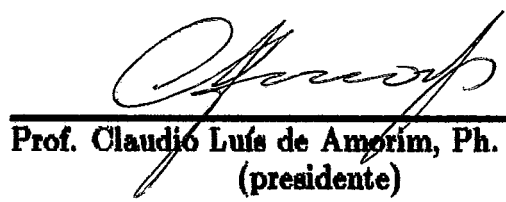
UMA BIBLIOTECA DE OPERAÇÕES VETORIAIS E MATRICIAIS PARALELAS PARA MULTIPROCESSADORES HIPERCÚBICOS BASEADOS EM TRANSPUTERS

Maria Clicia Stelling de Castro

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

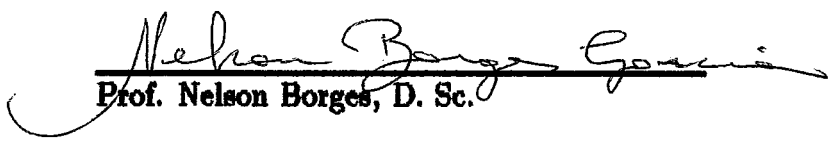
Aprovada por:



Prof. Claudio Luís de Amorim, Ph. D.
(presidente)

Prof. Valmir Carneiro Barbosa, Ph. D.

Prof. Eugenius Kaszkurewicz, D. Sc.



Prof. Nelson Borges, D. Sc.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 1991

CASTRO, MARIA CLICIA STELLING DE

Uma Biblioteca de Operações Vetoriais e Matriciais Paralelas para Multiprocessadores Hipercúbicos Baseados em *Transputers* [Rio de Janeiro] 1991

VI, 109 p., 29.7 cm, (COPPE/UFRJ, M. Sc., ENGENHARIA DE SISTEMAS E COMPUTAÇÃO, 1991)

TESE – Universidade Federal do Rio de Janeiro, COPPE

1 – Arquitetura de Computadores 2 – Processamento Paralelo

I. COPPE/UFRJ II. Título(Série).

Aos meus pais Maria José e Alcides

Agradecimentos

Agradeço aos meus familiares e amigos por todo apoio e carinho sempre presentes.

Em especial, agradeço ao professor Claudio Luís de Amorim pela sua orientação, apoio e estímulo.

Aos meus pais Maria José e Alcides,e meus irmãos Maria Clara e Dario por todo carinho e compreensão.

A professora Arzelina Mendes que com seu exemplo, seu carinho e sua dedicação me ensinou o valor do conhecimento.

Ao amigo Ildemar, pelo seu apoio e estímulo durante a realização desse trabalho.

Agradeço também aos funcionários técnicos e administrativos do Programa de Engenharia de Sistemas e Computação pela atenção e prestatividade.

Agradeço à Capes, ao CNPq, à Finep e à Sociedade Cutural e Beneficente Guilherme Guinle pelo apoio para a realização deste trabalho.

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

Uma Biblioteca de Operações Vetoriais e Matriciais Paralelas para Multiprocessadores Hipercúbicos Baseados em *Transputers*

Maria Clicia Stelling de Castro

Maio de 1991

Orientador: Claudio Luís de Amorim
Programa: Engenharia de Sistemas e Computação

A solução de muitos problemas científicos e de engenharia requer uma significativa quantidade de computação de sistemas de equações. Os sistemas de computação paralelos, e em particular os de topologia hipercúbica, oferecem a perspectiva de um ganho de potência e de velocidade de processamento. Este é um ponto ainda em estudo e pesquisa.

Neste trabalho procuramos implementar uma biblioteca de operações paralelas para multiprocessadores hipercúbicos baseados em *transputers*, e que está relacionado ao modelo de Oliver Mcbryan e Erick Van de Velde.

O sucesso de projetos hipercúbicos está relacionado à possibilidade de implementação eficiente de algoritmos paralelos para uma extensa faixa de aplicações numéricas. A arquitetura hipercúbica tem características que permitem eficiente realização de outras topologias de interconexão apropriadas (grade, árvore e anel) dependendo da aplicação.

Nosso trabalho está concentrado no desenvolvimento de algoritmos

que são passos elementares de algoritmos maiores que solucionem, por exemplo, sistemas de equações elípticas, parabólicas e hiperbólicas encontrados em problemas das Engenharias, Física, Ciência da Computação entre outras.

As rotinas que formam a biblioteca de operações paralelas tratam do armazenamento de vetores e matrizes distribuídos de várias formas através dos processadores, da conversão entre esses tipos de armazenamento, de operações básicas da álgebra linear tais como produto interno, produtos de uma matriz por um vetor e de matrizes, e de operações de difusão e convergêngia de escalares, vetores e matrizes. A biblioteca opera tanto com matrizes densas quanto esparsas (matrizes de banda). As rotinas foram implementadas na linguagem *C Paralela* do *transputer.*

A aplicação de uma operação é realizada com os processadores chamando o procedimento individualmente com seus argumentos apropriados. O código executado em cada processador é o mesmo. Para a solução de um problema real, basta agrupar os procedimentos convenientes de modo a obter o algoritmo desejado, tendo a facilidade da modularidade para a depuração. A biblioteca de operações paralelas torna invisível a arquitetura da máquina e a comunicação entre os processadores realizada pelos procedimentos.

A finalidade é obter metodologias de programação portáveis de forma a facilitar o desenvolvimento de aplicações numéricas em hipercubos baseados em *transputers.*

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

A Library of Vector and Matrix Parallel Operations for Hypercube Multiprocessors Based on Transputers

Maria Clicia Stelling de Castro

May, 1991

Thesis Supervisor: Claudio Luís de Amorim
Department: Programa de Engenharia de Sistemas e Computação

The solution of many scientific and engineering problems requires a significant amount of computation of system of equations. Parallel computer systems, in particular hypercubes, open a perspective on high speedup and an increase of computing power. This point is still a subject of intensive research.

In this work we implement a library of parallel operations which has been developed for hypercube multiprocessors based on transputers. It is related to the Oliver Mcbryan and Erick Van de Velde's model.

The success of hypercube designs is related to the possibility of implementing parallel algorithms efficiently to a wide range of numerical applications. The hypercube architecture has caracteristics that make possible to embbed other topologies efficiently such as binary trees, hierarchies of rings and retangular grids, according to the applications.

Our work foccuses the development of algorithms that are elementary steps of larger algorithms, to solve elliptc, parabolic and hyperbolic equations which

occur in of Engineering, Phisical and Computing Science problems among others.

The routines in the library of parallel operations deal with distributed vector and matrix allocation, basic linear algebra operations such as inner products, matrix transpose and matrix-vector products and scalar, vector and matrix broadcasting. This library is efficient to both dense and sparse matrix (band matrix). The routines was implemented in *C Parallel* linguage of the transputer.

The use of a parallel operation is accomplished by having all of the processors calling the individual routines, with the appropriate arguments. The same code is executed in all the processors. To solve realistic problems, the routines must be grouped in order to obtain larger algorithms. This approach offers modularity that facilitates debugging. The library makes transparent the dependency on machine architecture and on the communication between processors.

Our goal is to obtain portable programming methodologies in order to simplify the development of numerical applications on Transputer based hypercubes.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

A idéia de incluir paralelismo nos sistemas de computadores é antiga. Tanto a execução simultânea de diversas atividades pelo *hardware*, quanto a realização antecipada de algumas dessas atividades, são idéias empregadas em arquiteturas da atualidade, e incluem conceitos tais como busca antecipada de instruções (*lookahead*), superposição das fases de execução das instruções (*overlap*), múltiplas unidades funcionais, estágios de execução (*pipeline*), processamento vetorial e multiprocessadores.

O que estimula a computação paralela e distribuída é o desempenho, já que se diversos agentes estão operando é de se esperar que o tempo de processamento diminua; a tolerância a falhas, onde o componente defeituoso pode ser retirado do sistema sem que haja interrupção na operação; e a modularidade, que permite a expansão através da adição de mais processadores. Outras razões para a computação paralela foram o desenvolvimento das tecnologias empregadas no *hardware* (circuitos VLSI), o desenvolvimento de linguagens para programação paralela e pesquisas em algoritmos paralelos.

Existem diversas possibilidades de organizar uma coleção de processadores e produzir uma grande variedade de computadores paralelos com o objetivo de atingir melhor desempenho. Uma de particular sucesso é organização hipercúbica [1].

O sucesso de projetos hipercúbicos está relacionado à possibilidade de implementação eficiente de algoritmos paralelos para uma extensa faixa de aplicações

numéricas.

O trabalho desta tese está relacionado ao modelo de Oliver Mcbryan e Eric Van de Velde, que propuseram uma biblioteca de operações paralelas para multiprocessadores hipercúbicos. Mcbryan e Van de Velde utilizaram a biblioteca desenvolvida para implementar algoritmos paralelos para solucionar problemas relativos à dinâmica dos fluidos computacionais e realizaram algumas análises de desempenho empregando simuladores de máquinas hipercúbicas o Caltech Mark II da Caltech e o iPSC da Intel.

Inicialmente, empregamos um simulador de uma arquitetura hipercúbica, o simulador do iPSC2 da Intel, devido à lenta importação de uma placa *quadputer*. Este simulador utilizava um computador compatível com o sistema IBM - PC, e era executado sob o sistema operacional Xenix. Com a chegada da placa, este trabalho foi descontinuado, utilizamos um sistema *Quadputer* que é composto por quatro placas de *transputers* interconectadas como um cubo de dimensão dois.

Posteriormente utilizamos um multiprocessador de memória distribuída com topologia hipercúbica baseado em *transputers* que foi desenvolvido pela COPPE, o NCP I. Esse multiprocessador possui atualmente oito processadores.

Nosso trabalho está concentrado no desenvolvimento de algoritmos que são passos elementares de algoritmos maiores que solucionem, por exemplo, sistemas de equações elípticas, parabólicas e hiperbólicas encontrados em problemas das Engenharias, Física, Ciência da Computação entre outras.

Os algoritmos paralelos que formam a biblioteca de operações paralelas tratam do armazenamento de vetores e matrizes distribuídos de várias formas através dos processadores, da conversão entre esses tipos de armazenamento, de operações básicas da álgebra linear tais como produto interno, produtos de uma matriz por um vetor e de matrizes entre outras, e de operações de difusão e convergêngia de escalares, vetores e matrizes. A biblioteca opera tanto com matrizes densas quanto esparsas, em particular matrizes de banda.

A aplicação de uma operação é realizada com os processadores cha-

mando o procedimento individualmente com seus argumentos apropriados. O código executado em cada processador é o mesmo. Para a solução de um problema real, basta agrupar os procedimentos convenientes de modo a obter um algoritmo maior, tendo a facilidade da modularidade para a depuração. A biblioteca de operações paralelas torna invisível a dependência da arquitetura da máquina e a comunicação entre os processadores realizada pelos procedimentos. Toda metodologia utilizada na comunicação desenvolvida para a rede de transputers é interna as rotinas.

A finalidade é obter metodologias de programação portáveis de forma a facilitar o desenvolvimento de aplicações numéricas em hipercubos baseados em *transputers*. Pretendemos diminuir a dificuldade existente na programação paralela facilitando o uso na área de aplicação numéricas, reduzindo o esforço redundante de se reescrever algoritmos em geral. E ainda, proporcionar ferramentas de desenvolvimento para implementadores de algoritmos numéricos. O desenvolvimento de bibliotecas de álgebra linear portáveis pode ser relacionado a desenvolvimento tais como as rotinas BLAS [7] - [8].

O capítulo II descreve a arquitetura do multiprocessador hipercúbico baseado em *transputers* e a seqüência "Binary Reflected Gray Code" utilizada no implemento das topologias de grade, árvore e anel que são fundamentais aos algoritmos da biblioteca. A metodologia empregada para a troca de mensagens no multiprocessador hipercúbico baseado em *transputers* é relatada no capítulo III. O capítulo IV descreve as estruturas de dados que representam os vetores e as matrizes, as formas em que eles podem ser armazenados e os passos elementares que compõem a biblioteca de operações paralelas. A aplicação das operações da biblioteca para solucionar um sistema de equações lineares utilizando o método do gradiente conjugado, sua implementação e sua avaliação são mostrados no capítulo V. Os resultados finais e as possibilidades futuras são analisados no capítulo VI.

Capítulo II

Mapeamentos de Grades, Árvores e Anéis em Hipercubos

A biblioteca de operações paralelas foi desenvolvida para um multiprocessador de memória distribuída com topologia hipercúbica baseado em *transputers* (Rede de *transputers*). Esse capítulo descreve a arquitetura do multiprocessador hipercúbico e a seqüência "Binary Reflected Gray Code" (BRGC) utilizada no implemento das topologias de grade, árvore binária e anel nessa arquitetura, e que serão fundamentais para os algoritmos apresentados no capítulo IV.

II.1 Multiprocessador Hipercúbico Baseado em *Transputers*

Um multiprocessador com topologia hipercúbica binária [1] consiste de 2^D processadores independentes, que podem ser vistos espacialmente como se estivessem localizados nos vértices de um cubo de dimensão D cujas D arestas de ligação em cada vértice correspondem aos canais de comunicação com os vértices vizinhos (figura II.1).

Os processadores são identificados por números na faixa $[0, 2^D - 1]$, tal que a representação binária de D-dígitos de processadores adjacentes fisicamente diferem em somente um bit, como ilustrado na figura II.1. Os processadores não compartilham memória e se comunicam através da troca de mensagens.

O multiprocessador hipercúbico baseado em *transputers* utilizado para a validação e a avaliação de desempenho da biblioteca de operações paralelas foi o NCP I desenvolvido pela COPPE, que possui atualmente oito *transputers*, sendo então um hipercubo de dimensão três.

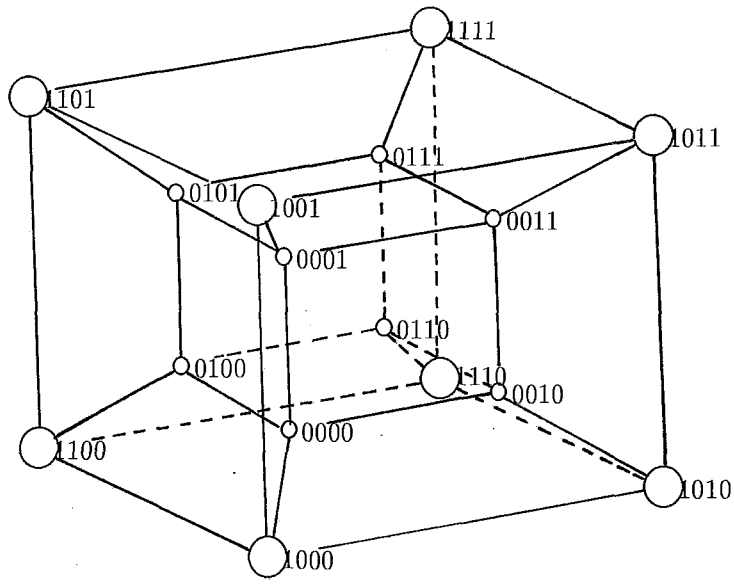


Figura II.1: Representação de um hipercubo de dimensão quatro

Cada processador individual é um *transputer* T800 [2] de 32 bits com 4K bytes de memória estática interna e 4G bytes de espaço total de endereçamento. Possui quatro ligações seriais bidirecionais com capacidade de 20M bits/s, para comunicação direta com outros quatro *transputers*, sendo a comunicação realizada de forma síncrona. Suporta também operações de ponto flutuante. Foi projetado inicialmente para dar suporte eficiente à liguagem *OCCAM*, sendo que já existem também disponíveis versões paralelas das linguagens *C*, *Pascal* e *Fortran*.

O multiprocessador hipercúbico NCP I utiliza um computador hospedeiro ligado ao processador zero, que atua como uma *interface* entre o usuário e o cubo, assim, todas as informações passam do computador hospedeiro ao processador zero através de um canal bidirecional de comunicação extra. O computador hospedeiro é um computador compatível com o sistema IBM PC.

II.2 A Seqüência "Binary Reflected Gray Code"

O código "Binary Reflected Gray Code" (BRGC) [4] tem uma importância fundamental no desenvolvimento da biblioteca de operações paralelas, pois é baseado nesta seqüência que são obtidas as topologias empregadas e distribuídos os dados através da rede hipercúbica.

Dependendo da aplicação, a interconexão entre os processadores deve ser organizada de forma a obter uma topologia apropriada. As topologias essenciais (grade, árvore e anel) aos algoritmos tratados posteriormente, são baseadas na seqüência "Binary Reflected Gray Code" (BRGC) para identificar os processadores [3]. A seqüência BRGC é definida recursivamente da seguinte forma:

- Com um bit a seqüência é 0 1;
- Com d-dígitos, o d-ésimo dígito é ativado e os $d-1$-dígitos restantes da seqüência são repetidos na ordem inversa. Assim com d-dígitos temos as seqüências para um, dois e três bits na tabela II.1:

1 bit	2 bits	3 bits	
0	00	000	
1	01	001	
	–	011	
	11	010	
	10	—	...
		110	
		111	
		101	
		100	

Tabela II.1: Geração da seqüência BRGC para um, dois e três bits

Essa seqüência apresenta propriedades convenientes a implementação das topologias de grade, árvore binária e anel:

- a seqüência é periódica;

- os elementos diferem em apenas um bit na seqüência binária;
- os elementos vizinhos na seqüência binária são vizinhos físicos no hipercubo e
- dois elementos que são uma potência de dois diferem em no máximo dois bits.

A periodicidade é útil no mapeamento do hipercubo em anel, pois os elementos dos extremos da seqüência também diferem em apenas um bit.

Se dois elementos diferem em apenas um bit, eles são vizinhos na seqüência e portanto no hipercubo (figura II.1), tendo então uma ligação física entre eles. Esse fato, auxilia no roteamento das mensagens (tempo gasto na comunicação) já que os mapeamentos de grade, árvore e anel são baseados nesta seqüência. Do mesmo modo, se dois elementos são uma potência de dois, eles diferem em no máximo dois bits e têem duas ligações físicas entre eles.

A seqüência BRGC para quatro bits onde podem ser verificadas essas propriedades é a seguinte.

0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

A periodicidade pode ser notada entre os elementos 0000 e 1000 que diferindo apenas no bit 3, demonstram também a segunda propriedade. Esses mesmos elementos são ainda vizinhos na seqüência binária e no hipercubo (figura II.1) e como são uma potência de dois diferem apenas em um bit. A última propriedade fica mais clara com os elementos 0001 e 1000 que são uma potência de dois e diferem nos bits 0 e 3.

A tabela II.2 apresenta os procedimentos relativos a seqüência BRGC, onde pode-se obter a identificação do processador a partir de sua posição na seqüência ou o inverso, a posição dada a identificação.

Os algoritmos da seqüência BRGC são mostrados a seguir:

identificação = gray (posição)
p = identificação do processador segundo a seqüência BRGC;
$p = (posição \gg 1)$ exor *posição*;
Retorna p.

gray ()	retorna a identificação do processador dada a posição na seqüência
ginv ()	operação inversa, retorna a posicão na seqüência dada a identificação do processador

Tabela II.2: Rotinas da Seqüência "Binary Reflected Gray Code"

posição = ginv (identificação)

p = identificação do processador;
estado = 0;
D = dimensão do hipercubo;
Se $D <> 0$
  máscara = potência (2, (D - 1));
Para todo $i = 0$ a $i < D$
  Se *estado* $\neq 0$
    $p = p$ exor *máscara*;
  estado = p and *máscara*;
Retorna p.

Supondo um hipercubo de dimensão três, temos demonstrado na tabela II.3 a posição na seqüência BRGC e a identificação do elemento correspondente à posição.

Posição	Identificação
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Tabela II.3: Posição e identificação dos elementos segundo a seqüência BRGC

Então se desejarmos saber a posição na seqüência BRGC do processador cuja identificação é 100 obteremos como resposta a posição 7, utilizando o procedimento *ginv()*. E para se saber qual o identificação do processador que ocupa a posição 3 na seqüência utilizamos o procedimento *gray()* e obtemos como resposta 010.

II.3 Mapeamentos

Para tornar eficiente a aplicação da biblioteca de operações paralelas, são empregados os mapeamentos em grade, árvore e anel. Dependendo da aplicação o hipercubo é visto como uma dessas topologias.

A tabela II.4 é relativa aos procedimentos de mapeamento de uma grade, uma árvore binária e um anel.

grade ()	constrói uma grade bidimensional
árvore_binária ()	constrói uma árvore
anel ()	constrói anéis de níveis 0 a $D-1$, onde D é a dimensão do hipercubo

Tabela II.4: Rotinas de Mapeamento do Hipercubo

II.3.1 Mapeamento em grade

Algumas aplicações requerem que grades periódicas sejam mapeadas em cubos. A seqüência BRGC é conveniente para esse propósito. As grades em hipercubos são construídas a partir de grades unidimensionais, com conexão de vizinhos mais próximos. Para se obter grades de dimensões maiores, é necessário que cada dimensão seja mapeada adeqüadamente e use mapeamentos unidimensionais àqueles subcubos, como pode ser visto na figura II.2.

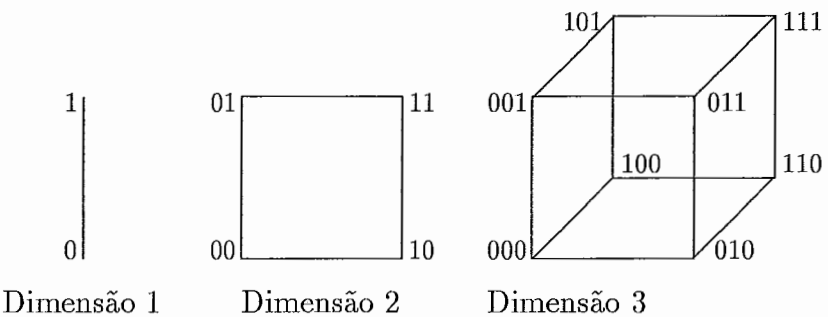


Figura II.2: Construção de uma grade de dimensão três

A estrutura de representação de uma grade bidimensional e o algoritmo de construção podem ser vistos a seguir.

estrutura em_grade {

processador ao norte

processador ao sul

processador ao leste

processador ao oeste

}

$$pt\_grade = grade\ ()$$

pt_grade = armazena área para a estrutura em grade;
x = posição do processador na direção x;
x_leste = identificação do processador $x + 1$;
x_oeste = identificação do processador $x - 1$;
y = posição do processador na direção y;
y_norte = identificação do processador $y - 1$;
y_sul = identificação do processador $y + 1$;
Retorna pt_grade.

Assim supondo um hipercubo de dimensão quatro, onde gostaríamos de ter uma grade (figura II.3) com quatro processadores na direção x e quatro na direção y, periódica em ambas as direções segundo a seqüência BRGC, teríamos para o processador $P_{(x,y)}$ cuja identificação é 0101 uma estrutura de dados preenchida da seguinte forma:

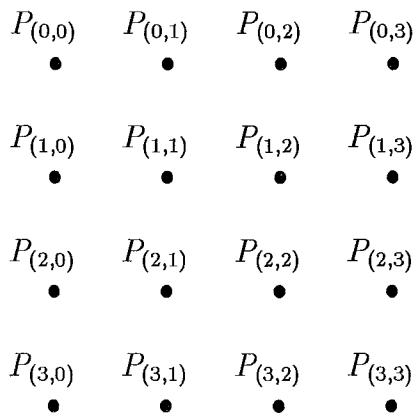


Figura II.3: Grade com quatro processadores na direção x e quatro na direção y

- processador ao norte = 0001;
- processador ao sul = 1101;
- processador ao leste = 0111;

- processador ao oeste = 0100.

A figura II.4 mostra uma grade bidimensional de um hipercubo de dimensão quatro com quatro processadores na direção x e quatro processadores na direção y.

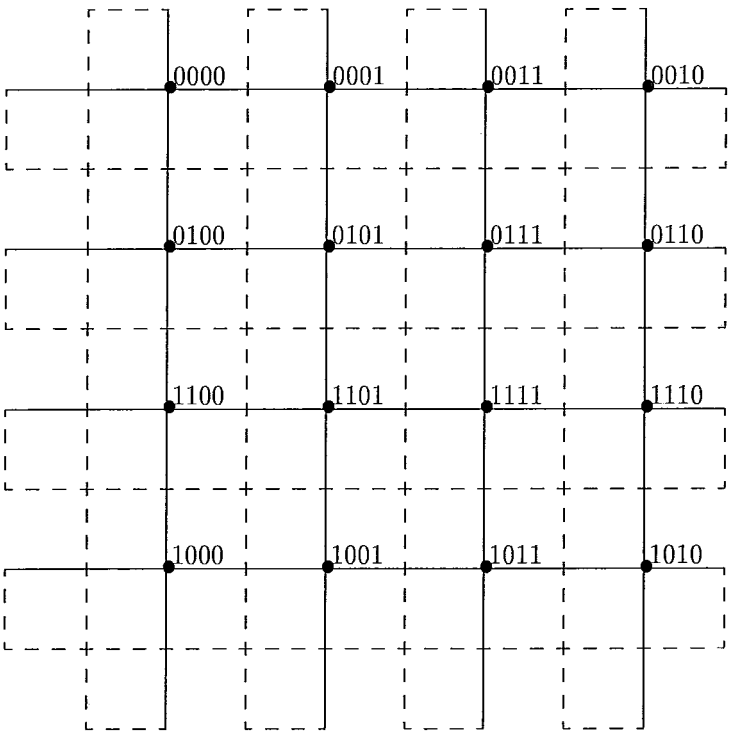


Figura II.4: Grade bidimensional de um hipercubo de dimensão quatro

II.3.2 Mapeamento em árvore

Entre as diversas maneiras de se construir uma árvore em um hipercubo, as mais comumente utilizadas são árvore binária ou árvore D-ária, onde D é a dimensão do cubo.

Uma árvore binária balanceada pode ser mapeada num hipercubo considerando o fato que um hipercubo de dimensão D pode ser construído com dois outros cubos de dimensão $D-1$ com cada processador correspondente ligado por um canal extra como foi ilustrado na figura II.1. Numerando então os dois cubos de dimensão $D-1$ como $p_0, \ldots, p_0 + 2^{D-1} - 1$, e $p_0 + 2^{D-1}, \ldots, p_0 + 2^D - 1$ respectivamente, e conectando o processador p_0 do primeiro subcubo com o processador

$p_0 + 2^{D-1}$ do segundo subcubo, a árvore binária é definida recursivamente como tendo o processador com numeração mais baixa como o raiz, e os cubos de dimensão $D-1$ como suas subárvores esquerda e direita. A tabela II.5 mostra a construção das subárvores direita e esquerda.

Dimensão	Subárvore		Raiz	
	Direita $p_0, \ldots, p_0 + 2^{D-1} - 1$	Esquerda $p_0 + 2^{D-1}, \ldots, p_0 + 2^D - 1$	Direita	Esquerda
1	0	1	0	1
2	0,1	2,3	0	2
3	$0, \ldots, 3$	$4, \ldots, 7$	0	4
4	$0, \ldots, 7$	$8, \ldots, 15$	0	8

Tabela II.5: Subárvores direita e esquerda para formação de uma arvóre binária

A figura II.5 ilustra esta árvore binária.

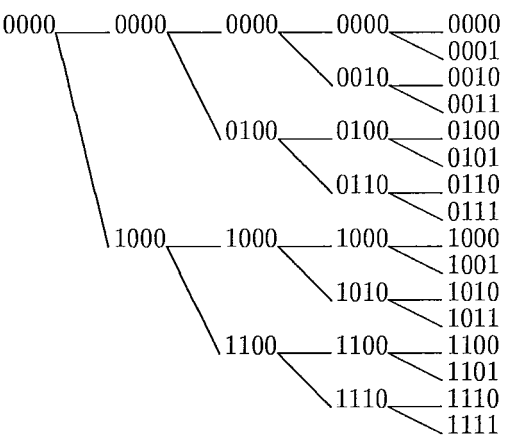


Figura II.5: Árvore binária balanceada para um hipercubo de dimensão quatro

Esta é somente uma árvore lógica já que alguns processadores ocorrem diversas vezes na árvore. Por exemplo, o processador 0000 ocorre D vezes, sendo a raiz de D subárvores. Cada processador está localizado exatamente uma vez em cada lado da árvore.

Uma alternativa é o mapeamento da rede hipercúbica numa árvore D-ária não balanceada.

No mapeamento em árvore, a rede hipercúbica é representada como uma árvore D-ária não balanceada, onde D é a dimensão do cubo. As árvores são formadas pela definição dos filhos de cada processador ou do pai de cada processador (nodo).

O filho de um nodo são os processadores cujos números de identificação são obtidos ativando-se cada bit de menor ordem na representação binária de D-dígitos. Por exemplo, na figura II.6 o nodo 000 tem como filhos 001, 010 e 100.

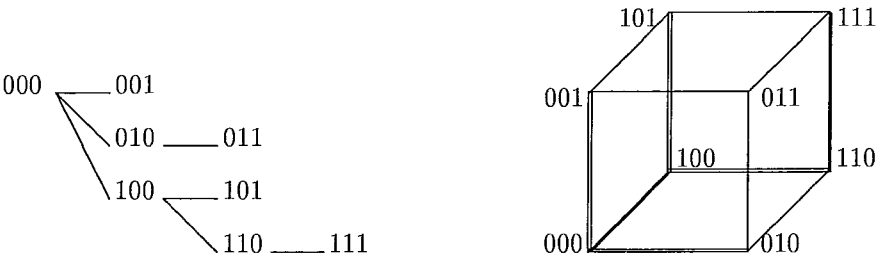


Figura II.6: Hipercubo mapeado em árvore de dimensão três

O pai de um nodo é obtido com uma operação inversa, isto é, desativando-se o bit de menor ordem na representação binária de D-dígitos do seu número de identificação. Essa árvore pode ser vista na figura II.6.

Essas duas árvores lógicas são duas maneiras diferentes de representar o mesmo conjunto de conexões. Portanto, no implemento do mapeamento em árvore em um hipercubo empregamos uma árvore D-ária.

O processador zero foi escolhido como o processador raiz por ser mais conveniente, já que este está conectado ao hospedeiro e realiza as funções de entrada e saída no hipercubo baseado em *transputers*, como citado anteriormente na seção II.1.

A seguir são mostrados a estrutura que representa uma árvore binária e o seu algoritmo de construção.

```
estrutura em_árvore {
        processador pai
        ponteiro para os processadores filhos
        }
```

pt_árvore = árvore_binária ()

pt_árvore = armazena área para a estrutura em árvore;
Se tem filhos
  pt_filhos = identificação dos processadores filhos;
Se *processador* $\neq$ *RAIZ*
  pai = identificação do processador pai;
Retorna *pt_árvore*.

Supondo então um hipercubo de dimensão três, teríamos a estrutura do processador cuja identificação é 100 preenchida com os seguintes valores:

- processador pai = 000;
- ponteiro para os processadores filhos = endereço de memória onde está armazenado num *array* os filhos 101 e 110.

II.3.3 Mapeamento em anel

É possível mapear uma estrutura de anel num hipercubo associando a cada processador a sua posição segundo a seqüência BRGC. Esta seqüência possui a propriedade de que vizinhos nesta ordenação lógica (vizinhos lógicos) são também vizinhos no hipercubo (vizinhos físicos). Os termos distância lógica e distância física são empregados sob o mesmo critério. Essa propriedade é ilustrada na figura II.7.

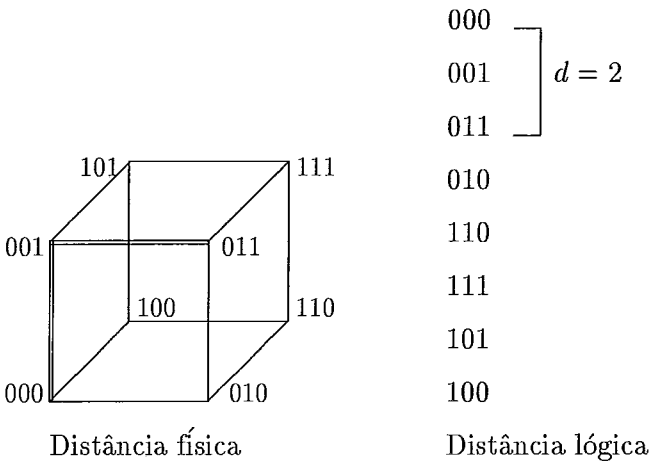


Figura II.7: Distâncias lógicas e físicas em um hipercubo de dimensão três

Em todo anel com distância lógica 2^d, onde $d = 1, \ldots, D-1$ e sendo D a dimensão do cubo, os processadores estão a uma distância física no máximo igual a 2 (Como ressaltado na figura II.7). Cada valor de d é um nível de subanel que pode ser criado no hipercubo. E todo processador está em exatamente um subanel de nível d.

Um subanel de distância lógica 2^d é composto por 2^{D-d+1} processadores, então existem $2^D/2^{D-d+1} = 2^{d-1}$ subanéis cobrindo o hipercubo no nível d. Assim para um cubo de dimensão D igual a três temos:

- níveis de subanéis possíveis $d = 1, 2$;
- nível $d = 1$;
  - distância lógica $2^d = 2$;
  - quantidade de subanéis cobrindo o hipercubo $2^{d-1} = 1$;
  - distância física = 1;
- nível $d = 2$;
  - distância lógica $2^d = 4$;
  - quantidade de subanéis cobrindo o hipercubo $2^{d-1} = 2$;
  - distância física = 2;

A figura II.8 mostra as estruturas de anéis lógicos e físicos que podem ser construídos em um hipercubo de dimensão três.

A estrutura que representa os níveis dos anéis e o algoritmo de construção dos mesmos são apresentados a seguir.

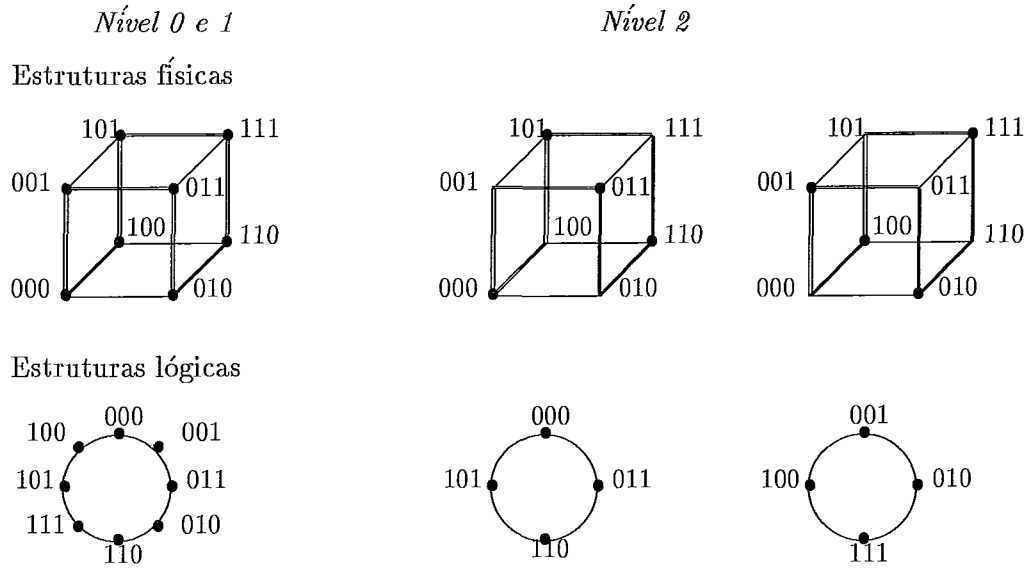


Figura II.8: Construção de anéis em um hipercubo de dimensão três

estrutura em_anel {
processador à direita no anel
processador à esquerda no anel
}

anel ()

Para todo *nível* = 0 a *nível* < D
posição = posição do processador à direita no *nível*;
p_direita = identificação do processador na posição à direita;
posição = posição do processador à esquerda no *nível*;
p_esquerda = identificação do processador na posição à esquerda;

Após a chamada dessa rotina de construção de anel temos, por exemplo, no processador cuja identificação é 000 de um hipercubo de dimensão três, uma estrutura preenchida da seguinte forma:

- no nível 0
  - processador à direita = 001;
  - processador à esquerda = 100;
- no nível 1
  - processador à direita = 001;
  - processador à esquerda = 100;
- no nível 2
  - processador à direita = 011;
  - processador à esquerda = 101;

II.4 Resumo

O multiprocessador hipercúbico baseado em *transputers* utilizado foi o NCP I desenvolvido pela COPPE que tem atualmente disponível oito *transputers* T800.

A seqüência BRGC apresentada além de fundamental para as topologias de grade, árvore e anel, também é de grande importância na distribuição, convergência e armazenamento dos dados nos processadores do sistema hipercúbico.

Os mapeamentos vistos (grade, árvore e anel) são utilizados internamente aos procedimentos que necessitam de comunicação entre os processadores.

Os procedimentos que utilizam a seqüência BRGC e os mapeamentos de grade, árvore e anel serão vistos no capítulo IV.

Capítulo III

Comunicação no Transputer

Alguns procedimentos da biblioteca de operações paralelas necessitam de comunicação entre os processadores. Toda comunicação realizada pela biblioteca de operações é feita internamente aos procedimentos tornando-a invisível. Nela está contida toda a dependência da arquitetura da rede hipercúbica. Esse capítulo relata a metodologia empregada para a troca de mensagens no multiprocessador hipercúbico baseado em *transputers*.

III.1 Os Processos

Um sistema complexo geralmente é composto por um conjunto de objetos ou dados que definem o escopo de um problema computacional. O primeiro passo na computação paralela consiste em decompor este sistema em partes (processos concorrentes) que operem de forma cooperativa para se obter uma solução em menor tempo para o problema.

Uma característica da biblioteca de operações paralelas é que cada processador recebe o mesmo código de programa, e se comunica através da troca de mensagens.

O multiprocessador hipercúbico baseado em *transputers* é uma coleção de elementos de processamento idênticos (como visto na seção II.1), chamados processadores ou nodos. Cada processador está relacionado a um único número de identificação que o diferencia dos demais. Uma vez que cada processador recebe a mesma cópia do programa (um único processo), o número de identificação é muito importante para se referenciar a um processador ou processo específico dentro da rede, pois é dado ao processo a mesma identificação do processador. Desta forma cada processador executa as mesmas instruções sobre um conjunto de dados diferentes, tornando o hipercubo, abstratamente, uma máquina virtual do tipo *Single-Instruction Multiple-Data* (SIMD).

No multiprocessador hipercúbico baseado em *transputers* não existe a facilidade direta para a entrada e saída paralela dos dados. Isso é feito de modo indireto através do processador raiz. Desse modo encontramos a necessidade de se criar um processo especial chamado processo de controle que manipula a entrada e saída de dados. Por causa de sua função especial, o processo de controle tem pouca realização computacional, tratando somente da captação, da difusão e da convergência dos dados. Este processo reside no processador raiz que está ligado ao computador hospedeiro. Todos os processos podem se referir ao processo de controle. Este também tem seu próprio número

de identificação cujo valor é igual a quantidade de processadores participantes da rede. Assim, supondo que o hipercubo tenha dimensão três, este é composto por $(2^3 = 8)$ oito processadores, cujos processadores têm números de identificação na faixa de $[0 \ldots 7]$, e o processo de controle recebe como identificador o número oito.

Os programas de aplicação neste modelo correspondem então a criação de dois processos: um de controle, que manipula a entrada e saída dos dados, e um processo igual para todos os nodos, que realizam a computação do problema propriamente dito. Esses dois processos se comunicam de maneira coordenada. A figura III.1 mostra o modo como estão interligados todos os processos em um hipercubo de dimensão três.

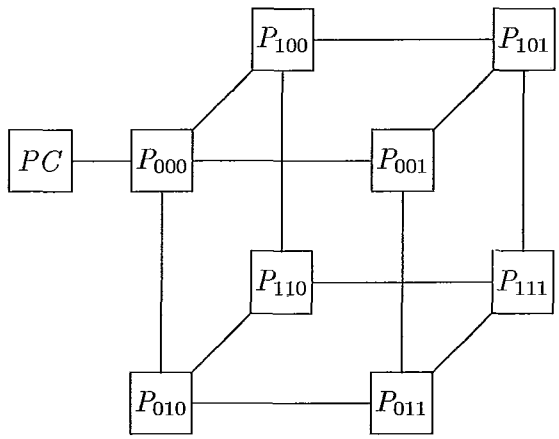


Figura III.1: Processos em um hipercubo de dimensão três

III.2 Os Canais de Comunicação

Toda comunicação entre os processadores é realizada pela troca de mensagens através de canais de comunicação. Assim, os canais são o único meio de comunicação entre os processadores. Cada canal realiza a comunicação ponto-a-ponto entre dois processadores e tem associado a ele um número de identificação. Uma maneira conveniente de distingüí-los é estabelecer uma relação entre os números de identificação dos processadores nos extremos do canal, já que estes são únicos e diferem em apenas um bit no seu número de identificação. Para tanto, utilizamos como identificador do canal o número do bit diferente entre os números de identificação dos nodos. A tabela III.1 mostra os identificadores dos canais de comunicação do processador 100 com seus processadores vizinhos em um hipercubo de dimensão três.

A figura III.2 mostra todos os canais de ligação entre os processadores de um hipercubo de dimensão três. A comunicação entre o processo de controle e o processo residente no processador raiz se dá através do canal com identificação sempre igual a dimensão do hipercubo adicionado de uma unidade. O canal com identificação igual a dimensão é deixado para a identificação dos processos como um resultado de implementação como pode ser visto no apêndice A.

Processadores	Canal
101	0
110	1
000	2

Tabela III.1: Identificadores dos canais de comunicação do processador 100 com seus vizinhos

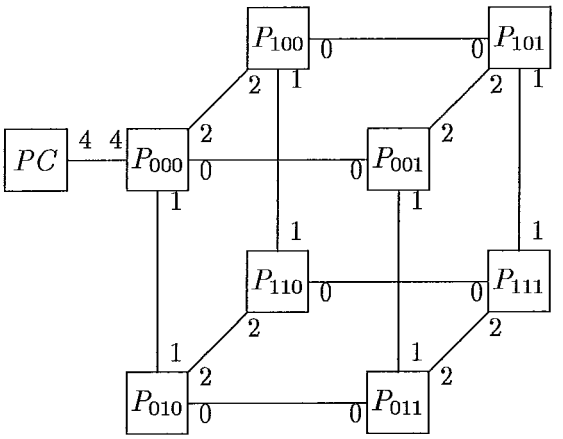


Figura III.2: Canais de ligação em um hipercubo de dimensão três

III.3 Roteamento

O multiprocessador hipercúbico baseado em *tranputers* possui um sistema de comunicação ponto-a-ponto e síncrono. A comunicação ponto-a-ponto nos leva a necessidade das mensagens precisarem ser enviadas ao longo de rotas que passam por outros processadores que não são sua origem ou seu destino. O sincronismo faz com que uma mensagem só possa ser enviada a um processador se este estiver preparado para recebê-la. Estas duas características do sistema de comunicação pode levar a uma situação de *deadlock* quando os processos não estiverem corretamente coordenados. Para que não haja *deadlock* na comunicação é necessário então se estabelecer um caminho para o envio e a recepção de mensagens de modo síncrono e coordenado.

O método de roteamento empregado nas comunicações realizadas internamente aos procedimentos da biblioteca de operações paralelas utiliza o roteamento *E-Cube*[6], que é considerado um método de caminho mais curto (*shortest path*). Se os processadores fonte e destino estão distantes K ligações, a mensagem passará exatamente por K processadores. A topologia hipercúbica binária tem uma rede de interconexão rica, com $K!$ rotas entre quaisquer dois processadores que estão distantes K ligações. Essas rotas formam uma árvore de busca de altura K com uma quantidade de folhas $K!$. A figura III.3 mostra uma árvore de busca de todas as rotas possíveis do processador zero ao processador sete em um hipercubo de dimensão três.

O caminho destacado é selecionado pelo método de roteamento *E-Cube* para enviar uma mensagem do processador zero ao processador sete. O roteamento de-

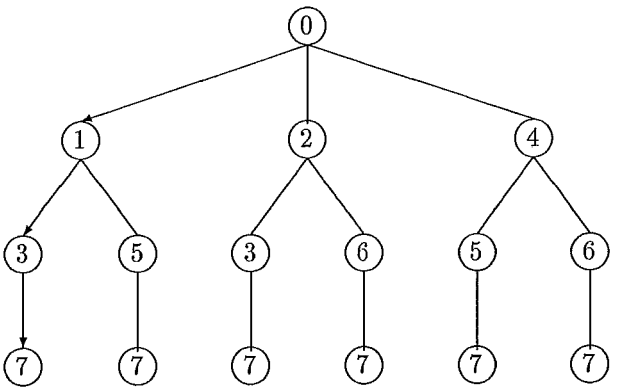


Figura III.3: Árvore de busca em um hipercubo de dimensões três

termina um caminho fixo, livre de *deadlock* entre quaisquer dois processadores usando a topologia da rede hipercúbica.

Embora qualquer rota possa ser utilizada, o roteamento *E-Cube* restringe o caminho escolhido dependendo da identificação dos processadores envolvidos na comunicação. O caminho é estabelecido trocando o bit diferente de mais baixa ordem entre os números de identificação dos processadores fonte e destino. O algoritmo *E-Cube* é mostrado a seguir:

p = identificação do processador;
p_destino = identificação do processador destino;
p_receptor = identificação do próximo processador na rota;
Se p = *p_destino*
    Libera mensagem;
    Retorna flag indicativa de sucesso;
Caso contrário
    p_receptor = troca o bit menos significativo diferente entre os números de identificação de p e *p_destino*;
    Envia mensagem ao *p_receptor*;
    Se mensagem enviada
        Retorna flag indicativa de sucesso;
    Caso contrário
        Retorna flag indicativa de erro.

Assim, para se transferir uma mensagem do processador fonte 000 ao processador destino 111 em um hipercubo de dimensão três, temos o seguinte caminho dado pela tabela III.2.

A quantidade de bits que diferem entre os números de identificação dos processadores de dois nodos dá a distância entre os nodos.

Processador Fonte	Processador Destino	Próximo Processador
000	111	001
001	111	011
011	111	111

Tabela III.2: Roteamento de uma mensagem do processador 000 ao processador 111

III.4 Envio e Recepção de Mensagens

Como mencionado anteriormente a comunicação entre os processadores é realizada pelo envio de mensagens através dos canais de comunicação que conectam pares de processadores. Toda comunicação entre processadores deve ter operações recíprocas envolvendo transmissão e recepção coordenadas. Estes procedimentos de transmissão e recepção devem ser complementares.

O procedimento criado para transmitir uma mensagem formata um identificador da mensagem com informações sobre os processadores fonte e destino e tamanho da mensagem. Transfere os dados da mensagem para um *buffer*, define o próximo processador na rota estabelecida e o canal pelo qual a mensagem deve ser enviada. Envia o identificador da mensagem seguida então da mensagem propriamente dita. Por fim, libera o *buffer* utilizado para armazenar a mensagem.

O procedimento de recepção complementar ao de transmissão, recebe inicialmente o identificador da mensagem e a mensagem em um *buffer*. Verifica se a mensagem chegou ao destino retransmitindo-a, se necessário, a um outro processador no caminho estabelecido pelo procedimento de roteamento. Depois de recebida a mensagem libera o *buffer* utilizado na transferência.

A tabela III.3 mostra as rotinas implementadas que combinadas realizam a troca de mensagens no sistema *transputer*. O apêndice A apresenta, passo a passo, a implementação e os algoritmos dessas rotinas.

canal_de_escrita ()	canal por onde a mensagem deve ser enviada
canal_de_leitura ()	canal por onde a mensagem deve ser recebida
roteamento ()	próximo processador a receber a mensagem
envia_mensagem ()	
recebe_mensagem ()	

Tabela III.3: Rotinas Relativas à Comunicação

III.5 Resumo

Na solução de problemas utilizando a biblioteca de operações paralelas cada processador deve receber um único processo. Esse está relacionado a um número de identificação que o diferencia dos demais. Além desses processos, existe um processo de controle que reside no processador raiz, que manipula a entrada e a saída dos dados.

Toda comunicação entre processadores é realizada pela troca de mensagens através de canais de comunicação que também são diferenciados por números de identificação.

A comunicação entre processadores é realizada através de rotas pré-estabelecidas, baseadas no algoritmo de roteamento *E-Cube*.

Os procedimentos para a troca de mensagens são síncronos e coordenados para evitar *deadlock*.

Capítulo IV

Organização de Memória

As várias operações que compõem a biblioteca de operações paralelas são realizadas com todos os processadores chamando simultaneamente a rotina apropriada com seus argumentos e operando sobre os seus segmentos locais, ignorando a existência de seus vizinhos na rede. O tamanho do vetor ou a dimensão da matriz deve ser maior ou pelo menos igual ao número de processadores participantes da rede para que todos tenham pelo menos um elemento do vetor ou da matriz. Neste capítulo descreveremos as estruturas de dados que representam os vetores e as matrizes, as formas em que eles podem ser armazenados e os passos elementares que compõem a biblioteca de operações paralelas.

IV.1 Rotinas Vetoriais Básicas

Todos os processadores recebem uma cópia da biblioteca de operações paralelas e do programa de aplicação, bem como os vetores distribuídos adeqüadamente. As rotinas são chamadas simultaneamente por todos os processadores e operam sobre os seus segmentos locais, ignorando a existência de seus vizinhos na rede.

Os vetores são representados por uma estrutura de dados que contém todas as informações necessárias à sua manipulação dentro das rotinas. Esta estrutura é mostrada a seguir:

estrutura vetorial {
tipo = tipo de armazenamento vetorial;
h = tamanho do segmento vetorial local ao processador;
tamanho = tamanho total do vetor;
pt_início = ponteiro do início da área de dados;
pt_fim = ponteiro do fim da área de dados;
pt_antes = ponteiro auxiliar de deslocamento;
pt_depois = ponteiro auxiliar de deslocamento;
}

O campo *tipo* especifica a forma de armazenamento do vetor. Vetores com diferentes tipos de armazenamento podem ser misturados livremente, quando necessário as conversões apropriadas podem ser aplicadas. Os dois tipos existentes são *SIMPLE* e *SHIFT*, cuja definição pode ser vista na seção IV.1.1.

Os campos h e *tamanho* são relativos ao tamanho do segmento local ao processador e ao tamanho total do vetor. Os outros campos são ponteiros para facilitar a manipulação da área de dados.

Algumas rotinas vetoriais não utilizam comunicação entre os processadores e são trivialmente paralelizáveis. Fazem parte deste grupo as rotinas de armazenamento e de liberação de memória, conversão entre os tipos de armazenamento, cópia vetorial, soma e multiplicação entre vetores e escalares. A operação correspondente é realizada em cada processador sobre o segmento local. Desse grupo somente o procedimento de armazenamento vetorial será apresentado detalhadamente neste capítulo. As rotinas onde há a necessidade de comunicação, como a de deslocamento vetorial e a de produto interno entre outras, serão vistas em detalhes nas seções seguintes. As rotinas vetoriais e suas funções são apresentadas na tabela IV.1. Todos os procedimentos vetoriais estão descritos no apêndice B.

aloca_vetor	armazena segmento local de um vetor
deleta_vetor	libera memória atribuída a um vetor
converte_vetor	troca o tipo de armazenamento
vetor_nulo	$u_i = 0$
copia_vetor	$u_i = v_i$
soma_vet_a_esc_vet	$u_i = u_i + sv_i$
soma_esc_vet_a_vet	$u_i = su_i + v_i$
soma_vet_a_vet_vet	$u_i = u_i + v_i w_i$
desloca_vetor	$u_i = u_{(i-s) \bmod N}$
desloca_vetor_ótimo	$u_i = u_{(i-s) \bmod N}$
máximo_vetor	$max_{0 \leq i < N} u_i$
soma_vetor	$\sum_{i=0}^{N-1} u_i$
produto_interno	$\sum_{i=0}^{N-1} u_i v_i$

Tabela IV.1: Rotinas Vetoriais

IV.1.1 Armazenamento Vetorial

Para distribuir vetores aos P processadores da rede hipercúbica, consideramos somente aqueles vetores cujo tamanho é igual ou maior a P. Assim todos os processadores receberão pelo menos um elemento do vetor.

Um vetor distribuído desta maneira é chamado vetor distribuído e todos os vetores do mesmo tamanho serão sempre distribuídos da mesma forma aos processadores, isto é, segmentos de vetores diferentes residindo no mesmo processador têm o mesmo tamanho. A fórmula para a distribuição de um vetor de tamanho N aos P processadores que compõem a rede hipercúbica é:

$$N = hP + r, \quad 0 \leq r < P.$$

Essa fórmula garante que cada processador tenha um segmento de tamanho igual a pelo menos $h = N/P$ elementos do vetor, caso o tamanho do vetor seja um múltiplo do número de processadores. Se o tamanho do vetor não for um múltiplo do número de processadores, então aos r primeiros processadores na seqüência BRGC (secão II.2) serão atribuídos os segmentos de tamanho $h + 1$ elementos e aos $P - r$ processadores restantes serão atribuídos segmentos de tamanho h elementos.

Existem dois tipos de representação vetorial que são chamados tipo *SIMPLE* e *SHIFT*. Na forma *SIMPLE*, cada segmento vetorial é armazenado em um *array* de tamanho igual a h ou $h+1$. No formato *SHIFT*, cada segmento local é armazenado em um *buffer* de tamanho $2h$ dentro do qual será permitido deslocamentos vetoriais como serão vistos mais adiante na seção IV.1.2. Inicialmente o segmento é centralizado no *buffer*.

As figuras IV.1 e IV.2 demonstram como um vetor de tamanho total igual a vinte elementos pode ser distribuído em um hipercubo de dimensão três, tanto para o tipo *SIMPLE* quanto para o tipo *SHIFT*.

$h = N/P$	$r = N \bmod P$
$h = 20/8$	$r = 20 \bmod 8$
$h = 2$	$r = 4$

Os quatro primeiros processadores na seqüência BRGC recebem três elementos do vetor e os outros quatro restantes recebem dois elementos.

IV.1.2 Deslocamento Vetorial

Esta operação é essencial uma vez que várias outras operações a utilizam. A operação de deslocamento vetorial consiste em deslocar cada elemento do vetor distribuído m vezes, enviando m elementos para o processador seguinte e recebendo m elementos do processador precedente. Para esta operação é necessário que cada processador contenha pelo menos um elemento do vetor.

Existem dois procedimentos para realizar o deslocamento vetorial, o *desloca_vetor* e o *desloca_vetor_ótimo*. Esses procedimentos vêem a rede hipercúbica como uma hierarquia de anéis e exigem que o vetor envolvido na operação esteja armazenado

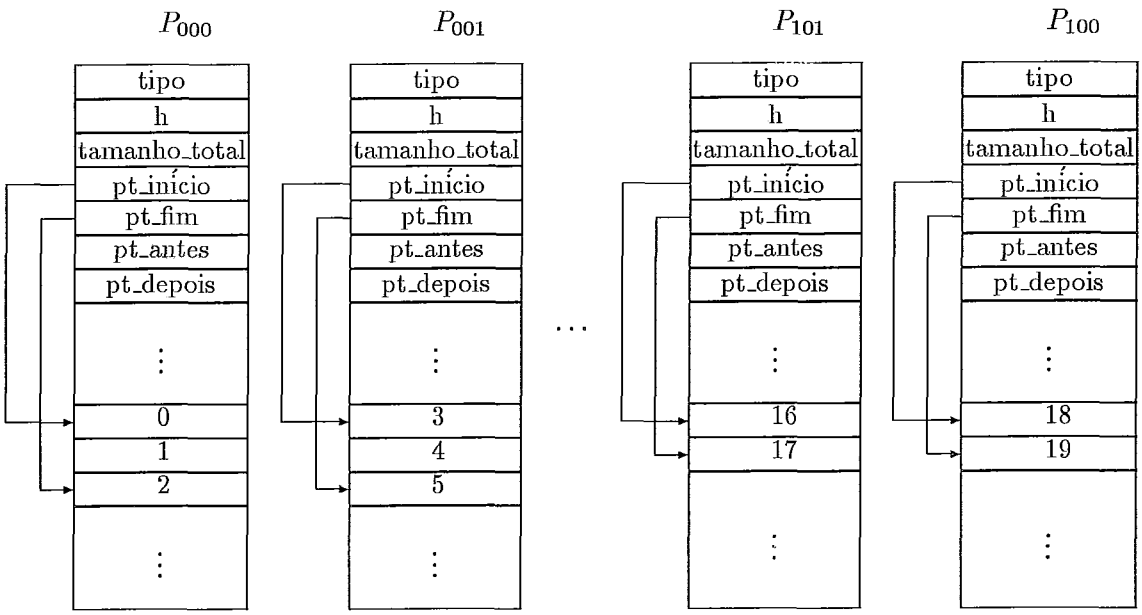


Figura IV.1: Vetor distribuído do tipo *SIMPLE* em um hipercubo de dimensão três

no formato *SHIFT*, isto é, tenha uma memória extra em cada processador, um *buffer* de extensão $h/2$ para cada lado do segmento.

• **Procedimento** *desloca_vetor*

O procedimento *desloca_vetor* é utilizado quando o tamanho total do vetor não é um múltiplo do número de processadores. Para deslocamentos pequenos como o de um elemento, os deslocamentos são feitos no anel de nível 0 e os vetores então em vez de serem deslocados são simplesmente deixados no lugar e, adiciona-se o novo elemento vindo do processador vizinho na posição apropriada do vetor local. O ponteiro do início do vetor na estrutura de dados correspondente é incrementado para cada deslocamento. Após $h/2$ deslocamentos elementares, se faz necessária uma operação para recolocar o vetor no seu ponto inicial em cada processador. A colocação na posição inicial é automaticamente detectada e realizada quando o ponteiro auxiliar de deslocamento alcança uma das extremidades do *buffer*.

As figuras IV.3, IV.4 e IV.5 ilustram as memórias dos processadores durante a operação de deslocamento de um elemento em um vetor distribuído de tamanho total dez em um hipercubo de dimensão dois. A figura IV.3 ilustra a área de dados da memória de cada processador com a posição inicial dos elementos do vetor distribuído.

A figura IV.4 mostra o novo elemento vindo do processador vizinho no anel e adicionado na posição apropriada.

A figura IV.5 ilustra a área de dados da memória de cada processador com a posição final do vetor com seus elementos já centralizados.

• **Procedimento** *desloca_vetor_ótimo*

A operação de deslocamento vetorial otimizada é utilizada somente quando o vetor é um múltiplo do número de processadores ($N = hP$). Se consistir de muitos des-

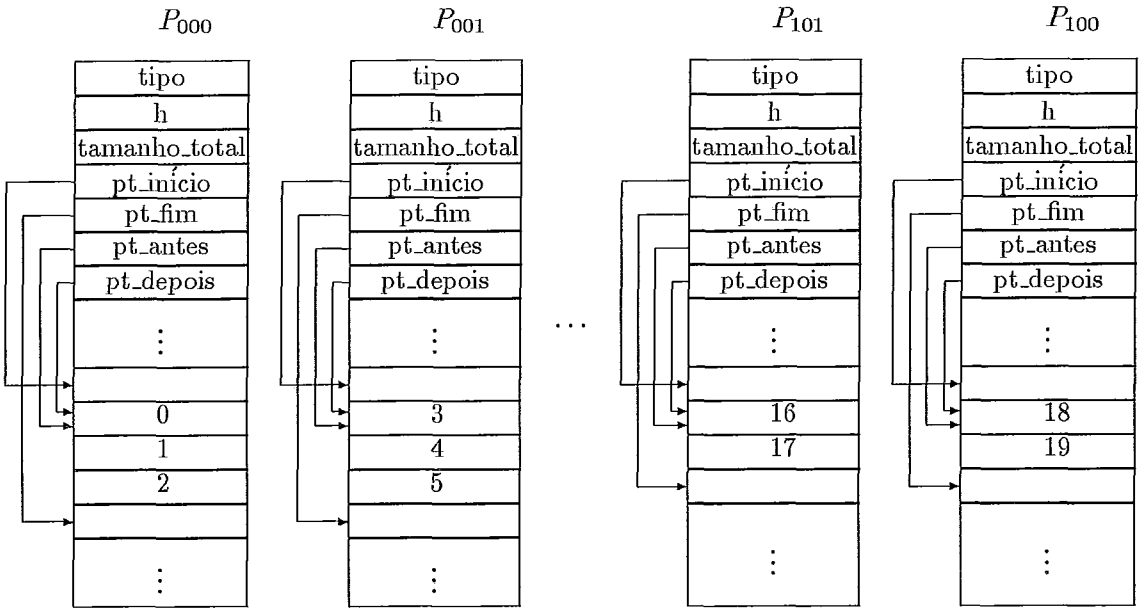


Figura IV.2: Vetor distribuído do tipo *SHIFT* em um hipercubo de dimensão três

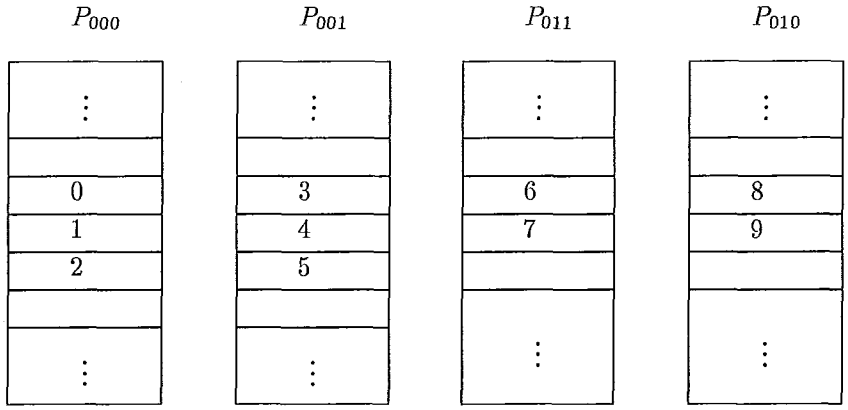


Figura IV.3: Posição inicial

locamentos de um elemento em sucessão, esta operação pode ser feita mais eficientemente utilizando a hierarquia de anéis em vários níveis ao invés de só no nível 0 como visto anteriormente em *desloca_vetor*.

Uma grande quantidade de deslocamentos m, pode ser dividida em quantidade de deslocamentos de segmento m_s e em quantidade de deslocamentos de elementos m_e. Então podemos escrever que

$$m = m_s h + m_e, \quad 0 \leq m_e < h,$$

onde h é o tamanho do segmento local ao processador, igual em todos os processadores já que $N = hP$.

Uma operação física de comunicação de transferência de elementos e transferência de segmentos aos processadores vizinhos seguem o mesmo critério. Então um deslocamento de segmento resulta em cada processador transmitir o seu segmento local completo ao processador seguinte no anel e receber um segmento completo do processador precedente no anel.

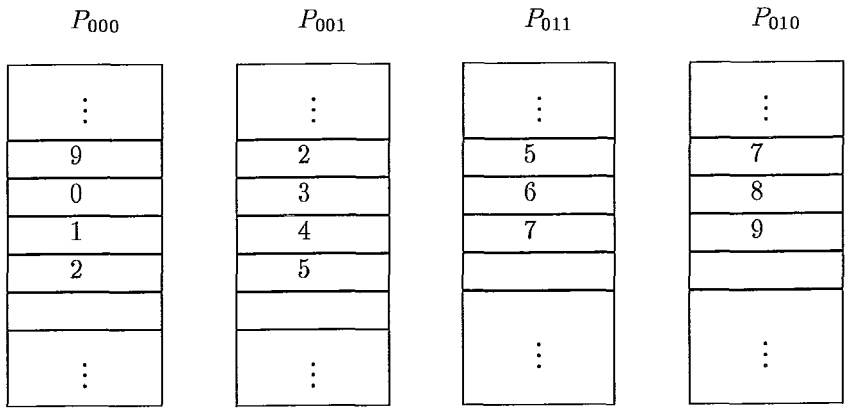


Figura IV.4: Deslocamento de elemento

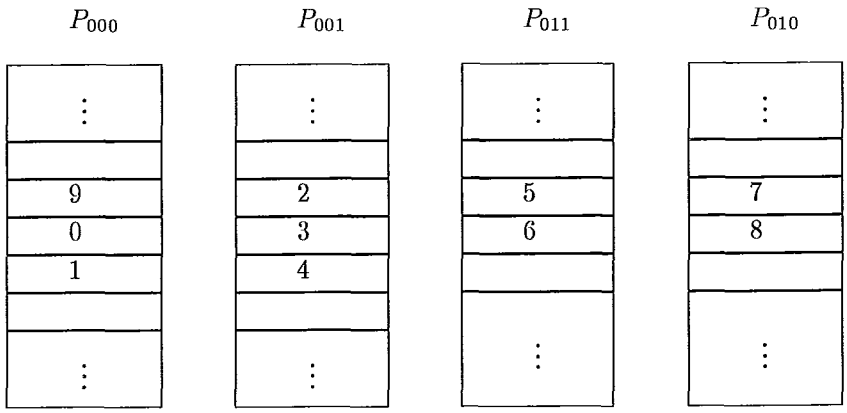


Figura IV.5: Vetor deslocado

Se o deslocamento m é muito grande podemos ter então $m_s \geq P$. Se $m_s \geq P$ podemos reduzir o deslocamento de segmentos fazendo $m_s = m_s \bmod P$, uma vez que deslocar de P segmentos é equivalente a um deslocamento de $N = hP$ elementos e cujo efeito é de não haver nenhum deslocamento. Uma outra redução é possível observando quando $m_s > P/2$. Nesse caso, é melhor deslocar os segmentos na direção oposta $m_s - P$ vezes. Se todas as reduções forem feitas em m_s então teremos deslocamentos somente na faixa de $-P/2+1, \ldots, P/2$.

Os m_e deslocamentos de elementos restantes são feitos como descrito anteriormente em *desloca_vetor*.

Para aumentar a velocidade dos deslocamentos de segmentos empregamos a hierarquia de anéis. Considere a representação binária de m_s dada por $b_{D-1} \ldots b_1 b_0$. Para cada $i > 0$ tal que b_i está ativado, necessitamos realizar 2^i deslocamentos de segmento, que reduz-se a apenas um deslocamento de segmento no subanel lógico de nível i. Como visto na seção II.3.3, isto é feito com no máximo dois deslocamentos físicos no subanel de nível i. Isto tem o efeito de realizar 2^i deslocamentos de segmento levando o tempo de somente duas transferências físicas de segmento. O bit 0 é uma excessão porque quando ativo requer somente um deslocamento no anel principal, levando o tempo de uma transferência de segmento. Assim no pior caso teremos todos os $D-1$ bits em m_s ativos.

Como exemplo, suponhamos um cubo de dimensão dois, um vetor de tamanho vinte e um deslocamento de 77 vezes. O procedimento então seria:

tamanho do segmento local	deslocamento de segmentos	deslocamento de elementos
$h = N/P$	$m = m_s h + m_e$	$m_e = m \bmod h$
$h = 20/4$	$m_s = 77/5$	$m_e = 77 \bmod 5$
$h = 5$	$m_s = 15$	$m_e = 2$

Realizando as reduções no deslocamento de segmentos temos:

$m_s \geq P$	$m_s > P/2$
$m_s = m_s \bmod P$	$m_s = m_s - P$
$m_s = 15 \bmod 4$	$m_s = 3 - 4$
$m_s = 3$	$m_s = -1$

Resumindo, o deslocamento total foi subdividido em um deslocamento de segmento na direção oposta mais dois deslocamentos de elementos individuais. A figuras IV.6, IV.7, IV.8 e IV.9 demonstram os passos realizados no deslocamento. A figura IV.6 mostra a área de dados da memória de cada processador com a posição inicial dos elementos do vetor distribuído.

Na figura IV.7 é dada a posição dos elementos do vetor após o deslocamento de um segmento na direção oposta. O deslocamento foi realizado no nível 0 do anel, já que a representação binária correspondente é 01, e somente o bit 0 está ativado.

A figura IV.8 mostra a transferência dos dois elementos ao processador vizinho no anel.

A figura IV.9 apresenta então a posição final dos elementos do vetor distribuído.

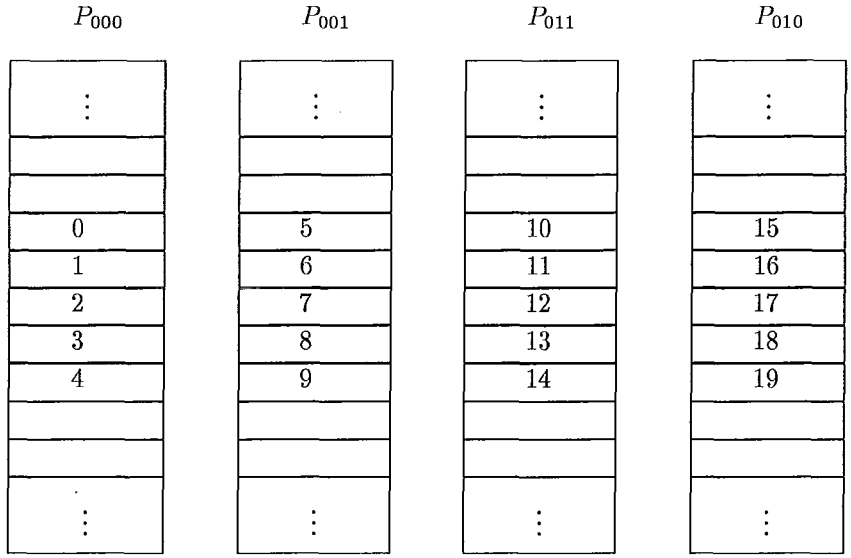


Figura IV.6: Posição inicial

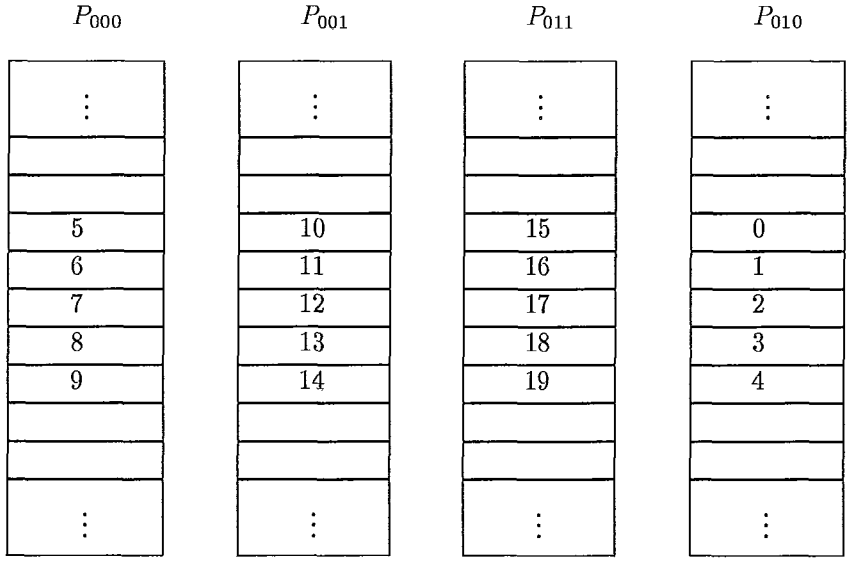


Figura IV.7: Deslocamento de segmento

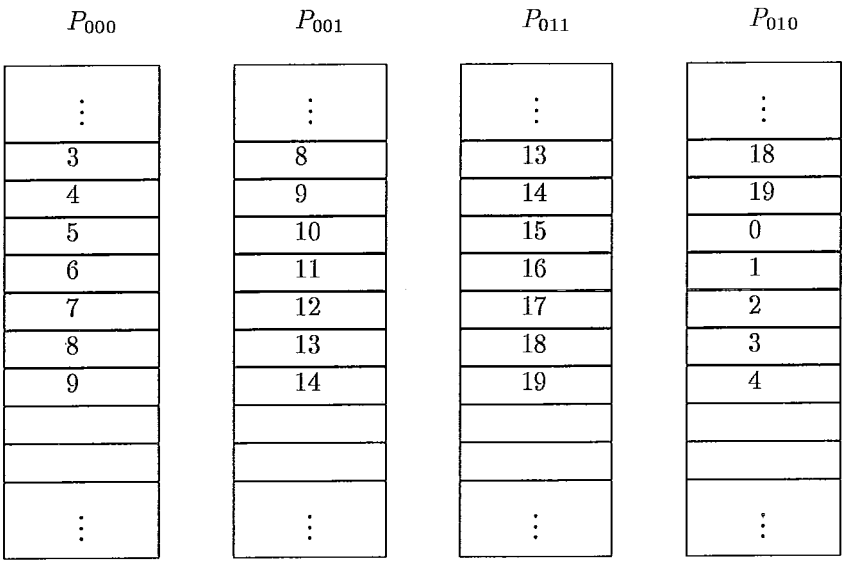


Figura IV.8: Deslocamento de elemento

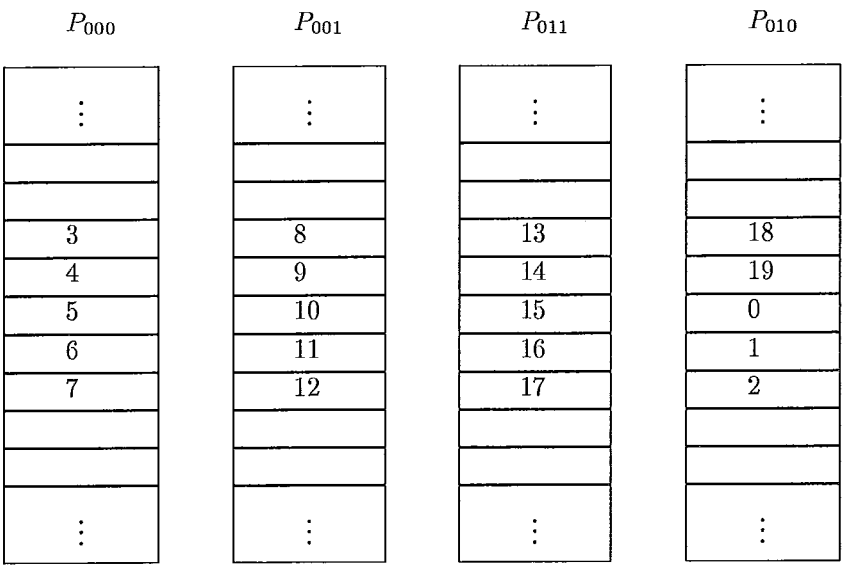


Figura IV.9: Vetor deslocado

IV.1.3 Elemento Máximo de um Vetor Distribuído

Para se obter o elemento máximo em um vetor basta que todos os processadores simultaneamente utilizem o procedimento *máximo_vetor*. Ao final desta operação todos os processadores terão o valor do elemento máximo do vetor distribuído. Nenhum processador é distingüido nesta semântica.

A implementação naturalmente distingüe os processadores. A operação realiza comunicação entre os processadores. Inicialmente cada processador encontra o seu elemento máximo do segmento local. Uma árvore D-ária é mapeada à rede hipercúbica, com o processador 0 como raiz (como visto na seção II.3.2). Os processadores se comunicam então com o pai e com os filhos e realizam novas comparações até que todos obtenham o mesmo resultado. O procedimento *máximo_vetor* tem o seu algoritmo mostrado a seguir.

D = dimensão do hipercubo;
pt_árvore = árvore_binária ();
máximo = máximo_vetor_local (pte_vetor);
Para $i = 0$ a $i < D$ faça
    Se tem filhos
        Recebe *máximo* dos filhos;
    Se tem pai
        Envia *máximo* ao pai;
    máximo = maior elemento entre os máximos local e dos filhos;
Para $i = 0$ a $i < D$ faça
    Se tem filhos
        Envia *máximo* aos filhos;
    Se tem pai
        Recebe *máximo* do pai;
Retorna *máximo*.

O custo dessa implementação é $O(N/P)+O(logP)$. Todas as comparações dos segmentos locais são realizadas em paralelo em um tempo $O(N/P)$, proporcional ao tamanho do segmento local $h = N/P$. As comparações parciais sobre uma árvore D-ária podem ser computadas em um tempo D, se a unidade de tempo consiste numa comparação de um número real mais a comunicação deste número ao seu vizinho no hipercubo.

Para comprovar esta afirmação note que uma árvore D-ária não balanceada com raiz no nodo 000..0 contém $(D-1)$-ária,$D-2$-ária,...,0-ária subárvores com seus filhos 100..0, 010..0, 001..0, respectivamente. Pelo princípio de indução assumimos que o resultado é verdadeiro para cada subárvore de $0, 1, \ldots, (D-1)$. A hipótese é verdade para $D = 0$ e $D = 1$. Pela hipótese de indução, a comparação do nodo 100..0 chegará ao 000..0 em um tempo $D-1$, do nodo 010..0 em um tempo $D-2$ e assim por diante. Cada comparação parcial pode ser realizada em 000..0 enquanto chegam as próximas comparações parciais. Assim todas as comparações serão completadas em um tempo D, completando a indução. A difusão do resultado aos processadores individuais seguem os passos acima e requerem um tempo $O(D)$.

A figura IV.10 e a figura IV.11 mostram o mapeamento em árvore empregado internamente à rotina e os passos do procedimento para achar o elemento máximo do vetor distribuído, respectivamente, em um hipercubo de dimensão dois.