável indexadora k ao longo de toda a i'ezima etapa corrente.

No passo (c) armazena-se em *kend* o valor máximo a ser correntemente assumido pelo índice k na atual varredura sobre a linha base. Tal valor garante que no máximo n_{bits} linhas contribuintes sejam processadas de cada vez.

No passo (d) incrementa-se o índice *reduzido* r, armazena-se em l o índice da atual linha contribuinte, e em *iupl* a posição na linha l correpondente a coluna do elemento diagonal básico.

No passo (e) incrementa-se o apontador dinâmico $\mathbf{IUP}[l]$, inicializando-se o apontador $p_{(r)}$ para a posição da primeira contribuição da linha *reduzida* r sobre a linha base.

No passo (f) armazena-se em $um_{(r)}$ o fator multiplicativo (normalizador), correspondente a linha *reduzida* corrente a contribuir sobre a linha base.

No passo (g) processam-se as acumulações (coluna a coluna) sobre cada um dos elementos não nulos da linha base.

No passo (h) inicializa-se o acumulador s das contribuições provenientes de produtos escalares, bem como incrementa-se a posição corrente na *lista de códigos binários.*

No passo (i) desvia-se para o trecho de código correspondente a acumulação por produto escalar, dedicada ao processamento do padrão de contribuições sobre a coluna corrente, na linha base.

No passo (j) incorpora-se o resultado da acumulação de tais contribuições, sobre o elemento correspondente na linha base.

No passo (k) atualiza-se o apontador para a próxima posição inicial dos índices k, para uma posição consecutiva a última utilizada na varredura atual da linha base corrente.

No passo (l), no caso de o número de varreduras necessárias, ainda não ter se esgotado, repete-se o processamento a partir do trecho α do procedimento em questão.

□

Procedimento 5.1(1) *Eliminação via Códigos Binários (Crout)*

($n_{bits} = 3$)

$ibin \leftarrow 0$ (a)
para i **de** 1 **até** n **faça**
  $piv \leftarrow \mathbf{DI}[i]$, $\mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$
  $kbeg \leftarrow \mathbf{IUT}[i]$, $kf \leftarrow \mathbf{IUTF}[i]$ (b)
α : $kend \leftarrow \min\{\ kbeg + n_{bits} - 1,\ kf\ \}$ (c)
  $r \leftarrow 0$
  para k **de** $kbeg$ **até** $kend$ **faça**
    $r \leftarrow r + 1$, $l \leftarrow \mathbf{JUT}[k]$, $iupl \leftarrow \mathbf{IUP}[l]$ (d)
    $\mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1$, $p_{(r)} \leftarrow \mathbf{IUP}[l]$ (e)
    $um_{(r)} \leftarrow \mathbf{UN}[iupl] * \mathbf{DI}[l]$ (f)
    $piv \leftarrow piv - \mathbf{UN}[iupl] * um_{(r)}$
    $\mathbf{UN}[iupl] \leftarrow um_{(r)}$
  fim para k
  (acumulação das contribuições sobre a *linha base*, coluna a coluna)

---

  para j **de** $\mathbf{IU}[i]$ **até** $\mathbf{IUF}[i]$ **faça** (g)
    $s \leftarrow 0.$, $ibin \leftarrow ibin + 1$ (h)
    va' para (0, 1, 2, 3, 4, 5, 6, 7) **em função de** $\mathbf{LBIN}[ibin]$ (i)
β : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - s$ (j)

  $kbeg \leftarrow kend + 1$ (k)
  se $kend \neq kf$ **va' para** α (l)
  $\mathbf{DI}[i] \leftarrow 1. \ / \ piv$
fim para i

termine procedimento

0 : **va' para** β
1 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1]$
  $p_1 \leftarrow p_1 + 1$
  va' para β
2 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2]$
  $p_2 \leftarrow p_2 + 1$
  va' para β
3 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2]$
  $p_1 \leftarrow p_1 + 1$, $p_2 \leftarrow p_2 + 1$
  va' para β
4 : $s \leftarrow s + um_3 * \mathbf{UN}[p_3]$
  $p_3 \leftarrow p_3 + 1$
  va' para β
5 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_3 * \mathbf{UN}[p_3]$
  $p_1 \leftarrow p_1 + 1$, $p_3 \leftarrow p_3 + 1$
  va' para β
6 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
  $p_2 \leftarrow p_2 + 1$, $p_3 \leftarrow p_3 + 1$
  va' para β
7 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
  $p_1 \leftarrow p_1 + 1$, $p_2 \leftarrow p_2 + 1$, $p_3 \leftarrow p_3 + 1$
  va' para β

Proc 5.1(1): *Eliminação via Códigos Binários (Crout)*

Apresenta-se a seguir, no Procedimento **5.1(2)** uma forma de geração da *codificação binária* característica de cada problema, partindo-se do conhecimento de sua estrutura triangular superior de fatores (determinada em alguma fase preliminar, como a de fatoração simbólica, apresentada na seção 3.3).

Este procedimento lança mão de um vetor auxiliar inteiro **JCOD** de dimensão n, a ser utilizado na acumulação dos bits contribuintes de cada um dos códigos sendo gerados. Este vetor auxiliar, desempenha uma função análoga a do vetor de trabalho expandido **W** (extensivamente empregado para o acúmulo de contribuições na fase numérica). Ao final de cada etapa, o padrão resultante de bits acumulado em **JCOD**, é transferido definitvamente para a lista de códigos **LBIN**.

Em face a analogia com o Procedimento **5.1(1)**, apenas os passos mais significativos do Procedimento **5.1(2)** serão detalhados a seguir.

Descrição de Passos do Procedimento **5.1(2)**

No passo (a) inicializa-se o indexador de posições geradas na lista de códigos binários.

No passo (b) anulam-se todas as posições em **JCOD** correspondentes a elementos não nulos sobre a linha base.

No passo (c) inicializa-se o valor da variável *cod*, de modo a indicar a primeira potência de 2 a ser adicionada aos códigos binários correspondentes a elementos não nulos da primeira linha *reduzida* a contribuir sobre a linha base.

No passo (d) processa-se o acúmulo em **JCOD** dos bits correspondentes aos elementos não nulos da linha l.

No passo (e) o valor de *cod* é dobrado, de modo a permitir o acúmulo dos bits referentes a próxima linha *reduzida* contribuinte sobre a linha base.

No passo (f) processa-se para cada um dos elementos não nulos da linha base, a transferência para a lista de códigos **LBIN**, dos valores acumulados nas colunas correspondentes de **JCOD**.

Finalmente, no passo (g), no caso da varredura das últimas n_{bits} linhas não ter sido suficiente para esgotar o total de linhas contribuintes sobre a linha base, retorna-se ao ponto α para uma nova varredura de até n_{bits} linhas (até se esgotarem todas as contribuições da i'ezima etapa).

□

5.2 Codificações Explorando Padrões por Envelope

A principal deficiência da estratégia de *codificação binária* apresentada na seção anterior, é o caso em que mais de uma varredura torna-se necessária de modo a cobrir o número total de linhas contribuintes a cada etapa.

Procedimento 5.1(2) *Geração da Codificação Binária*

$ibin \leftarrow 0$ (a)

para i **de** 1 **até** n **faça**

IUP[i] $\leftarrow$ **IU**[i]

para j **de** **IU**[i] **até** **IUF**[i] **faça** (b)
JCOD[**JU**[j]] $\leftarrow$ 0

$kbeg \leftarrow$ **IUT**[i], $kf \leftarrow$ **IUTF**[i]

α : $kend \leftarrow \min \{ kbeg + n_{bits} - 1,\ kf \}$

$cod \leftarrow 1$ (c)

para k **de** $kbeg$ **até** $kend$ **faça**

$l \leftarrow$ **JUT**[k], **IUP**[l] $\leftarrow$ **IUP**[l] + 1

para j **de** **IUP**[l] **até** **IUF**[l] **faça** (d)
JCOD[**JU**[j]] $\leftarrow$ **JCOD**[**JU**[j]] + cod

$cod \leftarrow cod * 2$ (e)

fim para k

para j **de** **IU**[i] **até** **IUF**[i] **faça** (f)
$ibin \leftarrow ibin + 1$
LBIN[$ibin$] $\leftarrow$ **JCOD**[**JU**[j]]

$kbeg \leftarrow kend + 1$

se $kend \neq kf$ **va' para** α (g)

fim para i

Proc 5.1(2): *Geração da Codificação Binária*

Em casos onde a presença de *supernodes* é significativa, os métodos baseados em codificações binárias requerem um espaço auxiliar proporcional ao número de operações aritméticas efetuadas, e análogamente ao esquema de Gustavson [59], nota-se que tal comportamento pode inviabilizar a aplicação desta técnica isolada em função do tamanho excessivo da lista de códigos gerada.

Nota-se no entanto, que em função do tamanho da lista de códigos ser proporcional ao número de operações, porém dividido por uma constante entre 8 a 10 (dependendo do número máximo de bits de codificação adotados), em alguns casos de matrizes com densidade intermediária, o espaço da lista de códigos pode vir a se mostrar aceitável, e num nível de proporcionalidade ao do número de seus fatores não nulos (multiplicado pelo número médio de varreduras necessárias por cada linha).

Esse é o caso de matrizes de Potência, onde normalmente o número de fatores não nulos por cada linha situa-se em geral abaixo de 10 elementos, permitindo que a técnica de codificação binária seja aplicada com sucesso, por dispender em média, apenas uma varredura por cada linha base gerada.

As técnicas apresentadas nesta seção visam amenizar o dispêndio de espaço adicional, mediante a compactação de informações passíveis de exploração, propiciando a utilização de duas novas formas de codificação, baseadas no conceito de *Envelopes*.

Reordenamento Ancestral de Contribuições

A técnica de *codificação binária*, parte de um princípio elementar, e que é o de explorar apenas a seqüência natural de disposição das informações a serem codificadas.

O exemplo ilustrado na Figura 5.1, mostra como em alguns casos, se pode buscar formas bem mais eficientes de se representar a estrutura de contribuições a cada coluna.

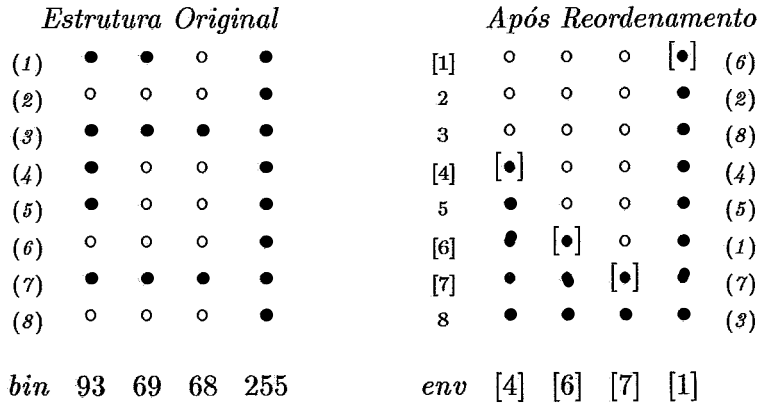


Fig 5.1: Reordenamento e Codificação por Envelopes

Nota-se que a disposição original (na ordem natural em que as linhas contribuintes foram enumeradas), requer inevitávelmente que 8 bits de código, sejam utilizados para a representação do padrão de contribuição de cada coluna.

Percebe-se no entanto, que se a primeira linha for permutada com a sexta, e a terceira com a oitava, a disposição estrutural após tal reordenamento permite se adotar uma outra forma de codificação para a caracterização dos padrões de contribuição.

Esta nova forma de codificação denominada por *Envelopes*, tem o seu nome decorrente dos métodos de eliminação utilizados normalmente na área de engenharia civil (e onde após a aplicação de técnicas visando a redução de banda, adota-se um esquema de processamento explorando a mesma informação básica empregada na codificação sendo proposta). Nota-se no entanto que o conceito de *envelopes* na forma de *codificações*, a ser empregado neste trabalho, foi originalmente introduzido pelo autor em [6], permanecendo desde então, inédito em toda a literatura.

Uma informação suficiente para a completa caracterização do padrão de contribuições, nos casos onde a cada coluna é notada a presença de elementos não nulos *somente a partir* de uma dada linha, (e em *todas as demais linhas* a ela subseqüentes), é simplesmente o índice desta primeira linha a partir da qual encontram-se confinadas todas as demais contribuições.

Para o exemplo em questão, percebe-se que o conjunto de índices $\{ 4, 6, 7, 1 \}$ é suficiente para se caracterizar plenamente do padrão de contribuições de cada uma das colunas associadas.

Nota-se neste caso (após o reordenamento, e com a utilização de códigos por *envelope*), que o número máximo de bits de codificação necessários para a representação de cada coluna, passou a ser de apenas 3 (em comparação com os 8 a 10 inevitávelmente presentes numa codificação do tipo *binário*).

Esta diferença faz-se notar mais acentuadamente, quão maior for o número de linhas contribuintes passíveis de caracterização pela técnica por *envelopes*. Num caso extremo, por cada *byte* de informação, é possível representar-se o padrão de até 256 linhas contribuintes na forma de *envelopes* (em comparação com apenas 8 no caso *binário*).

De posse de um mecanismo eficaz de codificação, como o apresentado, o que fica faltando mostrar, é como se determinar os casos onde a partir de reordenamentos das contribuições, é possível se alcançar padrões estruturais, na forma desejada.

Para tal, tomemos como exemplo a matriz de fatores da Figura 5.2, e sua Árvore de Eliminação ilustrada na Figura 5.3.

O que se irá explorar a seguir, é uma característica intrínseca aos métodos de eliminação direta, e que possuem uma natureza *estruturalmente aditiva* no que tange ao padrão de elementos não nulos resultantes do processo de contribuições a cada etapa.

Esta estrutura *aditiva* foi notada na seção 3.3, onde durante a fatoração simbólica, processa-se a união das estruturas contribuintes sobre cada linha base.

Percebe-se que o padrão estrutural de uma dada linha, é particularmente determinado pelo de suas linhas contribuintes (antecessoras na árvore).

Este padrão estrutural (agregado a linha base gerada), será integralmente incorporado a todas as suas sucessoras na árvores que porventura venham a depender de

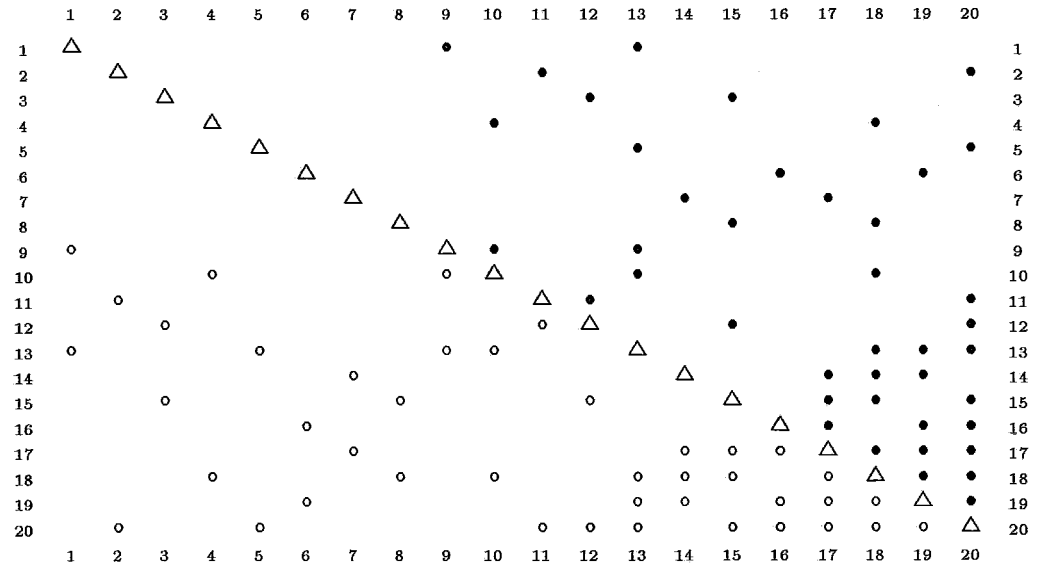


Fig 5.2: Matriz de Fatores

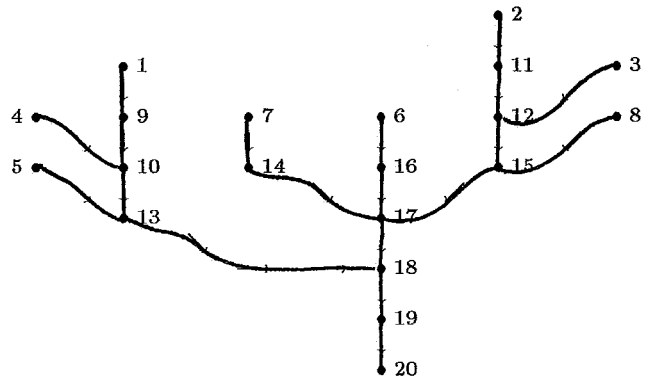


Fig 5.3: Árvore de Eliminação

sua contribuição.

Desta forma, nota-se que impondo-se uma ordem de contribuições, partindo-se de um nó mais remotamente ancestral e seguindo-se por um dado *ramo* na árvore de eliminação, explorando-se a hierarquia de parentesco entre cada uma das linhas envolvidas, o padrão estrutural *reduzido* resultante (confinado apenas as linhas envolvidas), corresponderá exatamente a forma desejada para uma codificação por *envelopes.*

Como exemplo ilustrando este fato, tomemos a linha 2, e a estrutura de todas as suas sucessoras, a partir de um ramo da árvore, num percurso em direção a raiz.

Percebe-se na Figura 5.4 (onde apenas o conjunto *reduzido* das linhas envolvidas foi apresentado), que o padrão estrutural a cada coluna, atende justamente ao requisito necessário para uma codificação por *envelopes.*

Tal fato sempre ocorrerá por construção, caso se explore um percurso de contribuições, a partir da estrutura hierárquica, ditada pela árvore de eliminação.

O que se percebe portanto é que uma forma apropriada para se proceder a um

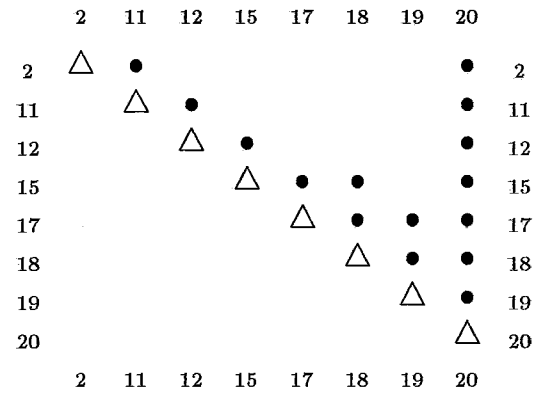


Fig 5.4: Estrutura Ancestral (*Reduzida*)

reconhecimento e reordenamento das contribuições, visando uma codificação por *envelopes*, é justamente um reordenamento baseado numa hierarquia *Ancestral* de contribuições a serem incorporadas a linha base a cada etapa.

Tal reordenamento, (denominado *Ancestral*) foi originalmente proposto e detalhadamente apresentado em [6], o qual passaremos a considerar, na forma de um exemplo (reportando-se o leitor a referência citada, para maiores informações).

Nota-se que por se proceder apenas a percursos seguindo por um *único* ramo em direção a raiz, nem todas as linhas contribuintes (de uma dada etapa), podem ser alcançadas. No caso em que apenas um percurso não se mostra suficiente, uma nova exploração da árvore é efetuada, repetindo-se o processo até que se consiga abranger todo o conjunto de linhas contribuintes.

A cada um dos ramos assim explorados, atribuiu-se a denominação de uma *família* em [6].

O que a técnica de reordenamento ancestral fornece ao final do processo de explorações, é um novo vetor **JUT** (com o índice das linhas contribuintes) particionado por *famílias* (cada uma das quais com os índices de suas linhas constituintes, *ancestralmente* reordenados).

Tomemos como exemplo para ilustrar esta técnica, inicialmente a linha 20 (da matriz da Figura 5.2) e vejamos como proceder para se obter um vetor **JUT** reordenado neste caso.

Originalmente o vetor **JUT** neste caso, contém:

linha 20 (original)

JUT: 2 5 11 12 13 15 16 17 18 19

Pelo fato da ordem natural especificada em **JUT** estar na forma crescente, pode-se iniciar a exploração da árvore a partir da primeira contribuição, correspondente a linha 2 (por possuir um índice seguramente inferior ao de todas as demais contribuições).

Assim o vetor **JUT** reordenado, a ser denominado **JUTORD**, passará a conter inicialmente:

JUTORD: 2

Caminhando-se pela árvore a partir deste nó, em direção a raiz, e acrescentando-se a **JUTORD** todas as contribuições (referentes a linha base 20), encontradas durante este percurso, obtemos:

JUTORD: 2 11 12 15 17 18 19

Uma vez concluído este percurso inicial, a primeira *família* de contribuições terá sido completamente determinada.

O processamento continua desta forma, tomando-se a primeira linha contribuinte (a partir da estrutura **JUT** original), ainda não incorporada ao vetor reordenado.

Neste caso, teremos como ancestral mais remoto, a iniciar um novo percurso, o nó 5.

Procedendo-se de forma análoga a anteriormente apresentada, e incluindo-se em **JUTORD** os descendentes do nó 5, que porventura contribuam sobre a linha base 20 (e ainda não tenham sido incorporados a alguma outra família), teremos:

JUTORD: 2 11 12 15 17 18 19 | 5 13 |

O processo de reordenamento ancestral para o exemplo considerado, é concluido incorporando-se a **JUTORD** o nó 16, correspondente a última família até então não explorada, fornecendo como resultado final:

linha 20 (reordenada)

JUTORD: 2 11 12 15 17 18 19 | 5 13 | 16 |

Um ponto que vale ser ressaltado é que um *reordenamento ancestral* segundo *famílias*, não é único, no sentido em que se alterando a ordem de escolha dos ancestrais iniciais (determinadores de cada família), pode-se vir a obter um particionamento distinto do originalmente encontrado (tomando-se como critério de escolha, sempre o nó ancestral de menor índice, ainda não incorporado a estrutura de **JUTORD**).

Tal ocorre, pelo fato da escolha do ancestral inicial, determinar a inclusão na família a ele associada, de todos os demais nós contribuintes sobre a linha base, encontrados no percurso em direção a raiz. Esta escolha inicial, inviabiliza, que uma porção terminal dos nós de uma dada família, possa ser incorporada na estrutura de descendentes de uma outra família, que porventura venha a possuir o mesmo conjunto de nós como descendentes comuns.

No exemplo em questão, nota-se que a família originada a partir do nó 5, só incuiu como descendente adicional o nó 13, pelo fato dos nós 18 e 19 já terem sido incluídos como descendentes na família originada a partir do nó 2.

Assim, caso se iniciasse o processo de reordenamento, a partir do nó 5, e se incluísse como próximo ancestral primordial (após a especificação da primeira família), o nó 16, para só a partir deste ponto proceder-se a exploração da família remanescente, a partir do nó 2, o vetor de reordenamento obtido seria:

linha 20 (reordenada) *alternativamente*

JUTORD: 5 13 18 19 | 16 17 | 2 11 12 15 |

Tal fato ilustra que o critério originalmente apresentado em [6], é apenas uma das possíveis escolhas, tomando-se arbitrariamente como ordem natural de especificação de cada família, os nós ancestrais de menor índice ainda não incorporados a estrutura reordenada. Tal procedimento, por construção, garante que a estrutura de contribuições seja completamente explorada, porém não garante que a ordem obtida seja a que otimize algum outro critério segundário, como o número de famílias, ou seu tamanho médio. Uma forma alternativa de sub-divisão por famílias é caminhar-se em sentido inverso, partindo-se inicialmente dos *supernodes* contribuintes a cada etapa (associando-se uma família distinta a cada um), para posteriormente se proceder a uma fase de aglutinação de famílias correspondentes aos *supernodes* diretamente interrelacionados entre si, até que todos os nós contribuintes isolados tenham sido processados ou aglutinados a alguma outra família.

O campo de critérios para o reordenamento ainda se encontra aberto a investigações, não sendo o objetivo do presente trabalho a exploração de novas estratégias visando elevar-se ainda mais o nível de compactação passível de obtenção (a partir de codificações por *envelope*).

Na prática, como se verá um pouco mais adiante, um outro tipo de codificação, baseado na noção de *supernodes*, mostra-se dominante na grande maioria dos casos, respondendo desta forma, por boa parte da sensível economia em termos de espaço adicional de trabalho, que se pode obter com codificações desta forma em particular.

Apresenta-se a seguir, a título de complementação (e sem maior nível de detalhamento, reportando-se o leitor a [6]), o Procedimento **5.2(1)**, voltado para a obtenção do vetor de *reordenamento ancestral* **JUTORD**, e das estruturas auxiliares **IFAM** e **KFAM**, visando a delimitação e especificação de cada uma das famílias contribuintes a cada etapa básica do processo de eliminação. (Além destas estruturas, o procedimento fornece também o número de famílias contribuintes sobre cada linha base, no vetor auxiliar **NFAM**).

Como estratégia adotada para a exploração eficiente dos ramos da árvore, adotou-se vetores auxiliares **CONTRIB** e **VISITED**, de modo a rotular os nós porventura visitados, evitando-se assim varreduras desnecessárias, por trechos préviamente explorados na determinação de outras famílias.

Processamento Restrito de Contribuições sobre a Linha Base

De posse da estrutura de contribuições reordenada *ancestralmente* (por famílias), mostra-se possível a implementação de uma nova forma de eliminação, baseada na utilização de códigos por *envelope* (como apresentado originalmente na Figura 5.1).

Uma vez que neste caso os acessos à linha base, correspondem apenas ao das contribuições relativas a cada família, um acesso a todos os elementos básicos por cada varredura, como no passo (g) do Procedimento **5.1(1)** pode mostrar-se computacionalmente oneroso, caso se encontre um número significativo de códigos nulos (a serem inútilmente processados).

Assim, mostra-se conveniente lançar mão de uma nova forma de acesso, *restrito* apenas aos elementos afetados por contribuições não nulas.

Procedimento 5.2(1) *Reordenamento Ancestral*

```
para i de 1 até n faça
    CONTRIB[i] ← falso
    VISITED[i] ← falso

IFAM[1] ← 1

para i de 1 até n faça

    NFAM[i] ← 0

    se ( IUTF[i] ≥ IUT[i] ) então

        para k de IUT[i] até IUTF[i] faça
            CONTRIB[JUT[k]] ← verdadeiro

        k ← IUT[i], kf ← IUTF[i]
        l ← JUT[k], lf ← JUTF[kf]

        VISITED[PARENT[lf]] ← verdadeiro
        VISITED[i] ← verdadeiro

        nl ← 0, nvis ← 0

α :     VISITED[l] ← verdadeiro
        nl ← nl + 1, JW[nl] ← l

β :     l ← PARENT[l]

        se VISITED[l] va' para γ
        se CONTRIB[i] va' para α

        VISITED[l] ← verdadeiro
        nvis ← nvis + 1, LVIS[nvis] ← l

        va' para β

γ :     KFAM[IFAM[i] + NFAM[i]] ← IUT[i] + nl
        NFAM[i] ← NFAM[i] + 1

δ :     k ← k + 1, se k > kf va' para ε
        l ← JUT[k], se VISITED[l] va' para δ

        va' para α

ε :     j ← 1

        para k de IUT[i] até IUTF[i] faça
            CONTRIB[JUT[k]] ← falso
            VISITED[JUT[k]] ← falso
            JUTORD[k] ← JW[j], j ← j + 1
        fim para k

        VISITED[PARENT[lf]] ← falso
        VISITED[i] ← falso

        para j de 1 até nvis faça
            VISITED[LVIS[j]] ← falso

    fim se ( IUTF[i] ≥ IUT[i] )

    IFAM[i + 1] ← IFAM[i] + NFAM[i]

fim para i
```

Proc 5.2(1): *Reordenamento Ancestral*

Apresenta-se deste modo na Alternativa **5.2(a)**, uma nova forma de processamento para a adição de vetores esparsos, onde se assume o conhecimento de listas **PJ** e **PJF**, contendo intervalos de endereços seqüencialmente contíguos no vetor **U**, correspondentes aos elementos de **V** a serem adicionados.

Alternativa 5.2(a) *Adição de Vetores Esparsos via Listas Restritas*

$jv \leftarrow iv$, $ipj \leftarrow 1$
enquanto (**PJ**[ipj] $\neq 0$) **faça**
    para j **de** **PJ**[ipj] **até** **PJF**[ipj] **faça**
        U[j] $\leftarrow$ **U**[j] + **V**[jv]
        $jv \leftarrow jv + 1$
    $ipj \leftarrow ipj + 1$
fim enquanto

Alt 5.2(a): *Adição de Vetores Esparsos via Listas Restritas*

Nota-se que esta alternativa, pode ser empregada em todos os casos préviamente considerados, baseados na adição de vetores esparsos, mediante a especificação de listas de endereços (como os Procedimentos **4.1(3)**, **4.1(4)** ou a Alternativa **4.2(b)**).

Codificação por Envelopes

No Procedimento **5.2(2)** a seguir, apresenta-se um esquema para uma eliminação numérica do tipo Crout simétrico, explorando uma lista **LENV** de códigos por envelope (espelhando os padrões relativos a cada *família* de contribuições), e efetuando *acessos restritos* aos elementos de cada linha base sendo gerada.

Descrição de Passos do Procedimento **5.2(2)**

No passo (a) do procedimento **5.2(2)** inicializa-se o indexador *ienv* para a lista de códigos por envelopes, o contador $nfam$ do número de famílias correntemente empregadas, e o indexador *ipj* para a lista *restrita* de endereços sobre a linha base (delimitados por **PJ** e **PJF**).

No passo (b) o contador de famílias e incrementado, e o último valor do indexador k, relativo a família corrente, é armazenado em *kend*.

No passo (c) inicializa-se a variável r de modo a apontar para o índice (relativo ao envelope), da primeira contribuição a partir da qual, todas as demais linhas contribuintes passarão a conter um padrão de elementos não nulos, numa dada coluna em particular.

No passo (d) o índice *reduzido* r é decrementado, de modo a atingir um valor unitário, por ocasião da última contribuição (de maior índice relativo).

No passo (e) incrementa-se o indexador *ipj* das listas de intervalos **PJ** e **PJF**.

Procedimento 5.2(2) *Eliminação via Códigos por Envelopes (Crout)*

($n_{bits} = 3$)

$ienv \leftarrow 0,\ nfam \leftarrow 0,\ ipj \leftarrow 0$ (a)
para i **de** 1 **até** n **faça**
  $piv \leftarrow$ **DI**$[i]$, **IUP**$[i] \leftarrow$ **IU**$[i]$
  $kbeg \leftarrow$ **IUT**$[i]$, $kf \leftarrow$ **IUTF**$[i]$
α : $nfam \leftarrow nfam + 1,\ kend \leftarrow$ **KFAM**$[nfam]$ (b)
  $r \leftarrow kend - kbeg + 1$ (c)
  para k **de** $kbeg$ **até** $kend$ **faça**
    $l \leftarrow$ **JUT**$[k]$, $iupl \leftarrow$ **IUP**$[l]$
    IUP$[l] \leftarrow$ **IUP**$[l] + 1,\ p_{(r)} \leftarrow$ **IUP**$[l]$
    $um_{(r)} \leftarrow$ **UN**$[iupl] *$ **DI**$[l]$
    $piv \leftarrow piv -$ **UN**$[iupl] * um_{(r)}$
    UN$[iupl] \leftarrow um_{(r)}$
    $r \leftarrow r - 1$ (d)
  fim para k

(acumulação *restrita* das contribuições sobre a *linha base*, coluna a coluna)

  $ipj \leftarrow ipj + 1$ (e)
  enquanto (**PJ**$[ipj] \neq 0$) **faça** (f)
    para j **de** **PJ**$[ipj]$ **até** **PJF**$[ipj]$ **faça** (g)
      $s \leftarrow 0.,\ ienv \leftarrow ienv + 1$ (h)
      va' para (1, 2, 3, 4, 5, 6, 7) **em função de** **LENV**$[ienv]$ (i)
β : **UN**$[j] \leftarrow$ **UN**$[j] - s$ (j)
    $ipj \leftarrow ipj + 1$ (k)
  fim enquanto

  $kbeg \leftarrow kend + 1$
  se $kend \neq kf$ **va' para** α
  DI$[i] \leftarrow 1. \,/\, piv$
fim para i

termine procedimento

1 : $s \leftarrow s +$ **UN**$[p_1]$
  $p_1 \leftarrow p_1 + 1$
  va' para β
2 : $s \leftarrow s +$ **UN**$[p_1] +$ **UN**$[p_2]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
  va' para β
3 : $s \leftarrow s +$ **UN**$[p_1] +$ **UN**$[p_2] +$ **UN**$[p_3]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
  va' para β
4 : $s \leftarrow s +$ **UN**$[p_1] +$ **UN**$[p_2] +$ **UN**$[p_3]$
    $+$ **UN**$[p_4]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1,\ p_4 \leftarrow p_4 + 1$
... ...
7 : $s \leftarrow s +$ **UN**$[p_1] +$ **UN**$[p_2] +$ **UN**$[p_3]$
    $+$ **UN**$[p_4] +$ **UN**$[p_5] +$ **UN**$[p_6]$
    $+$ **UN**$[p_7]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1,\ p_4 \leftarrow p_4 + 1$
  $p_5 \leftarrow p_5 + 1,\ p_6 \leftarrow p_6 + 1,\ p_7 \leftarrow p_7 + 1$
  va' para β

No passo (f) enquanto não se houver chegado ao final do processamento das colunas relativas à família corrente, processam-se as contribuições de um novo conjunto *restrito* de colunas (até se esgotarem as contribuições da família sendo considerada).

No passo (g) processa-se a acumulação de contribuições das colunas restritas ao intervalo ditado por **PJ**[ipj] a **PJF**[ipj].

No passo (h) inicializa-se o acumulador de contribuições s, e incrementa-se o apontador na lista de envelopes.

No passo (i) desvia-se condicionalmente para o trecho de código apropriado ao tratamento de cada padrão estrutural de contribuições.

No passo (j) tais contribuições na dada coluna, são agregadas definitivamente ao elemento da linha base corrente, sendo gerado.

No passo (k) o indexador ipj é incrementado de modo a se processarem novas contribuições *restritas* sobre a linha base corrente.

□

Geração da Codificação por Envelopes

Mostra-se conveniente, apresentar a seguir, um procedimento voltado para a determinação da lista de *envelopes*.

No Procedimento **5.2(3)** apresentam-se pois todos os passos necessários para se gerar a lista de códigos **LENV**.

Descrição de Passos do Procedimento **5.2(3)**

No passo (a) do Procedimento **5.2(3)** inicializa-se o indicador do número de códigos do tipo envelope correntemente gerados, e o indexador ipj das listas de intervalos restritos **PJ** e **PJF**.

No passo (b) anulam-se todas as posições associadas a elementos não nulos da linha base, no vetor auxiliar **JCOD**.

No passo (c) processa-se a determinação dos códigos associados a cada uma das *famílias* de contribuições, relativas a linha base i a ser gerada na fase numérica.

No passo (d) armazena-se em $kend$, o índice do maior valor do indexador k, associado a família sendo correntemente processada.

No passo (e) inicializa-se o indexador *reduzido* r, para o maior valor relativo a ser assumido durante a geração dos códigos associados a família corrente.

No passo (f) atualiza-se o vetor auxiliar **JCOD**, por ocasião da primeira ocorrência de uma contribuição não nula a cada coluna associada.

No passo (g) o valor do indexador *reduzido* r é decrementado (correspondendo a um valor unitário, por ocasião da última contribuição processada).

No passo (h) inicializa-se o sinalizador $jflag$, utilizado para se detectar os intervalos de posições j *restritas* a códigos não nulos (a serem incorporados na lista por envelopes **LENV**).

Procedimento 5.2(3) *Geração da Codificação por Envelopes*

```
ienv ← 0, ipj ← 0                                                    (a)

para i de 1 até n faça

    IUP[i] ← IU[i]

    para j de IU[i] até IUF[i] faça                                  (b)
        JCOD[JU[j]] ← 0

    kbeg ← IUT[i]

    para ik de IFAM[i] até IFAM[i + 1]−1 faça                        (c)

        kend ← KFAM[ik]                                              (d)

        r ← kend − kbeg + 1                                          (e)

        para k de kbeg até kend faça

            l ← JUT[k], IUP[l] ← IUP[l] + 1

            para j de IUP[l] até IUF[l] faça
                se ( JCOD[JU[j]] = 0 ) então                         (f)
                    JCOD[JU[j]] ← r

            r ← r − 1                                                (g)

        fim para k

        jflag ← falso                                                (h)

        para j de IU[i] até IUF[i] faça                              (i)

            se ( JCOD[JU[j]] ≠ 0 ) então                             (j)

                ienv ← ienv + 1
                LENV[ienv] ← JCOD[JU[j]]                             (k)

                se ( jflag = falso ) então                           (l)
                    ipj ← ipj + 1, PJ[ipj] ← j                       (m)
                    jflag ← verdadeiro                               (n)
                fim se

            senão

                se ( jflag = verdadeiro ) então                      (o)
                    PJF[ipj] ← j − 1                                 (p)
                    jflag ← falso                                    (q)
                fim se

            fim se

        fim para j

        se ( jflag = verdadeiro ) então                              (r)
            PJF[ipj] ← IUF[i]                                        (s)

        ipj ← ipj + 1
        PJ[ipj] ← 0, PJF[ipj] ← −1                                   (t)

        kbeg ← kend + 1

    fim para ik

fim para i
```

Proc 5.2(3): *Geração da Codificação por Envelopes*

No passo (i) processa-se a incorporação dos códigos (não nulos) gerados temporáriamente em **JCOD**, transferindo-os definitivametne para a *lista de envelopes* **LENV**, no passo (k).

No passo (j) processa-se separadamente o caso em que o código corrente possui um valor não nulo (a ser incorporado na lista por *envelopes*), do caso em que o mesmo assume um valor nulo.

No passo (k) incorpora-se o código não nulo gerado, na lista por *envelopes*.

No passo (l) processa-se o caso em que o sinalizador *jflag* ainda se encontre falso (indicando deste modo a presença do primeiro código não nulo do intervalo a ser correntemente especificado na listas *restrita* **PJ**).

No passo (m) incorpora-se a posição j corrente na lista de posições iniciais.

No passo (n) atualiza-se o estado do sinalizador $jflag$ para um valor verdadeiro, indicando que a posição inicial correspondente a um código não nulo já se encontra detectada.

No passo (o) processa-se o caso em que o código correntemente especificado é nulo e o sinalizador $jflag$ tenha tido seu valor atualizado para verdadeiro em alguma instância passada (indicando que a posição inicial do atual intervalo *restrito* tenha sido préviamente detectada).

Neste caso, no passo (p) incorpora-se a posição consecutivamente anterior de j na lista de delimitadores finais **PJF** (onde encontra-se necessáriamente associado, por construção, o último código não nulo do intervalo *restrito* corrente).

No passo (q) o sinalizador $jflag$ é reinicializado para um valor falso, permitindo a deteção de novos intervalos *restritos*, sobre a linha base corrente.

No passo (r) testa-se o caso em que apenas um intervalo restrito foi necessário para a especificação das contribuições sobre a linha base corrente, não tendo se detectado a presença de um código nulo não ao longo de todo o passo (i).

Neste caso, no passo (s) incorpora-se a posição correspondente ao último elemento não nulo da linha base, na lista delimitadora final **PJF**.

Em todos os casos, após o processamento de todos os intervalos *restritos* de posições sobre a linha base corrente, introduz-se no passo (t) um delimitador nulo, na lista inicial **PJ**, de modo a indicar o fim da especificação de endereços base, relativos a família correntemente sendo processada.

□

Codificação Supernodal

Um caso particular de codificação por *envelopes* (encontrado com significativa freqüência), ocorre quando os códigos relativos as contribuições de uma mesma família (sobre uma dada linha base), são todos *idênticos*.

Neste caso, (que inclui o de contribuições da forma *supernodal plena* como um caso particular), nota-se que as únicas informações necessárias para a caracterização

de todo o processo de contribuições de uma dada família, são:

- o padrão (idêntico) de contribuições sobre cada coluna
- o particular grupo de colunas da linha base, afetadas pelo mesmo padrão

Esta caracterização é mais ampla que a noção de processamento *supernodal* introduzida no capítulo 4, onde se considerava apenas os casos de contribuições de forma *plena*, ou seja, abrangendo exatamente *todos* os elementos não nulos da linha base sendo gerada.

A noção proposta nesta seção, corresponde ao que se poderia classificar como contribuições supernodais de forma *parcial* (podendo abranger apenas um selecionado grupo de elementos da linha base). Esta nova noção, naturalmente induz a um esquema de *codificação* de forma *supernodal*, que será particularmente considerado nesta sub-seção.

Nota-se em princípio, que a exploração da noção de contribuições *parciais* repetitivas, poderia ser igualmente aplicada no caso de padrões (idênticos), originalmente codificados de forma *binária*. Tal porém não se verifica com significativa freqüência na prática, não sendo merecedor de atenção do presente trabalho.

Deste modo, os padrões repetitivos a serem considerados na forma de *codificações*, serão apenas os relativos a famílias de linhas contribuintes (originalmente codificáveis até então, na forma por *envelope*).

Apresenta-se a seguir, no Procedimento **5.2(4)**, uma eliminação baseada em códigos *supernodais*, análoga ao procedimento **5.2(2)** (baseado em codificações por envelope) apresentado na sub-seção anterior.

Nota-se que no procedimento **5.2(4)**, lança-se mão de uma lista auxiliar **KSUP**, análoga a **KFAM** (utilizada no caso por envelopes), só que com cada *supernode* correspondendo a uma família distinta (garantindo desta forma a adoção de um esquema de decodificação baseado apenas em códigos da forma *supernodal*).

Descrição de Passos do Procedimento **5.2(4)**

No passo (a) inicializam-se os indexadores para as listas de códigos supernodais, de delimitação de famílias contribuintes a cada etapa, e de acesso restrito aos elementos da linha base.

No passo (b) atualiza-se o delimitador final das linhas contribuintes relativas a um mesmo *supernode*, a serem processadas simultâneamente na etapa básica corrente.

No passo (c) incrementa-se o indexador *isup* para a posição do código *supernodal* corrente (a ser explorado ao longo de todo o passo (e)).

No passo (d) incrementa-se o indexador *ipj* para a posição a ser correntemente acessada na lista *restrita* **PJ**.

No passo (e), enquanto não se houver chegado ao término do processamento restrito relativo a família corrente, repetem-se os passos (f) e (g), até que um delimitador inicial nulo seja encontrado.

Procedimento 5.2(4) *Eliminação via Códigos Supernodais (Crout)*

($n_{bits} = 3$)

```
        isup ← 0, nsup ← 0, ipj ← 0                                        (a)
        para i de 1 até n faça
            piv ← DI[i], IUP[i] ← IU[i]
            kbeg ← IUT[i], kf ← IUTF[i]
α :         nsup ← nsup +1, kend ← KSUP[nsup]                              (b)
            r ← kend − kbeg + 1
            para k de kbeg até kend faça
                l ← JUT[k], iupl ← IUP[l]
                IUP[l] ← IUP[l] + 1, p(r) ← IUP[l]
                um(r) ← UN[iupl] * DI[l]
                piv ← piv − UN[iupl] * um(r)
                UN[iupl] ← um(r)
                r ← r − 1
            fim para k
            isup ← isup + 1                                                (c)
            ( acumulação restrita das contribuições sobre a linha base, coluna a coluna )
        ---------------------------------------------------------------
            ipj ← ipj + 1                                                  (d)
            enquanto ( PJ[ipj] ≠ 0 ) faça                                  (e)
                va' para ( 1,2,3,4,5,6,7 ) em função de LSUP[isup]         (f)
β :             ipj ← ipj + 1                                              (g)
            fim enquanto
        ---------------------------------------------------------------
            kbeg ← kend + 1
            se kend ≠ kf va' para α
            DI[i] ← 1. / piv
        fim para i

        termine procedimento

1 :     para j de PJ[ipj] até PJF[ipj] faça                                (h)
            UN[j] ← UN[j] − um1 * UN[p1]
            p1 ← p1 + 1
        va' para β
2 :     para j de PJ[ipj] até PJF[ipj] faça
            UN[j] ← UN[j] − um1 * UN[p1] − um2 * UN[p2]
            p1 ← p1 + 1, p2 ← p2 + 1
        va' para β
3 :     para j de PJ[ipj] até PJF[ipj] faça
            UN[j] ← UN[j] − um1 * UN[p1] − um2 * UN[p2] − um3 * UN[p3]
            p1 ← p1 + 1, p2 ← p2 + 1, p3 ← p3 + 1
        va' para β
...     ...
7 :     para j de PJ[ipj] até PJF[ipj] faça
            UN[j] ← UN[j] − um1 * UN[p1] − um2 * UN[p2] − um3 * UN[p3]
                          − um4 * UN[p4] − um5 * UN[p5] − um6 * UN[p6]
                          − um7 * UN[p7]
            p1 ← p1 + 1, p2 ← p2 + 1, p3 ← p3 + 1, p4 ← p4 + 1
            p5 ← p5 + 1, p6 ← p6 + 1, p7 ← p7 + 1
        va' para β
```

Proc 5.2(4): *Eliminação via Códigos Supernodais (Crout)*

No passo (f) processa-se o desvio condicional em função do código *supernodal* (contendo o número de linhas a contribuírem simultâneamente sobre cada coluna, sendo considerado).

No passo (g) incrementa-se o indexador *ipj* de modo a se permitir a especificação de um novo intervalo *restrito*, relativo as contribuições de novas colunas, da mesma família de contribuições sendo correntemente processada.

Finalmente no passo (h) e em todos os demais passos correlatos (nos trechos rotulados desde 1 : até 7 :), processam-se as contribuições do intervalo restrito de colunas, diretamente sobre as posições seqüenciais correspondentes na linha base.

□

5.3 Codificações Mais Elaboradas

Os esquemas de codificação até então apresentados, foram todos baseados na utilização de códigos de tamanho *fixo* (correspondendo a um *byte* ou *palavra* de representação).

Nesta seção, se apresentarão codificações mais elaboradas, capazes de representar mais de um conjunto de operações básicas por cada código.

Tais codificações assim denominadas como *múltiplas*, exploram a noção de *multiplexação*, até então não considerada, permitindo que mediante desvios condicionais duplamente encadeados, se possa extender o número total de casos particulares a serem dedicadamente processados.

Codificações voltadas para Processamento Dedicado

Uma primeira extensão passível de incorporação em esquemas de codificação *binária* é o tratamento *dedicado* de operações sobre elementos de *fill-in*.

Ocorre particularmente neste caso, que nos métodos tradicionais, o tratamento dispensado à primeira vez que se opera uma contribuição sobre um elemento de *fill-in* (originalmente nulo) é exatamente o mesmo dispensado aos demais fatores não nulos, ou seja uma operação de subtração sobre o elemento básico a ser operado.

Nota-se no entanto, no caso da primeira contribuição sofrida por cada elemento de *fill-in*, que em função de seu valor originalmente nulo, mostra-se possível evitar a operação de subtração do valor contribuinte, substituindo-se diretamente no elemento de *fill-in*, o valor da contribuição a ser recebida (com o sinal trocado).

Apresenta-se pois na Codificação **5.3-1** uma forma de se processar particularmente as operações sobre cada elemento de *fill-in*, mediante códigos análogos aos adotados para o tratamento *binário* mais geral (sobre os demais elementos).

Nesta codificação, os trechos rotulados de f_1 a f_7 correspondem ao tratamento dedicado, com os demais trechos correspondendo ao processamento tradicional.

Uma forma de se evitar a operação de inversão de sinal, por ocasião da atribuição do primeiro valor não nulo a cada elemento de *fill-in*, é proceder-se a inversão do sinal

Codificação 5.3-1 *Acumulação Dedicada de Contribuições sobre Fill-In's*

```
0 :   retorne ao processamento principal

1 :   UN[j] ← UN[j] + umn_1 * UN[p_1]
      p_1 ← p_1 + 1
      retorne ao processamento principal
2 :   UN[j] ← UN[j] + umn_2 * UN[p_2]
      p_2 ← p_2 + 1
      retorne ao processamento principal
3 :   UN[j] ← UN[j] + umn_1 * UN[p_1] + umn_2 * UN[p_2]
      p_1 ← p_1 + 1, p_2 ← p_2 + 1
      retorne ao processamento principal
4 :   UN[j] ← UN[j] + umn_3 * UN[p_3]
      p_3 ← p_3 + 1
      retorne ao processamento principal
5 :   UN[j] ← UN[j] + umn_1 * UN[p_1] + umn_3 * UN[p_3]
      p_1 ← p_1 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal
6 :   UN[j] ← UN[j] + umn_2 * UN[p_2] + umn_3 * UN[p_3]
      p_2 ← p_2 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal
7 :   UN[j] ← UN[j] + umn_1 * UN[p_1] + umn_2 * UN[p_2] + umn_3 * UN[p_3]
      p_1 ← p_1 + 1, p_2 ← p_2 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal

f_1 : UN[j] ← umn_1 * UN[p_1]
      p_1 ← p_1 + 1
      retorne ao processamento principal
f_2 : UN[j] ← umn_2 * UN[p_2]
      p_2 ← p_2 + 1
      retorne ao processamento principal
f_3 : UN[j] ← umn_1 * UN[p_1] + umn_2 * UN[p_2]
      p_1 ← p_1 + 1, p_2 ← p_2 + 1
      retorne ao processamento principal
f_4 : UN[j] ← umn_3 * UN[p_3]
      p_3 ← p_3 + 1
      retorne ao processamento principal
f_5 : UN[j] ← umn_1 * UN[p_1] + umn_3 * UN[p_3]
      p_1 ← p_1 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal
f_6 : UN[j] ← umn_2 * UN[p_2] + umn_3 * UN[p_3]
      p_2 ← p_2 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal
f_7 : UN[j] ← umn_1 * UN[p_1] + umn_2 * UN[p_2] + umn_3 * UN[p_3]
      p_1 ← p_1 + 1, p_2 ← p_2 + 1, p_3 ← p_3 + 1
      retorne ao processamento principal
```

Cod 5.3-1: *Acumulação Dedicada de Contribuições sobre Fill-In's*

dos fatores multiplicativos $um_{(.)}$. Este fato foi explorado na codificação apresentada, onde se denotou por $umn_{(.)}$ tais fatores com o sinal préviamente invertido.

A troca de sinal (não exibida no trecho de codificação apresentado) pode ser processada de forma oportuna, por ocasião da incorporação das contribuições sobre a diagonal, e da normalização de elementos sobre a coluna base.

No caso do tratamento de sistemas reais, a economia de uma operação de subtração para cada elemento de *fill-in*, só é justificada para matrizes com elevado grau de esparsidade, como as típicamente encontradas na area de potência (com um percentual de elementos não nulos a cada linha muito baixo, muitas das vezes situado na faixa de 3 a 4 elementos).

Onde a técnica de contribuições dedicadas sobre *fill-in's* mostra-se mais vantajosa, é no caso do processamento de sistemas complexos, onde a economia de uma operação de subtração complexa se faz notar mais acentuadamente do que no caso real.

Codificações Supernodais Múltiplas

Uma forma de se elevar o desempenho dos métodos baseados em codificações, é elevar-se o percentual de operações de ponto flutante especificadas por cada código.

Uma forma simples de extensão do número de operações se dá no caso *supernodal*, onde como se observa, o padrão estrutural de contribuições se mantém constante por todo o conjunto de linhas constituintes de um mesmo *supernode*.

Assim, uma forma de se aumentar a quantidade de informação representada por cada código, é adotar-se códigos *comuns* para o processamento de mais de uma linha base simultâneamente, visto ser idêntico, o padrão de contribuições a ser recebido por ambas as linhas de um mesmo supernode.

Os esquemas de codificação assim desenvolvidos, passam portanto a espelhar contribuições sobre mais de um elemento base, razão pela qual se optou por denominá-los como *múltiplos*.

Apresenta-se inicialmente na Codificação **5.3-2** uma forma de acumulação *supernodal múltipla*, com geração simultânea de mais de uma *linha base*.

Nesta codificação, os trechos rotulados de $1'$ até $7'$ correspondem ao tratamento *múltiplo*, com os demais casos correspondendo a um tratamento *supernodal* convencional (de uma única linha base).

De modo a se acessar as posições correspondentes na segunda linha base, lança-se mão de um indexador j_2 (assumido como inicializado, antes de cada um dos *loops* apresentados).

De forma análoga, assume-se que os fatores multiplicativos $um^2_{(.)}$ relativos a segunda linha base, tenham sido inicializados juntamente com os fatores $um_{(.)}$ tradicionais.

Da mesma forma como se procedeu a uma codificação *múltipla*, especificando-se operações simultâneas sobre os elementos de mais de uma linha base, pode-se adotar esquemas de acumulação *múltipla*, baseados na geração de mais de uma

Codificação 5.3-2 *Acumulação Supernodal Múltipla (Linhas)*

1 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1$

retorne ao processamento principal

2 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$

retorne ao processamento principal

3 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$

retorne ao processamento principal

... ...
7 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3]$
$- um_4 * \mathbf{UN}[p_4] - um_5 * \mathbf{UN}[p_5] - um_6 * \mathbf{UN}[p_6]$
$- um_7 * \mathbf{UN}[p_7]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1,\ p_4 \leftarrow p_4 + 1$
$p_5 \leftarrow p_5 + 1,\ p_6 \leftarrow p_6 + 1,\ p_7 \leftarrow p_7 + 1$

retorne ao processamento principal

1′ : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1$
$j_2 \leftarrow j_2 + 1$

retorne ao processamento principal

2′ : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
$j_2 \leftarrow j_2 + 1$

retorne ao processamento principal

3′ : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2] - um_3^2 * \mathbf{UN}[p_3]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
$j_2 \leftarrow j_2 + 1$

retorne ao processamento principal

... ...
7′ : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3]$
$- um_4 * \mathbf{UN}[p_4] - um_5 * \mathbf{UN}[p_5] - um_6 * \mathbf{UN}[p_6]$
$- um_7 * \mathbf{UN}[p_7]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2] - um_3^2 * \mathbf{UN}[p_3]$
$- um_4^2 * \mathbf{UN}[p_4] - um_5^2 * \mathbf{UN}[p_5] - um_6^2 * \mathbf{UN}[p_6]$
$- um_7^2 * \mathbf{UN}[p_7]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1,\ p_4 \leftarrow p_4 + 1$
$p_5 \leftarrow p_5 + 1,\ p_6 \leftarrow p_6 + 1,\ p_7 \leftarrow p_7 + 1$
$j_2 \leftarrow j_2 + 1$

retorne ao processamento principal

Cod 5.3-2: *Acumulação Supernodal Múltipla (Linhas)*

coluna simulâneamente sobre uma mesma linha base.

Apresenta-se pois na Codificação **5.3-3**, um esquema voltado para uma acumulação *supernodal múltipla* de até 2 *colunas* simultâneas, sobre cada linha base.

Neste esquema, nota-se que os apontadores $p_{(.)}$ passam a ser incrementados de duas posições a cada interpretação de um código *supernodal.*

Como os elementos sendo gerados encontram-se sobre a mesma linha base, os fatores multiplicativos neste caso, se mantém os mesmos, para ambos os casos.

Em vista do que se apresentou até o momento, não se mostra difícil a extensão da técnica proposta, para um processamento *supernodal múltiplo*, com geração simultânea de 2 linhas base, e 2 colunas sobre cada uma delas.

Tal processo, indicado na Codificação **5.3-4**, corresponde a uma geração simultânea de fatores especificados por até 2 *linhas* (geradas) $\times$ 2 *colunas* (contribuintes).

Todas as codificações *supernodais múltiplas*, individualmente apresentadas até então, podem ser incorporadas num único esquema de eliminação, mediante a adoção de um esquema de *multiplexação*, a ser comentado um pouco mais adiante.

Apresenta-se deste modo a seguir, na Codificação **5.3-5** um esquema *múltiplo* unificado, tratando os casos de contribuições sobre até 2 linhas e/ou 2 colunas simultâneas.

Nota-se que a codificação apresentada, encontra-se na forma de um pseudo-código mais abstrato, onde se omitiu o detalhamento de algumas das operações de inicialização, normalização e atualização de valores, para o caso do tratamento simultâneo de 2 linhas.

Descrição de Passos da Codificação **5.3-5**

No passo (a) da Codificação **5.3-5**, inicializam-se os indexadores para as listas de códigos *supernodais* e demais listas auxiliares, como a de desvios *multiplexados*, especificados em **LMPLX**.

No passo (b) processa-se a atualização do indexador *isup* da lista de códigos *supernodais.*

No passo (c) atualiza-se o indexador *implx*, da lista de devios a serem *multiplexados* para o tipo apropriado de processamento.

No passo (d) incrementa-se o indexador de posições na lista *restrita* de endereços sobre a primeira linha base.

No passo (e) processa-se o acúmulo de todas contribuições relativas ao *supernode* contribuinte corrente.

No passo (f) inicializa-se o apontador para a posição inicial na segunda linha base a ser simultâneamente operada.

No passo (g), processa-se o desvio para o trecho apropriado do programa (onde após um novo desvio, partindo-se de um dos passos desde γ até γ''', se conseguirá chegar definitivamente ao trecho de programa correspondente a cada caso).

Codificação 5.3-3 *Acumulação Supernodal Múltipla (Colunas)*

1 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1]
$\quad$ $p_1 \leftarrow p_1 + 1$

retorne ao processamento principal

2 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2]
$\quad$ $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$

retorne ao processamento principal

3 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2] $- um_3 *$ UN[p_3]
$\quad$ $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$

retorne ao processamento principal

... ...

7 : **para** j **de** PJ[ipj] **até** PJF[ipj] **faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2] $- um_3 *$ UN[p_3]
$\quad\quad\quad\quad - um_4 *$ UN[p_4] $- um_5 *$ UN[p_5] $- um_6 *$ UN[p_6]
$\quad\quad\quad\quad - um_7 *$ UN[p_7]
$\quad$ $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1,\ p_4 \leftarrow p_4 + 1$
$\quad$ $p_5 \leftarrow p_5 + 1,\ p_6 \leftarrow p_6 + 1,\ p_7 \leftarrow p_7 + 1$

retorne ao processamento principal

1'' : **para** j **de** PJ[ipj] **até** PJF[ipj] **de 2 em 2 faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1]
$\quad$ UN[$j+1$] $\leftarrow$ UN[$j+1$] $- um_1 *$ UN[$p_1 + 1$]
$\quad$ $p_1 \leftarrow p_1 + 2$

retorne ao processamento principal

2'' : **para** j **de** PJ[ipj] **até** PJF[ipj] **de 2 em 2 faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2]
$\quad$ UN[$j+1$] $\leftarrow$ UN[$j+1$] $- um_1 *$ UN[$p_1 + 1$] $- um_2 *$ UN[$p_2 + 1$]
$\quad$ $p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2$

retorne ao processamento principal

3'' : **para** j **de** PJ[ipj] **até** PJF[ipj] **de 2 em 2 faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2] $- um_3 *$ UN[p_3]
$\quad$ UN[$j+1$] $\leftarrow$ UN[$j+1$] $- um_1 *$ UN[$p_1 + 1$] $- um_2 *$ UN[$p_2 + 1$] $- um_3 *$ UN[$p_3 + 1$]
$\quad$ $p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2,\ p_3 \leftarrow p_3 + 2$

retorne ao processamento principal

... ...

7'' : **para** j **de** PJ[ipj] **até** PJF[ipj] **de 2 em 2 faça**
$\quad$ UN[j] $\leftarrow$ UN[j] $- um_1 *$ UN[p_1] $- um_2 *$ UN[p_2] $- um_3 *$ UN[p_3]
$\quad\quad\quad\quad - um_4 *$ UN[p_4] $- um_5 *$ UN[p_5] $- um_6 *$ UN[p_6]
$\quad\quad\quad\quad - um_7 *$ UN[p_7]
$\quad$ UN[$j+1$] $\leftarrow$ UN[$j+1$] $- um_1 *$ UN[$p_1 + 1$] $- um_2 *$ UN[$p_2 + 1$] $- um_3 *$ UN[$p_3 + 1$]
$\quad\quad\quad\quad\quad\quad - um_4 *$ UN[$p_4 + 1$] $- um_5 *$ UN[$p_5 + 1$] $- um_6 *$ UN[$p_6 + 1$]
$\quad\quad\quad\quad\quad\quad - um_7 *$ UN[$p_7 + 1$]
$\quad$ $p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2,\ p_3 \leftarrow p_3 + 2,\ p_4 \leftarrow p_4 + 2$
$\quad$ $p_5 \leftarrow p_5 + 2,\ p_6 \leftarrow p_6 + 2,\ p_7 \leftarrow p_7 + 2$

retorne ao processamento principal

Cod 5.3-3: *Acumulação Supernodal Múltipla (Colunas)*

Codificação 5.3-4 *Acumulação Supernodal Múltipla (Linhas × Colunas)*

$1'''$: **para** j **de** **PJ**$[ipj]$ **até** **PJF**$[ipj]$ **de** 2 **em** 2 **faça**

$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1+1]$
$\mathbf{UN}[j_2+1] \leftarrow \mathbf{UN}[j_2+1] - um_1^2 * \mathbf{UN}[p_1+1]$
$p_1 \leftarrow p_1 + 2$
$j_2 \leftarrow j_2 + 2$

retorne ao processamento principal

$2'''$: **para** j **de** **PJ**$[ipj]$ **até** **PJF**$[ipj]$ **de** 2 **em** 2 **faça**

$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1+1] - um_2 * \mathbf{UN}[p_2+1]$
$\mathbf{UN}[j_2+1] \leftarrow \mathbf{UN}[j_2+1] - um_1^2 * \mathbf{UN}[p_1+1] - um_2^2 * \mathbf{UN}[p_2+1]$
$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2$
$j_2 \leftarrow j_2 + 2$

retorne ao processamento principal

$3'''$: **para** j **de** **PJ**$[ipj]$ **até** **PJF**$[ipj]$ **de** 2 **em** 2 **faça**

$\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3]$
$\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] - um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2] - um_3^2 * \mathbf{UN}[p_3]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1+1] - um_2 * \mathbf{UN}[p_2+1] - um_3 * \mathbf{UN}[p_3+1]$
$\mathbf{UN}[j_2+1] \leftarrow \mathbf{UN}[j_2+1] - um_1^2 * \mathbf{UN}[p_1+1] - um_2^2 * \mathbf{UN}[p_2+1] - um_3^2 * \mathbf{UN}[p_3+1]$
$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2,\ p_3 \leftarrow p_3 + 2$
$j_2 \leftarrow j_2 + 2$

retorne ao processamento principal

... ...

$7'''$: **para** j **de** **PJ**$[ipj]$ **até** **PJF**$[ipj]$ **de** 2 **em** 2 **faça**

$$\begin{aligned}
\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] &- um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2] - um_3 * \mathbf{UN}[p_3] \\
&- um_4 * \mathbf{UN}[p_4] - um_5 * \mathbf{UN}[p_5] - um_6 * \mathbf{UN}[p_6] \\
&- um_7 * \mathbf{UN}[p_7] \\
\mathbf{UN}[j_2] \leftarrow \mathbf{UN}[j_2] &- um_1^2 * \mathbf{UN}[p_1] - um_2^2 * \mathbf{UN}[p_2] - um_3^2 * \mathbf{UN}[p_3] \\
&- um_4^2 * \mathbf{UN}[p_4] - um_5^2 * \mathbf{UN}[p_5] - um_6^2 * \mathbf{UN}[p_6] \\
&- um_7^2 * \mathbf{UN}[p_7] \\
\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] &- um_1 * \mathbf{UN}[p_1+1] - um_2 * \mathbf{UN}[p_2+1] - um_3 * \mathbf{UN}[p_3+1] \\
&- um_4 * \mathbf{UN}[p_4+1] - um_5 * \mathbf{UN}[p_5+1] - um_6 * \mathbf{UN}[p_6+1] \\
&- um_7 * \mathbf{UN}[p_7+1] \\
\mathbf{UN}[j_2+1] \leftarrow \mathbf{UN}[j_2+1] &- um_1^2 * \mathbf{UN}[p_1+1] - um_2^2 * \mathbf{UN}[p_2+1] - um_3^2 * \mathbf{UN}[p_3+1] \\
&- um_4^2 * \mathbf{UN}[p_4+1] - um_5^2 * \mathbf{UN}[p_5+1] - um_6^2 * \mathbf{UN}[p_6+1] \\
&- um_7^2 * \mathbf{UN}[p_7+1]
\end{aligned}$$

$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2,\ p_3 \leftarrow p_3 + 2,\ p_4 \leftarrow p_4 + 2$
$p_5 \leftarrow p_5 + 2,\ p_6 \leftarrow p_6 + 2,\ p_7 \leftarrow p_7 + 2$
$j_2 \leftarrow j_2 + 2$

retorne ao processamento principal

Cod 5.3-4: *Acumulação Supernodal Múltipla (Linhas × Colunas)*

Codificação 5.3-5 *Eliminação Supernodal Múltipla via Multiplexação*

(2 *linhas* × 2 *colunas*)

($n_{bits} = 3$)

```
        isup ← 0, nsup ← 0, ipj ← 0, implx ← 0                          (a)
        para i de 1 até n de 2 em 2 faça
            ...
            kbeg ← IUT[i], kf ← IUTF[i]
α :         nsup ← nsup +1, kend ← KSUP[nsup]
            ...
            para k de kbeg até kend faça
                atualize apontadores iniciais
                normalize elementos das colunas base
                processe contribuições sobre as diagonais
                determine fatores multiplicativos
            fim para k
            isup ← isup + 1                                             (b)
            implx ← implx + 1                                           (c)
            ipj ← ipj + 1                                               (d)
            ( acumulação restrita das contribuições sobre as linhas base, coluna a coluna )
        ------------------------------------------------------------------
            enquanto ( PJ[ipj] ≠ 0 ) faça                               (e)
                inicialize j2                                           (f)
                va' para ( γ, γ', γ'', γ''' ) em função de LMPLX[implx] (g)
β :             ipj ← ipj + 1
            fim enquanto
        ------------------------------------------------------------------
            kbeg ← kend + 1
            se kend ≠ kf va' para α
            atualize elementos diagonais
        fim para i
        ...
        termine processamento

γ :     va' para ( 1, 2, 3, 4, 5, 6, 7 ) em função de LSUP[isup]        (h)
γ' :    va' para ( 1', 2', 3', 4', 5', 6', 7' ) em função de LSUP[isup]
γ'' :   va' para ( 1'', 2'', 3'', 4'', 5'', 6'', 7'' ) em função de LSUP[isup]
γ''' :  va' para ( 1''', 2''', 3''', 4''', 5''', 6''', 7''' ) em função de LSUP[isup]

1 :     para j de PJ[ipj] até PJF[ipj] faça
            UN[j] ← UN[j] − um1 * UN[p1]
            p1 ← p1 + 1
        va' para β
...     ...
1' :    para j de PJ[ipj] até PJF[ipj] faça
            UN[j]  ← UN[j]  − um1 * UN[p1]
            UN[j2] ← UN[j2] − um1^2 * UN[p1]
            p1 ← p1 + 1
            j2 ← j2 + 1
        va' para β
...     ...
1'' :   para j de PJ[ipj] até PJF[ipj] de 2 em 2 faça
            UN[j] ← UN[j] − um1 * UN[p1]
            UN[j + 1] ← UN[j + 1] − um1 * UN[p1 + 1]
            p1 ← p1 + 2
        va' para β
...     ...
1''' :  para j de PJ[ipj] até PJF[ipj] de 2 em 2 faça
            UN[j]  ← UN[j]  − um1 * UN[p1]
            UN[j2] ← UN[j2] − um1^2 * UN[p1]
            UN[j + 1] ← UN[j + 1] − um1 * UN[p1 + 1]
            UN[j2 + 1] ← UN[j2 + 1] − um1^2 * UN[p1 + 1]
            p1 ← p1 + 2
            j2 ← j2 + 2
        va' para β
...     ...
```

Cod 5.3-5: *Eliminação Supernodal Múltipla via Multiplexação*

Nos passos a partir de (h) processa-se o desvio para o trecho dedicado mais apropriado a cada caso.

Finalmente, nos passos desde 1 até $7'''$, processam-se todas as classes de atualização *simultânea* consideradas.

□

A principal técnica introduzida nesta codificação, é a de *multiplexação* visando a seleção do tipo de códigos a serem processados no tratamento de cada *supernode.*

Tal é efetuado, mediante a introdução de um nível adicional de desvios condicionais computados, no passo (g), transferindo-se a execução para um dos passos desde γ até γ''', e de onde se desvia definitivamente para o trecho de programa correspondente a forma de contribuição a ser efetuada.

Sem a adoção da técnica de *multiplexação*, no caso de codificações com 8 ou mais bits, muitos dos compiladores FORTRAN atuais, não conseguiriam tratar todas as formas variantes possíveis, com 256 casos cada uma, visto que em algumas implementações o limite máximo de casos em desvios do tipo computado situa-se abaixo de 1000. (Mesmo em compiladores onde o número de desvios não é limitado, uma outra restrição pode vir a se fazer notar, como a do número máximo de linhas de continuação, para especificação de todos os desvios, e que em algumas implementações situa-se na faixa de até 20 linhas apenas).

Codificações Binárias Múltiplas

Um outro caso onde se pode aplicar com sucesso a noção de codificações *múltiplas*, é o caso *binário*, adotando-se códigos representativos para padrões de mais de uma coluna contribuinte de forma simultânea.

Um exemplo é ilustrado na Figura 5.5, onde se apresentam todos os possíveis padrões de contribuições de até 2 linhas, sobre 2 colunas simultâneas.

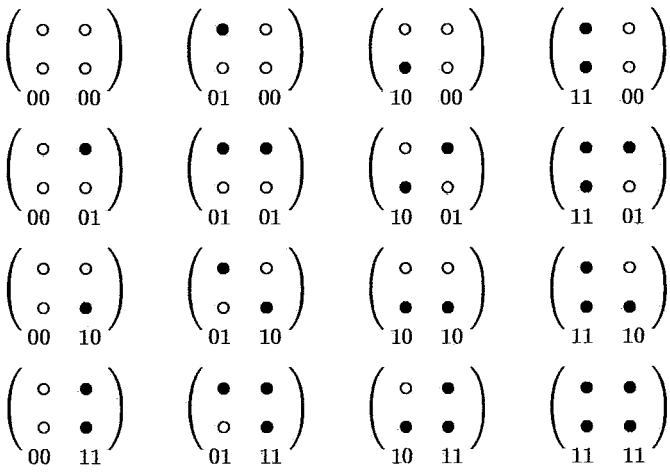


Fig 5.5: Codificação Binária Múltipla

Este exemplo corresponde a uma das possíveis formas de exploração de códigos

de 4 bits, no lugar da forma usualmente adotada até então, correspondendo a contribuições sobre apenas 1 coluna.

O que se ganha com codificações binárias *múltiplas*, é a possibilidade de se explorar plenamente a capacidade de representação de cada código, nos casos onde o número original de contribuições sobre um particular conjunto de colunas seja inferior ao da capacidade máxima de representação isolada (sobre apenas 1 coluna).

Com isso, evita-se o desperdício de bits não utilizados de informação, que estariam inevitávelmente presente, caso se adotassem apenas códigos *plenos* para cada coluna contribuinte.

As possibilidades no caso mais geral, de códigos com 8 ou mais bits, é ilustrada na tabela a seguir, onde se representou todas as possíveis configurações de *linhas* × *colunas* para cada número máximo de bits de representação.

n_{bits}	4	8	9	10	12
	4×1	8×1	9×1	10×1	12×1
	2×2	4×2	4×2	5×2	6×2
	1×4	2×4	3×3	3×3	4×3
		1×8	2×4	2×5	3×4
			1×9	1×10	2×6
					1×12

Apresenta-se a seguir, na Codificação **5.3-6** uma das possibilidades, para o tratamento de códigos de 4 bits (com o processamento simultâneo de até 2 linhas contribuintes, sobre 2 colunas, correspondendo ao caso exemplificado na Figura 5.5).

Nota-se em particular, que os apontadores $p_{(.)}$ são incrementados em alguns casos de apenas 1 unidade (quando apenas uma contribuição não nula se encontra presente a cada linha), e incrementados de 2 unidades quando ambas as colunas possuem contribuição não nula.

Para fins práticos portanto, o tratamento de códigos binários *múltiplos*, processa-se como se houvesse aplicado duas ou mais etapas de decodificação, aplicadas consecutivamente para cada uma das colunas contribuintes envolvidas.

Codificações Múltiplas por Envelope

Um outro caso onde se pode cogitar a aplicação de codificações *múltiplas*, é o do tratamento por *envelopes*.

Neste caso, a informação representável por cada *byte* ou *palavra* de codificação, pode ser aplicada ao processamento de padrões típicos como os apresentados na Figura 5.6 para códigos de até 9 bits.

Os padrões exemplificados na Figura 5.6 correspondem apenas a uma dentre muitas outras possibilidades de representação para configurações por *envelope*.

Codificação 5.3-6 *Acumulação Binária Múltipla (Colunas)*

00 00 : **retorne ao processamento principal**

01 00 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1$
retorne ao processamento principal

10 00 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UN}[p_2]$
$p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

11 00 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

00 01 : $\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1$
retorne ao processamento principal

01 01 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1 + 1]$
$p_1 \leftarrow p_1 + 2$
retorne ao processamento principal

10 01 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

11 01 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1 + 1]$
$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

00 10 : $\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_2 * \mathbf{UN}[p_2]$
$p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

01 10 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

10 10 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_2 * \mathbf{UN}[p_2 + 1]$
$p_2 \leftarrow p_2 + 2$
retorne ao processamento principal

11 10 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_2 * \mathbf{UN}[p_2 + 1]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 2$
retorne ao processamento principal

00 11 : $\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

01 11 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1 + 1] - um_2 * \mathbf{UN}[p_2]$
$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 1$
retorne ao processamento principal

10 11 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2 + 1]$
$p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 2$
retorne ao processamento principal

11 11 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UN}[p_1] - um_2 * \mathbf{UN}[p_2]$
$\mathbf{UN}[j+1] \leftarrow \mathbf{UN}[j+1] - um_1 * \mathbf{UN}[p_1 + 1] - um_2 * \mathbf{UN}[p_2 + 1]$
$p_1 \leftarrow p_1 + 2,\ p_2 \leftarrow p_2 + 2$
retorne ao processamento principal

Cod 5.3-6: *Acumulação Binária Múltipla (Colunas)*

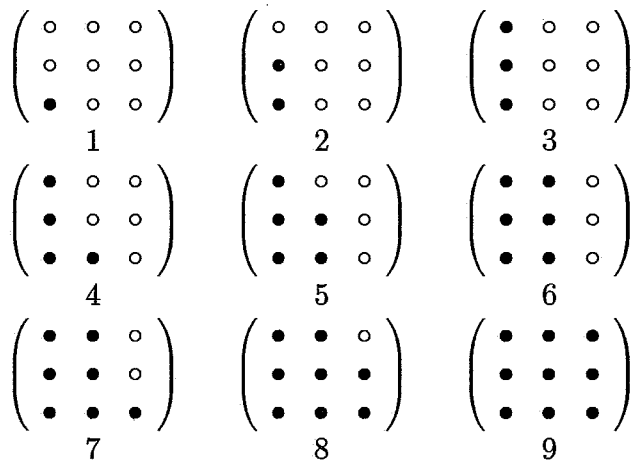


Fig 5.6: Codificação Múltipla por Envelopes

Mediante a adoção de técnicas de *multiplexação* como as anteriormente apresentadas, torna-se possível portanto uma extensão, para o tratamento de um número significativo de outras classes típicamente encontráveis.

Codificações Binárias Recorrentes Multiplexadas

Mediante a exploração dos recursos provenientes da adoção de tabelas de *multiplexação*, mostra-se viável também, uma técnica inteiramente inédita de aproveitamento de códigos, e que pode ser aplicada com extremo sucesso no caso *binário*.

Com a adoção de tais técnicas, consegue-se trivialmente mostrar ser possível a codificação de todo o conjunto de operações de ponto flutuante a serem realizadas ao longo do processo de eliminação, a partir de uma lista de códigos proporcional apenas aos dados de entrada (correspondendo ao número de fatores não nulos a serem operados).

Todos os métodos de codificação empregados até então, embora obtendo considerável compactação das informações necessárias para a caracterização de todo o processo de eliminação, podem em situações de pior caso exibir um comportamento proporcional ao volume de operações artiméticas.

O mesmo ocorre com a metodologia proposta por Gustavson [59], onde o principal ponto desfavorável, é justamente o de sua proporcionalidade ao número de operações a serem efetuadas.

A metodologia apresentada nesta sub-seção, consegue contornar este obstáculo, mediante o reaproveitamento de códigos estruturais, utilizados na caracterização de padrões de contribuição sobre linhas passadas.

Este fato é ilustrado na Figura 5.7, onde se considera o processo de contribuições sobre as etapas básicas associadas as linhas 8 e 9.

Nesta figura, representou-se inicialmente o padrão binário de contribuições, correspondente as tres primeiras linhas a serem utilizadas no processamento sobre as linhas base 8 e 9.

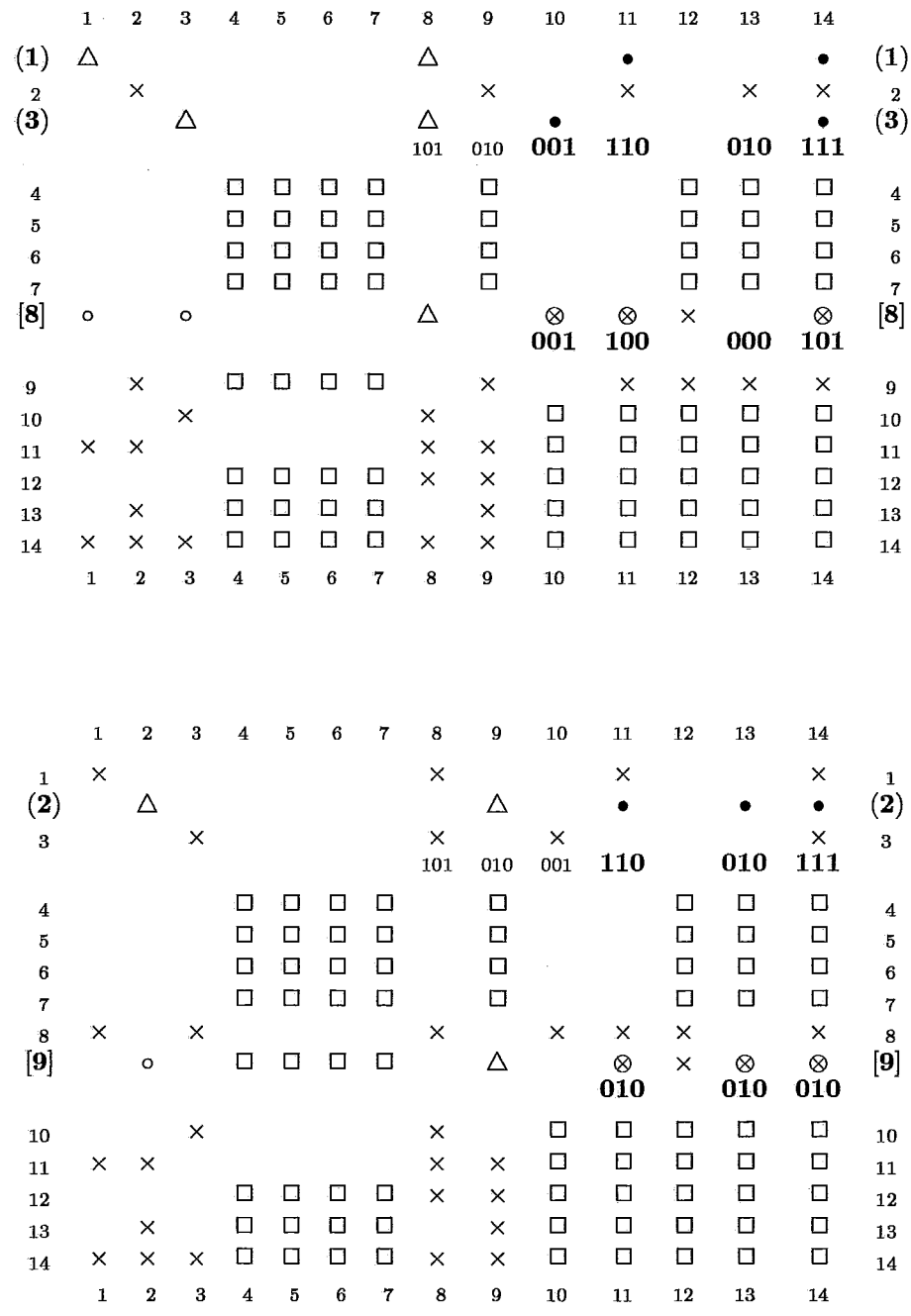


Fig 5.7: Reaproveitamento de Códigos Binários

O principal fato a ser explorado de modo a permitir o reaproveitamento de tais códigos para o processamento sobre cada uma das linhas base isoladamente, é o de que apenas determinados padrões de *bits* devem estar *ativos* por ocasião de seu reprocessamento para novas contribuições.

Tal ocorre, pois apenas um particular sub-conjunto de linhas contribuintes (dentre todas as consideradas na codificação original) pode se mostrar efetivamente necessário por ocasião de seu reaproveitamento para novos casos.

Uma exploração *seletiva* do padrão de *bits* pode ser obtida, lançando-se mão de uma *tabela de remapeamento*, como a ilustrada em 5.1, onde se apresentam os códigos resultantes para cada uma das possíveis exclusões de linhas, numa codificação binária de 3 *bits*.

padrão (*orig*)	(*excl*)	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
000		000	000	000	000	000	000	000	000
001		001	000	001	000	001	000	001	000
010		010	010	000	000	010	010	000	000
011		011	010	001	000	011	010	001	000
100		100	100	100	100	000	000	000	000
101		101	100	101	100	001	000	001	000
110		110	110	100	100	010	010	000	000
111		111	110	101	100	011	010	001	000

Tab 5.1: Remapeamento de Códigos Binários via Multiplexação

Nota-se que os valores remapeados apresentados, correspondem a operação lógica $orig \wedge \overline{excl}$ (efetuada *bit* a *bit*), com *orig* denotando o padrão binário originalmente codificado, e *excl* o padrão de linhas contribuintes a serem excluídas no remapeamento da mesma codificação.

De posse da tabela 5.1, nota-se que o padrão $\{ 001, 100, 000, 001 \}$ de contribuições sobre a linha base 8 pode ser trivialmente obtido a partir do padrão $\{ 001, 110, 010, 111 \}$ originalmente empregado na codificação das tres primeiras linhas (o mesmo se sucedendo com o padrão necessário para o processamento da linha básica 9).

Nota-se também, que para o processamento das contribuições sobre a linha 9, apenas um sub-conjunto terminal dos códigos originalmente adotados para o processamento da linha 8 mostra-se efetivamente necessário (visto que para o processamento das contribuições sobre a linha 9, não ser mais requerido o padrão correspondente a coluna 10).

Outro ponto a se notar, é que em alguns casos (correspondentes a códigos não nulos no padrão de codificação original), por ocasião do processo de exclusão de *bits* de linhas não necessárias, pode-se eventualmente chegar a códigos *nulos* como resultado do remapeamento. (Este fato encontra-se ilustrado por exemplo, no caso da coluna 13, por ocasião do processamento sobre a linha base 8).

A principal característica de listas de padrões binários codificados da forma apresentada, é a de poderem ser acessadas *recorrentemente,* permitindo o compartilhamento de informações *passadas*, para a caracterização de contribuições sobre *distintas* etapas *futuras* ao longo do processo de eliminação.

É essa característica, que confere a esta nova abordagem proposta, o caráter de proporcionalidade apenas aos dados de entrada, e não ao volume de operações aritméticas.

A proporcionalidade aos dados de entrada pode ser trivialmente notada, caso se

lançe mão de um esquema de codificação binária sub-dividindo-se a matriz de fatores original, em grupos de até n_{bits} linhas consecutivas (a terem seus padrões reaproveitados por ocasião do processamento das contribuições sobre suas descendentes futuras).

A lista de códigos assim construída, possuirá seguramente um tamanho inferior a $\eta(U)/n_{bits}$, mostrando ser plenamente viável portanto a utilização de esquemas simbólicos com espaço de codificação proporcional apenas aos dados de entrada.

Apresenta-se a seguir, no Procedimento **5.3(1)**, um processo de eliminação voltado para a exploração de códigos binários de forma *recorrente*, mediante a técnica de *remapeamento de códigos.*

Nota-se que o remapeamento pode ser implementado a partir de operações da forma $\mathbf{LREC}[irec] \wedge \overline{\mathbf{LEXCL}[iexcl]}$, obtendo-se dinâmicamente o trecho correspondente de programa a ser executado em cada caso, ou mediante a adoção de desvios duplamente encadeados, efetuando-se a *multiplexação* necessária.

No procedimento listado, optou-se por esta segunda forma de implementação, visto requerer apenas a incorporação de uma tabela estática de remapeamento de desvios, evitando-se desta forma um *overhead* em tempo de execução, caso se optasse por determinar dinâmicamente os desvios resultantes.

Um fato a se observar com relação ao Procedimento **5.3(1)**, é que embora a lista *recorrente* de códigos binários **LREC** possua neste caso, um espaço proporcional apenas aos dados de entrada, mostra-se necessário a adoção de uma lista auxiliar de posicionamento sobre a linha base, de modo a se poder determinar a cada reaproveitamento, a posição correspondente a coluna originalmente codificada.

Isso pode ser feito a partir de listas *restritas* de intervalos **PJ** e **PJF**, introduzidas na seção anterior, e cujo espaço de armazenamento pode num pior caso vir a se mostrar proporcional ao número de operações aritméticas.

Na prática no entanto, em face a presença de padrões *supernodais*, e ao fato de se representar em cada código o padrão resultante de n_{bits} linhas simultâneas, os intervalos de posições contíguas correspondentes sobre cada linha base (a serem armazenadas nas listas **PJ** e **PJF**) tendem a abranger uma parcela significativa do total de elementos não nulos de cada linha base a ser operada.

Uma solução de modo a tornar o espaço de trabalho do procedimento apresentado, garantidamente proporcional aos dados de entrada, é lançar mão de um vetor auxiliar **IW**, análogo ao empregado no procedimento **4.1(1)**, e que uma vez inicializado a cada etapa base (de modo a conter a posicão de cada um dos elementos não nulos da linha a ser gerada), permite que mediante acessos a **UN[IW[JU[.]]]** se possa implementar de forma alternativa, o passo (h).

Por efetuar os acessos aos elementos sobre a linha base de forma indireta, optou-se por não adotar esta forma alternativa de processamento, no lugar das listas *restritas* de intervalos (onde a seqüencialidade dos padrões de acesso é plenamente explorada).

Apresenta-se a seguir, um detalhamento dos principais passos do Procedimento **5.3(1)**, que lança mão (além das estruturas usualmente adotadas), de um vetor auxiliar **IPREC**, de modo a apontar para a posição recorrente inicial a ser ex-

Procedimento 5.3(1) *Eliminação Binária Recorrente via Multiplexação*

($n_{bits} = 3$)

$ipj \leftarrow 0,\ iexcl \leftarrow 0,\ iprec \leftarrow 0,\ nkend \leftarrow 0$ (*a*)
para i **de** 1 **até** n **faça**
  $piv \leftarrow \mathbf{DI}[i],\ \mathbf{IUP}[i] \leftarrow \mathbf{IU}[i]$
  $kbeg \leftarrow \mathbf{IUT}[i],\ kf \leftarrow \mathbf{IUTF}[i]$
α : $nkend \leftarrow nkend + 1,\ kend \leftarrow \mathbf{IKEND}[nkend]$ (*b*)
  para k **de** *kbeg* **até** *kend* **faça**
    $l \leftarrow \mathbf{JUT}[k],\ r \leftarrow l\ mod\ 3 + \lfloor\ (3 - l\ mod\ 3)\ /\ 3\ \rfloor * 3$ (*c*)
    $iupl \leftarrow \mathbf{IUP}[l],\ \mathbf{IUP}[l] \leftarrow \mathbf{IUP}[l] + 1,\ p_{(r)} \leftarrow \mathbf{IUP}[l]$
    $um_{(r)} \leftarrow \mathbf{UN}[iupl] * \mathbf{DI}[l]$
    $piv \leftarrow piv - \mathbf{UN}[iupl] * um_{(r)}$
    $\mathbf{UN}[iupl] \leftarrow um_{(r)}$
  fim para k
  $iprec \leftarrow iprec + 1,\ irec \leftarrow \mathbf{PREC}[iprec]$ (*d*)
  $iexcl \leftarrow iexcl + 1,\ ipj \leftarrow ipj + 1$ (*e*)
  (acumulação *restrita* das contribuições sobre a *linha base*, coluna a coluna)

---

  enquanto ($\mathbf{PJ}[ipj] \neq 0$) **faça** (*f*)
    para j **de** $\mathbf{PJ}[ipj]$ **até** $\mathbf{PJF}[ipj]$ **faça**
      $s \leftarrow 0.$
      va' para ($\gamma_{000}, \gamma_{001}, \gamma_{010}, \gamma_{011}, \gamma_{100}, \gamma_{101}, \gamma_{110}, \gamma_{111}$)
        em função de $\mathbf{LEXCL}[iexcl]$ (*g*)
β : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - s$ (*h*)
      $irec \leftarrow irec + 1$ (*i*)
    $ipj \leftarrow ipj + 1$ (*j*)
  fim enquanto

  $kbeg \leftarrow kend + 1$
  se $kend \neq kf$ **va' para** α
  $\mathbf{DI}[i] \leftarrow 1.\ /\ piv$
fim para i

termine procedimento

γ_{000} : **va' para** (000, 001, 010, 011, 100, 101, 110, 111) **em função de** $\mathbf{LREC}[irec]$ (*k*)
γ_{001} : **va' para** (000, 000, 010, 010, 100, 100, 110, 110) **em função de** $\mathbf{LREC}[irec]$
γ_{010} : **va' para** (000, 001, 000, 001, 100, 101, 100, 101) **em função de** $\mathbf{LREC}[irec]$
γ_{011} : **va' para** (000, 000, 000, 000, 100, 100, 100, 100) **em função de** $\mathbf{LREC}[irec]$
γ_{100} : **va' para** (000, 001, 010, 011, 000, 001, 010, 011) **em função de** $\mathbf{LREC}[irec]$
γ_{101} : **va' para** (000, 000, 010, 010, 000, 000, 010, 010) **em função de** $\mathbf{LREC}[irec]$
γ_{110} : **va' para** (000, 001, 000, 001, 000, 001, 000, 001) **em função de** $\mathbf{LREC}[irec]$
γ_{111} : **va' para** (000, 000, 000, 000, 000, 000, 000, 000) **em função de** $\mathbf{LREC}[irec]$

000 : **va' para** β
001 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1]$
  $p_1 \leftarrow p_1 + 1$
  va' para β
010 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2]$
  $p_2 \leftarrow p_2 + 1$
  va' para β
011 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1$
  va' para β
100 : $s \leftarrow s + um_3 * \mathbf{UN}[p_3]$
  $p_3 \leftarrow p_3 + 1$
  va' para β
101 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_3 * \mathbf{UN}[p_3]$
  $p_1 \leftarrow p_1 + 1,\ p_3 \leftarrow p_3 + 1$
  va' para β
110 : $s \leftarrow s + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
  $p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
  va' para β
111 : $s \leftarrow s + um_1 * \mathbf{UN}[p_1] + um_2 * \mathbf{UN}[p_2] + um_3 * \mathbf{UN}[p_3]$
  $p_1 \leftarrow p_1 + 1,\ p_2 \leftarrow p_2 + 1,\ p_3 \leftarrow p_3 + 1$
  va' para β

plorada a cada reaproveitamento de códigos, e de uma tabela de *multiplexação* de desvios **LEXCL**, representando o padrão de linhas a serem excluídas a cada etapa do processamento.

Tal lista de desvios é também limitada pelos dados de entrada, uma vez que corresponde a um particionamento por n_{bits} colunas consecutivas, da estrutura de contribuições ditada por U^T.

Descrição de Passos do Procedimento **5.3(1)**

No passo (a) inicializam-se os indexadores para as listas *restritas*, de desvios *multiplexados*, de posições recorrentes iniciais e de delimitação das contribuições a serem processadas a cada etapa.

No passo (b) atualiza-se a posição final *kend* especificada numa lista auxiliar **IKEND**, de modo a se considerar apenas um grupo de até n_{bits} linhas contribuintes consecutivas (adotadas na codificação binária *recorrente* original).

No passo (c) atualiza-se o índice *reduzido* r, assumindo-se uma codificação original, particionada em grupos de n_{bits} linhas consecutivas (correspondendo a representação de padrões de 3 em 3 linhas no caso particular da codificação apresentada).

No passo (d) inicializa-se o apontador para a posição *recorrente* inicial da lista de códigos binários, a serem explorados na etapa corrente.

No passo (e) atualizam-se os indexadores das listas de *multiplexação* de desvio, e de intervalos *restritos*, a serem utilizadas ao longo do passo (f).

No passo (f) processa-se a acumulação das contribuições sobre a linha base, de forma seqüencial *restrita*.

No passo (g) processa-se o desvio *multiplexado*, para um dos pontos rotulados desde γ_{000} até γ_{111}, correspondendo em cada caso, a particular exclusão de um padrão de linhas, de cada um dos códigos binários armazenados em **LREC**, a serem reaproveitados para o acúmulo dos produtos escalares.

No passo (h) incorpora-se o resultado do acúmulo de contribuições para cada coluna, subtraindo-se tal valor da posição correspondente a ser atualizada na linha base.

No passo (i) incrementa-se o indexador *irec* de posições da lista *recorrente* de códigos binários **LREC**.

No passo (j) atualiza-se o indexador da lista *restrita* de intervalos.

Nos passos correspondentes aos trechos desde γ_{000} até γ_{111} a partir do passo (k), processam-se *multiplexadamente* os desvios para o trecho definitivo de programa, a efetuar as operações de contribuições referentes ao padrão a ser atualmente aproveitado da lista *recorrente* de códigos.

□

Esquema Recorrente Voltado para o Aproveitamento de Cache

A utilização de listas *recorrentes* de códigos, permite se estabelecer uma forma mais eficiente de acesso e geração de fatores, ao longo do processo de decomposição.

Recordando-se a noção de *Janelas de Eliminação*, introduzida na seção 4.3, percebe-se que um particionamento das linhas contribuintes de forma consecutiva a cada n_{bits} permite se estabelecer naturalmente *Janelas*, delimitadas por grupos consecutivos de n_{bits} linhas base, a serem operadas por grupos igualmente particionados de n_{bits} linhas contribuintes consecutivas (correspondendo a uma particular faixa de colunas na estrutura de U^T).

Ilustra-se a seguir, nas Figuras 5.8, 5.9, 5.10 e 5.11, quatro etapas sucessivas de um processo de eliminação, assumindo-se um particionamento de linhas naturalmente ditado segundo a estrutura de *supernodes* da matriz de fatores resultante.

O que se percebe em cada caso, é que as únicas linhas base a dependerem de contribuições de cada grupo de linhas codificadas, são justamente aquelas contidas na *Janela de Processamento* naturalmente associada ao grupo de linhas originalmente particionadas.

Como exemplo, na Figura 5.8, percebe-se que as únicas linhas dependendes de contribuições das tres primeiras, são as linhas 8, 9, 10, 11, 13 e 14, e que se encontram confinadas na porção especificada pelas tres primeiras colunas de U^T.

O que se propõe como um esquema de modo a se aproveitar ao máximo recursos como o de memórias *cache*, é que uma vez identificado o grupo de linhas base a serem afetadas pelas contribuições das linhas originalmente codificadas, que se proceda a efetivação das contribuições a serem parcialmente incorporadas a cada uma das linhas base envolvidas.

Este processo, se levado adiante, acabará por gerar completamente os fatores U, a medida que se forem processando as contribuições de novos intervalos de linhas consecutivas, codificadas visando uma exploração *recorrente* de padroes, como a proposta na sub-seção anterior.

Percebe-se que este processo acaba correspondendo a um método de geração (implícita) dos fatores por *submatrizes*, mantendo-se no entanto todas as características básicas de um método de gerações por *linhas* mediante esquemas do tipo *Crout* simétrico.

Esta nova abordagem, permite que se explore pois com sucesso, não apenas os padrões de *recorrência* nas listas de codificações, como se aproveite concomitantemente o acesso ao particular grupo de linhas orignalmente codificadas, para se processar imediatamente todas as operações artiméticas envolvendo seus elementos, o que permite um eficiente aproveitamento de recursos secundários de armazenamento rápido, como o de memórias do tipo *cache* (presente em práticamente todas as arquiteturas escalares de alto desempenho atuais).

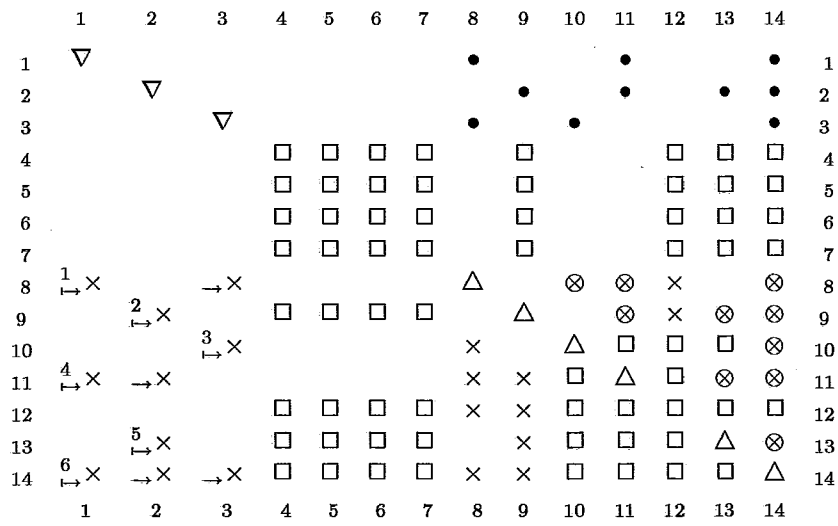


Fig 5.8: Primeira Etapa de Cancelamentos Explorando Cache

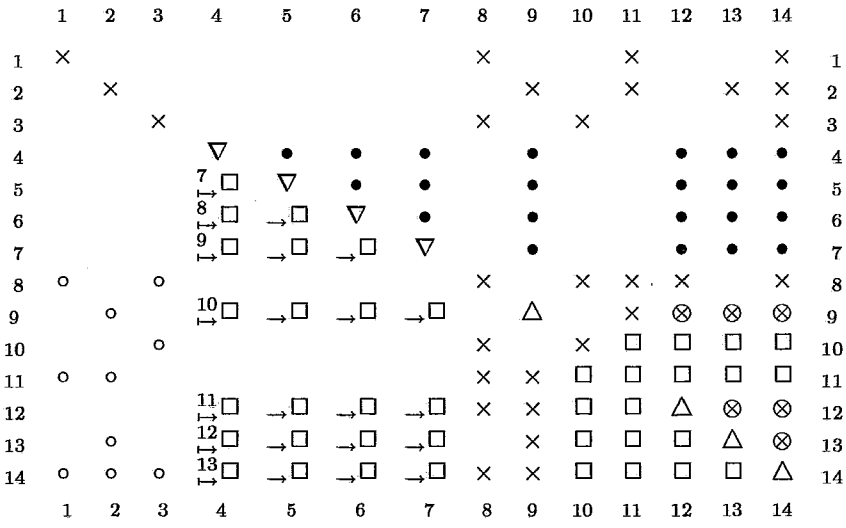


Fig 5.9: Segunda Etapa de Cancelamentos Explorando Cache

5.4 Acumulações via Múltiplos Vetores de Trabalho

Apresenta-se a seguir, uma metodologia baseada na exploração recorrente de códigos binários (como a ilustrada nas figuras 5.8 (**5.8**) a 5.11 (**5.11**) da sub-seção anterior), utilizando porém *múltiplos* vetores de trabalho **W** para o armazenamento temporário dos fatores de cada uma das linhas a contribuírem sobre as linhas base (descendentes) afetadas.

O método a ser considerado é pois do tipo Crout simétrico (com geração dos fa-

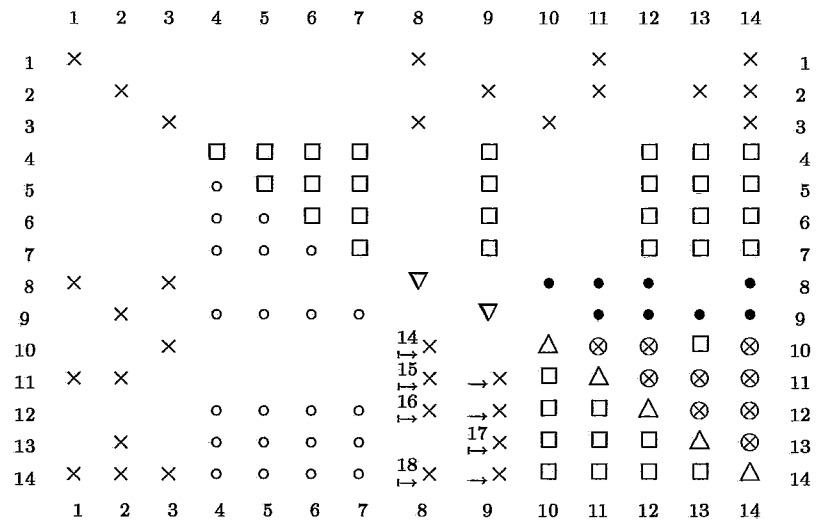


Fig 5.10: Terceira Etapa de Cancelamentos Explorando Cache

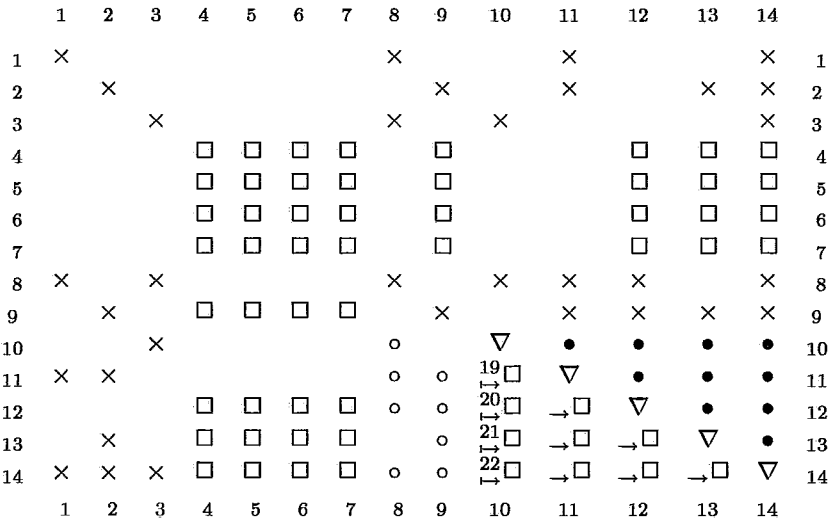


Fig 5.11: Quarta Etapa de Cancelamentos Explorando Cache

tores por *sub-matrizes*), mediante cancelamentos inferiores por *linhas*, gradualmente restritos a faixas sucessivas de *colunas* consecutivas de U^T.

A nova metodologia, baseia-se numa inicialização preliminar dos vetores expandidos (associados a cada linha contribuinte), permitindo seu acesso por colunas, durante o processo de decodificação e acúmulo de cada uma das contribuições.

Apresenta-se esquemáticamente na Figura 5.12 um exemplo em que se ilustra o estado dos vetores **W** após o processo de sua inicialização com os respectivos valores **UN**.

A inicialização dos vetores expandidos não é feita simultâneamente para todas as linhas contribuintes envolvidas numa mesma fase, mas sim apenas oportunamente, a partir do momento em que algum de seus elementos mostra-se necessário para o processo de acúmulo de contribuições sobre uma dada coluna.

Na figura 5.12, destacou-se pois entre colchetes (para cada uma das linha base envolvidas), os índices das linhas ainda não incorporadas a algum dos vetores expandidos $\mathbf{W}$.

Neste caso (a ser ilustrado detalhadamente um pouco mais adiante), nota-se que por ocasião do processamento das contribuições sobre a linha 5, mostra-se necessário que os vetores $\mathbf{W}_1$ e $\mathbf{W}_4$ tenham sido préviamente inicializados com os valores respectivos de **UN**. Por ocasião do processamento das contribuições sobre a linha 7, apenas $\mathbf{W}_2$ necessita ser inicializado (visto que $\mathbf{W}_1$ e $\mathbf{W}_4$ já se encontram inicializados). Finalmente, ao se processarem as contribuições sobre a linha 10, mostra-se necessária a inicialização de $\mathbf{W}_3$ (uma vez que a primeira contribuição não nula da linha 3, ocorre exatamente a partir do elemento $U_{3,10}$).

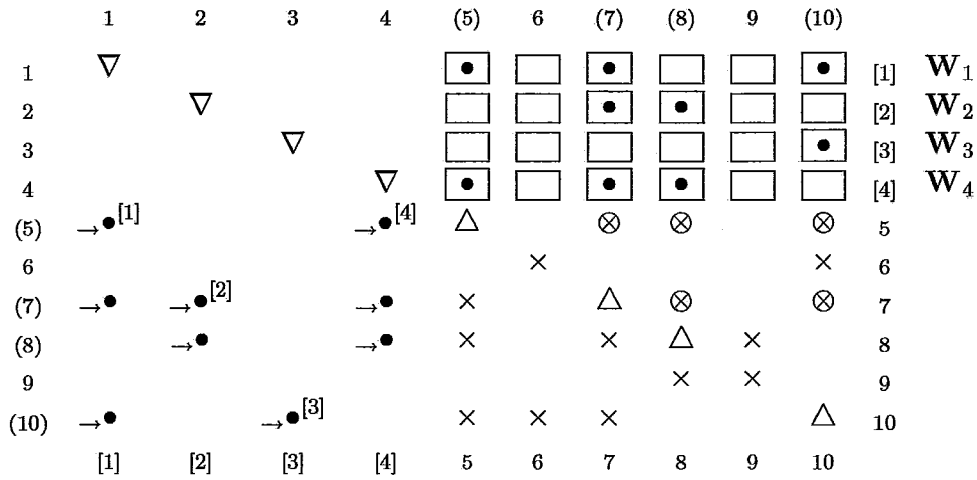


Fig 5.12: Uso de Múltiplos Vetores de Trabalho

Ilustrar-se-á detalhadamente a seguir (nas Figuras 5.13 a 5.16), cada uma das etapas de contribuições sobre as linhas $5, 7, 8$ e 10, identificando-se em cada caso o padrão de alguma das linhas contribuintes $1, 2, 3$ ou 4 que necessitam ser gradualmente inicializadas nos vetores $\mathbf{W}$, para que se possa processar o acúmulo de contribuições.

Nota-se que a determinação do sub-conjunto de linhas a serem inicializadas em $\mathbf{W}$, depende necessáriamente do padrão global de contribuições relativas a cada etapa, excluindo-se deste conjunto, as linhas que porventura tenham sido préviamente inicializadas em alguma etapa passada.

Para facilitar a compreensão deste processo, representou-se auxiliarmente em cada figura, os valores de variáveis *cod*, *excl* e *init* a conterem respectivamente o padrão global de contribuições a serem processadas na etapa corrente, o padrão de linhas já inicializadas (a serem excluídas do padrão global), e o padrão resultante de linhas a serem inicializadas nos vetores $\mathbf{W}$.

Nota-se que o processo de identificação das linhas a serem inicializadas a cada etapa, pode ser implementado a partir de seqüências de operações lógicas (bit a bit) da forma:

$$excl \leftarrow excl \vee cod$$

$$init \leftarrow cod \wedge \overline{excl}$$

assumindo-se que *excl* encontre-se préviamente com um valor nulo ao inicio da primeira etapa.

Uma vez inicializados os vetores $\mathbf{W}_1$ e $\mathbf{W}_4$ (como ilustrado na figura 5.13), pode-se processar o acúmulo das contribuições provenientes das linhas 1 e 4 sobre a linha 5, mediante a técnica de processamento *recorrente* da lista de códigos (via exclusão multiplexada de padrões), descrita na seção anterior.

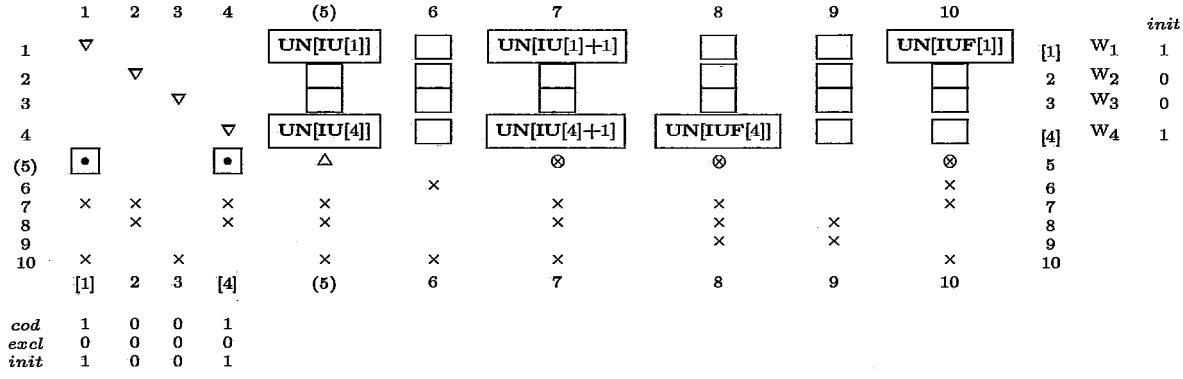


Fig 5.13: Primeira Etapa de Inicializações

Na segunda etapa, após a inicialização de $\mathbf{W}_2$, pode-se proceder ao acúmulo *recorrente* das contribuições sobre a linha 7, a partir das posições correspondentemente armazenadas nos vetores $\mathbf{W}$ a cada coluna.

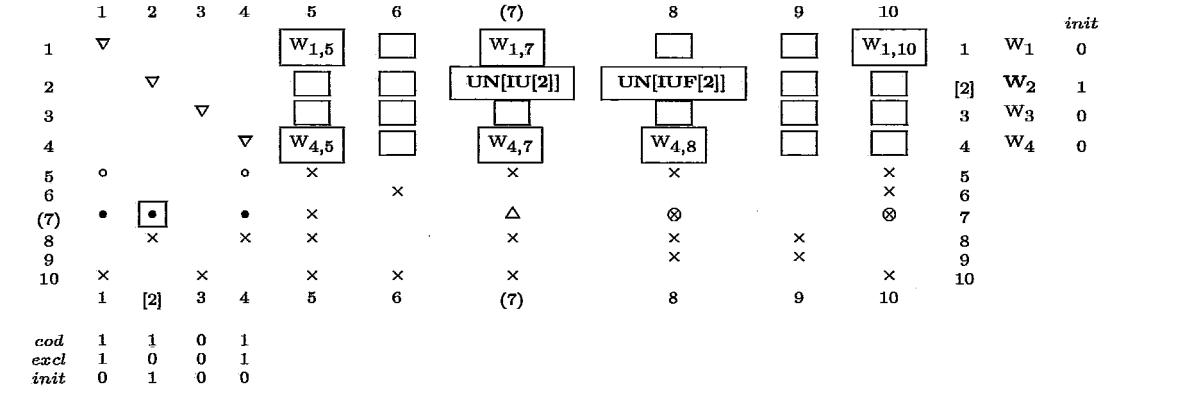


Fig 5.14: Segunda Etapa de Inicializações

Na terceira etapa, não se mostra necessária a inicialização de vetores expandidos, podendo se proceder diretamente ao acúmulo das contribuições sobre a linha 8 (uma vez que todas as linhas contribuintes envolvidas já foram préviamente inicializadas nos vetores expandidos associados, em alguma etapa passada).

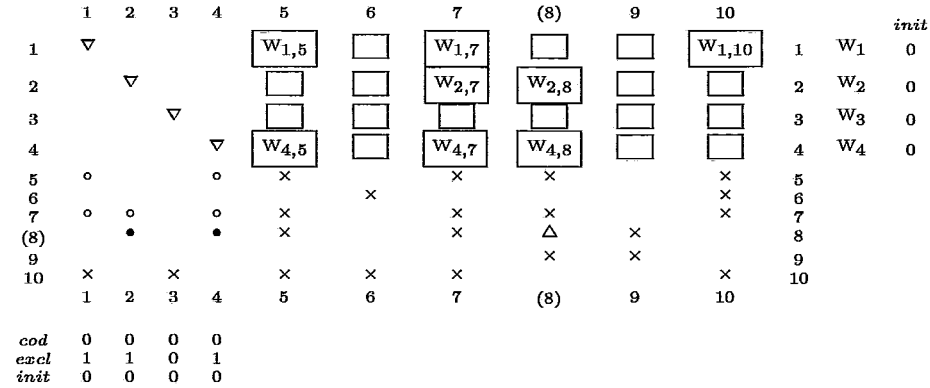


Fig 5.15: Terceira Etapa de Inicializações

Finalmente na quarta etapa, após se processar a inicialização de $\mathbf{W}_3$ pode-se proceder ao acúmulo das contribuições sobre a linha 10, concluindo-se o processo de contribuições relativo as quatro primeiras linhas originalmente consideradas no exemplo.

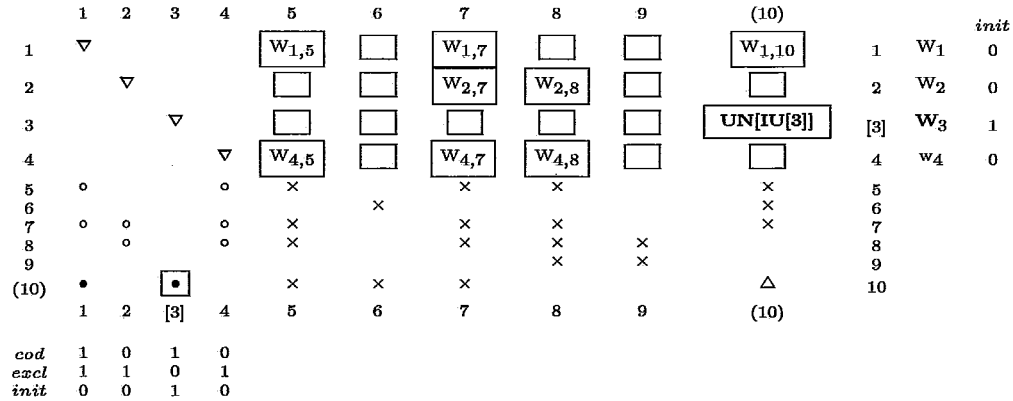


Fig 5.16: Quarta Etapa de Inicializações

Apresenta-se a seguir, na Codificação **5.4-1** um esquema voltado para o processamento de códigos relativos a inicialização dos vetores expandidos, e ao acúmulo de contribuições ditadas por padrões binários a cada coluna.

Nos passos de i_{000} até i_{111}, processa-se para cada etapa, as inicializações relativas ao padrão de especificado pela variável *init* (descrita anteriormente).

Nos passos de 000 até 111 processa-se, o acúmulo das contribuições sobre a linha base, de forma análoga aos métodos baseados em codificações binárias usuais, operando-se porém com os dados préviamente armazenados na coluna p de cada um dos vetores de trabalho expandidos.

Codificação 5.4-1 *Acumulação via Múltiplos Vetores de Trabalho*

i_{000} : **retorne ao processamento principal**
i_{001} : **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{W}_1[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{010} : **para** j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{W}_2[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{011} : **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{W}_1[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{W}_2[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{100} : **para** j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{W}_3[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{101} : **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{W}_1[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{W}_3[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{110} : **para** j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{W}_2[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{W}_3[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
i_{111} : **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{W}_1[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{W}_2[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{W}_3[\mathbf{JU}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal

000 : **retorne ao processamento principal**
001 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{W}_1[p]$
retorne ao processamento principal
010 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{W}_2[p]$
retorne ao processamento principal
011 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{W}_1[p] - um_2 * \mathbf{W}_2[p]$
retorne ao processamento principal
100 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_3 * \mathbf{W}_3[p]$
retorne ao processamento principal
101 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{W}_1[p] - um_3 * \mathbf{W}_3[p]$
retorne ao processamento principal
110 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{W}_2[p] - um_3 * \mathbf{W}_3[p]$
retorne ao processamento principal
111 : $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{W}_1[p] - um_2 * \mathbf{W}_2[p] - um_3 * \mathbf{W}_3[p]$
retorne ao processamento principal

Cod 5.4-1: *Acumulação via Múltiplos Vetores de Trabalho*

Assume-se na codificação apresentada, que a inicialização e atualização das posições a serem endereçadas a partir de p, sejam feitas no procedimento principal (não ilustrado) e análogo aos demais utilizados para o processamento recorrente de códigos binários.

Este fato, confere a esta nova metodologia, uma característica inédita entre todos os métodos baseados em codificações estruturais até então apresentados, por justamente operar-se de forma descompactada, garantindo a necessidade de um *único* apontador p de modo a se obter o acesso a uma dada coluna de cada uma das linhas contribuintes envolvidas. Nos demais métodos tal não se mostrava possível, por estar se operando diretamente sobre a estrutura de elementos **UN** contribuintes (armazenados originalmente de forma seqüencial compacta, e não necessáriamente alinhados por colunas a cada linha).

Isso acarreta, que na nova metodologia, se incremente apenas uma vez o (único) apontador p, ao contrário do incremento explícito de cada um dos apontadores p_l (para cada uma das linhas l envolvidas), no caso dos processamentos até então considerados.

Com isso, reduz-se de forma proporcional ao número total de operações de ponto flutuante, os *overheads* decorrentes do incremento de cada um dos apontadores originais, dispensáveis na atual abordagem.

Esta nova metodologia apresenta também outras vantagens, a serem consideradas um pouco mais adiante, e que permitem se obter níveis ainda mais eficientes de compactação de códigos binários ou por envelopes.

Todo o processamento considerado até então nesta seção, baseou-se na utilização de vetores de trabalho *expandidos* **W**. Ocorre ser possível uma forma análoga de implementação das mesmas técnicas, lancando-se mão de vetores de trabalho *reduzidos* **UW** como passaremos a apresentar adiante, um pouco mais sucintamente.

Ilustra-se na Figura 5.17 um esquema de armazenamento para as mesmas contribuições consideradas originalmente na Figura 5.12, só que agora de uma forma *reduzida*, a partir de uma representação compacta, abrangendo apenas o espectro de colunas contendo contribuições não nulas (dentre o conjunto de todas as linhas a serem simultâneamente processadas).

A estrutura assim obtida (do mesmo modo como a estrutura de vetores expandidos) permite trivialmente o acesso por cada uma das colunas de elementos não nulos de U, viabilizando os mesmos benefícios, como o de um só incremento a um único apontador p, por exemplo.

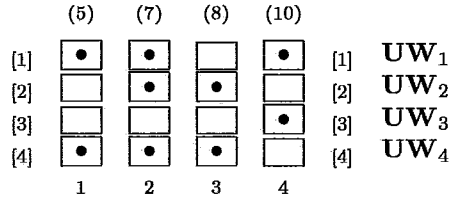


Fig 5.17: Múltiplos Vetores Reduzidos

A grande vantagem na utilização da estrutura de múltiplos vetores *reduzidos* está na significativa economia de espaço adicional de trabalho, uma vez que os vetores **UW** necessitam ser dimensionados apenas para o número máximo de elementos não nulos passíveis de serem encontrados dentre todas as linhas de U (o que normalmente situa-se bem aquém da dimensão n requerida pelas abordagens mediante vetores *expandidos*). Uma vez que processam-se normalmente 8 a 10 contribuições simultâneas (em função do número de bits adotado para codificação), o total adicional de espaço dipendido com o uso de vetores *expandidos* passa a ser de 8 n a 10 n, em contrapartida a 8 $\times \max \{ \eta^{(i)}(U) \}$ ou 10 $\times \max \{ \eta^{(i)}(U) \}$ no caso de vetores *reduzidos*.

Apresenta-se a seguir, na Codificação **5.4-2**, um esquema análogo ao da codificação anterior, só que voltado para a acumulação de contribuições mediante vetores *reduzidos*.

No caso da codificação **5.4-2**, mostra-se necessário um vetor auxiliar **IUW** de dimensão igual ao número de fatores não nulos de U, a fim de permitir o fácil mapeamento de cada um dos valores originalmente armazenados em **UN** a cada linha, para as posições equivalentes (não necessáriamente contíguas) nas estruturas *reduzidas* **UW**, durante a fase de inicializações.

Assumindo-se que a especificação das posições j associadas nas linhas base sendo geradas, seja fornecida por meio de uma lista tradicional de endereços **PSUB**, nota-se que a diferença básica no padrão de inicialização e acessos aos apontadores p de ambas as codificações apresentadas, é que no caso do acúmulo por vetores *expandidos*, procede-se usualmente a $j \leftarrow \mathbf{PSUB}[nsub]$, $nsub \leftarrow nsub + 1$, $p \leftarrow \mathbf{JU}[j]$, ao passo que no acúmulo por vetores *reduzidos*, pode-se lançar mão de apenas um incremento simples $p \leftarrow p + 1$ ao apontador p (além do acesso a lista **PSUB** para a determinação da posição j correspondente, na linha base).

Compactação Mediante o Reordenamento de Contribuições por Colunas

A técnica de *múltiplos* vetores de trabalho permite que se alcance como subproduto, níveis ainda mais elevados de compactação de códigos, tanto para padrões *binários* como por *envelope*.

Em face a similiaridade de tratamento para ambos os casos, apenas codificações *binárias* serão exemplificadas nesta sub-seção.

Inicialmente nota-se para o exemplo na Tabela 5.2, que no caso da presença de um número não unitário de colunas contribuintes com um mesmo padrão, pode-se lançar mão de um esquema baseado numa *Tabela de Índices*, de modo a se substituir os padrões *binários* originais, por respectivos *índices* na tabela auxiliar (onde se encontram armazenados apenas os padrões distintos necessários para a caracterização de todo o espectro de contribuições por colunas considerado).

Para o exemplo em questão, percebe-se que o número total de bits necessários para a caracterização das contribuições mediante o uso da tabela de índices, é justamente a metade dos 44 bits que seriam dispendidos, caso não se adotasse um esquema compacto para a representação.

Uma solução ainda mais econômica é se lançar mão de esquemas de codificação

Codificação 5.4-2 *Acumulação via Múltiplos Vetores Reduzidos*

$i_{000'}$: **retorne ao processamento principal**
$i_{001'}$: **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{UW}_1[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{010'}$: **para** j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{UW}_2[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{011'}$: **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{UW}_1[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{UW}_2[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{100'}$: **para** j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{UW}_3[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{101'}$: **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{UW}_1[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{UW}_3[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{110'}$: **para** j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{UW}_2[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{UW}_3[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal
$i_{111'}$: **para** j **de** $\mathbf{IU}[l_1]$ **até** $\mathbf{IUF}[l_1]$ **faça**
$\mathbf{UW}_1[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_2]$ **até** $\mathbf{IUF}[l_2]$ **faça**
$\mathbf{UW}_2[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
para j **de** $\mathbf{IU}[l_3]$ **até** $\mathbf{IUF}[l_3]$ **faça**
$\mathbf{UW}_3[\mathbf{IUW}[j]] \leftarrow \mathbf{UN}[j]$
retorne ao processamento principal

$000'$: **retorne ao processamento principal**
$001'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UW}_1[p]$
retorne ao processamento principal
$010'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UW}_2[p]$
retorne ao processamento principal
$011'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UW}_1[p] - um_2 * \mathbf{UW}_2[p]$
retorne ao processamento principal
$100'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_3 * \mathbf{UW}_3[p]$
retorne ao processamento principal
$101'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UW}_1[p] - um_3 * \mathbf{UW}_3[p]$
retorne ao processamento principal
$110'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_2 * \mathbf{UW}_2[p] - um_3 * \mathbf{UW}_3[p]$
retorne ao processamento principal
$111'$: $\mathbf{UN}[j] \leftarrow \mathbf{UN}[j] - um_1 * \mathbf{UW}_1[p] - um_2 * \mathbf{UW}_2[p] - um_3 * \mathbf{UW}_3[p]$
retorne ao processamento principal

Cod 5.4-2: *Acumulação via Múltiplos Vetores Reduzidos*

Padrão por Colunas

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)
0	1	0	1	0	0	1	1	0	1	0
1	1	1	0	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0	1	0	1
1	0	1	1	0	1	0	0	0	0	0
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
[0]	[3]	[0]	[2]	[1]	[0]	[3]	[3]	[1]	[3]	[1]
00	11	00	10	01	00	11	11	01	11	01

Tabela de Índices

00	[0]	→	0	1	0	1
01	[1]	→	0	1	1	0
10	[2]	→	1	0	0	1
11	[3]	→	1	1	0	0

Tab 5.2: Codificação via Tabela de Índices

baseados em tabelas do tipo Huffman, onde simplesmente se representa por códigos com um menor número de bits, os padrões mais freqüentemente encontrados.

Assim, para o exemplo em questão, mediante o uso de Tabelas de Huffman, percebe-se na Tabela 5.3, que o número total de bits necessário para a codificação dos padrões de contribuição das 11 colunas envolvidas, passa a ser de apenas 15 bits, em comparação com os 22 utilizados numa codificação por *Tabelas de Índices.*

Padrão por Colunas

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)
0	1	0	1	0	0	1	1	0	1	0
1	1	1	0	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0	1	0	1
1	0	1	1	0	1	0	0	0	0	0
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
[1]	[0]	[1]	[01]	[00]	[1]	[0]	[0]	[00]	[0]	[00]
1	0	1	01	00	1	0	0	00	0	00

Tabela de Huffman

0	[0]	→	1	1	0	0
1	[1]	→	0	1	0	1
00	[00]	→	0	1	1	0
01	[01]	→	1	0	0	1

Tab 5.3: Codificação via Tabela de Huffman

O que não se considerou até então, foi uma extensão do nível de reordenamento das contribuições, de modo a se levar o padrão de colunas a uma forma desejável, capaz de permitir uma maior compactação de informações.

Apresenta-se a seguir, uma forma de se alcançar formas desejáveis, e que só pode ser plenamente adotada por métodos baseados em *múltiplos* vetores de trabalho (como os considerados nesta seção).

A idéia consiste simplesmente em se reordenar de forma crescente, os códigos (originais, ou os por tabela de índices ou de Huffman), de tal sorte que ao final do processo se possa aplicar uma nova técnica de codificação, baseada na noção de *Run Length Encoding* (*RLE*), apresentada na Tabela 5.4.

Padrão Reordenado por Colunas

(b)	(g)	(h)	(j)	(a)	(c)	(f)	(e)	(i)	(k)	(d)
1	1	1	1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	1	1	1	0	0	0	1
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
[0]	[0]	[0]	[0]	[1]	[1]	[1]	[00]	[00]	[00]	[01]
0	0	0	0	1	1	1	00	00	00	01

Tabela de Huffman

0	[0]	→	1	1	0	0
1	[1]	→	0	1	0	1
00	[00]	→	0	1	1	0
01	[01]	→	1	0	0	1

Tab 5.4: Reordenamento por Colunas

O que se obtém com o reordenamento crescente (da codificação por colunas), é a possibilidade de se lançar mão de codificações da forma *supernodal*, ou seja com um dado padrão de bits de contribuição, se mantendo constante por todo um intervalo de colunas contribuintes.

Padrão Reordenado por Colunas (RLE)

$4 \times [0]$	$3 \times [1]$	$3 \times [00]$	$1 \times [01]$
100 0	11 1	11 00	1 01

Tabela de Huffman

0	[0]	→	1	1	0	0
1	[1]	→	0	1	0	1
00	[00]	→	0	1	1	0
01	[01]	→	1	0	0	1

Padrão de Freqüência RLE (Huffman)

$\underline{00} \times [0]$	$\underline{0} \times [1]$	$\underline{0} \times [00]$	$\underline{1} \times [01]$
00 0	0 1	0 00	1 01

Tabela de Freqüência

0	$\underline{0}$	→	3
1	$\underline{1}$	→	1
00	$\underline{00}$	→	4

Tab 5.5: Codificação por Colunas (RLE) e Freqüências (Huffman)

A técnica RLE consiste justamente em se identificar os padrões repetitivos, e agrupá-los de forma tal, que se possa caracterizar completamente o padrão de contribuições, mediante a especificação de duas informações básicas: a freqüência de ocorrência de um dado código, e o valor (ou um índice para uma tabela) a ele associado.

Para o exemplo em questão, ilustrou-se inicialmente na Tabela 5.5, uma codificação para o padrão reordenado por colunas, onde os bits iniciais de cada código, são utilizados para a especificação de sua freqüência de ocorrência, e os bits restantes usados para a representação do código de Huffman associado ao padrão de contribuições.

Percebe-se no entanto, que na presença de mais de um padrão com a mesma *freqüência de ocorrência*, que a técnica de codificação de Huffman, pode ser aplicada

para uma codificação adicional, do Padrão de Freqüências encontrado.

Para este caso, mostra-se desejável a utilização de códigos de Huffman com um menor número de bits, para os Padrões de Freqüência mais repetitivamente encontrados. Como critério adicional para a determinação da codificação adicional por Huffman (nos casos com um número idêntico de ocorrências de um dado Padrão de Freqüência), nota-se ser desejável proceder-se a uma codificação de Huffman com menos bits para os Padrões de Contribuições originalmente representados com o maior número de bits. (A razão para tal é que desta forma compensa-se com um menor código de Huffman para o Padrão de Freqüência, os casos com os maiores códigos de Huffman associados ao Padrão de Contribuições).

No caso do exemplo considerado na Tabela 5.5, nota-se que a lista de códigos resultante passou a requerer desta forma, um total de apenas 11 bits, para a completa caracterização de todo o padrão de contribuições das 11 colunas (correspondendo deste modo a uma redução para 1/4 do tamanho da codificação *binária* original).

Considerações Finais sobre o Novo Método Apresentado

As técnicas de compactação apresentadas só são possíveis num esquema mediante *múltiplos* vetores de trabalho, por requererem um reordenamento (implícito) da ordem de processamento das contribuições (por cada grupo de colunas com um mesmo padrão). Nos métodos usuais, acessando diretamente os elementos **UN** de cada linha contribuinte, tal não se mostra possível (com a exceção de processamentos da forma *supernodal*), em função do armazenamento seqüencial compacto (com padrões distintos) adotado para cada uma das linhas da estrutura de fatores.

Com o uso de múltiplos vetores, o processo de atualização do apontador p (indicador da coluna a ser correntemente processada) não necessita corresponder a uma varredura linear crescente em função dos índices de elementos não nulos de cada linha base, podendo seguir naturalmente a seqüência ditada pelo reordenamento de colunas (visando uma maior compactação). Tal é possível, pois nota-se que caso se especifique (a partir de uma lista de posições **PSUB**) uma dada ordem para o acesso aos elementos j sobre a linha base sendo gerada, os valores associados de p podem ser trivialmente obtidos a partir de $p \leftarrow \mathbf{JUT}[j]$ com $j \leftarrow \mathbf{PSUB}[nsub]$.

Um outro fato a se notar com relação a técnica de processamento ilustrada na Figura 5.12 é que se dispensa completamente a necessidade da estrutura de **JUT**, até então adotada em todos os métodos considerados neste trabalho.

Tal ocorre, pois os padrões de contribuições por colunas, codificados e armazenados na lista *recorrente*, correspondem exatamente ao padrão binário de linhas contribuintes sobre cada linha base a ser operada.

Como exemplo, tomemos as contribuições sobre a linha 5, e que provém das linhas 1 e 4. Nota-se que tal padrão 1001 é exatamente o codificado para o processamento de contribuições sobre a coluna 5. No padrão imediatamente posterior codificado na lista *recorrente*, corresponde ao da coluna 7, nota-se que a codificação 1101 corresponde exatamente ao padrão de linhas contribuintes a serem processadas para a geração parcial da linha 7. O padrão 0101 seguinte, correspondente a coluna 8, é

exatamente o mesmo necessário para a especificação das contribuições sobre a linha 8, o mesmo se sucedendo com o código 1010 relativo a coluna 10, e que corresponde ao padrão parcial de contribuições sobre a linha 10 (confinado exclusivamente as quatro primeiras linhas consideradas).

O método apresentado, dispensa pois completamente as estruturas **IUT**, **JUT**, **IUTF**, pelo fato do padrão de linhas contribuintes poder ser determinado dinâmicamente sem qualquer *overhead*, a partir do mesmo padrão adotado para a codificação de cada uma das colunas contribuintes.

Um outro ponto extremamente favorável é o fato de com isso, se ganhar em termos de uma melhor exploração de *cache*, visto que ao se acessar *recorrentemente* a lista de códigos, tanto para o processamento das contribuições a cada coluna, como para a determinação do padrão de linhas contribuintes, todos os valores necessários estarão disponíveis em meio rápido de armazenamento, após a primeira varredura da lista de codificações referente a fase considerada no processo de eliminações.

Um fato interessante com relação a metodologia proposta, é que ela de uma certa forma *"simula"* um mecanismo similar ao utilizado durante acessos a meios secundários como os de *cache*, *paginação*, ou memória *virtual*, em que só se buscam dados de um meio secundário para um nível mais rápido de armazenamento, nos casos em que a informação necessária não se encontra préviamente já armazenada na estrutura de acesso mais rápida para onde os dados deveriam ser transferidos.

Isso é explorado no método apresentado, por ocasião da inicialização dos *múltiplos* vetores de trabalho, em que se considera apenas a inicialização dos vetores cujos valores de **UN**, ainda não tenham sido transferidos para a estrutura temporária de trabalho.

Finalmente, percebe-se que toda a metodologia proposta, pode ser diretamente implementada a nível de circuitos de *hardware*, uma vez que opera e acessa exclusivamente com padrões a nível binário (ou por *envelopes*), adotados para fins de codificação.

Não é de todo ocasional portanto, que as técnicas necessárias para a implementação desta metodologia, se fundamentem em *multiplexações*, ou operações *lógicas* de *and*, *or* ou *negações*, trivialmente implementáveis a nível de circuitos lógicos de *hardware*.

Convém pois notar, que todas as metodologias apresentadas neste capítulo, possuem como característica comum, o fato de se basearem numa técnica de pré-compilação de dados (gerando-se as codificações associadas), a serem posteriormente *"interpretadas"* durante o processo de decodificação, mediante os desvios dedicados a cada um dos trechos de programa préviamente já compilados (uma única vez, por ocasião da montagem da biblioteca de subrotinas).

É esse fato que confere a tais abordagens, a possibilidade de se explorar tanto a eficiência computacional a nível de menores tempos de execução, como a nível de menores espaços temporários de trabalho, a partir de técnicas voltadas para a compactação de informações.

Percebe-se assim, que ambos os requisitos e objetivos apresentados na seção

2.4 podem ser plenamente alcancáveis a partir de técnicas baseadas na noção de *codificações estruturais* para a resolução de sistemas lineares esparsos, originalmente introduzida no presente trabalho.

Capítulo 6

Resultados Computacionais

Neste capítulo apresentam-se resultados computacionais preliminares, visando a comparação do desempenho de algumas das metodologias apresentadas ao longo do texto, em duas classes de arquiteturas monoprocessadas típicamente disponíveis nos dias atuais.

Nota-se particularmente, que o desempenho final obtido (por rotinas codificadas exclusivamente em FORTRAN 77), não ilustra plenamente o potencial de aprimoramento de algumas das técnicas propostas, e que necessitariam de uma programação mais refinada em C/C^{++} ou ASSEMBLER para uma melhor exploração de toda a gama de recursos de *hardware* ou de *software* passíveis de incorporação nos códigos apresentados.

Para fins de comparação, no primeiro caso utilizou-se uma configuração *escalar* IBM-PC compatível, com processador INTEL 80486/DX-2 com clock interno de 66 Mhz, barramento ISA a 11 Mhz, 8 Mb de memória principal em *zero wait-state* (com tempo de acesso de 70 nano-segundos), 256 Kb de memória *cache* externa em *zero wait-state* (com tempo de acesso de 20 nano-segundos e esquema de leitura em 2-1-1-1 ciclos), rodando sob um *bios* AMI e *chipset* OPTi modelo OF495SLC.

Nesta arquitetura, o ambiente de programação contou com o MS-DOS 6.2, gerenciador de memória EMM386, e compilador WATCOM FORTRAN versão 9.01, rodando sob o *dos extender* DOS/4GW.

No segundo caso, utilizou-se um dos nós de um *cluster* IBM/SP2, formado por estações *superescalares* IBM-RS/6000 modelo 370.

Nesta arquitetura, contou-se com o sistema operacional AIX, e com o compilador FORTRAN XLF/F77 do próprio fabricante.

Os problemas teste considerados foram extraídos de aplicações do mundo real, a partir de dois conjuntos básicos, disponíveis na NETLIB e na Harwell-Boeing.

No primeiro caso, os problemas utilizados foram convertidos do formato padrão MPS (voltado originalmente para programação linear), obtendo-se matrizes simétricas, definidas positivas, mediante o produto de matrizes $A\ A^T$ (envolvendo a matriz de coeficientes do problema de programação linear original).

No segundo caso, os problemas foram extraídos do formato Harwell-Boeing, e nos casos em que apenas a estrutura de elementos não nulos estava originalmente disponível, tornou-se a matriz do sistema arbitráriamente definida positiva, mediante

a imposição de valores suficientemente dominantes sobre sua diagonal.

Apresenta-se a seguir, uma série de tabelas, resumindo os resultados e estatísticas coletadas, incialmente para a arquitetura IBM-PC/486, e posteriormente para uma IBM-RS/6000.

	problema	n	$\eta_{\sup(A)}$	*fill-in's*	(%)	$\eta_{(U)}$	(%)	*flops*	
1	*afiro*	27	90	17	(15.9)	107	(29.4)	455	1
2	*adlittle*	56	377	27	(6.7)	404	(25.8)	3515	2
3	*sc205*	205	654	528	(44.7)	1182	(5.6)	7612	3
4	*share1b*	96	871	155	(15.1)	1026	(22.2)	22880	4
5	*scorpion*	388	1915	409	(17.6)	2324	(3.0)	20979	5
6	*sctap1*	300	1686	981	(36.8)	2667	(6.0)	29526	6
7	*scagr25*	471	2370	578	(19.6)	2948	(2.6)	22324	7
8	*israel*	174	11097	162	(1.4)	11259	(74.4)	1011965	8
9	*brandy*	220	2190	660	(23.2)	2850	(11.8)	101861	9
10	*bandm*	305	2929	1185	(28.8)	4114	(8.8)	96143	10
11	*scfxm1*	330	3143	1820	(36.7)	4963	(9.2)	106152	11
12	*e226*	223	2683	733	(21.5)	3416	(13.8)	84349	12
13	*scrs8*	490	1953	3181	(62.0)	5134	(4.2)	170003	13
14	*beaconfd*	173	1720	7	(0.4)	1727	(11.6)	70292	14
15	*scsd6*	147	2099	446	(17.5)	2545	(23.6)	47455	15
16	*ship04s*	402	2827	307	(9.8)	3134	(3.8)	49654	16
17	*czprob*	929	6265	390	(5.9)	6655	(1.6)	106113	17

Tab 6.1: Dados Característicos dos Problemas

Na tabela 6.1 indicou-se grandezas características, como o a dimensão, número de elementos não nulos originais, de *fill-in's* e de fatores, bem como os percentuais do número de *fill-in's* sobre o número de elementos não nulos, e o percentual destes elementos, sobre o número total de elementos de cada matriz. Adicionalmente apresentou-se também o número de operações de ponto flutuante necessárias para a solução de cada problema.

Na tabela 6.2 apresentou-se uma medida de tempo de várias fases do processamento esparso, visando dar uma noção da ordem de grandeza requerida por cada uma em comparação com as demais. Adicionalmente enumerou-se o percentual de tempo dispendido na fase de substituições, em comparação com o tempo médio dispendio na fase de fatoração.

Na tabela 6.3 indicou-se o tempo necessário para a geração de algumas das formas de codificações consideradas, bem como de fases preliminares auxiliares, como a de *reordenamento ancestral*. A título de comparação, ilustrou-se na mesma tabela, os tempos dispendidos nas fases de fatoração e de substituições.

Na tabela 6.4 apresentou-se os tempos de execução de rotinas *convencionais* de fatoração, baseadas em vetores de trabalho (sem exploração de *supernodes*), com geração dos fatores por *colunas* e por *linhas*. A título de comparação, ilustrou-se conjuntamente, o tempo dispendido pela fase de sustituições.

Na tabela 6.5 foi apresentado o desempenho das rotinas convencionais, mensurando-se a taxa de Milhões de operações de ponto flutuante efetuadas por segundo, em cada

	problema	Orden.	F.Simb.	Transp.	Fator.	Subst.	(%)	
1	*afiro*	9.94E-03	1.70E-04	2.20E-04	2.88E-04	2.20E-04	(76.4)	1
2	*adlittle*	1.75E-02	4.40E-04	7.60E-04	1.72E-03	7.80E-04	(45.3)	2
3	*sc205*	2.25E-02	1.65E-03	2.20E-03	4.29E-03	3.00E-03	(69.9)	3
4	*share1b*	4.77E-02	1.95E-03	3.00E-03	9.60E-03	2.75E-03	(28.6)	4
5	*scorpion*	6.10E-02	3.30E-03	5.50E-03	1.06E-02	5.50E-03	(51.9)	5
6	*sctap1*	6.78E-02	3.30E-03	4.95E-03	1.38E-02	5.50E-03	(39.9)	6
7	*scagr25*	5.88E-02	4.30E-03	6.60E-03	1.15E-02	7.20E-03	(62.6)	7
8	*israel*	1.74E-00	1.70E-02	2.70E-02	4.60E-01	2.80E-02	(6.1)	8
9	*brandy*	2.07E-01	3.80E-03	7.10E-03	3.92E-02	7.70E-03	(19.6)	9
10	*bandm*	2.41E-01	6.10E-03	9.90E-03	3.99E-02	9.80E-03	(24.6)	10
11	*scfxm1*	2.01E-01	6.60E-03	1.10E-02	4.35E-02	1.10E-02	(25.3)	11
12	*e226*	1.96E-01	4.40E-03	7.80E-03	3.37E-02	6.60E-03	(19.6)	12
13	*scrs8*	2.55E-01	7.60E-03	1.32E-02	6.93E-02	1.86E-02	(26.8)	13
14	*beaconfd*	1.93E-01	3.55E-03	5.75E-03	2.78E-02	6.60E-03	(23.7)	14
15	*scsd6*	9.23E-02	2.80E-03	4.90E-03	1.90E-02	5.50E-03	(28.9)	15
16	*ship04s*	1.49E-01	4.40E-03	7.60E-03	2.17E-02	8.20E-03	(37.8)	16
17	*czprob*	9.89E-01	6.00E-03	2.20E-02	5.53E-02	2.54E-02	(45.9)	17

Tab 6.2: Tempo (seg) Várias Fases (PC-486)

	problema	CodBin	CodEnv	Reorden.	Fator.	Subst.	
1	*afiro*	1.60E-04	2.70E-04	2.20E-04	2.88E-04	2.20E-04	1
2	*adlittle*	1.20E-03	1.76E-03	9.80E-04	1.72E-03	7.80E-04	2
3	*sc205*	3.00E-02	4.65E-03	3.05E-03	4.29E-03	3.00E-03	3
4	*share1b*	6.55E-03	9.35E-03	3.30E-03	9.60E-03	2.75E-03	4
5	*scorpion*	8.20E-03	1.10E-02	5.50E-03	1.06E-02	5.50E-03	5
6	*sctap1*	9.90E-03	1.70E-02	7.40E-03	1.38E-02	5.50E-03	6
7	*scagr25*	9.30E-03	1.37E-02	1.10E-02	1.15E-02	7.20E-03	7
8	*israel*	2.53E-01	4.78E-01	2.20E-02	4.60E-01	2.80E-02	8
9	*brandy*	2.80E-02	3.73E-02	7.70E-03	3.92E-02	7.70E-03	9
10	*bandm*	2.84E-02	5.18E-02	1.16E-02	3.99E-02	9.80E-03	10
11	*scfxm1*	3.06E-02	5.40E-02	1.10E-02	4.35E-02	1.10E-02	11
12	*e226*	2.42E-02	3.64E-02	8.80E-03	3.37E-02	6.60E-03	12
13	*scrs8*	5.38E-02	1.84E-01	1.86E-02	6.93E-02	1.86E-02	13
14	*beaconfd*	1.97E-02	3.24E-02	6.35E-03	2.78E-02	6.60E-03	14
15	*scsd6*	1.37E-02	1.98E-02	6.60E-03	1.90E-02	5.50E-03	15
16	*ship04s*	1.75E-02	3.90E-02	8.80E-03	2.17E-02	8.20E-03	16
17	*czprob*	4.18E-02	1.48E-01	2.20E-02	5.53E-02	2.54E-02	17

Tab 6.3: Tempo (seg) Geração de Codificações (PC-486)

um dos casos. Estes valores ilustram quais os problemas em que um melhor aproveitamento de recursos é possivel, em face a uma taxa reduzida, em função de *overheads* intrínsecos não contornados.

Nota-se particularmente que o desempenho da fase de substituições, mostrou-se consistentemente, sempre aquém do obtido na fase de fatoração, indicando ser esta uma potencial fase a demandar maiores aprimoramentos computacionais.

Na tabela 6.6 apresentou-se o tempo de execução de algumas das rotinas baseadas

	problema	*ColFat*	*RowFat*	*Subst.*	
1	*afiro*	2.88E-04	3.01E-04	1.56E-04	1
2	*adlittle*	1.72E-03	1.81E-03	5.06E-04	2
3	*sc205*	4.29E-03	4.90E-03	1.81E-03	3
4	*share1b*	9.60E-03	1.03E-02	1.70E-03	4
5	*scorpion*	1.06E-02	1.16E-02	3.35E-03	5
6	*sctap1*	1.38E-02	1.49E-02	3.61E-03	6
7	*scagr25*	1.15E-02	1.28E-02	4.70E-03	7
8	*israel*	4.60E-01	4.66E-01	1.42E-02	8
9	*brandy*	3.92E-02	4.12E-02	4.39E-03	9
10	*bandm*	3.99E-02	4.22E-02	5.96E-03	10
11	*scfxm1*	4.35E-02	4.60E-02	6.26E-03	11
12	*e226*	3.37E-02	3.55E-02	4.36E-03	12
13	*scrs8*	6.93E-02	7.34E-02	9.15E-03	13
14	*beaconfd*	2.78E-02	2.84E-02	3.70E-03	14
15	*scsd6*	1.90E-02	2.00E-02	3.17E-03	15
16	*ship04s*	2.17E-02	2.18E-02	5.22E-03	16
17	*czprob*	5.53E-02	5.33E-02	1.38E-02	17

Tab 6.4: Tempo (seg) Rotinas Convencionais de Fatoração (PC-486)

	problema	*ColFat*	*RowFat*	*Subst.*	
1	*afiro*	1.58	1.51	1.20	1
2	*adlittle*	2.05	1.94	1.51	2
3	*sc205*	1.77	1.55	1.19	3
4	*share1b*	2.38	2.23	1.65	4
5	*scorpion*	1.97	1.81	1.37	5
6	*sctap1*	2.14	1.98	1.34	6
7	*scagr25*	1.95	1.74	1.17	7
8	*israel*	2.20	2.17	1.60	8
9	*brandy*	2.60	2.47	1.53	9
10	*bandm*	2.41	2.28	1.55	10
11	*scfxm1*	2.44	2.31	1.53	11
12	*e226*	2.50	2.38	1.64	12
13	*scrs8*	2.45	2.32	1.36	13
14	*beaconfd*	2.53	2.47	1.53	14
15	*scsd6*	2.49	2.37	1.56	15
16	*ship04s*	2.29	2.28	1.32	16
17	*czprob*	1.92	1.99	1.15	17

Tab 6.5: (Mflops/seg) Rotinas Convencionais de Fatoração (PC-486)

na utilização de *codificações estruturais*, propostas neste trabalho.

Apresentou-se na tabela 6.7, o desempenho em MegaFlops/segundo das rotinas baseadas em *codificações estruturais.*

Apresenta-se a seguir, a série de resultados obtidos numa arquitetura IBM-RS/6000 modelo 370.

Na tabela 6.8 apresentou-se os tempos de execução em segundos, de rotinas *convencionais* de fatoração, baseadas na geração de fatores por *colunas* e por *linhas*, respectivamente. Adicionalmente, a título de comparação, apresentou-se o tempo gasto na fase de substituições.

	problema	*BinFat*	*AdrFat*	
1	*afiro*	3.54E-04	3.46E-04	1
2	*adlittle*	1.86E-03	1.87E-03	2
3	*sc*205	5.22E-03	5.50E-03	3
4	*share*1*b*	8.17E-03	8.61E-03	4
5	*scorpion*	1.26E-02	1.29E-02	5
6	*sctap*1	1.56E-02	1.73E-02	6
7	*scagr*25	1.55E-02	1.65E-02	7
8	*israel*	1.65E-01	2.01E-01	8
9	*brandy*	2.99E-02	3.19E-02	9
10	*bandm*	4.18E-02	4.28E-02	10
11	*scfxm*1	4.31E-02	4.50E-02	11
12	*e*226	3.07E-02	3.21E-02	12
13	*scrs*8	6.44E-02	6.36E-02	13
14	*beaconfd*	2.78E-02	2.84E-02	14
15	*scsd*6	2.03E-02	2.27E-02	15
16	*ship*04*s*	2.00E-02	1.98E-02	16
17	*czprob*	4.83E-02	4.05E-02	17

Tab 6.6: Tempo (seg) Rotinas Estruturais de Fatoração (PC-486)

Na tabela 6.9 apresentou-se o desempenho computacional das rotinas convencionais numa arquitetura IBM-RS/6000, ilustrando-se a taxa de MegaFlops/segundo alcançada em cada caso.

Na tabela 6.10 ilustrou-se o tempo de execução de algumas das rotinas baseadas em *codificações estruturais* propostas neste trabalho.

Na tabela 6.11 apresentou-se o desempenho computacional das rotinas baseadas em *codificações estruturais*, mensurando-se a taxa de MegaFlops/segundo obtida em cada caso.

Finalmente, na tabela 6.12 ilustrou-se o dispêndio típico de recursos adicionais, em termos de posições de memória, para algumas das estruturas de dados amplamente utilizadas nos métodos propostos no presente trabalho. Entre as grandezas ilustradas, incluiu-se a título de comparação, a dimensão e o número de fatores não nulos de cada problema, bem como o tamanho das *listas de códigos* e de *endereços*, para cada um dos tipos de codificação considerados. Adicionalmente, ilustrou-se também o número total de famílias nos quais se conseguiu particionar a estrutura de contribuições de cada problema.

	problema	*BinFat*	*AdrFat*	
1	*afiro*	1.28	1.31	1
2	*adlittle*	1.88	1.88	2
3	*sc*205	1.46	1.38	3
4	*share*1*b*	2.80	2.66	4
5	*scorpion*	1.66	1.62	5
6	*sctap*1	1.89	1.71	6
7	*scagr*25	1.44	1.35	7
8	*israel*	6.14	5.03	8
9	*brandy*	3.40	3.19	9
10	*bandm*	2.30	2.25	10
11	*scfxm*1	2.46	2.36	11
12	*e*226	2.75	2.63	12
13	*scrs*8	2.64	2.67	13
14	*beaconfd*	3.65	3.30	14
15	*scsd*6	2.34	2.09	15
16	*ship*04*s*	2.48	2.50	16
17	*czprob*	2.20	2.62	17

Tab 6.7: (Mflops/seg) Rotinas Estruturais de Fatoração (PC-486)

	problema	*ColFat*	*RowFat*	*Subst.*	
1	*afiro*	5.40E-05	5.65E-05	3.70E-05	1
2	*adlittle*	3.06E-04	3.28E-04	1.14E-04	2
3	*sc*205	7.90E-04	8.95E-04	3.45E-04	3
4	*share*1*b*	1.66E-03	1.76E-03	3.20E-04	4
5	*scorpion*	1.88E-03	2.06E-03	6.8E-04	5
6	*sctap*1	2.46E-03	2.62E-03	7.00E-04	6
7	*scagr*25	1.15E-03	1.28E-03	4.70E-04	7
8	*israel*	7.66E-02	7.92E-02	2.3E-03	8
9	*brandy*	7.04E-03	7.16E-03	8.60E-04	9
10	*bandm*	7.72E-03	7.78E-03	1.20E-03	10
11	*scfxm*1	7.62E-03	7.90E-03	1.36E-03	11
12	*e*226	6.08E-03	6.22E-03	9.40E-04	12
13	*scrs*8	1.36E-02	1.28E-02	1.94E-03	13
14	*beaconfd*	4.98E-03	4.84E-03	7.40E-04	14
15	*scsd*6	3.30E-03	3.32E-03	6.20E-04	15
16	*ship*04*s*	3.87E-03	3.93E-03	9.67E-04	16
17	*czprob*	1.23E-02	1.15E-02	2.76E-03	17

Tab 6.8: Tempo (seg) Rotinas Convencionais de Fatoração (RS-6000)

	problema	*ColFat*	*RowFat*	*Subst.*	
1	*afiro*	8.43	8.05	5.05	1
2	*adlittle*	11.49	10.72	6.72	2
3	*sc205*	9.64	8.51	6.27	3
4	*share1b*	13.78	13.00	8.77	4
5	*scorpion*	11.16	10.18	6.74	5
6	*sctap1*	12.00	11.27	6.90	6
7	*scagr25*	19.41	17.44	11.70	7
8	*israel*	13.21	12.78	9.87	8
9	*brandy*	14.47	14.23	7.82	9
10	*bandm*	12.45	12.36	7.68	10
11	*scfxm1*	13.93	13.94	7.03	11
12	*e226*	13.87	13.56	7.59	12
13	*scrs8*	12.48	13.32	6.41	13
14	*beaconfd*	14.11	14.52	7.63	14
15	*scsd6*	14.38	14.29	7.97	15
16	*ship04s*	12.84	12.62	7.14	16
17	*czprob*	8.64	9.21	5.74	17

Tab 6.9: (Mflops/seg) Rotinas Convencionais de Fatoração (RS-6000)

	problema	*BinFat*	*AdrFat*	
1	*afiro*	1.06E-04	9.65E-05	1
2	*adlittle*	4.62E-04	4.44E-04	2
3	*sc205*	1.31E-03	1.32E-03	3
4	*share1b*	2.02E-03	1.90E-03	4
5	*scorpion*	3.10E-03	3.02E-03	5
6	*sctap1*	3.60E-03	3.56E-03	6
7	*scagr25*	1.55E-03	1.65E-03	7
8	*israel*	3.00E-02	2.95E-02	8
9	*brandy*	6.70E-03	6.50E-03	9
10	*bandm*	8.76E-03	8.46E-03	10
11	*scfxm1*	9.08E-03	8.78E-03	11
12	*e226*	6.58E-03	6.46E-03	12
13	*scrs8*	1.24E-02	1.18E-02	13
14	*beaconfd*	4.62E-03	5.42E-03	14
15	*scsd6*	4.42E-03	4.50E-03	15
16	*ship04s*	4.93E-03	4.30E-03	16
17	*czprob*	1.16E-02	9.46E-03	17

Tab 6.10: Tempo (seg) Rotinas Estruturais de Fatoração (RS-6000)

	problema	*BinFat*	*AdrFat*	
1	*afiro*	4.29	4.72	1
2	*adlittle*	7.61	7.92	2
3	*sc*205	5.81	5.77	3
4	*share*1*b*	11.33	12.04	4
5	*scorpion*	6.77	6.95	5
6	*sctap*1	8.20	8.29	6
7	*scagr*25	14.40	13.53	7
8	*israel*	33.73	34.30	8
9	*brandy*	15.20	15.67	9
10	*bandm*	10.98	11.36	10
11	*scfxm*1	11.69	12.09	11
12	*e*226	12.82	13.06	12
13	*scrs*8	13.71	14.43	13
14	*beaconfd*	15.21	12.97	14
15	*scsd*6	10.74	10.55	15
16	*ship*04*s*	10.07	11.55	16
17	*czprob*	9.15	11.22	17

Tab 6.11: (Mflops/seg) Rotinas Estruturais de Fatoração (RS-6000)

	problema	n	$\eta_{(U)}$	n_{bin}	n_{adr}	n_{fam}	n_{env}	n_{cpt}	
1	*afiro*	27	107	86	34	55	123	48	1
2	*adlittle*	56	404	511	285	210	1343	151	2
3	*sc*205	205	1182	1375	814	602	3681	512	3
4	*share*1*b*	96	1026	2925	1936	555	7767	490	4
5	*scorpion*	388	2324	3408	2173	907	6371	1071	5
6	*sctap*1	300	2667	4922	3226	1297	16463	1075	6
7	*scagr*25	471	2948	3929	2528	1220	8964	1333	7
8	*israel*	174	11259	78777	70633	1078	90116	3258	8
9	*brandy*	220	2850	11289	8914	1251	31990	1303	9
10	*bandm*	305	4114	12997	9586	2061	38857	2050	10
11	*scfxm*1	330	4963	13758	10339	2047	39279	2032	11
12	*e*226	223	3416	10724	7691	1616	34780	1450	12
13	*scrs*8	490	5134	22106	15522	3319	84494	2863	13
14	*beaconfd*	173	1727	8147	6296	1680	31223	1005	14
15	*scsd*6	147	2545	5610	4652	1220	17393	824	15
16	*ship*04*s*	402	3134	7362	3266	2782	34811	1482	16
17	*czprob*	929	6655	19036	7650	7181	92344	3423	17

Tab 6.12: Dispêndio (Memória) Estruturas de Rotinas de Fatoração

Capítulo 7

Conclusões

Como resultado da presente pesquisa, conclui-se que as novas metodologias apresentadas, possuem um potencial até então não explorado pelos demais métodos da literatura.

Como principal ponto a se destacar, nota-se que o presente trabalho mostrou ser técnicamente viável a geração de códigos *dedicados* à cada estrutura matricial sendo processada, num estilo similar ao dos métodos originalmente propostos por Gustavson [60].

Conseguiu-se mostrar em algumas das variantes propostas, que é possível a utilização de listas de códigos com espaço proporcional apenas aos dados de entrada, e não ao número de operações a serem efetuadas ao longo de todo o processo de eliminação. Tal ponto, se constituiu por mais de duas décadas, no principal entrave a ampla utilização de abordagens simbólicas *completamente livres de loops.*

Dentre todas as demais abordagens consideradas até os dias atuais, a técnica de *codificações estruturais*, é a que consegue mais se aproximar dos desempenhos computacionais obtidos com implementações *completamente livres de loops* (consideradas em maior detalhe pelo autor em [6]). O uso de técnicas de codificação mais elaboradas como as apresentadas na seção 5.4 (explorando-se técnicas de compactação), possui no entanto, como ponto favorável em comparação com abordagens *livres de loops*, uma utilização de recursos bem mais parcimoniosa, podendo se manter em muitos dos casos, proporcional apenas ao número de fatores não nulos resultantes (caso se lançe mão de implementações baseadas em *listas recorrentes*, como as propostas nas seções 5.3 e 5.4).

O que a compactação de informações traz como benefício secundário é justamente o fato de se poder aumentar a taxa de operações de ponto flutuante em comparação com os *overheads* decorrentes das demais operações, como as de indexação indireta, ao se elevar o número de operações de ponto flutuante passíveis de serem implícitamente representadas por cada código.

Em face a sua densidade elevada, o problema Israel ilustra que o potencial de execução em Mflops de ambas as arquiteturas comparadas, encontra-se quase duas vezes acima da média obtida na resolução dos demais problemas. Tal ocorre por nos demais casos, justamente não estar se explorando tão plenamente os recursos dedicados ao processamento de ponto flutuante, como no caso do problema Israel.

As taxas em Mflops obtidas na fase de substuições (situadas normalmente na metade do desempenho obtido na fase de fatoração) ilustra novamente o fato de que quanto menor o volume de operações simultâneas de ponto flutuante passíveis de serem despachadas a cada etapa, menor o desempenho computacional obtido. Tal ocorre, pelo fato de na fase de substituições, o volume de operações de ponto flutuante ser proporcional apenas ao número de fatores não nulos, situando-se pois sempre aquem do volume efetuado na fase de fatoração.

Pode-se esperar, que com a aplicação de técnicas de codificações estruturais para a fase de substituições, parte da ineficiência computacional venha a ser suprimida, caso se consiga alcançar um nível aceitável de compactação das informações.

Em casos onde o número de elementos não nulos de cada linha da matriz de fatores situa-se numa faixa inferior a 10 elementos (como típicamente ocorre na área de potência), a técnica de codificação *binária*, isoladamente já garante um espaço adicional de trabalho da ordem apenas do número de fatores não nulos resultantes.

Nos casos onde a presença *supernodal* faz-se notar maciçamente, o grau de compactação obtido com o uso de *códigos supernodais*, permite uma vez mais que o espaço adicional de trabalho seja limitado apenas a estrutura de *supernodes* resultantes, viabilizando compactações ainda maiores do que no caso anterior.

Os casos intermediários, podem ser cobertos por uma combinação selecionada de códigos *binários*, por *envelopes*, ou *supernodais* (de forma plena ou *parcial*, em versões por linhas ou colunas), além da possível extensão a códigos mais elaborados, como os *multiplexados*, incluindo-se neste caso os códigos *recorrentes* e os por *varredura múltipla* de linhas e/ou colunas.

Entre as vantagens dos métodos apresentados encontra-se o fato de permitir uma melhor exploração de recursos de armazenamento secundário *rápido*, como memórias do tipo *cache* (presentes nas mais modernas estações de trabalho), num nível até então não considerado, como no caso da contribuição de poucas linhas isoladas.

A metodologia apresentada possui no entanto um caráter geral, e permite que tanto o caso de pequenas contribuições, como o de contribuições da forma *supernodal*, possam ser tratados de uma forma unificada, tomando-se como critério para uma melhor exploração de recursos secundários de armazenamento, um particionamento da representação por *supernodes*, de modo a se ajustar ao tamanho da *cache* físicamente disponível nas arquiteturas a serem utilizadas.

Além do melhor aproveitamento da *cache*, os métodos baseados no conceito de *codificações estruturais*, viabilizam procedimentos com um menor *overhead* de indexação indireta e de acessos temporários à memória, por permitir a operação simultânea sobre um grupo de linhas contribuintes, acessando-se diretamente a posição correspondente na linha base sendo gerada a cada etapa.

Desta forma, pode-se dispensar estutruras auxiliares como a de vetores de trabalho expandidos, lançando-se mão de listas *restritas* de endereços, entre outros recursos propostos para o acesso direto aos elementos sobre cada linha base a ser operada.

A utilização de métodos do tipo Crout, traz consigo um benefício numérico

adicional, e que é o de permitir a acumulação dos produtos escalares em precisão mais elevada do que a adotada para o armazenamento final, dos fatores resultantes gerados.

A plena potencialidade dos métodos de natureza *simbólica* (baseados em *codificações estruturais*) propostos nestes trabalho, só pode ser atingida em implementações mais elaboradas em linguagens como C/C^{++} ou ASSEMBLER. Assim, dos resultados computacionais obtidos (utilizando-se apenas procedimentos programados em FORTRAN), não se poderia esperar um resultado amplamente favorável numa maioria absoluta de casos.

Mesmo assim, em alguns casos (como nos problemas Israel, Czprob, Beaconfd, Brandy e Scrs8) a potencialidade das novas técnicas conseguiu se sobressair, obstante todos os *overheads* adicionais, oriundos da codificação em FORTRAN resultante, o que mostra que quando eficientemente programadas em linguagens mais apropriadas, seguramente poderão apresentar um desempenho ainda melhor do que o já obtido.

Como resultado secundário desta pesquisa, percebe-se que alguns recursos de programação como o de desvios condicionais computados, não foram projetados prevendo-se sua utilização na forma como foram estensivamente adotados neste trabalho. Assim, nota-se que parte dos *overheads* das implementações dos novos métodos em FORTRAN, decorre da ineficiência a que os compiladores são inevitavelmente forçados, na presença de GOTO's *Computados* (por não se ter como prever a priori, que apenas os casos previstos e enumerados na lista de desvios, venham a ser utilizados).

Cabe notar que os códigos atualmente adotados pela literatura, escritos na sua quase totalidade em FORTRAN, não se utilizam de recursos que poderiam vir a ser melhor implementados em linguagens de mais baixo nível (como acessos diretos a posições de memória, mediante listas de apontadores por exemplo). Desta forma, não se pode esperar um ganho significativo, com a tradução de tais códigos para outras linguagens, tendo em vista que o código gerado pelos compiladores FORTRAN disponíveis atualmente, encontra-se muito próximo do *"ótimo"* possível para cada arquitetura, nestes casos.

Como conclusão do presente trabalho, nota-se que a incorporação de algumas das mais avançadas técnicas adotadas para o processamento *supernodal* pela literatura (como a chamada a subprogramas dedicados do tipo BLAS, para operações na forma de produtos matriciais), podem ser trivialmente incorporados nas abordagens baseadas no conceito de *listas de códigos* propostas neste trabalho.

A extensão da metodologia ora apresentada, em sentido reverso, aplicando-a aos métodos convencionalmente adotados atualmente, não se mostra trivial no entanto, em função de não se ter contado com o conceito de *codificações estruturais* como base fundamental no projeto original de tais subprogramas.

Alguns conceitos como o de *janelas de eliminação* (na forma como foram presentemente apresentadas), podem vir a ser trivialmente adotados nos códigos convencionais, permitindo com isso, uma melhor exploração de sub-casos particulares não previstos originalmente nos procedimentos genéricos até então utilizados.

Concomitantemente com a introdução de novas técnicas mediante a utilização

de *listas de códigos*, mostrou-se também ser possível o aprimoramento de algumas das metodologias tradicionais, conseguindo-se incorporar a noção de processamento *supernodal* em métodos baseados na geração de fatores por *colunas*. Tal extensão, permite que tais métodos, venham a se situar competitivamente em relação as demais abordagens, quase sempre baseadas na geração de fatores por linhas (ou por submatrizes). Entre todas as abordagens convencionais para o processamento esparso, a geração por *colunas* é a que possui o menor nível de *overheads* intrínsecos, e a incorporação da exploração de características *supernodais* em tais métodos, permite elevar ainda mais a sua eficiência, a um custo extremamente baixo em termos de espaço adicional de trabalho e de armazenamento.

Direções Futuras de Pesquisa

Como possíveis direções futuras à extensão do presente trabalho, pode-se apontar as seguintes áreas:

- Codificação das metodologias propostas, em linguagens como C/C^{++} explorando plenamente recursos como o de listas de apontadores para posições de mémoria (visando sua comparação com os códigos atualmente em FORTRAN). Adicionalmente pode-se considerar a geração de código otimizado em ASSEMBLER, tomando-se por base, o código gerado pelos compiladores C/C^{++} ou FORTRAN utilizados, removendo-se ineficiências intrínsecas ao processamento de desvios condicionais computados, e explorando-se plenamente o conjunto de registradores de ponto flutuante disponíveis (o que normalmente não consegue ser feito de forma eficiente, pela maioria dos compiladores FORTRAN atuais).

- Implementação de métodos de eliminação *mista* (operando sobre matrizes com elementos reais e complexos), mediante a noção de códigos especialmente dedicados a cada um dos possíveis casos de operações de ponto flutuante a serem encontrados durante o processo de fatoração.

- Extensão da técnica de processamento mediante listas de códigos, para as fases de *substituições forward*, *diagonal* e *backward*.

- Implementação de métodos pré-condicionadores baseados em fatorações *incompletas* para a solução *iterativa* de sistemas lineares, adotando-se a técnica de codificações *estruturais*, tanto para a geração da matriz pré-condicionadora, como para o cálculo dos produtos *esparsos*, na forma *matriz* $\times$ *vetor* (requeridos a cada etapa do algoritmo iterativo central).

- Extensão das metodologias apresentadas, à solução paralela de sistemas lineares, em métodos como os do tipo *Fan-In Distribuído*, adotando-se o conceito de *codificações estruturais*, de modo a permitir uma melhor implementação da fase de geração das contribuições resultantes a serem intercambiadas entre os processadores. (Esta é uma operação que atualmente não é efetuada de forma eficiente ou compacta, por não se levar em conta, informações adicionais como a de *códigos estruturais*, que permitam a montagem de mensagens inteiramente dedicadas a cada matriz sendo tratada).

- Extensão das presentes metodologias ao caso assimétrico, aplicando-as à métodos onde pivoteamentos estáticos (de natureza estrutural) possam ser empregados com sucesso (sem requerer a adoção de técnicas de pivoteamento dinâmico visando a estabilização numérica).

- Aplicação da técnica de *codificações estruturais* em métodos de *refatoração parcial* (explorando *sparse vectors*), mediante a geração de códigos dedicados ao processamento apenas dos elementos alterados em cada uma das linhas base envolvidas.

- Incorporação de algumas das técnicas como as de *multiplexação* e de processamento mediante *listas de códigos* (com desvios para trechos dedicados apropriados), à outras fases do processamento esparso, como a de ordenamento ou fatoração simbólica, possivelmente viabilizando uma melhor implementação de operações como a de atualização do grau dos nós remanescentes, ou a concatenação da estrutura de contribuições resultantes a cada etapa.

APÊNDICES

Apêndice A

Estruturas de Dados para Fatorações Esparsas

Neste apêndice enumeram-se as estruturas de dados adotadas para a representação de matrizes esparsas, envolvidas na solução por métodos diretos, de sistemas lineares irredutíveis

$$A\,x = b$$

com:

A (*esparsa*) simétrica, definida positiva
x, b (*densos*)

mediante fatorações da matriz original na forma:

$$A = U^T D\,U$$

com:

U^T (*esparsa*) triangular inferior, com diagonal unitária
D (*diagonal*) com elementos diagonais estritamente positivos
U (*esparsa*) triangular superior, com diagonal unitária

É assumido que apenas representações esparsas para a matriz de fatores U e densas para a diagonal de D, sejam utilizadas para fins de armazenamento, com os elementos da matriz original A, mapeados diretamente sobre as posições correspondentes nas estruturas das matrizes de fatores U e D resultantes.

Assume-se também que a estrutura de fatores (como descrito nas seções 3.2 e 3.3) tenha sido determinada após a conclusão de uma fase de reordenamento de equações (visando a redução de *fill-in's*), ou como resultado da aplicação de uma fase de fatoração simbólica.

Os valores numéricos dos coeficientes originais de A, não mais estarão disponíveis ao final do processo de eliminações, sendo reescritos pelos valores associados da matriz de fatores U, e pelo inverso dos elementos diagonais de D.

As representações esparsas (simétricas) para a matriz original A e de fatores U estarão sempre baseadas num esquema de armazenamento seqüencial "segmentado"

por linhas (como apresentado na seção 3.1), excluindo-se desta representação os elementos diagonais (unitários) de U.

Apresenta-se a seguir, uma descrição resumida das estruturas de dados adotadas ao longo de cada uma das seções deste trabalho, indicando-se sempre que disponível, uma cota superior para o espaço de armazenamento dispendido, em função de grandezas associadas como a dimensão do problema, número de seus elementos não nulos originais ou de fatores triangulares, entre outros parâmetros característicos.

Juntamente com o nome de cada um dos vetores associados às estruturas, fornece-se entre colchetes, uma estimativa para a cota máxima do número de seus elementos, bem como entre parênteses, o tipo mais adequado para sua representação numérica (seguindo-se uma nomenclatura similar a adotada na linguagem C).

Como regra geral (exceto nos casos explicitamente mencionados em contrário), procurou-se adotar uma nomenclatura obedecendo a seguinte convenção:

- Vetores de apontadores de dimensão n (ou $n+1$), para posições Iniciais na representação de estruturas por linhas (como **IU, IUT, IUP**), são sempre precedidos pela inicial **I** em seu nome.
- Vetores de apontadores de dimensão n para posições Finais na representação de estruturas por linhas (como **IUF, IUTF, IUPF**), são sempre sucedidos pela terminação **F** em seu nome.
- Vetores indicadores do número de elementos não nulos por cada linha (como **NA, NU**) são sempre precedidos pela inicial **N** em seu nome.
- Vetores de índices de Colunas dos elementos não nulos por cada linha (como **JA, JU, JUT**), são sempre precedidos pela inicial **J** em seu nome.
- Vetores associados a representação de estruturas de U^T (como **IUT, JUT**) são sempre sucedidos pela terminação **T** em seu nome.
- Vetores de trabalho Expandidos (como **W, IW**) são sempre sucedidos pela terminação **W** em seu nome.
- Listas (simbólicas) de apontadores para posições de memória de elementos sobre a estrutura básica gerada a cada etapa (como **PLIN, PCOL, PJ** e **PJF**) são sempre precedidas pela inicial **P** em seu nome.
- Listas de códigos Binários, por Envelopes, Supernodais ou de Multiplexação (como **LBIN, LENV, LSUP, LMPLX**) são sempre precedidas pela inicial **L** em seu nome.

Seção 3.1:

Porção Triangular Superior da Matriz Original (A)

IA $[n]$ (long ptr) *Apontadores para o Início da representação dos elementos não nulos de cada linha na porção triangular superior da matriz original A.*

IAF $[n]$ (long ptr) *Apontadores para o Final da representação dos elementos não nulos de cada linha na porção triangular superior da matriz original A.*

NA $[n]$ (unsigned short int) *Número de elementos não nulos da representação de cada linha na porção triangular superior da matriz original A.*

JA $[\eta_{\text{sup}}(A)]$ (unsigned int) *Representação das Colunas dos elementos não nulos de cada linha na porção triangular superior da matriz original A.*

AN $[\eta_{\text{sup}}(A)]$ (float) *Valores Numéricos dos elementos não nulos de cada linha na porção triangular superior da matriz original A.*

Diagonal da Matriz Original (A)

DN $[n]$ (float) *Valores Numéricos dos elementos Diagonais da matriz original A.*

Matriz Triangular Superior de Fatores (U)

IU $[n]$ (long ptr) *Apontadores para o Início da representação dos elementos não nulos de cada linha da matriz de fatores U.*

IUF $[n]$ (long ptr) *Apontadores para o Final da representação dos elementos não nulos de cada linha da matriz de fatores U.*

NU $[n]$ (unsigned short int) *Número de elementos não nulos da representação de cada linha da matriz de fatores U.*

JU $[\eta(U)]$ (unsigned int) *Representação das Colunas dos elementos não nulos de cada linha da matriz de fatores U.*

UN $[\eta(U)]$ (float ou double float) *Valores Numéricos dos elementos não nulos de cada linha da matriz de fatores U (ao final do processo de eliminações). Originalmente contendo os valores numéricos dos elementos da matriz original A (mapeados nas posições correspondentes de U), e os elementos de fill-in inicialmente zerados.*

Matriz Diagonal de Fatores (D)

DI $[n]$ (float ou double float) *Inverso dos valores numéricos dos elementos Diagonais da matriz de fatores D (resultantes ao final do processo de eliminações). Originalmente contendo os valores numéricos dos elementos diagonais da matriz original A.*

Estrutura Transposta da Matriz Triangular de Fatores ($\mathbf{U^T}$)

IUT $[n]$ (long ptr) *Apontadores para o Início da representação (Transposta) dos elementos não nulos de cada linha da matriz de fatores U^T.*

IUTF $[n]$ (long ptr) *Apontadores para o Final da representação (Transposta) dos elementos não nulos de cada linha da matriz de fatores U^T.*

NUT $[n]$ (unsigned short int) *Número de elementos não nulos da representação (Transposta) de cada linha da matriz de fatores* U^T.

JUT $[\eta(U^T)]$ (unsigned int) *Representação (Transposta) das Colunas dos elementos não nulos de cada linha da matriz de fatores* U^T *(indicando as linhas contribuintes a serem subtraídas da linha base corrente a cada etapa do processo de eliminações).*

Seção 3.2:

IORD $[n]$ (unsigned int) *Vetor contendo a Ordem de permutação simétrica de linhas e colunas, visando a redução do número de fill-in's da matriz de fatores* U.

Seção 3.3:

$C_*^{(i)}$ $[\ \leq \eta^{(i)}(U^T)\ ]$ (unsigned int) *Conjunto de linhas Características da matriz de fatores, geradoras do espectro de colunas de todos os elementos não nulos contribuintes sobre a linha básica* i. *(Estes conjuntos satisfazem a propriedade:* $\sum_{i=1}^{n} \mid C_*^{(i)} \mid \ \leq \ n$, *permitindo serem implementados em uma única estrutura de dimensão* n, *mediante recursos de encadeamento)*

Seção 3.4:

Acumulo de Contribuições sobre a Diagonal

piv (double float) *Variável em precisão dupla, a conter temporariamente o valor do elemento diagonal corrente, e a acumular as contribuições por ele sofridas ao longo de cada etapa (armazenando-se posteriormente o seu inverso de volta à estrutura* **DI[.]** *associada)*

Vetor de Trabalho Expandido

W $[n]$ (double float) *Vetor de trabalho "expandido" em precisão dupla, contendo temporariamente os valores numéricos dos elementos não nulos originais da linha base, descompactados segundo o índice de suas colunas associadas. Utilizado para a acumulação das contribuições na forma descompactada, das linhas antecessoras contribuintes sobre a linha básica, armazenando ao final de cada etapa (nos processos de geração por linhas), os valores numéricos dos fatores associados, a serem compactados novamente na representação da linha correntemente gerada*

Vetor de Contribuições

W $[n]$ (double float) *Vetor de acumulo de contribuições na forma descompactada, das linhas antecessoras contribuintes sobre a linha básica, armazenando ao final de cada etapa (nos processos de geração por colunas), os valores numéricos dos fatores associados, a serem paulatinamente incorporados à representação da coluna sendo correntemente gerada. Assumido como originalmente zerado antes do início do processo de eliminação*

Apontadores Dinâmicos delimitadores de Posições de acesso

IUP [n] (long ptr) *Apontadores (dinâmicos) para a posição Inicial na representação dos elementos não nulos de cada linha da matriz de fatores U, a partir da qual se efetuarão as contribuições sobre a linha básica corrente. (Originalmente contendo valores idênticos aos de* **IU**, *sendo posteriormente incrementados para cada uma das linhas acessadas nas sucessivas etapas do processo de eliminações)*

IUPF [n] (long ptr) *Apontadores (dinâmicos) para a posição Final na representação dos elementos não nulos de cada linha da matriz de fatores U, até a qual se efetuarão as contribuições sobre a linha básica corrente. (Originalmente contendo valores idênticos aos de* **IU**, *sendo posteriormente incrementados para cada uma das linhas acessadas nas sucessivas etapas do processo de eliminações)*

Seção 4.1:

Listas Simbólicas de Endereços

PLIN [$*$] (long ptr) *Lista "simbólica" com uma seqüência de apontadores para as Posições de armazenamento dos elementos não nulos da Linha Base corrente, a serem acessados durante o recebimento de contribuições oriundas de linhas antecessoras (a cada etapa do processo de eliminação)*

PCOL [$*$] (long ptr) *Lista "simbólica" com uma seqüência de apontadores para as Posições de armazenamento dos elementos não nulos da Coluna Base corrente, a serem acessados durante o recebimento de contribuições oriundas de linhas antecessoras (a cada etapa do processo de eliminação)*

Vetor de Apontadores Expandido

IW [n] (long ptr) *Vetor de apontadores expandido, contendo temporariamente a cada etapa, as Posições de armazenamento dos fatores não nulos da linha base, descompactadas segundo o índice de suas colunas associadas*

Seção 4.2:

Estrutura de Sucessores na Árvore de Eliminação

PARENT [n] (unsigned int) *Estrutura de Sucessores na árvore de eliminação da matriz de fatores resultantes U. (Indicando a primeira de cada uma das linhas a dependerem préviamente da eliminação da linha básica associada em alguma etapa anterior do processo, de modo a se poder dar continuidade ao processo de eliminações)*

Seção 4.3:

Estrutura de Janelas de Eliminação

nwind (unsigned byte) *Número de "Janelas" (constituídas por porções de linhas básicas contíguas e de característica estrutural similar), adotadas na sub-divisão do processo de eliminação triangular*

WINTYP [*nwind*] (unsigned byte) *Vetor especificando o Tipo de processamento a ser adotado para cada uma das Janelas de Eliminação associadas*

WINROW [*nwind*] (unsigned int) *Vetor contendo a linha Base Inicial (de cada Janela)*

WINROWF [*nwind*] (unsigned int) *Vetor contendo a linha Base Final (de cada Janela)*

CONTRB [*nwind*] (unsigned int) *Vetor contendo a linha Contribuinte Inicial (a ser considerada no processamento de cada Janela)*

CONTRBF [*nwind*] (unsigned int) *Vetor contendo a linha Contribuinte Final (a ser considerada no processamento de cada Janela)*

Seção 5.1:

Listas Simbólicas de Códigos

LBIN [*] (unsigned byte) *Lista simbólica de Códigos Binários associados ao padrão de elementos não nulos de cada coluna de contribuições sobre a linha base da matriz de fatores U, a serem posteriormente decodificados, fornecendo as seqüências de operações de acumulo por produto escalar (para fases numéricas projetadas, de modo a explorar tal informação)*

Seção 5.2:

Listas Simbólicas de Códigos

LENV [*] (unsigned byte) *Lista simbólica de Códigos espelhando padrões de contribuição por colunas na forma por Envelopes*

LSUP [*] (unsigned byte) *Lista simbólica de Códigos espelhando padrões de contribuição (comuns a todo um grupo de colunas contribuintes) de forma Supernodal sobre a linha base*

Listas Restritas de Endereços

PJ [*] (unsigned byte) *Lista simbólica Restrita, com o endereço Inicial de um grupo de posições contíguas sobre a linha base, a serem consecutivamente acessadas e operadas, numa mesma etapa do processo de contribuições*

PJF [*] (unsigned byte) *Lista simbólica Restrita, com o endereço Final de um grupo de posições contíguas sobre a linha base, a serem consecutivamente acessadas e operadas, numa mesma etapa do processo de contribuições*

Seção 5.3:

Listas de Multiplexação de Desvios

LMPLX [$*$] (unsigned byte) *Lista simbólica de Códigos de Multiplexação, possibilitando uma translação para os trechos definitivos de programa a serem executados, a partir de um remapeamento encadeado de desvios condicionais computados*

LEXCL [$*$] (unsigned byte) *Lista simbólica de Padrões de Exclusão, utilizados em processos de remapeamento de desvios, de modo a se tratar cada um dos possíveis sub-casos de padrões de contribuição a serem excluídos no reprocessamento de um dado código*

Seção 5.4:

Múltiplos Vetores de Trabalho

W$_k$ [n] (unsigned byte) *Vetores de Trabalho Expandidos associados a cada uma das linhas contribuintes a serem processadas simultaneamente por ocasião do acumulo de produtos escalares sobre a linha base (em função dos padrões codificação adotados para cada coluna)*

UW$_k$ [$\eta^{(k)}(U)$] (unsigned byte) *Vetores de Trabalho Reduzidos associados a cada uma das linhas contribuintes a serem processadas simultaneamente por ocasião do acumulo de produtos escalares sobre a linha base (em função dos padrões codificação adotados para cada coluna)*

Finalmente, apresenta-se nas tabelas a seguir, um resumo das principais estruturas de dados consideradas ao longo do trabalho, indicando-se uma esquematização típica do padrão de elementos armazenados.

Apontadores estáticos

IU	ptr^1	...	ptr^n
IUF	ptr_f^1	...	ptr_f^n
IUT	ptr_t^1	...	ptr_t^n
IUTF	$ptr_{t_f}^1$	...	$ptr_{t_f}^n$

Apontadores dinâmicos

IUP	ptr_p^1	...	ptr_p^n
IUPF	$ptr_{p_f}^1$	...	$ptr_{p_f}^n$

Vetores auxiliares

IORD	ind_{piv}^1	...	ind_{piv}^n
W	u_{i,j_1}	...	u_{i,j_n}
IW	$pos_{u_{i,j_1}}$	...	$pos_{u_{i,j_n}}$

Representação de U e da estrutura de U^T

JU	$col_{1,jbeg^1}$	...	$col_{1,jend^1}$	...	$col_{n,jbeg^n}$	...	$col_{n,jend^n}$
UN	$u_{col_{1,jbeg^1}}$	...	$u_{col_{1,jend^1}}$	...	$u_{col_{n,jbeg^n}}$	...	$u_{col_{n,jend^n}}$
JUT	$lin_{1,ibeg^1}$	...	$lin_{1,iend^1}$	...	$lin_{n,ibeg^n}$	...	$lin_{n,iend^n}$

Listas Simbólicas de Endereços

PLIN	pos^{lin}_{1,j_1^1}	...	$pos^{lin}_{1,j^1_{nlin^1}}$	...	pos^{lin}_{n,j_1^n}	...	$pos^{lin}_{n,j^n_{nlin^n}}$
PCOL	pos^{col}_{1,j_1^1}	...	$pos^{col}_{1,j^1_{ncol^1}}$	...	$pos^{col}_{n,j^n_{ncol^1}}$	...	$pos^{col}_{n,j^n_{ncol^n}}$

Listas Restritas de Endereços

PJ	$pos^{base}_{1,1}$	...	$pos^{base}_{1,nbase^1}$	...	$pos^{base}_{n,1}$	...	$pos^{base}_{n,nbase^n}$
PJF	$posf^{base}_{1,1}$	...	$pos^{base}_{1,nbase^1}$	...	$posf^{base}_{n,1}$	...	$posf^{base}_{n,nbase^n}$

Listas Simbólicas de Códigos

LBIN	$cod^{bin}_{1,1}$	...	$cod^{bin}_{1,nbin^1}$	...	$cod^{bin}_{n,1}$	...	$cod^{bin}_{n,nbin^n}$
LENV	$cod^{env}_{1,1}$	...	$cod^{env}_{1,nenv^1}$	...	$cod^{env}_{n,1}$	...	$cod^{env}_{n,nenv^n}$
LNOD	$cod^{nod}_{1,1}$	...	$cod^{nod}_{1,nnod^1}$	...	$cod^{nod}_{n,1}$	...	$cod^{nod}_{n,nnod^n}$

Listas de Multiplexação de Desvios

LMPLX	$cod^{mplx}_{1,1}$	...	$cod^{mplx}_{1,nmplx^1}$	...	$cod^{mplx}_{n,1}$	...	$cod^{mplx}_{n,nmplx^n}$
LEXCL	$cod^{excl}_{1,1}$	...	$cod^{excl}_{1,nexcl^1}$	...	$cod^{excl}_{n,1}$	...	$cod^{excl}_{n,nexcl^n}$

Múltiplos Vetores de Trabalho

$\mathbf{W}_k$	u_{k,j^k_1}	...	$u_{k,j^k_{n^k}}$
$\mathbf{UW}_k$	u_{k,j^k_1}	...	$u_{k,j^k_{n^k}}$

Apêndice B

Métodos Diretos de Eliminação

Neste apêndice serão considerados métodos para a solução direta de sistemas lineares da forma:

$$A\,x = b \tag{B.1}$$

com a matriz de coeficientes, real, simétrica e definida positiva, e os vetores solução e lado direito, reais.

Tais métodos se baseiam numa decomposição da matriz do sistema original na forma:

$$U^T D\,U = A \tag{B.2}$$

com a matriz U triangular superior com diagonal unitária e D diagonal com elementos estritamente positivos.

Uma vez obtida a fatoração triangular, o sistema original pode ser solucionado mediante fases de substituições *forward*, *diagonais* e *backward* a partir dos seguintes sub-sistemas:

$$U^T x'' = b \tag{B.3}$$

$$D\,x' = x'' \tag{B.4}$$

$$U\,x = x' \tag{B.5}$$

Tomaremos como exemplo, um sistema linear simétrico, definido positivo de dimensão 4.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \tag{B.6}$$

O que se deseja inicialmente é conduzi-lo a uma forma triangular superior, e para tanto serão efetuadas combinações lineares entre suas linhas, gradativamente

cancelando elementos na porção triangular inferior dos sub-sistemas obtidos a cada etapa.

Com o decorrer das n etapas deste processo, associa-se a cada uma delas, uma linha ou coluna "base", correspondente ao índice da etapa corrente.

Em face a simetria do sistema original, apenas as operações de combinação linear, visando a determinação dos elementos triangulares superiores e diagonais necessitam ser de fato efetuadas.

Nos exemplos deste apêndice, optou-se por detalhar todo o processo de eliminações, convencionando-se representar por $a_{ij}^{(k)^*}$ os elementos que em função da simetria supra mencionada, não necessitam ser efetivamente calculados, por serem idênticos aos valores $a_{ij}^{(k)}$ previamente determinados ou por se determinar em alguma outra etapa do processo.

Cancelamentos por Colunas gerando fatores por Sub-Matrizes

O primeiro processo de eliminação considerado, leva o sistema a forma triangular, mediante o cancelamento sucessivo de elementos abaixo da diagonal na coluna base associada a cada etapa.

O cancelamento dos elementos desta forma, implica a cada etapa, numa atualização de toda a *Sub-Matriz* definida a partir das linhas e colunas subseqüentes ao elemento diagonal básico.

Iniciando-se a primeira etapa do processo, multiplicando-se a primeira equação por (a_{12}/a_{11}) e subtraindo-a da segunda obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3 \\ b_4 \end{pmatrix} \tag{B.7}$$

com

$$a_{22}^{(2)} = a_{22} - (a_{12}/a_{11})\, a_{12} \tag{B.8}$$

$$a_{23}^{(2)} = a_{23} - (a_{12}/a_{11})\, a_{13} \tag{B.9}$$

$$a_{24}^{(2)} = a_{24} - (a_{12}/a_{11})\, a_{14} \tag{B.10}$$

e

$$b_2^{(2)} = b_2 - (a_{12}/a_{11})\, b_1 \tag{B.11}$$

Do mesmo modo, multiplicando-se a primeira equação por (a_{13}/a_{11}) e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{23}^{(2)*} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4 \end{pmatrix} \tag{B.12}$$

com

$$a_{23}^{(2)*} = a_{23} - (a_{13}/a_{11})\, a_{12} \tag{B.13}$$

$$a_{33}^{(2)} = a_{33} - (a_{13}/a_{11})\, a_{13} \tag{B.14}$$

$$a_{34}^{(2)} = a_{34} - (a_{13}/a_{11})\, a_{14} \tag{B.15}$$

e

$$b_3^{(2)} = b_3 - (a_{13}/a_{11})\, b_1 \tag{B.16}$$

Repetindo-se o processo uma vez mais, multiplicando-se a primeira equação por (a_{14}/a_{11}) e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{23}^{(2)*} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{24}^{(2)*} & a_{34}^{(2)*} & a_{44}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \end{pmatrix} \tag{B.17}$$

com

$$a_{24}^{(2)*} = a_{24} - (a_{14}/a_{11})\, a_{12} \tag{B.18}$$

$$a_{34}^{(2)*} = a_{34} - (a_{14}/a_{11})\, a_{13} \tag{B.19}$$

$$a_{44}^{(2)} = a_{44} - (a_{14}/a_{11})\, a_{14} \tag{B.20}$$

e

$$b_4^{(2)} = b_4 - (a_{14}/a_{11})\, b_1 \tag{B.21}$$

Iniciando-se agora a segunda etapa do processo, multiplicando-se a segunda equação por $(a_{23}^{(2)*}/a_{22}^{(2)})$ e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & a_{24}^{(2)*} & a_{34}^{(2)*} & a_{44}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(2)} \end{pmatrix} \tag{B.22}$$

com

$$a_{33}^{(3)} = a_{33}^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.23}$$

$$a_{34}^{(3)} = a_{34}^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.24}$$

e

$$b_{3}^{(3)} = b_{3}^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, b_{2}^{(2)} \tag{B.25}$$

Multiplicando-se agora a segunda equação por $(a_{24}^{(2)^*}/a_{22}^{(2)})$ e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & a_{34}^{(3)^*} & a_{44}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(3)} \end{pmatrix} \tag{B.26}$$

com

$$a_{34}^{(3)^*} = a_{34}^{(2)^*} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.27}$$

$$a_{44}^{(3)} = a_{44}^{(2)} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.28}$$

e

$$b_{4}^{(3)} = b_{4}^{(2)} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, b_{2}^{(2)} \tag{B.29}$$

Finalmente, iniciando-se a terceira etapa do processo de eliminação, multiplicando-se a terceira equação por $(a_{34}^{(3)^*}/a_{33}^{(3)})$ e subtraindo-a da quarta obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{pmatrix} \tag{B.30}$$

com

$$a_{44}^{(4)} = a_{44}^{(3)} - (a_{34}^{(3)^*}/a_{33}^{(3)})\, a_{34}^{(3)} \tag{B.31}$$

e

$$b_{4}^{(4)} = b_{4}^{(3)} - (a_{34}^{(3)^*}/a_{33}^{(3)})\, b_{3}^{(3)} \tag{B.32}$$

Retro-substituições Triangulares

Uma vez transformado o sistema original em um equivalente na forma triangular superior, a fase subseqüente para a obtenção de sua solução, consiste na determinação em ordem reversa, de cada uma das componentes do vetor x.

Procedendo-se a estas retro-substituições triangulares, os valores da solução do sistema poderão ser gradativamente obtidos mediante:

$$x_4 = b_4^{(4)}/a_{44}^{(4)} \tag{B.33}$$

$$x_3 = (b_3^{(3)} - a_{34}^{(3)} x_4)/a_{33}^{(3)} \tag{B.34}$$

$$x_2 = (b_2^{(2)} - a_{23}^{(2)} x_3 - a_{24}^{(2)} x_4)/a_{22}^{(2)} \tag{B.35}$$

$$x_1 = (b_1 - a_{12} x_2 - a_{13} x_3 - a_{14} x_4)/a_{11} \tag{B.36}$$

Cancelamentos por Linhas gerando fatores por Linhas

Outra forma de se solucionar o problema original (B.6), é efetuar-se o processo de eliminação gerando-se gradualmente cada uma das *Linhas* da matriz de fatores, anulando-se para tal, os coeficientes a esquerda da diagonal da linha base a cada etapa.

Na primeira etapa, não se efetuam operações, visto não existirem elementos a esquerda do elemento diagonal a_{11} a serem cancelados.

Iniciando-se a segunda etapa, multiplicando-se a primeira equação por (a_{12}/a_{11}) e subtraindo-a da segunda obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3 \\ b_4 \end{pmatrix} \tag{B.37}$$

com

$$a_{22}^{(2)} = a_{22} - (a_{12}/a_{11})\, a_{12} \tag{B.38}$$

$$a_{23}^{(2)} = a_{23} - (a_{12}/a_{11})\, a_{13} \tag{B.39}$$

$$a_{24}^{(2)} = a_{24} - (a_{12}/a_{11})\, a_{14} \tag{B.40}$$

e

$$b_2^{(2)} = b_2 - (a_{12}/a_{11})\, b_1 \tag{B.41}$$

Iniciando-se agora a terceira etapa, multiplicando-se a primeira equação por (a_{13}/a_{11}) e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{23}^{(2)^*} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4 \end{pmatrix} \tag{B.42}$$

com

$$a_{23}^{(2)^*} = a_{23} - (a_{13}/a_{11})\, a_{12} \tag{B.43}$$

$$a_{33}^{(2)} = a_{33} - (a_{13}/a_{11})\, a_{13} \tag{B.44}$$

$$a_{34}^{(2)} = a_{34} - (a_{13}/a_{11})\, a_{14} \tag{B.45}$$

e

$$b_3^{(2)} = b_3 - (a_{13}/a_{11})\, b_1 \tag{B.46}$$

Multiplicando-se agora a segunda equação por $(a_{23}^{(2)^*}/a_{22}^{(2)})$ e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4 \end{pmatrix} \tag{B.47}$$

com

$$a_{33}^{(3)} = a_{33}^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.48}$$

$$a_{34}^{(3)} = a_{34}^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.49}$$

e

$$b_3^{(3)} = b_3^{(2)} - (a_{23}^{(2)^*}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.50}$$

Iniciando-se a quarta etapa, multiplicando-se a primeira equação por (a_{14}/a_{11}) e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & a_{24}^{(2)^*} & a_{34}^{(2)^*} & a_{44}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(2)} \end{pmatrix} \tag{B.51}$$

com

$$a_{24}^{(2)^*} = a_{24} - (a_{14}/a_{11})\, a_{12} \tag{B.52}$$

$$a_{34}^{(2)^*} = a_{34} - (a_{14}/a_{11})\, a_{13} \tag{B.53}$$

$$a_{44}^{(2)} = a_{44} - (a_{14}/a_{11})\, a_{14} \tag{B.54}$$

e

$$b_4^{(2)} = b_4 - (a_{14}/a_{11})\, b_1 \tag{B.55}$$

Multiplicando-se agora a segunda equação por $(a_{24}^{(2)^*}/a_{22}^{(2)})$ e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & a_{34}^{(3)^*} & a_{44}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(3)} \end{pmatrix} \tag{B.56}$$

com

$$a_{34}^{(3)^*} = a_{34}^{(2)^*} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.57}$$

$$a_{44}^{(3)} = a_{44}^{(2)} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.58}$$

e

$$b_4^{(3)} = b_4^{(2)} - (a_{24}^{(2)^*}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.59}$$

Finalmente, multiplicando-se a terceira equação por $(a_{34}^{(3)^*}/a_{33}^{(3)})$ e subtraindo-a da quarta obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{pmatrix} \tag{B.60}$$

com

$$a_{44}^{(4)} = a_{44}^{(3)} - (a_{34}^{(3)^*}/a_{33}^{(3)})\, a_{34}^{(3)} \tag{B.61}$$

e

$$b_4^{(4)} = b_4^{(3)} - (a_{34}^{(3)^*}/a_{33}^{(3)})\, b_3^{(3)} \tag{B.62}$$

O que se pode perceber é que ambos os processos de eliminação exemplificados, conduzem à mesma matriz de fatores triangulares, diferindo apenas na ordem de geração e acesso a seus elementos.

O processo de retro-substituições para este segundo caso é portanto idêntico ao anteriormente apresentado, não sendo novamente exemplificado.

Cabe notar também, que o número de operações aritméticas efetuadas em ambos os processos de eliminação é exatamente o mesmo, visto diferirem apenas ao nível da seqüência com que as operações de combinação linear visando os cancelamentos, foram sendo gradativamente efetuadas.

Cancelamentos por Linhas gerando fatores por Colunas

Apresentaremos agora uma última variante do processo de eliminação para o caso simétrico, com a geração dos fatores triangulares superiores por *Colunas*, mediante o cancelamento dos elementos à esquerda da diagonal na linha básica de cada etapa.

Neste caso a geração por *Colunas* pode ser processada em face a simetria do problema original, pois ao invés de se desconsiderar o cálculo redundante dos elementos da porção triangular inferior (como nos processos anteriores), a cada etapa o que se efetua é justamente o cálculo dos fatores situados na porção inferior (na linha base), lançando-os nas posições de seus simétricos superiores na coluna associada.

O que se poderá perceber é que neste processo, a atualização dos fatores da porção triangular superior é na verdade "retardada" até que se efetue efetivamente o cálculo dos valores inferiores correspondentes.

Por esta razão, este processo recebeu na literatura [83] a designação de método de "*Atualizações Retardadas*".

Na primeira etapa, como no caso anterior, não se efetuam operações de combinação linear visando o cancelamento de elementos.

Iniciando-se a segunda etapa, multiplicando-se a primeira equação por (a_{12}/a_{11}) e subtraindo-a da segunda obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)^*} & a_{24}^{(2)^*} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3 \\ b_4 \end{pmatrix} \tag{B.63}$$

com

$$a_{22}^{(2)} = a_{22} - (a_{12}/a_{11})\, a_{12} \tag{B.64}$$

$$a_{23}^{(2)^*} = a_{23} - (a_{12}/a_{11})\, a_{13} \tag{B.65}$$

$$a_{24}^{(2)^*} = a_{24} - (a_{12}/a_{11})\, a_{14} \tag{B.66}$$

e

$$b_2^{(2)} = b_2 - (a_{12}/a_{11})\, b_1 \tag{B.67}$$

Iniciando-se agora a terceira etapa, multiplicando-se a primeira equação por (a_{13}/a_{11}) e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)^*} & a_{24}^{(2)^*} \\ 0 & a_{23}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)^*} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4 \end{pmatrix} \tag{B.68}$$

com

$$a_{23}^{(2)} = a_{23} - (a_{13}/a_{11})\, a_{12} \tag{B.69}$$

$$a_{33}^{(2)} = a_{33} - (a_{13}/a_{11})\, a_{13} \tag{B.70}$$

$$a_{34}^{(2)^*} = a_{34} - (a_{13}/a_{11})\, a_{14} \tag{B.71}$$

e

$$b_3^{(2)} = b_3 - (a_{13}/a_{11})\, b_1 \tag{B.72}$$

Multiplicando-se agora a segunda equação por $(a_{23}^{(2)}/a_{22}^{(2)})$ e subtraindo-a da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)^*} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)^*} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4 \end{pmatrix} \tag{B.73}$$

com

$$a_{33}^{(3)} = a_{33}^{(2)} - (a_{23}^{(2)}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.74}$$

$$a_{34}^{(3)^*} = a_{34}^{(2)} - (a_{23}^{(2)}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.75}$$

e

$$b_3^{(3)} = b_3^{(2)} - (a_{23}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.76}$$

Iniciando-se a quarta etapa, multiplicando-se a primeira equação por (a_{14}/a_{11}) e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)^*} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)^*} \\ 0 & a_{24}^{(2)} & a_{34}^{(2)} & a_{44}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(2)} \end{pmatrix} \tag{B.77}$$

com

$$a_{24}^{(2)} = a_{24} - (a_{14}/a_{11})\, a_{12} \tag{B.78}$$

$$a_{34}^{(2)} = a_{34} - (a_{14}/a_{11})\, a_{13} \tag{B.79}$$

$$a_{44}^{(2)} = a_{44} - (a_{14}/a_{11})\, a_{14} \tag{B.80}$$

e

$$b_4^{(2)} = b_4 - (a_{14}/a_{11})\, b_1 \tag{B.81}$$

Multiplicando-se agora a segunda equação por $(a_{24}^{(2)}/a_{22}^{(2)})$ e subtraindo-a da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)*} \\ 0 & 0 & a_{34}^{(3)} & a_{44}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(3)} \end{pmatrix} \tag{B.82}$$

com

$$a_{34}^{(3)} = a_{34}^{(2)} - (a_{24}^{(2)}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.83}$$

$$a_{44}^{(3)} = a_{44}^{(2)} - (a_{24}^{(2)}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.84}$$

e

$$b_4^{(3)} = b_4^{(2)} - (a_{24}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.85}$$

Finalmente, multiplicando-se a terceira equação por $(a_{34}^{(3)}/a_{33}^{(3)})$ e subtraindo-a da quarta obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{pmatrix} \tag{B.86}$$

com

$$a_{44}^{(4)} = a_{44}^{(3)} - (a_{34}^{(3)}/a_{33}^{(3)})\, a_{34}^{(3)} \tag{B.87}$$

e

$$b_4^{(4)} = b_4^{(3)} - (a_{34}^{(3)}/a_{33}^{(3)})\, b_3^{(3)} \tag{B.88}$$

Substituições Triangulares e Diagonais

Reagrupando-se o conjunto de operações efetuadas nas equações (B.11), (B.16, B.25) e (B.21, B.29, B.32) para cada componente do vetor lado direito b, (e substituindo-se $a_{ij}^{(k)*}$ por seus correspondentes valores $a_{ij}^{(k)}$) obtem-se:

$$b_2^{(2)} = b_2 - (a_{12}/a_{11})\, b_1 \tag{B.89}$$

$$b_3^{(3)} = b_3 - (a_{13}/a_{11})\, b_1 - (a_{23}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.90}$$

$$b_4^{(4)} = b_4 - (a_{14}/a_{11})\, b_1 - (a_{24}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} - (a_{34}^{(3)}/a_{33}^{(3)})\, b_3^{(3)} \tag{B.91}$$

Do mesmo modo, se considerarmos as equações (B.41), (B.46, B.50) e (B.55, B.59, B.62) para o segundo processo de eliminação, chegaremos ao mesmo resultado, tal ocorrendo também para as equações (B.67), (B.72, B.76) e (B.81, B.85, B.88) no caso do terceiro processo.

O que se mostrará a seguir, é que as operações expressas em (B.89), (B.90) e (B.91) na verdade correspondem a aplicação conjunta das fases de substituições *forward* (B.3) e *diagonal* (B.4) apresentadas no início do apêndice.

Por simplicidade de cálculo, em todos os exemplos apresentados, o sistema triangular obtido ao final do processo de eliminação não foi explicitamente normalizado, deixando as matrizes triangular superiores obtidas em (B.30), (B.60) e (B.86), com valores diagonais não unitários.

De modo a que se possa comparar os resultados obtidos, com uma fatoração na forma $U^T D\, U$, iniciaremos normalizando as equações (B.30), (B.60) e (B.86), dividindo cada uma das linhas do sistema por seus elementos diagonais associados, fornecendo:

$$U\ x\ =\ x'$$

$$\begin{pmatrix} 1 & a_{12}/a_{11} & a_{13}/a_{11} & a_{14}/a_{11} \\ & 1 & a_{23}^{(2)}/a_{22}^{(2)} & a_{24}^{(2)}/a_{22}^{(2)} \\ & & 1 & a_{34}^{(3)}/a_{33}^{(3)} \\ & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1/a_{11} \\ b_2^{(2)}/a_{22}^{(2)} \\ b_3^{(3)}/a_{33}^{(3)} \\ b_4^{(4)}/a_{44}^{(4)} \end{pmatrix} \tag{B.92}$$

De posse desta expressão, pode-se escrever as equações correspondentes as etapas *forward* e *diagonal* associadas:

$$U^T x''\ =\ b$$

$$\begin{pmatrix} 1 & & & \\ a_{12}/a_{11} & 1 & & \\ a_{13}/a_{11} & a_{23}^{(2)}/a_{22}^{(2)} & 1 & \\ a_{14}/a_{11} & a_{24}^{(2)}/a_{22}^{(2)} & a_{34}^{(3)}/a_{33}^{(3)} & 1 \end{pmatrix} \begin{pmatrix} x_1'' \\ x_2'' \\ x_3'' \\ x_4'' \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \tag{B.93}$$

$$D\ x'\ =\ x''$$

$$\begin{pmatrix} a_{11} & & & \\ & a_{22}^{(2)} & & \\ & & a_{33}^{(3)} & \\ & & & a_{44}^{(4)} \end{pmatrix} \begin{pmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{pmatrix} = \begin{pmatrix} x_1'' \\ x_2'' \\ x_3'' \\ x_4'' \end{pmatrix} \tag{B.94}$$

Da equação (B.94) podem ser explicitados os valores de x' componente a componente, obtendo-se:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \end{pmatrix} = \begin{pmatrix} x''_1/a_{11} \\ x''_2/a^{(2)}_{22} \\ x''_3/a^{(3)}_{33} \\ x''_4/a^{(4)}_{44} \end{pmatrix} \tag{B.95}$$

Comparando-se este resultado com as expressões correspondentes em (B.92), obtem-se:

$$\begin{pmatrix} x''_1 \\ x''_2 \\ x''_3 \\ x''_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b^{(2)}_2 \\ b^{(3)}_3 \\ b^{(4)}_4 \end{pmatrix} \tag{B.96}$$

O que comprova o fato de que os valores atualizados no vetor lado direito b, ao final do processo de eliminações triangulares, correspondem a aplicação conjunta das fases de substituição *forward* (B.3) e *diagonal* (B.4).

A título de complementação, pode-se explicitar a partir da equação $U^T x'' = b$ as expressões para cada uma das componentes de x'', como em (B.98), (B.99) e (B.100), comparando-as com as expressões (B.89), (B.90) e (B.91).

$$x''_1 = b_1 \tag{B.97}$$

$$x''_2 = b_2 - (a_{12}/a_{11})\, x''_1 \tag{B.98}$$

$$x''_3 = b_3 - (a_{13}/a_{11})\, x''_1 - (a^{(2)}_{23}/a^{(2)}_{22})\, x''_2 \tag{B.99}$$

$$x''_4 = b_4 - (a_{14}/a_{11})\, x''_1 - (a^{(2)}_{24}/a^{(2)}_{22})\, x''_2 - (a^{(3)}_{34}/a^{(3)}_{33})\, x''_3 \tag{B.100}$$

Cancelamentos por Linhas via Produtos Escalares

Além das alternativas consideradas préviamente para o cancelamento de elementos por linhas, uma outra forma, baseada no acumulo de global de todas contribuições sobre cada elemento da linha base, efetuado mediante operações de produto escalar, mostra-se possível (constituindo-se a base para as novas metodologias esparsas propostas a partir do capítulo 5).

O método a ser apresentado é conhecido na literatura como método de Crout, no caso de eliminações $L\,U$ de matrizes genéricas (de forma assimétrica). Outra variante possível no caso assimétrico é a eliminação de Doolittle, diferindo da anterior apenas na localização dos fatores diagonais unitários (que no primeiro caso por construção, encontram-se confinados na matriz U, e no segundo na matriz L).

No caso simétrico apenas um único procedimento (variante das eliminações supra mencionadas) é possível, e difere de uma metodologia de cancelamento por *linhas*

com geração dos fatores por *linhas*, apenas na ordem como se processam as contribuições sobre cada elemento da linha base de U sendo gerada.

Como nos demais métodos de cancelamentos por linhas, na primeira etapa, não se efetuam operações de eliminação por não existirem elementos a esquerda da diagonal a_{11}.

Iniciando-se a segunda etapa, multiplicando-se a primeira equação por (a_{12}/a_{11}) e subtraindo-a da segunda obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3 \\ b_4 \end{pmatrix} \tag{B.101}$$

com

$$a_{22}^{(2)} = a_{22} - (a_{12}/a_{11})\, a_{12} \tag{B.102}$$

$$a_{23}^{(2)} = a_{23} - (a_{12}/a_{11})\, a_{13} \tag{B.103}$$

$$a_{24}^{(2)} = a_{24} - (a_{12}/a_{11})\, a_{14} \tag{B.104}$$

e

$$b_2^{(2)} = b_2 - (a_{12}/a_{11})\, b_1 \tag{B.105}$$

Iniciando-se agora a terceira etapa, multiplicando-se a primeira equação por (a_{13}/a_{11}) e a segunda por $(a_{23}^{(2)}/a_{22}^{(2)})$, subtraindo-as da terceira, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4 \end{pmatrix} \tag{B.106}$$

com

$$a_{33}^{(3)} = a_{33} - (a_{13}/a_{11})\, a_{13} - (a_{23}^{(2)}/a_{22}^{(2)})\, a_{23}^{(2)} \tag{B.107}$$

$$a_{34}^{(3)} = a_{34} - (a_{13}/a_{11})\, a_{14} - (a_{23}^{(2)}/a_{22}^{(2)})\, a_{24}^{(2)} \tag{B.108}$$

e

$$b_3^{(3)} = b_3 - (a_{13}/a_{11})\, b_1 - (a_{23}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} \tag{B.109}$$

Iniciando-se a quarta etapa, multiplicando-se a primeira equação por (a_{14}/a_{11}), a segunda por $(a_{24}^{(2)}/a_{22}^{(2)})$ e a terceira por $(a_{34}^{(3)}/a_{33}^{(3)})$, subtraindo-as da quarta, obtem-se:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{pmatrix} \tag{B.110}$$

com

$$a_{44}^{(4)} = a_{44} - (a_{14}/a_{11})\, a_{14} - (a_{24}^{(2)}/a_{22}^{(2)})\, a_{24}^{(2)} - (a_{34}^{(3)}/a_{33}^{(3)})\, a_{34}^{(3)} \tag{B.111}$$

e

$$b_4^{(4)} = b_4 - (a_{14}/a_{11})\, b_1 - (a_{24}^{(2)}/a_{22}^{(2)})\, b_2^{(2)} - (a_{34}^{(3)}/a_{33}^{(3)})\, b_3^{(3)} \tag{B.112}$$

Este processo se comparado com o descrito desde as equações B.37 até B.62 mostra que o número total de operações aritméticas efetuadas é exatamente o mesmo, diferindo apenas na forma de geração dos fatores a cada etapa.

É possível também uma variante do processo de "atualizações retardadas" (com a geração dos fatores por *colunas*), como o apresentado originalmente desde as equações B.63 até B.88, seguindo-se os mesmos moldes da geração por *produtos escalares* anteriormente apresentada. Tal método não será formalmente apresentado, por não ser considerado ao longo deste trabalho.

Resumindo-se as alternativas de eliminação apresentadas neste apêndice, nota-se que os métodos classicamente consagrados na literatura são:

- Cancelamento (inferior) por *Colunas* com geração dos fatores (superiores) por *Sub-matrizes.*
- Cancelamento (inferior) por *Linhas* com geração dos fatores (superiores) por *Linhas*
- Cancelamento (inferior) por *Linhas* com geração dos fatores (superiores) por *Colunas.*
- Cancelamento (inferior) por *Linhas* com geração dos fatores (superiores) por *Linhas*, mediante *produtos escalares.*

Apêndice C

Dispêndio de Recursos Computacionais

Neste apêndice apresentam-se estimativas para o dispêndio de recursos computacionais de métodos diretos baseados em eliminações $U^T D U$ para a resolução de sistemas lineares.

Inicialmente se contabilizará o número de operações aritméticas e o espaço de armazenamento, para as alternativas clássicas da literatura, no caso de matrizes *densas*.

A seguir, comparam-se estes resultados, com o dispêndio tipicamente exibido por algumas das alternativas *esparsas* consideradas neste trabalho.

MATRIZES DENSAS

Contabilizando-se o número de multiplicações (idêntico ao de subtrações), para cada uma das alternativas consideradas na seção 2.4, obtem-se:

Alternativa 2.4(a) *Multiplicações ou Subtrações*

$$\begin{aligned}\sum_{i=1}^{n}\sum_{k=i+1}^{n}\sum_{j=k}^{n}(1) &= \sum_{i=1}^{n}\sum_{k=i+1}^{n}(n-k+1) \\ &= \sum_{i=1}^{n}\left(\frac{(i-n)(i-n-1)}{2}\right) \\ &= \frac{n^3}{6}-\frac{n}{6}\end{aligned} \tag{C.1}$$

Alternativa 2.4(b) *Multiplicações ou Subtrações*

$$\sum_{i=1}^{n}\sum_{k=1}^{i-1}\sum_{j=i}^{n}(1) = \sum_{i=1}^{n}\sum_{k=1}^{i-1}(n-i+1)$$

$$= \sum_{i=1}^{n} ((n - i + 1) (i - 1))$$
$$= \frac{n^3}{6} - \frac{n}{6} \qquad \text{(C.2)}$$

Alternativa 2.4(c) *Multiplicações ou Subtrações*

$$\sum_{i=1}^{n} \sum_{k=1}^{i-1} \sum_{j=k+1}^{i} (1) = \sum_{i=1}^{n} \sum_{k=1}^{i-1} (i - k)$$
$$= \sum_{i=1}^{n} \left(\frac{i\,(i-1)}{2} \right)$$
$$= \frac{n^3}{6} - \frac{n}{6} \qquad \text{(C.3)}$$

Contabilizando-se agora isoladamente as operações de divisão (efetuadas ao se normalizar os fatores) temos:

Alternativa 2.4(a) *Divisões*

$$\sum_{i=1}^{n} \sum_{k=i+1}^{n} (1) = \sum_{i=1}^{n} (n - i)$$
$$= \frac{n^2}{2} - \frac{n}{2} \qquad \text{(C.4)}$$

Alternativa 2.4(b) *Divisões*

$$\sum_{i=1}^{n} \sum_{k=1}^{i-1} (1) = \sum_{i=1}^{n} (i - 1)$$
$$= \frac{n^2}{2} - \frac{n}{2} \qquad \text{(C.5)}$$

Alternativa 2.4(c) *Divisões*

$$\sum_{i=1}^{n} \sum_{k=1}^{i-1} (1) = \sum_{i=1}^{n} (i - 1)$$
$$= \frac{n^2}{2} - \frac{n}{2} \qquad \text{(C.6)}$$

Sintetizando-se os resultados obtidos (uma vez que o volume total de cálculo de cada uma das alternativas é o mesmo), temos:

Volume de Cálculo (*Denso*)

Eliminação (*Densa*)	*Operações Aritméticas*
Multiplicações	$1/6\ n^3 - 1/6\ n$
Subtrações	$1/6\ n^3 - 1/6\ n$
Divisões	$1/2\ n^2 - 1/2\ n$
(*Total*)	$1/3\ n^3 + 1/2\ n^2 - 5/6\ n$

Eliminação (*Densa*)	*Operações Aritméticas*
	$O(n^3)$

(C.7)

Uma vez que o processo completo de solução, engloba também as fases de substituições de variáveis, a título de comparação, apresenta-se a seguir, o volume de cálculo dispendido globalmente nas 3 fases (*forward*, *diagonal* e *backward*).

Substituição (*Densa*)	*Operações Aritméticas*
Multiplicações	$n^2 - n$
Subtrações	$n^2 - n$
Divisões	n
(*Total*)	$2\ n^2 - n$

Substituição (*Densa*)	*Operações Aritméticas*
	$O(n^2)$

(C.8)

Uma vez contabilizadas as operações aritméticas, passaremos a considerar o dispêndio de recursos em termos de espaço de armazenamento.

Espaço de Armazenamento (*Denso*)

Em todos os casos, assume-se que uma posição de memória corresponda a um valor múltiplo de uma palavra de armazenamento, suficiente para comportar a representação numérica de coeficientes em ponto flutuante numa dada precisão.

Matriz de Fatores (Densa)	*Posições de Memória*
Porção Triangular	$1/2\ n^2 - 1/2\ n$
Diagonal	n
(*Total*)	$1/2\ n^2 + 1/2\ n$
Matriz de Fatores (Densa)	*Posições de Memória*
	$O(n^2)$

(C.9)

MATRIZES ESPARSAS

Passaremos agora a considerar estimativas para o volume de cálculo (e posteriormente de armazenamento) no caso de alternativas para a resolução de sistemas *esparsos*.

Volume de Cálculo (*Esparso*)

Nas contabilizações a seguir, lança-se mão das notações $\eta(U)$ para o número total de elementos não nulos da matriz de fatores U, e $\eta^{(\cdot)}(U)$ para o número de elementos não nulos em cada uma de suas linhas, introduzidas na seção 2.2.

Eliminação (Esparsa)	*Operações Aritméticas*
Multiplicações	$1/2 \sum_{i=1}^{n-1} \left(\eta^{(i)}(U)^2 + \eta^{(i)}(U) \right)$
Subtrações	$1/2 \sum_{i=1}^{n-1} \left(\eta^{(i)}(U)^2 + \eta^{(i)}(U) \right)$
Divisões	$\sum_{i=1}^{n-1} \left(\eta^{(i)}(U) \right)$
(*Total*)	$\sum_{i=1}^{n-1} \left(\eta^{(i)}(U)^2 \right) + 2\ \eta(U)$
Eliminação (Esparsa)	*Operações Aritméticas*
	$O(\sum_{i=1}^{n-1} \eta^{(i)}(U)^2)$

(C.10)

Contabilizando-se agora o volume global de cálculo das fases de substituições esparsas, teremos:

Substituição (Esparsa)	*Operações Aritméticas*
Multiplicações	$2\sum_{i=1}^{n-1}\left(\eta^{(i)}(U)\right)$
Subtrações	$2\sum_{i=1}^{n-1}\left(\eta^{(i)}(U)\right)$
Divisões	n
(*Total*)	$4\eta(U)+n$

Substituição (Esparsa)	*Operações Aritméticas*
	$O(\eta(U))$

(C.11)

Consideraremos a seguir, especificamente apenas o espaço de armazenamento para a representação numérica dos fatores triangulares e diagonais nas alternativas de resolução esparsa.

Espaço de Armazenamento (*Esparso*)

Matriz de Fatores (Esparsa)	*Posições de Memória*
Porção Triangular	$\sum_{i=1}^{n-1}\left(\eta^{(i)}(U)\right)$
Diagonal	n
(*Total*)	$\eta(U)+n$

Matriz de Fatores (Esparsa)	*Posições de Memória*
	$O(\eta(U))$

(C.12)

Apresenta-se a seguir, um breve resumo do dispêndio de recursos, tomando-se para fins de comparação, os dispêndios de algumas das caracterizações para matrizes *esparsas*, apresentadas na seção 2.2, bem como o volume dispendido particularmente no caso de matrizes *densas*.

Dispêndio de Recursos (em função de alguns *Padrões de Esparsidade*)

Na Tabela C.1 a seguir, $\eta_{flop}^{fator.}(U)$ denota uma estimativa para o número total de operações aritméticas efetuadas em procedimentos de fatoração *esparsa*, e $\eta_{flop}^{subst.}(U)$ o volume de cálculo dispendido globalmente nas fases de substituições de variáveis. *Dens.* indica a densidade tomada como padrão, *Armaz.* o espaço de armazenamento, em termos de posições de memória requeridas e *Defin. Espars.* a definição de *esparsidade* utilizada para fins de contabilização.

Defin. Espars.	*Dens.*	$\eta^{(i)}(U)$	$\eta_{flop}^{fator.}(U)$	$\eta_{flop}^{subst.}(U)$	*Armaz.*
2.2.11	ρ	$\rho\, n$	$\rho^2 n^3$	$4\,\rho^2 n^2$	$\rho\, n^2$
2.2.14	$1\,/\,n$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
2.2.15	$\kappa\,/\,n$	κ	$\kappa^2 n$	$4\,\kappa\, n$	$\kappa\, n$
2.2.16	$n^\gamma/\,n$	n^γ	$n^{1+2\gamma}$	$4\,n^{1+\gamma}$	$n^{1+\gamma}$
Matriz Densa	1	$n-i$	$n^3/\,3$	$2\,n^2$	$n^2/\,2$

Tab C.1: Dispêndio de Recursos em função de Padrões de Esparsidade

Apêndice D

Glossário

Visando complementar a notação introduzida na seção 1.1, apresenta-se a seguir, um glossário de terminologias e expressões relacionadas aos temas abordados no presente trabalho.

Acesso Leitura ou Escrita a dados armazenados em Posições de Memória.

Acesso por Apontadores Esquema particular de acesso Randômico a dados cuja localização na memória se encontra especificada por meio de mecanismos (ou variáveis) "apontadores". Disponível em linguagens como C/C^{++}, FORTRAN 90, ADA, PASCAL, PL/I e Assembler.

Acesso Contíguo Acesso a dados armazenados em posições Consecutivas de memória.

Acesso Direto Acesso a posições Pré-Estabelecidas de memória (como UN[5] ou DI[7] por exemplo).

Acesso Indireto Acesso do tipo Randômico obtido particularmente a partir da utilização conjugada de vetores "duplamente aninhados" (como W[JU[j]] por exemplo).

Acesso Linear O mesmo que *Acesso Contíguo.*

Acesso Randômico Acesso a posições de memória impossíveis de serem precisadas Determinísticamente durante a fase de compilação (ou geração de código de Máquina). Normalmente efetuado a partir de acessos da forma Indireta.

Acesso Seqüencial O mesmo que *Acesso Contíguo.*

Algoritmo Seqüência de etapas (instruções ou regras) a serem seguidas de modo a se alcançar uma Solução Computacional para um dado problema.

Alocação Dinâmica Recurso disponível em algumas linguagens computacionais, e que permite a definição "dinâmica" de estruturas de dados (durante a execução de um programa). As estruturas alocadas dinâmicamente oferecem a vantagem de poderem ser "liberadas" quando não mais necessárias, permitindo a utilização das áreas de memória liberadas para alocações dinâmicas de novas estruturas.

Alocação Estática Forma convencional como se reservam previamente em tempo de compilação, as áreas de memória a serem utilizadas por cada uma das estruturas

de dados declaradas num programa.

Alternativa Utilizada para a designação de algoritmos para os quais se conhece um leque de metodologias "alternativas" de solução (tipicamente com dispêndios similares a nível de recursos computacionais).

Ancestral Nó de uma Árvore, cujo índice associado é inferior ao de um nó (Descendente) originalmente especificado.

Ancestral Direto Particular nó Ancestral em uma Árvore, "diretamente" interconectado a um dado nó (Descendente) originalmente especificado.

Antecessor O mesmo que *Ancestral Direto.*

Apontador Tipo especial de variável, disponível em algumas linguagens computacionais, cujo conteúdo especifica diretamente uma localização de memória. Utilizado na construção e alocação de estruturas dinâmicas, como listas, ou em algumas formas de acesso do tipo Randômico.

Arquitetura Computacional Designação empregada para a classificação de aparatos computacionais ao nível das características intrínsecas de suas unidades de processamento, sub-dividas em classes como as Escalares, Vetoriais, Paralelas, Superescalares, Hipercúbicas, CISC, RISC, SIMD, MIMD entre outras.

Árvore de Eliminação Estrutura característica do grau de "parentesco" e de Interdependência entre linhas da Matriz de Fatores (associadas respectivamente a cada um de seus nós), podendo-se por seu intermédio trivialmente determinar-se subconjuntos de linhas cujo Processo de Eliminação é totalmente Independente entre si.

Assembler Linguagem computacional de mais Baixo Nível, específica para cada tipo de máquina ou "classe" de Arquiteturas Computacionais (de uma mesma família ou série).

Atualização O mesmo que *Eliminação.*

Balanceamento Sub-divisão de tarefas a serem executadas "paralelamente" em Múltiplos Processadores, visando um melhor desempenho global, através da minimização do tempo ocioso entre processadores a espera de dados provenientes de outros correntemente envolvidos em atividades de cálculo.

Base Termo associado a uma Etapa Fundamental do Processo de Eliminação na qual se efetuam todas as operações necessárias para se gerar uma particular linha ou coluna da Matriz de Fatores.

Bit Menor quantidade Digitalmente Armazenável de informação, podendo assumir apenas os valores 0 ou 1.

Buffer Estrutura de dados ou aparato computacional projetado para o armazenamento temporário de informações. No caso de estruturas de dados, normalmente são utilizadas áreas (ou vetores) de trabalho (podendo ser dinâmicamente alocadas durante a execução e posteriormente liberadas, nas linguagens onde tais recursos

são oferecidos).

Byte Grandeza correspondente ao numero binário formado por 8 bits consecutivos.

C Linguagem computacional de Alto Nível, desenvolvida na ATT Bell Labs ao final da década de 70, e que possui como principais vantagens do ponto de vista da eficiência de código gerado, facilidades na manipulação de apontadores e no endereçamento direto à memória, além de primitivas muito próximas as encontradas apenas em linguagens do tipo Assembler.

C++ Extensão da linguagem C, de modo a torna-la Orientada a Objetos.

Cache Memória intermediária de Acesso Rápido, utilizada para o armazenamento temporário de dados ou instruções a serem potencialmente requeridas em algum momento futuro, durante a execução de um programa. O tempo de acesso às memórias Cache, situa-se normalmente entre o tempo de acesso à Registradores e o tempo de acesso à Memória Principal.

Cache de Dados Memória Cache voltada para o armazenamento de Dados recentemente acessados de registradores ou da memória (sendo normalmente do tipo "associativo" nas arquiteturas de maior desempenho atuais).

Cache de Instruções Memória Cache voltada para o armazenamento de Instruções de programa a serem potencialmente executadas ou re-executadas em algum momento futuro durante a execução de um programa.

Ciclos de Maquina Menor unidade de contabilização do tempo de tarefas computacionais (baseada no Clock de operação da CPU). Normalmente um ciclo de máquina corresponde ao menor tempo de execução passível de ser dispendido na execução das mais simples dentre todas as Macro-Instruções de Máquina disponíveis em cada arquitetura.

CISC (Complex Instruction Set Computer) Arquitetura Computacional baseada em processadores capazes de executar Macro-Instruções "elaboradas" (podendo vir a requerer um número não unitário de ciclos de máquina para sua execução).

Clique Conjunto particular de nós de um grafo, totalmente interconectados entre si.

Clock Grandeza característica da velocidade de processamento de uma CPU (indicativa do número de Ciclos de Máquina proporcionalmente executáveis por segundo). Normalmente medida Hertz, com valores para as arquiteturas escalares de médio porte atuais, na faixa de uma ou duas centenas de MHz.

Código Informação necessária para a caracterização de seqüências de operações aritméticas a serem efetuadas durante a Fase Numérica do Processo de Eliminações. Normalmente contendo de modo "agregado" a especificação de operandos efetivamente necessários em acumulações da forma $x_{i,j} \leftarrow x_{i,j} - \sum_k y_k * z_{k,j}$ (para cada coluna j da i'ezima etapa) de modo a se explorar os padrões de esparsidade envolvidos neste processo. Na literatura computacional, o termo Código muitas vezes é utilizado para a designação de um programa em Linguagem de Máquina (já compilado

ou montado) a ser executado, o que neste trabalho será destacado explicitamente quando se tratar deste caso, de modo a evitar-se ambigüidades com a designação primordial adotada ao longo do texto.

Código Binário Código baseado numa representação "binária" para o Padrão de Contribuições (Não Nulas) por colunas, restrito ao grupo de linhas k envolvidas em acumulações da forma $x_{i,j} \leftarrow x_{i,j} - \sum_k y_k * z_{k,j}$ para cada coluna j (associada a cada código).

Código por Envelopes Código baseado numa representação por "envelopes" para o Padrão de Contribuições (Não Nulas) por colunas, restrito ao grupo de linhas k envolvidas em acumulações da forma $x_{i,j} \leftarrow x_{i,j} - \sum_k y_k * z_{k,j}$ para cada coluna j (associada a cada código).

Código Supernodal Código baseado numa representação "supernodal" para o Padrão de Contribuições (Não Nulas) de um grupo de colunas, restrito ao grupo de linhas k envolvidas em acumulações da forma $x_{i,j} \leftarrow x_{i,j} - \sum_k y_k * z_{k,j}$ para um dado grupo de colunas j (associadas a um mesmo código).

Codificação Esquema de processamento voltado para geração de informações Simbólicas na forma de "listas de códigos" (ou de endereços) a serem interpretadas durante a Fase Numérica de rotinas de fatoração (especificamente projetadas para sua exploração). Em todos os Esquemas de Codificação, é assumido o pleno conhecimento das Características Estruturais da Matriz de Fatores resultante ao final do processo de fatoração (normalmente obtida em uma fase preliminar de processamento, como a de Fatoração Simbólica). Na literatura o termo Codificação é utilizado também para a designação de "código de máquina" ou programa (já codificado) para a execução, o que análogamente ao comentado na definição de *Código*, se evitará adotar ao longo deste trabalho, de modo a minimizar-se eventuais ambigüidades de interpretação.

Codificação Binária Esquema de processamento voltado para a geração de uma Codificação a ser interpretada durante uma Fase Numérica de fatoração, dedicada ao tratamento de "listas de códigos" do tipo Binário.

Codificação por Envelopes Esquema de processamento voltado para a geração de uma Codificação a ser interpretada durante uma Fase Numérica de fatoração, dedicada ao tratamento de "listas de códigos" do tipo Envelope.

Codificação Simbólica O mesmo que *Codificação.*

Codificação Supernodal Esquema de processamento voltado para a geração de uma Codificação a ser interpretada durante uma Fase Numérica de fatoração, dedicada ao tratamento de "listas de códigos" do tipo Supernodal.

Coluna Base O mesmo que *Coluna Básica.*

Coluna Básica Coluna associada ao índice de uma dada Etapa Básica do Processo de Eliminação, a ser operada (sofrendo Contribuições de elementos de outras linhas ou colunas) de modo a se obterem seus Fatores Associados.

Compactação Transformação de uma seqüência de códigos em outra de menor ta-

manho, visando a redução de espaço de armazenagem. O processo de transformação deve ser tal que garanta a unicidade da seqüência de tamanho reduzido obtida, de modo a permitir uma futura descompactação das informações, reproduzindo a seqüência originalmente compactada.

Compilação Tradução de um programa em uma linguagem de Alto Nível para uma versão executável em Linguagem de Máquina.

Complexidade Medida normalmente utilizada para se quantificar o dispêndio de recursos de uma determinada grandeza, pela associação a funções limitantes conhecidas, cujo comportamento ao se tender ao infinito venha a diferir apenas por um fator escalar (independente) do comportamento da grandeza originalmente analisada.

Concorrência O mesmo que *Paralelismo.*

Concorrente Relativo ao que pode ser executado de forma simultânea, por múltiplos processadores (em Paralelo).

Condicionamento Medida característica de uma matriz, auxiliar na estimação do grau de sensibilidade à variações nos dados de entrada, e da "estabilidade numérica" e precisão teoricamente esperável durante o processo de fatoração matricial. Formalmente pode ser expresso pela razão entre o maior e o menor autovalor associados a matriz analisada, na prática sendo porém aproximado por medidas de mais simples determinação computacional como as propostas em [41], [24] e [4].

Contíguo Acesso ou armazenamento a elementos em posições Consecutivas de memória.

Contribuição Grandeza correspondente ao valor de um dado Coeficiente de alguma das linhas da Matriz de Fatores a serem subtraídas da Linha Base numa Etapa Corrente do Processo de Eliminação.

Corrente O mesmo que *Base.*

CPU (Central Processor Unit). O mesmo que *Unidade Central de Processamento.*

Densidade Medida do percentual de elementos não nulos de uma matriz.

Descendente Nó de uma Árvore, cujo índice associado é superior ao de um nó (Ancestral) originalmente especificado.

Descendente Direto Particular nó Descendente em uma Árvore, "diretamente" interconectado a um dado nó (Ancestral) originalmente especificado.

Desenrolamento de Loops Técnica voltada para o aumento na eficiência de execução de *loops*, passível de incorporação automática em compiladores, geradores de código de Máquina, e mais recentemente até mesmo pelo próprio *hardware* de arquiteturas mais avançadas, e que originalmente era aplicada "manualmente" por programadores mais experientes. Consiste em se explicitamente codificar cada uma das instruções que seriam executadas em um determinado número de iterações de um *loop*, simplesmente enumerando-as seqüencialmente num longo trecho "desenrolado" de código,

reduzindo-se com isso a freqüência de desvios, incrementos da variável de controle e testes condicionais que seriam desnecessariamente executados no *loop* original.

Desvio Computado O mesmo que *Desvio Indexado.*

Desvio Condicional Desvio da execução de um programa para um novo ponto, em função do valor dinâmicamente assumido por uma expressão de "controle" (correspondendo na maioria das linguagens de Alto Nível à construções da forma IF *expressão* THEN GOTO *label*).

Desvio Indexado Desvio condicional Múltiplo com o endereço de desvio Indexado por intermédio de um valor inteiro positivo, indicador da posição relativa do endereço a se desviar (dentre o conjunto de endereços especificados numa tabela de desvios). Em FORTRAN correspondendo ao "Computed" GOTO, e em linguagens como PASCAL ou C/C^{++}, podendo ser implementado respectivamente por intermédio das construções CASE e SWITCH respectivamente.

Desvio Especulativo Desvio para um ponto "teoricamente esperado" na execução de um programa, normalmente para a posição de re-início de um *loop* ou para um ramo selecionado em expressões da forma IF ... THEN ... ELSE. A previsão pode ser efetuada estáticamente durante a compilação, ou dinâmicamente durante a execução (em função de alguma estatística como a melhor taxa de acerto dentre as seleções). No caso de "erro" na previsão do desvio, normalmente um mecanismo do próprio hardware se encarrega de restaurar todo o contexto (valores e *status* originais de todos os registradores e posições de memória erroneamente afetadas). Recursos como os de Desvio Especulativo normalmente são encontrados em arquiteturas do tipo RISC ou Superescalares, onde o acerto na previsão de desvios, especialmente durante a execução de *loops* pode vir a levar a um aumento significativo na eficiência computacional (por permitir uma plena exploração de mecanismos de "encadeamento" de instruções como os do tipo pipeline).

Desvio Incondicional Desvio na execução de um programa, para um determinado ponto Estáticamente especificado (previamente conhecido durante a fase de compilação). Em quase todas as linguagens computacionais designado através da construção GOTO (simples) seguida de um "label" associado a posição especificada.

Diretiva O mesmo que *Diretiva de Compilação.*

Diretiva de Compilação Mecanismo disponível em algumas linguagens e ambientes computacionais, em que por meio de instruções não executáveis se especificam opções ou ações a serem seguidas durante a fase de compilação de um programa. Muito utilizado em ambientes do tipo Vetorial, onde diretivas características a cada compilador ou arquitetura se encontram normalmente disponíveis, permitindo uma melhor exploração de recursos e controle do grau de Vetorização efetivamente obtido nos Códigos de Máquina gerados.

Dupla Palavra Quantidade correspondente a Duas Palavras de armazenamento, normalmente utilizada no armazenamento de dados de ponto flutuante em Precisão Dupla (ou "estendida").

Dupla Precisão O mesmo que *Precisão Dupla.*

Elementos Menores entidades passíveis de extração Isolada de uma estrutura de dados, como os coeficientes de uma matriz ou as componentes de um vetor. Quando relativos a uma matriz Esparsa designam apenas os seus coeficientes Estruturalmente Não Nulos. Quando relativos a uma Matriz de Fatores, designam seus Fatores Triangulares ou Diagonais (incluindo-se deste modo os elementos de *Fill-In*, além dos demais elementos não nulos oriundos da Matriz Original).

Elementos Associados O mesmo que *Fatores Associados.*

Elementos Estruturalmente Não Nulos Elementos "potencialmente" não nulos de uma matriz Esparsa, cuja existência "teórica" pode ser determinada tomando como base apenas informações de seu caráter Estrutural. Podem em raras ocasiões, eventualmente virem a se tornar numericamente nulos em virtude de cancelamentos numéricos durante o processo de eliminação de variáveis.

Eliminação O mesmo que *Eliminação de Variáveis.*

Eliminação de Variáveis Processo de cancelamento de elementos de modo a se levar a Matriz Original de um sistema linear à Forma Triangular.

Endereçamento Apontado Esquema particular de endereçamento Randômico e acesso a dados cuja localização em memória se encontra especificada por meio de mecanismos ou variáveis "apontadoras" (como *ptr por exemplo). Disponível em linguagens como C/C^{++}, PASCAL, FORTRAN 90 e Assembler.

Endereçamento por Apontadores O mesmo que *Endereçamento Apontado.*

Endereçamento Direto Esquema de endereçamento e acesso a dados cuja localização foi explícita e "diretamente" especificada no programa fonte (como UN[5] ou DI[7] por exemplo), permitindo sua determinação estática durante a fase de compilação.

Endereçamento Indexado Esquema de endereçamento e acesso a dados a partir da especificação (dinâmica) de um Índice relativo a um endereço base conhecido (como UN[j] por exemplo) permitindo uma determinação simplificada da localização requerida, durante a fase de execução. Requer Macro-Instruções de Máquina apropriadas para acessos por meio de deslocamentos relativamente indexados a partir de um endereço base (encontradas com freqüência em praticamente todas as arquiteturas computacionais desenvolvidas).

Endereçamento Indireto Esquema particular de endereçamento Randômico, baseado na utilização de vetores "duplamente aninhados" (como W[JU[j]] por exemplo).

Endereçamento Randômico Esquema de endereçamento e acesso a dados armazenados em posições de memória impossíveis de serem precisadas Determinísticamente durante a compilação de um programa (ou geração de seu código de Máquina). Mais frequentemente encontrado sob a forma de endereçamentos do tipo Indireto.

Endereço Posição na memória, associada a um dado elemento na representação de

uma estrutura de dados.

Escalar Tipo convencional de arquitetura computacional, baseada em um único Processador Central, e capaz de executar no máximo uma Macro-Instrução por vez a cada Ciclo de Máquina.

Esparsidade Característica Estrutural de uma matriz, normalmente explorada na presença de poucos elementos não nulos a serem computacionalmente tratados (armazenados ou operados).

Estrutura de Armazenamento Agrupamento ou Disposição Estrutural de dados idealizada de modo a ser implementada em uma determinada linguagem computacional (visando a solução de um problema).

Estrutural Informação apenas a nível da localização (*linha, coluna*) associada aos elementos "potencialmente" não nulos de uma matriz.

Etapa O mesmo que *Etapa Básica.*

Etapa Base O mesmo que *Etapa Básica.*

Etapa Básica Seqüência Fundamental de operações no Processo de Eliminação, correspondendo a uma iteração de seu *loop* mais externo (em termos da variável i), onde são gerados os elementos na i'ezima linha ou coluna da Matriz de Fatores.

Etapa Fundamental O mesmo que *Etapa Básica.*

Família Grupo de nós constituintes de uma cadeia de descendentes diretos, localizados num mesmo "ramo" de uma Árvore de Eliminação.

Fase Grupo particular de operações voltadas para se alcançar um mesmo objetivo, dentro do processo de resolução de sistemas de equações lineares como um todo. Correspondendo a resolução de sub-problemas com características peculiares e distintas entre si, como as fases de reordenamento visando redução de *fill-in*, fatoração simbólica, codificação simbólica, fatoração numérica e substituições (*forward, diagonal* e *backward*).

Fase Numérica O mesmo que *Fatoração Numérica.*

Fatoração O mesmo que *Eliminação.*

Fatoração Matricial O mesmo que *Fatoração.*

Fatoração Numérica Fase do processo de solução de um sistema linear, correspondente a geração dos Valores Numéricos dos elementos da Matriz de Fatores.

Fatoração Simbólica Fase correspondente a determinação da posição Estrutural dos fatores Não Nulos de U, incluindo-se a disposição dos elementos de *fill-in* a serem gerados na Fase Numérica.

Fatoração Triangular O mesmo que *Fatoração.*

Fatores Coeficientes Estruturalmente Não Nulos da Matriz (de Fatores) resultante do processo de Eliminação de Variáveis, efetuado visando a transformação do Sis-

tema Original numa Forma Triangular.

Fatores Associados Elementos de uma dada Linha Base da Matriz de Fatores, respectivamente correspondentes a elementos não nulos de Mesma Coluna de um particular grupo de linhas Contribuintes especificado.

Fatores Diagonais Elementos da Matriz Diagonal de Fatores D, normalmente armazenados sob a forma do inverso de seus valores numéricos, visando uma maior eficiência durante as fases de Substituição de Variáveis subseqüentes.

Fatores Triangulares Elementos da Matriz Triangular de Fatores (excluindo-se a diagonal unitária), gerada gradualmente a cada etapa do processo de Eliminação de Variáveis.

Fill-In Elemento Não Nulo introduzido na Matriz de Fatores (em uma posição correspondente a um elemento nulo na Matriz Original), em decorrência da Contribuição de outras linhas durante a geração de uma linha ou coluna base a cada etapa do Processo de Eliminação.

Flop Unidade de contabilização na qual pares de operações de Ponto Flutuante da forma $x \pm y * z$ são considerados como uma Única operação "agregada".

Forma Triangular Forma em que é levada a Matriz Original de um sistema, após sua Fatoração por algum processo de Eliminação de Variáveis. Estruturalmente correspondendo a uma matriz do tipo Triangular (normalmente Superior) com elementos diagonais (normalmente) unitários.

FORTRAN (Formula Translator) Uma das primeiras linguagens computacionais desenvolvidas, cuja primeira implementação é datada de meados dos anos 50. Voltada basicamente para o processamento numérico, e posteriormente estendida ao longo de sucessivas versões de modo a acompanhar os aprimoramentos introduzidos em outras linguagens.

FORTRAN IV Uma das primeiras especificações consagradas e universalmente aceitas da linguagem, na qual se produziu uma vasta quantidade de programas em décadas passadas.

FORTRAN 66 O mesmo que *FORTRAN IV*.

FORTRAN 77 Extensão de maior reconhecimento e utilização da linguagem até os dias atuais, e que introduziu algumas facilidades no processamento de caracteres e expressões alfanuméricas, além de alguns recursos de estruturação como construções IF ... THEN ... ELSE.

FORTRAN 90 Padrão mais recentemente especificado para extensão da linguagem, e que incorpora entre outras vantagens, facilidades padronizadas de vetorização (e de operação sobre estruturas vetoriais), maior estruturação com um amplo leque de tipos abstratos de dados, incluindo também recursos como os de alocação dinâmica e acesso direto a posições de memória via o uso de estruturas apontadoras.

Gather/Scatter Recurso disponível em algumas arquiteturas mais avançadas (como as do tipo Vetorial), e que permite aumentar-se a eficiência de acessos do tipo Indi-

reto, mediante a utilização de mecanismos de *hardware* dedicados a: (a) "agrupar" num mesmo vetor, dados que se encontrem espaçados na memória. (b) operar de forma compacta, sobre os mesmos. (c) "devolve-los" aos seus endereços originais.

Geração de Código Sub-produto da compilação de um programa em linguagem de Alto Nível, ou da montagem de um programa em Assembler, fornecendo a seqüência correspondente de instruções de Máquina a serem executadas. Em muitos compiladores, a Geração de Código é uma das fases terminais do processo de compilação, sendo muitas das vezes opcionalmente sucedida por uma fase de otimização do código de Máquina gerado.

Grafo Associado Grafo conexo e não orientado, associado a representação esparsa dos elementos não nulos de uma matriz irredutível e simétrica.

Grafo de Eliminação Grafo conexo e não orientado, inicialmente associado a representação esparsa dos elementos não nulos de uma matriz original irredutível e simétrica, sendo transformado a cada etapa da simulação de um processo de eliminação (inferior) por colunas, com geração dos fatores triangulares superiores por sub-matrizes.

Grafo de Fatores Grafo não orientado, associado a representação esparsa dos fatores não nulos gerados a cada etapa de um processo de eliminação de uma matriz simétrica.

Grafo de Fatores Resultantes O mesmo que Grafo de Fatores.

Hardware Relativo a qualquer aparato físico ou componente eletro-mecânico, voltado para o desempenho de alguma tarefa ou função de natureza computacional.

Huffman Idealizador de uma das formas mais eficientes de compactação, baseada na noção de códigos de tamanho variável, selecionados em função dos padrões mais freqüentemente encontrados (ou supostamente passíveis de se encontrar) em uma dada seqüência de informações a ser codificada de forma compacta. A adoção de códigos de menor tamanho para a representação dos padrões mais freqüentes, viabiliza a compactação, por minimizar o volume de informação necessário para a completa caracterização de toda a seqüência de códigos.

i Normalmente correspondente a variável Básica do processo de eliminação, associada ao *loop* Mais Externo, indicando a linha ou coluna da matriz de fatores a ser gerada a cada Etapa.

Incremento Implícito Mecanismo auxiliar de acesso, disponível em algumas arquiteturas (como a VAX), e que automaticamente permite o incremento "implícito" do registrador utilizado para indexação no acesso a dados (antes ou após a conclusão de um grupo particular de Macro-Instruções de Máquina, dotadas desta facilidade). Sua maior utilidade está na obtenção de economia de instruções e ciclos de maquina, durante acessos do tipo seqüencial a memória, como os tipicamente encontráveis em *loops*.

Interpretação Forma de translação "dinâmica" (e implícita) de programas escritos em determinadas linguagens de Alto Nível, mediante chamadas a procedimentos em

Linguagem de Máquina dedicados a processar especificamente cada uma das de instruções de Alto Nível (a cada instância em que suas execuções são requeridas). Por esta razão, programas "interpretados" tendem a ser naturalmente menos eficientes do que os escritos nas demais linguagens passíveis de Compilação. Algumas linguagens "pseudo-interpretadas", na verdade só "interpretam" um dado programa fonte uma única vez, pois após a primeira interpretação, acabam por de fato gerar um Código de Máquina (dinâmicamente) Compilado e que será o chamado nas próximas instâncias em que uma nova execução for requerida.

Irredutível Sistema linear, cuja matriz de coeficientes possua ao menos um elemento não nulo excluindo-se a diagonal, em cada uma de suas linhas e colunas.

j Normalmente correspondente a variável associada ao *loop* Mais Interno do processo de eliminação, correspondendo a posição dos elementos não nulos de cada uma das linhas anteriores, a serem subtraídas da linha base corrente.

Janela de Processamento Porção determinada de uma matriz de fatores, a sofrer um tratamento diferenciado e particularmente mais indicado para uma implementação localizada do processo de eliminação envolvendo apenas os elementos contidos nesta porção. Como exemplos pode-se citar as janelas diagonais (ou completamente esparsas), as janelas "quase" densas, as janelas completamente densas (ou cheias) e as janelas supernodais.

k Normalmente correspondente a variável associada ao *loop* Intermediário do processo de eliminação, varrendo a estrutura da matriz U^T, de modo a se determinar quais linhas anteriores deverão ser subtraídas da linha base corrente.

Label Indicador (ou rótulo mnemônico / alfanumérico) associado a algum ponto particular de um programa. Normalmente utilizado para a especificação de pontos a sofrerem eventualmente um posterior desvio durante a fase de execução de um programa.

Linguagem Computacional Padrão reconhecido e formalizado de linguagem, voltada para a especificação e desenvolvimento de "programas" (ou bibliotecas de subrotinas) a serem posteriormente transportados para alguma arquitetura computacional particular, e compilados ou interpretados para a linguagem da Máquina por ocasião em que sua execução for requerida.

Linguagem de Alto Nível Linguagem computacional em que as instruções e facilidades de manipulação de dados disponíveis são mais "amenas" ao programador comum, e normalmente abrangem um conjunto de recursos mais elaborados que os disponíveis diretamente nas linguagens de mais Baixo Nível. Como exemplo de linguagens de Alto Nível pode-se citar ADA, ALGOL, C/C^{++}, FORTRAN, MODULA 2, PASCAL e PL/I entre outras.

Linguagem de Baixo Nível Linguagem computacional voltada especialmente para a interação direta com qualquer dos recursos disponíveis no hardware da arquitetura a ser utilizada. Abrange deste modo não apenas as linguagens a nível de Macro-Instruções de Máquina (como as do tipo Assembler) como também as fundamentadas a nível de Micro-Programa.

Linguagem de Máquina Normalmente associada a uma seqüência de "códigos" (de Máquina) resultantes da "tradução" de programas (ou trechos de instruções) escritos em alguma linguagem de nível superior, a serem executados pelo *hardware*. Corresponde sempre a um padrão de codificação especificamente desenvolvido para explorar plenamente os recursos de cada arquitetura em particular. Em um nível ainda mais baixo que o da linguagem de Máquina encontra-se normalmente o de Micro-Programação, utilizado para se acessar diretamente todos os dispositivos lógicos, de controle ou aritméticos disponíveis no processador central ou periféricos, e no qual se "interpretam" dinâmicamente as Macro-Instruções de Máquina.

Linha Antecessora O mesmo que *Linha Predecessora.*

Linha Anterior O mesmo que *Linha Contribuinte.*

Linha Base O mesmo que *Linha Corrente.*

Linha Básica O mesmo que *Linha Corrente.*

Linhas Características Sub-Conjunto particular de Linhas Contribuintes de uma matriz de fatores, suficientes para a especificação completa do Padrão Estrutural de colunas de Todos os elementos não nulos Contribuintes numa dada Etapa básica.

Linha Contribuinte Linha Não Básica da Matriz de Fatores, contendo elementos de Contribuição a serem necessariamente subtraídos da Linha Base associada à Etapa Corrente do processo de eliminação.

Linha Corrente Linha correspondente ao índice de uma Etapa Básica do processo de eliminação, a ser operada (sofrendo Contribuições) de modo a se obterem seus Fatores Associados.

Linhas Predecessoras Linhas correspondentes a ancestrais diretos do nó associado à linha originalmente especificada, na estrutura da Árvore de Eliminação.

Linha Sucessora Linha correspondente ao descendente direto do nó associado à linha originalmente especificada, na estrutura da Árvore de Eliminação.

Lista de Códigos Lista de informações na forma de Códigos do tipo Binário, por Envelopes ou Supernodais, obtidos numa fase preliminar de Codificação Simbólica a partir das características estruturais da matriz de fatores resultante (a serem decodificados durante posteriores fases numéricas dedicadas a fatoração de matrizes com mesma estrutura).

Lista de Endereços Lista de informações do tipo Simbólico, contendo explicitamente para cada etapa de fases numéricas de eliminação, os endereços de memória dos fatores por se gerar, a serem acessados ao se aplicar cada uma das contribuições (de fatores não nulos anteriores) sobre seus respectivos Elementos Associados.

Loop Trecho repetitivo de código, normalmente utilizado em processos de acumulação, ou no acesso a componentes de um vetor. Fundamentalmente consiste num trecho de código re-executado um determinado número de vezes, ditado por uma variável indexadora de controle (incrementada a cada "iteração") e por valores limitantes indicadores do termino da execução. Em FORTRAN dispõem-se da cons-

trução DO, especificamente voltada para a implementação de *loops*, ao passo que em linguagens como C/C^{++} ou Pascal, dispõem-se da construção *for*. *Loops* podem ser também "simulados" mediante trechos de programa contendo um ponto inicial de retorno, testes condicionais, instruções a serem repetidas, e desvios incondicionais ao ponto inicial até que condição de teste seja satisfeita. *Loops* implementados desta forma normalmente não são eficientemente tratados ou reconhecidos, até mesmo pelos melhores compiladores disponíveis, aconselhando-se o uso controlado desta técnica, apenas em programas escritos diretamente em Assembler.

Loop-Free Relativo a trechos de um programa "livres de *loops*", onde não se efetuam desvios ou operações de indexação de qualquer espécie. Originalmente este esquema de processamento foi proposto por Fred G. Gustavson ao final da década de 60, tendo como objetivo a implementação eficiente de programas voltados para a resolução dedicada de sistemas lineares esparsos.

Macro-Instruções Conjunto de instruções de Baixo Nível, disponíveis nas linguagens Assembler ou de Máquina. Hierarquicamente superiores em recursos e grau de elaboração se comparadas com Micro-Instruções, situando-se no entanto abaixo das "construções", "comandos" ou "instruções" das linguagens de mais Alto Nível.

Mapeamento Sub-divisão de tarefas e estruturas de dados entre múltiplos processadores em arquiteturas paralelas, de modo a se melhorar alguma função de mérito como o balanceamento da carga alocada a cada processador.

Matriz de Fatores Exceto quando explicitamente indicado, referente apenas a matriz triangular superior com diagonal unitária, obtida ao final do processo de Eliminação de Variáveis (no qual a matriz do sistema original é fatorada num produto de matrizes da forma triangular e diagonal).

Matriz Original Matriz especificada na formulação "original" do sistema linear a ser solucionado.

Matriz Resultante O mesmo que *Matriz de Fatores.*

Meia Palavra Especifica particularmente o conteúdo armazenado em Metade do número de bytes componentes da Palavra de Dados de uma dada arquitetura. Na quase totalidade das máquinas atuais, Meia Palavra corresponde a 2 bytes = 16 bits de informação.

Memória Unidade computacional de armazenamento, normalmente identificada com a *Memória Principal* (fisicamente acessível por um processador) existindo no entanto uma extensa gama de níveis auxiliares disponíveis, percorrendo desde as memórias de acesso mais rápido como as do tipo Cache, até as de maior capacidade e acesso mais lento como as do tipo secundário.

Memória Central O mesmo que *Memória Principal.*

Memória Compartilhada Memória do tipo Principal em arquiteturas Multiprocessadas, na qual apenas uma única memória (Global) pode ter os acessos "compartilhados" por todos os processadores.

Memória Distribuída Memória do tipo Principal em arquiteturas Multiprocessadas, disponível apenas (Localmente) a cada processador, com acesso isolado dos demais, podendo ser encarada como um recurso "distribuído" por entre os distintos componentes da estrutura computacional.

Memória Global O mesmo que *Memória Compartilhada.*

Memória Local O mesmo que *Memória Distribuída.*

Memória Principal Unidade "principal" (ou central) de armazenamento computacional de dados, diretamente acessível a partir de Unidades Centrais de Processamento mediante mecanismos de conexões (barramentos) especialmente dedicados. Não se constitui no entanto no tipo de unidade de mais rápido acesso a dados, situando-se aquém de memórias mais velozes como as do tipo Cache, e substancialmente aquém da vazão de acesso à registradores (localizados internamente nas unidades de processamento). Constitui-se portanto no "elo" entre as memórias do tipo secundário, e os demais níveis hierárquicos de armazenamento mais próximos a CPU.

Memória Secundária Meio auxiliar e mais lento de armazenamento, normalmente com capacidade maior que o de memórias do tipo Principal. Alguns exemplos de memórias secundárias são tipicamente os Hard Disks, Floppy Disks, Fitas e mais recentemente as novas tecnologias opto-magnéticas.

Memória Virtual Recurso disponível em alguns ambientes computacionais, e que viabiliza a utilização de recursos secundários de armazenamento, de modo a se estender fisicamente a quantidade de memória transparentemente acessível por um programa. O acesso alternado ou randômico a dados localizados em posições virtualmente distantes entre si, normalmente induz a uma degradação significativa do desempenho, pois a parcela de dados não disponíveis fisicamente na memória Principal acaba tendo de ser obtida do meio secundário (com tempos de acesso consideravelmente mais longos que os da memória central).

Merge Processo de "concatenação" (ordenada ou desordenada) dos componentes de várias estruturas, em uma só estrutura final "agregada" resultante.

Método O mesmo que *Algoritmo.*

Metodologia O mesmo que *Algoritmo.*

Micro-Código Seqüência de Micro-Instruções a serem executadas pelos mecanismos de *hardware* de um processador. Constitui-se no Mais Baixo Nível de programação possível nas arquiteturas que se utilizam desta camada de interface entre os circuitos de *hardware* propriamente dito, e as Macro-Instruções de Máquina.

Micro-Instruções Instruções de Mais Baixo Nível possível, atuando diretamente sobre os recursos do *hardware* de cada processador (a nível do controle de suas portas e unidades lógico-aritméticas entre outras).

Micro-Programa O mesmo que *Micro-Código.*

Micro-Programação Programação e controle de recursos de hardware ao nível de

Micro-Instruções.

Multiplexação Técnica empregada de modo a se extender o número de casos passíveis de codificação, utilizando-se pares (ou n'uplas) de códigos, de modo a se especificar todo o processamento dedicado a ser decodificado em um número sucessivos de desvios múltiplos encadeados. Na prática, pode-se adotar uma tabela de remapamentos de desvios, ou se computar o código resultante, a partir de operações lógicas apropriadas sobre os dados originalmente fornecidos.

Nó Ancestral O mesmo que *Ancestral.*

Nó Antecessor O mesmo que *Nó Predecessor.*

Nó Descendente O mesmo que *Descendente.*

Nó Predecessor O mesmo que *Ancestral Direto.*

Nó Sucessor O mesmo que *Descendente Direto.*

Ordenação Reagrupamento das componentes de uma dada estrutura de dados, segundo um critério (crescente ou decrescente) associado a alguma informação extraída dos dados originalmente armazenados, de modo a que a estrutura resultante venha a conter seus dados na nova seqüência ditada pelo critério de ordenação especificado.

Ordenação Topológica Qualquer reagrupamento dos nós de uma Árvore, de modo a preservar a propriedade de que os índices associados aos Descendentes de cada um de seus nós, sejam superiores aos de seus Ancestrais.

Ordenamento O mesmo que *Ordenamento para Minimização de Fill-In's.*

Ordenamento para Minimização de Fill-In's Fase preliminar do processo de solução de sistemas esparsos por métodos diretos, na qual as linhas e colunas da matriz original são permutadas segundo algum critério visando a redução do número de elementos de *fill-in* da matriz de fatores resultante. Em alguns casos, é possível incorporar-se critérios secundários ao ordenamento, como o aumento da independência entre as equações do sistema, de especial interesse nas implementações em arquiteturas do tipo paralelo.

Ordenamento Ótimo O mesmo que *Ordenamento.*

Otimização de Código Conjunto de regras e técnicas passíveis de serem empregadas tanto a nível automatizado durante fases de compilação e geração de código de Máquina, como em alguns casos "manualmente" por um programador experiente, visando aumentar-se a eficiência computacional da implementação de um programa. No caso de arquiteturas do tipo Vetorial, parte destas técnicas voltadas especificamente para o tratamento de entidades do tipo matriz ou vetor, recebem a designação particular de Vetorização.

Overhead Dispêndio adicional de recursos computacionais, advindo de ineficiências impostas pela estratégia de implementação adotada para a solução de um dado problema.

Page Fault Penalização em termos de uma degradação a nível de tempo de execução,

imposta pela necessidade de dados não encontrados nas páginas correntemente disponíveis na memória principal, induzindo-se forçosamente a uma espera pelo carregamento das informações a partir de meios secundários de armazenamento.

Paginação Recurso disponível em algumas arquiteturas computacionais (como a VAX), onde "páginas" de dados são transferidas de meios secundários de armazenamento para a memória principal na medida em que se mostram necessárias. Este esquema é um modo alternativo de extensão da memória disponível durante a execução de um programa, análogamente aos mecanismos de memória virtual. No caso específico dos mecanismos de paginação, a técnica foi originalmente desenvolvida com o objetivo de permitir o compartilhamento da memória entre vários processos ou programas sendo parcialmente executados de forma concorrente, em sistemas operacionais multi-tarefa. Um dos principais fatores determinantes de um bom desempenho em sistemas paginados, é a "política de descarte" com a qual informações em páginas correntemente não acessadas, são liberadas para o carregamento de novos valores a medida em que se mostrem necessários.

Palavra Uma das unidades básicas de armazenamento computacional de informações, correspondendo a um determinado múltiplo de unidades menores como bits ou bytes. Na maioria das arquiteturas atuais, uma palavra corresponde a 4 bytes (32 bits), embora em algumas das arquiteturas mais avançadas uma palavra corresponda a 8 bytes (64 bits). Por construção (e definição), uma "palavra" constitui a quantidade de informação mais rapidamente acessível e endereçável de cada arquitetura. Para o acesso a componentes menores ou maiores, como Meia Palavra ou Dupla Palavra, alguns ciclos adicionais de máquina são normalmente necessários de modo a no primeiro caso se descartar parte da informação não relevante, ou se buscar a informação complementar armazenada na palavra seguinte.

Paralelismo Possibilidade de exploração simultânea de múltiplos aparatos computacionais visando a solução de um mesmo problema. Normalmente por este intermédio, viabiliza-se uma redução a nível de menores tempos globais de execução se comparados com os que seriam obtidos em um único aparato isoladamente. Pode-se lançar mão de paralelismo não apenas para se reduzir tempos de execução, como também para se subdividir problemas de porte elevado, incapazes de serem plenamente tratados ao nível da disponibilidade de memória de um só aparato.

Paralelização Transformação manual ou automática de um programa ou algoritmo desenvolvido originalmente para arquiteturas seqüenciais, de modo a transporta-lo para algum ambiente computacional paralelo.

Paralelizável Relativo a tarefas ou programas que possam ser sub-divididos entre processadores e executados de forma simultânea em um ambiente computacional paralelo.

Paralelo Classe de arquiteturas computacionais onde se dispõe de múltiplos processadores interconectados de algum modo, com a capacidade de executar simultaneamente tarefas visando a solução de um mesmo problema.

PASCAL Linguagem computacional baseada no Algol e desenvolvida originalmente com o objetivo de ser fortemente estruturada e de fácil aprendizado para inician-

tes em computação. Ganhou relativa popularidade em função de sua ampla disseminação no meio acadêmico, especialmente em micro-computadores pessoais, com o advento de uma de suas extensões mais conhecidas: o Turbo-Pascal. Outras implementações de sucesso durante o "período áureo" incluem o UCSD Pascal, desenvolvido juntamente com um sistema operacional (P-System) na universidade de San Diego. Caiu em desuso nos dias atuais, em face a introdução de extensões mais elaboradas como Modula 2/3 ou de linguagens Orientado a Objeto como C^{++}.

Pipeline Mecanismo de aumento da concorrência pela sub-divisão da execução computacional de tarefas em uma série de passos, permitindo que a cada passo se inicie a execução de novas tarefas (sem que seja necessário aguardar-se a conclusão da tarefa inicial).

Pivô Elemento chave escolhido num processo de Pivoteamento (a ser normalmente permutado até a posição diagonal associada à etapa corrente).

Pivoteamento Designação atribuída ao Intercâmbio entre linhas e/ou colunas durante alguma etapa do processo de eliminação de variáveis. Alguns dos critérios utilizados como base para o estabelecimento das linhas ou colunas a serem permutadas podem advir de características puramente Estruturais da matriz de fatores, ou de natureza meramente Numérica. Na maioria das vezes quando não se faz menção explícita ao tipo de Pivoteamento, assume-se alguma forma de Pivoteamento Numérico, como o do tipo Parcial. O Pivoteamento Estrutural por sua vez, tende a ser normalmente denominado "Ordenamento" na literatura, razão pela qual optou-se por adotar tal nomenclatura no corrente trabalho.

Pivoteamento Estrutural Pivoteamento entre linhas e/ou colunas de uma matriz, com base apenas em suas características estruturais (normalmente visando um Ordenamento para Minimização dos *Fill-In's* introduzidos no processo de sua fatoração).

Pivoteamento Numérico Pivoteamento tomando como base aspectos de natureza puramente Numérica, na escolha do elemento Pivô a cada etapa (visando normalmente um aumento da Estabilidade Numérica do processo de fatoração matricial). Nos casos mais comuns de Pivoteamento Numérico, o elemento Pivô escolhido, tende a ser o de maior valor absoluto, dentre um grupo selecionado de "candidatos" (particularmente especificado por cada tipo de critério).

Pivoteamento Parcial Forma particular de Pivoteamento Numérico, em que o elemento Pivô é escolhido tomando-se por base apenas candidatos situados numa mesma coluna, abaixo da posição diagonal corrente a cada etapa.

Pivoteamento Total Forma mais elaborada de Pivoteamento Numérico, em que o elemento Pivô é escolhido dentre quaisquer dos elementos da sub-matriz definida a partir da posição diagonal corrente (até a posição diagonal associada a dimensão do problema).

Ponto Flutuante Forma de representação computacional, voltada para a operação e armazenamento (aproximado) de grandezas do tipo real. Para cada tipo de precisão (simples ou dupla), é alocado um determinado número de bytes para a representação do sinal aritmético juntamente com a mantissa (quase sempre normalizada), sendo

o restante utilizado para a representação do expoente (expresso em alguma base normalmente múltipla de 2).

Porção Determinada "região" (retangular) de uma matriz, completamente especificada pela localização de um elemento inicial, e pelas dimensões de uma sub-matriz definida a partir deste ponto.

Porção Densa Maior porção quadrada, centrada sob a diagonal de uma Matriz de Fatores, (correspondente as etapas finais do processo de eliminação) a partir da qual a sub-matriz associada é completamente densa.

Porção Diagonal Maior porção quadrada, centrada sob a diagonal de uma Matriz de Fatores, (correspondente as etapas iniciais do processo de eliminação) até onde a sub-matriz associada é completamente esparsa.

Porção Quase Densa Porção quadrada, centrada sob a diagonal de uma Matriz de Fatores, (correspondente as etapas finais do processo de eliminação) a partir da qual a sub-matriz associada assume características "quase densas".

Precisão Dupla Tipo de dados destinado ao armazenamento de números em ponto flutuante, com precisão mais elevada (ou estendida) que a usual. Nas linguagens que contam com este recurso, normalmente se utilizam 8 bytes para o armazenamento de grandezas em Precisão Dupla. Em muitas arquiteturas que dispõem de processadores de ponto flutuante "dedicados" (como co-processadores por exemplo), é possível encontrar-se precisões internas de trabalho superiores a 8 bytes como em processadores (seguindo o padrão IEEE) como os da linha Intel, capazes de operar e armazenar temporariamente em seus registradores, dados em ponto flutuante de 10 bytes.

Precisão Simples Correspondente ao tipo de dados de menor precisão utilizado para o armazenamento de grandezas em ponto flutuante. Em praticamente todas as linguagens, se utilizam 4 bytes para o armazenamento dos dados neste tipo de precisão.

Predecessor O mesmo que *Ancestral Direto.*

Pré-Processamento Estrutural O mesmo que *Pré-Processamento Simbólico.*

Pré-Processamento Simbólico Conjunto de técnicas como as de Fatoração ou Codificação Simbólica, em que características Estruturais da Matriz de Fatores associada a um sistema linear esparso são previamente obtidas, com o objetivo de tornar mais eficiente posteriores fases de Fatoração Numérica, aplicadas a matrizes com Mesma Estrutura que a Originalmente analisada.

Previsão de Desvio Mecanismo disponível em algumas das arquiteturas mais avançadas, permitindo que antes da execução de testes do tipo condicional se possa obter uma "estimativa" indicadora do ponto para onde tenderá a ser desviada a execução de um programa. Vide também *Desvio Especulativo.*

Procedimento O mesmo que *Algoritmo.*

Processador O mesmo que *Unidade Central de Processamento.*

Processador Central O mesmo que *Unidade Central de Processamento.*

Processo Referente ao processo de Eliminação de Variáveis, no qual a Matriz Original de um sistema linear é Fatorada numa Forma Triangular.

Processo de Cancelamento O mesmo que *Processo de Eliminação.*

Processo de Eliminação O mesmo que *Fatoração.*

Programa Seqüência de instruções a serem aplicadas sobre estruturas de dados, em alguma linguagem computacional visando a implementação de um algoritmo.

Programa Fonte Programa armazenado em alguma linguagem computacional, a ser fornecido como entrada para uma fase de compilação (ou interpretação) no caso de linguagens de Alto Nível (ou de montagem, no caso de linguagens de mais Baixo Nível como Assembler).

Recorrente Que "retorna" a ocorrer (com alguma freqüência). Em programas pode-se lançar mão de trechos Recorrentes mediante o uso de "laços de repetição" como *loops* ou desvios condicionais para pontos de retorno comuns (ou aninhados dentro de uma mesma porção do código). A noção de Recorrência implica em que para se poder chegar a um determinado ponto (ou resultado parcial) de um processamento, se tenha obrigatoriamente efetuado todo o processamento relativo a suas instâncias anteriores.

Recursivo Forma de processamento na qual se obtém a solução para um dado problema, invocando-se a solução para novas instâncias de sub-problemas (a partir do original) e que por sua vez podem vir a invocar a solução de sub-problemas menores (a partir de si), até se chegar a sub-casos em que soluções independentes de si ou de outros sub-problemas possam ser obtidas.

Reduzida Estrutura formada apenas pelas linhas e colunas efetivamente contribuintes a cada etapa básica.

Registrador Unidade de armazenamento temporário e de mais rápido acesso a dados, localizada dentro de Unidades de Processamento Central ou de Ponto Flutuante. Os registradores podem ser construidos e destinados a funções específicas como o armazenamento de dados inteiros, endereços ou valores em ponto flutuante, sendo a quantidade disponível de cada tipo, uma característica de cada arquitetura computacional. A tendência das novas arquiteturas superescalares é possuir um número suficientemente elevado (em torno de 64 ou mais) de registradores, especialmente os de ponto flutuante, em face ao elevado desempenho das unidades funcionais voltadas ao processamento numérico, e que demandam uma plena disponibilidade dos dados a serem processados, sob o risco de serem obrigadas a interromper cadeias de pipeline caso um determinado valor tenha de ser buscado da memória principal.

Reordenamento Ancestral Caso particular de Reordenamento Dinâmico de Contribuições, em que se toma como base para esquemas de Codificação Simbólica por Envelope, grupos de linhas pertencentes a uma mesma Família na estrutura da Árvore de Eliminação. Este processo, é implementado mediante a subdivisão do conjunto global de linhas contribuintes numa mesma etapa, reagrupando-o por subconjuntos

associados a cada uma das Famílias isoladamente envolvidas (e identificadas).

Reordenamento Dinâmico de Contribuições Esquema empregado em procedimentos de fatoração baseados em Codificações do tipo Simbólico, em que se reagrupa o conjunto global de índices das linhas contribuintes a cada etapa, subdividindo-o por subconjuntos particularmente adaptados a cada tipo de Codificação adotada. Este nível de reagrupamento da seqüência como as linhas contribuintes passarão a ser acessadas e operadas sobre a linha ou coluna base sendo gerada, embora determinado estáticamente durante a fase de Codificação, só é de fato implementado "dinâmicamente" por ocasião da execução da fase numérica do processo de eliminaçõcs.

Representação O mesmo que *Estrutura de Armazenamento.*

Restrita Lista de endereços, contendo a especificação de intervalos de endereços contíguos na estrutura base (desde **PJ** até **PJF**), a receberem contribuições.

RISC (Reduced Instruction Set Computer) Arquitetura Computacional baseada em processadores com um conjunto "simplificado" de Macro-Instruções, normalmente capazes de serem executadas em apenas um único ciclo de máquina.

RISC Superescalar Arquitetura Computacional do tipo RISC, em que a meta de execução de Macro-Instruções em apenas um ciclo de máquina é relaxada, de modo a permitir a inclusão de instruções um pouco mais elaboradas como as de acumulação parcial de produtos escalares (passíveis de serem "encadeadas" em *pipelines*, e executadas concorrentemente em múltiplas unidades funcionais dedicadas a cada uma das operações de ponto flutuante envolvidas).

RLE (Run Length Encoded) Técnica de compactação, baseada na especificação da freqüência de ocorrências repetitivas de cada código, ao longo de toda a seqüência dos dados de entrada.

Segmentada Estrutura de armazenamento na qual Grupos de elementos Não Nulos das linhas de uma matriz são armazenados "blocadamente" (por cada linha) em posições Contíguas de memória, com o posicionamento de blocos consecutivos Não necessariamente contíguo entre si.

Speedup Para um dado número de processadores numa dada arquitetura paralela, expressa a razão entre o tempo gasto com a execução (em um único processador) da implementação do melhor algoritmo seqüencial conhecido, dividido pelo tempo gasto na execução com o número especificado de processadores em paralelo.

Seqüencial O mesmo que *Contíguo.*

Seqüencial Segmentada O mesmo que *Segmentada.*

Simbólica Informação auxiliar extraída da Estrutura de elementos Não Nulos da Matriz de Fatores, com objetivo de viabilizar implementações mais eficientes da Fase Numérica do processo de eliminação.

Sistema Irredutível Sistema linear, cuja matriz de coeficientes possua ao menos um elemento não nulo excluindo-se a diagonal, em cada uma de suas linhas e colunas.

Sistema Original Sistema de equações lineares contendo a matriz da formulação original a ser solucionada.

Software Relativo a qualquer produto computacionalmente desenvolvido e utilizável na forma programas, compiladores, interpretadores, pacotes, sistemas operacionais, utilitários, bibliotecas de subrotinas ou bases de dados (requeridos para a execução de um programa).

Sort Ordenação dos elementos de um vetor, por um critério ascendente ou descendente segundo os valores numéricos de suas componentes armazenadas.

Subrotina Trecho particular e dedicado dentro de um programa (ou biblioteca), e que devido a suas particularidades ou freqüência de utilização, mostra-se conveniente isolar-se na forma de um sub-procedimento, a ter sua execução invocada a partir de um programa mestre (ou principal).

Sucessor O mesmo que *Descendente Direto.*

Substituições Fases terminais no processo de solução de um sistema linear esparso, após ter-se levado a matriz original à forma Triangular, englobando as fases de Substituição *Forward*, *Diagonal* e (de Retro-Substituição) *Backward.*

Supercomputador Designação atribuída as arquiteturas computacionais de maior desempenho nos dias atuais, correspondendo na maioria dos casos à classes Vetoriais ou Paralelas, ou a classes "híbridas" do tipo Paralelo, onde cada unidade de processamento é também dotada de recursos Vetoriais (e muitas das vezes capaz de processar trechos escalares de código de forma Superescalar).

Superescalar Classe de arquiteturas escalares, na qual múltiplas unidades funcionais se encontram disponíveis, permitindo execuções (parcialmente) concorrentes de operações mais elaboradas (como as de produto escalar). Em face e este aumento de desempenho constitui-se numa das mais eficientes classes de arquiteturas "convencionais" em utilização, sendo encontrada particularmente nas famílias RISC mais modernas, (ou em unidades de processamento escalar "dedicadas" de algumas arquiteturas Vetoriais).

Supernode Conjunto de linhas consecutivas de uma matriz de fatores (associadas a uma seqüência de nós descendentes diretos na Árvore de Eliminação) com o mesmo padrão estrutural de elementos não nulos (a partir de suas posições diagonais).

Supernode Relaxado Seqüência ordenada de linhas Contribuintes de uma matriz de fatores, cujo padrão estrutural de elementos não nulos (a partir das posições diagonais associadas) engloba necessariamente os padrões estruturais (dos trechos correspondentes) das demais linhas anteriores, pertencentes ao mesmo conjunto.

Tempo de Acesso Tempo durante a execução de um programa, no qual se buscam ou retornam dados à algum tipo de memória. Em muitos casos (especialmente nas novas arquiteturas), algumas formas de acesso a dados da memória podem vir a se constituir em "gargalos" na eficiência global de um programa, pois em uma parcela significativa de situações, acaba-se por forçar a ociosidade de alguma unidade de processamento, à espera da conclusão da operação de acesso.

Tempo de CPU Grandeza que expressa o volume de tempo dispendido pela Unidade Central de Processamento na execução de um programa. Nas arquiteturas escalares, normalmente globaliza o dispêndio total de "recursos de tempo" gasto para se alcançar a solução de um dado problema.

Triangularização O mesmo que *Fatoração Triangular.*

Unidade Central de Processamento Principal aparato do *hardware* de cada arquitetura computacional, dotado de unidades especiais e dedicadas, voltadas para a decodificação e processamento das instruções de máquina e de onde se controlam todos os demais dispositivos a ela subordinados. Normalmente contém além das unidades de controle e lógico-aritméticas, uma parcela de circuitos especialmente dedicados ao armazenamento interno e temporário de dados a serem operados, denominados Registradores.

Unidade Funcional Unidade computacional fisicamente incorporada ao *hardware* de um processador, voltada especificamente para a execução de um determinado tipo de operação de modo independente das demais unidades de processamento. Quase sempre disponível em um grande número e variedade de funções (do tipo aritmético ou de endereçamento), nas arquiteturas mais avançadas, como as do tipo RISC Superescalar.

Unidade de Processamento Central O mesmo que *Unidade Central de Processamento.*

Unidade de Processamento em Ponto Flutuante Unidade computacional especial, disponível no *hardware* de algumas arquiteturas, com a finalidade de elevar seu desempenho no processamento de dados de Ponto Flutuante. Em alguns casos este aumento é conseguido mediante a viabilização de cálculos concorrentes e executados independentemente do controle da CPU, como no caso das Unidades Co-Processadoras de Ponto Flutuante. Outra forma adotada para se alcançar uma elevação no desempenho, é lançar mão de circuitos de *hardware* "dedicados" ao processamento numérico (especialmente o de funções transcendentais) eliminando-se assim os *overheads* presentes nas implementações convencionais, baseadas no uso de Micro-Programação para o controle e execução das operações aritméticas.

Vetor de Contribuições Vetor de Trabalho Expandido (originalmente "zerado") voltado para a acumulação das contribuições a serem sofridas por cada um dos componentes de uma dada linha (ou coluna) base a ser gerada em cada etapa do processo de eliminação.

Vetor de Trabalho Vetor (ou área de memória) a ser utilizado temporariamente de forma auxiliar na implementação de algum procedimento computacional.

Vetor de Trabalho Expandido Vetor de Trabalho de dimensão n, onde originalmente se "descompacta" a representação de elementos não nulos de uma dada linha base em função de suas colunas associadas, para a seguir se operar sobre suas componentes não nulas de forma "expandida" a medida em que se efetuam as contribuições sobre seus elementos associados, para finalmente se re-compactar e armazenar de volta os valores atualizados correspondentes aos fatores não nulos gerados.

Vetorial Tipo de arquitetura computacional de elevado desempenho, especificamente no processamento numérico de dados da forma de matrizes ou vetores, contando normalmente com dispositivos de *hardware* especificamente projetados para este tipo de processamento (com a capacidade de operar concorrentemente sobre um grupo de componentes de cada vetor).

Vetorização Processo automatizado (ao nível de compilação) capaz de detectar e explorar plenamente a independência de cálculos sobre dados a serem operados na forma de matrizes ou vetores, de modo a viabilizar o melhor desempenho computacional possível especificamente para cada família ou classe de arquitetura Vetorial. Normalmente para cada fabricante ou arquitetura, um conjunto de diretivas de compilação se encontra padronizado nas principais linguagens, facilitando-se a customização de programas para cada ambiente em particular.

Referências Bibliográficas

[1] Adler, I., Karmarkar, N., and Resende, M. and Veiga, G. Data structures and programming techniques for the implementation of Karmarkar's algorithm for linear programming. *ORSA J. Computing*, 1(2):84–106, 1989.

[2] Alvarado, F. L. and Tinney, W. F. and Enns, M. K. Sparse matrix inverse factors. Paper 88-SM-728-8, Univ. of Wisconsin, Madison, 1988. (Apresentado no IEEE Summer Power Meeting, Portland, Publicado na IEEE Trans. Power Systems, PWRS-5:466-473, 1990).

[3] Alvarado, F. L., Yu, D. C., and Betancourt, R. Ordering schemes for partitioned sparse inverses. Technical report, Univ. of Wisconsin, Madison, 1989. (Apresentado no SIAM Symposium on Sparse Matrices, Salishan Lodge, Oregon).

[4] Anderson, E, Bai, Z., Bischof, C., Demmel, J., Dogarra, J. J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. C. *LAPACK "User's Guide"*. SIAM Publications, 1992.

[5] Arantes, R. D. Uma abordagem híbrida via processamento simbólico para a resolução eficiente de sistemas lineares esparsos com estrutura estática especialmente aplicável à algoritmos de pontos interiores para programação linear. Relatório Técnico 102/91, Centro de Pesquisas em Engharia Elétrica (CEPEL), Rio de Janeiro, 1990.

[6] Arantes, R. D. Metodologias simbólicas para a resolução eficiente de sistemas lineares esparsos com estrutura estática. Master's thesis, Depto. de Eng. de Sistemas e Computação, COPPE/UFRJ, 1994.

[7] Ashcraft, C., Grimes, R. G., Lewis, J. G., Peyton, B. W., and Simon, H. D. Progress in sparse matrix methods for large linear systems on vector supercomputers. *Int. J. of Supercomputer Appl.*, 1(4):10–30, 1987.

[8] Ashcraft, C. and Grimes, R. G. The influence of relaxed supernode partitions on the multifrontal method. *ACM Trans. Math. Software*, 15(4):291–309, 1989.

[9] Atkinson, K. E. *An Introduction to Numerical Analysis.* John Wiley & Sons, 1978.

[10] Bank, R. E. and Rose, D. J. On the complexity of sparse Gaussian elimination via bordering. *SIAM J. Sci. and Stat. Comput.*, 11(1):145–160, 1990.

[11] Bank, R. E. and Smith, R. K. General sparse elimination requires no permanent integer storage. *SIAM J. Sci. Stat. Comput.*, 8(4):574–584, 1987.

[12] Barret, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Romine, C., and van der Vorst, H. *Templates for the Solution of Linear Systems : Building Blocks for Iterative Methods.* SIAM Publications, 1993.

[13] Benner, R. and Montry, G. and Weigand, G. Concurrent multifrontal methods : Shared memory, cache and frontwidth issues. *Int. J. of Supercomputer Appl.*, 1(3):26–44, 1987.

[14] Betancourt, R. An efficient heuristic ordering algorithm for partial matrix refactorization. *IEEE Trans. Power Systems*, PWRS-3(3):1181–1187, 1988.

[15] Browne, J., Dongarra, J., Karp, A., and Kennedy, K. and Kuck, D. 1988 Gordon Bell prize. *IEEE Software*, 6:78–85, 1989. (Special Report).

[16] Bunch, J. R. and Parlett, B. N. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

[17] Chan, S. M. and Brandwajn, V. Partial matrix refactorization. *IEEE Trans. Power Systems*, PWRS-1(1):193–200, 1986.

[18] Chu, E. C. H. and George, A. and Ng, E. "User's Guide" for SPARSPAK-A : Waterloo sparse linear equations package. Tech. Report CS-84-36, Univ. Waterloo, Canada, 1984.

[19] de Carvalho, M. L. B. On the minimization of work needed to factor a symmetric positive definite matrix. Manuscript ORC 87-14, Dept. of Industrial Eng. and Oper. Research, Univ. of California, Berkeley, 1987.

[20] Dembart, B. and Erisman, A. M. Hybrid sparse matrix methods. *IEEE Trans. Circuit Theory*, CT-20:641–649, 1973.

[21] Dembo, R. S. Solving box-constrained quadratic programming problems on a vector processor. Tech. Report, Dept. of Computer Science, Univ. of Toronto, Canada, 1987.

[22] Demidovich, B. P. and Maron, I. A. *Computational Mathematics.* Mir Publishers, 1976.

[23] Dongarra, J. J., Duff, I. S., Sorensen, D. C., and van der Vorst, H. A. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM Publications, 1991.

[24] Dongarra, J. J. et al. *LINPACK "User's Guide".* SIAM Publications, 1979.

[25] Dongarra, J. J. and Hinds, A. R. Unrolling loops in FORTRAN. *Software Practice and Experience*, 9:219–229, 1979.

[26] Duff, I. S. On the number of nonzeros added when Gaussian elimination is performed on sparse random matrices. *Math. of Computation*, 28(125):219–230, 1974.

[27] Duff, I. S. A survey of sparse matrix research. *Proc. of the IEEE*, 65(4):500–535, 1977.

[28] Duff, I. S. Full matrix techniques in sparse Gaussian elimination. Tech. Report CSS-114, AERE, Harwell Laboratory, 1981.

[29] Duff, I. S. A sparse future. In Duff, I. S., editor, *Sparse Matrices and Their Uses*, pages 1–29. Academic Press, 1981.

[30] Duff, I. S. MA27 : A set of FORTRAN subroutines for sparse symmetric linear equations. Tech. Report R-10533, HMSO, AERE Harwell Laboratory, 1982.

[31] Duff, I. S. Data structures, algorithms and software for sparse matrices. Tech. Report 84-1846, Harwell Laboratory, 1984.

[32] Duff, I. S., Erisman, A. M., and Reid, J. K. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[33] Duff, I. S., Gould, N., and Lescrenier, M. and Reid, J. K. The multifrontal method in a parallel environment. Tech. Report CSS-211, Computer Science and Systems Division, Harwell Laboratory, Oxon, England, 1987. (Publicado em Advances in Numerical Computation, Cox, M. and Hammarling, S., eds., Oxford University Press, 1990).

[34] Duff, I. S. and Grimes, R. G. and Lewis, J. G. "User's Guide" for the Harwell-Boeing sparse matrix collection (Release I). Tech. Report RAL-92-086, Rutherford Appleton Laboratory, 1992.

[35] Duff, I. S. and Reid, J. K. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9(3):302–325, 1983.

[36] Duff, I. S. and Reid, J. K. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. and. Stat. Comput.*, 5(3):633–641, 1984.

[37] Durand, E. *Solutions Numériques des Équations Algébriques, tome II : Systèmes de Plusieurs Équations.* Masson & Cie, 1972.

[38] Eisenstat, S. C., Gursky, M. C., and Schultz, M. H. and Sherman, A. H. YALE sparse matrix package – I : The symmetric codes. *Int. J. Numer. Meth. in Eng.*, 18:1145–1151, 1982.

[39] Eisenstat, S. C. and Schultz, M. H. and Sherman, A. H. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM J. Sci. and Stat. Comput.*, 2(2):225–237, 1981.

[40] Eskow, E. and Schnabel, R. B. Software for a new modified Cholesky factorization. *ACM Trans. Math. Software*, 17(3):306–312, 1991.

[41] Forsythe, G. E. and Moler, C. B. *Computer Solution of Linear Algebraic Systems.* Prentice-Hall, 1967.

[42] Gay, D. M. Massive memory buys little speed for complete in-core sparse Cholesky factorizations on some scalar computers. *Lin. Alg. Appl.*, 152:291–314, 1991.

[43] Gentleman, W. M. and George, A. Sparse matrix software. In Bunch, J. R. and Rose, D. J., editors, *Sparse Matrix Computations.* Academic Press, New York, 1976.

[44] George, A. and Liu, J. W. H. The design of a user interface for a sparse matrix package. *ACM Trans. Math. Software*, 5(2):139–162, 1979.

[45] George, A. and Liu, J. W. H. An optimal algorithm for symbolic factorization of symmetric matrices. *SIAM J. Comput.*, 9(3):583–593, 1980.

[46] George, A. and Liu, J. W. H. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Software*, 6(3):337–358, 1980.

[47] George, A. and Liu J. W. H. A minimal storage implementation of the minimum degree algorithm. *SIAM J. Numer. Anal.*, 17(2):282–299, 1980.

[48] George, A. and Liu, J. W. H. *Computer Solution of Large Positive Definite Systems.* Prentice-Hall, 1981.

[49] George, A. and Liu, J. W. H. The evolution of the minimmum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[50] George, A. and Rashwan, H. Auxiliary storage methods for solving finite element systems. *SIAM J. Sci. and Stat. Comput.*, 6(4):882–910, 1985.

[51] Gilbert, J. R., Lewis, J. G., and Schreiber, R. Parallel preordering for sparse matrix factorization. (Em Preparação), 1992.

[52] Gilbert, J. R., Ng, E., and Peyton, B. W. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. and Appl.*, 15(4):1075–1091, 1994.

[53] Gill, P. E. and Murray, W. and Wright, M. H. *Practical Optimization.* Academic Press, 1981.

[54] Golub, G. H. and Meurant, G. A. *Résolution Numérique des Grands Systèmes Linéaires.* Eyrolles, Paris, 1983.

[55] Golub, G. H. and Van Loan, C. F. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 2^{nd} edition, 1989.

[56] Golub, G. H. and O'Leary, D. P. Some history of the conjugate gradient and Lanczos algorithms : 1948-1976. *SIAM Review*, 31(1):50–102, 1989.

[57] Gomez, A. and Franquelo, L. G. An efficient ordering algorithm to improve sparse vector methods. *IEEE Trans. Power Systems*, PWRS-3(4):1538–1544, 1988.

[58] Gomez, A. and Franquelo, L. G. Node ordering algorithms for sparse vector method improvement. *IEEE Trans. Power Systems*, PWRS-3(1):73–79, 1988.

[59] Gustavson, F. G. Some basic techniques for solving sparse systems of linear equations. In Rose, D. J. and Willoughby, R. A., editors, *Sparse Matrices and Their Applications*, pages 41–52. Plenum Press, New York, 1972.

[60] Gustavson, F. G., Liniger, W. M., and Willoughby, R. A. Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations. *J. ACM*, 17(1):87–109, 1970.

[61] Hachtel, G. D. Vector and matrix variability type in sparse matrix algorithms. In Rose, D. and Willoughby, R., editors, *Sparse Matrices and their Applications*, pages 53–66. Plenum Press, 1972.

[62] Hachtel, G. D. and Brayton, R. K. and Gustavson, F. G. The sparse tableau approach to network analysis and design. *IEEE Trans. Circuit Theory*, CT-18(1):101–113, 1971.

[63] Heath, M. T. and Ng, E. and Peyton, B. W. Parallel algorithms for sparse linear systems. *SIAM Review*, 33(3):420–460, 1991.

[64] Hopper, M. J. Harwell subroutine library : A catalogue of subroutines. Tech. Report AERE-R-9185, Harwell Laboratory, 1989. (9^{th} Edition).

[65] Hwang, K. and Briggs, F. A. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1985.

[66] Jung, H. W. and Marsten, R. E. and Saltzman, M. J. Numerical factorization methods for interior point algorithms. *ORSA J. Computing*, 6(1), 1994.

[67] Karmarkar, N. A new polinomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[68] Karmarkar, N. and Ramakrishnan, K. Implementation and computational results of the Karmarkar algorithm for linear programming using an iterative method for computing projections. Tech. Report, AT&T, BELL Labs., New Jersey, 1988.

[69] Liu, J. W. H. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Trans. Math. Software*, 12(2):127–148, 1986.

[70] Liu, J. W. H. On the storage required in the out-of-core multifrontal method for sparse factorization. *ACM Trans. Math. Software*, 12(3):249–264, 1986.

[71] Liu, J. W. H. An adaptive general sparse out-of-core Cholesky factorization scheme. *SIAM J. Sci. and Stat. Comput.*, 8(4):585–599, 1987.

[72] Liu, J. W. H. A note on sparse factorization in a paging environment. *SIAM J. Sci. and Stat. Comput.*, 8(6):1085–1888, 1987.

[73] Liu, J. W. H. Reordering sparse matrices for parallel elimination. *Parallel Comput.*, 11:73–91, 1989.

[74] Liu, J. W. H. The role of elimination trees in sparse factorizations. *SIAM J. Matrix Anal. and Appl.*, 11(1):134–172, 1990.

[75] Liu, J. W. H. A generalized envelope method for sparse factorization by rows. *ACM Trans. Math. Software*, 17(1):112–129, 1991.

[76] Liu, J. W. H. The multifrontal method for sparse matrix solution : Theory and practice. *SIAM Review*, 34(1):82–109, 1992.

[77] Liu, J. W. H., Ng, E., and Peyton, B. W. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. and Appl.*, 14(1):242–252, 1993.

[78] Lustig, I. J. The influence of computer language on computational comparisons : An example from network optimization. *ORSA J. Computing*, 2(2):152–161, 1990.

[79] Markowitz, H. M. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.

[80] Marsten, R. E. *"User's Manual" for the Research Version of OB1.* School of Ind. and Systems Eng., Georgia Institute of Technology, Atlanta, 1990.

[81] Morozowski Filho, M. *Matrizes Esparsas em Redes de Potência.* LTC / FEESC, Rio de Janeiro, 1981.

[82] Ng, E. and Peyton, B. W. Block sparse cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. and Stat. Comput.*, 14:1034–1056, 1993.

[83] Ortega, J. M. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[84] Ortega, J. M. and Rheinboldt, W. C. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, 1970.

[85] Paige, C. C. and Saunders, M. A. LSQR : An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8(1):43–71, 1982.

[86] Peters, F. J. Parallel pivoting algorithms for sparse symmetric matrices. *Parallel Computing*, 1:99–110, 1984.

[87] Pissanetzky, S. *Sparse Matrix Technology.* Academic Press, Inc., 1984.

[88] Rose, D. J. and Tarjan, R. E. and Lueker, G. Algorithm aspects of vertex elimination on graphs. *SIAM J. Computing*, 5(2):266–283, 1976.

[89] Rothberg, E. and Gupta, A. Techniques for improving the performance of sparse Cholesky factorization on multiprocessor workstations. *Proc. Supercomputing*, 90:232–243, 1990.

[90] Rothberg, E. and Gupta, A. Efficient sparse matrix factorization on high-performance workstations : Exploiting the memory hierarchy. *ACM Trans. Math. Software*, 17:313–334, 1991.

[91] Rothberg, E. and Gupta, A. Fast sparse matrix factorization on modern workstations. Tech. Report STAN-CS-89-1286, Stanford Univ., 1989.

[92] Schnabel, R. B. and Eskow, E. A new modified Cholesky factorization. *SIAM J. Sci. Stat. Comput.*, 11(6):1136–1158, 1990.

[93] Schreiber, R. A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Software*, 8(3):256–276, 1982.

[94] Sherman, A. H. On the efficient solution of sparse systems of linear and non-linear equations. Tech. Report 46, Dept. of Computer Science, Yale Univ., 1975.

[95] Stewart, G. W. *Introduction to Matrix Computations.* Academic Press, 1973.

[96] Tarjan, R. E. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[97] Tarjan, R. E. A class of algorithms which require nonlinear time to mantain disjoint sets. *J. of Computer and System Sciences*, 18:110–127, 1979.

[98] Tinney, W. F. and Walker, J. Direct solution of sparse network equations by optimally ordered triangular factorization. *Proc. of the IEEE*, 55:1801–1809, 1967.

[99] Traub, J. F. *Iterative Methods for the Solution of Equations.* Prentice-Hall, 1964.

[100] Varga, R. S. *Matrix Iterative Analysis.* Prentice-Hall, 1962.

[101] Yannakakis, M. Computing the minimum fill-in is NP-Complete. *SIAM J. Alg. and Disc. Meth.*, 2(1):77–79, 1981.

[102] Young, D. M. *Iterative Solution of Large Linear Systems.* Academic Press, 1971.

[103] Zenios, S. A. Parallel computing : The introduction of novel computer architectures greatly alters technical landscape. *OR/MS Today*, pages 44–49, August 1992.

[104] Zenios, S. A. and Mulvey, J. M. Vectorization and multitasking of nonlinear network programming algorithms. *Math. Programming*, 42:449–470, 1988.