# DESENVOLVIMENTO FORMAL
# DE
# ALGORITMOS PARALELOS SISTÓLICOS

*Kunio Okuda*

DISSERTAÇÃO APRESENTADA

AO

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

DA

UNIVERSIDADE DE SÃO PAULO

PARA OBTENÇÃO DO GRAU DE MESTRE

EM

MATEMÁTICA APLICADA

ÁREA DE CONCENTRAÇÃO: *Ciência da Computação*

ORIDENTADOR: Prof. Dr. *Siang Wun Song*

- São Paulo, julho de 1989 -

À minha mãe

À M. Alícia

AGRADECIMENTOS

Ao Prof. Dr. Siang Wun Song, pela orientação deste trabalho, pela atenção e paciência que me possibilitaram concluir esta tese, pelos valiosos conselhos e por ter me propiciado conhecer um assunto tão fascinante como o sistema sistólico.

Ao Prof. Dr. Arnaldo Mandel, como orientador de programa quando do meu ingresso no MAP.

Ao Prof. Dr. Francisco Cesar Polcino Milies, como meu orientador em minha época de álgebra, e que continua me dando valiosos conselhos, pela nossa amizade.

Aos colegas do IME, e em especial, aos colegas do MAC.

Aos funcionários da Xerox e da gráfica.

À Beth, cuja companhia me animou e me ajudou a superar a preguiça.
Ao Marcelo, pela amizade desde o 1º ano de graduação.
À Líria pelo apoio moral.
À Raquel, Secretária do MAP, pela atenção.
Ao Ueda, pelo estímulo indireto.
À Antonia, minha simpática vizinha, pelo excelente trabalho de datilografia.

---

O autor é integrante do Projeto de Pesquisa em Computação Paralelo do IME/USP, que conta com apoio do Programa BID/USP, subprograma de Recursos Humanos da Área Científica e da Fundação de Amparo à Pesquisa do Estado de São Paulo - (Auxílio para material permanente - processo número 88/3095-9).

O autor também foi bolsista de mestrado da Fundação de Amparo à Pesquisa do Estado de São Paulo (processo nº 11-80/1500-1).

RESUMO

Neste trabalho, desenvolvemos um método formal para a transformação de um algoritmo sequencial, expresso em malhas encaixadas, em um algoritmo paralelo sistólico. Apresentamos primeiro, resultados sobre a detecção de computações independentes e a ordenação das computações para execução em paralelo. Usando esses resultados, apresentamos um método original que, aplicado a um algoritmo sequencial satisfazendo uma certa condição ("uso de iteração anterior"), produz o algoritmo sistólico ótimo, tanto em tempo como em espaço.

ABSTRACT

In this work we develop a formal method for the transformation of a sequential algorithm expressed as nested loops into a parallel systolic algorithm. First we present results on the detection of independent computations and the ordering of these computations for parallel execution. Using these results, we present an original method that, applied to a sequential algorithm satisfying a certain condition ("use of previous iteration"), gives the optimum systolic algorithm both in time and space.

oOo

# Í N D I C E

CAPÍTULO I

Introdução . . . . . 01

CAPÍTULO II - O CONCEITO DE ALGORÍTMO SISTÓLICO e EXEMPLOS

§1. Algorítmo Sistólico . . . . . 03

§2. Exemplos . . . . . 09

§3. Motivação para Sistematizar a Geração de Algorítmos Sistólicos . . . . . 42

CAPÍTULO III - DETECÇÃO DE PARALELISMO

§1. Definições . . . . . 43

§2. Detecção de Paralelismo . . . . . 53

§3. Ordenação de execução . . . . . 85

CAPÍTULO IV - TRANSFORMAÇÃO DE ALGORITMO E OBTENÇÃO DE SISTEMAS SISTÓLICOS

§1. Transformação . . . . . 91

§2. Sistema Sistólico . . . . . 108

CAPÍTULO V - A ESCOLHA DO TRANSFORMADOR ÓTIMO T

§1. Algumas Considerações Iniciais . . . . . 127

§2. Otimização de Tempo . . . . . 131

§3. Otimização de Espaço . . . . . 148

CAPÍTULO VI - CONCLUSÕES . . . . . 151

BIBLIOGRAFIA . . . . . 155

# Capítulo 1

# Introdução

Este trabalho trata do desenvolvimento formal de algoritmos paralelos sistólicos. O conceito de algoritmo sistólico, a ser exposto no próximo capítulo, foi introduzido em fins de 1970 e tem causado um impacto muito grande no projeto de algoritmos altamente paralelos, com fina granularidade de paralelismo.

O nosso trabalho de sistematização do projeto de algoritmos sistólicos pode ser dividido em duas partes: detecção de paralelismo e transformação de um algoritmo sequencial em sistólico .

Na parte relativa à detecção de paralelismo, o nosso trabalho é baseado essencialmente no trabalho de Fortes e Maldovan [85] Os principais resultados, entretanto, são reorganizados de maneira mais clara, com ilustração por meio de muitos exemplos.

A contribuição mais importante do nosso trabalho reside na parte referente ao método de síntese formal de algoritmos sistólicos a partir de algoritmos sequenciais expressos em malhas encaixadas ou equações de recorrência uniforme. O método consiste em obter um transformador T que distribui de maneira adequada as computações originais em hiperplanos onde as computações são independentes e portanto podem ser executadas em paralelo. No trabalho original de Fortes e Maldovan, esta parte é tratada de maneira informal, conten

tando-se em obter uma solução boa, não necessariamente ótima. Nós propusemos um método original que, dentro de certas condições, produz um sistema sistólico ótimo, em termos de tempo de execução e em termos de número de processadores ou células do sistema.

Um outro resultado original é o algoritmo sistólico para a decomposição LU.

No capítulo 2, introduzimos o conceito de sistema sistólico apresentando vários exemplos para ilustrar suas características e vantagens.

No capítulo 3, desenvolvemos a técnica de detectar computações independentes para um algorítmo escrito na forma de malhas encaixadas, que servirá de base para o capítulo 4.

No capítulo 4, definimos o transformador T de algorítmo que modifica um algoritmo sequencial na forma de malhas encaixadas para que ele fique nas condições de poder aplicar um resultado do capítulo 3 e em seguida como ele é executado sistolicamente.

No capítulo 5, trataremos da otimização do transformador T do capítulo anterior, para que se obtenha um sistema de menor tempo de execução e ocupando menor espaço.

No capítulo 6, apresentamos comentários e conclusões sobre o trabalho.

# Capítulo 2

# O Conceito de Algoritmo Sistólico e Exemplos

## §.1 Algoritmo Sistólico

Nos últimos anos, a tecnologia de circuitos integrados VLSI ("Very Large Scale of Integration") (Mead & Conway [80]) tem desenvolvido bastante, com enormes avanços tanto no tamanho dos componentes eletrônicos, como na velocidade de computação.

A densidade de componentes num "chip" ou pastilha tem crescido de maneira surpreendente. Uma pastilha de silício, que integrava apenas algumas centenas de componentes vinte anos atrás, hoje chega a conter cerca de um milhão de componentes em menos de 1 centímetro quadrado de área.

Motivado por tais avanços tecnológicos, um ramo da ciência de computação que tem despertado muita atenção nos últimos anos é o de desenvolvimento de algorítmos paralelos com características apropriadas para implementação direta em silício.

Pastilhas de uso específico, implementando tais algoritmos, são projetadas para a resolução de problemas que requerem pesadas computações, impossíveis de serem atendidas por máquinas de arquitetura convencional de Von Newmann.

São enquadrados nesse tipo de problemas que demandam computações de alto desempenho, muitos problemas das áreas de processamento de imagens, processamento de sinais e computação científica.

O processo do projeto de uma pastilha de circuito integrado, entretanto, é em geral complexo e de alto custo.

Para que o esquema de computação paralela mencionado seja viável, é necessário que o algoritmo paralelo possua características apropriadas para implementação em pastilha.

O conceito de algoritmo sistólico foi introduzido em fins da década de 70, por Kung & Leiserson [79], justamente para responder a este desafio.

## 1.1. O que é sistema sistólico

O termo "sistólico" refere-se a contração rítmica do coração que impulsiona o sangue no sistema circulatório humano.

Um sistema sistólico é um sistema de computação baseado numa rede de processadores chamados de "células" que têm as seguintes características.

1. Os processadores que compõem a rede são todos idênticos ou constituídos de poucos tipos diferentes de processadores.
2. Cada processador efetua operações bastante simples sobre os dados que o atravessam, com pequeno repertório de instruções.
3. Geralmente, em cada ciclo de relógio, os processadores efetuam exatamente a mesma sequência de operações e depois impulsionam os dados para outros processadores para serem usados no ciclo seguinte.
4. O fluxo de dados e sinais de controle é bastante simples e regular, requerendo apenas comunicações locais entre processadores vizinhos.

5. Uso extensivo de paralelismo e "pipelining" ou computação serial encadeada.

## 1.2. Vantagens

Por suas características, o sistema sistólico apresenta inúmeras vantagens, descritas a seguir:

1. Necessita projetar apenas poucos tipos de células básicas diferentes (às vezes apenas um tipo).
2. Uma pastilha é basicamente formada por um grande número de cópias de células básicas.
3. A interconexão entre as células é apenas local.
4. A interconexão entre células é regular.
5. Alto desempenho devido ao uso de paralelismo e "pipelining" de modo muito intensivo.

As vantagens 1 a 4 proporcionam relativa facilidade para a fabricação de pastilha.

Em particular, a vantagem 3 evita um dos grandes problemas do projeto de circuitos que é a comunicação a grande distância e a vantagem 4 torna o projeto modular, i.e. pode ser estendido com facilidade.

Como foi mencionado em 1.1., várias classes de problemas a serem resolvidos por métodos computacionais requerem curto tempo de resposta ou alto desempenho no poder computacional, não satisfeitos pelas soluções tradicionais de processamento sequencial.

A vantagem 5 serve justamente para superar esta barreira.

## 1.3. Estrutura de interconexão

Como foi dito em 1.2. as células básicas (processadores) de um sistema sistólico são colocadas numa disposição regular com comunicação local.

Ilustramos algumas topologias mais típicas.

Fig. 2.1.a. Estrutura de lista linear, já usada para reconhecimento de padrão (Foster & Kung [80]), transformada discreta de Fourier por exemplo.

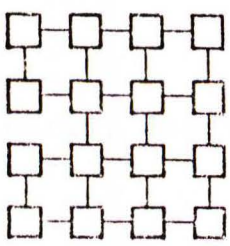


Fig. 2.1.b. Estrutura de matriz retangular, usada para programação dinâmica por exemplo.

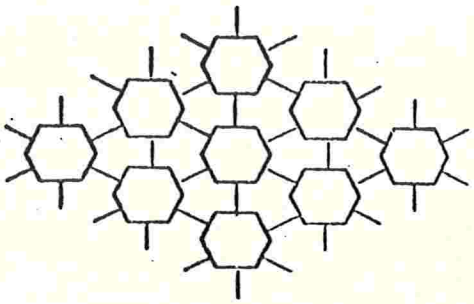


Fig. 2.1.c. Matriz hexagonal, bastante usada para cálculos matri - ciais. Pode ser vista como uma matriz retangular com uma conexão diagonal.

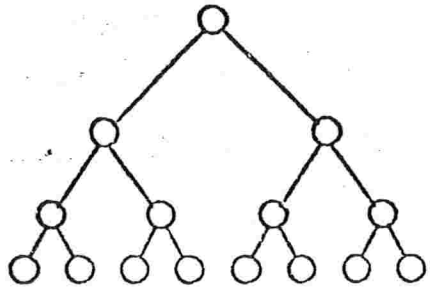


Fig. 2.1.d. Árvore binária, usada para busca e ordenação (Song [81]).

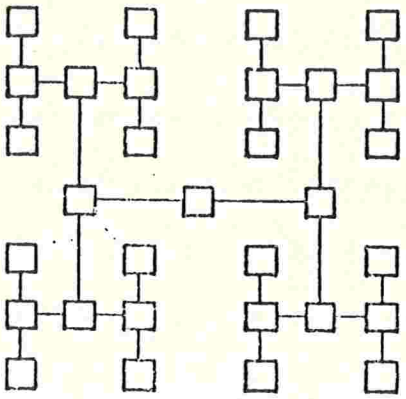


Fig. 2.1.e. É chamada "disposição H" que é nada mais que uma disposição compacta de uma árvore binária.

Na próxima seção veremos alguns exemplos de sistemas sistólicos usando algumas destas topologias.

oOo

## §.2 Exemplos

Nesta seção,veremos 4 exemplos de sistemas sitólicos para ilustrar melhor as características mencionadas na seção anterior.

Os dois primeiros exemplos são do trabalho pioneiro de Kung & Leiserson [79] que introduziram o conceito de sistema sistólico e são as soluções sistólicas para problemas de multiplicação de matrizes de banda e de sistemas linerares triangulares.

O terceiro exemplo é proposto por Okuda & Song [86] e é um sistema sistólico para multiplicação de matrizes com desempenho melhorado em relação ao de Kung & Leiserson.

O último exemplo proposto aqui por Okuda também é uma alternativa melhorada para decomposição LU.

### 2.1. Multiplicação de Matrizes (Kung & Leiserson)

Sejam A e B duas matrizes quadradas de banda de ordem n. A matriz produto $\mathbb{C}$=AB é obtida usando a seguinte recorrência:

$$c_{ij}^{(1)} := 0 \ ,$$

$$c_{ij}^{(k+1)} := c_{ij}^{(k)} + a_{ik}b_{kj} \ ,$$

$$c_{ij} := c_{ij}^{(n+1)}$$

para todo i,j,k, $1 \leq i,j,k \leq n$

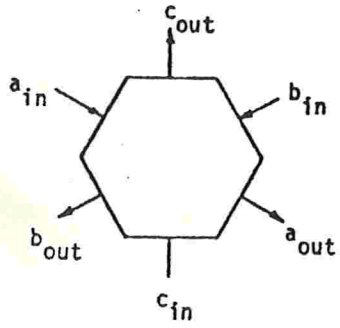
$$
\underset{q_A}{\overbrace{\phantom{x}}}\begin{pmatrix} \overbrace{a_{11} & a_{12}}^{P_A} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & \ddots & \ddots & \\ 0 & & & & \end{pmatrix}
\quad
\underset{q_B}{\overbrace{\phantom{x}}}\begin{pmatrix} \overbrace{b_{11} & b_{12} & b_{13}}^{P_B} & & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ & & b_{43} & \ddots & \ddots \\ 0 & & & & \end{pmatrix}
=
\begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & \ddots & \ddots & \ddots \\ & & & & \\ 0 & & & & \end{pmatrix}
$$

A B C

Fig. 2.1.

Se as matrizes A e B são de banda, como na Fig. 2.1., com a largura de banda $W_A = P_A + q_A - 1$ e $W_B = P_B + q_B - 1$, então os elementos da matriz C podem ser obtidos usando um sistema sistólico com $W_A.W_B$ células dispostas numa matriz hexagonal como mostra a Fig. 2.3.

Na Fig. 2.2., temos a representação esquemática de cada célula, denominada de processador de produto interno, com três direções para o fluxo de informações.




$a_{out} \leftarrow a_{in}$
$b_{out} \leftarrow b_{in}$
$c_{out} \leftarrow c_{in} + a_{in}b_{in}$

Fig. 2.2.

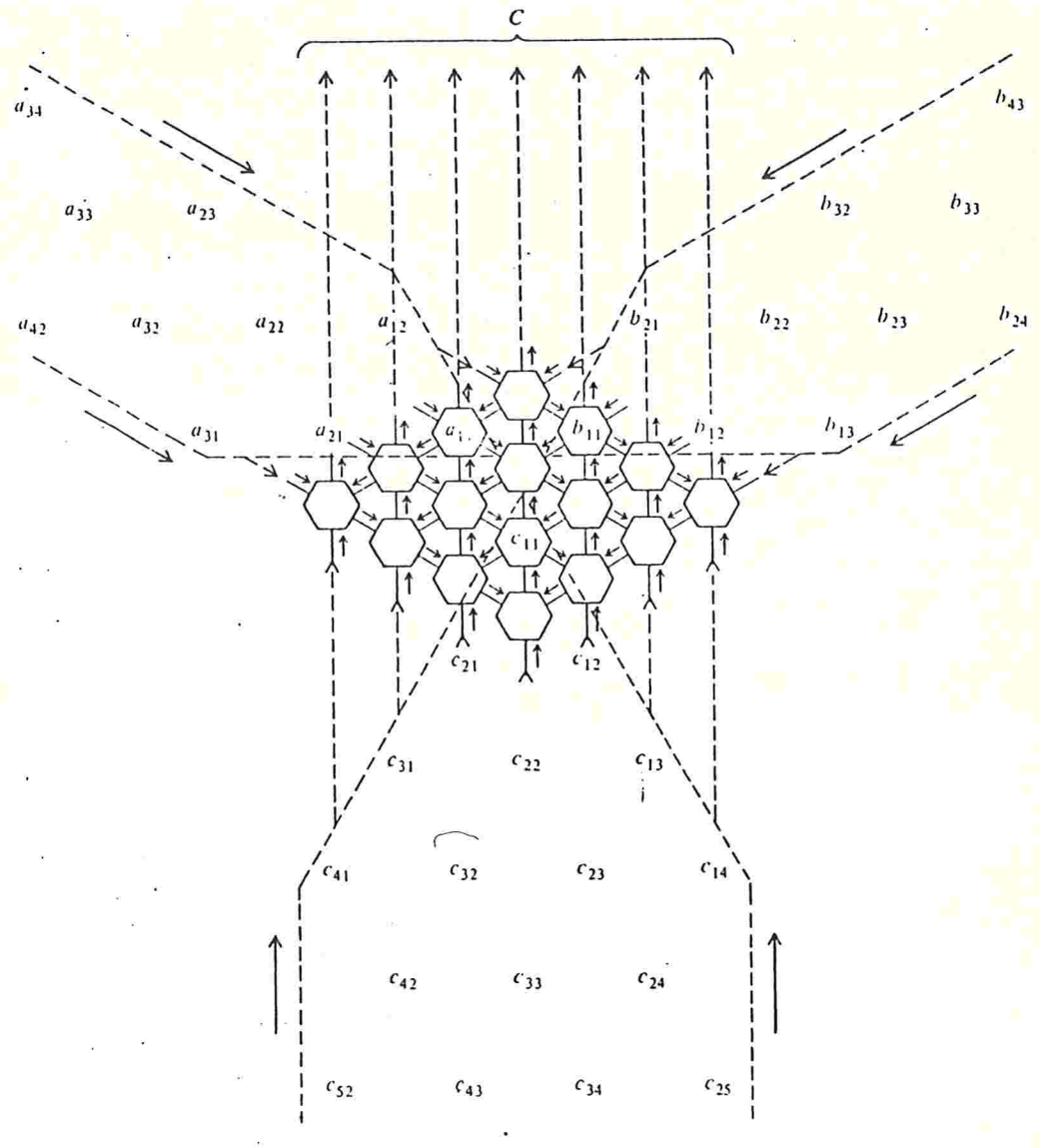



Fig. 2.3.

No caso do exemplo da Fig. 2.1., $W_A = 4$ e $W_B = 4$ e a matriz hexagonal do sistema sistólico é constituída de $4 \times 4 = 16$ células conforme Fig. 2.3.

Os fluxos dos elementos das matrizes A, B e C movem-se na matriz hexagonal do sistema em três direções diferentes, os elementos

da matriz C entram no lado inferior do sistema com valores inicializados como zero e movem-se para cima, os elementos de A movem-se da esquerda para a direita de modo descendente e os elementos de B movem-se da direita para a esquerda, também de modo descendente.

Cada $C_{ij}$ entra no sistema com valor inicial zero e durante sua trajetória ascendente e acumula todas as suas parcelas, de forma que ao sair do sistema pelo lado de cima, conterá exatamente o valor desejado.

A Fig. 2.4. mostra quatro etapas da multiplicação do exemplo da Fig. 2.1.

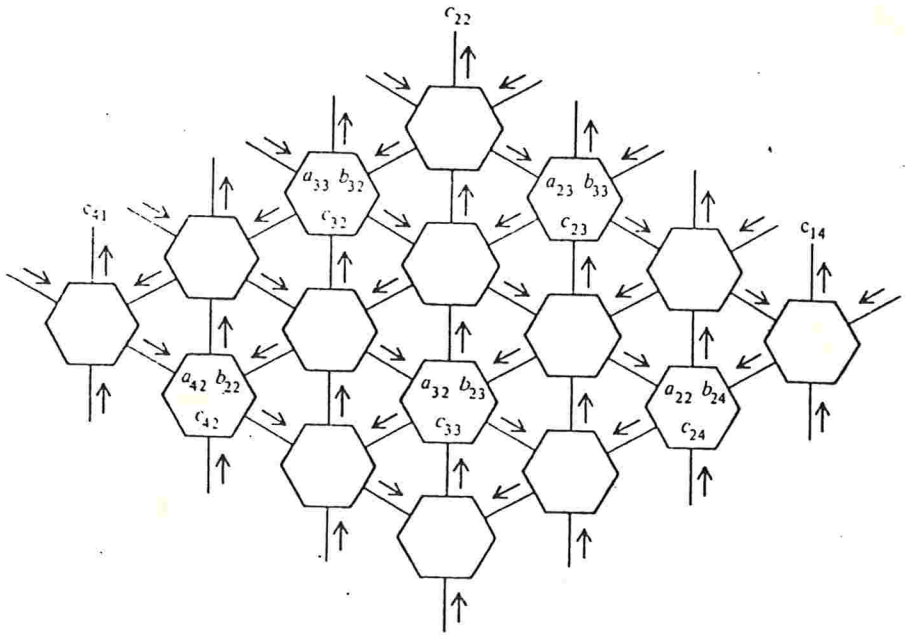



Fig. 2.4.a.

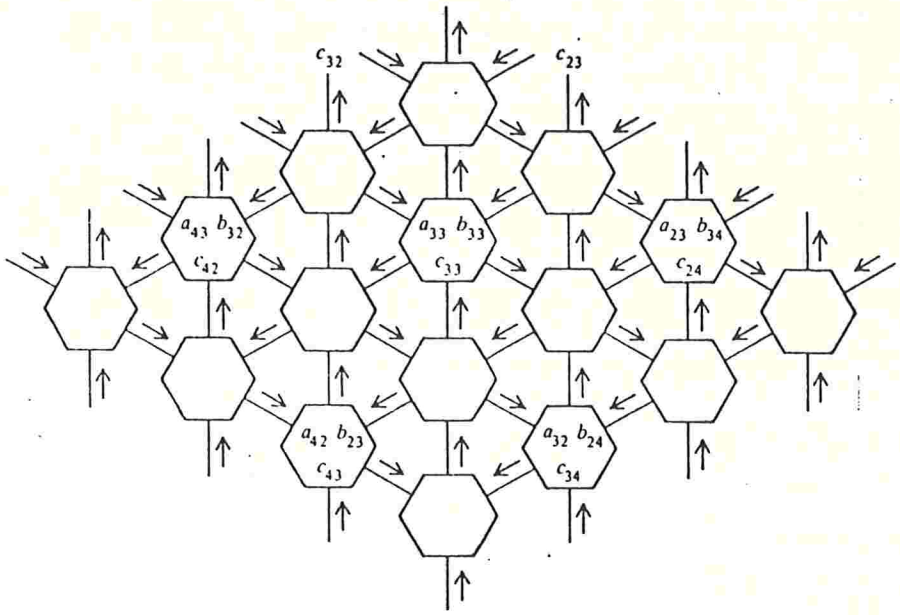



Fig. 2.4.b.

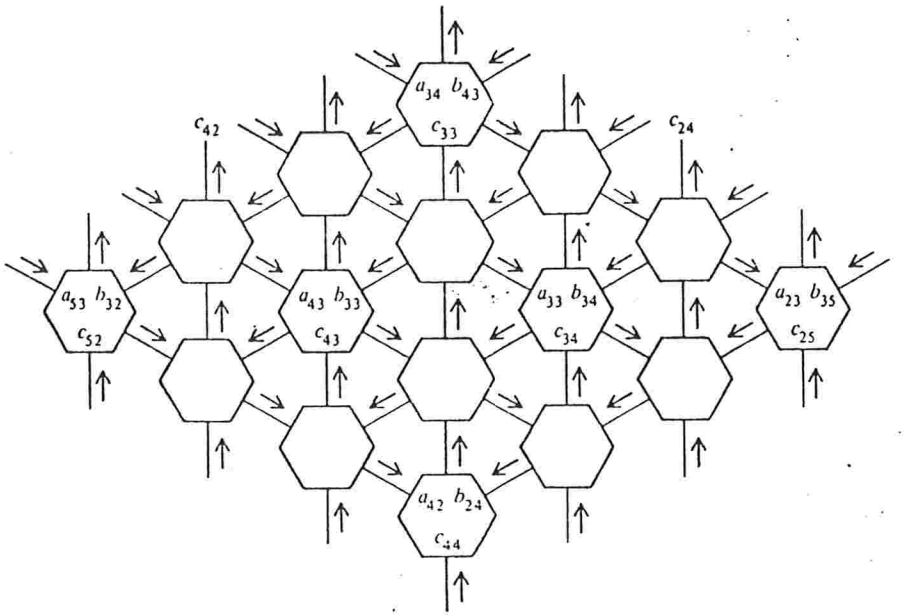



Fig. 2.4.c.

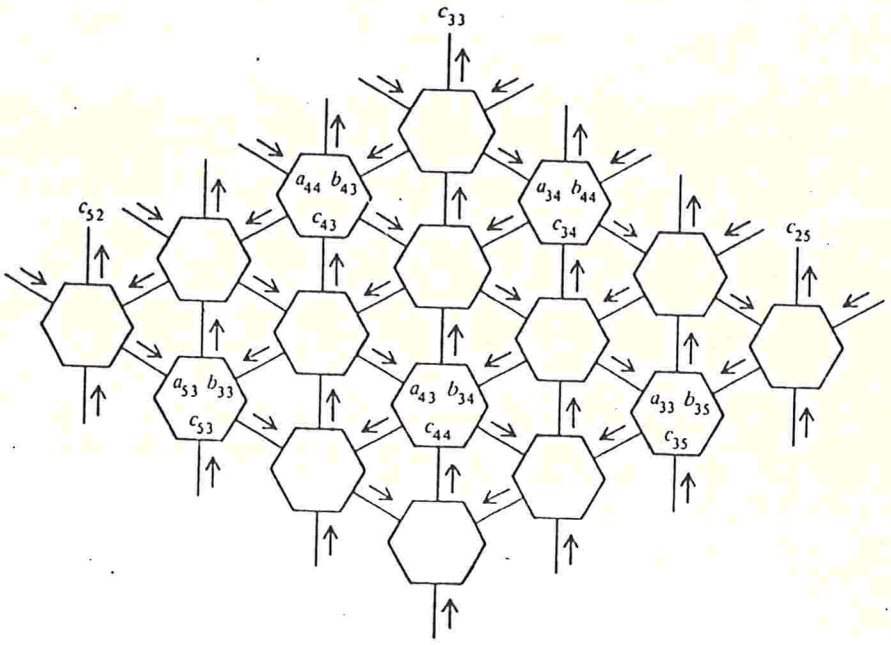



Fig. 2.4.d.

A multiplicação de duas matrizes quadradas de ordem n de banda A e B com larguras de banda $W_A$ e $W_B$, respectivamente, pode ser realizada em $3n + \min(W_A, W_B)$ passos, usando uma matriz hexagonal de $W_A$ $W_B$ células.

Em qualquer uma das três direções do fluxo de dados, apenas uma célula de cada três células consecutivas está ativa em qualquer etapa do processamento.

## 2.3. Sistema linear triangular

Uma vez feita a decomposição LU de uma matriz A do sistema de equações lineares AX = b, resta-nos resolver 2 sistemas lineares triangulares.

Seja $A = (a_{ij})$ uma matriz $n \times n$ não singular, triangular inferior de banda. (Vamos tratar apenas do sistema linear triangular inferior pois o outro é análogo)

Seja $X = (x_1, \ldots x_n)^T$ o vetor solução do sistema $AX = b$.

$$\underbrace{q\left\{\begin{pmatrix} a_{11} & & & & & 0 \\ a_{21} & a_{22} & & & & \\ a_{31} & a_{32} & a_{33} & & & \\ a_{41} & a_{42} & a_{43} & a_{44} & & \\ & a_{52} & a_{53} & a_{54} & a_{55} & \\ & & a_{63} & & & \ddots \\ 0 & & & & & \end{pmatrix}\right.}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ \vdots \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ \vdots \end{pmatrix}}_{b}$$

Fig. 2.8. Sistema com largura de banda $W = q = 4$

Os elementos $x_i$ do vetor solução X podem ser obtidos utilizando-se a seguinte recorrência:

$$Y_i^{(1)} = 0$$

$$Y_i^{(k+1)} = Y_i^{(k)} + a_{ik}x_k$$

$$x_i = (b_i - Y_i^{(i)})/a_{ii}$$

O sistema linear triangular de banda pode ser resolvido por um sistema sistólico em disposição linear conforme a Fig. 2.9

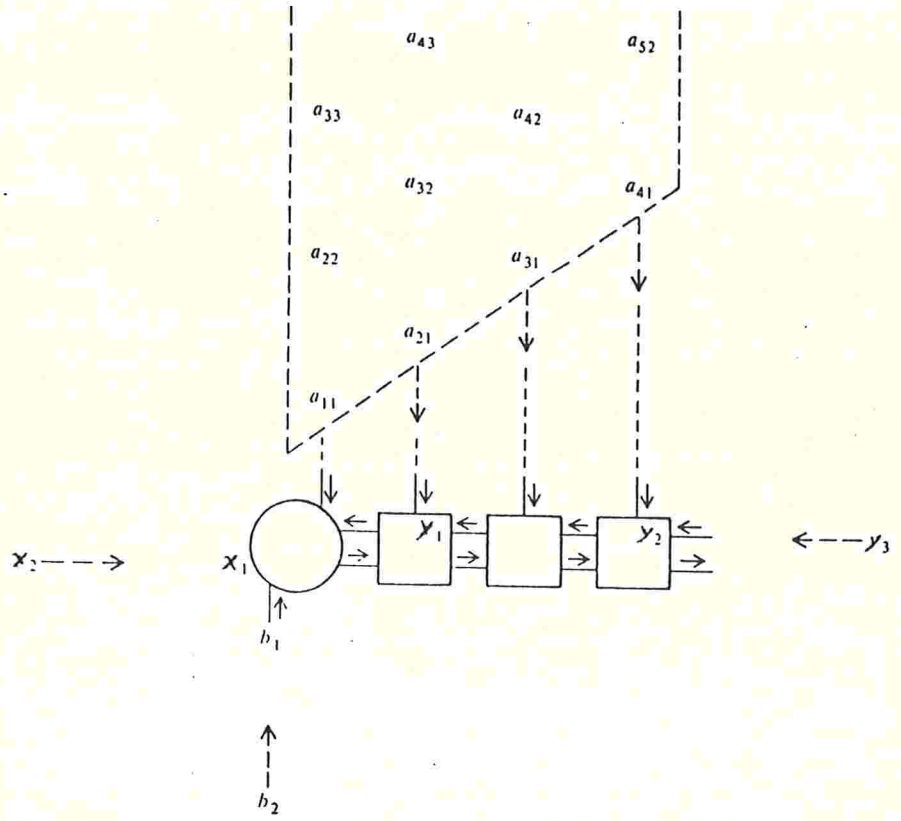
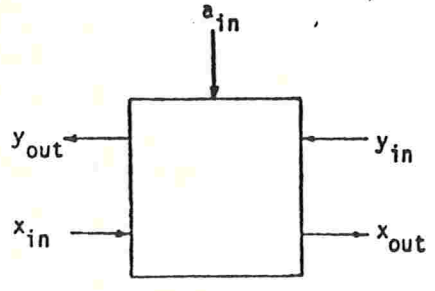


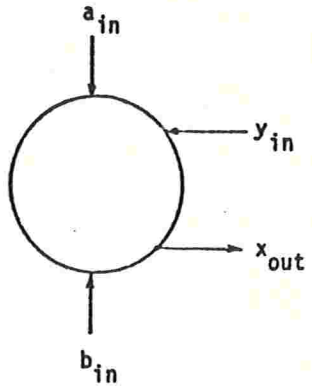

Fig. 2.9. Sistema sistólico para resolução de sistema linear triangular de banda.

O sistema sistólico é constituído por uma lista linear de células do tipo processador de produto interno, representadas por quadrados, e uma célula especial representada por círculo na extremidade de esquerda da lista. O funcionamento dos dois tipos de células é o seguinte:




$x_{out} \leftarrow x_{in}$

$y_{out} \leftarrow y_{in} + a_{in}\, x_{in}$




$x_{out} \leftarrow (b_{in} - y_{in})/a_{in}$

Fig. 2.10.

Os elementos x movimentam-se na direção horizontal da esquerda para a direita. Os elementos a e b movimentam-se na direção vertical mas nos sentidos conforme a Fig. 2.9., os elementos y, com valor inicial zero, entram no sistema sistólico pela extremidade de direita e caminham para a esquerda.

Cada $y_i$ acumula termos do produto interno à medida em que anda pelas células do tipo produto interno (célula quadrada), recolhendo uma parcela em cada célula por onde passa.

Ao chegar à célula circular, $y_i$ já terá acumulado o valor $a_{i1}x_i + a_{i2}x_2 + \ldots a_{ii-1}x_{i-1}$ e a célula circular dá o valor $(b_i - y_i)/a_{ii}$.

Este sistema sistólico resolve um sistema linear trinagular de banda $n \times n$, com largura de banda $W = q$, em $2n+q$ passos.

## 2.3. Multiplicação de matrizes (Okuda & Song)

Em 1986, Okuda e Song propuseram um sistema sistólico alternativo para a multiplicação matricial com algumas vantagens em relação ao de Kung & Leisersen.

### a) Descrição do sistema

Supondo A, B e C matrizes $n \times n$, não de banda, o sistema a ser descrito realiza o produto $C = A \times B$.

O sistema usa uma matriz de células interligadas conforme a Fig. 2.11.

Existem três direções para o fluxo de dados: os elementos da matriz A, juntamente com os dados f a serem descritos a seguir, procedem da esquerda para a direita, os elementos da matriz B, de cima para baixo e os da matriz C seguem direções diagonais.

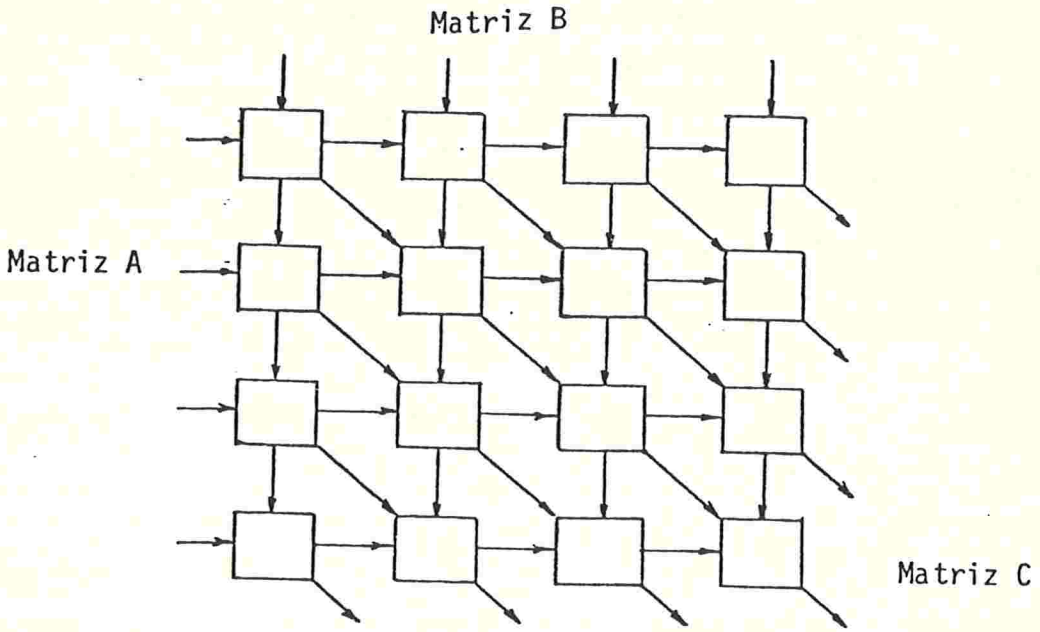
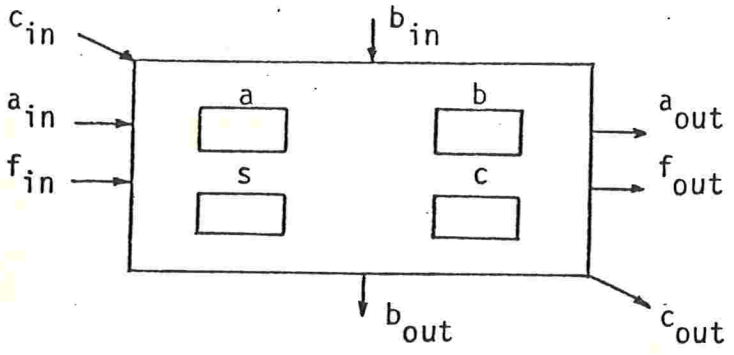



Fig. 2.11.

Uma célula básica, ilustrada na Fig. 2.12., é composta essencialmente de quatro registradores a, b, c e s.




Fase de entrada

$a := a_{in}$

$b := b_{in}$

$c := c_{in}$

Fase de Computação

$s := s + a.b$

if $f_{in} = 1$ then
$c := s$ e $s := 0$

Fase de saída

$a_{out} := a$

$b_{out} := b$

$c_{out} := c$

$f_{out} := f_{in}$

Fig. 2.12.

As entradas à célula são: $a_{in}$, $b_{in}$, $c_{in}$ e $f_{in}$ e as saídas são $a_{out}$, $b_{out}$, $c_{out}$ e $f_{out}$.

O dado f serve para indicar o fim do fluxo de $a_{in}$ de cada linha da matriz A, valendo 1 para o último elemento e 0 para os anteriores.

O registrador s, inicializado com zero, serve para acumular uma somatória e o seu valor é transferido para o registrador c quando $f_{in} = 1$.

Cada célula básica realiza um passo que é constituído de três fases: uma fase de entrada, uma fase de computação e uma fase de saída. A fase de saída de uma célula coincide com a fase de entrada da próxima célula.

A Fig. 2.13. mostra a multiplicação de duas matrizes 4 × 4, em cada célula estão mostrados os valores dos registradores a, b e c e por simplicidade, não estão mostrados os valores do registrador s, contendo as somatórias parciais acumuladas, mostrando apenas as saídas que interessam, isto é, os elementos da matriz produto.

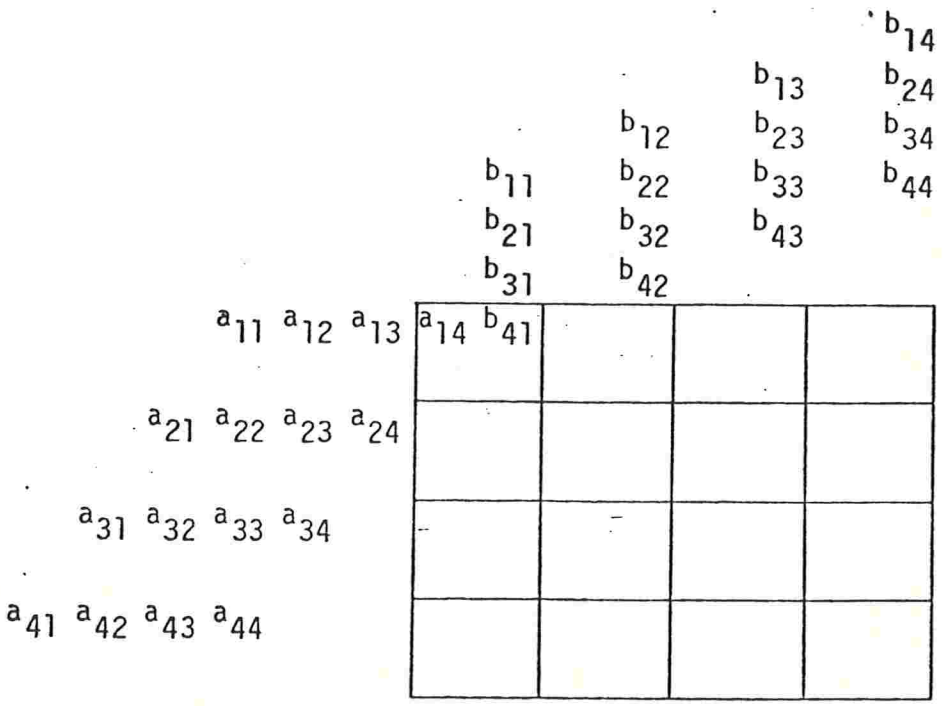



Fig. 13.a.

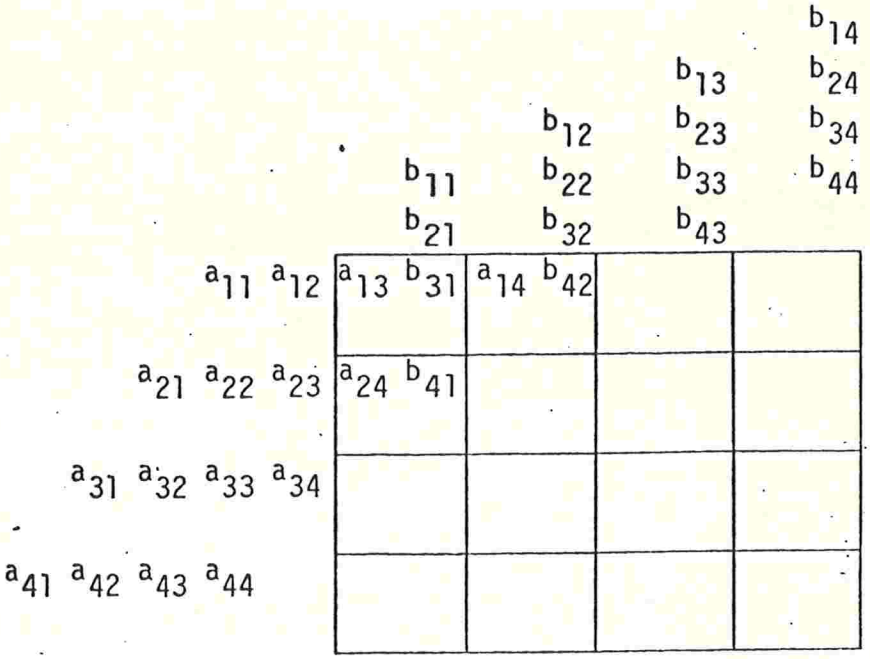



Fig. 13.b.

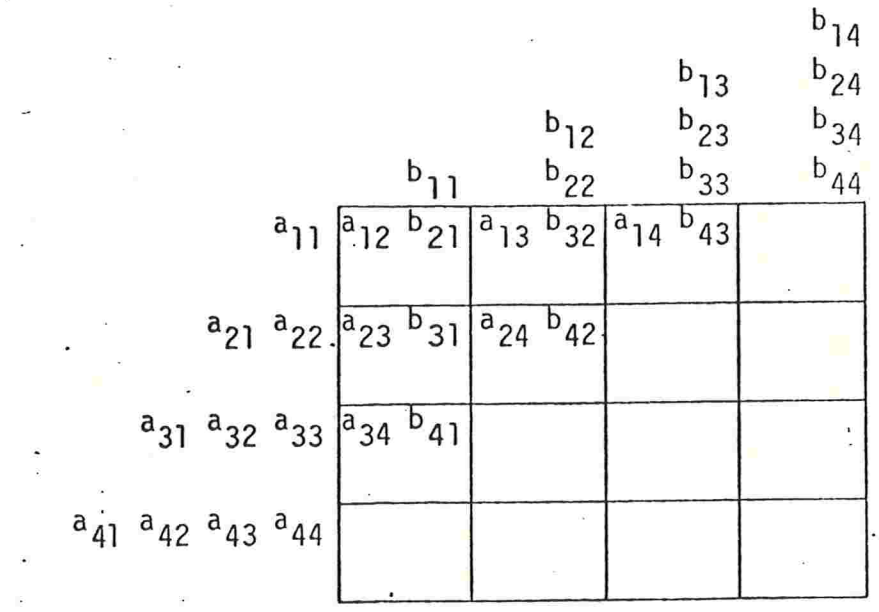



Fig. 13.c.

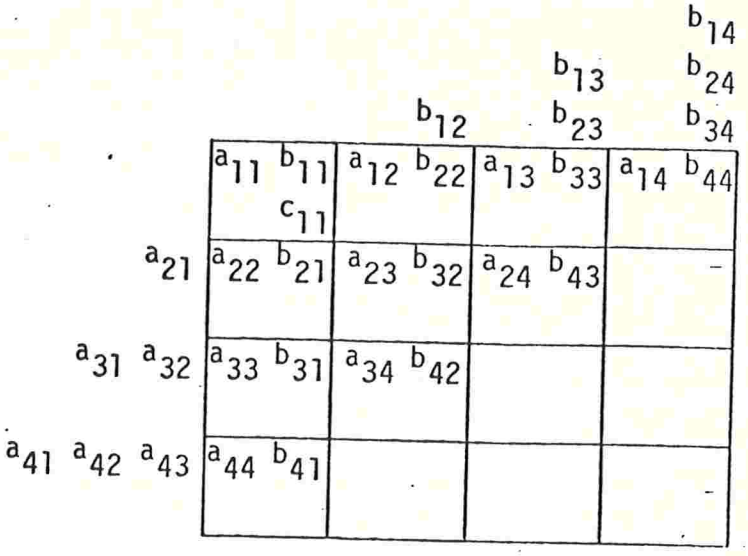



Fig. 13.d.

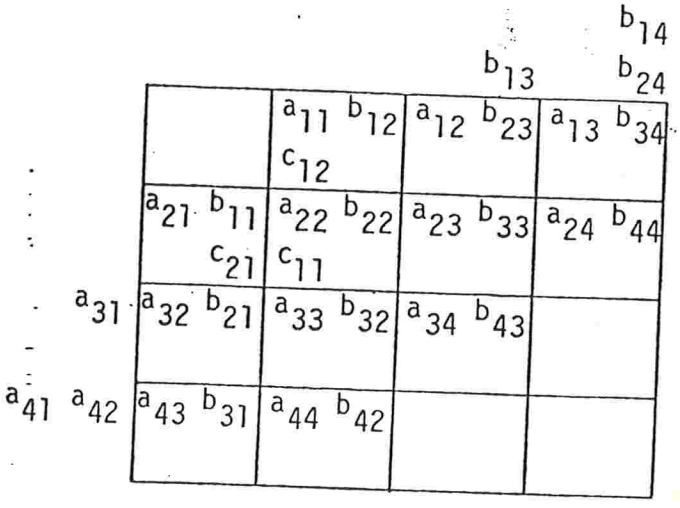



Fig. 13.e.

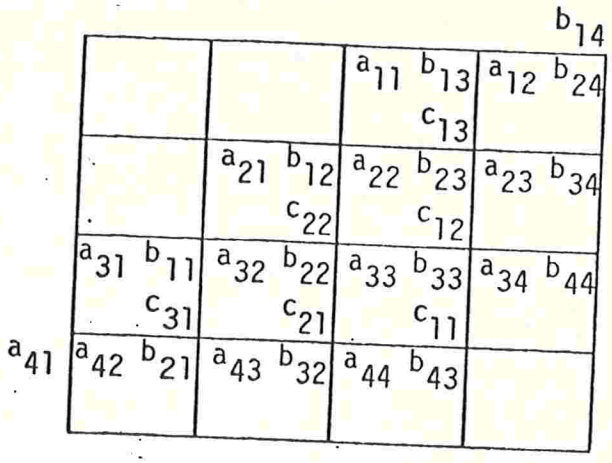



Fig. 13.f.

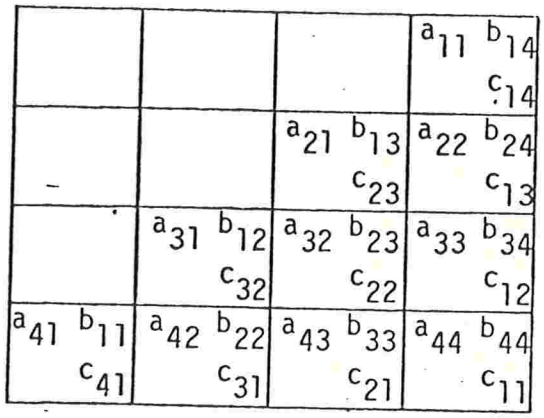



Fig. 13.g.

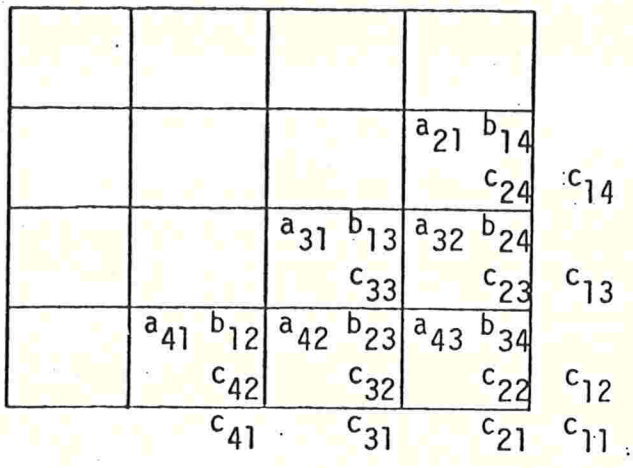



Fig. 2.13.h.

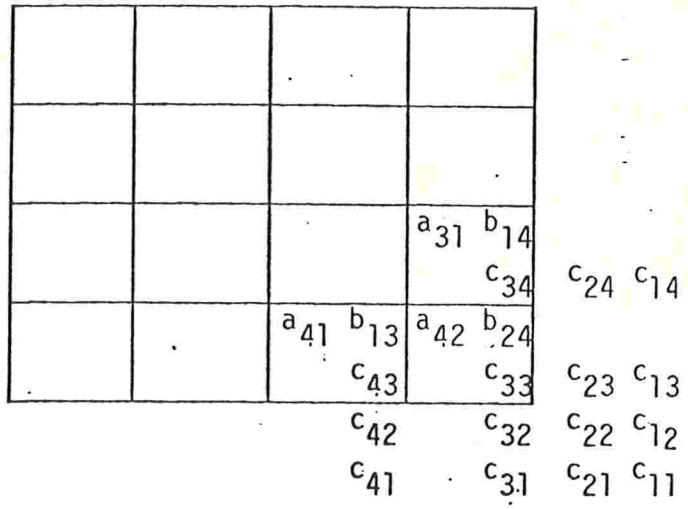



Fig. 2.13.i.

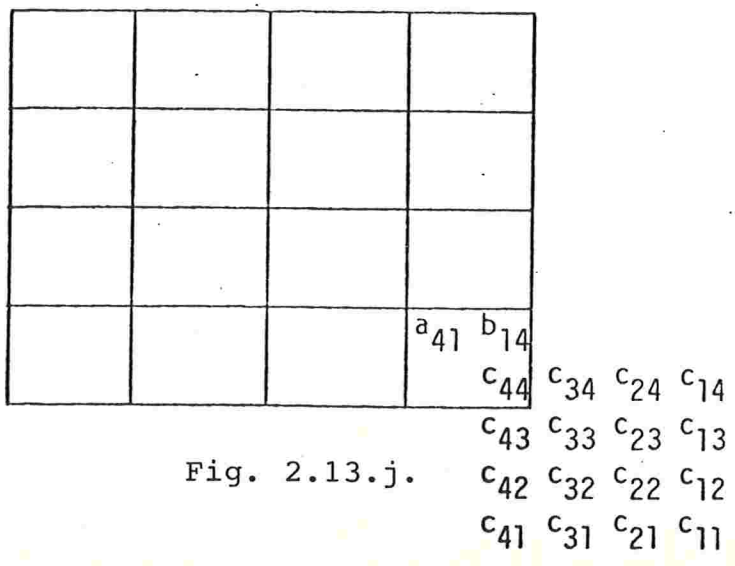



Fig. 2.13.j.

b) Observações

i) para matrizes completas n × n, o sistema leva 3n passos usando $n^2$ células, contra o tempo de 5n passos com aproximadamente $4n^2$ células necessárias pelo sistema em 2.1.

ii) Neste sistema, todas as células podem estar ativas enquanto que no algorítmo de 2.1, em qualquer instante, no max 1/3 das células estão ativas.

iii) Uma outra vantagem do método apresentado é a possibilidade do uso de um sistema sistólico de tamanho n × n para multiplicação de 2 matrizes maiores que n × n, através de um processo de decomposição descrito com detalhes em Okuda & Song [86].

## 2.4. Decomposição LU de uma matriz

O problema de decomposição LU consiste na decomposição de uma matriz A em um produto de uma matriz triangular inferior L com uma matriz triangular superior U.

Seja AX = b um sistema de equações lineares.

Uma vez que A é decomposto em LU, o sistema AX = b é equivalente a LUX = b e ele é resolvido da seguinte maneira:

i) resolve-se o sistema linear triangular LY = b, obtendo-se Y.

ii) em seguida, resolve-se o sistema linear triangular UX = Y, obtendo-se X.

Supõe-se que a matriz A tem a propriedade de que sua decomposição LU possa ser feita com o método de eliminação de Gauss sem pivo

tação (isto é verdade para A simétrica positiva definida, irredutível ou diagonal dominante, por exemplo).

A decomposição de uma matriz A em LU, onde $L = (\ell_{ij})$ é uma matriz triangular inferior e $U = (u_{ij})$ é uma matriz triangular superior, é obtida com a seguinte recorrência:

$$a_{ij}^{(1)} := a_{ij} \ ,$$

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} + \ell_{ik}(-u_{kj}) \ ,$$

$$\ell_{ik} := \begin{cases} 0 & \text{se } i < k \\ 1 & \text{se } i = k, \\ a_{ik}^{(k)} u_{kk}^{-1} & \text{se } i > k \end{cases}$$

$$u_{kj} := \begin{cases} 0 & \text{se } k > j \\ a_{kj}^{(k)} & \text{se } k \leq j \end{cases}$$

Em 1979, junto com seus outros trabalhos pioneiros vistos em 2.2.1 e 2.2.2., Kung & Leiserson apresentaram uma solução sistólica para a decomposição LU de uma matriz de banda usando um dispositivo semelhante ao de 2.1. (Kung & Leiserson, [79])

Naquela solução, assim como em 2.1., uma das três células está ativa em cada instante e o tempo é de 4n passos para uma matriz completa de n × n.

A seguir, mostraremos um sistema sistólico alternativo proposto por Kunio Okuda, cujo tempo é de 3n passos para uma matriz completa de n × n.

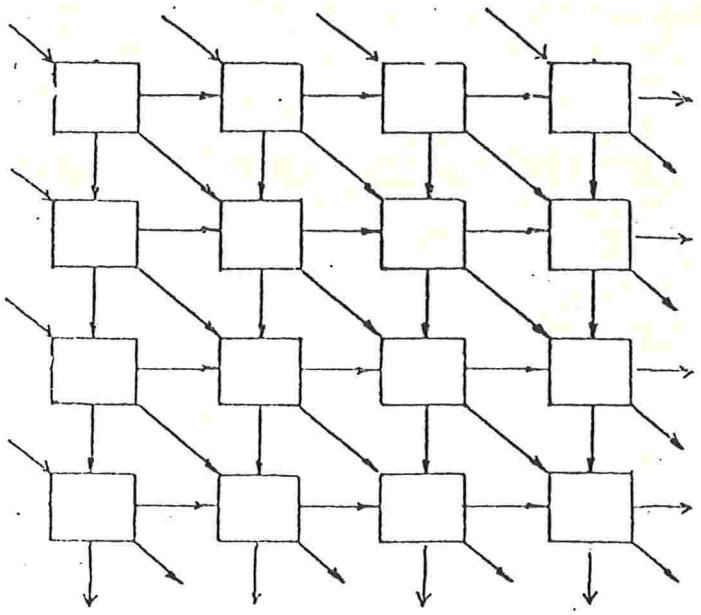
Vamos considerar a decomposição LU de uma matriz completa A(n × n). Para n = 4 o exemplo é mostrado na Fig. 2.14.

$$\overset{A}{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}} = \overset{L}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ \ell_{21} & 1 & 0 & 0 \\ \ell_{31} & \ell_{32} & 1 & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{pmatrix}} \times \overset{U}{\begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}}$$

Fig. 2.14

O sistema usa uma matriz de interconexão idêntica àquele de 2.3. e os fluxos de dados são como na Fig. 2.15.

matriz A



matriz L

matriz U

Fig. 2.15

O sistema possui apenas um tipo de célula básica, enquanto que o de Kung & Leiserson tem 2 tipos básicos.

Este sistema só trata de saídas de $\ell_{ij}$ para $i > j$ e $u_{ij}$ para $i \leqq j$ pois somente estes elementos nos interessam. Para $n = 4$, a matriz A pode ser escrita como na Fig. 2.16.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ \ell_{21}u_{11} & \ell_{21}u_{12}+u_{22} & \ell_{21}u_{13}+u_{23} & \ell_{21}u_{14}+u_{24} \\ \ell_{31}u_{11} & \ell_{31}u_{12}+\ell_{32}u_{22} & \ell_{31}u_{13}+\ell_{32}u_{23}+u_{33} & \ell_{31}u_{14}+\ell_{32}u_{24}+u_{34} \\ \ell_{41}u_{11} & \ell_{41}u_{12}+\ell_{42}u_{22} & \ell_{41}u_{13}+\ell_{42}u_{23}+\ell_{43}u_{33} & \ell_{41}u_{14}+\ell_{42}u_{24}+\ell_{43}u_{34}+u_{44} \end{pmatrix}$$

Fig. 2.16

A idéia básica do sistema é a seguinte: à medida que o elemento da matriz A anda na direção diagonal, cada célula deve reconstituir os elementos das matrizes U e L da maneira "inversa".

Cada célula básica é composta de 5 registradores Ra, Rb, Ru, Rℓ e Rx, sendo que Ra, Ru e Rℓ são para elementos de matrizes A, U e L, respectivamente, Rb é para valores intermediários e Rx é um registrador auxiliar que indica se a célula já transferiu para registrador $R_b$ algum valor a recebido pela célula.

$$R_x = \begin{cases} 1 & \text{a célula ainda não recebeu a;} \\ 0 & \text{a célula já recebeu a.} \end{cases}$$

$R_x$ é necessário, pois, dos fluxos de a que entram em cada célula, somente um a pode entrar no cálculo nesta célula e a célula deve ignorar a passagem de outros a's.

Antes de mostrar o funcionamento do sistema, temos algumas observações preliminares.

i) O a que já participou em cálculo, i.e., cujo valor já foi transferido para $R_b$ em alguma célula, não deve participar em nenhum outro cálculo e apenas deve passar pelas células.

ii) Toda vez que um b é transferido para $R_u$ ou para $R_\ell$ a célula serve apenas para passagem de valores de variáveis como a, $V_u$ e $V_\ell$ (estas duas últimas a serem descritas logo a seguir).

iii) O tipo de operações que cada célula executa em cada instante depende da configuração de 4 registradores (temos 7 casos a considerar).

Devido às observações acima, vamos usar mais três variáveis auxiliares Va, Vu e Vℓ que se deslocam junto com a, u e ℓ respectivamente, e suas funções são:

Va: para indicar se o valor de a já foi transferido para Rb em alguma célula.

$$V_a = \begin{cases} 1 & \text{se o valor de a não foi transferido} \\ 0 & \text{se o valor de a já foi transferido} \end{cases}$$

Vu: para indicar a presença de u

$$V_u = \begin{cases} 1 & \text{se um u está na célula} \\ 0 & \text{se não há u na célula} \end{cases}$$

Vℓ: para indicar a presença de ℓ

$$V\ell = \begin{cases} 1 & \text{se um } \ell \text{ está na célula} \\ 0 & \text{se não há } \ell \text{ na célula} \end{cases}$$

Considere-se uma célula básica, como ilustrada na Fig. 2.17.

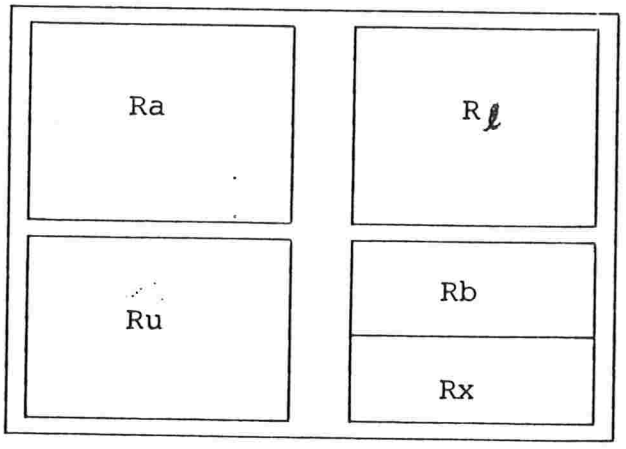



Fig. 2.17

As três fases de cada passo são:

Fase de entrada

$a := a_{in}$

$u := u_{in}$

$\ell := \ell_{in}$

$Vu := V_{u_{in}}$

$V_\ell := V_{\ell_{in}}$

$V_a := V_{a_{in}}$

Fase de computação

```
if Vx . Va = 1
   Then begin
        b:= a;
        va:= 0 ;
        x: = 0 ;
        end;
if Vℓ= Vu = 1
  then b:= b-ℓu;
if Vu = 1 and Vℓ = 0
   then begin
        b:= b/u;
        ℓ:= b;
        Vℓ:= 1
        end;
if Vx = Vu = Vℓ = 0
   them begin
        u:= b;
        Vu:= 1;
        end;
```

Fase de saída

$a_{out} := a$ ;

$u_{out} := u$;

$\ell_{out} := \ell$;

$V_{uout} := V_u$;

$V_{\ell out} := V_\ell$;

$V_{aout} := V_a$;

A seguir, a Fig. 2.17 mostra o funcionamento do sistema para uma matriz 4 × 4.

Note que cada figura mostra o fim de um passo. Por exemplo, da Fig. 2.18.1, na célula do canto esquerdo superior, $V_a = 1$ passou a $V_a = 0$ e $V_x = 1$ passou a $V_x = 0$.

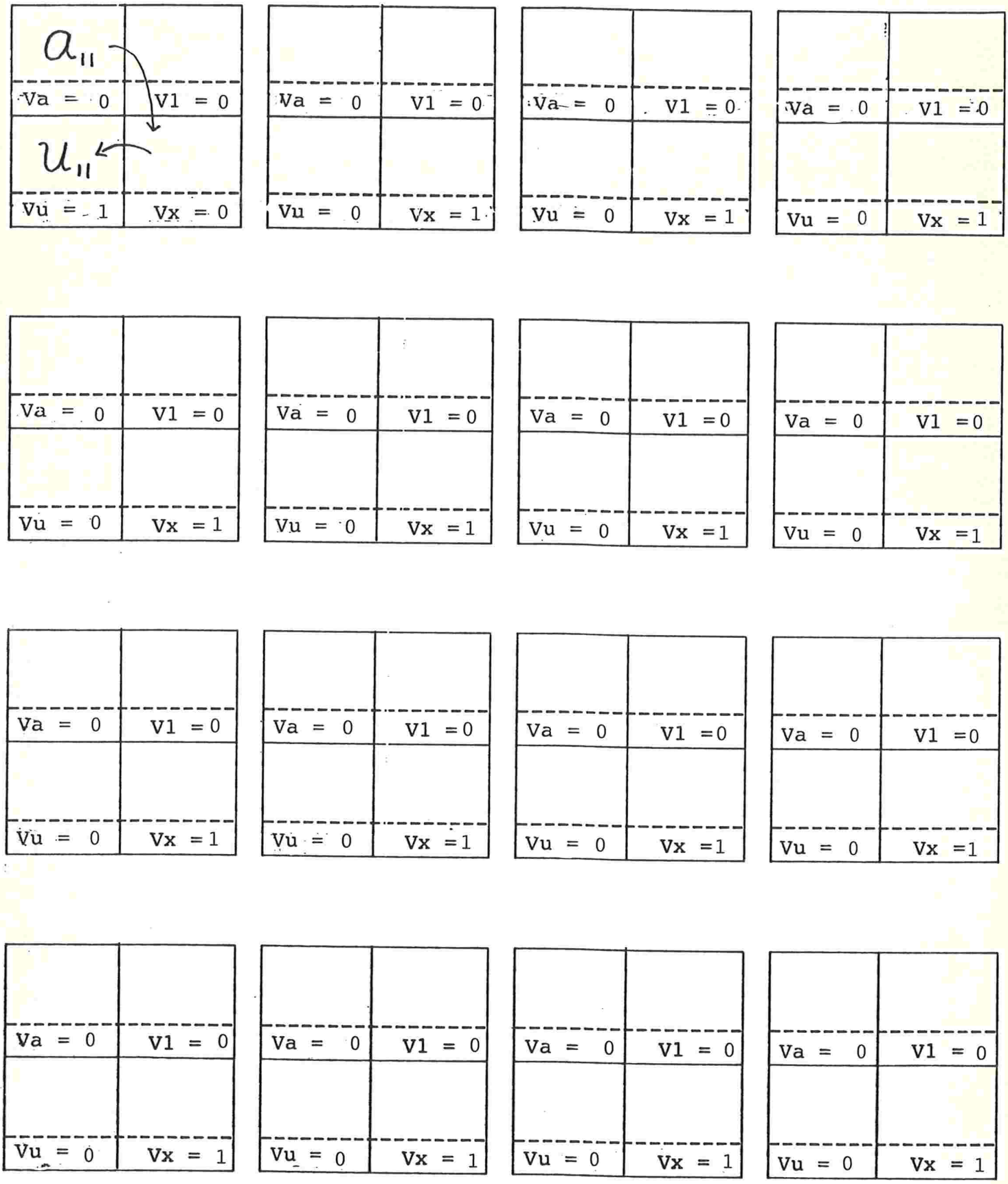

$a_{11}$
$u_{11}$
Va = 0
V1 = 0
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1

Fig. 2.18.1.

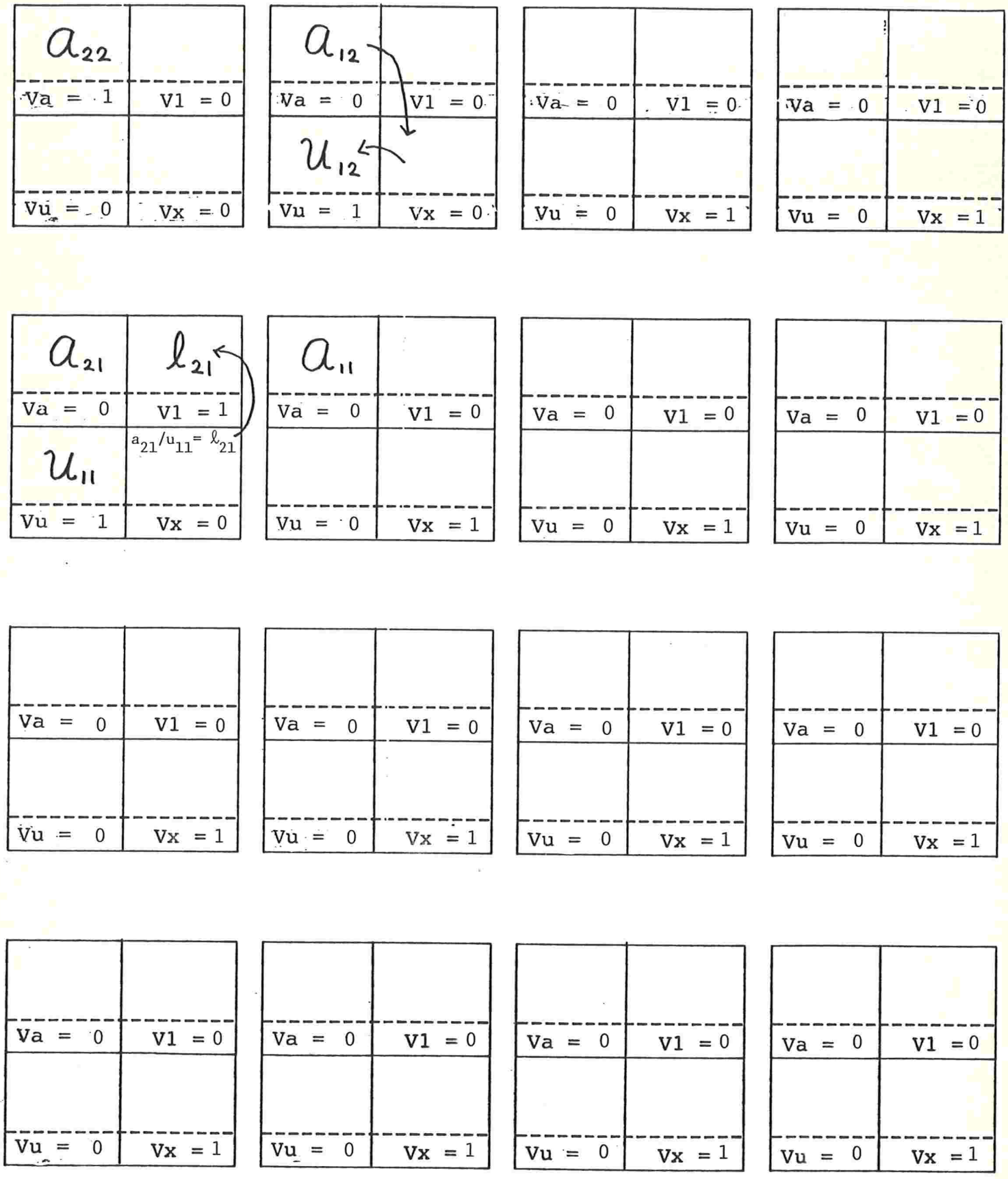

$a_{22}$
Va = 1
V1 = 0
Vu = 0
Vx = 0
$a_{12}$
Va = 0
V1 = 0
$u_{12}$
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
$a_{21}$
$l_{21}$
Va = 0
V1 = 1
$a_{21}/u_{11} = l_{21}$
$u_{11}$
Vu = 1
Vx = 0
$a_{11}$
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1

Fig. 2.18.2

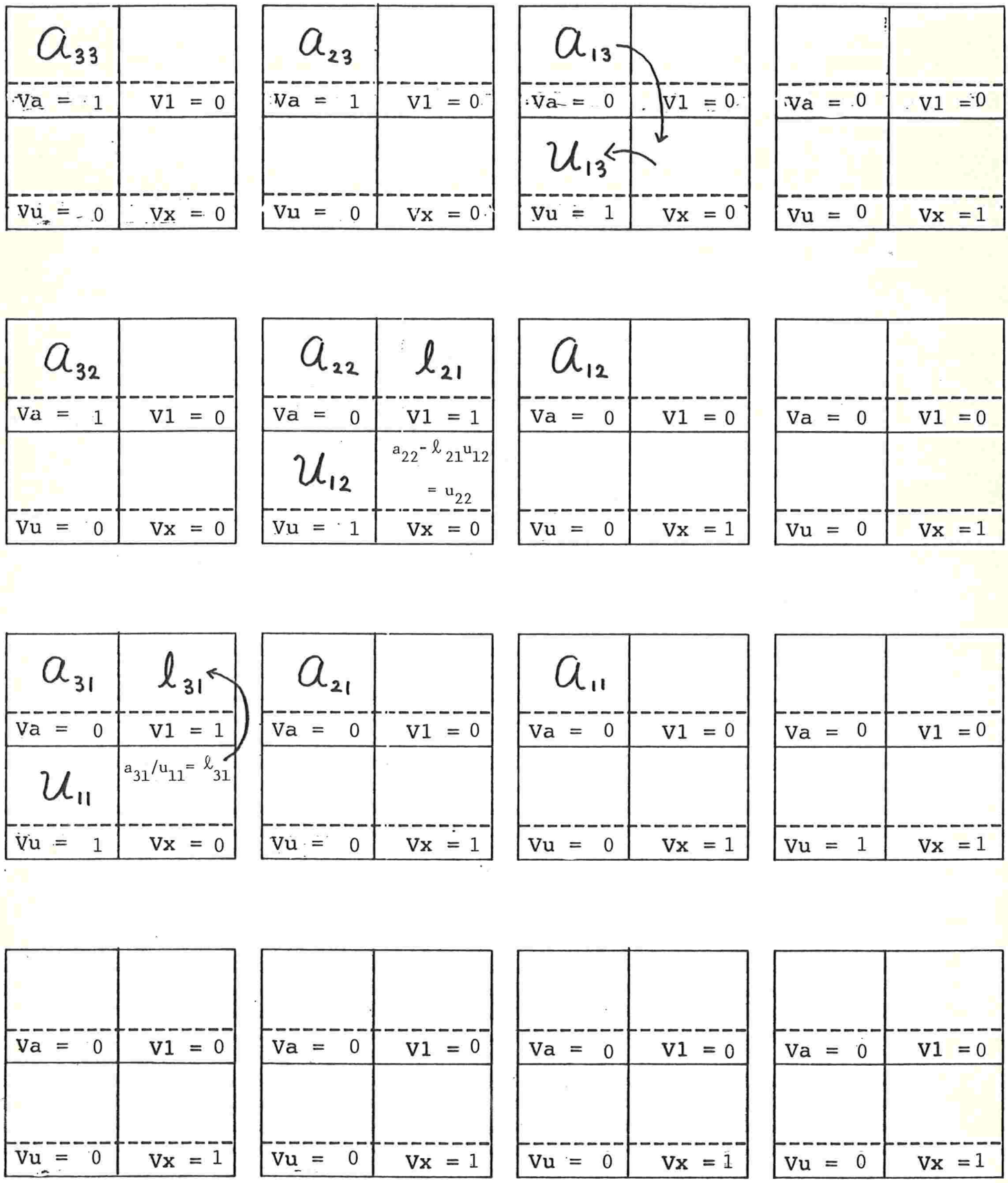

a33
Va = 1
V1 = 0
Vu = 0
Vx = 0
a23
Va = 1
V1 = 0
Vu = 0
Vx = 0
a13
Va = 0
V1 = 0
u13
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 1
a32
Va = 1
V1 = 0
Vu = 0
Vx = 0
a22
l21
Va = 0
V1 = 1
u12
a22 - l21u12 = u22
Vu = 1
Vx = 0
a12
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
a31
l31
Va = 0
V1 = 1
u11
a31/u11 = l31
Vu = 1
Vx = 0
a21
Va = 0
V1 = 0
Vu = 0
Vx = 1
a11
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 1
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1
Va = 0
V1 = 0
Vu = 0
Vx = 1

Fig. 2.18.3.

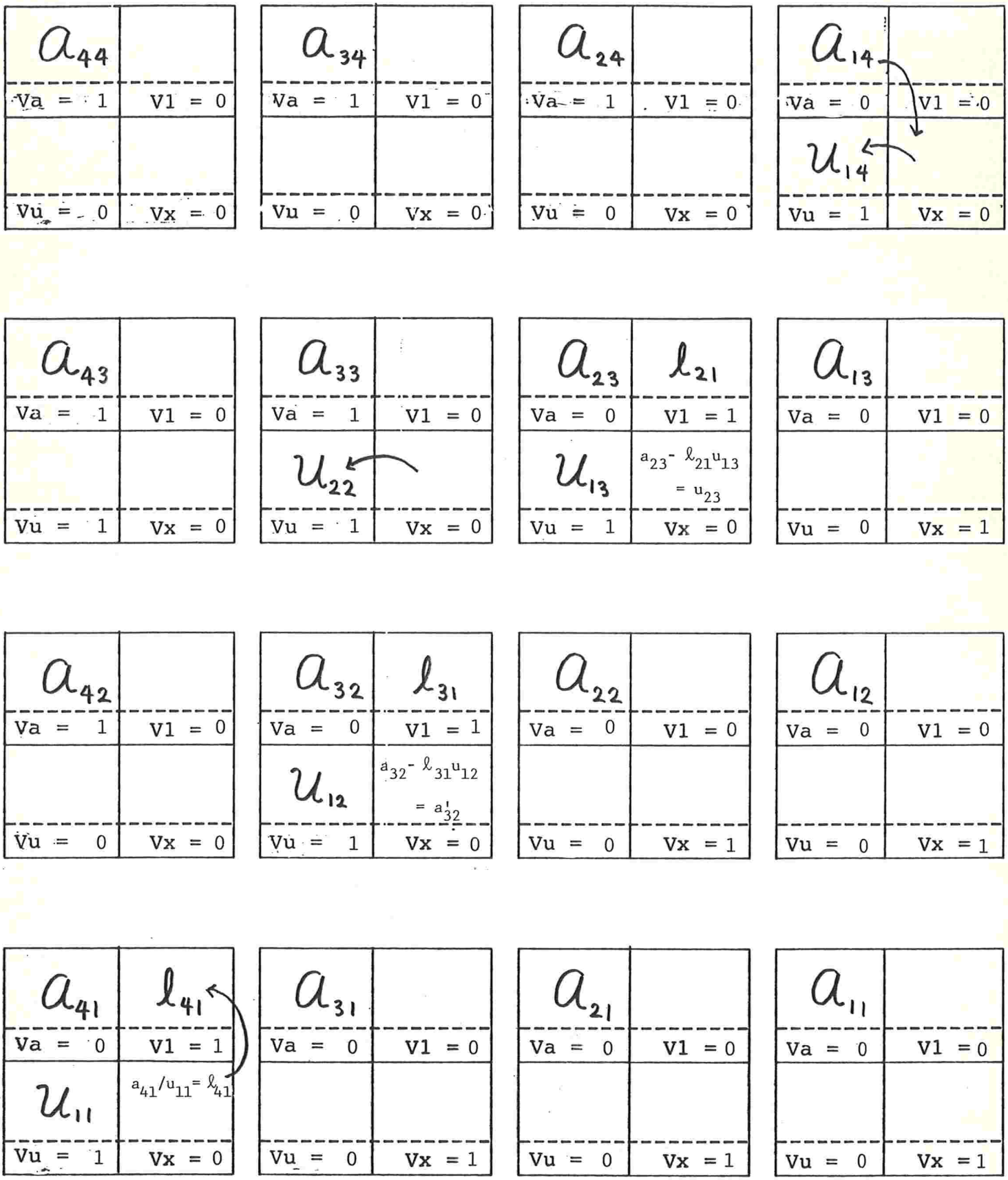

$a_{44}$ Va = 1 V1 = 0 Vu = 0 Vx = 0
$a_{34}$ Va = 1 V1 = 0 Vu = 0 Vx = 0
$a_{24}$ Va = 1 V1 = 0 Vu = 0 Vx = 0
$a_{14}$ Va = 0 V1 = 0 $u_{14}$ Vu = 1 Vx = 0
$a_{43}$ Va = 1 V1 = 0 Vu = 1 Vx = 0
$a_{33}$ Va = 1 V1 = 0 $u_{22}$ Vu = 1 Vx = 0
$a_{23}$ $l_{21}$ Va = 0 V1 = 1 $u_{13}$ $a_{23} - l_{21}u_{13} = u_{23}$ Vu = 1 Vx = 0
$a_{13}$ Va = 0 V1 = 0 Vu = 0 Vx = 1
$a_{42}$ Va = 1 V1 = 0 Vu = 0 Vx = 0
$a_{32}$ $l_{31}$ Va = 0 V1 = 1 $u_{12}$ $a_{32} - l_{31}u_{12} = a'_{32}$ Vu = 1 Vx = 0
$a_{22}$ Va = 0 V1 = 0 Vu = 0 Vx = 1
$a_{12}$ Va = 0 V1 = 0 Vu = 0 Vx = 1
$a_{41}$ $l_{41}$ Va = 0 V1 = 1 $u_{11}$ $a_{41}/u_{11} = l_{41}$ Vu = 1 Vx = 0
$a_{31}$ Va = 0 V1 = 0 Vu = 0 Vx = 1
$a_{21}$ Va = 0 V1 = 0 Vu = 0 Vx = 1
$a_{11}$ Va = 0 V1 = 0 Vu = 0 Vx = 1

Fig. 2.18.4.

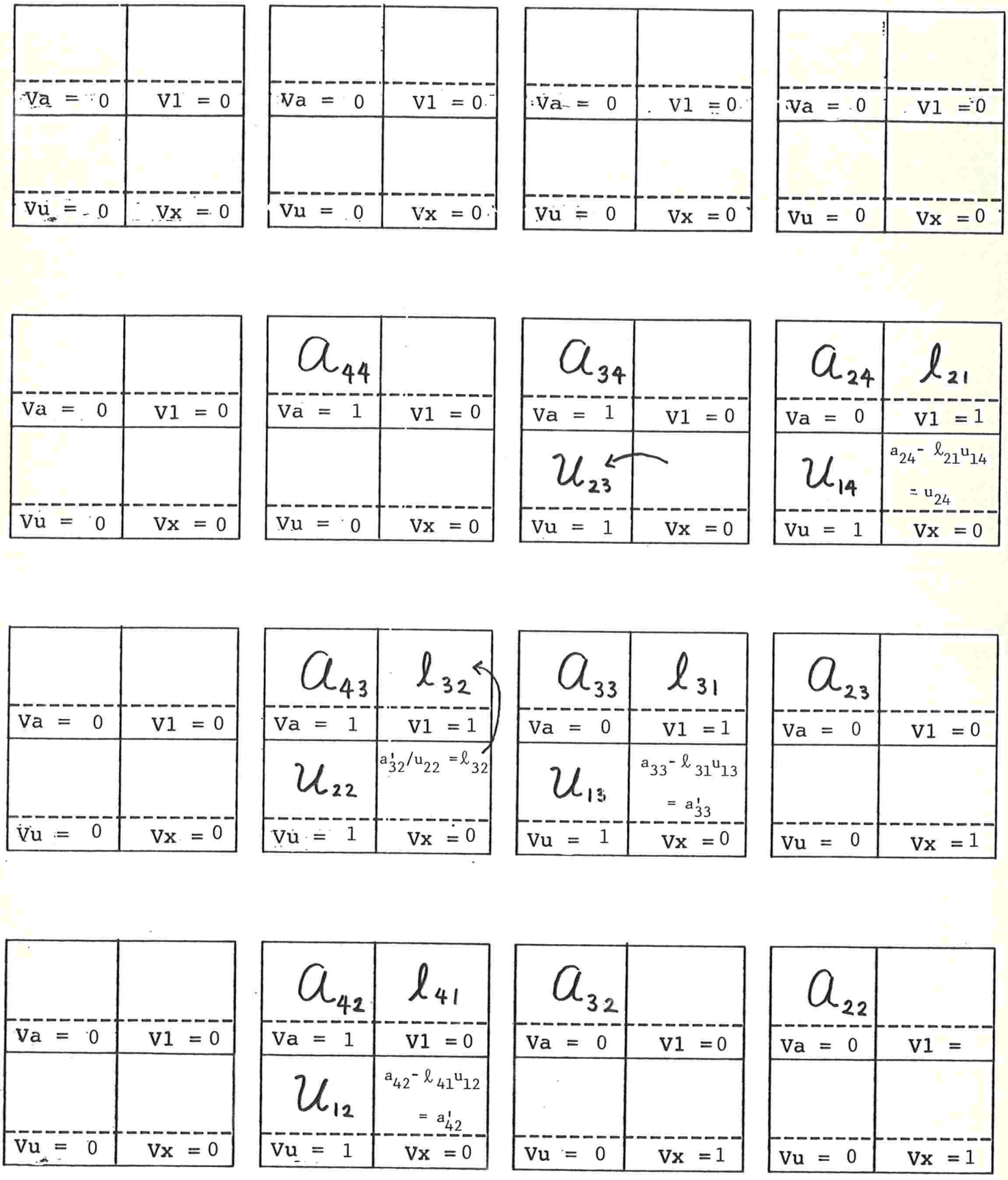

Va = 0 V1 = 0 Vu = 0 Vx = 0
Va = 0 V1 = 0 Vu = 0 Vx = 0
Va = 0 V1 = 0 Vu = 0 Vx = 0
Va = 0 V1 = 0 Vu = 0 Vx = 0
Va = 0 V1 = 0 Vu = 0 Vx = 0
a44 Va = 1 V1 = 0 Vu = 0 Vx = 0
a34 Va = 1 V1 = 0 u23 Vu = 1 Vx = 0
a24 l21 Va = 0 V1 = 1 u14 a24 - l21u14 = u24 Vu = 1 Vx = 0
Va = 0 V1 = 0 Vu = 0 Vx = 0
a43 l32 Va = 1 V1 = 1 u22 a'32/u22 = l32 Vu = 1 Vx = 0
a33 l31 Va = 0 V1 = 1 u13 a33 - l31u13 = a'33 Vu = 1 Vx = 0
a23 Va = 0 V1 = 0 Vu = 0 Vx = 1
Va = 0 V1 = 0 Vu = 0 Vx = 0
a42 l41 Va = 1 V1 = 0 u12 a42 - l41u12 = a'42 Vu = 1 Vx = 0
a32 Va = 0 V1 = 0 Vu = 0 Vx = 1
a22 Va = 0 V1 = Vu = 0 Vx = 1

Fig. 2.18.5.

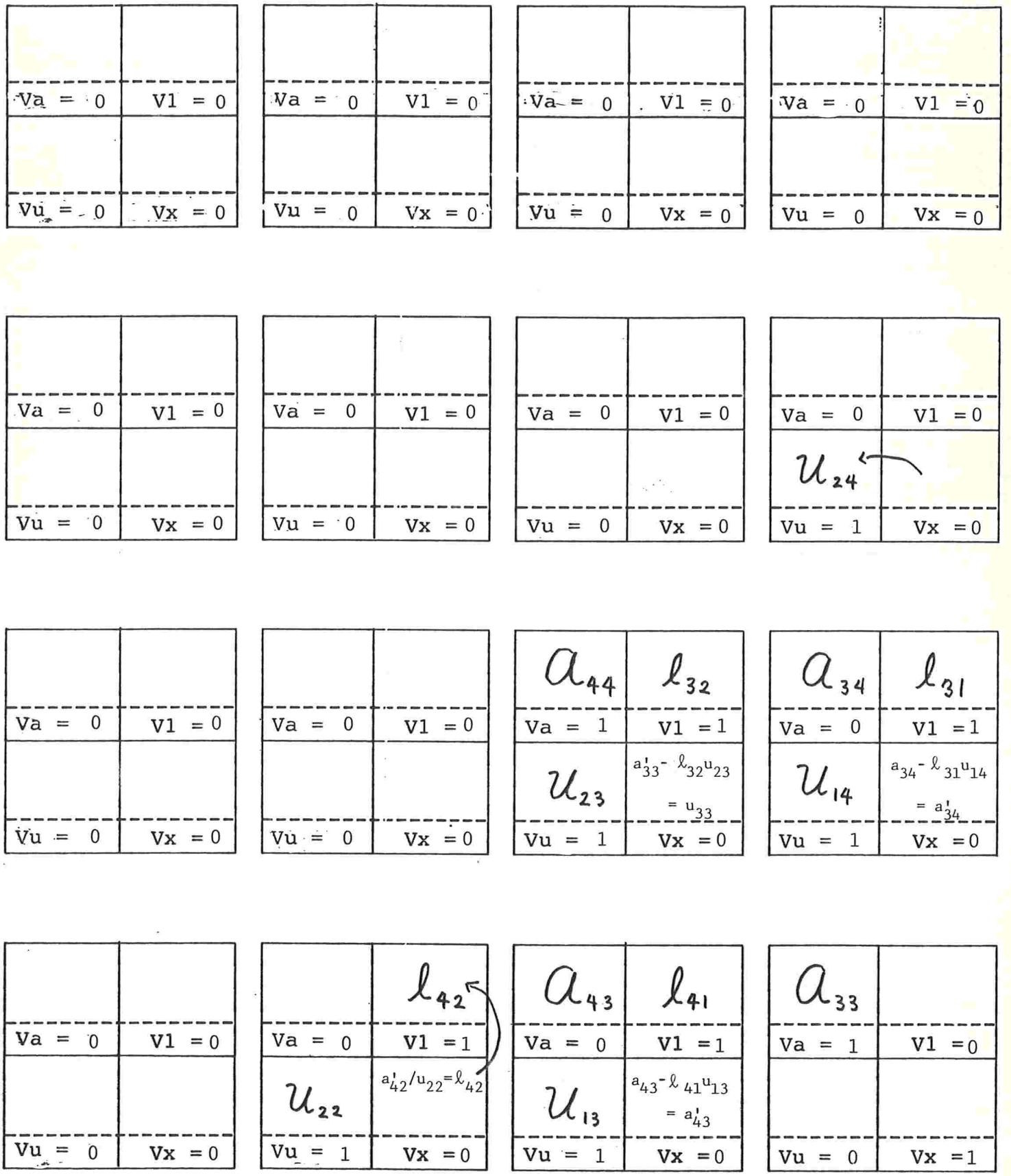

Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
u24
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
a44
l32
Va = 1
V1 = 1
u23
a'33 - l32u23 = u33
Vu = 1
Vx = 0
a34
l31
Va = 0
V1 = 1
u14
a34 - l31u14 = a'34
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
l42
Va = 0
V1 = 1
u22
a'42/u22 = l42
Vu = 1
Vx = 0
a43
l41
Va = 0
V1 = 1
u13
a43 - l41u13 = a'43
Vu = 1
Vx = 0
a33
Va = 1
V1 = 0
Vu = 0
Vx = 1

Fig. 2.18.6.

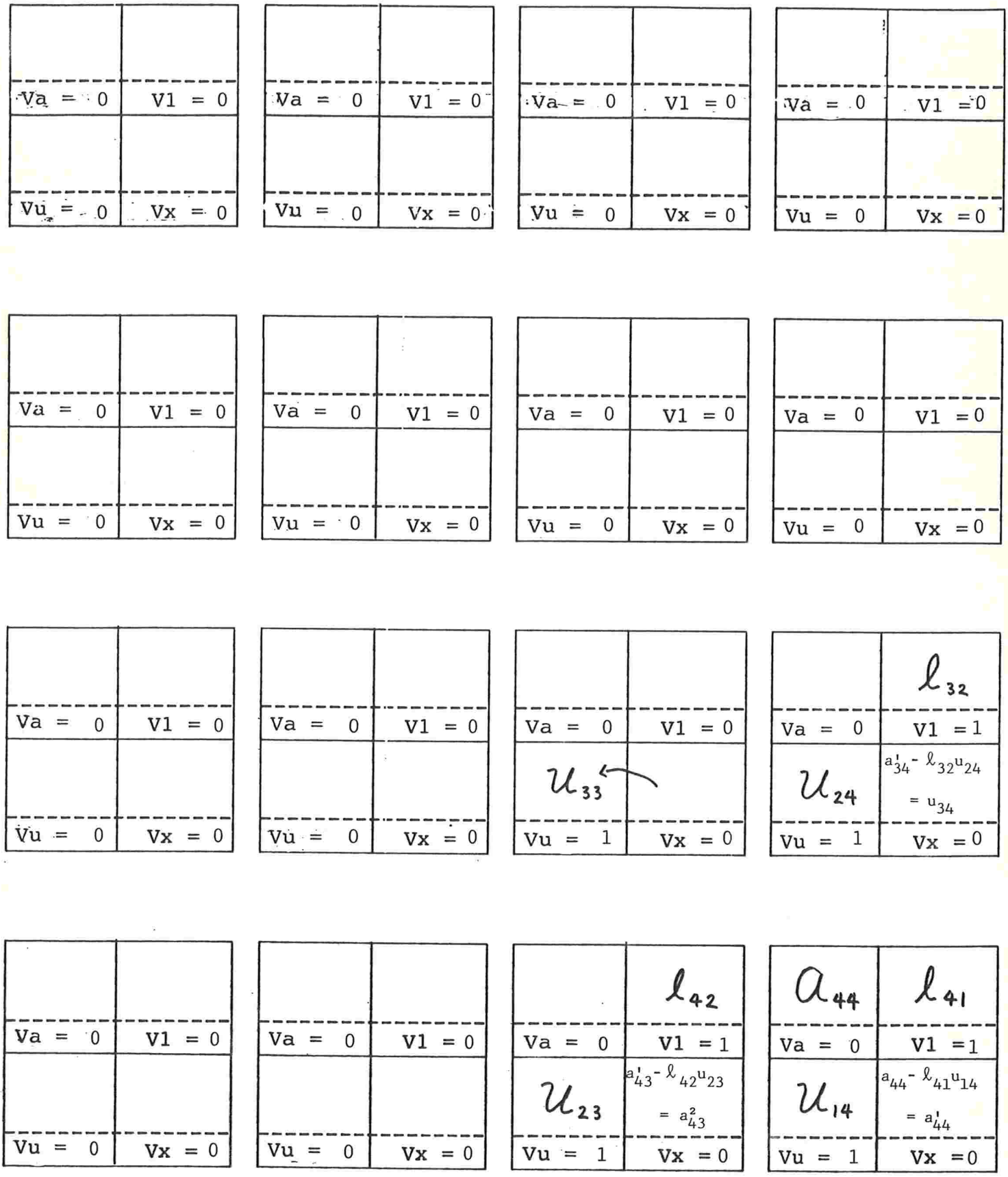
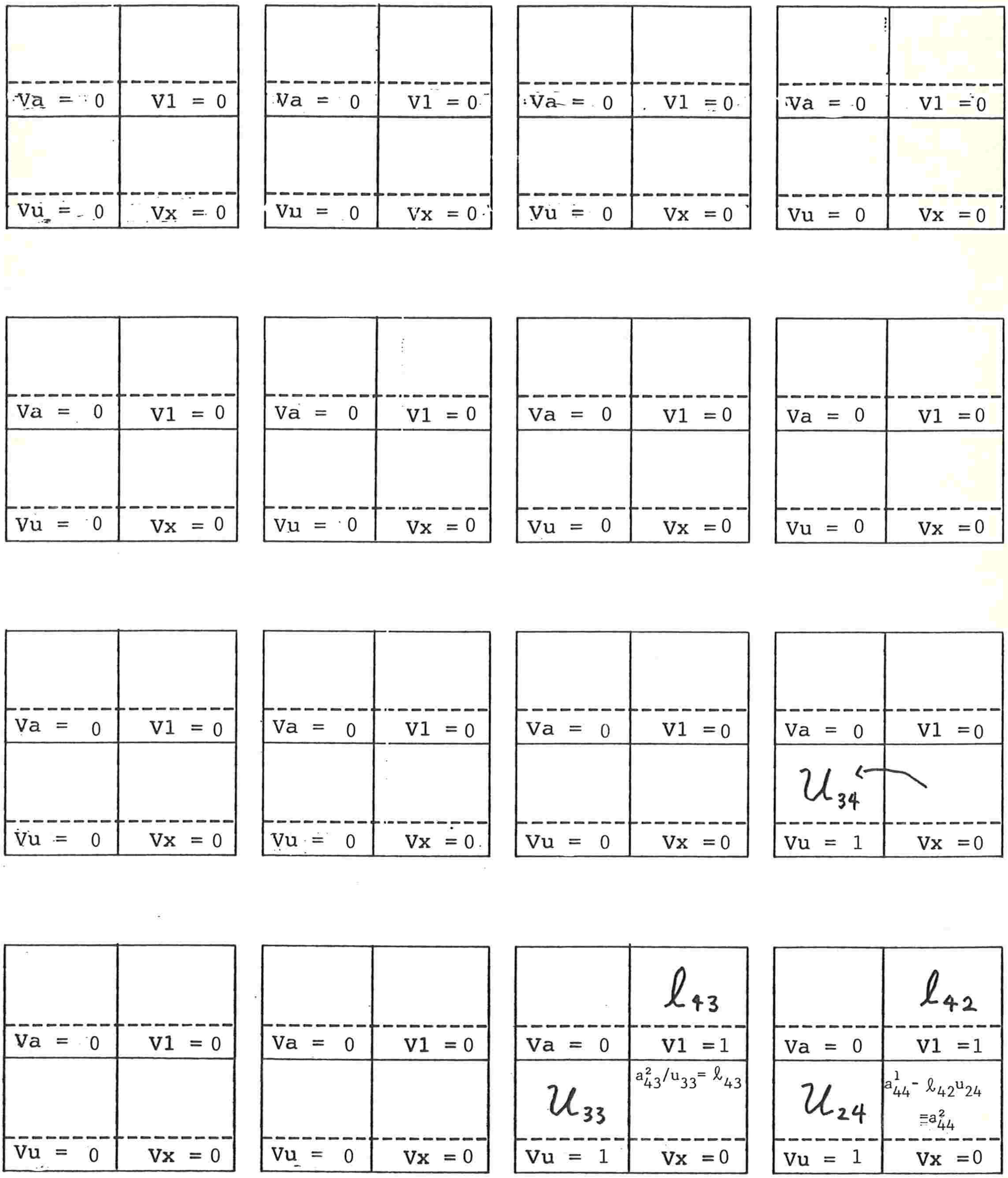

Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 1
Vu = 1
Vx = 0
$u_{33}$
$l_{32}$
$a'_{34} - l_{32}u_{24} = u_{34}$
$u_{24}$
$l_{42}$
$a'_{43} - l_{42}u_{23} = a^2_{43}$
$u_{23}$
$a_{44}$
$l_{41}$
$a_{44} - l_{41}u_{14} = a'_{44}$
$u_{14}$

Fig. 2.18.7.



Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
$u_{34}$
Vu = 1
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
$\ell_{43}$
Va = 0
V1 = 1
$a^2_{43}/u_{33} = \ell_{43}$
$u_{33}$
Vu = 1
Vx = 0
$\ell_{42}$
Va = 0
V1 = 1
$a^1_{44} - \ell_{42}u_{24}$
$= a^2_{44}$
$u_{24}$
Vu = 1
Vx = 0


Fig. 2.18.8.

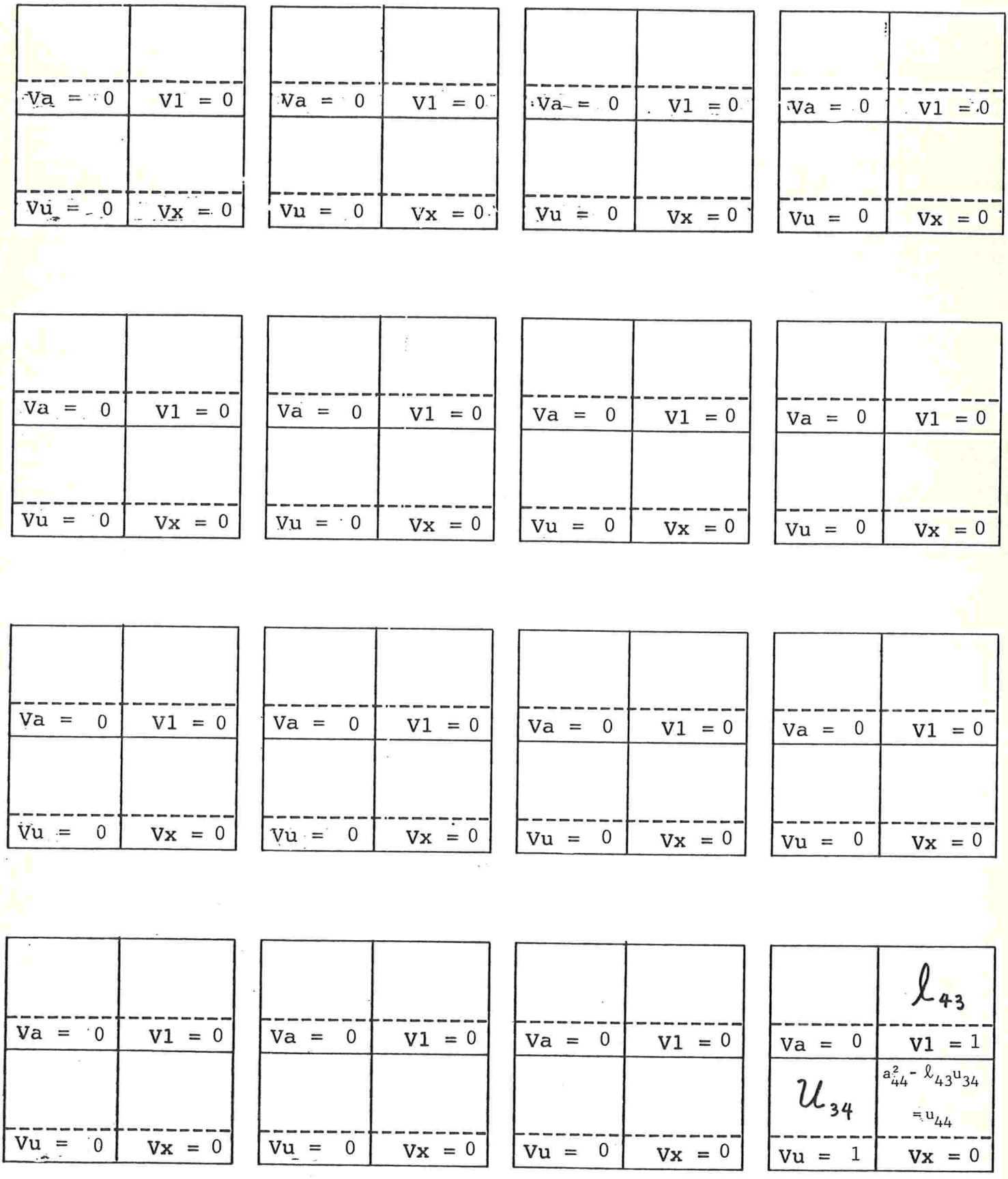

Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 1
Vu = 1
Vx = 0
$l_{43}$
$u_{34}$
$a_{44}^2 - l_{43}u_{34} = u_{44}$

Fig. 2.18.9.

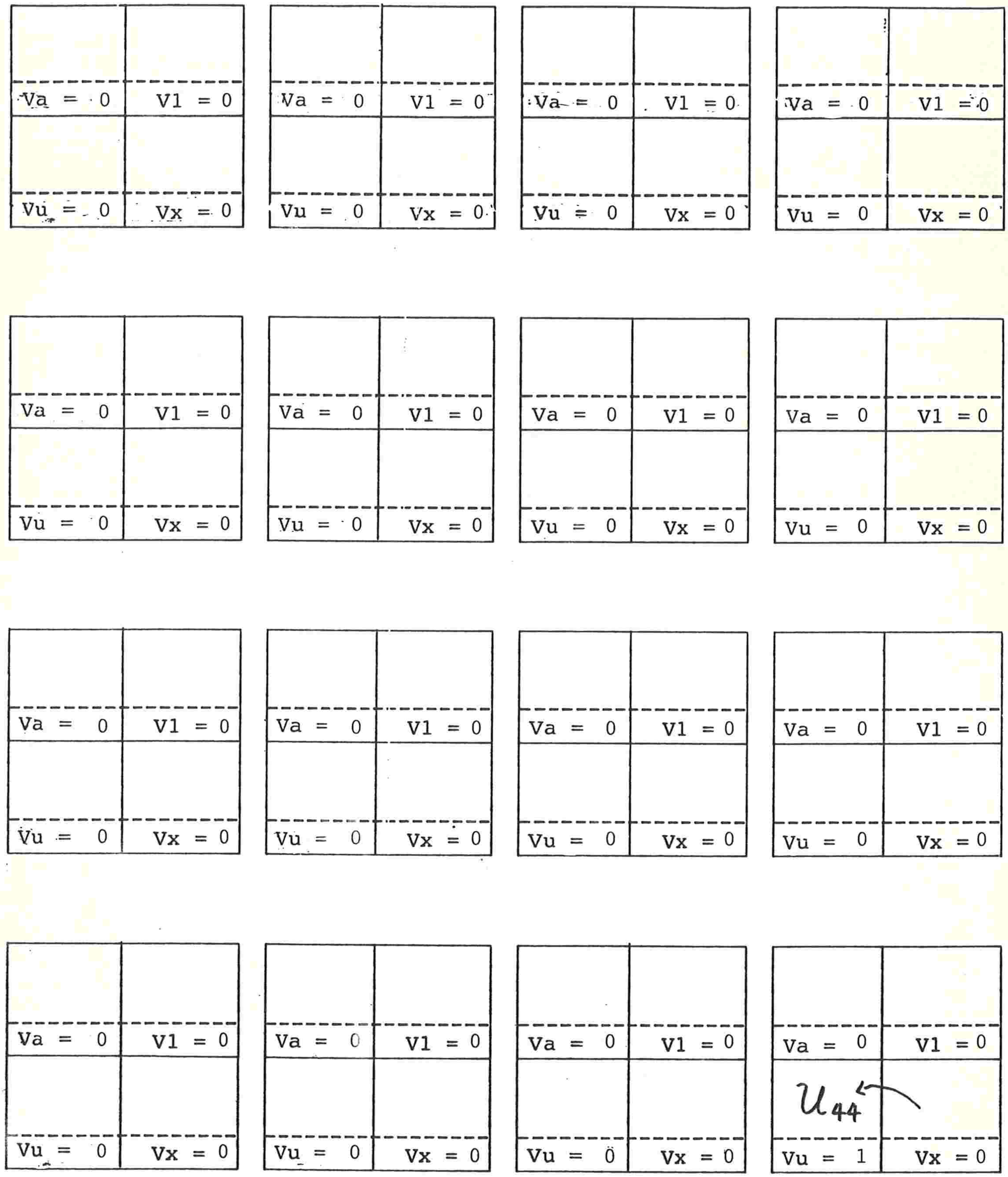

Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
Vu = 0
Vx = 0
Va = 0
V1 = 0
U44
Vu = 1
Vx = 0

Fig. 2.18.10.

## 2.5. **Outros exemplos**

Vimos 4 exemplos de sistemas sistólicos para problemas matriciais. Aqui mencionaremos brevemente outros exemplos para mostrar a potencialidade do sistema sistólico.

a) Convolução

Devido à quantidade de computações envolvidas e o curto tempo de resposta exigido, a convolução usada em processamento de imagens e sinais é um bom exemplo para ser resolvido pela abordagem sistólica.

Um sistema sistólico para convolução bi-dimensional foi desenvolvido e implementado em pastilha por Kung e Song [82].

Outros exemplos para a convolução são encontrados em Kung, Ruane e Yen [81] e Robert e Tchuente [86]

b) Algoritmo combinatório para programação dinâmica

Usa-se o sistema sistólico em disposição triangular no projeto proposto por Guibas, Kung e Thompson [79].

c) Reconhecimento de Padrão

Foster e Kung projetaram uma pastilha implementando um sistema sistólico em disposição linear para o reconhecimento de um padrão de bits (Foster e Kung [80])

d) Busca

Usa-se um sistema sistólico em disposição de árvore binária (Fig. 2.21) no projeto proposto por Bentley e Kung [79].

c) Ordenação

O sistema sistólico (Fig. 2.21) é usado também para problemas de ordenação. Algumas soluções sistólicas foram propostas por Song [81].

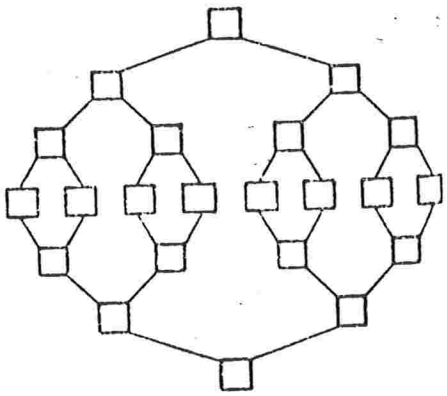


Fig. 2.21

oOo

## §.3 Motivação para Sistematizar a Geração de Algoritmos Sistólicos

Desde o surgimento do Algoritmo sistólico para a multiplicação de matrizes, proposto por Kung e Leiserson, muitos pesquisadores ficaram fascinados com a engenhosidade do algoritmo e começaram a tentar a decomposição de outros algoritmos em algoritmos sistólicos.

Na seção anterior ilustramos apenas alguns exemplos. No decorrer dos anos, muitos projetos sistólicos foram propostos para os mais variados problemas.

Passada a primeira fase em que os algoritmos sistólicos eram concebidos de maneira "ad hoc", requerendo bastante criatividade e paciência de seus autores, uma pergunta natural era a seguinte: Será que não haveria uma maneira mais sistemática e formal para a derivação ou síntese desses algorítmos?

O trabalho pioneiro neste sentido é o trabalho de Fortes & Moldovan [85], surgiram também outros trabalhos propondo método para o desenvolvimento formal de algoritmo sistólico, como Ipsen & Delosme [86], Quinton [87] e Li & Wah [85].

Nos próximos capítulos, seguiremos de perto as linhas de trabalho de Fortes & Maldovan dando ênfase na sua interpretação geométrica.

oOo

# Capítulo 3



# Deteção de Paralelismo

Na seção 1, vamos introduzir algumas definições básicas e usando estas definições, a seção 2 mostra alguns resultados sobre detecção de paralelismo.

Os resultados da seção 3 serão de fundamental importância para transformação de algoritmos a ser tratada no Capítulo 4.

## §.1 Definições

Neste trabalho, estamos interessados em algoritmos expressos como "loops" ou malhas encaixadas da seguinte forma:

for $I_1 = N_1$ to $M_1$ do
  for $I_2 = N_2$ to $M_2$ do
    .
     .
      .
       for $I_t = N_t$ to $M_t$ do
        begin

$$a_1(I_1,\ldots I_t) = f_1[a_1(I_1-d^1_{11},\ldots I_t-d^1_{t1}),\ldots a_s(I_1-d^1_{1s},\ldots I_t-d^1_{ts})];$$
$$a_2(I_1,\ldots I_t) = f_2[a_1(I_1-d^2_{11},\ldots I_t-d^2_{t1}),\ldots a_s(I_1-d^2_{1s},\ldots I_t-d^2_{ts})];$$
$$\vdots$$
$$a_s(I_1,\ldots I_t) = f_s[a_1(I_1-d^s_{11},\ldots I_t-d^s_{t1}),\ldots a_s(I_1-d^s_{1s},\ldots I_t-d^s_{ts})];$$

        end

onde, para i = 1,...t:

$I_{i's}$ são índices de malha

$N_i$, $M_i \in \mathbb{Z}$ com $N_i < M_i$

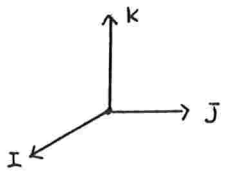
$a_i$'s são variáveis.

$f_i$'s são operações aritméticas envolvendo $a_i$'s (eventualmente algumas delas podem não participar nas operações) e

$d_{ik}^{j}$ são constantes inteiras para i = 1,...t , j = 1,...s e k= 1,..S

Em geral os índices ou coordenadas usados neste trabalho serão $i_1, i_2, \ldots, i_t$ ou $j_1, j_2, \ldots j_t$. Os valores das coordenadas serão $p_1, p_2, \ldots, p_t$. No caso de 3 malhas encaixadas, como ocorre em muitos exemplos, convencionamos usar os índices K,I,J nesta ordem (i.e. , $i_1 = K_1$ ; $i_2 = I$ e $i_3 = J$). A sua representação geométrica é como na Figura abaixo e a coordenada $(p_1, p_2, p_3)$ significa $K = p_1$ ; $I = p_2$ $J = p_3$.

Exemplo 3.1. Algoritmo $A_1$:




```
for K:= 1 to 10 do
 for I:= 1 to 5 do
  for J:= 0 to 8 do
      begin
        A(K,I,J): = A(K-1,I-1,J) + B(K,I,J-1)*C(K,I-1,J):
        B(K,I,J): = 5*B(K,I-1,J-1);
        C(K,I,J): = 2*(A(K,I,J-1)+3*C(K-1,I-1,J)-B(K,I,J));
      end ;
```

Definição 3.1. Seja A um algoritmo então $CI = \{(I_1, I_2, \ldots I_t) \in \mathbb{Z}^t$ / $N_K \leq I_K \leq M_K$, $1 \leq K \leq t\}$ é chamado *conjunto de índices* do algoritmo A.

Exemplo 3.2. No caso de 3 malhas encaixadas, $\mathbb{CI}$ é o conjunto dos pontos de coordenadas inteiras de um paralelepípedo retangular no $\mathbb{R}^3$ cujas arestas são paralelas aos 3 eixos

```
Algorítmo A2: for K = 0 to 3 do
                  for I: = 0 to 4 do
                     for J: = 0 to 8 do
                        begin
                         "corpo do algoritmo A2"
                        end;
```

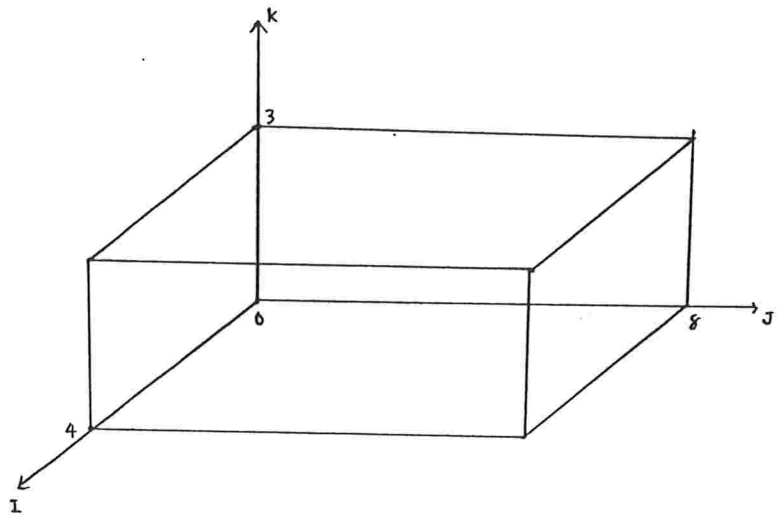



Fig. 3.1. paralepepípedo retangular que contem $\mathbb{CI}$ de A no ex. 3.2.

Definição 3.2. Seja A um algoritmo e $\mathbb{CI}$ seu conjunto de índices. A *ordem lexicográfica* de execução de A é uma ordem total sobre $\mathbb{CI}$ tal que:

$$(I_1, I_2, \ldots, I_t) < (I'_1, I'_2, \ldots I'_t)$$

se e somente se

$$\exists K \ (1 \leq k \leq t) \begin{cases} I_h = I'_h \quad \text{para } h < k \\ \\ I_k < I'_k \end{cases}$$

Numa máquina sequencial, a ordem natural para executar malhas encaixadas é a lexicográfica.

Podemos realizar a execução de um algoritmo A do seguinte modo: dado o algoritmo A e seu $\mathbb{CI}$, percorremos os pontos de $\mathbb{CI}$ na ordem lexicográfica e em cada ponto $(I_1, I_2, \ldots I_t)$ nós executamos a computação do corpo de A, i.e.,calculamos $a_1[I_1, I_2, \ldots I_t)$, $a_2(I_1, I_2, \ldots I_t)$ $\ldots a_s(I_1, I_2, \ldots I_t)$.

Exemplo 3.3. Algoritmo $A_3$:

```
for  K: = to 2 do
   for I: = 0 to 2 do
      for J: = 0 to 2 do
       begin
         a(K,I,J): = a(K,I-1,J);
         b(K,I,J): = b(K,I,J-1);
         C(K,I,J): = C(K-1,I,J) + a(K,I,J)*b(K,I,J)
       end;
```

A Fig. 3.2.1 mostra os pontos de $\mathbb{CI}$ para A e na Fig. 3.2.2. estes pontos são marcados na sua ordem lexicográfica.

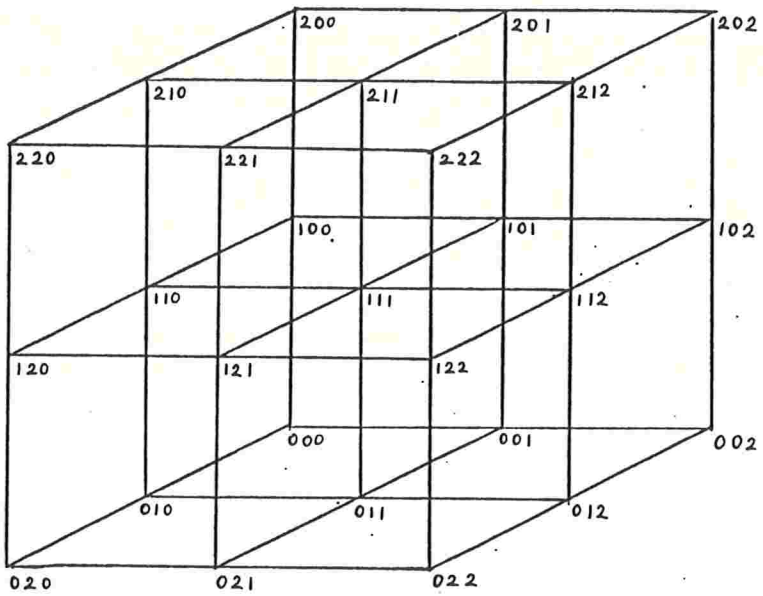



Fig. 3.2.1.

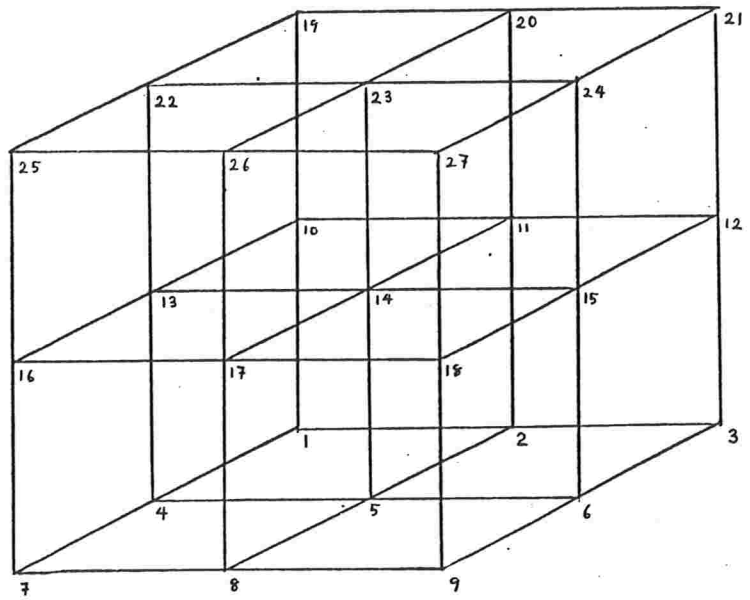



Fig. 3.2.2.

Obs.: Para evitar que os índices completos dificultem a compreensão das idéias descritas nas definições e exemplos, usaremos, no decorrer do capítulo, uma notação mais simplificada para os índices, sempre que for possível.

Definição 3.3. Seja A um algoritmo e considere a seguinte computação no corpo de A.

$$a(I_1,I_2,\ldots I_t) = f[\ldots,b(I_1-d_1,I_2-d_2,\ldots I_t-d_t),\ldots]$$

Então dizemos que

$$a(I_1,I_2,\ldots I_t) \quad \textit{depende de} \quad b(I_1-d_1,I_2-d_2\ldots I_t-d_t).$$

Exemplo: No ex.3.3.

a(K,I,J) depende de a(K,I-1,J),

b(K,I,J) depende de b(K,I,J-1)

e

C(K,I,J) depende de C(K-1,I,J), a(K,I,J) e b(K,I,J).

Veremos mais adiante que não estamos preocupados com dependência do tipo C(K,I,J) em relação a a(K,I,J) e b(K,I,J).

Definição 3.4. Dizemos que a dependência de $a(I_1,I_2,\ldots I_t)$ em relação a $b(I'_1,I'_2,\ldots I'_t)$ é uma *anti-dependência* se

$$(I_1,I_2,\ldots I_t) < (I'_1,I'_2,\ldots I'_t)$$

em ordem lexicográfica.

Exemplo 3.4. Algoritmo $A_4$:

```
for I: = 1 to 3 do
   for J: = 1 to 3 do
      begin
        a(I,J): = 3*a(I,J-1);
        b(I,J): = a(I,J+1)+2*b(I-1,J);
      end;
```

Neste exemplo, b(I,J) tem anti-dependência em relação a a(I,J+1).

Vamos impor a condição de que daqui em diante o algoritmo A é tal que não tem anti-dependência.

Definição 3.4. Suponha que $a(I_1,I_2,\ldots I_t)$ depende de $b(I_1-d_1,I_2-d_2,\ldots I_t-d_t)$, então o vetor $d = (d_1,d_2,\ldots d_t)^T$ é chamado de *vetor de dependência*.

Podemos ilustrar a dependência imposta pelo vetor $d = (d_1 d_2,\ldots d_t)^T$ para cada ponto $(I_1,I_2,\ldots I_t)$ de CI, consideramos um vetor $d = (d_1 d_2,\ldots d_t)^T$ que sai de $(I_1-d_1,I_2-d_2,\ldots,I_t-d_t)$ e chega neste ponto. Isso é mostrado no exemplo a seguir.

Exemplo 3.5. Algoritmo $A_5$:

```
for  I: = 0 to 3 do
   for J: = 0 to 3 do
      a(I,J): = a(I-1,J-1)+a(I-1,J)*a(I,J-1)
```

Como a(I,J) depende de a(I-1,J-1), a(I-1,J) e a(I,J-1) temos 3 vetores de dependência:

$$d_1 = \binom{1}{1} \ , \ d_2 = \binom{1}{0} \ , \ d_3 = \binom{0}{1}$$

A fig. 3.3. ilustra a situação

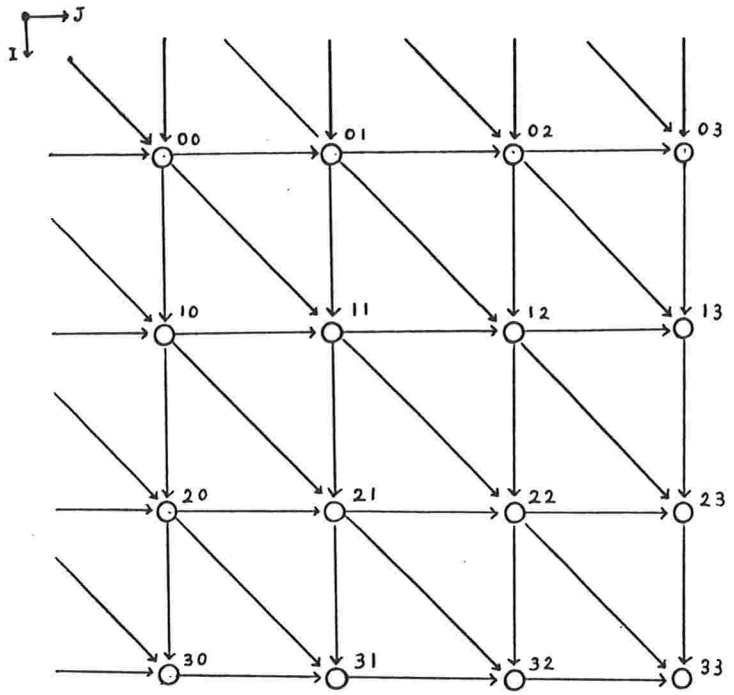



Fig. 3.3.

Observemos os vetores vindos fora de $\mathbb{I}$. Eles representam as entradas de valores não computadas em A mas sim fornecidas ao algoritmo $A_5$

Por exemplo,a(1,0) recebe a(0,-1) e a(1,-1) que são lidos ou computados antes de começar $A_5$.

Exemplo: para o ex.3.3. temos 3 vetores de dependência:

$d_1 = \begin{pmatrix}1\\0\\0\end{pmatrix}$ (devido a dependência de c(K,I,J) em relação a c(K-1,I,J)),

$d_2 = \begin{pmatrix}0\\1\\0\end{pmatrix}$ (devido a dependência de a(K,I,J) em relação a a(K,I-1,J)),

$d_3 = \begin{pmatrix}0\\0\\1\end{pmatrix}$ (devido a dependência de b(K,I,J) em relação a b(K,I,J-1)).

A Fig. 3.4. mostra os vetores de dependência para este exemplo. Note que a dependência de c(K,I,J) em relação a a(K,I,J) vai resultar em vetor de dependência nula assim como a dependência de c(K,I,J) em relação a b(K,I,J).

Como veremos mais adiante, nosso interesse se concentra de onde vem os dados necessários para a computação em cada ponto de $\mathbb{I}$; portanto a dependência dentro do próprio ponto em si não é relevante.

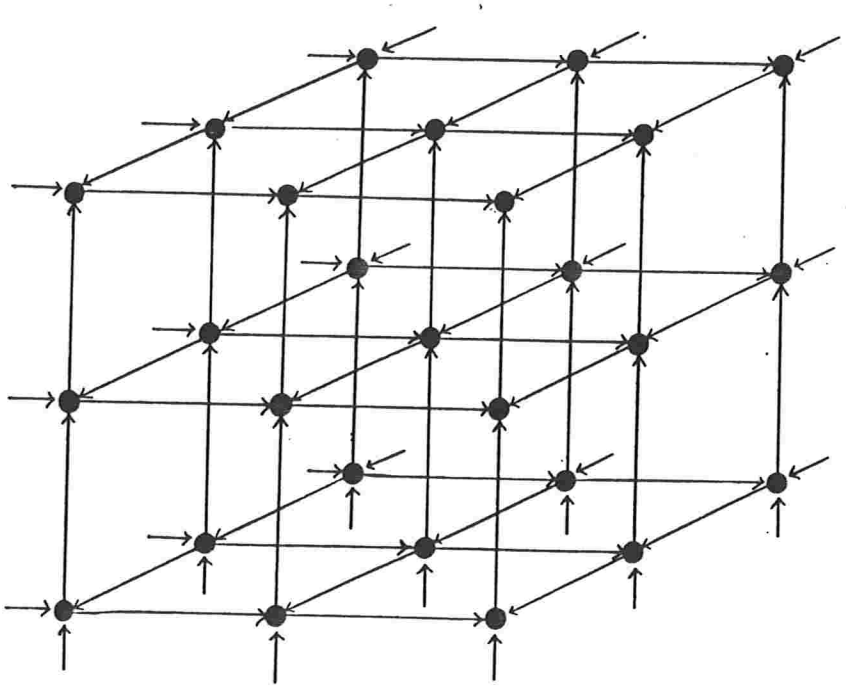


Fig. 3.4.

Pela maneira como foram definidas no começo do capítulo, as computações feitas em cada ponto de $\mathbb{C}I$ são idênticas em todos os pontos de $\mathbb{C}I$ com seus valores diferentes de $I_1, I_2, \ldots I_t$ assumidos em cada ponto.

No capítulo 4, veremos que a consequência deste fato é de que as células básicas que compõem o sistema sistólico são todas idênticas.

Também é óbvia a homogeneidade da maneira como os vetores de dependência chegam para cada ponto de $\mathbb{C}I$ e este fato é ilustrado nas Figs. 3.3. e 3.4.

Também veremos que o fato de os vetores de dependência serem os mesmos em todos os pontos de $\mathbb{C}I$, como exemplificado nas Fig. 3.3 e Fig. 3.4 tem como consequência a regularidade de ligações entre as células básicas do sistema sistólico correspondente.

oOo

# §.2 Deteção de Paralelismo

Definição 3.5. Seja A um algoritmo, então chama-se *matriz completa de dependência* para A, a seguinte matriz:

$$D_c = \begin{array}{c} \overbrace{\qquad a_1 \qquad}^{} \quad \overbrace{\qquad\qquad a_2 \qquad\qquad}^{} \quad \cdots \quad \overbrace{\qquad\qquad a_s \qquad\qquad}^{} \\ \left[\begin{array}{ccccccccccc} d^1_{11} & d^1_{12} & \cdots & d^1_{1s} & d^2_{11} & d^2_{12} & \cdots & d^2_{1s} & \cdots & d^s_{11} & d^s_{12} & \cdots & d^s_{1s} \\ \\ d^1_{t1} & d^1_{t2} & \cdots & d^1_{ts} & d^2_{t1} & d^2_{t2} & \cdots & d^2_{ts} & \cdots & d^s_{t1} & d^s_{t2} & \cdots & d^s_{ts} \\ \underbrace{\quad}_{a_1} & \underbrace{\quad}_{a_2} & & \underbrace{\quad}_{a_s} & \underbrace{\quad}_{a_1} & \underbrace{\quad}_{a_2} & & \underbrace{\quad}_{a_s} & \cdots & \underbrace{\quad}_{a_1} & \underbrace{\quad}_{a_2} & & \underbrace{\quad}_{a_s} \end{array}\right] \end{array}$$

Os rótulos superiores identificam as variáveis do lado esquerdo de comando de atribuição do corpo de A; os rótulos inferiores identificam as variáveis do lado direito dos comandos, i.e., as variáveis envolvidas em computações de cada variável, e as colunas são seus respectivos vetores de dependência.

Exemplo: Para o algoritmo do ex. 3.1. a matriz completa de dependência será:

$$D_c = \begin{array}{c} \overbrace{\quad A \quad}^{} \quad \overbrace{\quad B \quad}^{} \quad \overbrace{\qquad C \qquad}^{} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ \underbrace{\quad}_{A} & \underbrace{\quad}_{B} & \underbrace{\quad}_{C} & \underbrace{\quad}_{B} & \underbrace{\quad}_{A} & \underbrace{\quad}_{C} & \underbrace{\quad}_{B} \end{array}\right] \end{array}$$

Para ex. 3.6. $\mathbb{CI}$ e $D_c$ são idênticos aos do ex. 3.3.

A única diferença está nas computações feitas em cada ponto de $\mathbb{CI}$.

Como veremos no Cap. IV, o algoritmo do ex.3.3. e o algoritmo do ex. 3.6. resultam em sistemas sistólicos semelhantes, i.e., mesmas configurações para disposição e interconexão de células básicas exceto estruturas internas das células básicas para executar computações diferentes.

Podemos afirmar que, do ponto de vista de detecção de paralelismo e sua consequente construção de sistema sistólico, o algoritmo do ex. 3.3. e o algoritmo do ex. 3.6 são "geometricamente equivalentes".

Como consequência, podemos adotar a seguinte postura:

Daqui em diante, iremos descrever muitas vezes um algoritmo A especificando apenas $\mathbb{CI}$ e $D_c$, sem dar toda sua especificação.

Observação: Para um algoritmo A dado, escrever sua matriz completa de dependência $D_c$ não só é um pouco tedioso como também é desnecessário para a análise de paralelismo, pois o que nos interessa é apenas como um ponto de $\mathbb{CI}$ está relacionado com outros pontos de $\mathbb{CI}$ via vetores de dependência.

Faremos então as seguintes simplificações

1. Omissão de rótulos superior e inferior:

Nosso interesse é de onde vêm as variáveis envolvidas em computações num ponto de $\mathbb{C}$ e, sendo irrelevantes, quais são essas variáveis.

Por exemplo, no ex. 3.1. a primeira coluna $\begin{pmatrix}1\\1\\0\end{pmatrix}$ da sua matriz completa de dependência tem rótulo superior A e rótulo inferior A, isto significa que em cada ponto (K,I,J) de $\mathbb{CI}$ A(K,I,J) necessita de A(K-1,I-1,J).

Mas do ponto de vista de detecção de paralelismo, a única coisa que interessa é a seguinte: computações no ponto (K,I,J) dependem de computações feitas em (K-1,I-1,J).

2. Omissão de coluna nula (i.e. vetor nulo de dependência):

O motivo é o mesmo do item 1., por exemplo, no ex. 3.3. a última coluna mostra que a computação de c(K,I,J) necessita de b(K,I,J) em cada ponto (K,I,J) de seu $\mathbb{C}$I. Dependências desse tipo, envolvendo o mesmo ponto de $\mathbb{C}$I são importantes quando a estrutura interna de célula básica é projetada.

3. Omissão de colunas idênticas:

No $D_C$ do ex. 3.1, a primeira e a sexta colunas são idênticas mostrando a chegada de A(K-1,I-1,J) e $\mathbb{C}$(K-1,I-1,J) ao ponto (K,I,J) a partir do ponto (K-1,I-1,J).

Novamente, nosso interesse é o fato de que existe fluxo de dados do ponto (K-1,I-1,J) para o ponto (K,I,J) e não de quantas variáveis sejam tais fluxos.

Apenas quando projetamos as ligações entre as células é que precisamos saber que de (K-1,I-1,J) temos duas ligações para (K,I,J). Na detecção de paralelismo, este fato não influencia em nada.

Há, entretanto, uma distinção conceitual entre essa simplificação e as 1. e 2. Ao contrário de duas omissões anteriores que dizem respeito mais a estrutura interna da célula básica do sistema sistólico, a simplificação 3. ignora o número de ligações entre células, o que é mais relevante que as anteriores.

Definição 3.6. Seja A um algoritmo e $D_c$ sua matriz completa de dependência como foi definida em def. 3.5. Então chama-se *matriz de dependência* para A a matriz D obtida com as simplificações acima.

Ex.: As três matrizes de dependência anteriores serão as seguintes:

para o ex. 3.1.

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

para o ex. 3.3.

$$D = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

para o ex. 3.5.

$$D = \begin{pmatrix} 1 & 1 & 0 \\ & & \\ 1 & 0 & 1 \end{pmatrix}.$$

Dado um algoritmo A, seu $\mathbb{CI}$ e D sugerem a definição de um grafo orientado de computação.

Definição 3.7: Dados um algoritmo A e seu $\mathbb{CI}$, o grafo orientado de computação para A é o seguinte:

Os pontos de $\mathbb{CI}$ serão vértices deste grafo orientado e existe uma aresta orientada de p para q, onde q = p+d, sendo d um vetor de dependência de D.

Na Figura 3.5,temos o grafo orientado correspondente ao ex. 3.5. Comparando com a Figura 3.3, notamos que não estão presentes aquelas flechas que representavam as entradas de dados. Tal informação, entretanto, pode ser recuperada sem dificuldade, uma vez que os vetores de dependência são constantes em todos os pontos.

Neste grafo, quaisquer dois vértices que não são ligados por um passeio, correspondem a dois pontos independentes em relação à sua computação, i.e., quaisquer dois pontos que não estão no fecho transitivo conexo de passeio são executáveis paralelamente.

Observando a Figura 3.5., notamos que (1,0) e (0,1) são pontos de computação independentes enquanto que (1,0) e (2,2) não são pontos de computação independentes.

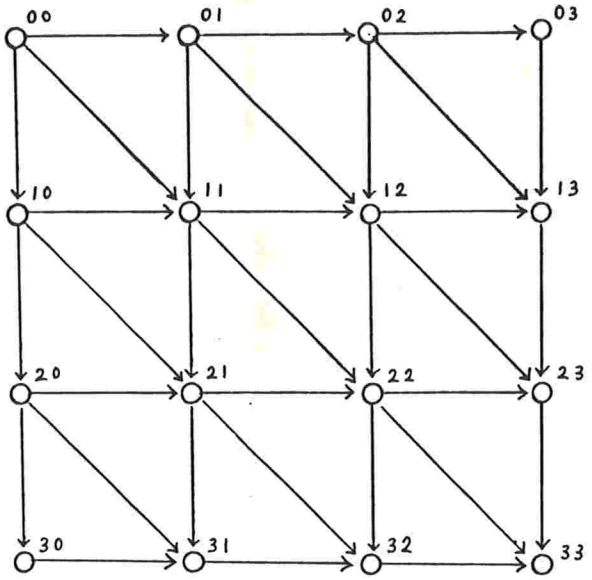



Figura 3.5.

Na Fig. 3.6, temos o grafo orientado de computação correspondente ao ex. 3.3. Os pontos (0,1,0), (1,0,0) e (0,0,1) são pontos de computação independente.

Observemos também que (0,0,0) não pode ser ponto de computação independente com nenhum outro ponto de $\mathbb{CI}$ pois todos os demais pontos dependem dele.

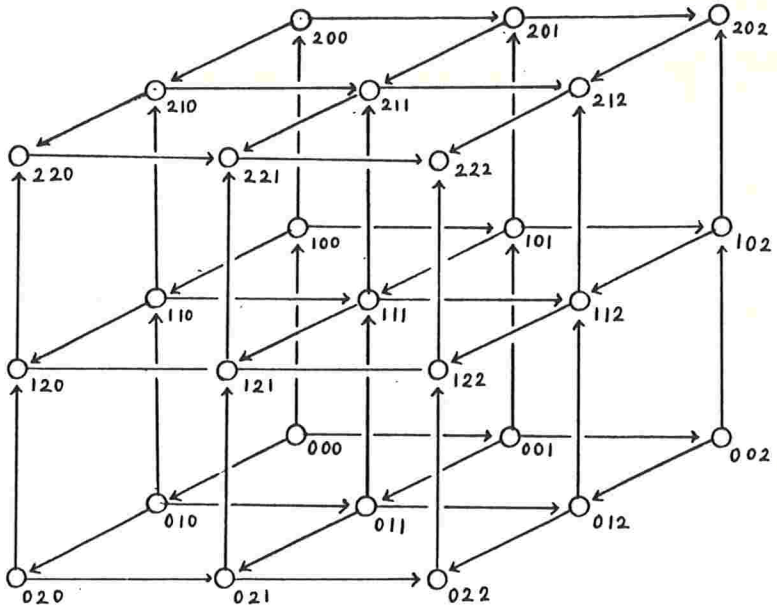



Figura 3.6.

Usemos a tripla (p,d,q) para representar a dependência do ponto q em relação a p via vetor d, i.e. q = p+d.

Definição 3.8: Dados $(p_{i-1}, d_i, p_i)$ tais que $p_i = p_{i-1}+d_i$, para i = 1,2,...,k, dizemos que $(p_o, d_1+d_2+\ldots+d_k, p_k)$ é *composição de dependência* $(p_{i-1}, d_i, p_i)_{1 \leq i \leq k}$, onde o vetor $d_1+d_2+\ldots+d_k$ é a composição dos vetores de dependência $d_1, d_2, \ldots, d_k$.

Podemos agora ampliar o conceito de dependência: na condição acima, dizemos que $p_k$ depende de $p_o$. Dizemos que q depende <u>diretamente</u> de p quando no grafo orientado de computação há uma aresta de p para q.

<u>Definição 3.9</u>: Seja $D_{n\times m}$ uma matriz de dependência para um dado algoritmo A

$$D = \left.\begin{pmatrix} \overbrace{\begin{matrix} d_{ij} \\ \vdots \\ d_{ij} \end{matrix}}^{m} \end{pmatrix}\right\} n$$

jª coluna = vetor de dependência $d_j$

Definimos $\tilde{D}_{n\times\tilde{m}}$ como a matriz cujas colunas são todas as composições possíveis de vetores de dependência no $\mathbb{CI}$.

$$D = \left.\begin{pmatrix} \overbrace{\begin{matrix} \tilde{d}_{1j} \\ \vdots \\ \tilde{d}_{nj} \end{matrix}}^{\tilde{m}} \end{pmatrix}\right\} n$$

$$\text{jª coluna} = \tilde{d}_j = \sum_{k=1}^{m} \alpha_{jk} d_k \ ,$$

$\alpha_{jk}$ é um inteiro não negativo e pelo menos um $\alpha_{jk}$ é diferente de zero.

Ex. para o ex. 3.5.: $\tilde{D} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{pmatrix}$

Ex. para o ex. 3.3.:

$$\tilde{D} = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2 \\ 0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2 \\ 1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2 \end{pmatrix}$$

Então é fácil observar que dados dois pontos p e q em $\mathbb{I}$, q depende de p se e somente se $d = q-p$ é uma coluna de $\tilde{D}$.

Definição 3.10: Na condição acima, dizemos que as computações em p e q são *dependentes*.

Notação: Na condição acima, escrevemos $d \in \tilde{D}$.

Assim, as computações em p e q são dependentes se e somente se $d \in \tilde{D}$ ou $-d \in \tilde{D}$, onde $d = p-q$; em caso contrário. p e q são independentes.

Para saber se as computações em dois pontos p e q são independentes ou não, a inspeção direta de $\tilde{D}$ é pouco prática devido ao seu tamanho. Entretanto, existem certas relações úteis entre D e $\tilde{D}$ que nos permitem detectar as computações independentes inspecionando apenas D. Para isso, necessitamos de alguns preparos.

Definição 3.11: Dado D e $\tilde{D}$ de um algoritmo A, adotamos as seguintes notações:

$L^{o}$ = conjunto de linhas de D cujos elementos são todos nulos,

$L^{+}$ = conjunto de linhas de D cujos elementos são todos positivos,

$L^{-}$ = conjunto de linhas de D cujos elementos são todos negativos,

$L^{>}$ = conjunto de linhas de D cujos elementos são todos não-negativos (mas não todos nulos, i.e. pelo menos um elemento é positivo),

$L^{<}$ = conjunto de linhas de D cujos elementos são todos não positivos, mas não todos nulos,

$L^{T}$ = conjunto de tuplas de linhas de D não-negativos ou não-positivas, cujas colunas são todas não nulas.

Definição 3.12: $\tilde{L}^{o}$, $\tilde{L}^{+}$, $\tilde{L}^{-}$, $\tilde{L}^{<}$, $\tilde{L}^{>}$, $\tilde{L}^{T}$ são definidas para linhas de $\tilde{D}$ analogamente a definição 3.11.

Lema 1:

i) Sejam $\ell_i$ a iª linha de D e $\tilde{\ell}_i$ a iª linha de $\tilde{D}$, então
$\ell_i \in L^*$ se e somente se $\tilde{\ell}_i \in \tilde{L}^*$ , onde $*\in\{0,+,-,<,>\}$ ;

temos ainda

ii) $$\{\ell_{i1},\ell_{i2},\ldots\ell_{it}\} \in L^T$$

se e somente se

$$\{\tilde{\ell}_{i1},\tilde{\ell}_{i2},\ldots,\tilde{\ell}_{it}\} \in \tilde{L}^T$$

Demonstração: seja $\tilde{d}_j$ uma coluna em $\tilde{D}$

$$\tilde{d}_j = \begin{pmatrix} \tilde{d}_{1j} \\ \vdots \\ \tilde{d}_{nj} \end{pmatrix}, \quad \tilde{d}_j = \sum_{k=1}^{m} \alpha_{jk} d_k \quad \text{conforme def. 3.9.}$$

$$d_k = \begin{pmatrix} d_{1k} \\ \vdots \\ d_{nk} \end{pmatrix}$$

assim temos

$$\tilde{d}_{ij} = \sum_{k=1}^{m} \alpha_{jk} d_{ik} \quad \text{com } d_{jk} \geq 0 \ldots\ldots\ldots (1)$$

i) a) $\alpha_{ik} \not= 0$ , $\forall$ k , $\not= \in \{=,>,<,\leqq,\geqq\}$

portanto $\tilde{d}_{ij} \not= 0$ , $\forall$ j por (1)

assim, se $\ell_i \in L^*$ então $\tilde{\ell}_i \in \tilde{L}^*$ onde $* \in \{0,+,-,<,>\}$
exceto que para * = < ou > ainda devemos verificar que os elementos não são todos nulos.

De fato, $d_{ik} \neq 0$ para algum k então pela maneira como foi construido $\tilde{D}$ existe $\bar{k}$ tal que $\tilde{d}_{i\bar{k}} \neq 0$.

b) $\tilde{d}_{ik} \not= 0$ , $\forall$ k, $\not= \in \{=,>,<,\geqq,\leqq\}$

portanto,

$d_{ij} \not= 0$, $\forall$ j, pois a iª linha de D é parte da iª linha de $\tilde{D}$

Se $\tilde{\ell}_i \in \tilde{L}^{>}$ (ou $\tilde{L}^{<}$) deve ocorrer que $\ell_i \in L^{>}$ (ou $L^{<}$) pois do contrário, todos os elementos de $\ell_i$ seriam nulos, ou seja, $\tilde{\ell}_i \in \tilde{L}^{o}$. Teríamos por a) $\tilde{\ell}_i \in \tilde{L}^{o}$, o que é uma contradição.

Portanto, se $\tilde{\ell}_i \in \tilde{L}^*$ então $\ell_i \in L^*$ $* \in \{0,+,-,<,>\}$

ii) decorre imediatamente de i)

ex. para o ex. 3.3. temos:

$$D = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\tilde{D} = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2 \\ 0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2 \\ 1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2 \end{pmatrix}$$

$\ell_1 \in L^{>}$ e $\tilde{\ell}_1 \in \tilde{L}^{>}$; $(\ell_1,\ell_2,\ell_3) \in L^T$ e $(\tilde{\ell}_1,\tilde{\ell}_2,\tilde{\ell}_3) \in \tilde{L}^T$

porém, note-se que $(\ell_i,\ell_j) \notin L^T$ e $(\tilde{\ell}_i,\tilde{\ell}_j) \notin \tilde{L}^T$ para $\forall i,j$, $1 \leq i,j \leq 3$.

O próximo Lema diz que se todas as dependências têm componente nulo ao longo de algumas coordenadas então as computações em pontos que diferem nestas coordenadas são independentes.

Lema 2.

Sejam A, $\mathbb{CI}$ e D como anteriormente e P e P' $\in \mathbb{CI}$.

i) Se P e P' diferem pelo menos no iº componente, tal que $\ell_i \in L^o$, então P e P' são independentes;

ii) Em particular, se P e P' diferem em algum componente i, tal que $\ell_i \in L^o$ (ou alguns componentes i's para os quais $\ell_i \in L^o$) e todos os outros componentes k, tais que $\ell_k \notin L^o$ têm os mesmos valores, então P e P' são independentes.

dem i) Seja $\ell_i \in L^o$ pelo Lema 1, $\tilde{\ell}_i \in \tilde{L}^o$. Assim, a iª linha de $\tilde{D}$ é toda nula. Se p e p' diferem no componente i, o vetor diferença de p e p' têm o 1º componente não nulo. Portanto, não pertence a $\tilde{D}$. Logo, p e p' são independentes.

ii) óbvio.

ex. 3.7. Algoritmo $A_7$:

```
for K: = 1 to 3 do
    for I: = 1 to 3 do
     for J: = 1 to 3 do
    a(K,I,J) = a(K,I-1,J) + 3*a(K-1,I-1,J)
```

$$D = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}$$

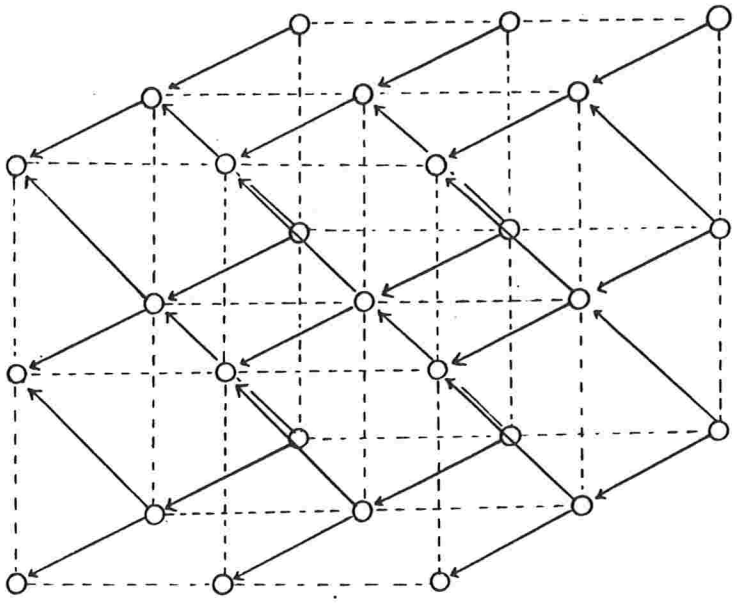


Fig. 3.7 - Grafo orientado de computação de $A_7$ baseado em seu $\mathbb{C}I$ e D

Por i) na Fig. 3.8, cada ponto marcado por 1 é independente de qualquer ponto marcado por 2 ou por 3 . E analogamente, por pontos marcados por 2 e 3 .

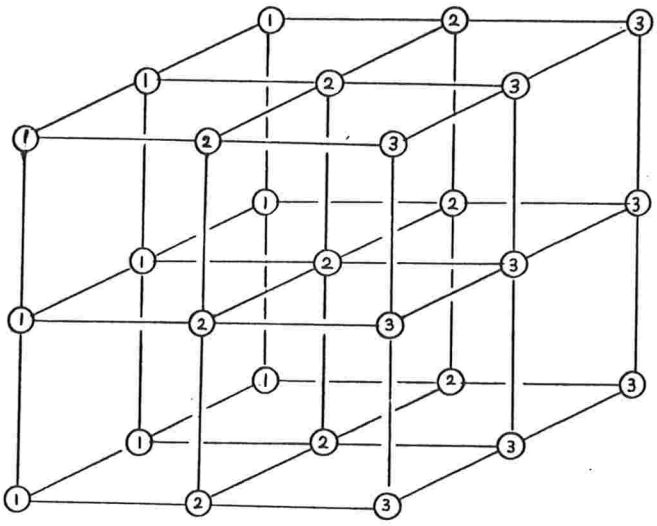


Fig. 3.8

Por ii), na Fig. 3.9 os pontos marcados com o mesmo número são independentes.

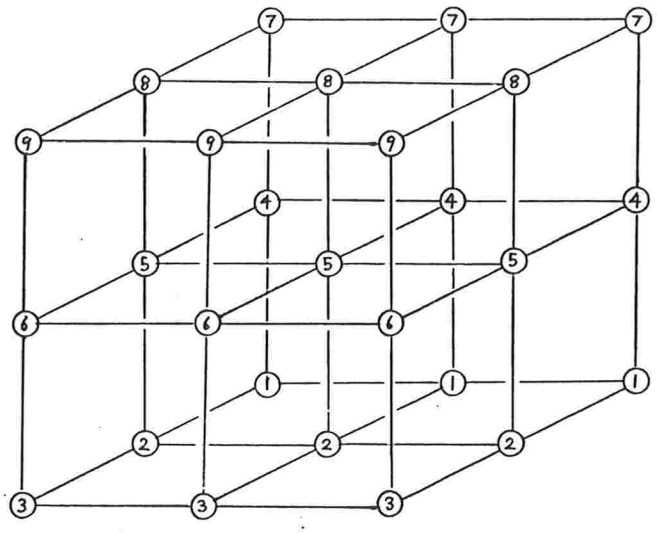


Fig. 3.9

Intuitivamente, o próximo lema diz que, se algumas linhas de D satisfazem a definição de $L^T$, então 2 pontos de $\mathbb{CI}$, para os quais a distância entre eles restrita a estas coordenadas é menor que certo valor, são independentes.

Lema 3.

Sejam A, CI, D, $L^>$, $L^<$ e $L^T$ conforme definição anterior.

Seja $g = (\ell_{k1},\ldots,\ell_{kt}) \in L^T$ e $h = (H_1,\ldots,H_t)$

onde

$$H_q = \begin{cases} \min\{|d_{kq^j}| : d_{kq^j} \neq 0\} & \text{se } \ell_{kq} \in L^> \\ -\min\{|d_{k_q j}| : d_{kq^j} \neq 0\} & \text{se } \ell_{kq} \in L^< \end{cases}$$

para $q = 1,2,\ldots,t$

o sinal de $H_q$ é o mesmo dos elementos não nulos de $\ell_{k_q}$.

Se $P = (p_1,p_2,\ldots,p_n) \in \mathbb{CI}$ e $P' = (p'_1,p'_2,\ldots p'_n) \in \mathbb{CI}$ são tais que

$$\sum_{q=1}^{t} \left|\frac{p_{kq} - p'_{kq}}{H_q}\right| < 1 ,$$

então as computações em P e P' são independentes.

Dem.: Caso 1: Consideremos $(p_{kq}-p'_{kq})/H_q = 0 \quad \forall q=1,\ldots,t$

então

$$p_{kq} - p'_{kq} = 0 \quad \forall\ q = 1,\ldots,t$$

Por outro lado, pela def. 3.11 as colunas formadas por $L^T$ não podem ser nulas e consequentemente, pelo Lema 1, as colunas formadas por $\tilde{L}^T$ também não podem ser nulas.

Logo, o vetor $d = P-P'$ (ou $-d$) $\notin \tilde{D}$.

Caso 2: $\exists\ q_o$ tal que $|(p_{k_{q_o}} - p'_{k_{q_o}})/H_{q_o}| \neq 0$.

Obviamente, $|p_{kq_o} - p'_{kq_o}|/|H_{q_o}| < 1$ (decorrente da hipótese do Lema 3), ou

$$0 < |p_{kq} - p'_{kq_o}| < |H_{q_o}|$$

Em outras palavras, $\exists\ q_o$ tal que $p_{kq_o} - p'_{kq_o}$ é não nulo e menor em valor absoluto do que o menor elemento não nulo da linha $\ell q_o$ em D ($\ell q_o \in L^{>}$ ou $L^{<}$), consequentemente, menor do que o menor elemento não nulo de $\tilde{\ell} q_o$ em $\tilde{D}$.

Portanto, $d = p-p'$ (ou $-d$) $\notin \tilde{D}$

Em ambos os casos, p e p' são independentes

□

O Lema 4 é mais abrangente que o Lema 3.

Lema 4: Sejam A, $\mathbb{C}$I, D, $L^{>}$, $L^{<}$, $L^T$, g,h, P e P' como no Lema 3. Se p e p' são tais que

$$\left| \sum_{q=1}^{t} \frac{p_{kq} - p'_{kq}}{H_q} \right| < 1,$$

então as computações em p e p' são independentes.

Demonstração. Se $\frac{p_{k_q} - p'_{k_q}}{H_q} \geq 0 \quad \forall\, q = 1,\ldots,t$ (ou $\frac{p_{k_q} - p'_{k_q}}{H_q} \leq 0$ $\forall\, q = 1,\ldots,t$), então recai-se no Lema 3.

Supomos que existem $q$ e $\bar{q} \in \{1,\ldots,t\}$ tais que

$(p_{k_q} - p'_{k_q})/H_q > 0$. e $(p_{k_{\bar{q}}} - p'_{k_{\bar{q}}})/H_{\bar{q}} < 0$

Seja $j = p-p'$, então $d$ tem $\bar{q}$-ésimo componente com sinal oposto de elementos não nulos de $\bar{q}$-ésima linha de D, pois $(p_{k_{\bar{q}}} - p'_{k_{\bar{q}}})/H_{\bar{q}} < 0$ significa numerador e denominador com sinais opostos, e consequentemente, sinal oposto de elementos não nulos da $\bar{q}$-ésima linha de $\tilde{D}$.

Logo, $d \notin \tilde{D}$.

Analogamente, consideramos $-d$. Temos $(p'_{k_q} - p_{k_q})/H_q < 0$ e $(p'_{k_{\bar{q}}} - p_{k_{\bar{q}}})/H_{\bar{q}} > 0$ e então $-d$ tem a $q$-ésimo componente com sinal oposto ao dos elementos não nulos da $q$-ésima linha de D e $-d \notin \tilde{D}$.

Assim, P e P' são independentes. □

Do Lema 4 temos o seguinte:

Corolário 1:
Sejam A, $\mathbb{I}$, D, P, P' como anteriormente e

$$\ell_k \in L^+ \quad (\text{ou } \ell_k \in L^-)$$

$$H_k = \min_j\{|d_{kj}|\} \quad (\text{ou} \quad -\min\{|d_{kj}|\} \quad \text{se} \quad \ell_k \in L^-)$$

Se P e P' são tais que $|p_k - p'_k| < |H_k|$, então P e P' são independentes.

Agora veremos vários exemplos para ilustrar o Lema 3, o Lema 4 e o Corolário 1.

Não especificamos $\mathbb{CI}$ e, nas figuras apenas uma parte de $\mathbb{CI}$, que está na vizinhança de um ponto em particular, é mostrada, pois $\mathbb{CI}$ toda não é de interesse nestes exemplos. Nosso interesse é a dependência de um ponto a outros pontos, sendo irrelevante se o I vai de 0 a 5 ou vai de 0 a 500.

Exemplos:

Voltamos ao Ex. 3.3, temos $D = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

$$(\ell_1, \ell_2, \ell_3) \in L^T$$

$$H_1 = H_2 = H_3 = 1$$

Sejam $P = (p_1, p_2, p_3)$ e $P' = (p_1', p_2' \text{ e } p_3')$.

Para satisfazer o Lema 3 deve acontecer

$|p_1 - p_1'| + |p_2 - p_2'| + |p_3 - p_3'| < 1$. Como todos os elementos são inteiros, temos $p_i = p_i'$ ; $i = 1,2,3$. Logo, pelo Lema 3, não detectamos pontos de computação independente.

Entretanto, podemos detectar pontos de computação independente pelo Lema 4, como se segue.

Para satisfazer o Lema 4, deve acontecer

$|(p_1 - p_1') + (p_2 - p_2') + (p_3 - p_3')| < 1$. Novamente, como todos os elementos são inteiros, temos $(p_1 - p_1') + (p_2 - p_2') + (p_3 - p_3') = 0$.

Logo $$p_1 + p_2 + p_3 = p_1' + p_2' + p_3' .$$

Na Fig. 3.10, os pontos independentes entre si são marcados com a mesma numeração.

Por outro lado, podemos notar que $(\ell_1, \ell_2)$, $(\ell_2, \ell_3)$, $(\ell_1, \ell_3) \notin L^T$ e como contra exemplo, tentemos aplicar o Lema 3 para $(\ell_1, \ell_2)$:

$$|p_1 - p_1'| + |p_2 - p_2'| < 1$$

$$p_1 = p_1' \quad e \quad p_2 = p_2'$$

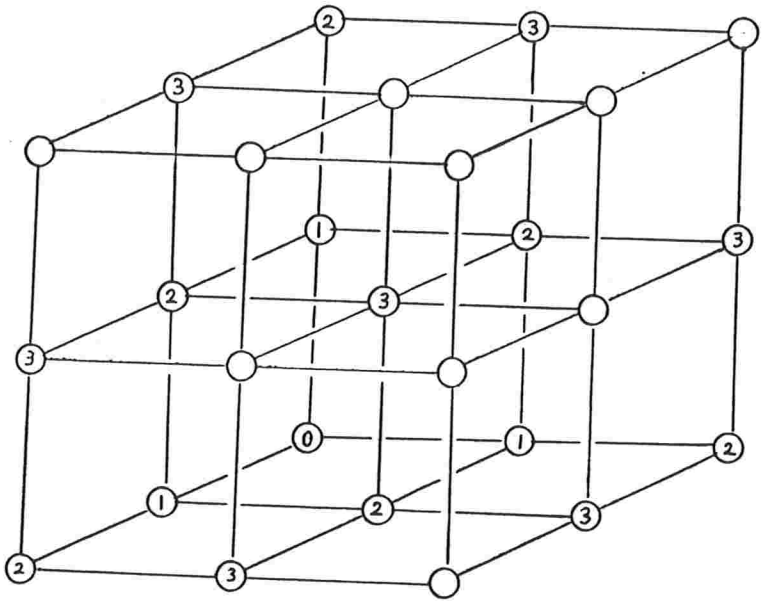



Fig. 3.10

então, P = (0,0,1) e P' = (0,0,2) seriam pontos independentes, o que não é verdade.

Como $\ell_i \notin L^+$ (ou $L^-$) para i = 1,2,3, o Corolário 1 não se aplica para o ex. 3.3.

2. Exemplo 3.8: Considere um algoritmo A com a seguinte D:

$$D = \begin{pmatrix} 0 & 0 & 3 \\ 3 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix}$$

$$(\ell_1, \ell_2, \ell_3) \in L^T$$

$$H_1 = H_2 = H_3 = 3$$

$$P = (p_1, p_2, p_3)$$

$$p' = (p'_1, p'_2, p'_3)$$

Pelo Lema 3

$$\left| \frac{p_1 - p'_1}{3} \right| + \left| \frac{p_2 - p'_2}{3} \right| + \left| \frac{p_3 - p'_3}{3} \right| < 1$$

portanto,

$$\sum_{q=1}^{3} |p_q - p'_q| < 3$$

ou

$$\sum_{q=1}^{3} |p_q - p'_q| \leq 2$$

Na Fig. 3.11, P = (2,2,2) assinalado com x é independente com os pontos marcados com ●

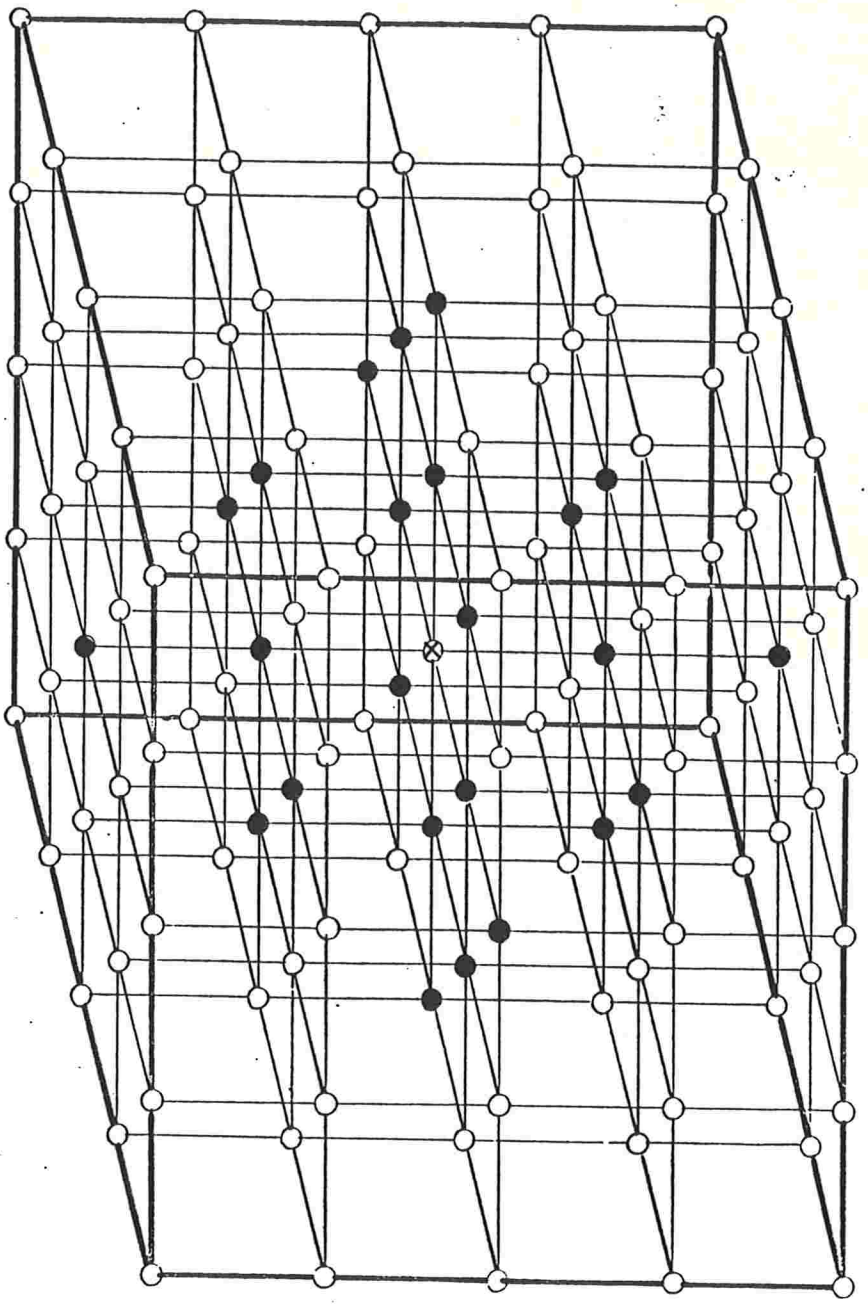


Fig. 3.11

3. Exemplo 3.9.

$$D = \begin{pmatrix} 2 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 2 \end{pmatrix}$$

$$(\ell_1, \ell_2, \ell_3) \in L^T$$

$$H_1 = H_2 = H_3 = 2$$

$$P = (p_1, p_2, p_3)$$

$$P' = (p'_1, p'_2, p'_3)$$

Pelo Lema 3

$$\sum_{q=1}^{3} \left| \frac{p_q - p'_q}{2} \right| < 1$$

Portanto,

$$\sum_{q=1}^{3} |p_q - p'_1| < 2$$

ou

$$\sum_{q=1}^{3} |p_q - p'_q| \leq 1$$

p e P' têm apenas uma coordenada que difere de uma unidade.

Na Figura 3.12, P = (1,1,1) é independente com pontos marcados com ●

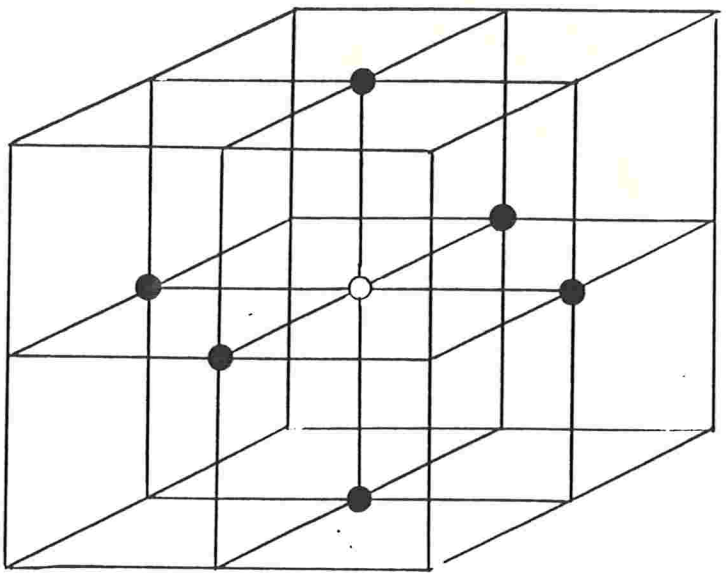


Fig. 3.12.

Por outro lado, podemos observar que $(\ell_1,\ell_2) \in L^T$:

$$H_1 = H_2 = 2$$

Pelo Lema 3

$$\left|\frac{p_1-p_1'}{2}\right| + \left|\frac{p_2-p_2'}{2}\right| < 1$$

ou

$$\sum_{q=1}^{2} |p_q-p_q'| \leq 1$$

Portanto, devemos ter

$$\begin{cases} |p_1-p_1'| = 0 \\ |p_2-p_2'| \leq 1 \end{cases} \quad \text{ou} \quad \begin{cases} |p_1-p_1'| \leq 1 \\ |p_2-p_2'| = 0 \end{cases}$$

Na Figura 3.13, P = (1,1,1) é independente aos pontos marcados.

Analogamente, $(\ell_2,\ell_3) \in L^T$ e $(\ell_1,\ell_3) \in L^T$ e os resultados são semelhantes.

Agora, aplicando o Lema 4, temos:

$(\ell_1,\ell_2,\ell_3) \in L^T$

$H_1 = H_2 = H_3 = 2$

$$|(p_1-p_1')+(p_2-p_2')+(p_3-p_3')| \leq 1$$

$$(p_1-p_1')+(p_2-p_2')+(p_3-p_3') = 0$$

ou

$$(p_1-p_1')+(p_2-p_2')+(p_3-p_3') = 1$$

ou

$$(p_1-p_1')+(p_2-p_2')+(p_3-p_3') = -1$$

$$p_1 + p_2 + p_3 = p_1' + p_2' + p_3'+1 \quad \ldots\ldots\ldots\ldots \textcircled{1}$$

ou

$$p_1 + p_2 + p_3 = p_1' + p_2' + p_3' \quad \ldots\ldots\ldots\ldots\ldots \textcircled{2}$$

ou

$$p_1 + p_2 + p_3 = p_1' + p_2' + p_3'-1 \quad \ldots\ldots\ldots\ldots \textcircled{0}$$

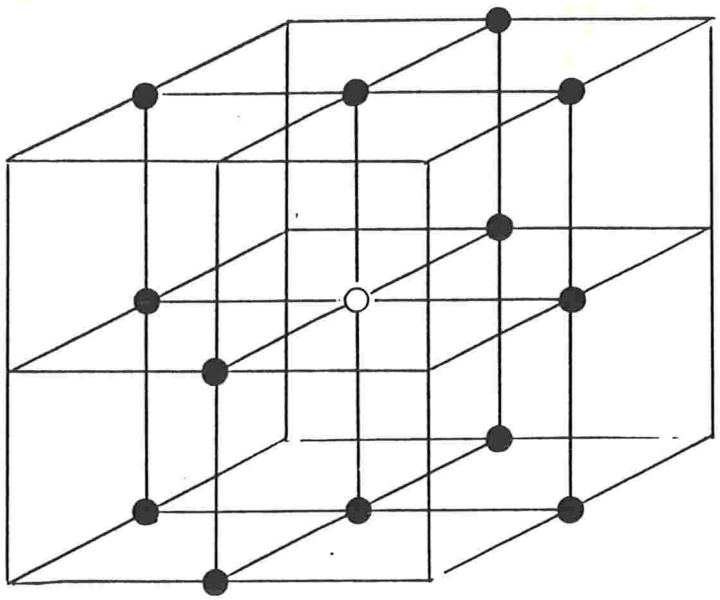


Fig. 3.13.

A Figura 3.14 ilustra a situação: Considerando P = (2,2,2), os pontos marcados com 1 são independentes com P devido à relação 1, analogamente para pontos marcados com 2 e 0.

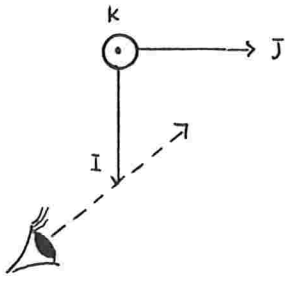



Fig. 3.15a.

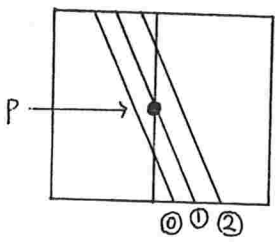



Fig. 3.15b.

Se olharmos a Fig. 3.14, na direção mostrada na Fig. 3.15a., podemos notar que os pontos marcados com o mesmo número estão situados no hiperplano com o respectivo número na Fig. 3.15b.

Notamos também que os três hiperplanos são paralelos.

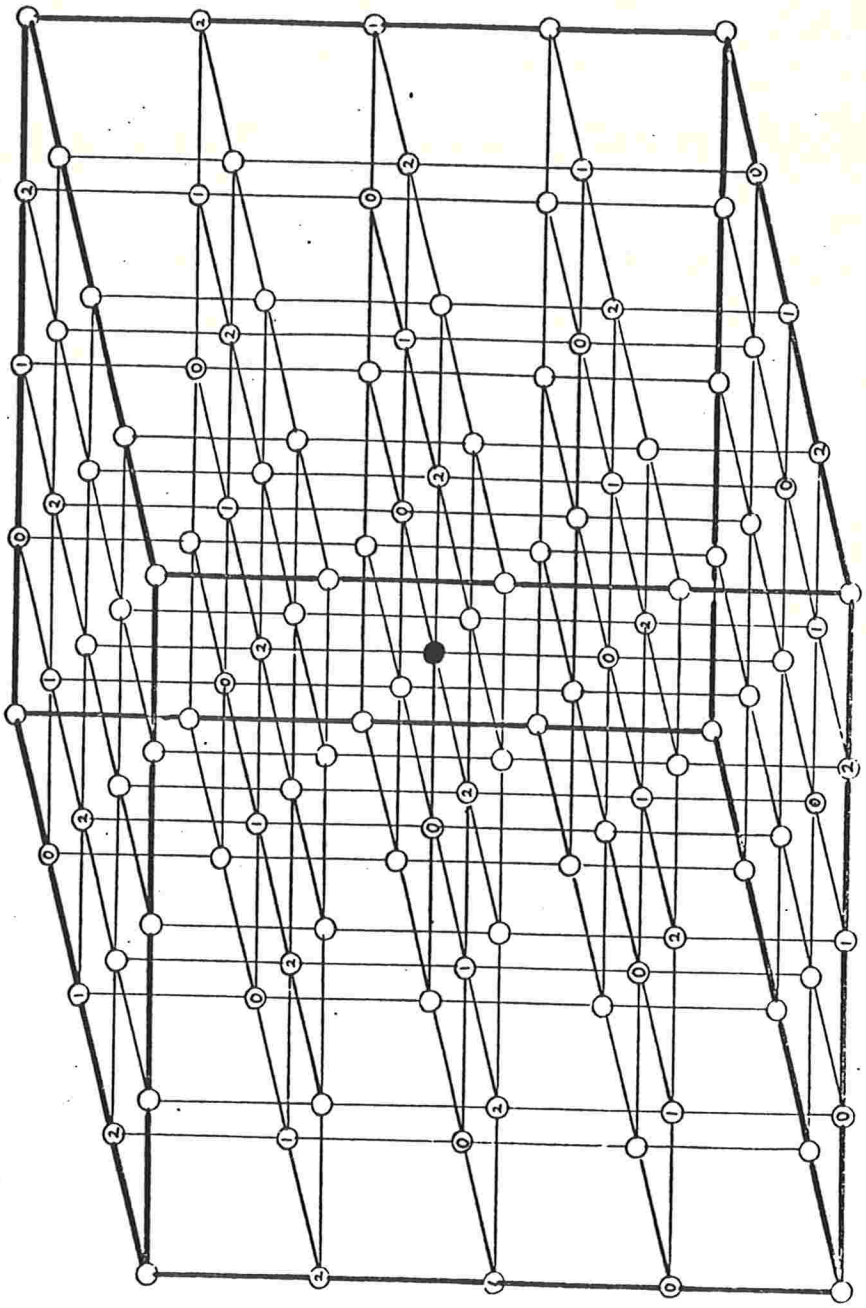


Fig. 3.14.

4. Exemplo 3.10

$$D = \begin{pmatrix} 2 & 3 & 1 \\ 3 & 2 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

Pelo Lema 2, $P = (p_1,p_2,p_3)$ e $P' = (p_1',p_2',p_3')$ tais que $p_3 \neq p_3'$ são independentes e na Fig. 3.16 os pontos marcados com números diferentes são independentes.

Pelo Corolário 1:

$\ell_1 \in L^+$,

$H_1 = 1$ ,

P e P' são pontos independentes se $|p_1-p_1'| < 1$ (i.e., $p_1 = p_1'$).

Na Fig. 3.17, P = (2,2,2) é independente com pontos marcados com ●.

Por outro lado, temos que $\ell_2 \in L^+$ e $H_2 = 2$ ,

P e P' são independentes se $|p_2-p_2'| < 2$ (i.e. $p_2-p_2' = 0$ ou $\pm 1$).

Na Fig. 3.18, p = (2,2,2) é independente com pontos marcados com ●.

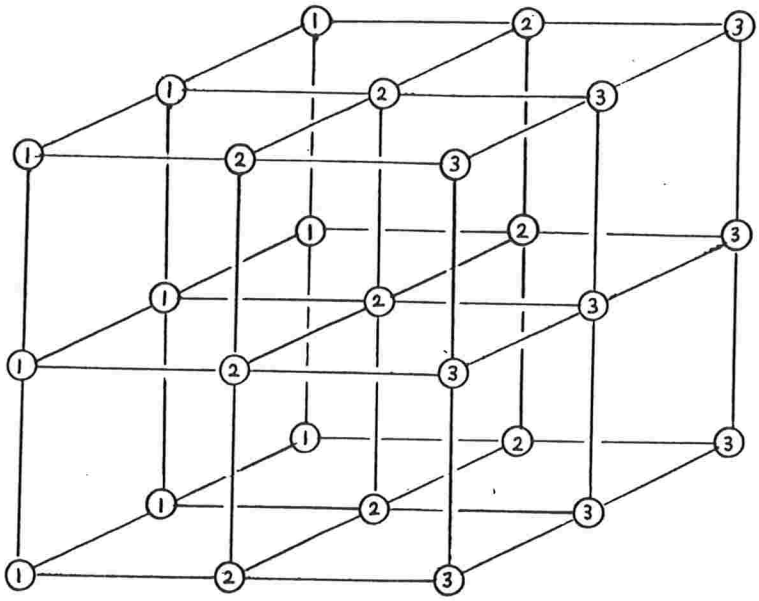



Fig. 3.16.

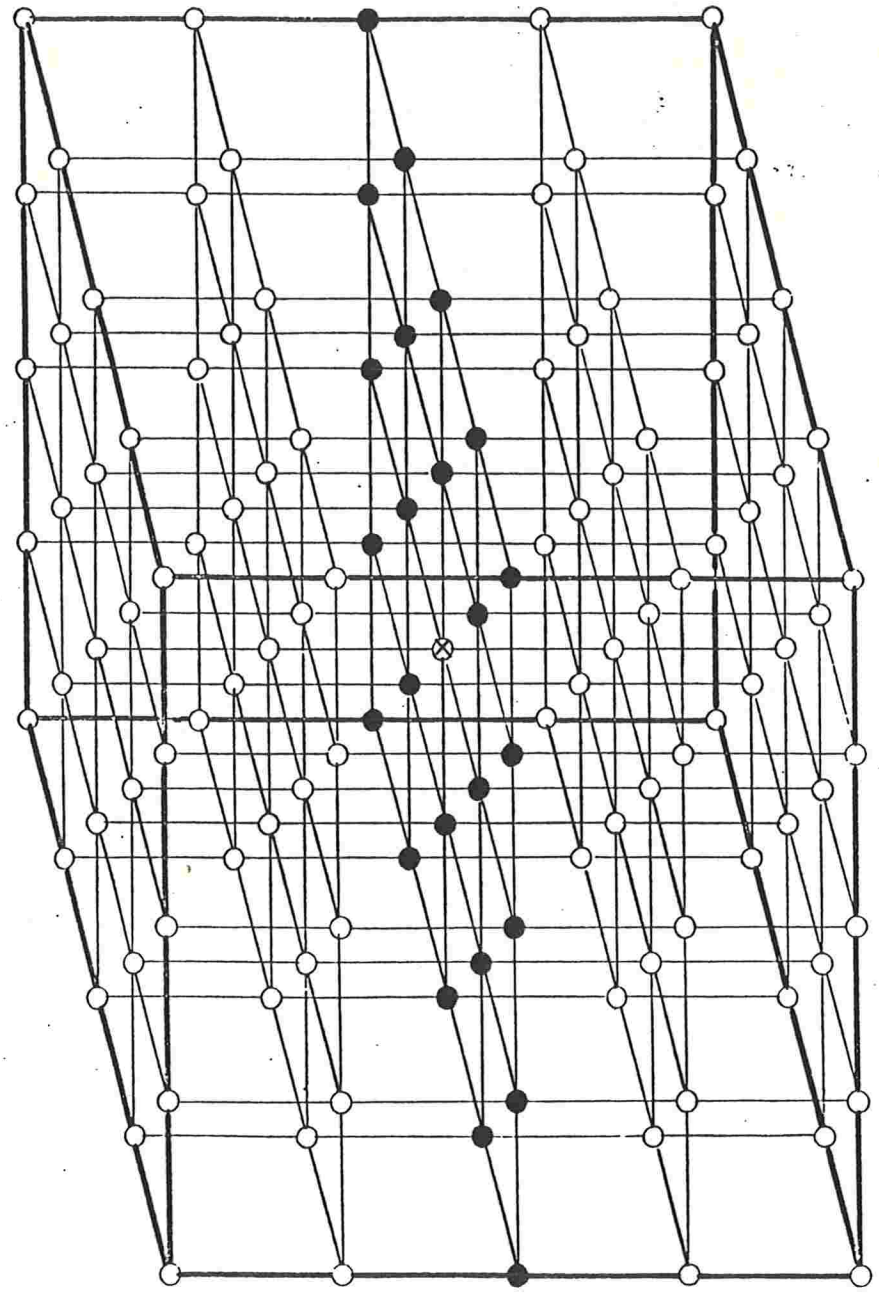


Fig. 3.17.

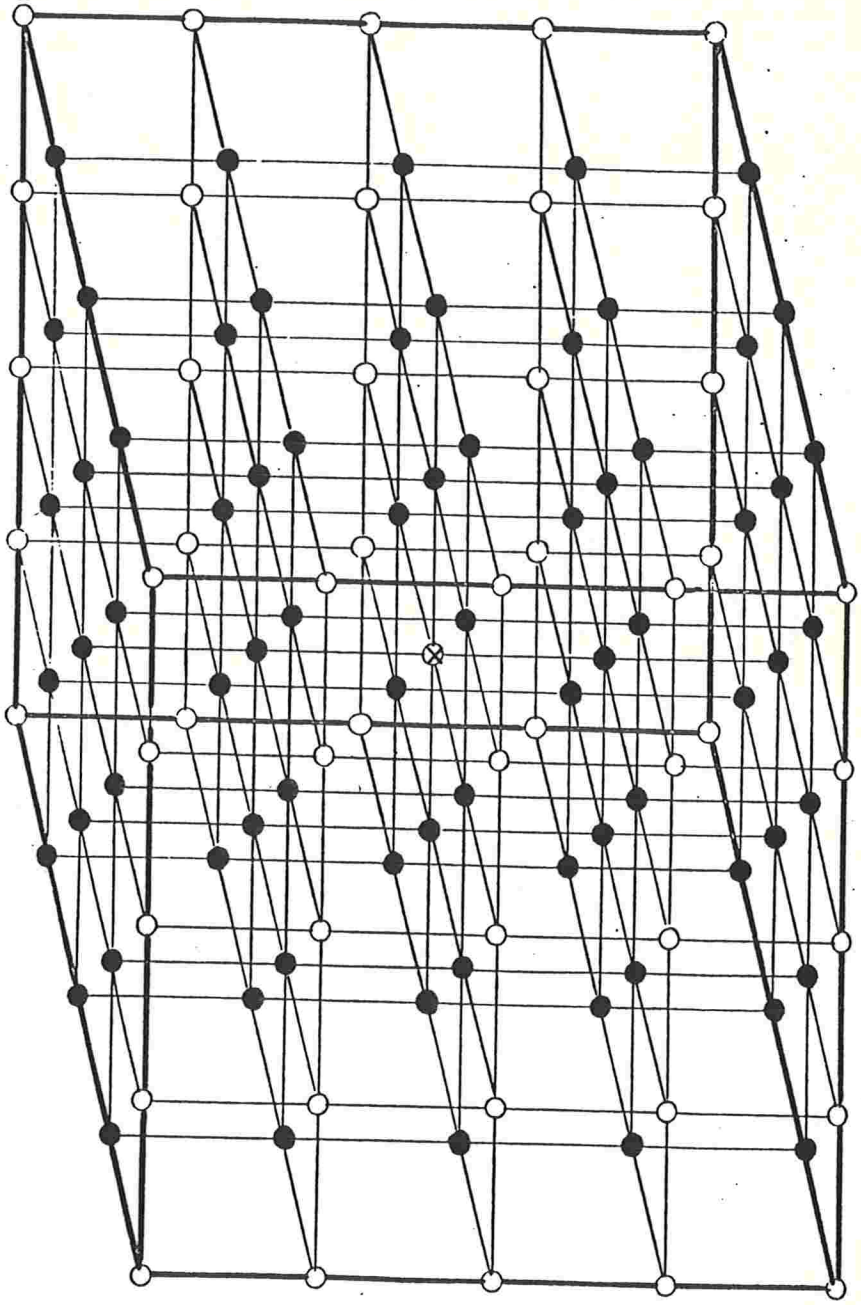


Fig. 3.18.

## §.3 Ordenação de Execução

Dado um algoritmo expresso em malhas encaixadas, uma ordem natural para a execução das computações nos pontos de $\mathbb{CI}$ é a ordem lexicográfica dos seus índices.

Tal ordem de execução corresponde à execução sequencial das computações.

Nós já temos vários resultados para detectar conjuntos de computações independentes. Vamos agora obter ordens de execução que adequadamente escalonam a execução simultânea de computações independentes.

Vamos introduzir dois tipos de ordenações: ordenação relaxada e ordenação linear. Elas estão ligadas intimamente com o Lema 2 e o Lema 4, respectivamente.

Daremos dois teoremas nesta seção e omitiremos suas demonstrações por serem relativamente imediatas.

Dada uma ordem de execução $>$ sejam P e P' dois pontos de $\mathbb{CI}$. $P < P'$ significa que a computação em P deve preceder a computação em P'.

Definição 3.12. Seja $>$ a ordem lexicográfica em $\mathbb{CI}$. *O relaxamento de $>$ sobre a coordenada $j_s$* é uma ordem parcial $>_s$ tal que:

$$\text{para } P = (p_1,\ldots p_n) \quad \text{e} \quad P' = (p'_1,\ldots,p'_n) \text{ de CI,}$$

$$P >_s P' \iff P > P'$$

e

$$p_i \neq p'_i \quad \text{para algum } i \neq s.$$

Exemplo. Voltamos ao ex. 3.7: Vamos relaxar a $>$ sobre a coordenada J, i.e. $s = 3$.

Na Fig. 3.9, os pontos marcados com o mesmo número são aqueles que perderam a relação de ordem na nova ordenação $>_3$.

Teorema 1. Sejam A, $\mathbb{CI}$, D e $L^o$ como anteriormente. O relaxamento de $>$ sobre qualquer coordenada $i_s$ onde $\ell_s \in L^o$ é uma ordem de execução de A.

Quaisquer pontos não relacionados por essa ordenação são independentes e podem ser executados em paralelo.

Demonstração: Fortes & Moldovan [85].

Exemplo: Voltemos para o ex. 3.7. , a Fig. 3.9 mostra exatamente a nova ordenação $>_3$.

Podemos executar este algoritmo em nove passos seguindo a numeração da Fig. 3.9 e executando computações de três pontos marcados com o mesmo número simultaneamente em cada passo.

Vamos introduzir mais um tipo de ordenação:

Definição 3.13: Seja $g = (j_{k1}, j_{k2}, \ldots, j_{kt})$ tupla de coordenadas de $\mathbb{CI}$ e seja $h = (H_1, \ldots, H_t)$ tupla de constantes inteiras.

Então a *ordenação linear sobre g com deslocamento* h é qualquer ordenação parcial $>^{g,h}$ tal que:

para $P, P' \in \mathbb{CI}$

se $\sum_{\ell=1}^{t} (p_{k_\ell} - p'_{k_\ell})/H_\ell \geq 1$ então $P >^{g,h} P'$

Exemplo 3.2: Considere $\mathbb{CI} = \{(K,I,J) \in \mathbb{Z}^3 / \ 0 \leq K,I,J \leq 2\}$ e

$$g = (I,J) \ ; \quad h = (1,2)$$

São ilustrados a seguir (Fig. 3.19) alguns pares de pontos relacionados pela ordenação linear $<_{g,h}$. Temos

$$(0,1,1) <_{g,h} (1,2,1)$$
$$(0,1,1) <_{g,h} (1,2,2)$$
$$(1,1,0) <_{g,h} (1,2,1)$$
$$(1,1,0) <_{g,h} (1,2,2)$$
$$(3,1,1) <_{g,h} (1,2,1)$$
$$(3,1,1) <_{g,h} (1,2,2)$$

Ver Fig. 3.19

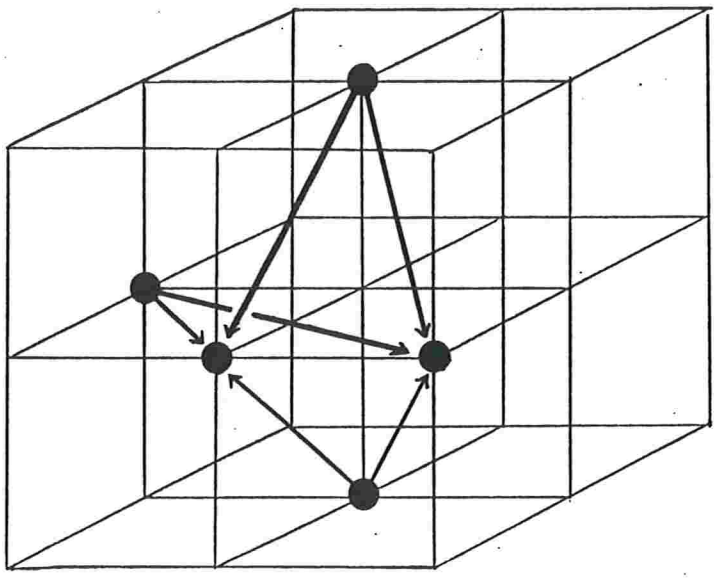


Fig. 3.19

Teorema 2: Sejam $A,\mathbb{I},D,L^T$, g h como no lema 4, então qualquer ordenação linear sobre g com deslocamento h, $>^{g,h}$, é uma ordem de execução de A. Quaisquer pontos não relacionados por essa ordenação são independentes e podem ser executados em paralelo.

Demonstração: Fortes & Moldovan [85].

Exemplos:

1) Retomemos o Ex. 3.3.: Na Figura 3.10, temos a ordenação $>^{g,h}$ onde g = (K,I,J), h = (1,1,1).

Podemos executar este algoritmo seguindo a numeração da figura, os pontos marcados com a mesma numeração serão executados simultaneamente. Observe que os pontos de computação paralela estão contidos em planos paralelos.

2) Voltamos ao Ex. 3.9: g = (K,I,J) ; h = (2,2,2).

Na Figura 3.14 a numeração mostra a ordenação $>^{g,h}$ sendo que os dois hiperplanos paralelos consecutivos são executados simultaneamente.

Corolário 2: Sejam $A,\mathbb{I},D,\ell_K,L^+$ e $H_K$ como no Corolário 1, então a ordenação linear sobre a coordenada $j_k$ com deslocamento $H_k$ é uma ordem de execução de A.

Todos os pontos não relacionados com essa nova ordenação são independentes e podem ser executados em paralelo.

Em outras palavras, a ordenação acima é uma ordenação relaxada

sobre $\{j_1,\ldots,j_{k-1},j_{k+1},\ldots,j_n\}$

Exemplos:

1) Voltamos ao ex. 3.10, com $\ell_1 \in L^+$ e $H_1 = 1$ .

O plano marcado na Fig. 3.17 mostra um passo de execução da nova ordenação. Os pontos situados em cada plano horizontal são executados em paralelo, começando com o plano K = 0 e avançando na direção K.

2) Ainda no ex. 3.10 com $\ell_2 \in H^+$ e $H_2 = 2$.

Os pontos situados nos dois planos paralelos consecutivos ao plano (I = 0) na Fig. 3.18 são executados simultaneamente, começando com os planos I=0 e I=1 e avançando na direção I.

oOo

# Capítulo 4

# Transformação de Algoritmo e Obtenção de Sistema Sistólico

No §1 vamos transformar um algoritmo para satisfazer o Corolário 2 do Capítulo 3, tomando um dos índices de $\mathbb{CI}$ como índice de tempo ou passo de execução.

NO §2 veremos como obter um sistema sistólico a partir do algorítmo transformado.

Os resultados deste Capítulo são válidos para um algoritmo constituído de um número qualquer de malhas encaixadas, conforme definido no Capítulo 3.

Algorítmos com três ou duas malhas, entretanto, são de particular interesse pois resultarão em sistemas sistólicos de duas e uma dimensões, respectivamente.

Neste capítulo, trataremos de casos com três malhas encaixadas.

## §.1 Transformador

Já sabemos como detectar pontos de computação paralela em $\mathbb{CI}$ de um dado algoritmo A no capítulo 3, nossa próxima tarefa agora é como utilizar esta informação para projetar um sistema sitólico para A.

Considere novamente o ex. 3.3., através do Lema 4 já conhecemos os pontos de computação paralela e basta seguir as numera

ções da Fig. 3.10 para a execução paralela conforme o Teorema 2. A nossa dificuldade é como obter um sistema sistólico que realiza tais computações.

A numeração como na Fig. 3.10, é pouco conveniente para descrever a execução paralela de um sistema sistólico. Vamos modificar um pouco o algoritmo A (consequentemente seu $\mathbb{C}I$ e D como veremos logo) para que novo algoritmo A' fique nas condições do Corolário 2.

Definição 4.1. Dado um algoritmo A, seu $\mathbb{C}I$ e $D\bar{c}$ vamos considerar T um operador linear bijetor de $\mathbb{R}^n$ e um algorítmo A' tais que:

1) T na sua representação matricial tem todos os elementos inteiros.

2) O algoritmo A' é o seguinte:

```
for I_1 := N_1  to  M_1  do
 for I_2 := N_2  to  M_2  do
   .
   .
  for I_t := N_t  to  M_t  do
   begin
    a_1[T(I_1,...,I_t)]:= f_1{ a_1[T(I_1-d^1_11,...,I_t-d^1_t1)],...
                               a_s[T(I -d^1_1s,...,I_t-d^1_ts)]} ;
    a_2[T(I_1,...,I_t)]:= f_2{a_1[T(I_1-d^2_11,..,I_t-d^2_t1)],...
                               a_s[T(I -d^2_1s,...,I_t-d^2_ts)]} ;
                 .
                 .
                 .
    a_s[T(I_1,...,I_t)]:= f_s{a_1[T(I,-d^s_11,...,I_t-d^s_t1)],...
                               a_s[T(I_1-d^s_1s,...,I_t-d^s_ts)]} ;
   end;
```

Chamamos A' de *algoritmo transformado* de A e T de *transformador* de A.

Também adotamos a notação A' = T(A).

Obs.: Permitimo-nos certo abuso de linguagem entre posição $(I_1,I_2,\ldots,I_t)$ e vetor $(I_1,I_2,\ldots,I_t)^T$. A maneira correta de escrever seria, por exemplo, $a_1[(T\{(I_1,\ldots,I_t)^T\})^T]$ em lugar de $a_1[T(I_1,\ldots,I_t)]$, mas tal formalismo sobrecarregaria a notação. Portanto, continuamos com este abuso a menos que haja perigo de confusão.

Daremos um exemplo para ilustrar a modificação causada por T, assim como seus efeitos.

Exemplo 4.1: Considere o ex. 3.3. e um transformador T dado por:

$$T = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ao ponto (K,I,J) = (1,1,1) vai corresponder a seguinte computação:

```
a[T(1,1,1)]:= a[T(1,0,1)]
b[T(1,1,1)]:= b[T(1,1,0)]
c[T(1,1,1)]:= c[T(0,1,1)]
              + a[T(1,1,1)]*b[T(1,1,1)]
```

i.e.

```
a(3,1,1):= a(2,0,1);
b(3,1,1):= b(2,1,0);
c(3,1,1):= c(2,1,1)+a(3,1,1)*b(3,1,1);
```

No ex. 3.3. o ponto (K,I,J) = (1,1,1) é associado ao ponto de coordenada (1,1,1) do seu ℂI e computamos a(1,1,1), b(1,1,1) e c(1,1,1), i.e., até agora tivemos coincidência entre índice de computação do algoritmo A, coordenada do ponto de computação de ℂI e índices de variáveis a serem computadas. No ex. 4.1. perdemos esta coincidência, pois no ponto (K,I,J) = (1,1,1) computamos a(3,1,1), b(3,1,1) e c(3,1,1).

A rigor,não sabemos o que seria o ℂI de A' pois ℂI foi definido na Def. 3.1 para o algoritmo A e o corpo do algoritmo A' é diferente de A. Entretanto, uma maneira natural de definir ℂI é a seguinte:

Definição 4.2: Dados A, ℂI e seu transformador T, o ℂI de T(A) é T(ℂI).

ex. para o ex. 4.1. ℂI de T(A) é mostrada na Fig. 4.1 e a Fig. 4.2 mostra como a ordem lexicográfica do ex. 3.3. (Fig. 3.2.2) foi transportada para T(A').

A numeração da Fig. 4.2 é uma ordem de execução do T(A) mas note que os novos pontos de computação já não conserva a ordem lexicográfica em termos de (K,I,J).

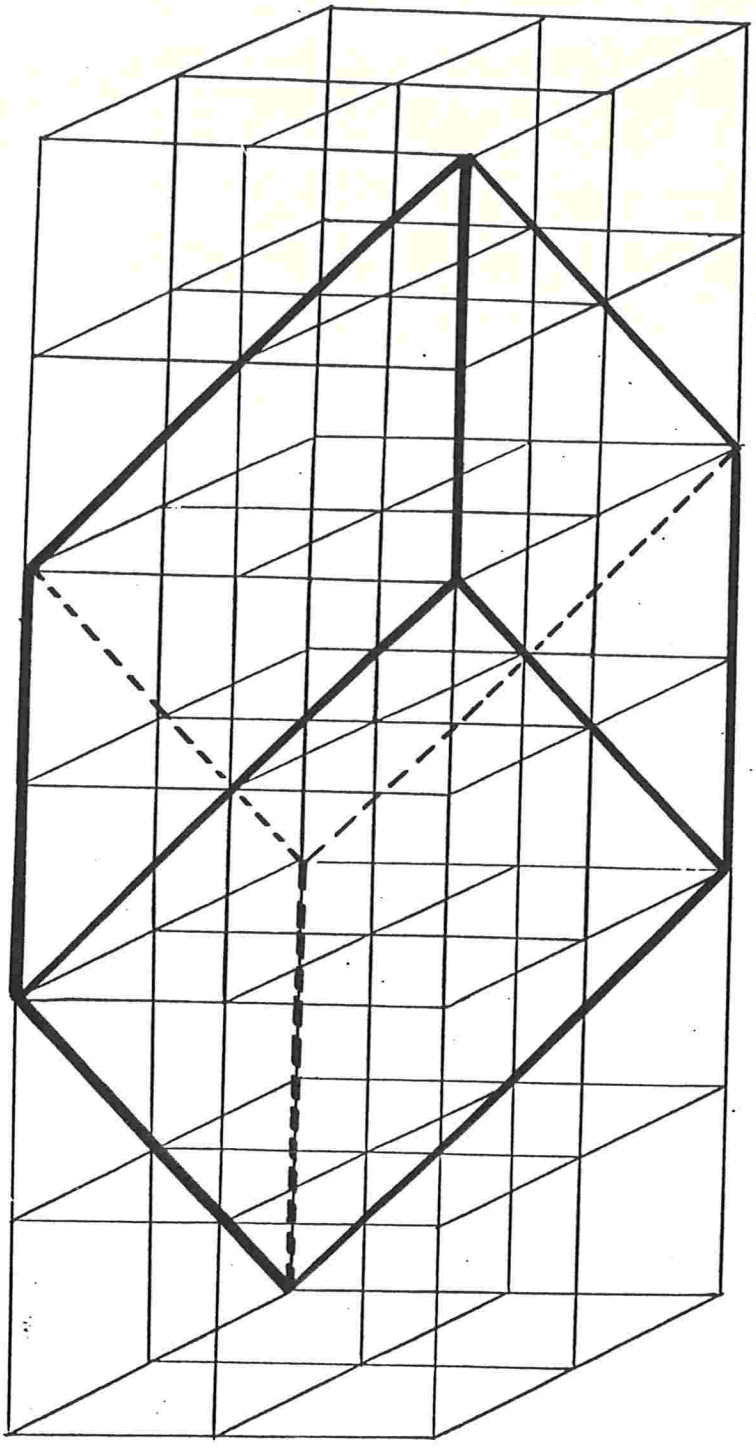


Fig. 4.1.

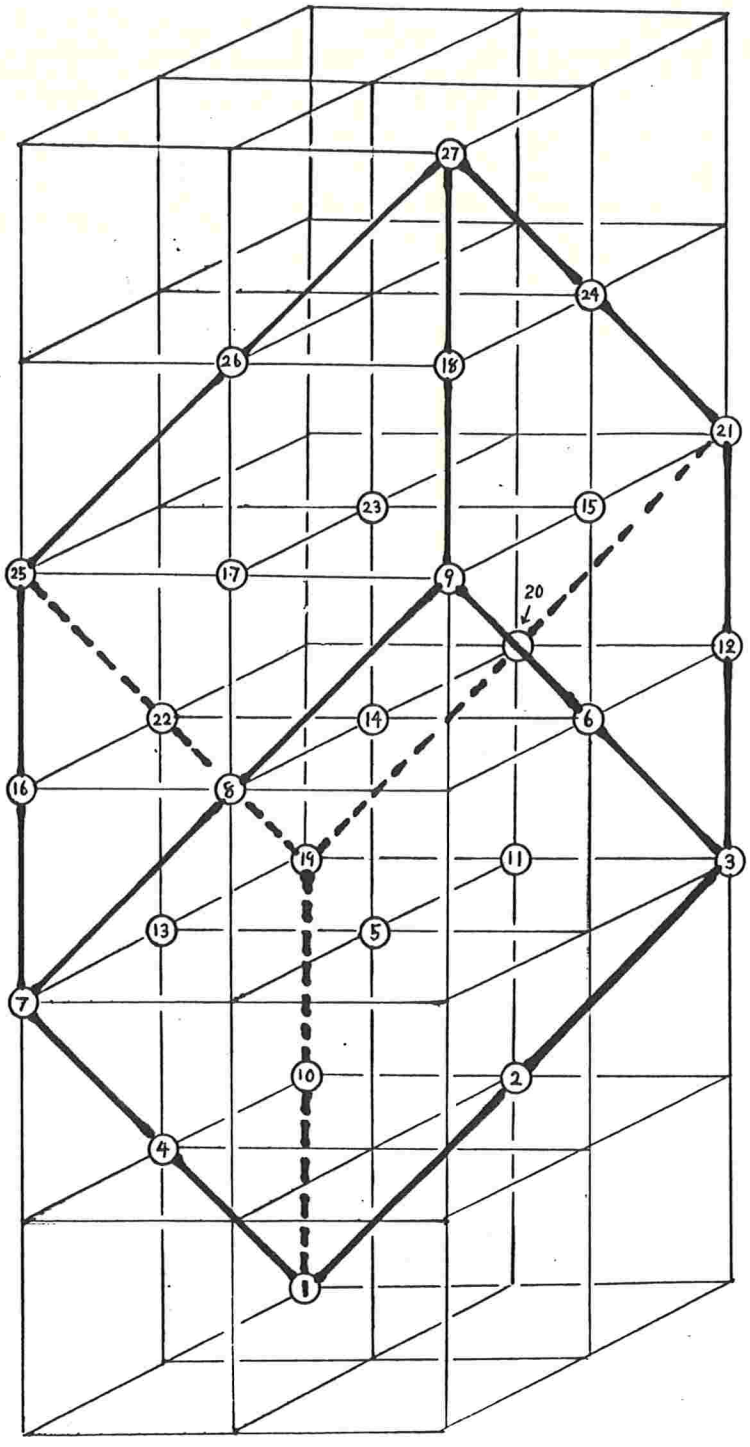



Fig. 4.2.

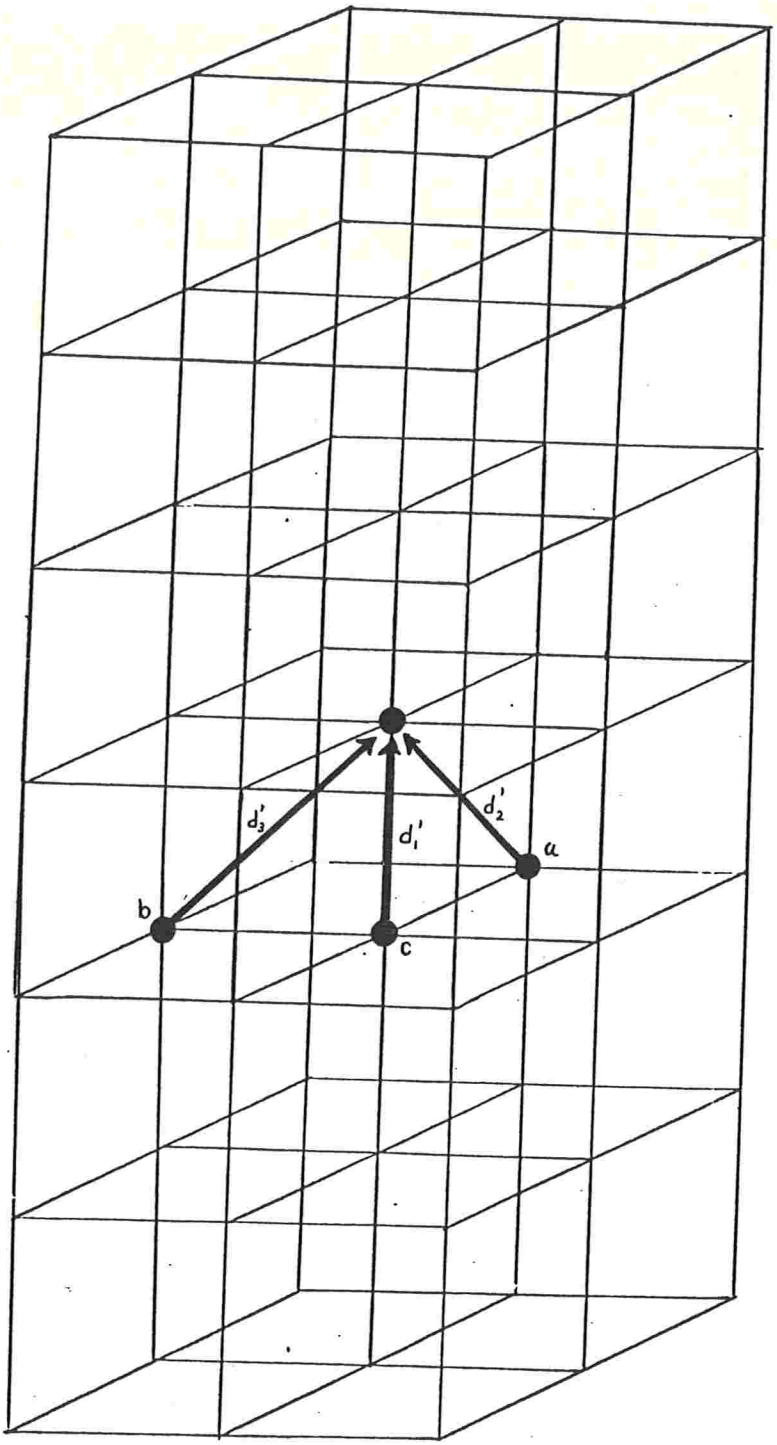



Fig. 4.3.

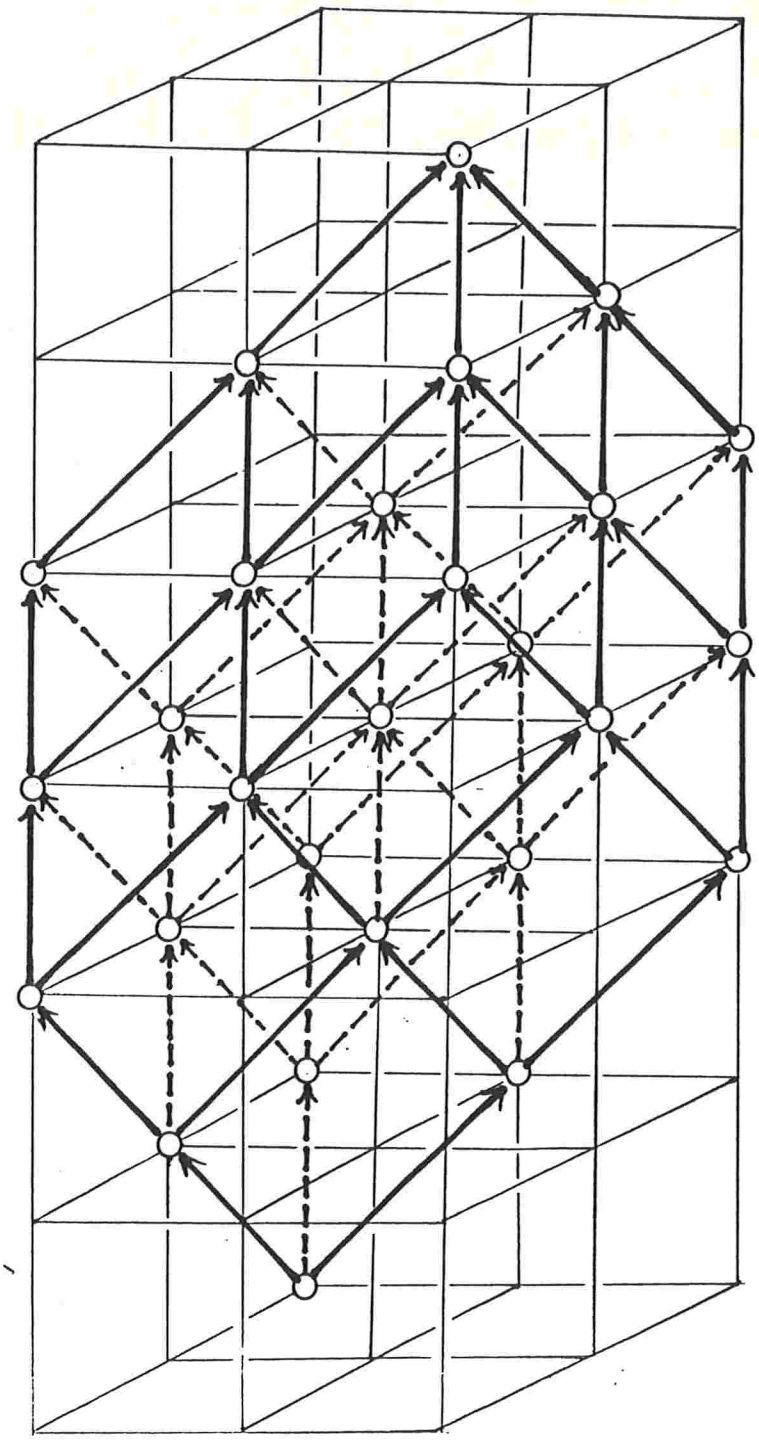


Fig. 4.4.

No ex. 3.3., (K,I,J) = (1,1,1) corresponde ao ponto marcado de 14 da Fig. 3.2.2. e ele foi levado para (3,1,1) via T, conforme mostra a Fig. 4.2. onde são computadas a(3,1,1), b(3,1,1) e c(3,1,1) recuperando deste modo a coincidência que perdemos.

Vejamos como ficam os vetores de dependência.

A Fig. 4.3. mostra o fluxo de dados para a computação em (3,1,1) com os três vetores de dependência:

$$d'_1 = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad d'_2 = \begin{pmatrix}1\\1\\0\end{pmatrix} \quad \text{e} \quad d'_3 = \begin{pmatrix}1\\0\\1\end{pmatrix} ,$$

i.e., $$d'_i = T(d_i) , \quad 1 \leq i \leq 3$$

Temos a seguinte definição, que decorre naturalmente da observação acima.

<u>Definição 4.3</u>: Dado A, seu $D_{\mathbb{C}}$ e o transformador T de A, então $D_{\mathbb{C}}$ de T(A) denotada por $T(D_{\mathbb{C}})$ é tal que:

1) os rótulos superior e inferior são inalterados
2) Ignorando os rótulos, $T(D_{\mathbb{C}}) = T.D_{\mathbb{C}}$ .

ex. No ex.3.3. tivemos:

$$D_{\mathbb{C}} = \begin{pmatrix} \overbrace{0}^{a} & \overbrace{0}^{b} & \overbrace{1}^{c} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ \underbrace{0}_{a} & \underbrace{1}_{b} & \underbrace{0}_{c} & \underbrace{0}_{a} & \underbrace{0}_{b} \end{pmatrix}$$

e no ex. 4.1. temos

$$D'_{\mathbb{C}} = T(D_{\mathbb{C}}) = \left( \begin{array}{ccccc} \overbrace{1}^{a} & \overbrace{1}^{b} & \overbrace{1 \quad 0 \quad 0}^{c} \\ 1 & 0 & 0 & 0 & 0 \\ \underbrace{0}_{a} & \underbrace{1}_{b} & \underbrace{0}_{c} & \underbrace{0}_{a} & \underbrace{0}_{b} \end{array} \right) .$$

Na Fig. 4.4.,temos o grafo orientado de computação correspondente ao algoritmo A'. Comparando a Fig. 4.4.e a Fig. 4.2 combinada com a Fig. 3.2.2. e a Fig. 3.6, percebemos que na essência nada foi mudado: a ordem de execução e a dependência entre pontos de $\mathbb{C}I$ continuam as mesmas e apenas mudaram os índices.

Os resultados do Capítulo III continuam válidos para algoritmos transformados.

Dados A, seu D e o transformador T de A, podemos notar que T(D) do algoritmo T(A) será dado por T.D. Voltando para o ex.4.1. vemos que T(D) será:

$$T(D) = \left( \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right)$$

Isto mostra que T(A) está nas condições do Corolário 2 (como a linha $\ell_1 \in L^+$). Assim, podemos executar T(A) com ordenação linear sobre a coordenada K com deslocamento 1. De fato, para cada nível de altura K, os fluxos de dados vêm do nível imediatamente inferior, não tendo nenhum fluxo no mesmo nível.

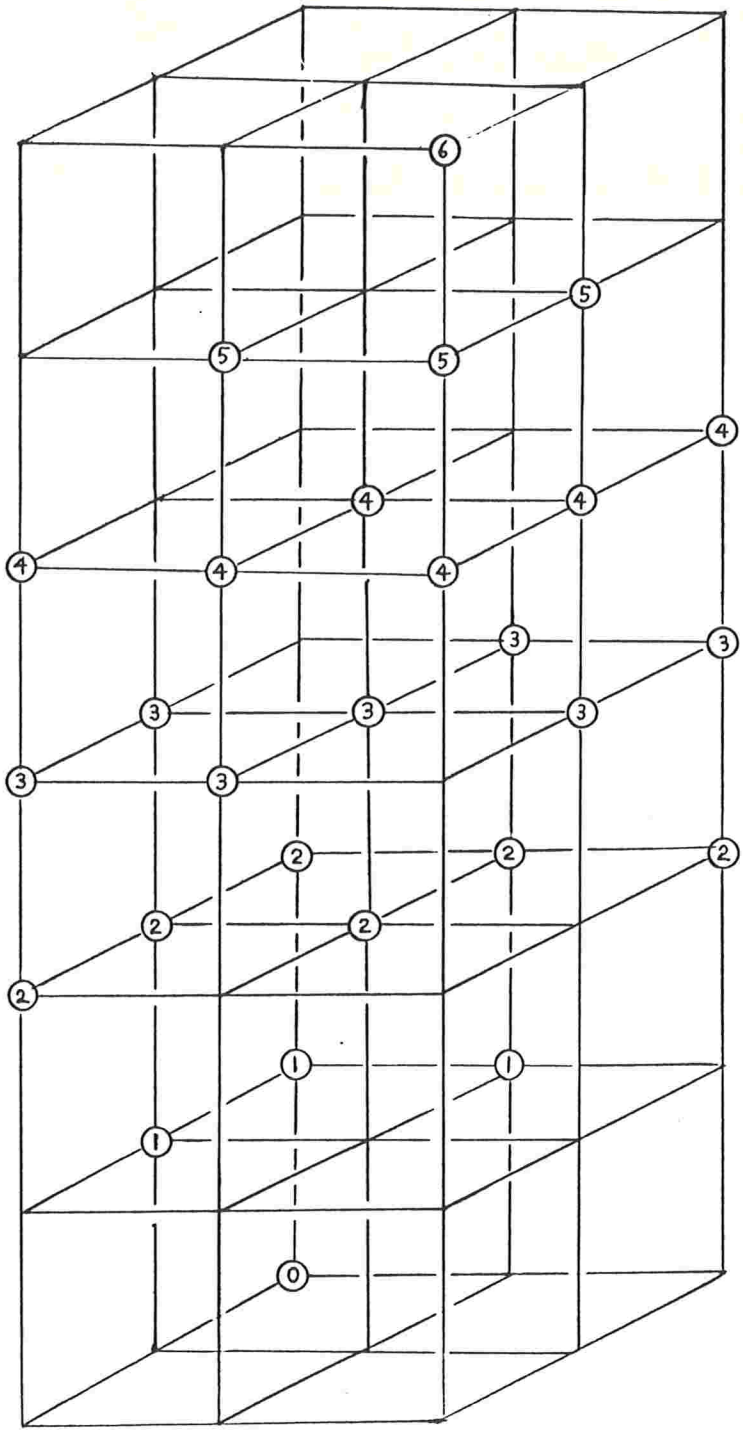



Fig. 4.5.

A Fig. 4.2. mostra que a ordem de execução é no sentido vertical de baixo para cima, executando-se primeiro o ponto marcado por 1 e em seguida executando-se os pontos marcados por 2,4 e 10 simultaneamente e assim por diante (ver também Fig. 4.5).

Notamos que a coordenada K está servindo como índice de tempo, pois no $T(\mathbb{CI})$ os pontos cujas coordenadas K valem $K_o$ são executados simultaneamente no $K_o$-ésimo passo.

Este exemplo é um caso particular de uma ordenação linear mas é um caso que tem a virtude de poder executar o algoritmo T(A), que no fundo é equivalente a A, de modo sistólico como veremos na seção seguinte. Daremos algumas nomenclaturas em relação a estas ordenações lineares particulares.

Definição 4.4.: Chamamos de *nível* $k_o$ o seguinte hiperplano em $\mathbb{R}^3$:

$$\{(K,I,J) \in \mathbb{R}^3 : K = K_o \in \mathbb{Z}\}$$

Quando não há perigo de ambiguidade, não faremos a distinção entre algoritmo A e T(A) e temos a seguinte nomenclatura:

Definição 4.5.: Dado $\mathbb{CI}$ de um algoritmo A, uma execução de A é *execução vertical sequencial* se as computações dos pontos situados no nível $K_o$ são executadas simultaneamente no $K_o$-ésimo passo, i.e., a execução de A é pela ordenação linear na coordenada K com deslocamento um, no mesmo sentido do eixo K.

Chamamos a atenção ao fato de que a numeração de ordem de execução de Def. 4.5. nem sempre começa com zero como mostra o seguinte exemplo:

Exemplo 4.2.: No lugar do algoritmo A do ex. 4.1. considere o seguinte algoritmo:

```
algoritmo A8:
 for K:= 2 to 4 do
  for I:= 0 to 2 do
   for J:= 0 to 2 do
    begin
     a(K,I,J) := b(K,I-1,J);
     b(K,I,J) := b(K,I,J-1);
     c(K,I,J):=c(K-1,I,J-1)+a(K,I,J)*b(K,I,J);
    end;
```

e considere o mesmo T do Ex.4.1.

É fácil ver que T(ℂI) deste exemplo é T(ℂI) do Ex.4.1. transladado verticalmente de duas unidades e a numeração de passos vai de 2 a 8.

Podemos imaginar um T que torne a ordem de execução no sentido inverso à coordenada K mas esta possibilidade será ignorada como veremos no Capítulo V.

Quando o mínimo H da primeira linha de D for maior que um, temos uma ordenação linear sobre coordenada K com H deslocamentos e temos a seguinte definição:

Definição 4.6.: Dado ℂI de um algoritmo A, uma execução de A é *execução vertical com H deslocamentos* se no $K_o$-ésimo passo são executados simultaneamente as computações dos pontos situados nos seguintes H níveis consecutivos:

$$\{\text{nível } \bar{K}\text{: } \bar{K} \text{ div } H = K_o\}$$

Neste caso, a coordenada K não representa exatamente os passos mas continua sendo um indicador de tempo.

Voltamos ao nosso Ex. 4.1. Comparando a fig. 3.10 e a Fig.4.5 e considerando como T levou cada ponto de $\mathbb{CI}$ ao ponto de T($\mathbb{CI}$), notamos que as numerações da Fig. 4.5. são exatamente as mesmas da Fig. 3.10.

Na essência, T não fez nada mais que mudar a disposição de $\mathbb{CI}$ em $\mathbb{R}^3$ de tal maneira que a ordem de execução seja "visível" pois agora a coordenada K representa o tempo como mostra o ex. 4.1.

Esta é, portanto, a idéia central desta secção:

1) Dado um algoritmo A cujo D tem a primeira linha com elementos positivos e alguns elementos nulos (note que elemento negativo não pode aparecer pois não temos anti-dependência);

2) Procuramos um T tal que T(D) tenha a primeira linha toda positiva;

3) Executamos T(A) por execução vertical sequencial ou com H deslocamentos conforme o caso.

A seguir, vamos tratar alguns detalhes em relação a esta idéia.

1. Com o transformador T, podemos obter uma ordenação de execução com menos passos do que outras ordenações obtidas apenas com os resultados do Capítulo III, como mostra o seguinte exemplo:

Exemplo 4.2.

```
algoritmo A9: for K:= 0 to 4 do
                  for I:= 0 to 4 do
                   for J:= 0 to 4 do
                    begin
                     a(K,I,J):= a(K-1,I-2,J) + a (K-1,I-2,J-1);
                     b(K,I,J):= b(K-1,I-2,J-1)+2*a(K-2,I-1,J);
                    end;
```

$$D = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

i) A já está na condição do Corolário 2 e podemos executar A em execução vertical sequencial.

ii) Podemos aplicar o Corolário 2 na coordenada I com deslocamento 1.

Agora vamos aplicar $T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ ao algoritmo A e temos

$$T(D) = \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 0 & 1 & 0 \end{pmatrix}$$

T(ℂI) intercepta com 9 níveis 0, 1, 2,...,8 com deslocamento 3. Logo podemos executar T(A) em apenas 3 passos ao passo que a ordem lexicográfica necessita de 125 passos e em i) ou ii) precisamos de 5 passos.

Deste modo, mesmo que um algoritmo A esteja nas condições do Corolário 2, o transformador T pode melhorar ainda mais o tempo de execução.

No capítulo seguinte iremos mostrar uma maneira de escolher o melhor transformador T que minimize o tempo.

2. O Transformador T do algoritmo A na sua representação matricial deve ter todos os seus elementos inteiros para que as coordenadas de T(ℤI) sejam todas inteiras e T deve ser bijeção por motivo óbvio de manter "equivalência" entre A e T(A).

3. Existência de T para um algoritmo A que não esteja nas condições do Corolário 2:

Teorema 3: Seja A um algoritmo cuja matriz de dependência D tem alguns elementos nulos na 1ª linha, então existe um transformador T do algoritmo A que torna a 1ª linha de T(D) toda positiva.

Demonstração: Em vez de considerar uma função linear bijetora T consideremos uma sequência de operações elementares sobre linhas de D e tomemos um T equivalente a esta sequência de operações elementares (Nering [63]).

Sejam $d_{j_1}, d_{j_2}, \ldots d_{j_m}$ colunas de D com 1º elemento nulo e $j_1 < j_2 < \ldots < j_m$. Observe-se que cada uma dessas colunas não pode ser toda nula pela definição de D a partir de $D_c$.

$$D = (\ldots d_{j_1} \ldots d_{j_2} \ldots\ldots d_{j_m} \ldots)$$

i) na coluna $d_{j_1} = (0, d_{2j_1}, d_{3j_1})^T$ pelo menos um dos $d_{2j_1}$ e $d_{3j_1}$ é positivo, pois caso contrário, ou $d_{j_1}$ é nulo ou temos anti-dependência.

Vamos supor que $d_{2j_1}$ é positivo. (Se apenas $d_{3j_1}$ é positivo, então no que se segue devemos substituir as palavras "2ª linha" por "3ª linha").

ii) Se somarmos a 2ª linha à 1ª linha, tornamos $j_1$-ésimo elemento da 1ª linha positiva mas ao mesmo tempo podemos tornar não positivos alguns outros elementos da 1ª linha.

Logo, faremos o seguinte: Seja $t = \min \{t_K \in \mathbb{N} / t_K * a_{1K} + a_{2K} > 0\}$ e multiplicamos a 1ª linha por t.

iii) Somemos 2ª linha à 1ª linha;

iv) Repetimos i) $\sim$ iii) a $d_{j_2}$ até $d_{j_m}$ ;

v) Escolhemos T que é o produto de matrizes correspondentes a esta sequência de operações elementares.

Como a multiplicação feita no passo ii) é por inteiro, T tem todos os seus elementos inteiros e T é bijetora, pois é produto de matrizes inversíveis.

4. Uma questão a ser levantada é a escolha de um transformador T conveniente, pois a simples existência de T garantida pelo Teorema 3 pode levar a um T muito ruim.

O que queremos é um T que proporcione um sistema sistólico mais eficiente possível.

Esta questão será tratada no Capítulo V. Vejamos agora como se constrói um sistema sistólico a partir de um algoritmo A e do transformador T.

oOo

# § 2 Sistema Sistólico

Dado um algoritmo A, o Teorema 3 da seção anterior garante a existência de um transformador T que permite a execução vertical sequencial ou com H deslocamentos.

Veremos agora como obter um sistema sistólico a partir de execução vertical.

Examinaremos primeiro o caso da execução vertical sequencial, uma vez que a execução vertical com H deslocamentos também pode ser executada como sequencial.

## 2.1. Execução vertical sequencial

Veremos 2 exemplos de construção de sistema sistólico a partir de um dado algoritmo A e um dado transformador T e em seguida formalizaremos a construção. A construção é bastante natural devido à observação de que cada nível $K_o$ é executado no $K_o$-ésimo passo.

a) Voltemos ao ex. 4.1.

i) como a coordenada K representa o tempo, podemos pensar que cada nível $K_o$ do T(ℂI) mostra o $K_o$-ésimo passo do sistema sistólico sendo que cada ponto de T(ℂI) situado no nível $K_o$ representa uma célula básica que está ativa neste passo. (Fig. 4.6.)

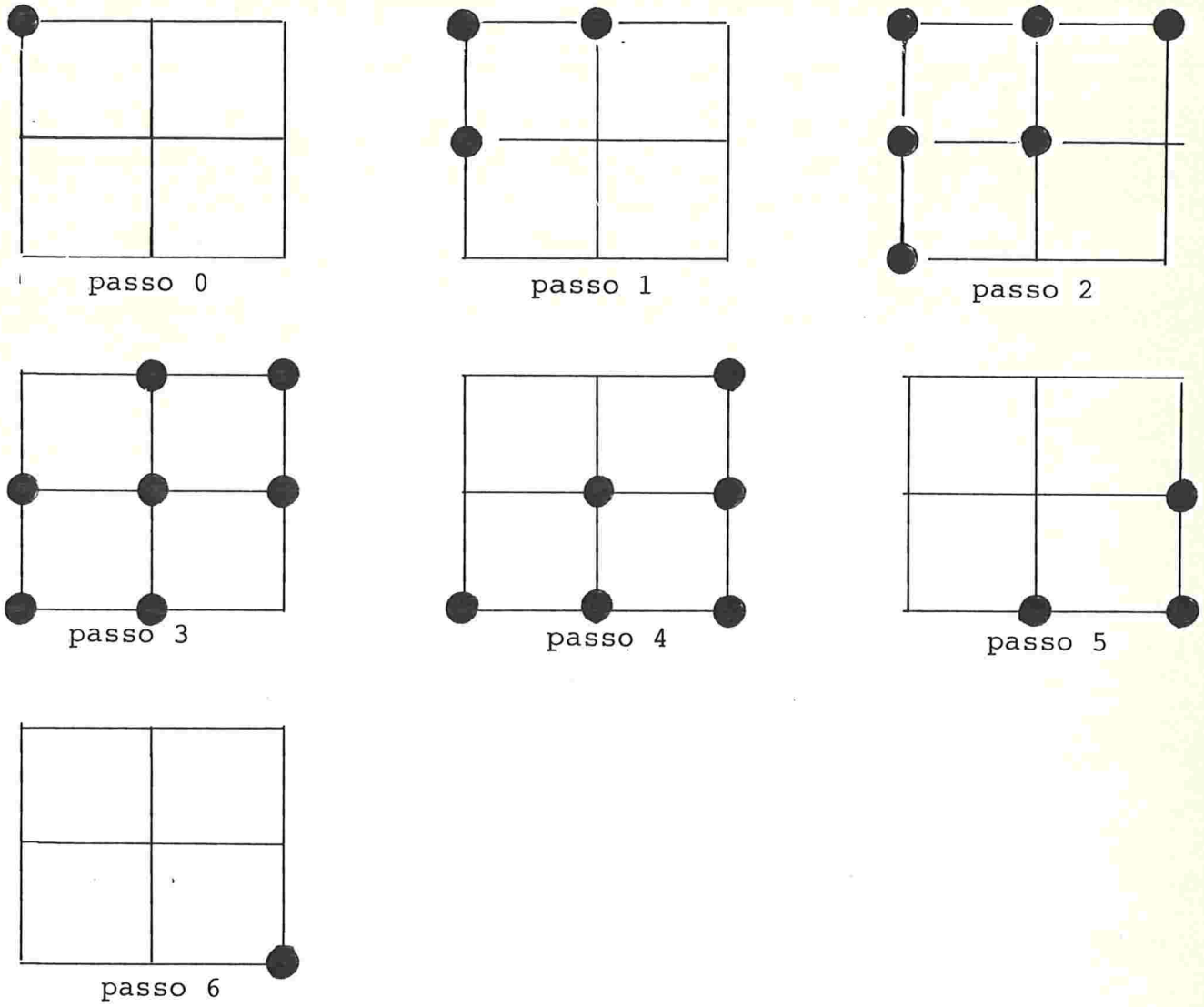



Fig. 4.6.

Assim sendo, o ponto (K,I,J) do T(ℂI) significa o K-ésimo passo da célula na posição (I,J) e Fig. 4.6 mostra que precisamos de 9 células básicas para nosso sistema sistólico, dispostas em matriz 3 × 3.

ii) interconexão entre células básicas:

O vetor $(1,1,0)^T$ de $T(D_c)$ representa fluxo de a que sai de (K-1,I-1,J) e chega a (K,I,J). Como K passou a representar o tempo, o vetor $(1,1,0)^T$ significa que o dado a que está na célula

(I-1,J) no (K-1)-ésimo passo entra na célula (I,J) no K-ésimo passo. Desta forma precisaremos ligação da célula (I-1,J) para a célula (I,J).

Analogamente temos ligação da célula (I,J-1) para a célula (I,J) proveniente do vetor $(1,0,1)^T$.

Por outro lado, o vetor $(1,0,0)^T$ significa que o dado C fica estacionado na célula.

Como estas observações valem para qualquer célula (I,J), nós temos as mesmas ligações para todas as células, o que é uma característica desejável do sistema sistólico.

iii) A Fig. 4.4. é um grafo orientado de computação para T(A). Não temos arestas que entram neste grafo indicando dados que vêm de fora, mas pela regularidade de dependência podemos conhecê-los facilmente.

A Fig. 4.7. mostra o fluxo de dados a e b em cada passo no sistema sistólico, omitindo-se os índices.

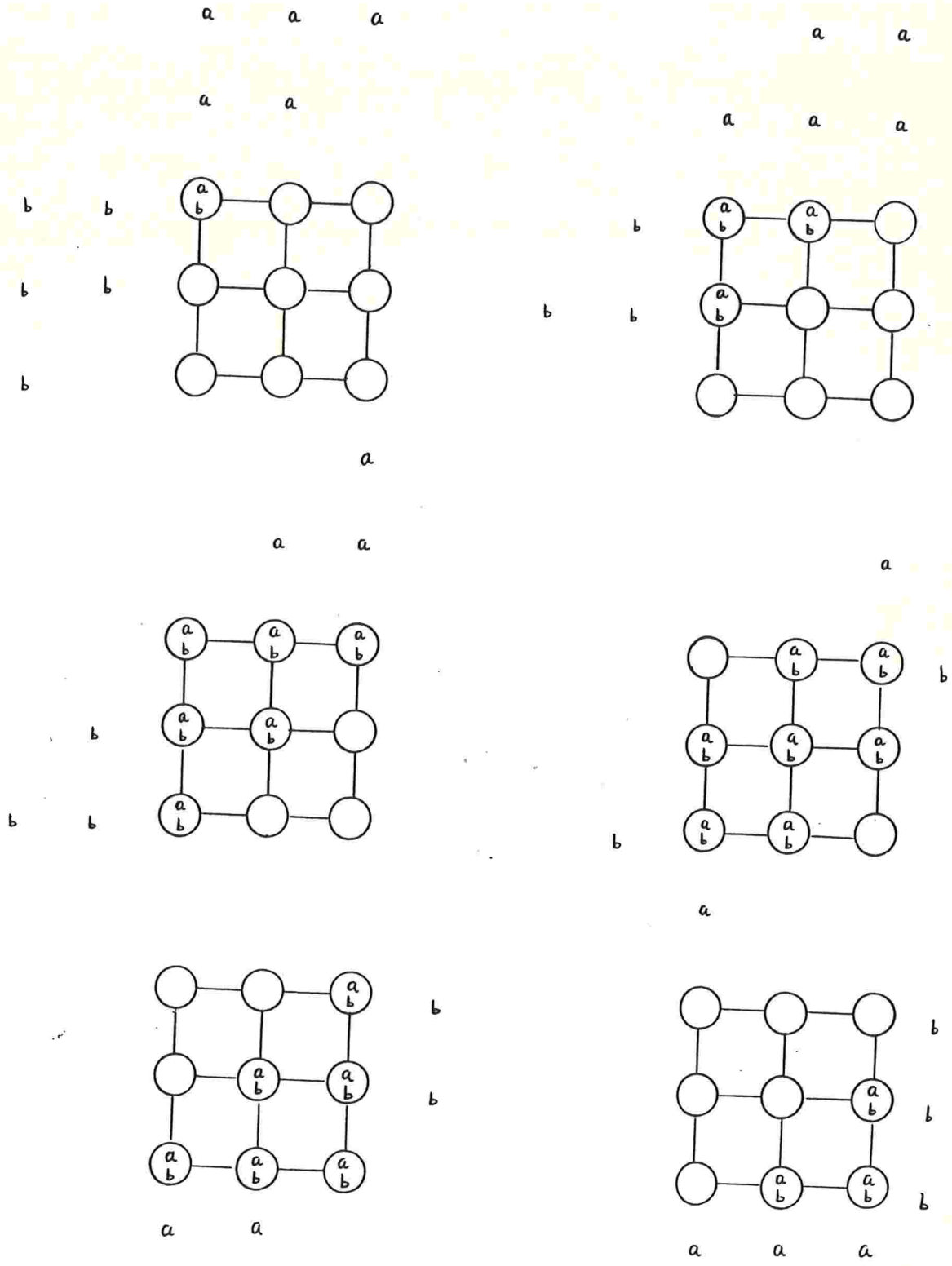


Fig. 4.7.

A Fig. 4.7 nada mais é que a versão 3 × 3 da Fig. 2.13 (exceto a posterior saída dos elementos C), mostrando que o trabalho de Okuda-Song é o resultado de uma transformação particular do algoritmo de multiplicação matricial.

b) Ex. 4.2.

Vejamos agora um exemplo em que retardos ou *"delay"* convenientes devem ser inseridos no fluxo de dados do sistema sistólico. Tais *"delays"* estão relacionados com os valores da 1ª linha da matriz $T(D_C)$ ou $T(D)$.

```
Algoritmo A:  for K:= 0 to 2 do
               for I:= 0 to 2 do
                for J:= 0 to 2 do
                 begin
                  a(K,I,J):= a(K,I-1,J)+2*b(K-1,I-1,J);
                  b(K,I,J):= b(K,I-1,J-1)+a(K-1,I,J);
                 end;
```

Para este exemplo temos

$$\mathbb{I} = \{(K,I,J) \in \mathbb{Z}^3 : 0 \leq K,I,J \leq 2\}$$

$$D_C = \begin{matrix} \overbrace{\phantom{00}}^{a} & & \overbrace{\phantom{00}}^{b} & \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ \underbrace{\phantom{0}}_{a} \; \underbrace{\phantom{0}}_{b} \; \underbrace{\phantom{0}}_{b} \; \underbrace{\phantom{0}}_{a} \end{matrix}$$

Na Fig. 4.8. temos $\mathbb{CI}$ e ilustramos 4 vetores de dependência entrando no ponto (1,1,1).

Agora, consideremos o seguinte T:

$$T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

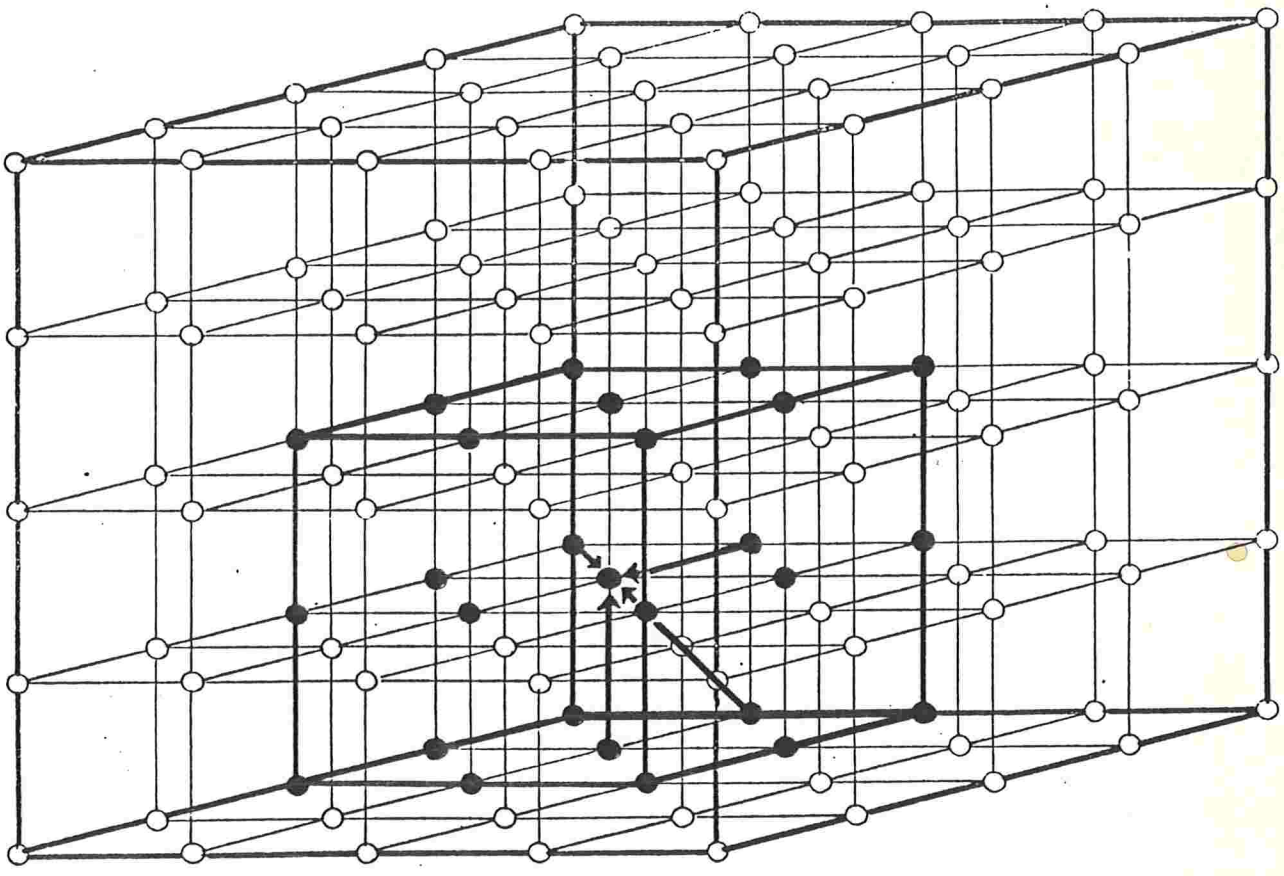


Fig. 4.8.

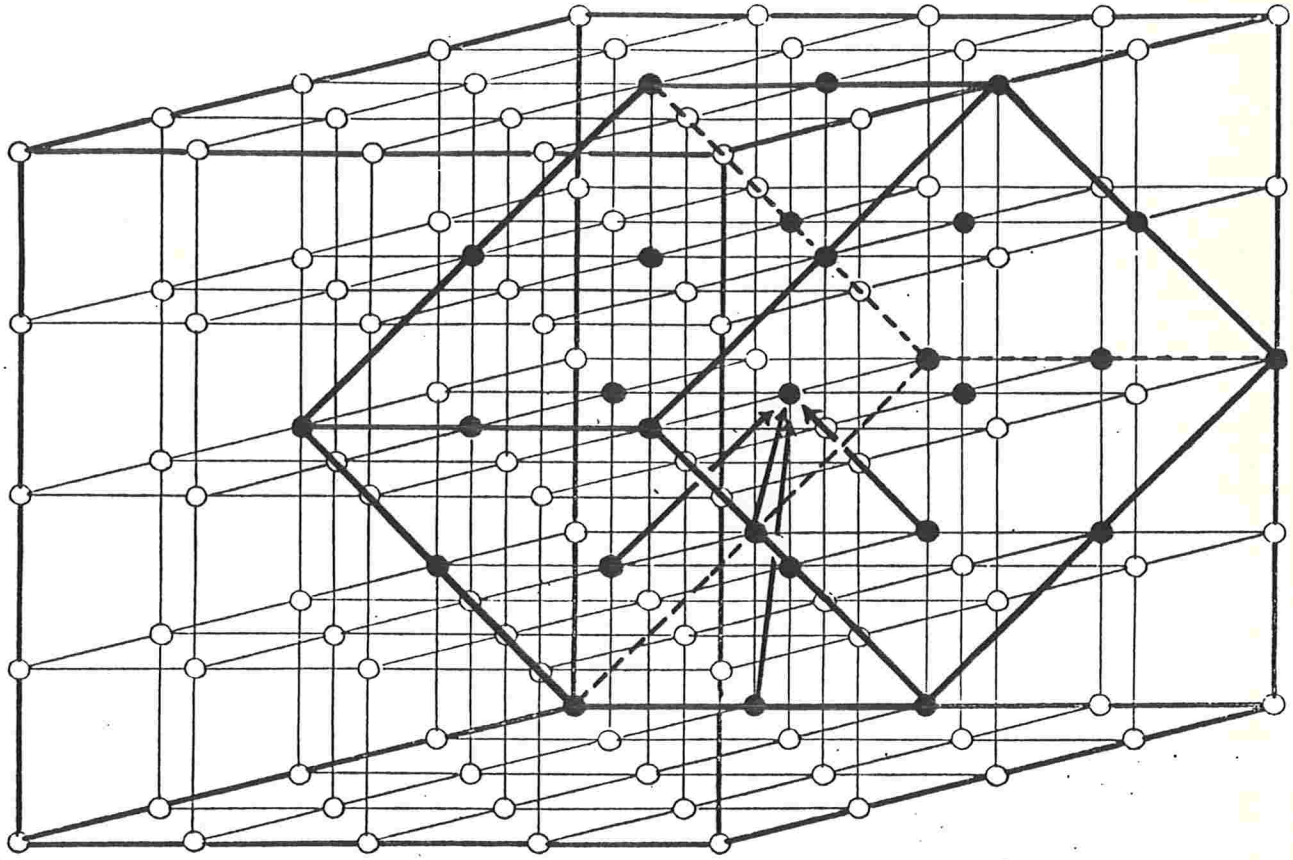
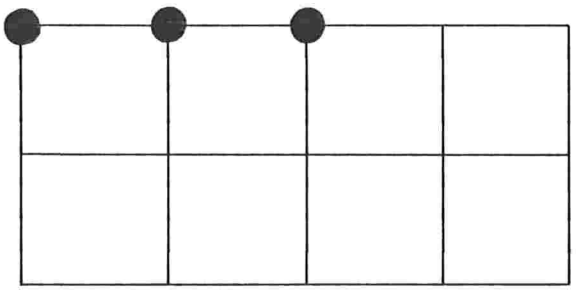


Fig. 4.9.

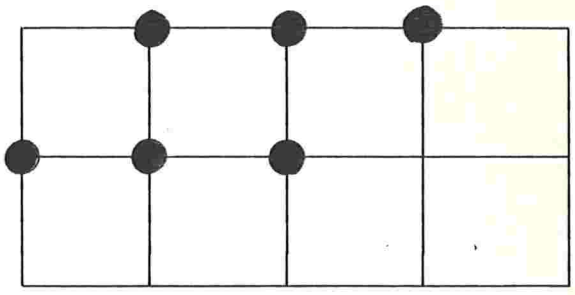
$$T(D_C) = \begin{pmatrix} \overbrace{1}^{a} & \overbrace{2}^{a} & \overbrace{1}^{b} & \overbrace{1}^{b} \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ \underbrace{a} & \underbrace{b} & \underbrace{b} & \underbrace{a} \end{pmatrix}$$

i) Na Fig. 4.9. temos $T(\mathbb{C}I)$ e 4 vetores de dependência entrando em $(2,1,2) = T[(1,1,1)^T]$.

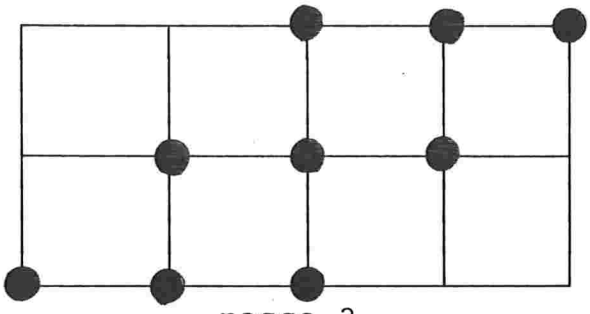
Nós temos 5 passos como mostra a Fig. 4.10, onde em cada passo mostramos as células ativas.



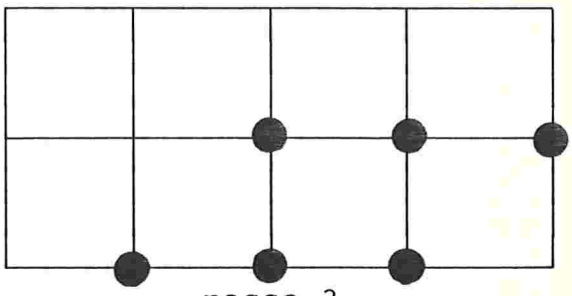
passo 0



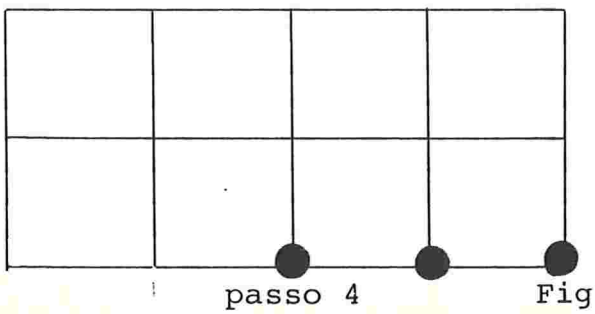
passo 1



passo 2



passo 3



passo 4

Fig. 4.10.

A Fig. 4.10 mostra também que precisamos de 15 células básicas dispostas em Matiz 3 × 5.

ii) interconexão entre células básicas:

De modo análogo ao exemplo anterior, os vetores $(1,1,0)^T$, $(1,1,1)^T$ e $(1,0,1)^T$ de $T(D_C)$ correspondem a 3 ligações:

1) de (I-1,J) à (I,J) para dado a
2) de (I,J-1) à (I,J) para dado a
3) de (I-1,J-1) à (I,J) para dado b

Quanto ao vetor $(2,1,1)^T$, ele significa que o dado b da célula (I,J) provém da célula (I-1,J-1) de 2 passos atrás.

Este dado b deve então sair da célula (I-1,J-1) e sofrer um retardo ou *"delay"* de um passo antes de participar das operações da célula (I,J).

Não precisamos de uma 4ª ligação ligando célula (I-1,J-1) à célula (I,J), pois a 3ª ligação acima une as mesmas células para o mesmo fluxo de dado b.

O esquema da célula básica está ilustrado na Fig. 4.11. O sistema sistólico com os 3 fluxos de dados está mostrado na Fig. 4.12.

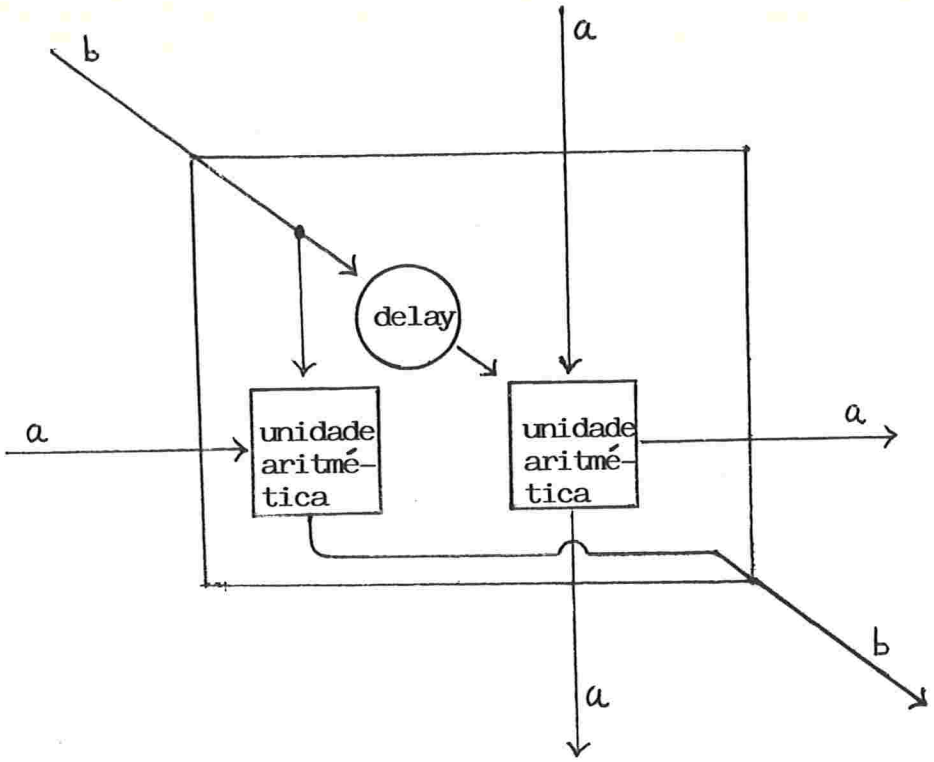



Fig. 4.11.

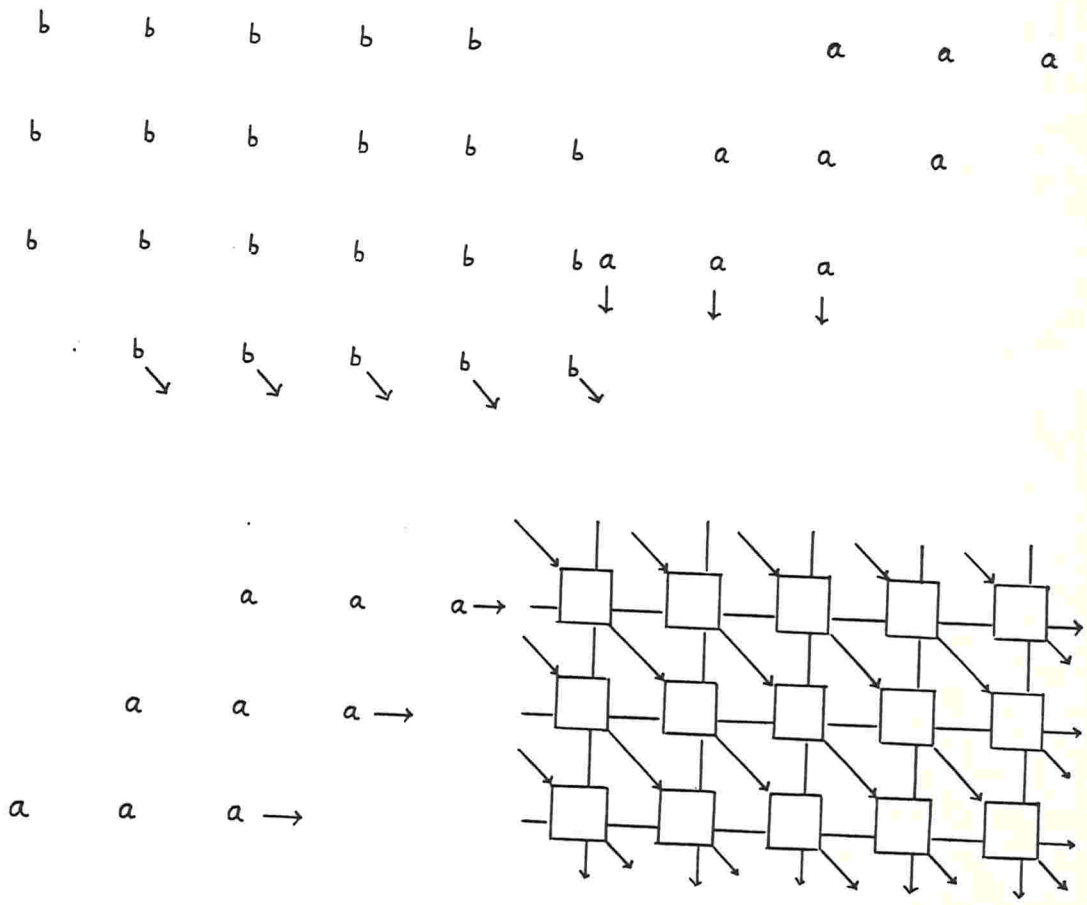



Fig. 4.12.

Observação: Seja um algoritmo A':

```
for K:= 0 to 2 do
 for I:= 0 to 2 do
  for J:= 0 to 2 do
   begin
   a(K,I,J):= f[a(K,I-1,J), b(K-1,I-1,J)];
   b(K,I,J):= g[a(K-1,I,J), b(K,I-1,J-1)];
   end;
```

$\mathbb{CI}$ e $D_C$ de A' são os mesmos de A. Se aplicarmos o mesmo T então teremos um sistema sistólico análogo, ao do ex. 4.2., mudando apenas detalhes internos da célula básica referentes à unidades aritméticas para a computação de f e g.

c. formalização:

Sejam dados um algoritmo A e seu transformador T que torna possível a execução vertical sequencial de T(A) possível.

i) Disposição de células básicas:

Seja Proj: $\mathbb{R}^3 \to \mathbb{R}^2$ tal que Proj (K,i,j) = (i,j).

Então Proj (T($\mathbb{CI}$)) será o conjunto de todas as coordenadas das células básicas do sistema sistólico para executar A.

ii) Número de passos necessários para a execução:

Sejam $K_{min} = \min\{K: (K,I,J) \in T(\mathbb{CI})\}$ e

$K_{max}=\max\{K: (K,I,J) \in T(\mathbb{CI})\}$, então o sistema sistólico necessita de $(K_{max}-K_{min}+1)$ passos.

Em cada passo $K_o$ ($K_{min} \leq K_o \leq K_{max}$) são ativadas as seguintes células básicas:

$$\{(K,I,J) \in T(\mathbb{CI}) : K = K_o\}$$

Observação: este é o número de passos em que unidades aritméticas das células básicas estão operando efetivamente.

Não inclui o retardo necessário para a entrada inicial de dados.

iii) interconexão entre células básicas e estrutura interna:

Considere o seguinte $T(D_C)$:

$$T(D_C) = \left( \begin{array}{cccc} \overbrace{d_{11} \quad d_{12} \cdots d_{1s}}^{a_1} & \overbrace{\quad}^{a_2} & \cdots & \overbrace{\quad}^{a_s} \\ d_{21} \quad d_{22} \cdots d_{2s} & & & \\ d_{31} \quad d_{31} \cdots d_{3s} & & \cdots & \\ \underbrace{\quad}_{a_1} \quad \underbrace{\quad}_{a_2} \quad \underbrace{\quad}_{a_s} & & & \end{array} \right)$$

Mostramos como são as interconexões para a variável $a_1$. A mesma idéia aplica-se a $a_2,\ldots,a_s$.

As S primeiras colunas de $T(D_C)$ correspondem à seguinte computação de $a_1$ de T(A):

$$a_1(K,I,J) := f_1\{a_1(K-d_{11},I-d_{21},J-d_{31}),\ a_2(K-d_{12},I-d_{22},J-d_{32}) \ldots \ldots a_s(K-d_{1s},I-d_{2s},J-d_{3s})\}$$

Para cada célula (I,J) temos ligações vindas de células $(I-d_{2\ell},J-d_{3\ell})$ correspondentes a $a_\ell$, $1 \leq \ell \leq S$, todas elas entrando numa unidade aritmética que efetua computação de $f_1$.

Da unidade aritmética temos ligações indo para células $(I+d_{2\ell}, J+d_{3\ell})$, $1 \leq \ell \leq S$.

Se algum vetor $(d_{1\ell},d_{2\ell},d_{3\ell})$, $1 \leq \ell \leq S$, tiver $d_{1\ell} > 1$, então as ligações correspondentes a $a_\ell$ deve ter um retardo ou *"delay"* de $(d_{1\ell} - 1)$ passos antes de entrar na unidade aritmética correspondente.

## 2.2. Execução vertical com H deslocamentos

Para ilustrar este caso, usemos um exemplo bem simples para que as figuras mais simples ajudem a captar melhor as idéias básicas.

Ex. 4.3. Algoritmo A :

```
for K:= 0 to 2 do
    for I:= 0 to 2 do
     for J:= 0 to 2 do
      begin
       a(K,I,J):= a(K-1,I,J-1) + b(K-1,I-1,J);
       b(K,I,J):= b(K-1,I-1,J)*2
      end;
```

$$D = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\text{Considere} \quad T = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{temos } T(D) = \begin{pmatrix} 2 & 2 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

A Fig. 4.13 mostra 6 passos de execução vertical sequencial do algoritmo A onde o pequeno retângulo do canto superior esquerdo de cada célula representa um retardo ou um *"delay"* de um passo.

A mistura de letras maiúsculas e minúsculas é proposital, apenas para distinguir instâncias diferentes dos fluxos a e b.

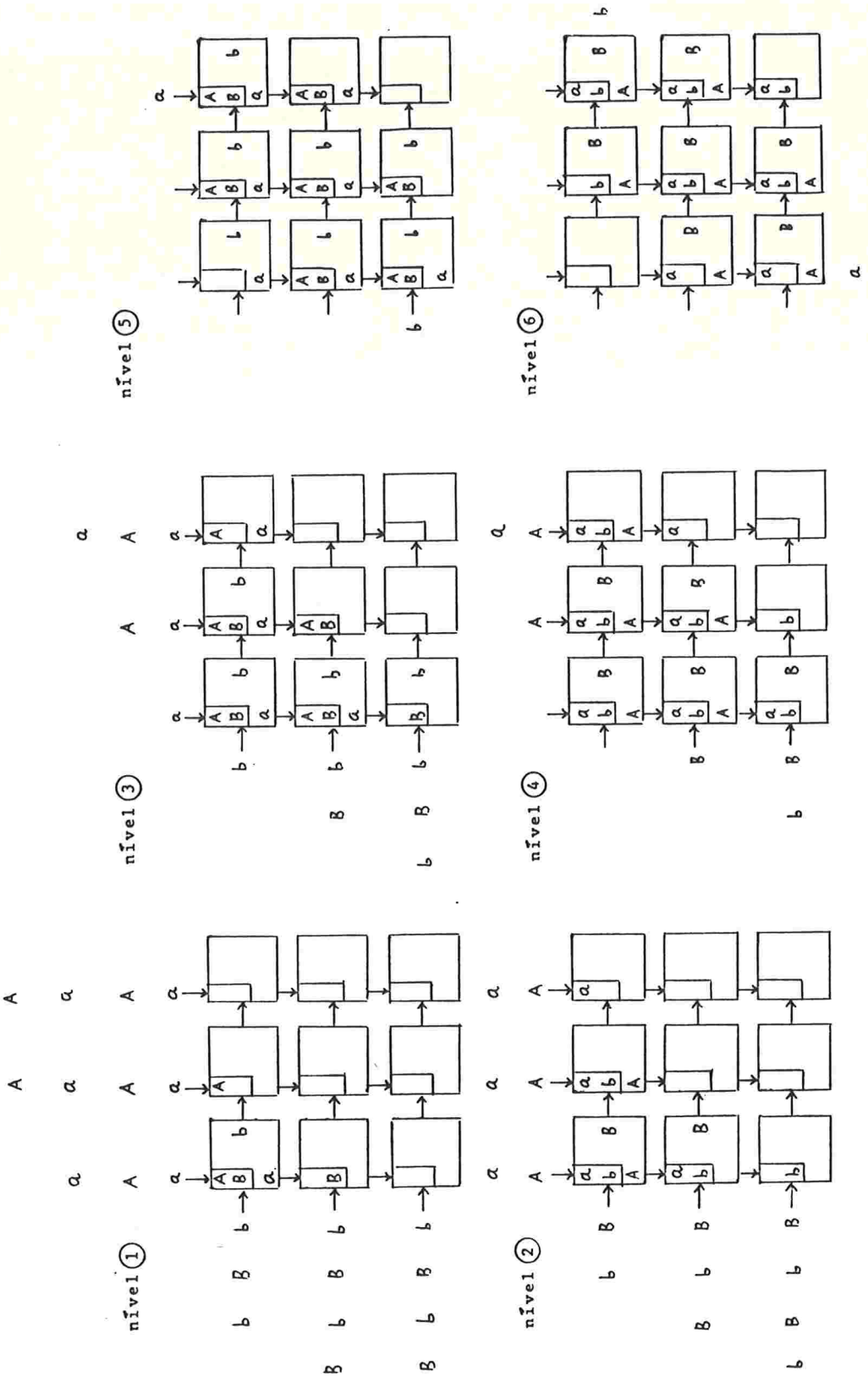



Fig. 4.13.

A execução vertical com 2 deslocamentos será feita da seguinte maneira:

Os níveis 1 e 2 da Fig. 13 correspondem ao passo 1 da Fig. 4.14. Observe-se que no nível 1, a operação aritmética envolve a e b e no nível 2, A e B. No passo 1 da Fig. 4.14, entretanto, o par A e B e o par a e b participam em operações aritméticas simultaneamente.

Assim, os níveis 1 e 2 são executados em um passo. Analogamente, os níveis 3 e 4 são executados no passo 2, e assim por diante.

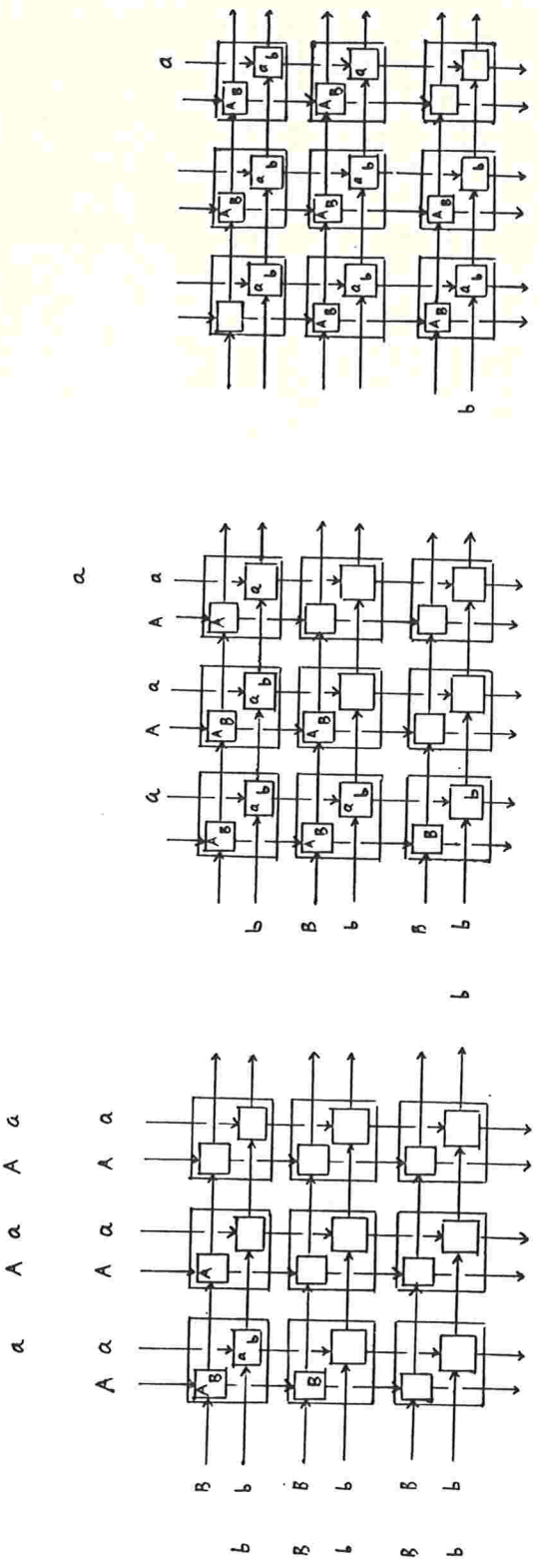



Fig. 4.14.

A passagem da execução vertical sequencial para execução vertical com deslocamento 2 foi feita da seguinte maneira:

i) Cada célula básica da Fig. 4.14 é composta de duas partes idênticas que por sua vez são iguais às céulas básicas da Fig.4.13, sem o dispositivo *"delay"*.

ii) Na Fig. 4.14 temos 2 fluxos (de b minúsculo e de B maiúsculo) no sentido horizontal.

iii) No sentido vertical ocorre o mesmo.

iv) O passo 1 da Fig. 4.14 é uma superposição do nível 1 e nível 2 da Fig. 4.13, as duas partes que compõem a célula básica executam as operações aritméticas correspondentes ao nível 1 e 2 simultaneamente, sem necessidade do *"delay"*.

# Capítulo 5

# A Escolha do Transformador Ótimo

## §.1 Algumas Considerações Iniciais

No capítulo anterior, observamos através de alguns exemplos que, dado um algoritmo A, um transformador T adequado nos proporcionou um sistema sistólico que executa T(A), equivalente a A, paralelamente e nossa tarefa agora é como escolher este T *"adequado"*.

Temos 2 critérios para julgar se T é bom ou não: o tempo que o sistema sistólico leva para executar T(A) e o espaço ocupado pelo sistema sistólico.

Sejam $T = \begin{pmatrix} a & b & c \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$ um transformador de um algoritmo A

e $p = (K,i,j)$ um ponto de $\mathbb{I}$ então $T(p) = (aK + bi + cj, x_{21}K + x_{22}i + x_{23}j , x_{31}K + x_{32}i + x_{33}j)$. Como vimos no capítulo IV, a primeira coordenada de T(p) indica o passo no qual a computação é feita e as duas coordenadas restantes indicam as coordenadas da célula na qual a computação é feita.

Em outras palavras, a primeira linha de T refere-se ao tempo e as duas últimas linhas referem-se ao espaço. Adotemos a seguinte nomenclatura:

Definição 5.1.: Dado um transformador de algoritmo

$$T = \begin{pmatrix} a & b & c \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$

$\pi = (a, b, c) : \mathbb{R}^3 \to \mathbb{R}$ é chamada *função tempo* e

$$S = \begin{pmatrix} x_{21} & x_{22} & x_{23} \\ & & \\ x_{31} & x_{32} & x_{33} \end{pmatrix} : \mathbb{R}^3 \to \mathbb{R}^2$$ é chamada *função espaço*

Queremos $\pi$ que minimize o tempo de execução, i.e.,o número de passos de computação e S que minimize o espaço, i.e.,o número de células do sistema sistólico e consequentemente a área para dispor as células numa pastilha.

Em muitos algoritmos iterativos definidos por recorrências encontramos frequentemente casos em que um valor $a^{(K)}$ depende de $a^{(k-1)}$, i.e., $a^{(K-1)}$ é um dos argumentos de uma função que determina o valor de $a^{(K)}$. Num caso mais geral, o valor $a_p^{(K)}$ da iteração k depende de algum valor $a_q^{(K-h)}$ da iteração K-h.

Definição 5.2.: Seja A um algoritmo. Diremos que A é um algoritmo *com o uso de iteração anterior* ($A^c$/UIA) se o corpo do algoritmo tiver o seguinte comando.

$$a_p(K,i,j) = f[\ldots,a_q(K-h,i,j),\ldots] \ , \ h \in \mathbb{N}$$

Se h = 1 diremos que A é um algoritmo *com o uso de iteração imediatamente anterior* ($A^C$/UIIA).

Se p = q diremos que A é um algoritmo *com o uso de iteração anterior próprio* ($A^C$/UIAP).

E, finalmente, se h = 1 e p = q diremos que A é um algoritmo *com uso de iteração imediatamente anterior próprio* ($A^C$/UIIAP).

Seja A um algoritmo $A^C$/UIA, então a sua matriz de dependências D tem uma coluna $(h,0,0)^T$ com $h \in \mathbb{N}$ e a presença do vetor do tipo $(h,0,0)^T$ implica que a função tempo $\pi = (a,b,c)$ deve ter $a > 0$ e neste caso podemos decompor T da seguinte maneira:

$$T = \begin{pmatrix} a & b & c \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ x_{21}/a & x_{22}-bx_{21}/a & x_{23}-cx_{21}/a \\ x_{31}/a & x_{32}-bx_{31}/a & x_{33}-cx_{31}/a \end{pmatrix} \cdot \begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Definição 5.3.: a matriz

$$\begin{pmatrix} 1 & 0 & 0 \\ x_{21}/a & x_{22}-bx_{21}/a & x_{23}-cx_{21}/a \\ x_{31}/a & x_{32}-bx_{31}/a & x_{33}-cx_{31}/a \end{pmatrix}$$ é chamada de $T_3$

e a matriz $\begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ é chamada de T'

É fácil ver que a transformação T' trata de deformação no sentido vertical (no eixo K) do conjunto de índices $\mathbb{I}$ do A sem mudar i e j e a transformação $T_3$ trata de deformação no sentido horizontal do mesmo, sem mudar k. Em outras palavras, T' trata de tempo e $T_3$ trata de espaço.

Também é fácil ver que T' é uma matriz inversível mas nada podemos garantir quanto a $T_3$, a inversibilidade de $T_3$ será tratada na seção 3.

Assim, para um algoritmo $^{C}$/UIA podemos pensar em otimizar tempo via T' e depois otimizar espaço via $T_3$. A Fig. 5.1. mostra graficamente seu efeito.

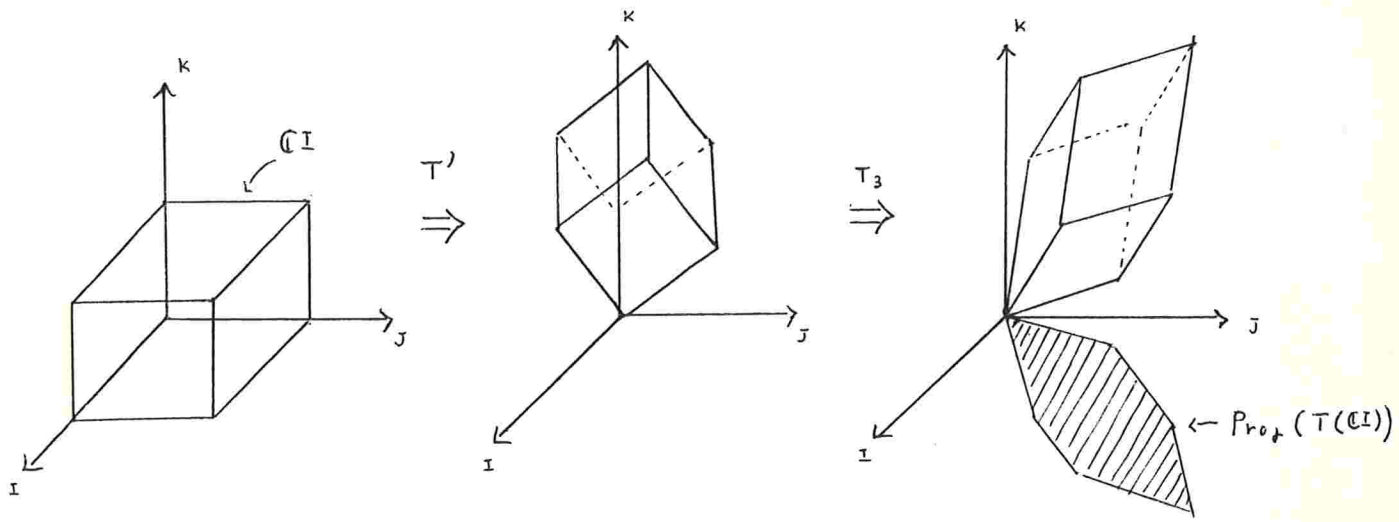



Fig. 5.1.

## §.2 Otimização de Tempo

Sejam $A^c$/UIA, T, $\pi$ e $\mathbb{CI}$ como foram definidos anteriormente, então o tempo de execução t de T(A) é dado por $t=\max\pi(P_1)-\min\pi(P_2)+1$, onde $P_1, P_2 \in \mathbb{CI}$, i.e.,igual ao número de níveis que interceptam com T($\mathbb{CI}$), no caso de execução vertical sequencial e

$$t = \left\lceil \frac{\max\pi(P_1)-\min\pi(P_2) + 1}{\min\pi(d_i)} \right\rceil$$

onde $d_i$ é vetor de dependência de T(A) no caso de execução vertical com H deslocamentos onde $H = \min \pi(d_i)$.

Prosseguimos a nossa estratégia de minimizar o t da seguinte maneira:

1ª etapa: Consideremos apenas a execução vertical sequencial e achemos $\pi$ que minimize $t_1$ onde $t_1 = \max \pi (P_1) - \min \pi (P_2)$.

2ª etapa: Consideremos a execução vertical com deslocamento e achemos $t = \left\lceil \frac{\max \pi (P_1) - \min \pi (P_2) + 1}{\min \pi (d_i)} \right\rceil$ usando as informações da 1ª etapa.

Daremos a seguir os detalhes.

### 2.1. Otimização para execução vertical sequencial

Prosseguimos da seguinte maneira:

i) Escolhemos um T' provisório chamado $\bar{T}'$ que possibilita execução vertical sequencial

ii) Usando este $\bar{T}'$ como teto superior, procuramos o melhor T'

2.1.1. Como Escolher $\bar{T}'$:

Vamos dividir vetores de dependência em 2 grupos.

Definição 5.4.: Seja A um algorítmo com três malhas encaixadas. O vetor de dependência com o 1º elemento positivo é chamado *vetor com K-dependência* e o vetor de dependência com o 1º elemento nulo é chamado *vetor sem K-dependência*.

A ação de T' deve transformar vetores sem K-dependência a vetores com K-dependência, i.e., falando geometricamente, T' "*levanta*" os vetores paralelos ao plano ij. Porém T' deve manter vetores com K-dependência.

T' pode ser decomposto em duas partes, como segue:

$$T' = \begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Definição 5.5.:

$$\begin{pmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ acima é chamado $T_1$,

$$\begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ acima é chamado $T_2$ .

Também é fácil ver que $T_1$ e $T_2$ são inversíveis e $T_1$ "estica" $\mathbb{C}I$ no sentido vertical e ela não tem nenhum efeito a vetores sem K-dependência, i.e. se d é sem K-dependência $T_1(d) = d$

Temos a seguinte decomposição de T:

$$T = T_3 . T' = T_3 . T_2 . T_1$$

$$\text{onde } T_1 = \begin{pmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$T_2 = \begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$T_3 = \begin{pmatrix} 1 & 0 & 0 \\ x_{21}/a & x_{22}-bx_{21}/a & x_{23}-cx_{21}/a \\ x_{31}/a & x_{32}-bx_{31}/a & x_{33}-cx_{31}/a \end{pmatrix}$$

$T_1$ estica $\mathbb{C}I$ verticalmente,

$T_2$ deforma $T_1(\mathbb{C}I)$ verticalmente sem mudar i,j,

$T_3$ deforma $T_2T_1(\mathbb{C}I)$ horizontalmente sem mudar K. (ver Fig. 5.2.)

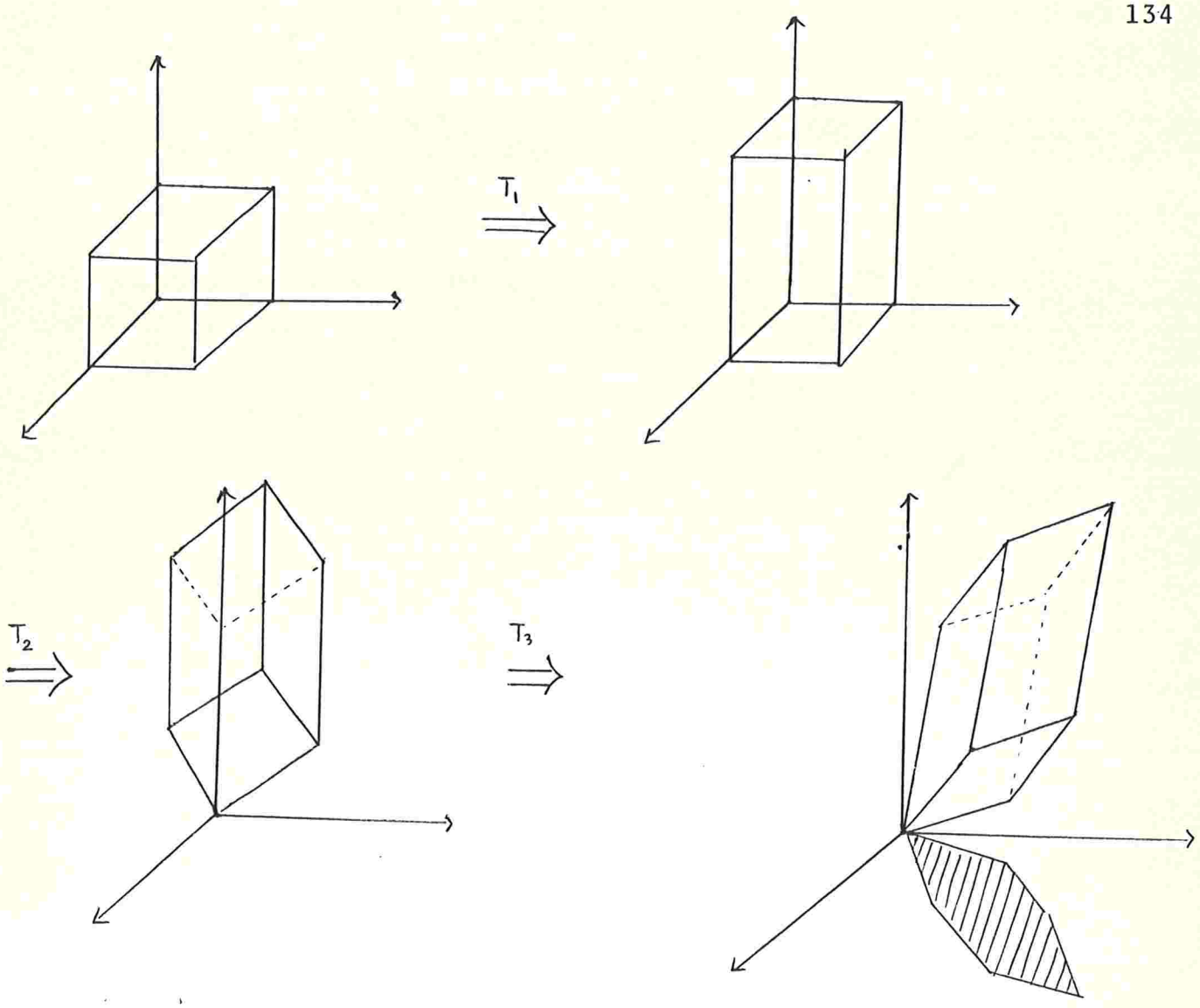



Fig. 5.2.

A escolha de $\bar{T}'$ será feita da seguinte maneira:

i) procuramos um $T_2$ que levante todo vetor sem K-dependência assumindo por enquanto $T_1 = I$ e consequentemente $\bar{T}' = T_2$.

ii) se este $\bar{T}'$ mantém todos os vetores com K-dependência, então ele será escolhido.

iii) Eventualmente pode ocorrer que $T_2$ *"abaixe"* alguns vetores com K-dependência transformando-os em vetores sem K-dependência ou em vetores de anti-dependência. Neste caso, então escolhemos um $T_1$ com a conveniente que permita $T_1$ compensar esta ação de $T_2$ .

a) Escolha de $T_2$

Como o algoritmo A não tem anti-dependência, os vetores sem K-dependência de A deve ser um dos seguintes 4 tipos (ver Fig. 5.3.)

$$\text{I:} \begin{pmatrix} 0 \\ 0 \\ \beta \end{pmatrix} \qquad \text{II:} \begin{pmatrix} 0 \\ \alpha \\ \beta \end{pmatrix} \qquad \text{III:} \begin{pmatrix} 0 \\ \alpha \\ 0 \end{pmatrix} \qquad \text{IV:} \begin{pmatrix} 0 \\ \alpha \\ -\beta \end{pmatrix}$$

onde $\alpha, \beta \in \mathbb{N}$

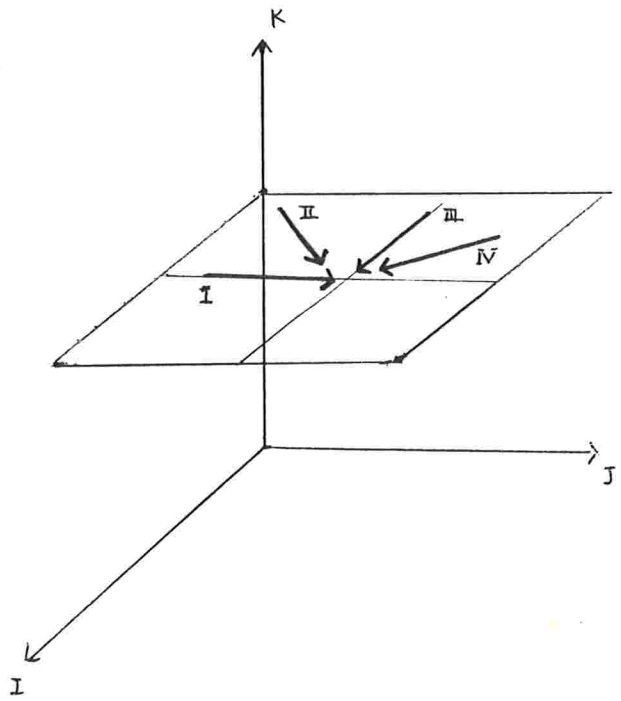



Fig. 5.3.

Seja $T_2 = \begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, então o efeito de $T_2$ sobre cada um dos 4 tipos de vetor sem K-dependência é o seguinte:

I) $T_2 \begin{pmatrix} 0 \\ 0 \\ \beta \end{pmatrix} = \begin{pmatrix} \beta c \\ 0 \\ \beta \end{pmatrix}$

II) $T_2 \begin{pmatrix} 0 \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha b + \beta c \\ \alpha \\ \beta \end{pmatrix}$

III) $T_2 \begin{pmatrix} 0 \\ \alpha \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha b \\ \alpha \\ 0 \end{pmatrix}$

IV) $T_2 \begin{pmatrix} 0 \\ \alpha \\ -\beta \end{pmatrix} = \begin{pmatrix} \alpha b - \beta c \\ \alpha \\ -\beta \end{pmatrix}$

onde $\alpha b - \beta c > 0$

Inspecionando este efeito, temos os seguintes passos para ter um $T_2$ que levante todos os vetores sem K-dependência.

1) $T_2 = \begin{pmatrix} 1 & b_o & c_o \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ onde $b_o$ e $c_o$ são definidos como se segue.

2) Se existem vetores sem K-dependência do tipo I, então $C_o = 1$ senão $C_o = 0$.

3) Se $C_o = 1$ e existem vetores sem K-dependência do tipo IV, então $b_o = \min\{b \in \mathbb{N} / \alpha_i b - \beta_i C_o > 0$ para todo vetor sem K-dependência do tipo IV $(0, \alpha_i, -\beta_i)\}$

Se $C_o = 1$ e não existem vetores sem K-dependência do tipo IV, então $b_o = 0$.

Se $C_o = 0$ ou existem vetores sem K-dependência do tipo II, então $b_o = 1$.

Observação: Notemos que foram escolhidos b e c cujos valores absolutos eram os menores possíveis e seu motivo será visto em 2.1.4.

b) Escolha de $T_1$

Com $T_2$ escolhido acima, todos os vetores sem K-dependência foram levantados e partimos para a escolha de $T_1$ lembrando que ele não tem nenhum efeito a vetores sem K-dependência.

Se $T_2$ mantiver todos os vetores com K-dependência, i.e., o 1º elemento de cada vetor continua positivo, então $T_1 = I$ (função identidade).

Caso contrário, seja $a_o = \min \{ a \in \mathbb{N} / K_i a + \alpha_i b_o + \beta_i C_o > 0$ para todo vetor com K-dependência $(K_i, \alpha_i, \beta_i) \}$ e

$$T_1 = \begin{pmatrix} a_o & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.1.2. Escolha de T'

Temos então $\bar{T}' = \begin{pmatrix} a_o & b_o & c_o \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ escolhido pela maneira descrita acima, resta procurar T' ótimo, usando $\bar{T}'$ como teto superior em relação ao tempo.

Antes, temos as seguintes observações:

Observações:

1) Considere os seguintes dois algoritmos cujos corpos são identicos:

```
Algoritmo A:  for K:= N_K to M_K do
                for I:= N_I to M_I do
                  for J:= N_J to M_J do
                    begin
                    corpo do algoritmo
                    end;

Algoritmo A': for K:= 0 to M_K-N_K do
                for I:= 0 to M_I-N_I do
                  for J:= 0 to M_J-N_J do
                    begin
                    corpo do algoritmo
                    end;
```

$\mathbb{CI}'$ para o algoritmo A' é $\mathbb{CI}$ transladado com vetor $v = (-N_{K_1} - N_{I_1} - N_J)$ como se observa na Fig. 5.4.

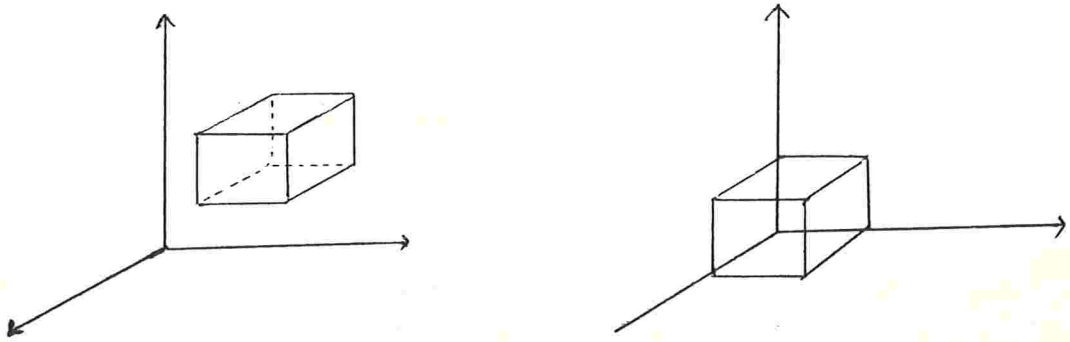


Fig. 5.4.

Seja T um transformador de A, então

$$T(CI') = T(CI+v) = \{T(p+v)/p\in\mathbb{C}I\} = \{T(p)+T(v)/p\in\mathbb{C}I\} =$$
$$= T(\mathbb{C}I) + T(v).$$

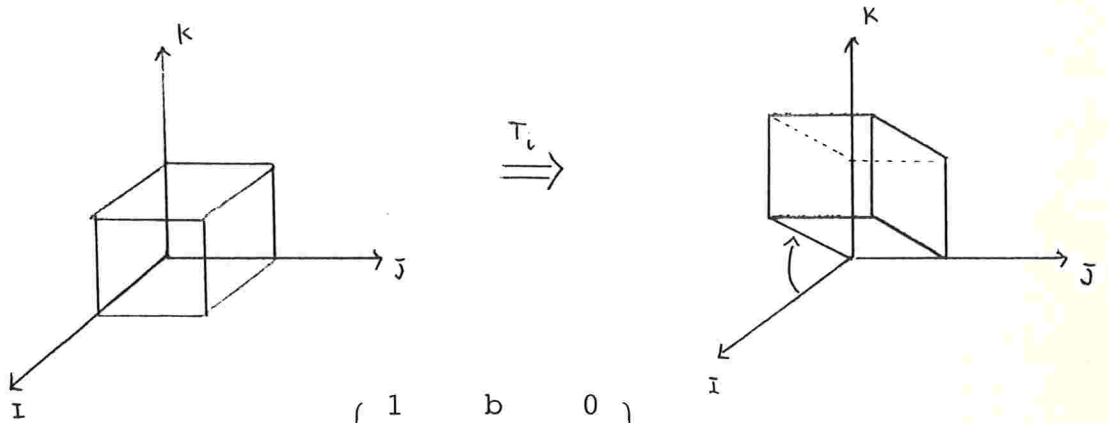
$T(\mathbb{C}I')$ é portanto $T(\mathbb{C}I)$ transladado por $T(v)$.

Nosso objetivo é analisar qual é a melhor deformação de $\mathbb{C}I$ via T, podemos considerar apenas os algortimos do tipo A' acima, sem que isto afete nosso estudo.

2) Voltemos um pouco ao $T_2$, cuja função é levantar os vetores sem K-dependência.

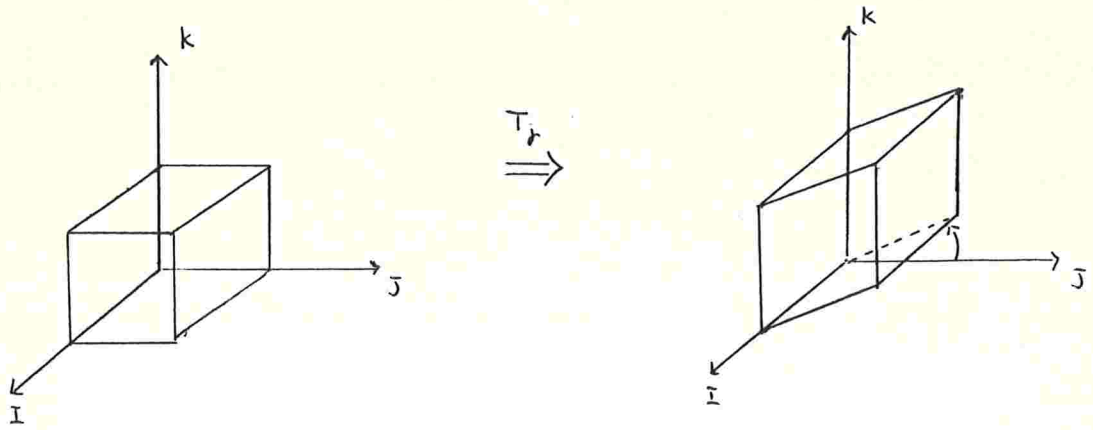
Como foi visto em 2.1.2 temos 4 tipos de vetores sem K-dependência e uma inspeção nos leva a concluir que existem 7 tipos de $T_2$ que realiza esta tarefa.

Estes 7 tipos de $T_2$ são identificados como $T_i, T_j, T_{-j}, T_{i+j}, T_{i-j}$, $T_{-i-j}$ e $T_{-i+j}$ , como mostrados na Fig. 5.5.



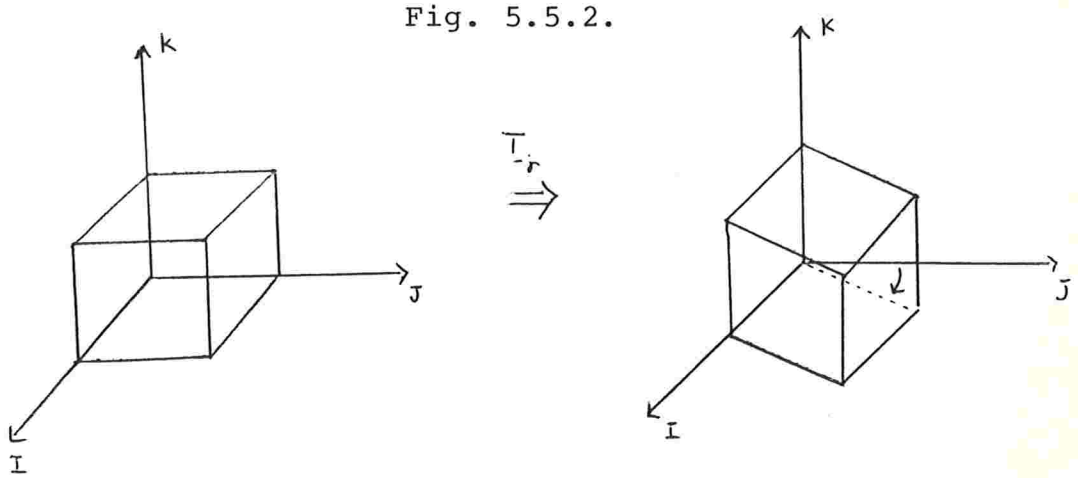

$$T_i = \begin{pmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad b\in\mathbb{N}$$

Fig. 5.5.1.



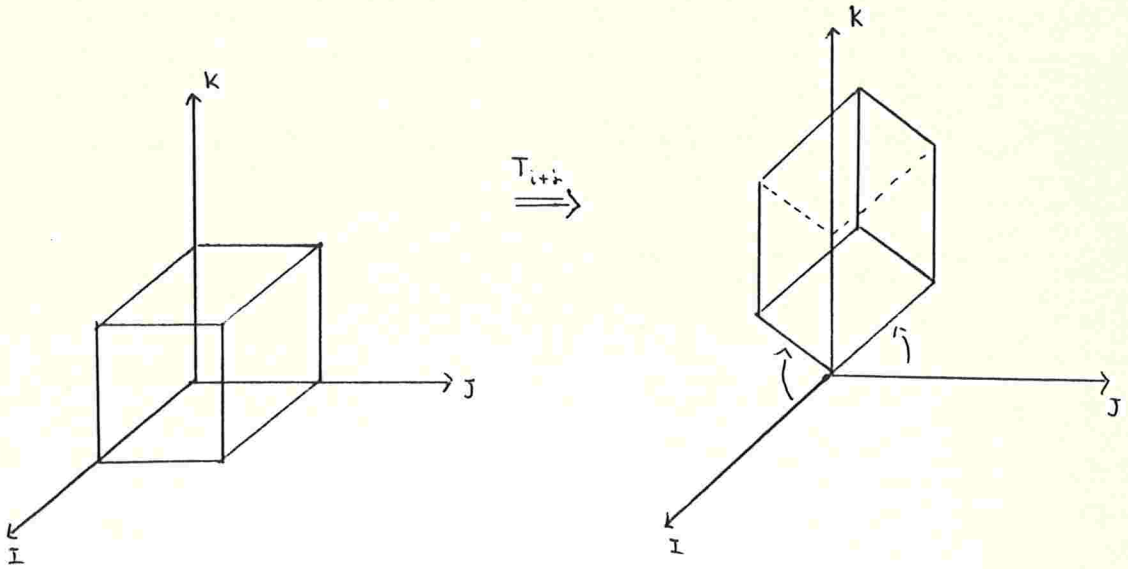

$$T_j = \begin{pmatrix} 1 & 0 & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad c \in \mathbb{N}$$

Fig. 5.5.2.



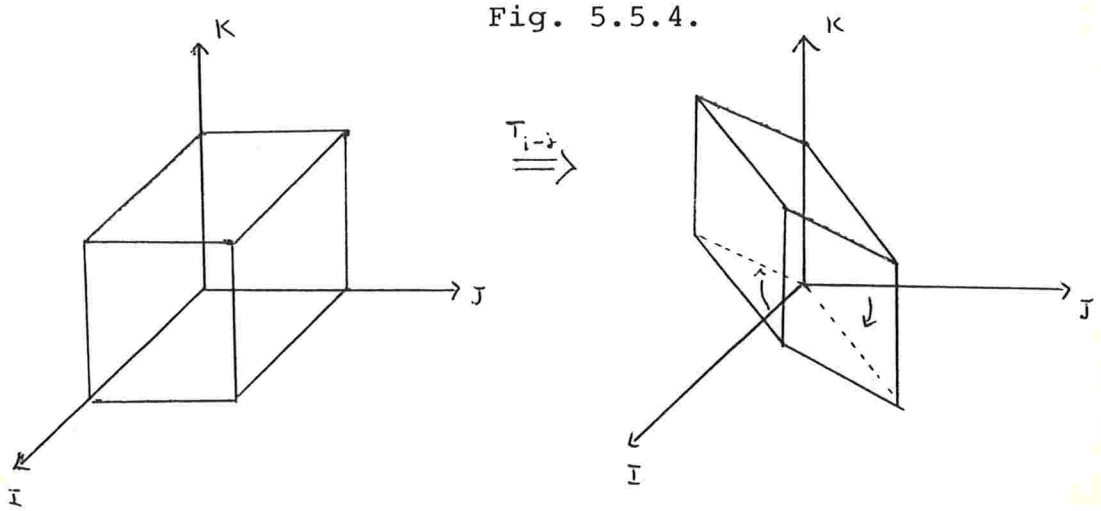

$$T_{-j} = \begin{pmatrix} 1 & 0 & -c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad c \in \mathbb{N}$$

Fig. 5.5.3.



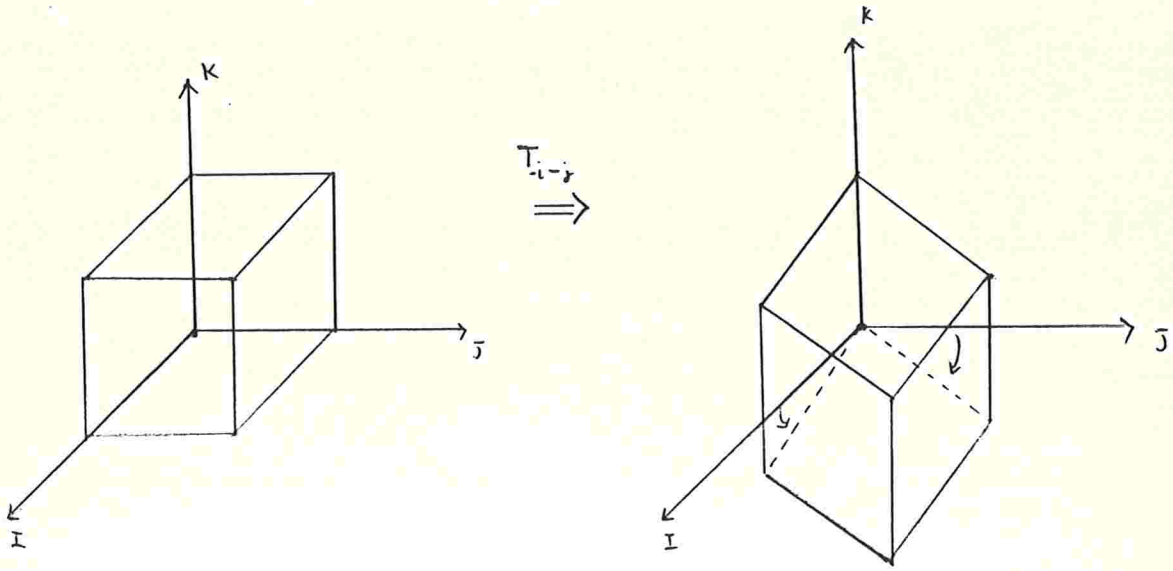

$$T_{i+j} = \begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad b,c \in \mathbb{N}$$

Fig. 5.5.4.



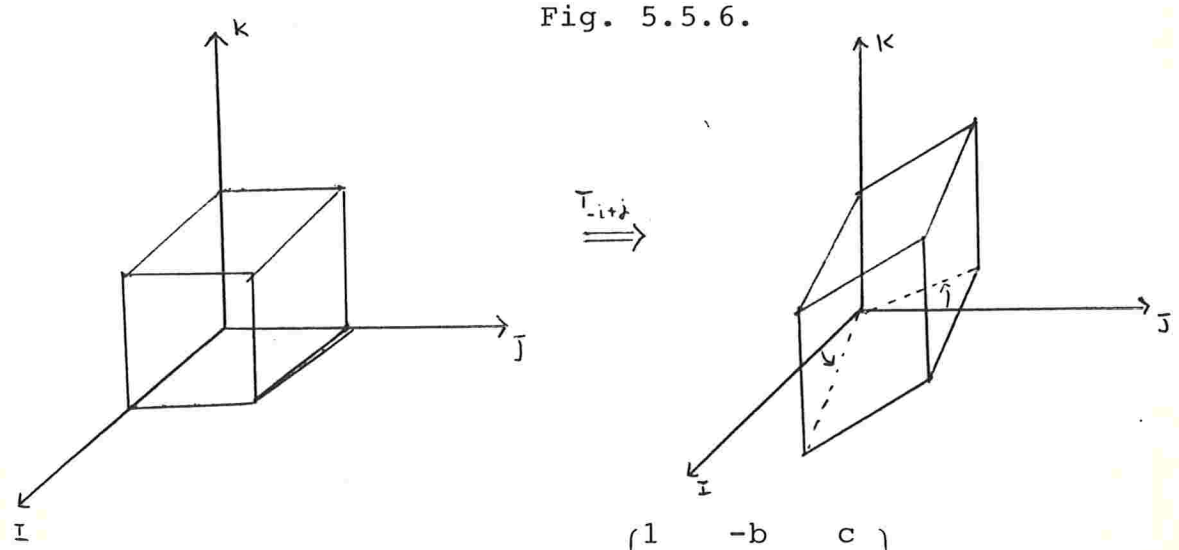

$$T_{i-j} = \begin{pmatrix} 1 & b & -c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad b,c \in \mathbb{N}$$

Fig. 5.5.5.




$$T_{-i-j} = \begin{pmatrix} 1 & -b & -c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Fig. 5.5.6.




$$T_{-i+j} = \begin{pmatrix} 1 & -b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Fig. 5.5.7.

Cada um destes 7 tipos de $T_2$ serve para certas combinações de vetores sem K-dependência e só temos estes 7 tipos.

Voltando para T', sabemos que $T' = T_2.T_1$ e o papel de $T_1$ é esticar $\mathbb{CI}$ verticalmente. Então podemos dizer que também temos 7 tipos de T', a Fig. 5.5. mostra estes 7 tipos para o caso particular de $T_1 = I$.

Por este motivo também usamos a mesma nomenclatura da Fig. 5.5. para os 7 tipos de T'.

3) O tempo de execução é dado por $t = \max\pi(P_1) - \min\pi(P_2) + 1$

$P_1, P_2 \in T(\mathbb{CI})$ e os 7 tipos de T' deformam $\mathbb{CI}$ como mostra a Fig. 5.5., exceto o fato de 'a' poder ser maior que 1. Podemos observar o seguinte:

i) para $T_i, T_j$ e $T_{i+j}$ :
   $\max\pi(P)$ é dado por $P = (N_K, N_I, N_J)$
   $\min\pi(P)$ é dado por $P = (0,0,0)$

ii) para $T_{-j}$ e $T_{-i-j}$ :
   $\max\pi(P)$ é dado por $P = (N_K, 0, 0)$
   $\min\pi(P)$ é dado por $P = (0, N_I, N_J)$

iii) para $T_{i-j}$ :
   $\max\pi(P)$ é dado por $P = (N_K, N_I, 0)$
   $\min\pi(P)$ é dado por $P = (0, 0, N_J)$

iv) para $T_{-i+j}$:
   $\max\pi(P)$ é dado por $P = (N_K, 0, N_J)$
   $\min\pi(P)$ é dado por $P = (0, N_I, 0)$

Calculemos t para cada caso:

i) $\pi_i = (a,b,0)$ $\quad t_i = \pi_i(N_K,N_I,N_J)^T - \pi_i(0,0,0)^T+1=aN_K+bN_I+0N_J+1$

$\pi_j = (a,0,c)$ $\quad t_j = \pi_j(N_K,N_I,N_J)^T - \pi_j(0,0,0)^T+1=aN_K+0N_I+cN_J+1$

$\pi_{i+j} = (a,b,c)$ $\quad t_{i+j}=\pi_{i+j}(N_K,N_I,N_J)^T-\pi_{i+j}(0,0,0)^T+1=aN_K+bN_I+cN_J+1$

ii) $\pi_{-j} = (a,0,-c)$ $\quad t_{-j} = \pi_{-j}(N_K,0,0)^T - \pi_{-j}(0,N_I,N_J)^T+1=aN_K+0N_I+cN_J+1$

$\pi_{-i-j} = (a,b,-c)$ $\quad t_{-i-j}=\pi_{-i-j}(N_K,0,0)^T-\pi_{-i-j}(0,N_I,N_J)^T+1=aN_K+bN_I+cN_J+1$

iii) $\pi_{i-j} = (a,b,-c)$ $\quad t_{i-j} = \pi_{i-j}(N_K,N_I,0)^T-\pi_{i-j}(0,0,N_J)^T+1=aN_K+bN_I+cN_J+1$

iv) $\pi_{-i+j} = (a,-b,c)$ $\quad t_{-i+j} = \pi_{-i+j}(N_K,0,N_J)^T-\pi_{-i+j}(0,N_I,0)^T+1=aN_K+bN_I+cN_J+1$

Assim podemos concluir que se $\pi = (a,b,c)$ onde $a \in \mathbb{N}$ e $b,c \in \mathbb{Z}$, então $t = aN_K+|b|N_I+|c|N_J+1$.

Logo, minimizar t é achar a tupla (a,b,c) que minimiza $aN_K+|b|N_I+|c|N_J$. Note que se $N_K = N_I = N_J$, então minimizar t é minimizar $a + |b|+|c|$.

Os passos para minimizar t são os seguintes:

1) Seja $t_o = a_oN_K+b_oN_I+c_oN_J+1$ como foi definido em 2.1.1.a

2) Faça $t_{min} := t_o$

3) Para cada tupla $(a,b,c) \in \mathbb{N} \times (\mathbb{N} \cup \{0\})^2$ tal que

$aN_K+bN_I+cN_J + 1 = t < t_o$ e $t < t_{min}$ faça:

3.1.) Faça $i := 1$ e $\pi_i := (a,b,c)$

3.2.) Se não existe vetor sem K-dependência do tipo III e $b \neq 0$ então faça $i := i+1$ e $\pi_i := (a,-b,c)$.

Se não existe vetor sem K-dependência do tipo I e $c \neq 0$ então faça i:= i+1 e $\pi_i := (a,b,-c)$.

Se não existe vetor sem K-dependência do tipo I e do tipo III, $b \neq 0$ e $c \neq 0$
então faça i:= i+1 e $\pi_i := (a,-b,-c)$.

3.3.) Se $\exists j$ tal que $1 \leq j \leq i$ e $\pi_j(d) > 0$ para todo vetor de dependência d, então faça $(a_o,b_o,c_o) := \pi_j$ e $t_{min} := t$

$$T' = \begin{pmatrix} a_o & b_o & c_o \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$, com a,b e c escolhidos acima, é ótimo em relação a execução vertical sequencial.

Observação: Os passos 3.1, 3.2 e 3.3 são executados para cada tripla (a,b,c) satisfazendo as condições mencionadas no passo 3. Supondo $N_K = N_I = N_J = N$, o número de triplas (a,b,c) possíveis satisfazendo $(a+b+c)N + 1 < t_o = (a_o+b_o+c_o)N + 1$ ou $a+b+c < a_o+b_o+c_o$ é da ordem de $(a_o+b_o+c_o)^3$. Pode-se mostrar que $a_o + b_o + c_o$ é da ordem de $\alpha^2$ com $\alpha = \max\{|d_{ij}|, d_{ij} \in D\}$.

O passo (3.3) leva o tempo de execução de $0(m)$, onde m é o número de vetores de dependência. O algorítmo terá portanto um tempo de $0(m\alpha^6)$. Notemos que $\alpha$ em geral é um inteiro positivo bem pequeno, em torno de 2.

## 2.2. Otimização para execução vertical com H deslocamentos

Vimos no final do capítulo 4, que numa execução vertical com H deslocamentos, H níveis são executados num único passo. Vimos também que cada célula básica do sistema sistólico é realmente constituída de H células básicas da execução vertical sequencial. Tal fato implica no aumento de ligações entre cada célula e suas vizinhas. O número de níveis que podem ser executados num passo está portanto relacionado e limitado pelo número de ligações possíveis. Este último,por sua vez,depende das características geométricas das dimensões da célula básica.

Seja $2N\ell$ o número máximo de ligações permitidas na borda de cada célula básica do sistema sistólico e $h\ell$ o número de ligações requeridas pela dependência de dados descritas no §2 do capítulo 4.

Seja $G = \left\lfloor \frac{N\ell}{h\ell} \right\rfloor$, então como cada célula básica requer $h\ell$ ligações para realizar a execução de um nível, o número máximo de H é G.

Assim sendo, seja $T' = \begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, então o tempo de execução vertical com H deslocamento ($H \leq G$) será dado por

$$t = \left\lceil \frac{aN_K + |b|N_I + |c|N_J + 1}{M} \right\rceil \text{ onde } M = \begin{cases} \min \pi(d_i) & \begin{cases} \text{se } \pi(d_i) \geq G \text{ para todo vetor de dependência } d_i. \end{cases} \\ G & \begin{cases} \text{caso contrário} \end{cases} \end{cases}$$

Seja $\tilde{T}' = \begin{pmatrix} a_o & b_o & c_o \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ o T' escolhido em 2.1.

1) Seja $t_o = a_o N_K + |b_o| N_I + |c_o| N_J + 1$

2) faça $t_{min} := t_o$

3) Para cada tripla $(a,b,c) \in \mathbb{N} \times (\mathbb{N} \cup \{0\})'$ tal que $aN_K + bN_I + cN_J + 1 < G.t_o$ faça

3.1.) faça $i := 1$ e $\pi_i := (a,b,c)$

3.2.) Se não existe vetor sem K-dependência do tipo III e $b \neq 0$, então faça $i := i+1$ e $\pi_i := (a, -b, c)$

Se não existe vetor sem K-dependência do tipo I e $c \neq 0$, então faça $i := i+1$ e $\pi_i := (a,b,-c)$

Se não existem vetor sem K-dependência do tipo I e do tipo III, $b \neq 0$ e $c \neq 0$, então faça $i := i+1$ e $\pi_i := (a,-b,-c)$

3.3.) Se $\exists j$ tal que $1 \leq j \leq i$ e $t_j < t_{min}$ e $\pi_j(d) > 0$ para todo vetor de dependência d, então faça

$(a_o, b_o, c_o) := \pi_j$ e $t_{min} := t_j$

## §.3 Otimização de Espaço

Escolhido T' ótimo, nos resta apenas escolhermos $T_3$.

Seja $T = \begin{pmatrix} a & b & c \\ x_{21} & x_{22} & x_{21} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$ e $S = \begin{pmatrix} x_{21} & x_{22} & x_{23} \\ & & \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$

Consideremos um cubo unitário como na Fig. 5.6.

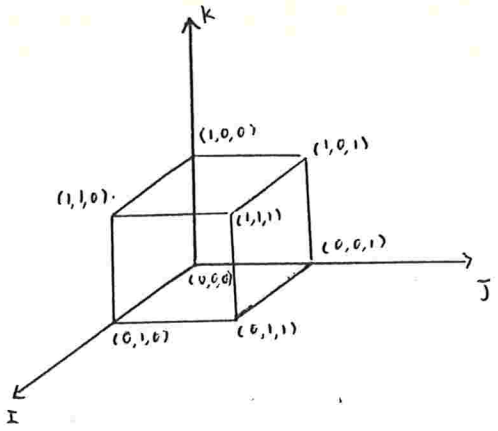



Fig. 5.6.

Considere as seguintes duas faces ($F_1$ e $F_2$) paralelas do cubo unitário. Veremos qual é o efeito de S a $F_1$ e $F_2$, observando como S levou os 4 vértices de cada face.

$$F_1 = \{(0,i,j)/0 \leq i,j \leq 1\}$$

$$F_2 = \{(1,i,j)/0 \leq i,j \leq 1\}$$

para $F_1$:

$$S\begin{pmatrix}0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\\\0\end{pmatrix} \qquad S\begin{pmatrix}0\\1\\0\end{pmatrix}=\begin{pmatrix}x_{22}\\\\x_{32}\end{pmatrix}$$

$$S\begin{pmatrix}0\\0\\1\end{pmatrix}=\begin{pmatrix}x_{23}\\\\x_{33}\end{pmatrix} \qquad S\begin{pmatrix}0\\1\\1\end{pmatrix}=\begin{pmatrix}x_{22}+x_{33}\\\\x_{32}+x_{33}\end{pmatrix}$$

para $F_2$:

$$S\begin{pmatrix}1\\0\\0\end{pmatrix}=\begin{pmatrix}x_{21}\\\\x_{31}\end{pmatrix} \qquad S\begin{pmatrix}1\\1\\0\end{pmatrix}=\begin{pmatrix}x_{21}+x_{22}\\\\x_{31}+x_{32}\end{pmatrix}$$

$$S\begin{pmatrix}1\\0\\1\end{pmatrix}=\begin{pmatrix}x_{21}+x_{23}\\\\x_{31}+x_{33}\end{pmatrix} \qquad \begin{pmatrix}1\\1\\1\end{pmatrix}=\begin{pmatrix}x_{21}+x_{22}+x_{23}\\\\x_{31}+x_{32}+x_{33}\end{pmatrix}$$

Observamos que $S(F_2) = S(F_1) + \begin{pmatrix}x_{21}\\x_{31}\end{pmatrix}$ e dado $P \in \mathbb{CI}$, $S(P) = Proj(T(P))$.

Então, se $x_{21}$ e $x_{31}$ não forem ambos nulos, $S(F_2)$ não coincide com $S(F_1)$ e, conseqüentemente, aumenta a área de projeção de $T(\mathbb{CI})$. Logo a melhor opção será $x_{21} = x_{31} = 0$.

Assim temos

$$T = \begin{pmatrix} a & b & c \\ 0 & x_{22} & x_{23} \\ 0 & x_{32} & x_{33} \end{pmatrix} \quad e \quad T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & x_{22} & x_{23} \\ 0 & x_{32} & x_{33} \end{pmatrix}$$

As células básicas de nosso sistema sistólico são dispostas num paralelograma.

Como as pastilhas VLSI são projetadas dentro de uma disposição quadrada (ou retangular), o melhor paralelograma de área constante que minimize o tamanho do quadrado (ou do retângulo) no qual ela é disposta, é o próprio quadrado (ou retângulo).

Assim sendo $T_3$ será $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ e

$$T = T_3 \cdot T_2 \cdot T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Até aqui só foi discutida a otimização de espaço sem falar em inversibilidade de $T_3$ mas sendo $T_3 = I$ esta questão foi automaticamente solucionada.

# Capítulo 6

# Conclusão

Nosso trabalho baseou-se, em grande parte, no trabalho de Fortes e Moldovan. Começaremos então este capítulo com uma comparação entre aquele trabalho e o nosso. Em seguida vamos resumir as contribuições da nossa dissertação.

## 6.1. Comparação com o método de Fortes e Maldovan

a) O método de Fortes e Moldovan é geral, ao passo que o método apresentado no Capítulo 5 aplica-se a algoritmos com UIA (uso de iteração anterior).

b) O método de Fortes e Moldovan parte de suposições iniciais para obter um transformador T. Se tal T for não inversível, as suposições iniciais são alteradas até a obtenção de um T inversível. O transformador T assim obtido pode não ser ótimo em relação ao tempo. O método apresentado produz um transformador T ótimo em relação ao tempo e sempre inversível, de maneira sistemática.

c) O método de Fortes e Moldovan começa com a escolha dos tipos de ligações permitidas. Dessa maneira, é possível restringir a obtenção de sistemas sistólicos onde apenas células vizinhas tenham ligações entre si. Pelo nosso método é possível a necessidade de ligações entre células não

vizinhas, embora próximas.

Em resumo, o nosso método consegue obter o transformador T ótimo a custa da necessidade das ligações entre células não vizinhas. Como a distância entre tais células não vizinhas depende do valor absoluto dos elementos da matriz de dependência, que em geral são pequenos, as ligações não devem constituir grande dificuldade para a sua implementação em circuitos de microeletrônica.

Em termos de complexidade de tempo, o algoritmo apresentado neste trabalho, conforme seção 2 do capítulo 5, é de $0(m\alpha^6)$, onde m é o número de vetores de dependência e $\alpha = \max\{|d_{ij}|, d_{ij} \in D\}$. ($\alpha$ é em geral um inteiro bem pequeno). Fortes e Parisi-Presicce [84] apresentaram um algoritmo para obter o tempo ótimo, cuja complexidade é de $0(m^3.\binom{3+m}{3})$. Em ambos os casos, a complexidade refere-se a transformações de algoritmos sequenciais expressos em três malhas encaixadas.

## 6.2. Contribuições

A seguir vamos resumir as contribuições deste trabalho de dissertação.

a) No capítulo 2, o algoritmo sistólico de decomposição LU, que reduz a complexidade de tempo de 4n (método de Kung e Leiserson) para 3n, para matriz n × n, é inédito.

b) No capítulo 3, re-organizamos os resultados sobre a detecção de paralelismo de Fortes e Moldovan [85], tornando-os mais claros e dando a ênfase no aspecto geométrico, por meio de abundantes exemplos.

c) No capítulo 4, utilizamos o tratamento geométrico e resultados do capítulo 3 para esclarecer a transformação de algoritmos e sua posterior adaptação ao sistema sistólico. A interpretação geométrica da transformação contribuiu para a melhor compreensão do método envolvido e a execução vertical com H deslocamentos foi descrita com maior clareza do que em outros trabalhos.

d) Ainda no capítulo 4, o teorema 3 que trata da existência do transformador T é inédito, existência essa não tratada de maneira clara em trabalhos anteriores.

e) No capítulo 5, todos os resultados incluindo a nomenclatura e a classificação dos transformadores, são originais. Embora restrito a algoritmos com UIA, o método inédito para obtenção do transformador ótimo é relevante pela constatação de que muitos algoritmos iterativos definidos por recorrências são desse tipo.

## 6.3. Possíveis direções para trabalhos futuros

1) Geração automática da descrição de todo esquema do sistema sistólico após a escolha do transformador T, o que é feito de maneira manual.

2) Projeto de um sistema sistólico reconfigurável e universal que pode realizar as computações de qualquer sistema sistólico resultante da aplicação do método apresentado.

3) Utilização e extensão dos resultados obtidos para a geração automática de código para maquinas paralelas específicas.

oOo

# BIBLIOGRAFIA


1. BENTLEY & KUNG [79]

J.L. Bently & H.T. Kung

*A tree Machine for Searching Problems*

Proceedings of 1979 International Conference on Parallel Processing IEEE, August, 1979, 257-266.

2. COSNARD & ROBERT [86]

M. Cosnard & Y. Robert

*Parallel Algorithms & Architectures*

Proceedings of the International Workshop on Parallel Algorithms & Architectures - Centre National de Rencontres Mathématiques, Luminy, France, 14-18, April, 1986

3. DELOSME & IPSEN [86]

J.M. Delosme & I.C.F. Ipsen

*Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations*

Linear Algebra & Its Applications 77: 75-111 (1986)

4. DELOSME & IPSEN [85]

J.M. Delosme & I.C.F. Ipsen

*Efficient Systolic Array for the Solution of Toaplitz Systems: An Illustration of a Methodology for the Construction of Systolic Architectures in VLSI.*

Technical Report YALEU/DCS/RR-370, Yale University, June, 1985.

5. FORTES & MOLDOVAN [85]

J.A.B. Fortes & D.I. Maldovan

*Parallelism Detection and Transformation Technique Useful for VLSI Algorithms*

Journal of Parallel and Distributed Computing 2, 277-301 (1985).

6. FORTES & PARISI-PRESICCE [89]

J.A.B. Fortes & F. Parisi-Presicce

*Optimal Linear-Schedules for the Parallel Exection of Algorithms*

Proc. 13th Int. Conf. on Parallel Processing, 1984.

7. FOSTER & KUNG [80]

M.J. Foster & H.T. Kung

*The Design of Special-Purpose Chips*

Computer, Vol. 13 nº 1, 26-40, Jan, 1980

7. GUIBAS, KUNG & THOMPSON [79]

L.J. Guibas, H.T. Kung & C.D. Thompson

*Direct VLSI Implementation of Combinatorial Algorithms*

Caltech Conference on VLSI, Jan, 1979

8. KARP, MILLER & WINOGRAD [67]

R.M. Karp, R.E. Miller & S. Winograd

*The Organization of Computation for Uniform Recurrence Equations*

Journal of the Association for Computing Machinery, Vol. 14, nº 3, 1967.

9. KUNG [81]

H.T. Kung

*Why Systolic Architectures?*

Computer 15, 1 (Jan, 1982)

10. KUNG & LEISERSON [79]

H.T. Kung & C.E. Leiserson

*Systolic Array for VLSI em Duff, I.S. e Stuwart, G.W. (ed.)*

Sparce Matrix Proceedings 1978, SIAM, 1979

11. KUNG, RUANE & YEN [81]

H.T. Kung, L.M. Ruane & D.W.L. Yen

*A two-Level Pipelined Systolic Array for Convolutions*

CMU Conference on VLSI Systems & Computations, Kung, Sproull & Steele (ed.)

Computer Science Press (1981) 255-264.

12. KUNG & SONG [82]

H.T. Kung & S.W. Song

*A Systolic 2-D Convolution Chip*

Multicomputers and Image Processing Algorithms and Programs, L. Uhr (ed.), Academic Press, 1982.

13. LI & WAH [85]

G.J. Li & B.W. Wah

*"The Design of Optimal Systolic Arrays"*

IEEE transactions on Computers, Vo. C-34, pp66-77, Jan,1985

14. MEAD & CONWAY [80]

C. Mead & L. Conway

*Introduction to VLSL Systems*

Addison Wesley Series in Computer Science, 1980

15. MIRANKER & WINKLER [84]

W.L. Miranker & A. Winkley

*Spacetime Representations of Computational Structure*

Computing 32, 93-114 (1984).

16. MOLDOVAN [82]

D.I. Moldovan

*On the Analysis and Synthesis of VLSL Algorithms*

IEEE Transactions on Computer, Vol. C-31, nº 11

November, 1982.

17. MOLDOVAN [83]

D.I. Moldovan

*On the Design of Algorithms for VLSI Systolic Arrays*

Proceeding of the IEEE Vol. 71, nº 1, Jan, 1983

18. MOLDOVAN [87]

D.I. Moldovan

*ADVIS: A software Package for the Design of Systolic Arrays*

IEEE Transactions on Computer - Aided Design Vol. CAD-6

nº 1, January, 1987.

19. MOLDOVAN & FORTES [86]

D.I. Moldovan & J.A.B. Fortes

*Partitioning and Mapping Algorithms into fixed Size Systolic Arrays*

IEEE Transaction on Computers Vol. C-35 nº 1, Jan, 1986.

20. MORDELL [69]

L.J. Mordell

*Diophantine Equations*

Academic Press, New York, 1969.

21. NERING [63]

E.D. Nering

*Linear Algebra and Matrix Theory*

John Wiley & Sons, Inc, 1963, 51-56.

22. OKUDA & SONG [86]

K. Okuda & S.W. Song

*Um Algoritmo de Multiplicação de Matrizes para Implementação em VLSI*

Anais do I congresso da Sociedade Brasileira de Microeletrônica, Campinas, Julho, 1986.

23. QUINTON [87]

P. Quinton

*Automata Networks in Computer Science Theory & Applications*

ed. F.F. Souliè et. sl., Manchester Univeristy Press, 1987.

24. ROBERT [87]

Y. Robert

*Systolic Algorithms and Architectures*

ed. F.F. Souliè et. sl.,

Manchester University Press, 1987.

25. ROBERT & TCHUENTE [86]

Y. Robert & M. Tchuente

*An Efficient Systolic Array for the ID REcursive Convolution Problem*

Journal of VLSI & Computer Systems, Vol. 1, nº 4, 1986.

26. ROCKAFELLAR [70]

R. T. Rockafellar

*Convex Analysis*

Princeton Univerity Press, Princeton, NJ, 1970

27. SONG [81]

S.W. Song

*On A High Performance VLSI Solution to Database Problems*

Ph. D. Thesis, Carnegie-Mellon University,

Computer Science Department, August, 1981.

28. SONG [84]

S. W. Song

*Algoritmos Paralelos e Arquitetura VLSI*

IV Escola de Computação, 84, São Paulo.

29. STERN [86]

Júlio M. Stern

*Fatoração L-U e Aplicações*

Relatório Técnico RT-MAP-8606

MAP-IME-USP, 1986

oOo