New Topologies and Parallel Algorithms for Static Interconnection Networks

Doctoral Dissertation

by

Srabani Sen Gupta

under the supervision of

Professor Bhabani P. Sinha

Indian Statistical Institute
203, B.T. Road, Calcutta - 700 035, India

1996

Dedication

To my mother

Acknowledgement

I never thought of writing a thesis in Computer Science. It was my supervisor Professor Bhabani P. Sinha who has initiated me to this subject. He taught me the bulk of whatever little I learnt in Computer Science. I express my deep gratitude for his invaluable guidance and supervision.

It was my privilege to have numerous interesting and fruitful discussions with Professor Bhargab B. Bhattacharya. Whenever I needed help, he always obliged. I must also mention the encouragement, advice and moral support I received from Professor Jayasree Dattagupta, Dr. Nabanita Das, Dr. Susmita Sur-Kolay and Professor Mihir K. Chakrabarti. My heartfelt thanks to all of them.

I express my gratitude to all my colleagues in the Institute, with whom I have the opportunity to work closely during the period of my research. I am specially indebted to the coauthors of my research papers.

I enjoyed the delightful company of my friends Rajib Das, Debasish Das, Mallika De, Sandeep Das and Sudeshna Basu who always cheered me up during the frustrations of research. I acknowledge their assistance and help on countless occasions.

I would like to thank the Indian Statistical Institute for providing me the financial support during this period and giving me the opportunity to get acquainted with distinguished researchers and many nice people, named and unnamed.

The love, affection and support of my family members has always given me the strength and inspiration. Special mention goes to Krishnendu, my friend, coauthor, critic, assistant and advisor who has always been with me in the best and worst of times during these days.

Srabani Sen Gupta

Srabani Sen Gupta

December, 1996

Indian Statistical Institute

Calcutta

Contents

Chapter 1

Introduction

1.1 Introduction

Many real-life applications in the areas of signal processing, image processing, etc., require a large amount of fast computations to be performed. Although high speed powerful processors are currently available due to the phenomenal advances in VLSI technology, the increasing demand for massive real-time computations can not be met just by a uniprocessor system. One way of achieving the goal of fast computation is through *parallel processing*. In parallel processing, a problem is broken into several subproblems, which are distributed among different processors so that each of the processors can perform its task simultaneously. Main areas of recent research in parallel processing include parallel architectures and algorithms, parallel models and complexity classes, programming languages, operating systems and compilers for parallel computers, etc.

Interprocessor communication in a parallel processing system is effected through either a shared memory or an interconnection network. Interconnection networks may be of two types : (i) static and (ii) dynamic. In static interconnection networks, there are fixed links among the nodes for which the connection pattern can not be changed. On the other hand, in dynamic interconnection networks, the interconnections among the nodes are made through links and switches and

as a result, different input-output connections can be established by changing the switch settings.

A static interconnection network is usually represented by means of a network graph. The nodes of this graph represent the processors and the edges stand for the interprocessor links. A good network topology should have the following desirable features :

i) small number of links to reduce the cost of interconnection, ii) low node degree to limit the number of I/O ports per processor, iii) low diameter to reduce the inter processor communication delay, iv) high degree of fault tolerance, v) regularity, vi) symmetry, vii) incremental extensibility, viii) high bisection width, ix) easy routing scheme in fault free as well as faulty situations, x) ease of mapping algorithms, etc.

The problem of designing a network topology, simultaneously satisfying all the above requirements, is very difficult as some of them are mutually conflicting. For example, the objectives of achieving low diameter as well as low node degree are in conflict with each other. Diameter can be reduced by introducing more links in the topology which, in turn, increases the node degree. Fault tolerance of the network is another issue which is also increased with increase in node degree. The usual approach to deal with all these problems is to design a near-optimal topology that offers a fair compromise among the above requirements, depending on the needs of the specific application. Designing such a network topology constitutes one of the interesting areas of research in parallel processing.

Ring, mesh, tree, hypercube, [A89], [L92], etc., are some examples of popular static networks. Ring topology is widely used mainly due to its structural symmetry and simplicity. Although the routing algorithm in this network is very simple, it suffers from a large communication delay. This communication delay can be reduced if additional links are introduced within a ring in a uniform way. One possible topology resulting from such modifications is the distributed loop network [BT91]. Variations of such modifications leave ample scope for further research.

Hypercubes [H69], [L92], star graphs [AK84], etc., offer a good compromise between the degree and the diameter. In a hypercube, both these parameters are reduced to the logarithmic order on the number of nodes N, while in a star graph, both degree and diameter are of sublogarithmic order on N. Moreover, there are some topologies in which the node degree is constant (independent of the number of nodes in the graph), but the diameter is still of logarithmic order. Moebius graphs [LS82], de Bruijn graphs [B46], cube-connected cycles (CCC) [PV81] are examples of such fixed degree graphs. The gaps between the successive values of N for which these graphs are defined, are however, quite large. Hence, these topologies are not incrementally extensible. Design of suitable topologies having logarithmic diameter and constant/logarithmic degree with lesser gaps between the successive values of N constitutes an interesting problem for investigation.

Different practical situations often impose restrictions on the size N (the number of nodes) of the network. The diameter (D) and the node degree (δ) are often influenced by the number of nodes and vice versa. The inter-dependence among these three design parameters N, δ, and D is well studied in the literature. In most of the existing topologies except a very few, like generalized hypercube (GHC) [BA84], radix-r de Bruijn graph [PR82], etc., only one out of these three parameters can be chosen independently. GHC, radix-r de Bruijn graph, however, offer more design flexibility in the sense that two of the three parameters can be chosen independently. Further investigations in this area are still called for.

Design of efficient parallel algorithms for various applications, e.g., numerical computations, signal processing, robotics, graph problems, etc. [A89], [J92], [R93], is another thrust area of research in parallel processing. In designing parallel algorithms, usually two different approaches are being followed. One approach is to start from a given network topology and then design a suitable algorithm for the problem in question that can be efficiently mapped on that architecture. The other approach is to start from a sequential algorithm for the given problem, to identify all the concurrent steps therein and finally to devise a suitable processor interconnection scheme so as to best exploit the inherent parallelism in the algo-

rithmic steps. The second approach is mainly followed in designing a dedicated system for a specific problem.

1.2 Scope of the Thesis

This thesis addresses some issues involving (i) the design of efficient static interconnection networks, and (ii) the design of parallel algorithms for some real-life problems implemented on different architectures.

1.2.1 Static Network Topologies

1.2.1.1 Topologies with Multiple Loops

We propose a new class of network topologies with multiple loops, which have some attractive properties such as low diameter and low degree, incremental extensibility, easy routing, etc. In the proposed interconnection network, N processors are interconnected to form a graph $G(m, N)$, where m is a parameter of the graph such that $(m-1) \times 2^{\lfloor \frac{m-1}{2} \rfloor + 1} < N \leq m \times 2^{\lfloor \frac{m}{2} \rfloor + 1}$. N is an even multiple of m. For a given m, we can construct the graph for different values of N, satisfying the above constraints.

The graph $G(m, N)$ can be constructed by suitably adding extra edges over a ring. $G(m, N)$ is thus hamiltonian. It has an average node degree of 3 when m is odd and $(3 + \frac{1}{m})$ when m is even, and the maximum node degree of 4. The diameter of $G(m, N)$ is bounded above by $\lfloor \frac{11m}{8} \rfloor + 1$. Another interesting feature of this graph is its incremental extensibility. The gaps between the successive values of N for which the network can be defined, is only $2m$.

For the proposed topology, there exists a simple routing algorithm for point-to-point communication in a path of length less than or equal to the diameter. The underlying graph is biconnected and in the presence of a single node failure, it

can be shown that the diameter may increase by at most 6. The implementation details of an important class of *Ascend/Descend* algorithms on this topology have also been discussed.

1.2.1.2. Generalized Hypercube-Connected-Cycles

A new family of interconnection networks, *Generalized Hypercube-Connected-Cycles* (GHCC), has been proposed. This topology is represented by a regular graph $G(l, m)$, where l and m are two integer parameters influencing the diameter and the degree respectively. The total number of nodes in $G(l, m)$ is lm^l. The degree of each node is $m-1$ for $l = 1$, m for $l = 2$ and $(m+1)$ for $l \geq 3$. The diameter of the graph is $\lfloor 5l/2 \rfloor - 2$ for $l \neq 1, 2, 3$, $m \neq 1$ and $\lfloor 5l/2 \rfloor - 1$ for $l \leq 3$, $m \neq 1$. As special cases, this network reduces to a ring for $m = 1$, while it reduces to a complete graph for $l = 1$. For $m = 2$ the topology reduces to a CCC [PV81]. This family of network topologies offers a specific design flexibility in that any two out of the three design parameters N, δ and D can be chosen independently. For example, suppose the diameter D and the size of the network N are given *a priori*. In that case, the value of l can be chosen appropriately so that the diameter of the topology will be in the range $D - 2 \leq$ diameter $\leq D$ and the total number of nodes lm^l will be very close to (greater than or equal to) the given value N. This will fix the value of m which, in turn, fixes the node degree at $\lceil \frac{N}{\lceil 0.4(D+2) \rceil} \rceil^{\frac{1}{\lceil 0.4(D+2) \rceil}}$. In case of Generalized Hypercubes (GHC) [BA84], the node degree will be bounded below by $D[N^{\frac{1}{D}} - 1]$ in a similar situation. For example, in the neighborhood of 150 nodes, GHCs with diameters 3, 4 and 5 should have node degrees at least 13, 10 and 9 respectively; whereas in GHCC, the node degrees for the required diameter should be 9, 6 and 5 respectively, which are lesser than those of GHC. On the other hand, with the degree-diameter pair (13, 3) and (9, 5), the proposed topology permits the construction of a network with 288 and 1563 nodes respectively compared to just 150 nodes in case of a GHC.

The connectivity of the graph is $(m+1)$ for $l \geq 3$. In the presence of m faulty

nodes, the diameter of the network increases by at most 6. Algorithms for point to point routing and single node broadcast have been designed. Implementations of different algorithms including the *Ascend/Descend* class of algorithms have also been discussed.

1.2.2 Design of Efficient Parallel Algorithms

1.2.2.1 Isomorphism of Maximal Outerplanar Graphs

Maximal outerplanar graphs [H69] constitute an important class of graphs, often encountered in various applications, e.g., computational geometry, robotics, etc. Testing isomorphism of maximal outerplanar graphs is an interesting problem of research. The best known sequential algorithm for testing isomorphism of maximal outerplanar graphs (with N vertices) requires linear time [BJM79], whereas, the best known parallel algorithm requires $O(\log^2 N)$ time on an EREW shared memory model with $\frac{N}{\log^2 N}$ processors [LLP+90].

We know that there exists a one-to-one correspondence between the maximal outerplanar graphs and the planar embedding of triangulated convex polygons. In this thesis, we consider the isomorphism testing of maximal outerplanar graphs in terms of triangulated convex polygons. We unfold different interesting features of triangulated convex polygons and then with the help of these characteristics, we introduce a new scheme for classification of triangulated convex polygons, namely, bisector, scalene, isosceles and equilateral triangulations. Next, we propose a parallel algorithm for identifying the class to which a given triangulation belongs. This algorithm is then utilized to design a parallel algorithm for testing isomorphism of triangulated convex polygons, which is equivalent to testing isomorphism of maximal outerplanar graphs. Given the ordered adjacency lists of the two graphs, the proposed algorithm tests their isomorphism in $O(\log N)$ time using N processors, for graphs with N nodes on an exclusive-read exclusive-write (EREW) shared memory model, as well as on a hypercube architecture. If, however, the graphs are

represented in terms of their adjacency matrices, this algorithm can be remodeled on N^2 processors to run in $O(\log N)$ time.

1.2.2.2 Lagrange Interpolation

In many real-time applications, we may need to evaluate a function, say $F(x)$, at a given value of x, whose analytical form is unknown, but only a set of values of the function $F(x)$ at some discrete values of x are given. Such problems can be tackled by different numerical interpolation techniques.

In this thesis, we propose a parallel algorithm, based on Lagrange interpolation method. For N-point Lagrange interpolation, our algorithm has been implemented using $N(2\log_2 N - 1)$ processors with $O(N/\log N)$ time complexity. Further, we have shown that this algorithm can also be implemented on $p(2\log_2 p - 1)$ processors where $N = kp$, k being an integer greater than 1. In that case, the time complexity is $O(k^2 p/\log p)$. This shows that the algorithm can also be implemented when processors fewer than $N(2\log_2 N - 1)$ are available. It is found that in both the cases, the AT cost is $O(N^2)$.

1.2.2.3 Recursive Matrix Algorithms

Parallel implementation of recursive algorithms on systolic architectures are not always straight-forward. Such parallelization requires a prior knowledge and planning of the intermediate computations and communications (e.g., broadcast of data items) involved in the recursive calls. As these communications and computations entirely depend on the nature of the recursive calls, devising a general technique for parallel implementation of such recursive algorithms on systolic architectures appears to be a difficult problem.

In this thesis, we would like to address the problem for parallelization of a specific class of recursive matrix algorithms where, in each recursive call, the matrices are partitioned into smaller submatrices of equal size. Strassen's algorithm for

matrix multiplication [S69], Pease's algorithm for matrix inversion [P69], etc., are examples of this class of algorithms. We have taken up the recursive matrix multiplication algorithm as a representative case, for implementing its parallel version on a systolic architecture.

It comes out that it is possible to implement the Strassen's recursive algorithm for multiplying two $n \times n$ matrices on a hypercube in $O(\log n)$ time using n^3 processors, keeping the essence of each recursive step in tact. The basic philosophy of intermediate computations and data communications adopted in this implementation can as well be extended to the recursive matrix inversion algorithm due to Pease.

Chapter 2

A Brief Review

2.1 Introduction

In this chapter, we briefly review some of the existing works on the topics which fall under the scope of this present thesis. Accordingly, we first discuss some important aspects which have been studied in designing interconnection networks and some popular static network topologies. Next, we would review the earlier works related to the three problems, namely, matrix multiplication, polynomial interpolation and testing isomorphism of maximal outerplanar graphs.

2.2 Static Network Topologies

2.2.1 Interdependence between Degree and Diameter

Among the different measures for the performance of an interconnection network, two key parameters are the maximum *node-degree* and the *diameter* of the network.

There are several related problems involving degree, diameter and the number of nodes in the network. The one which is discussed most in the literature is the (d, k) graph problem [HS60], [E64], [A65], [F66], [S70], [W72], [MR82]. The (d, k)

graph problem consists in finding the maximum number of vertices $N(d,k)$ of a graph with given maximum degree d and the diameter k.

A theoretical bound on $N(d,k)$ was given by Moore [B74], [B78]. It has been shown that

$$N(2,k) \leq 2k+1$$

and for $d > 2$

$$N(d,k) \leq (d(d-1)^k - 2)/(d-2)$$

The above bounds are known as *Moore bounds* and the graphs satisfying the Moore bounds are called the *Moore graphs*. However, it has been proved by different authors [HS60], [L70],[F71], [BI73], [B74] that Moore graphs exist only for a few combinations of d and k. Hoffman and Singleton [HS60] have proved that Moore graphs of diameter 2 exist only for $d = 2$, 3, 7 and possibly for 57; for $d = 3$ and 7 the graphs being a Petersen's graph with 10 vertices and Hoffman-Singleton's graph on 50 vertices respectively. Sachs [S64] proved that for $k = 3$ the Moore graphs exist only if $d-1$ is a power of some prime number.

The determination of exact value of $N(d,k)$ appears to be a very difficult problem. A lot of constructions have been given in the literature [F66], [K67], [S70], [TS79], [MR82], [D84], [SB93] which exhibit large graphs with given degree and diameter. The results on (d,k) graphs are summarized in [BDQ82].

From the point of view of the degree and diameter, the existing topologies can broadly be classified into two categories as discussed below.

The first category includes the hypercubes [H69], [L92], generalized hypercubes [BA84], folded hypercubes [LA89], star graphs [AK84], pancake graphs [AK89], radix-r de Bruijn graphs [PR82], etc. For these topologies, there exist functional relationships (explicit or implicit) among the number of nodes N, degree δ and the diameter D. For example, in a hypercube with N nodes, both the degree and the diameter are $\log_2 N$, an order n star graph has node-degree $n-1$, diameter $\lfloor \frac{3(n-1)}{2} \rfloor$ and total number of nodes $(n-1)!$.

The second class consists of topologies with Constant degree. The ring [FL72], chordal ring [AL81], distributed loop [BT91], mesh [AK83], cube-connected-cycles [PV81], Moebius graph [LS82], radix-2 de Bruijn graph [B46] etc. are well known members of this class. In case of these constant degree graphs the maximum node-degree does not depend on the network size, it is fixed for all. For example, cube-connected-cycles, Moebius graph etc. are regular graphs with degree 3. However, the diameters of these networks vary with the sizes of the networks.

We present here a brief review on a few popular members of both these two classes of networks.

2.2.2 Hypercube

Hypercube is one of the most popular network topologies in the area of parallel and distributed processing. An n-cube or an n-dimensional hypercube, Q_n, is recursively defined as $Q_n = Q_{n-1} \times K_2$ $(n > 0)$, where K_2 is a complete graph of 2 vertices, Q_0 consists of a single node and '$\times$' represent cartesian product on graphs [H69]. A 3-cube is shown in Fig. 2.1.

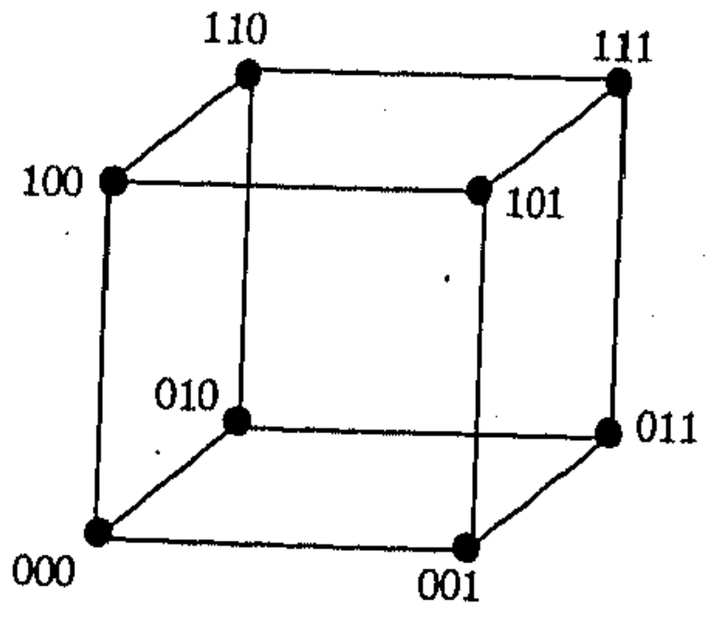


Figure 2.1: The Q_3 with 8 nodes

Different attractive features of a hypercube Q_n are :

1) the total number of nodes in Q_n is $N = 2^n$.

2) it is a regular network topology with node degree n.

3) it is node symmetric as well as edge symmetric.

4) it has a diameter $D = n = \log_2 N$.

5) an easy routing scheme exists for this topology.

6) it is strongly connected.

7) it can be constructed recursively.

Each node in Q_n can be labeled by a n-bit binary string $a_{n-1}a_{n-2}\cdots a_0$ such that the labels of two adjacent vertices differ in exactly one bit position.

A tremendous amount of research has been done on this topology due to its versatile applications in the field of parallel processing. Several interesting properties of a hypercube architecture has been investigated in the literature [H76], [F77], [M82], [AG81], [ES83], [HHW88], [SS88], [AP89], [BOS+91], [L92]. Algorithms for various problems have been designed and implemented [L92] on this architecture as well. A lot of work has also been done on reliability and fault tolerance of hypercube [L89], [TRS90], [YTR94]. Different fault-tolerant communication scheme has been reported in [F92], [LH92].

In spite of several good features, the hypercube topology has inadequacies as well.

1) An n-cube is defined on 2^n nodes, which shows that this architecture is defined on a very restricted class of points. Moreover, the gap between consecutive allowable sizes of hypercubes is $O(N)$, which is very large for practical applications.

2) The degree of the nodes in the hypercube increases with the increase in size of the network.

To overcome these drawbacks, several variations of hypercubes have been reported in the literature. Incomplete hypercube [K88], generalized hypercube [BA84], cube-connected-cycle [PV81] etc. are examples of such attempts. Another approach, which we often find in the literature, is to reduce the diameter through

some modifications over the original hypercube architecture. These include the folded hypercube [LA89], bridged hypercube [AL90], twisted cube [ENS91], crossed cube architecture [E92], varietal hypercube [CC94], bridged and twisted hypercube [DMS94], dBcube [CAB94a] etc. This chapter includes reviews on some of these architectures also.

2.2.3 Generalized Hypercube

A general class of hypercube structure, the generalized hypercube (GHC), was presented by Bhuyan and Agrawal in [BA84]. The interconnection is based on mixed radix number system and the technique results in a variety of hypercube structures for a given number of processors N, depending on the desired diameter of the network. *The generalized hypercube structure is defined as follows :*

Let N be the total number of processors and let it be represented as a product of r integers m_i $(m_i > 1)$, $1 \leq i \leq r$.

$$N = m_r * m_{r-1} * \cdots * m_1$$

Each processor X $(0 \leq X \leq N-1)$ is expressed as an r-tuple $(x_r, x_{r-1}, \cdots x_1)$, where, $0 \leq x_i \leq (m_i - 1)$. Processor $X = (x_r x_{r-1} \cdots x_{i+1} x_i x_{i-1} \cdots x_1)$ in GHC is connected to the processors $(x_r x_{r-1} \cdots x_{i+1} x_i' x_{i-1} \cdots x_1)$, $\forall\, i,\ 1 \leq i \leq r$, where x_i' takes all integer values between 0 and $(m_i - 1)$ except x_i itself.

Fig. 2.2 shows a 4 * 3 * 2 GHC structure.

The diameter of the GHC structure defined above is r and the degree of each node is $\sum_{i=1}^{r} (m_i - 1)$. When N_r^1 is an integer, a cost optimal GHC with diameter r is obtained if $m_i = N_r^1, \forall\, i,\ 1 \leq i \leq r$.

The GHC structure possesses the following characteristics :

1) the interconnection supports any number of nodes N.
2) the design is based on the allowable diameter of the network. If the diameter is increased, a structure with lower degree can be obtained.

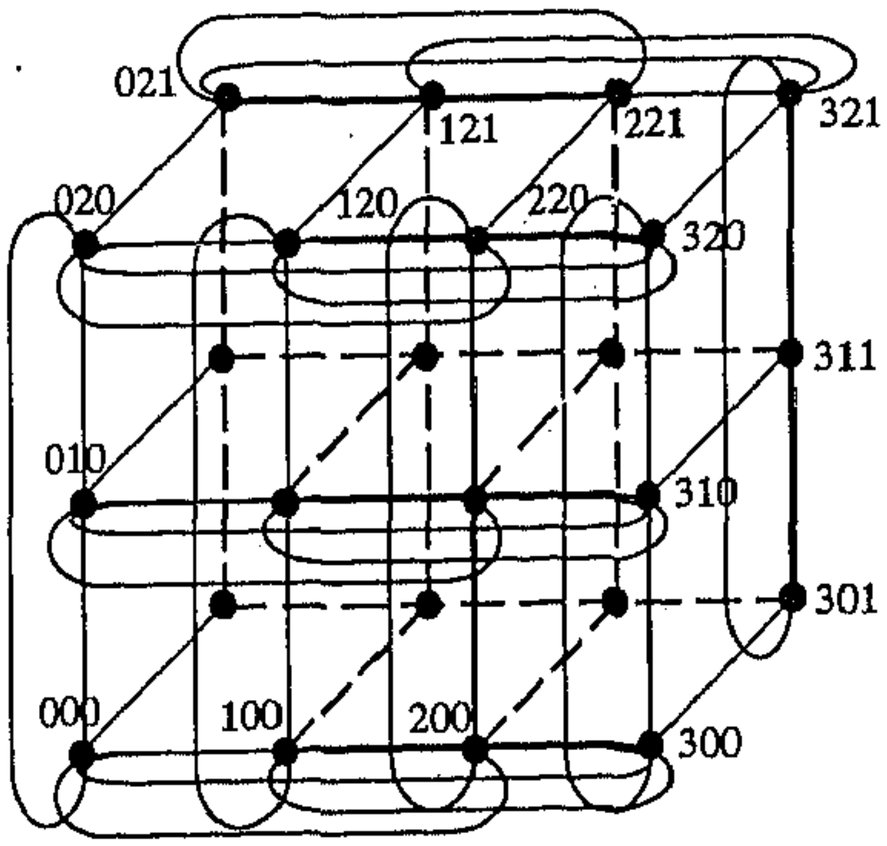


Figure 2.2: A 4 * 3 * 2 GHC structure

3) the structure is very general in nature. Single loop, boolean n-cube, nearest neighbor mesh hypercube and complete graph can be considered as a part of this general structure.
4) the structure is highly fault-tolerant.

2.2.4 dBCube

A dBCube, presented by Chen et al. [CAB94a], employs the hypercube topology as a basic cluster. It connects many such clusters using a de Bruijn graph. This class of networks have comparatively low node-degree while they retain many of the properties and the advantages of the hypercube topology by using it as a basic structure. The size of this class of regular networks can be easily extended by increment of a cluster size. Moreover, the local communications (to be satisfied by the hypercube topology) allow easy embedding of the existing parallel algorithms while the de Bruijn graph provides the shortest distance between clusters running different parts of an application. This is an example of two-level hierarchical network [DE91].

Chen et al. have also presented partially connected dBCube subclass (PdBCube) [CAB94b] in which the use of fewer nodes as gateways makes the network scalable in terms of both the hypercube and de Bruijn in size.

2.2.5 Star Graph

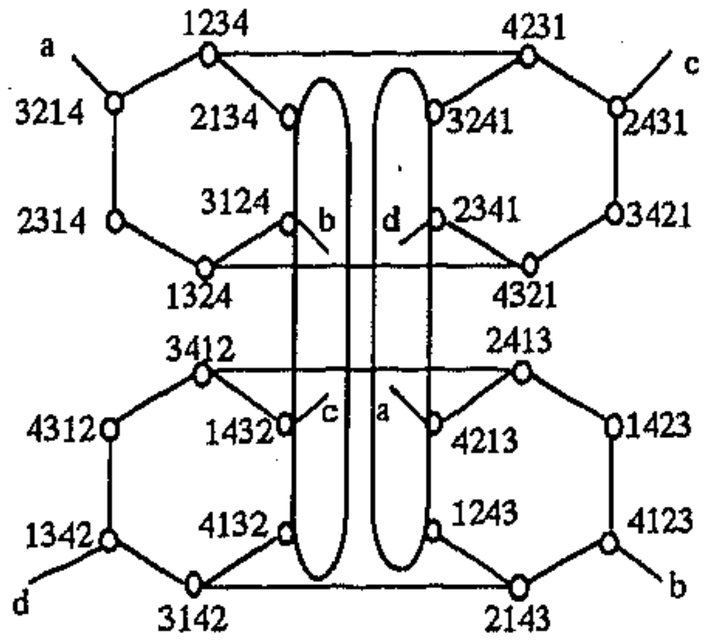
In recent years a special class of symmetric graphs, called *Cayley graphs*, is drawing a lot of attentions from the researchers. This class of graphs uses a group theoretic approach as a basis of defining graphs. Here, we consider a group to be the set of permutations generated by a set of generators.

Definition 2.1 *Given a set of generators for a finite group G, we can construct a graph, called Cayley graphs, in which the nodes correspond to the elements of the group G and the edges correspond to the action of the generators.*

A special class of Cayley graphs constitutes the Star graphs.

Definition 2.2 *An n-star graph is the Cayley graph on the group G consisting of all permutations on n symbols, and the set of generators g defined as follows. The set g consists of $n-1$ generators $\{g_2, g_3, \cdots, g_n\}$ where g_i switches the i^{th} symbol with the first (from the left) and leaves the remaining symbols in their original positions. So g consists of the following generators :*

$$g_2 = (2134\cdots n)$$
$$g_3 = (3214\cdots n)$$
$$g_4 = (4231\cdots n)$$
$$\vdots$$
$$g_n = (n234\cdots 1)$$



Generators:

$g_1 = 2134$

$g_2 = 3214$

$g_3 = 4231$

Figure 2.3: The 4-star network

A 4-star graphs is shown in Fig. 2.3.

An n-star graph is defined on $n!$ nodes. Different important features of an n-star graph as follows.

1) It is regular with node-degree $(n-1)$.

2) The diameter of this topology is $\lfloor 3(n-1)/2 \rfloor$.

3) The graph is node symmetric as well as edge symmetric [AK89].

4) An n-star graph can be viewed as n copies of $(n-1)$ star graphs which are interconnected by the edges corresponding to g_n. This decomposition can be carried out recursively.

5) It is maximally fault-tolerant [AK84]. Fault diameter of an n-star graph in the presence of upto $(n-2)$ faults, is only one more than the original diameter [RS93].

6) The gap between two consecutive allowable sizes of star graphs is very large and the topology is not at all incrementally extensible. To overcome this drawback, a variation of star graph, *incomplete star*, has been proposed [LB94].

In terms of degree and diameter, the star graphs are clearly superior to the hypercubes. Because of this and other interesting features of star graphs, this topology has been considered to be a good alternative to the popular hypercube structure for parallel processing.

2.2.6 Radix-r de Bruijn Graph

Pradhan and Reddy presented [PR82] a communication architecture for distributed processors which may be viewed as a generalization of the well known de Bruijn graph [B46] in radix-r number system, r is a parameter of the graph. The total number of nodes, N, is assumed to be r^n. This architecture interconnects N nodes by rN links, where the maximum internode distance is $\log_r N$ and each node has at most $2r$ I/O ports. The connection pattern is as follows :

Let $(i_{n-1}, i_{n-2}, \cdots, i_0)$ and $(j_{n-1}, j_{n-2}, \cdots, j_0)$ be the radix-r representations of the nodes i and j; $0 \leq i, j \leq N-1$. Node i is connected to the node j if at least one of the following two conditions holds :

$$i_w = j_{w-1}, \quad 1 \leq w \leq n-1$$

$$or, \; i_w = j_{w+1}, \quad 0 \leq w \leq n-2$$

Given N, one can use different values of r to construct different interconnections. The value of r may be selected on the basis of the number of I/O ports available per processors (i.e., the maximum node degree in the topology) and/or maximum allowable transmission delays for messages (i.e., the diameter of the topology).

This network is fault-tolerant and can tolerate upto $(r-1)$ node failures. Moreover, the authors developed a procedure using which each fault free node can diagnose the faulty nodes independently without use of any central observer.

2.2.7 Cube-Connected Cycles

An interconnection pattern of processing elements, *the cube-connected cycles* (CCC), was proposed by Preparata and Vuillemin [PV81]. This topology is a well known member of the class of constant degree graphs.

A cube-connected cycle is a network of N identical processors, where N usually takes the form $n.2^n$, for any positive integer n. These N processors are grouped into 2^n cycles, each consisting of n processors. The cycles are then interconnected in the form of an n-cube. Fig. 2.4 shows a CCC with $N = 3.2^3$ nodes.

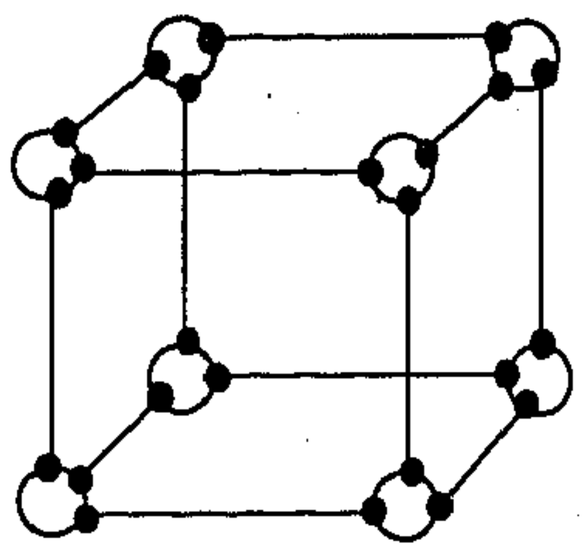


Figure 2.4: The cube-connected cycle with 3.2^3 nodes

CCC is a regular network with constant node-degree 3 and diameter $\lfloor \frac{5n}{2} \rfloor - 1$, when the total number of nodes is $N = n.2^n$.

By combining the principles of parallelism and pipelining, the CCC can emulate the cube-connected machine with no significant degradation of performance. In that sense, this structure is a feasible substitute for the cube-connected structure, as it has bounded number of communications per processor. The authors have also described how to utilize CCC for efficiently solving a large class of problems, called ASCEND and DESCEND class of problems which include FFT, sorting, permutations and other derived algorithms.

2.2.8 Loop Networks

The ring network [FL72] is widely used in design and implementation of local area networks and other configurations. This network topology is popular not only due to its structural symmetry and simplicity but it can also be easily extended by adding additional nodes. Moreover, the routing schemes for ring network is very simple. At the same time, this topology has some serious disadvantages also : (i) the ring topology has a low degree of reliability and hence low fault tolerance (ii) the high value of the diameter of a ring (in a ring with N nodes, the maximum distance between any pair of nodes is $\lfloor \frac{N}{2} \rfloor$) may in turn cause a large transmission delay.

Both of these problems can be overcome if additional links are introduced within a ring topology. For cost-effectiveness, it is desirable to add as few links as possible. Moreover, by adding links in a uniform manner, the other significant features of the ring topology like symmetry, expandability, uniform token passing, etc. can be preserved. The networks obtained by introducing chords over a ring are called *loop networks*. The Goal is to design loop networks with low degree and diameters.

Chordal Ring, proposed by Arden and Lee [AL81], is an example of such an extension. A chordal ring is a ring structured network where there is only one additional chord from every node in the ring. More formally, if the nodes in a chordal ring are numbered as 0, 1, $\cdots$, $N-1$ (N is assumed to be even) along the ring, then every odd numbered node i is connected to the node $(i+w) \mathrm{mod} N$. Here, w is called the chord length and assumed to be odd positive. Fig. 2.5 shows a chordal ring with 16 nodes and chord length 3. Thus chordal ring is a regular degree-3 network. In this topology the increment in node-degree results in reducing the diameter to $O(\sqrt{N})$ from $\lfloor \frac{N}{2} \rfloor$ in ring.

Another approach is to introduce two chords from every node in the ring. This results in regular graphs of degree 4. These structures are known as *distributed loop networks* or *double loop networks*. The underlying network graph $G(N;h)$ [T91] has the vertex set

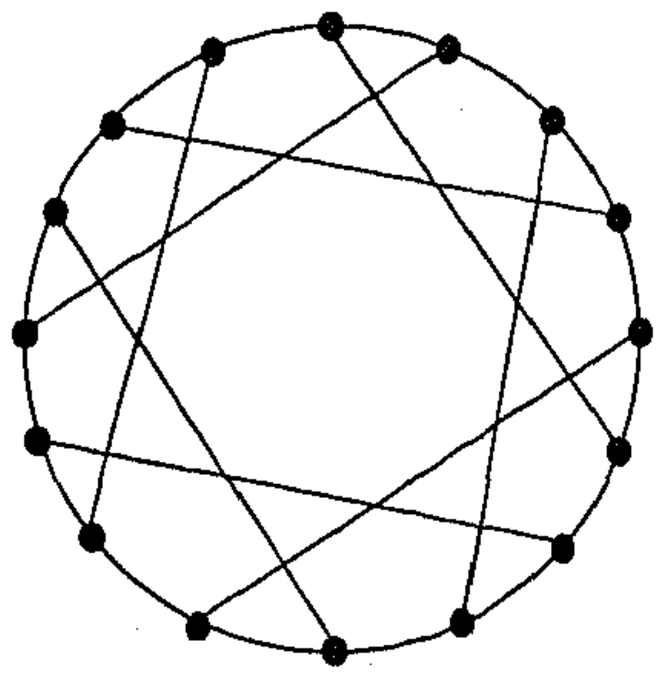


Figure 2.5: The chordal ring with $N = 16$ and $w = 5$

$$V(G) = \{0, 1, \cdots, N-1\}$$

and the edge set

$$E(G) = \{(i,j) | \, j = (i \pm 1) \mathrm{mod} N \text{ or } j = (i \pm h) \mathrm{mod} N\}$$

Here, h is referred to as the hop-size or simply hop. The graph $G(16,6)$ has been shown in Fig. 2.6. The term "double loop" signifies that these structures contain two hamiltonian cycles [BFM89]. These networks can be depicted as ILLIAC interconnection networks as well. The double loop networks are well studied in the literature and these are considered to be one of the popular networks. Several works have been done on the bound on the diameter and many other properties of $G(N;h)$.

Let $D^*_N = min_h\{DiamG(N;h) | 2 \leq h \leq N-2\}$. D^*_N denotes the minimal diameter of such a network with N nodes. Wong and Copersmith [WC74] found a lower bound $lb(N) = \sqrt{2N-3}/2$ for D^*_N. This bound has been made tighter to $\lceil(\sqrt{2N-1}-1)/2\rceil$ by [DHL+90], [BW85], [BIP85]. However, the lower bound $lb(N)$ may not be achievable for all values of N. Bermond and Tzviele [BT91] proved that $lb(N) = k$ if $N \in R[k]$, where $R[k] = \{2k^2 - 2k + 2, \cdots, 2k^2 + 2k + 1\}$.

For network size N, a hop h and the graph $G(N;h)$ are called optimal if the diameter of $G(N;h)$ equals to D^*_N and they are tight optimal if the diameter equals to $lb(N)$. Thus, a double loop network $G(N;h)$ can be optimal for some

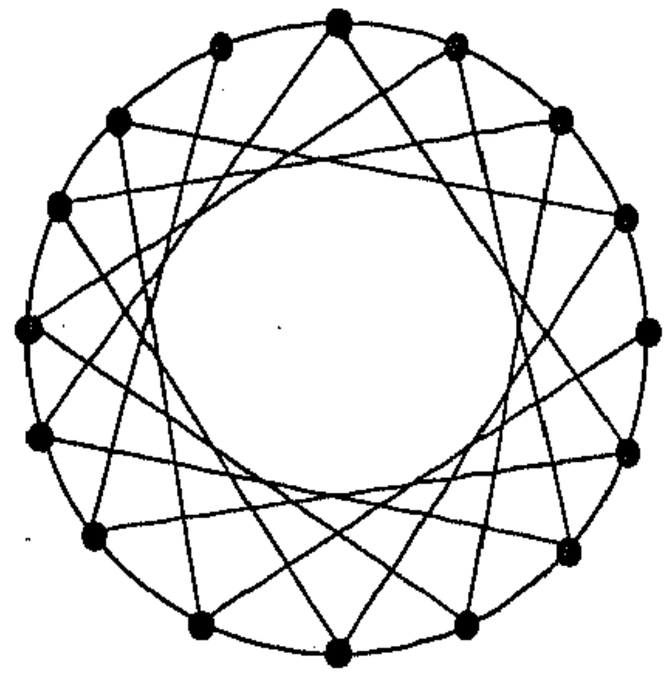


Figure 2.6: The distributed loop network $G(16, 5)$

hop h but may not be tight optimal if $D^*_N > lb(N)$.

In [BT91], authors determined dense families of values of N that are optimal and such that the computation of the optimal hop is easy. Infinite families of optimal networks along with the optimal hops are also identified in [T91] and [DHL90]. Several features regarding routing in double loop networks have been well studied in [MS95] and [DS95].

The generalization of double loop networks can be found in the literature. These double loop networks are actually special cases of an important class of graphs, called *circulants*. Circulants have been known in the graph theory for a long time. They were first introduced by Catalan in 1846 [D79]. A circulant $C_N(s_1, s_2, \cdots, s_k)$ is a graph with N nodes, numbered from 0 to $N-1$ and node i is connected to the nodes $(i \pm s_j) \mathrm{mod} N$, $\forall j, 1 \leq j \leq k$. Several works have been done on these graphs [D79], [BT84], [BW85].

Recently, a new topology called *recursive circulant* has been proposed by Park and Chwa in [PC94]. Recursive circulant $G(N; d)$ is defined to be a circulant graph with N nodes and jumps of powers of d, $d \geq 2$. That is, $G(N, d)$ is a circulant graph $C_N(d^0, d^1, d^2, d^3, \cdots, d^{\lceil \log_d N \rceil - 1})$. This topology can be recursively constructed when $N = cd^m$, $1 \leq c \leq d$. In that case the diameter will be $\lceil (3m - 1)/4 \rceil$. As an example, a recursive circulant has been shown in Fig. 2.7.

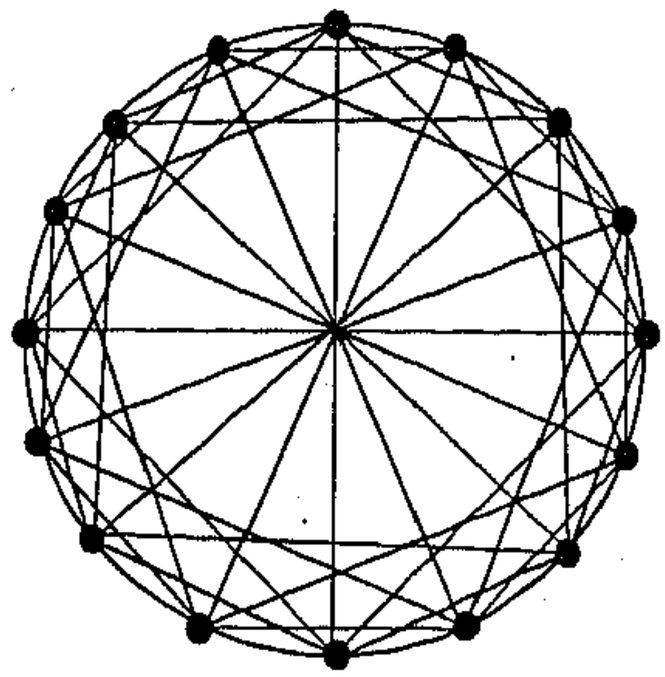


Figure 2.7: The recursive circulant $G(16, 2)$

2.3 Design of Parallel Algorithms

Design of efficient parallel algorithms is an area of extreme importance. In this section we discuss some of the earlier works for a few selected problems.

2.3.1 Isomorphism of Maximal Outerplanar Graphs

Two graphs $G_1 = (V_1, \varepsilon_1)$ and $G_2 = (V_2, \varepsilon_2)$ are isomorphic to each other iff there exist a one-to-one function f from V_1 onto V_2 such that v_1 and v_2 are adjacent in V_1 iff $f(v_1)$ and $f(v_2)$ are adjacent in V_2. The problem of testing whether two graphs G_1 and G_2 are isomorphic or not is known as the *graph isomorphism* problem. This problem has been much studied in the literature [RC77].

It is still not known that whether the general graph isomorphism problem is solvable in polynomial time or it is an NP complete problem [GJ79]. But the related subgraph isomorphism problem is proved to be NP complete. Good heuristics exist in the literature for the general problem, but in the worst case these algorithms have exponential time complexity. One such algorithm is due to Corneil and Gotlieb [CG70]. However, the isomorphism problem for some special classes of graphs can be solved in polynomial time. Trees [CB81a], [AHU74], interval graphs

[CB81], [LB79], series-parallel graphs [W66], planar graphs [HW74], hamiltonian 2-sep chordal graphs [VKV91], partial k-trees [B90], etc. are examples of such special graphs.

Regarding planar graphs, Weinburg [W66] developed a sequential algorithm for testing isomorphism of triply connected planar graphs in $O(N^2)$ time, where, N stands for the number of vertices in the graphs. An improved algorithm for triply connected planar graphs and an $O(N^2)$ isomorphism algorithm for general planar graphs was given by Hopcroft and Tarjan [HT71]. In [HT72], the same authors again reduced the time complexity to $O(N \log N)$ time. Finally, Hopcroft and Wong were able to obtain a linear time algorithm [HW74] for this problem.

In the hierarchy, next comes the outerplanar graphs, which is a subclass of planar graphs. A planar graph is outerplanar if it can be embedded in a plane so that all its vertices lie on the same face. We call this face as exterior face [H69]. Capitalizing the fact that every biconnected outerplanar graph has a unique hamiltonian cycle [H69]; Colbourn and Booth developed an interesting method for testing isomorphism of outerplanar graphs [CB81]. They noticed that the hamiltonian adjacency sequence (defined in [CB81]) can characterize a graph uniquely (upto isomorphism) instead of a hamiltonian degree sequence. A hamiltonian degree sequence may generate more than one non-isomorphic outerplanar graphs. These hamiltonian adjacency sequences for two graphs G_1 and G_2 can be constructed in linear time and then using a linear time pattern matching algorithm [CLR90] those sequences can be tested for isomorphism of G_1 and G_2.

A graph is maximal outerplanar (MOP) if no edge can be added to the graph without violating the outerplanarity [H69]. Unlike outerplanar graphs, the hamiltonian degree sequence can uniquely (upto isomorphism) characterize a MOP. Using this fact Beyer, Jones and Mitchel [BJM79] obtained a simpler linear time algorithm for MOPs than that in [CB81] as applied to MOPs. Also the algorithm in [BJM79] is more efficient than that in [HW74] when applied to MOPs. In [BJM79], the hamiltonian degree sequences D_1 and D_2 for the two graphs G_1 and G_2 were computed which were used with the fact that G_1 and G_2 are isomorphic iff D_2 is substring

of $D_1D_1\$D_1^RD_1^R$, where D_1^R is the inversion of D_1 and '\$' represents the 'end of string' character. In their first approach of comparing these two strings, they used Morris and Pratt's pattern matching algorithm [MP70]. In the second approach, they found a set of either two or three vertices (termed as central vertices) which can be uniquely identified in a MOP and for comparison of the degree sequences, they start the hamiltonian cycle only at these vertices. An approach, based on number theory, for testing isomorphism of two maximal outerplanar graphs also exists in the literature [C82]. It is known that there is a one to one correspondence between maximal outerplanar graphs and Farey graphs. With this idea, an $O(N^2)$ algorithm for testing isomorphism of maximal outerplanar graphs has been described in [C82].

Regarding parallel algorithms, in [JK88], Ja Ja and Kosaraju parallelized the idea given in [HT73] for isomorphism testing of planar graphs with time complexity of $O(\log^3 N)$ when implemented on a CREW PRAM model with $O(N^2)$ processors and with $O(\sqrt{N})$ time when implemented on a $\sqrt{N} \times \sqrt{N}$ array of processors. Gazit and Reif [GR90] also parallelized the same approach with a randomized algorithm of time complexity $O(\log N)$ using $N^{1.5}\sqrt{\log(N)}$ processors, assuming that the families of separators for both the graphs are given. A deterministic algorithm for planar graphs was also presented in [G91] which requires $O(\log^3 N)$ time using $\frac{N^{1.5}}{\log^2(N)}$ processors when implemented on a CRCW PRAM model.

Parallel algorithms for testing isomorphism of outerplanar graphs also exist in the literature. In [LP89], it is shown that isomorphism problem restricted to 2-connected outerplanar graphs is in the class NC^3 (they can be solved by an algorithm running in $O(\log^3(N))$ time using polynomial number of processors). In [LLP+90], Levcopoulos et al. have shown that isomorphism of trees and outerplanar graphs can be tested in $O(\log N)$ time with $\frac{N}{\log N}$ processors on a CRCW PRAM model and in $O(\log^2 N)$ time with $\frac{N}{\log^2 N}$ processors on an EREW PRAM model.

2.3.2 Numerical Interpolation

Several real time situation may demand fast evaluation of a function at a given value. However, in many cases only the values of the function at some discrete points are known, instead of the analytical form of the function. In such cases, the function can be evaluated at a given point by using different numerical interpolation techniques. Polynomial interpolation is widely used for curve and surface fitting to scattered data. In many applications of the surface fitting technique, the goal is to construct a *contour map* of the unknown function [S76] using the given data. This contour map construction has applications to the oil industries, geological maps, cardiology, etc.

The best known sequential algorithm for polynomial interpolation requires $O(N \log^2 N)$ time [AHU74].

In recent years, a number of parallel algorithms for polynomial interpolation have been proposed in the literature. McKeown [M86] presented a systolic implementation of Aitken's method of iterative interpolation to compute the Lagrange interpolation polynomial. Capello, Gallopoulos and Koc presented several spacetime-optimal systolic arrays for computing process dependence graph corresponding to the Aitken algorithm [CGK90]. By studying these process dependence graphs in detail, the authors derived a method related to that of McKeown for a systolic version of Newton and Hermite interpolation using the algorithm of Aitken and Neville [CB81b]. This method requires $2N-1$ steps on $\lfloor \frac{n}{2} \rfloor$ processors (when N is the number of interpolating points). Each step consisting of two subtractions and one division. When reprogrammed to compute iterated interpolation, each step of this method requires two additional multiplications along with that subtractions and division.

In the paper [SMK91], Schroder et al. described a systolic parallel/distributed algorithm for interpolation and evaluation of polynomials over any field using a linear array of processors. The time complexity of this algorithm was $O(N)$ when implemented on an $1 \times (N+1)$ processing array. This algorithm was based on

Newton's divided difference scheme. An extension of this idea was presented in [MKC92] for rational interpolation based on Thiele's reciprocal differences and continued fraction approximation. The paper also described a systolic algorithm, whose period is $O(N)$ to produce M/M Pade approximation ($M = \lceil \frac{N}{2} \rceil$) using N processors.

A parallel algorithm has been described in [G94] by Goertzel which computes the Lagrange polynomial interpolating N points in $\lfloor \frac{N}{2} \rfloor + O(\log N)$ steps. Each step consists of two subtractions and four multiplications. Unlike the systolic algorithms mentioned above, this method makes use of a processor tree with ring connections. The architecture exploits the inherent redundancy of the Lagrange coefficient in a novel way. Without the tree connection, the method requires $N+1$ steps on N processors which, with respect to the space-time cost, is similar to that obtained in [CGK90]. The additional tree connection reduces the time complexity by approximately half.

2.3.3 Matrix Multiplication and Inversion

Matrix multiplication has tremendous applications in numerous fields. It is useful for numerical problems (e.g., solution of linear system of equations) as well as non-numeric applications (e.g., graph problems). Development of faster algorithms to multiply two matrices has been well studied in the literature. As opposed to the conventional $O(n^3)$ serial algorithm for multiplying two $n \times n$ matrices, Strassen first gave a new serial algorithm which required only $O(n^{2.81})$ time [S69]. Following Strassen's method, other faster serial algorithms also appeared due to Coppersmith and Winograd [CW80], [CW87] and Strassen [S86]. In [CW87], the authors finally reduced the time complexity to $O(n^{2.376})$. A discussion on the development of faster serial algorithms to multiply matrices and their varied applications can be found in [P84].

Quite a few array processors exist which have been designed for hardware implementation of matrix multiplication algorithms. For example, the design of linear

arrays [PT89], [PT91], [RV84] [VR86], the mesh arrays [K88c], [MC87], [TC95], the hexagonal arrays [LW85], the cylindrical array [PA88] and the two-layered (or the multi-layered) mesh array [K88a]. Besides, Benaini-Robert [BR89] and Jagdish-Kailath [JK89] used Winograd algorithm for matrix multiplication and implemented them on their systolic arrays.

Several mesh algorithms have been proposed in the literature, whose running time is $\Omega(n)$. Such algorithms appear in [FK76], [PV80], [U84] and [V76]. The mesh matrix multiplication proposed in [A89] is the fastest achievable on a mesh of processors. Algorithms on multidimensional meshes are described in [NMB83] and [L83].

Numerous algorithms have been proposed in the literature for performing matrix multiplication on the hypercube structure, each having its own characteristic of speed-up factor and memory usage.

A number of other matrix multiplication algorithms for the cube and perfect shuffle interconnection networks are described in [DNS81]. The idea of cube matrix multiplication described in [A89] originated in [DNS81] and its running time is $O(log\ n)$. This running time is the fastest achievable by any parallel algorithm for multiplying two $n \times n$ matrices on the cube. Matrix multiplication algorithms for the cube and other interconnection networks and their applications are proposed in [CS87], [FOH87], [HZ83], [HC82], [HB84], [K80], [MC80], [RV84] and [VR86]. A new parallel computation model, called a permutation network processor was introduced in [LO94] for inner product and other matrix computations. It shows that the matrix multiplication can be computed on this model in $O(1)$ time at the cost of $O(n^3)$ processors for $n \times n$ matrices. The result compares well with the time and cost complexities of other high level parallel computer models such as PRAM and CRCW PRAM. Algorithms which run on a CRCW shared-memory computers can be found in [C76], [HZ83], [S78] and [S80]. A discussion of various implementation issues regarding parallel matrix multiplication algorithms is provided in [CPH+83].

Matrix inversion is one of the important problems in numerical linear algebra. The best known parallel algorithm for computing matrix inversion is due to Csanky [C76a]. The algorithm requires $O(\log^2 n)$ computational steps using $O(n^4)$ processors for inverting a matrix of size $n \times n$. Though this is the fastest known method, its implementation is questionable due to its unstable nature. Another matrix inversion algorithm based on matrix partitioning is due to Pease [P69]. Several systolic algorithms for matrix inversion implementable on VLSI architectures can be found in the literature [R85], [GHV+88], [AD89], [E89], [ED89], [L90], [MMS92], [KLY92]. A parallel algorithm based on Givens plane rotations is described in [E89], [ED89]. The algorithm inverts a dense matrix of order $n \times n$ on a systolic array consisting of $n^2 + n$ processors, in $5n$ units of time. Parallel Gauss-Jordan algorithm suitable for implementation on a pyramidal architecture has been analyzed by Geus et al. [GHV+88]. A parallel algorithm based on Gauss-Jordan elimination method for dense matrix inversion has been presented in [MMS92]. A Strassen-type matrix inversion algorithm is discussed in [BH94].

Chapter 3

Topologies with Multiple Loops

3.1 Introduction

In this chapter, we propose a new family of network topologies, by adding a few extra links over the ring. A topology in this family has nodes with degrees only 2, 3, and 4 (average node degree is less than or equal to 3.25) and has the diameter bounded above by $\lfloor \frac{11m}{8} \rfloor + 1$, where m is a parameter of the graph such that $m \geq 3$, N is an even multiple of m and $(m-1) \times 2^{\lfloor \frac{m-1}{2} \rfloor + 1} < N \leq m \times 2^{\lfloor \frac{m}{2} \rfloor + 1}$. This shows that the diameter of the topology is $O(\log N)$. With respect to the diameter, this topology is thus superior to both the chordal ring and the distributed loop network. The total number of links used in this network is less than that in a distributed loop network and is no more than $1/12^{\text{th}}$ of that in a chordal ring. An algorithm for point to point routing has also been presented. The *Ascend* and *Descend* types of algorithms [PV81] can be efficiently implemented on this topology. Moreover, the successive values of N, for which the proposed topology can be defined, are at an interval of $2m$, which is less than $4 \log_2 N$. That is, if N and N', are two successive values of the total number of nodes with $N' > N$, then $N' - N < 4 \log_2 N$. This may be contrasted with other fixed degree topologies having $O(\log N)$ diameter, e.g., Moebius graph [LS82], de Bruijn graph [B46], cube-connected cycles [PV81], etc., for which the successive values of N are at much larger intervals. For all such graphs, N' is at least $2N$. The proposed network graph is hamiltonian and in case

of a single node failure, the diameter may increase by at most 6.

3.2 Description of the Topology

We describe the topology in terms of a graph G(m, N), having the following characteristics:

a) N is the total number of nodes in the graph. Let the nodes be numbered as 0, 1, $\cdots$ $N-1$.

b) m is a parameter of the graph such that $m \geq 3$.

c) N is an even multiple of m such that $N = 2k \times m$, for some positive integer k.

d) $(m-1) \times 2^{\lfloor \frac{m-1}{2} \rfloor + 1} < N \leq m \times 2^{\lfloor \frac{m}{2} \rfloor + 1}$.

e) The nodes are connected by the following three types of edges (all operations below are treated under modulo N, unless otherwise mentioned) :

1) The node i is connected to the nodes $(i+1)$ and $(i-1)$. Thus the nodes are connected in the form of a cycle. We call these edges as c-edges (cyclic edges).

2) For $0 \leq i \leq 2k-1$, the node $i.m$ is connected to the diametrically opposite node $(i.m + N/2)$. We call these edges as d-edges (diagonal edges).

After introducing d-edges, there are $2k$ number of nodes in the network, which are of degree 3. These degree 3 nodes divide the cycle, formed in (1), into $2k$ parts. We call each of these parts as a sector of length m. The sector j consists of the nodes, $\{j.m,\ j.m+1,\ j.m+2,\ \cdots,\ (j+1)m\}$, $0 \leq j \leq (2k-1)$.

3) Nodes in different sectors are interconnected by a third type of edges, called as hops or h-edges, as described below.
Starting from the node $j.m+1$, in sector j, $0 \leq j \leq 2k-1$, h-edges are introduced at every alternate node of degree 2 (there are $\lfloor \frac{m}{2} \rfloor$ such nodes in every sector). These connections are done by the following ways depending on the value of m.

Let $\lfloor m/2 \rfloor - 1 = r$.

Case 1 : r is even.

i) The node $j.m+i$ is connected to the nodes $[(j.m+i) \pm m \times 2^{r-(i-1)}]$. These h-edges are termed as hops and are denoted by $\pm h_{r-(i-1)}$ respectively, $i = 1,\ 3,\ 5,\ \cdots,\ r+1$.

ii) The node $j.m+i+r$ is connected to the nodes $[(j.m+i+r) \pm m \times 2^{i-2}]$. These edges are likewise termed as $\pm h_{i-2}$ hops, $i = 3, 5, \cdots, r+1$.

Case 2 : r is odd.

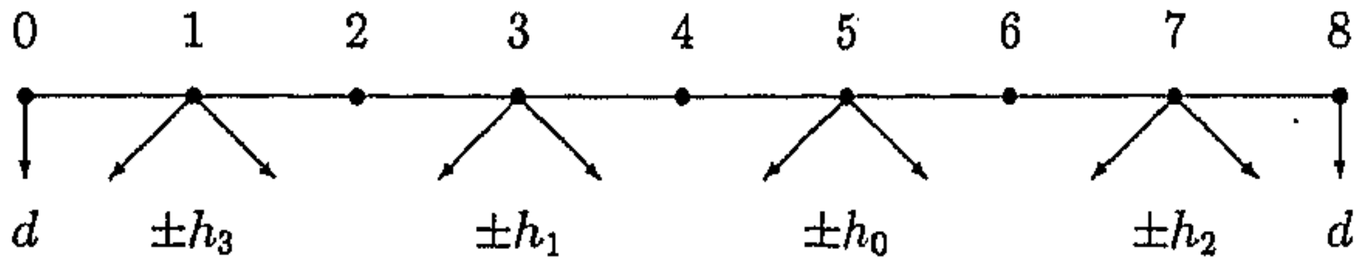
i) The node $j.m+i$ is connected to the nodes $[(j.m+i) \pm m \times 2^{r-(i-1)}]$, by the hops $h_{r-(i-1)}$, $i = 1, 3, 5, \cdots, r$.

ii) By the hops $\pm h_{i-2}$, the node $j.m+i+r$ is connected to the nodes $[(j.m+i+r) \pm m \times 2^{i-2}]$, $i = 2, 4, \cdots, r+1$. □

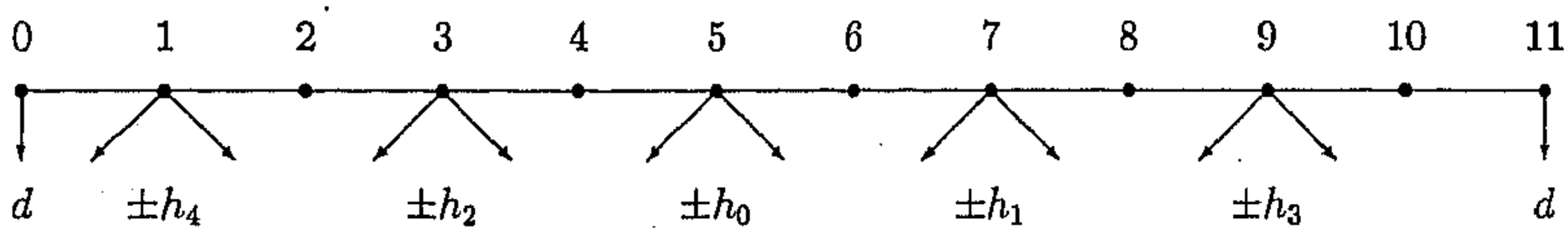
An example of the hop distribution in sector 0 of the proposed topology when $m = 8$ and $m = 11$ is given in Fig. 3.1. Fig. 3.2 shows the complete connection pattern in $G(5, 40)$. It is to be noted here that the largest hop originated from a sector, is $h_{\lfloor m/2 \rfloor - 1}$. A hop h_i will be referred to as *even (odd)* if i is even (odd). The connection pattern shows that even and odd hops originate from different halves of a sector. The two hops connecting a vertex v to $v + m.2^l$ and $v - m.2^l$ will be referred to as $+h_l$ and $-h_l$ respectively.

Average node degree of this graph is $3 + 1/m$, when m is even and exactly 3 when m is odd. It shows that the average node degree approaches to 3 for large N. The total number of edges in the graph is thus asymptotically $\sim 1.5N$.

Remark : If all the nodes in each sector (that is, the nodes jm, $jm+1$, $\cdots$, $(j+1)m-1$ in sector j, $\forall\ j$, $0 \leq j \leq \frac{N}{m}$) of $G(m, N)$ are coalesced to form a single node representing the whole sector then the resultant graph will be a supergraph of the circulant $G(\frac{N}{m}, \pm 1, \pm 2, \pm 2^2, \pm 2^3, \cdots, 2^{\lfloor \log \frac{N}{m} \rfloor - 1})$ [BW84], [D79] (or recursive circulant G(N, d) for d = 2 [PC94]). Only in a special case, when $N = m.2^{\lfloor \frac{m}{2} \rfloor + 1}$ the graph obtained by coalescing the nodes in each sector of $G(m, N)$



(a) For $m = 8$



(b) For $m = 11$

Figure 3.1: Hop distribution in sector 0

will be identical to $G(\frac{N}{m}, \pm 1, \pm 2, \pm 2^2, \pm 2^3, \cdots, 2^{\log \frac{N}{m}-1})$. However, because of the typical degree distribution of the nodes in a sector, the extension of the results regarding diameter, routing, etc. of circulant graphs [BW84] or recursive circulant graphs [PC94] can not be directly extended to $G(m, N)$ and then need a separate treatment as given in the following sections. Moreover, the node degree (approximately equal to $\log_2 N$) in the recursive circulant graph increases with the total number of nodes N. In the proposed topology the average node degree is approximately equal to 3 and the maximum node degree is 4. In this regard the proposed topology is more cost effective.

3.3 Diameter of the Network

In this section, we find an upper bound on the diameter D of the graph $G(m, N)$. To start with, we discuss about a restricted redundant binary number system to represent a node of the graph.

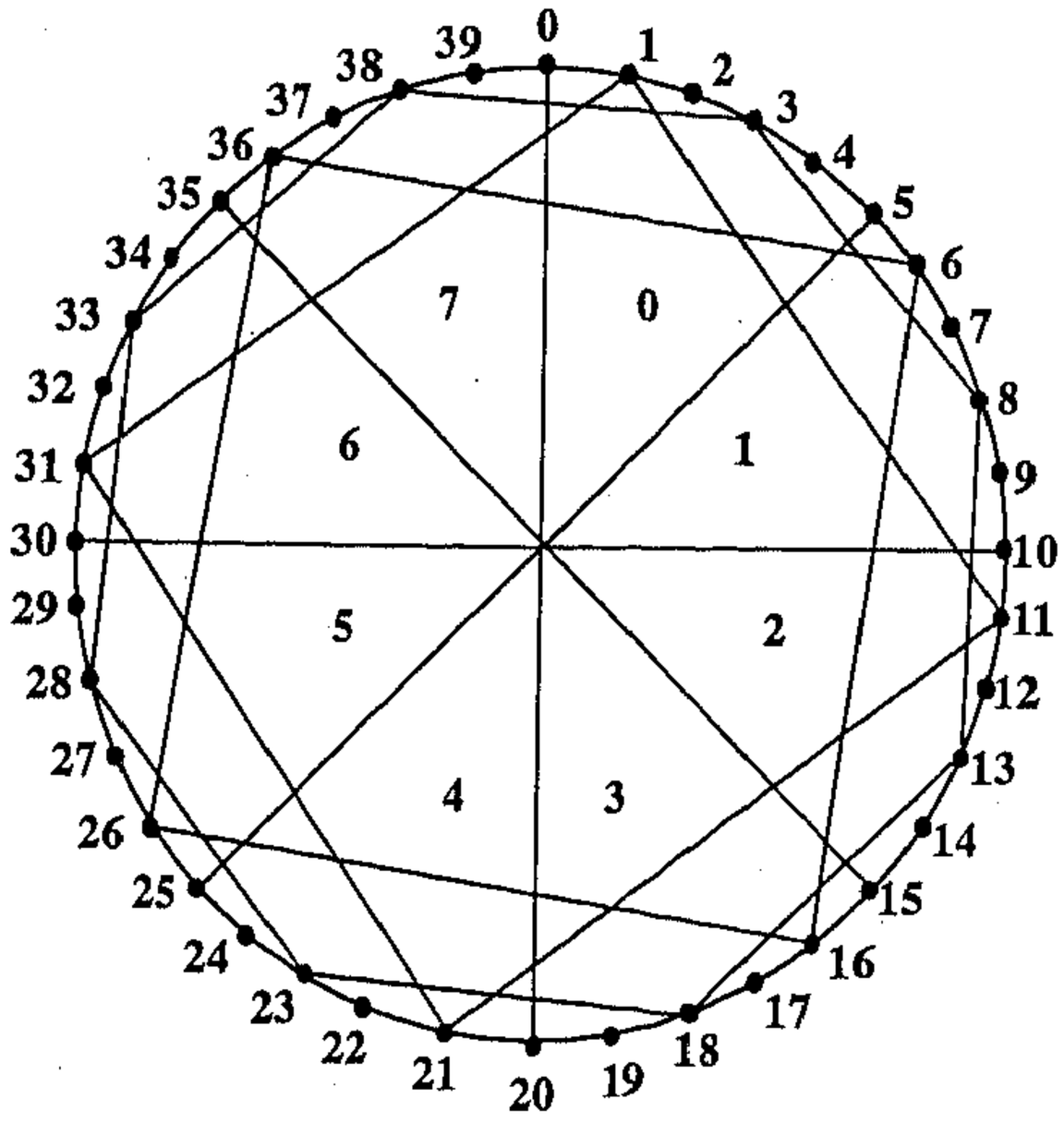


Figure 3.2: The proposed graph $G(5, 40)$

3.3.1 Restricted Redundant Binary Representation

Definition 3.1 *A Redundant Binary representation* $k_{r-1}\ k_{r-2} \cdots k_0$ *of a number* K *is one in which each digit* k_i, $0 \leq i \leq r-1$, *is an element of* $\{0, 1, \bar{1}\}$ *and* $K = \sum_{i=0}^{r-1} 2^i k_i$.

Naturally, K does not have a unique representation in redundant binary.

Example 3.1 *The binary number 0111011 has the equivalent representations* $100\bar{1}10\bar{1}$ *and* $1000\bar{1}0\bar{1}$ *in redundant binary.*

Definition 3.2 *A redundant binary representation, in which there is at least one zero bit in between two non-zero bits, will be termed as a Restricted Redundant Binary (RRB) representation.*

Example 3.2 *In 3.1, $1000\bar{1}0\bar{1}$ is the RRB representation of the binary string 0111011.*

Remark : This RRB representation is similar to the *canonical signed digit (CSD)* representation [AW93] of a positive integer for the radix-2. It follows from the results in [R60] that the CSD (as well as RRB) representation of a number can be computed sequentially by scanning the binary representation of the number from right to left. Thus, the RRB representation of a number N can be computed sequentially in $O(\log N)$ time. How this conversion can be done by scanning the binary string from right to left is shown in the following example.

Example 3.3 *The string 111101110101 1 can be converted into the equivalent RRB representation by the following steps :*
$111101110101 \rightarrow 11110111010\bar{1} \rightarrow 1111011101\bar{1}0\bar{1} \rightarrow 111101\bar{1}0\bar{1}0\bar{1} \rightarrow 10000\bar{1}00\bar{1}0\bar{1}0\bar{1}$.

We state the following results.

Lemma 3.1 *In the RRB representation of a number K, the number of non-zero bits can be at most $\lceil \frac{\lceil \log K \rceil + 1}{2} \rceil$.*

Proof : *case 1*: When K is not a power of 2.
Binary representation of K requires $\lceil \log K \rceil$ number of bits. In RRB representation, the number of bits require to represent K may be one more than that, i.e., $\lceil \log K \rceil + 1$. Since in between two consecutive non-zero bits there should be at least one zero bit in RRB representation, the maximum number of non-zero bits in this representation of K will be $\lceil \frac{\lceil \log K \rceil + 1}{2} \rceil$.
case - 2: When K is a power of 2, RRB representation of K contains exactly one non-zero bit.
Thus in RRB representation of any number K, the number of non-zero bits can be at most $\lceil \frac{\lceil \log K \rceil + 1}{2} \rceil$. □

Lemma 3.2 *The RRB representation of a number is unique.*

Proof : It is clear that $(00\cdots0)$ is the unique representation of 0.
Now, if possible let $a = (a_k a_{k-1} \cdots a_1 a_0)$ and $b = (b_k b_{k-1} \cdots b_1 b_0)$ be two distinct RRB representation of a number, say x. Then $a - b$ must be 0.
Since a and b are distinct representations, they must differ by at least one bit position. Let $a_i \neq b_i$.
If one of them is zero, then the i^{th} bit of a-b will be non zero and since in RRB representation every non-zero bit is padded by at least one zero bit from both the sides, no carry bit can make this i^{th} bit zero in the final representation of $(a - b)$.
When both of them are non-zero, without loss of generality, let us suppose $a_i = 1$ and $b_i = -1$. In that case, the i^{th} bit of $a - b$ will be 0 and the carry 1 will be propagated to the next bit. Now, as we are considering RRB representation, both a_{i+1} and b_{i+1} will be 0. Thus the carry of the i^{th} bit will make this $(i+1)^{th}$ bit of $(a - b)$ non-zero.
Thus we have seen that $a - b$ can not be equal to 0, which contradicts our assumption. Therefore RRB representation of a number is unique. □

Lemma 3.3 *In the RRB representation, the largest number M that can be obtained using b number of bits is given by :*

$$M = \begin{cases} \frac{2}{3}(2^b - 1), & \text{if } b \text{ is even;} \\ \\ \frac{2}{3}(2^b - 1) + \frac{1}{3}, & \text{otherwise.} \end{cases}$$

Proof : It is clear that for the largest number, every alternate bit starting from the most significant bit will be 1.
If b is even, then

$$M = 2^{b-1} + 2^{b-3} + \cdots + 2 = \frac{2}{3}(2^b - 1)$$

Otherwise,

$$M = 2^{b-1} + 2^{b-3} + \cdots + 2^0 = \frac{2}{3}(2^b - 1) + \frac{1}{3}$$

□

Let w_x be the total number non-zero bits in the RRB representation of a number x. Further suppose that w_x^e and w_x^o be the number of non-zero bits corresponding to the even and odd powers ('0' is excluded) of 2 respectively, in the RRB representation of x.

Example 3.4 *For* $x = 100\bar{1}0\bar{1}01$, $w_x = 4$, $w_x^e = 2$ *and* $w_x^o = 1$; *for* $x = 100\bar{1}0\bar{1}00$, $w_x = 3$, $w_x^e = 2$ *and* $w_x^o = 1$.

It is clear that when x is odd, $w_x = w_x^e + w_x^o + 1$, otherwise $w_x = w_x^e + w_x^o$.

3.3.2 Upper Bound on the Diameter

To find the diameter it is enough to consider the paths from a node s in sector 0, to a node $d \leq \frac{N}{2}$, since all sectors look alike.

Suppose, d belongs to the sector δ, $0 \leq \delta \leq \lfloor \frac{N}{2m} \rfloor$. Starting from s, we use hops to move across different sectors. But to avail a hop we may need to traverse some cyclic edges as well. With a view of minimizing the total walk along the outer cycle (i.e., along the c-edges) we would take either odd hops or even hops (depending on the value of s and d) which emanate from only one half of every sector. h_0 may be included in both the cases. Since accessing h_l is equivalent to accessing h_{l-1} twice, $\lfloor \frac{m}{2} \rfloor - 1 \geq l \geq 1$, it is always possible to get such a collection of hops. The method how we select the hops to reach sector δ starting from sector 0 is discussed first and then the specific order in which these hops are to be accessed to reduce the total walk along the cyclic edges is found.

First we find the RRB representation of the sector difference δ (since the source is 0 here). The set of hops corresponding to the non-zero bits of this representation will be sufficient to cover these δ sectors. The selection process is thus if the j^{th} bit of the RRB representation is 1 or $\bar{1}$ then the path includes a hop h_j or $-h_j$ respectively, $1 \leq j \leq \lfloor \frac{m}{2} \rfloor - 1$. For example, if we assume that the value of δ to be 13, the RRB representation of δ will be $10\bar{1}01$ and accordingly the set of hops

that can be used to reach the destination sector will be $\{h_4, -h_2, h_0\}$. It can be verified that the hops present in the network would support this selection process.

Without loss of generality, let us assume that $w_\delta^e \geq w_\delta^o$. As we have already mentioned that for minimizing the total path length we should restrict ourselves to use either even or odd hops only (h_0 may be included in both the cases), it is wise to convert odd hops into even hops when $w_\delta^e \geq w_\delta^o$. In that case, instead of using $w_\delta^e + w_\delta^o$ number of hops we shall use $w_\delta^e + 2w_\delta^o$ number of hops along with some $\pm h_0$. Now, let us try to estimate a bound on the total number of hops required after conversion.

Case 1 : $0 \leq d \leq \frac{N}{4}$, i.e., $0 \leq \delta \leq \lfloor \frac{N}{4m} \rfloor$.

As $0 \leq \delta \leq \lfloor \frac{N}{4m} \rfloor$, the number of bits reqiured for RRB representation of δ is $\lfloor \frac{m}{2} \rfloor$. Let us consider the following two subcases separately.

subcase 1a. When δ is odd.

For odd δ, the least significant bit (lsb) in RRB representation of δ is non-zero and hence the second lsb is 0. Among the remaining $\lfloor \frac{m}{2} \rfloor - 2$ bits the number of non-zero bits can be at most $\lceil \frac{\lfloor \frac{m}{2} \rfloor - 2}{2} \rceil = \lfloor \frac{m-2}{4} \rfloor$ (by lemma 3.1. Therefore, $w_\delta^o + w_\delta^e = \lfloor \frac{m-2}{4} \rfloor$.

Now, according to the RRB representation of δ, w_δ is the total number of hops required to reach sector δ. Since the lsb of the RRB representation is 1, $w_\delta = w_\delta^e + w_\delta^o + 1$. Thus, after conversion we need at most $2w_\delta^o + w_\delta^e + 1$ number of hops.

Now, $2w_\delta^o + w_\delta^e + 1 \leq \lfloor \frac{3}{2}(w_\delta^o + w_\delta^e) \rfloor + 1 \leq \lfloor \frac{3}{2} \lfloor \frac{m-2}{4} \rfloor + 1$.

Thus the maximum number of hops required to reach sector δ (using either even or odd hops along with some $\pm h_0$ hops) is at most $\lfloor \frac{3}{2} \lfloor \frac{m-2}{4} \rfloor + 1$.

Subcase 1b : When δ is even.

For even δ, the lsb in the RRB representation of δ will be 0. In the remaining $\lfloor \frac{m}{2} \rfloor - 1$ bits, the number of non-zero bits can be at most $\lceil \frac{\lfloor \frac{m}{2} \rfloor - 1}{2} \rceil = \lfloor \frac{m}{4} \rfloor$.

Here, we have $w_\delta^e + w_\delta^o = w_\delta$ and $w_\delta \leq \lfloor \frac{m}{4} \rfloor$. Thus after conversion of the odd hops

into even hops, the total number of hops can be at most $\lfloor \frac{3}{2}(\lfloor \frac{m}{4} \rfloor) \rfloor$.

Hence, the maximum number of hops required to reach sector δ, under the restriction that either even or odd hops (h_0 may be included in both the cases) can be used, is

$$B_1 = \begin{cases} \lfloor \frac{3}{2}(\lfloor \frac{m}{4} \rfloor) \rfloor & \text{when } m \text{ mod } 8 \text{ is } 0 \text{ or } 1; \\ \lfloor \frac{3}{2} \lfloor \frac{m-2}{4} \rfloor \rfloor + 1, & \text{otherwise} \end{cases}$$

Case 2 : $\lfloor \frac{N}{4m} \rfloor \leq \delta \leq \lfloor \frac{N}{2m} \rfloor$.

To reach this part of our network, we may utilize diagonal edges. But if we include a diagonal edge in our path, we would not use any $\pm h_{\lfloor \frac{m}{2} \rfloor - 1}$.

From sector 0, if we use a d-edge we will reach the boundary of the sectors $\lfloor \frac{N}{2m} \rfloor$ and $\lfloor \frac{N}{2m} \rfloor - 1$. From there we have to cover a distance of j sectors to reach sector δ, where $j = \lfloor \frac{N}{2m} \rfloor - \delta$. To traverse this distance we would not use any $\pm h_{\lfloor \frac{m}{2} \rfloor - 1}$ hop. Here, by Lemma 3.3 we have, $0 \leq j \leq \frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor - 1} - 1) + \frac{1}{3}$, which implies, $\lfloor \frac{N}{2m} \rfloor - \frac{1}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1) \leq \delta \leq \lfloor \frac{N}{2m} \rfloor$. For such a δ, the maximum number of hops required to reach the destination sector can be enumerated as follows :

In this range, j can be represented in RRB using a maximum of $(\lfloor \frac{m}{2} \rfloor - 1)$ bits. As discussed in subcase 1a, the value of $w_\delta^o + w_\delta^e$ can be at most $\lceil \frac{\lfloor \frac{m}{2} \rfloor - 3}{2} \rceil = \lfloor \frac{m}{4} \rfloor - 1$, when j is odd. Hence, after conversion of the hops into either odd or even type, at most $\lfloor \frac{3}{2}(\lfloor \frac{m}{4} \rfloor - 1) \rfloor + 1$ number of hops can be required to reach the destination sector. Including the diagonal edge, the number of hops will be one more than that.

When j is even, following the same arguments given in subcase 1b, the number of hops (including the diagonal edge) can be at most $\lfloor \frac{3}{2}(\lfloor \frac{m-2}{4} \rfloor) \rfloor + 1$.

If B_2 represents the upper bound on the number of hops required to reach the destination sector δ, when $\lfloor \frac{N}{2m} \rfloor - \frac{1}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1) - 1 \leq \delta \leq \lfloor \frac{N}{2m} \rfloor$, then

$$B_2 = \begin{cases} \lfloor \frac{3}{2}(\lfloor \frac{m-2}{4} \rfloor) \rfloor + 1 & \text{when } m \text{ mod } 8 \text{ is } 2 \text{ or } 3; \\ \lfloor \frac{3}{2}(\lfloor \frac{m}{4} \rfloor - 1) \rfloor + 2 & \text{Otherwise} \end{cases}$$

Now, we are left with the case when $\lfloor \frac{N}{4m} \rfloor \leq \delta \leq \lfloor \frac{N}{2m} \rfloor - \frac{1}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1) - 1$. Using the upper bound on $\lfloor \frac{N}{2m} \rfloor$, the above range is redefined as $\lfloor \frac{N}{4m} \rfloor \leq \delta \leq \frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 2)$.

By Lemma 3.3, the farthest sector (from sector 0), whose RRB representation contains $\lfloor \frac{m}{2} \rfloor$ bits is $\frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1)$. This shows that in the range when $0 \leq \delta \leq \frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1)$, we need not use d-edges. Only hops are sufficient here, as in case 1. The range $\lfloor \frac{N}{4m} \rfloor \leq \delta \leq \lfloor \frac{N}{2m} \rfloor - \frac{1}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1) - 1$ is totally contained in the range $0 \leq \delta \leq \frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1)$. Thus in this case, the number of hops required to reach the destination sector δ is same as the bound obtained in case 1.

Hence for $0 \leq \delta \leq \lfloor \frac{N}{2m} \rfloor$, the number of hops required to reach sector δ, is at most max (B_1, B_2). It can be easily verified that max(B_1, B_2) is always $B_2 \; \forall \, m$. □

So far we have identified the appropriate hops required to reach the destination sector and have also enumerated a bound on the number of hops. Now, we would find a specific order in which these hops are to be availed so that the walk along the cycle will be small enough to minimize the total path length. Let us introduce the following notations first.

1) x_j denotes the node in the sector j, from which hops of type 0 are originated. Thus x_j divides the sector j into two halves. Even and odd hops are emanated from these two halves.

2) By $[a, b]$ we mean the set of the nodes $a,\ a+1,\ a+2,\ \cdots,\ b$.

3) Let H_δ be the set of hops of either even or odd types, required to reach sector δ. Suppose $|H_\delta| = p$. Let $u_j^1,\ u_j^2,\ \cdots,\ u_j^p$ be the nodes in sector j such that the hops in H_δ originate from these nodes and $u_j^1 < u_j^2 < \cdots < u_j^p$. Again, we call $[u_j^1, u_j^p]$ as the range of H_δ in sector j.
With this notation, it is clear that $w_\delta^e + w_\delta^o \leq \lceil \frac{u_i^p - u_i^1 + 1}{2} \rceil$. Thus, the number of hops required to reach sector δ would be at most $\lfloor \frac{3}{2} \lceil \frac{u_i^p - u_i^1 + 1}{2} \rceil \rfloor \leq \frac{3}{4} u_0^p - \frac{3}{4} u_0^1 + \frac{5}{2}$.

4) Let the *projection* of d onto sector j be a node $\bar{d}_j$ in sector j, which is related to d by the equation $d = (\delta - j) \times m + \bar{d}_j$

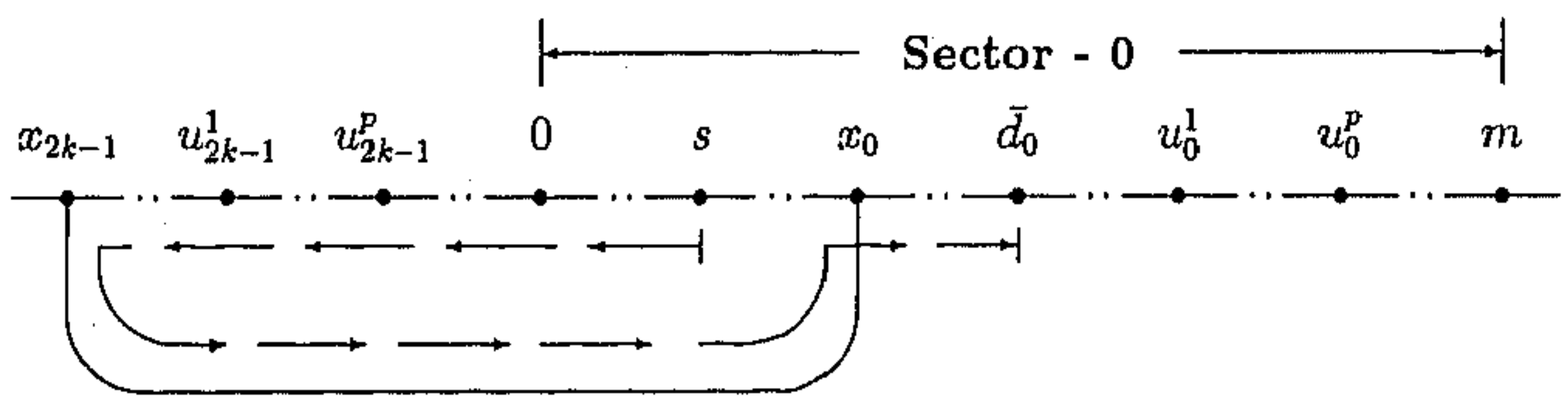
It is clear that to use the hops in H_δ (some of which may be repeated in the actual path), $\forall r, 1 \leq r \leq p$, we must traverse through u_j^r at least once for some sector j, $0 \leq j \leq 2k - 1$. Now, counting the number of c-edges involved in this

traversal can be equivalently mapped to the problem of counting the number of c-edges required to reach $\bar{d}_0$ starting from s, with the restriction that any one of $\{u_0^r,\ u_{2k-1}^r\}$ is traversed at least once. We will now show that by appropriately choosing the order of the hops, this number can always be made less than or equal to m, by using *at most* one additional hop h_0.

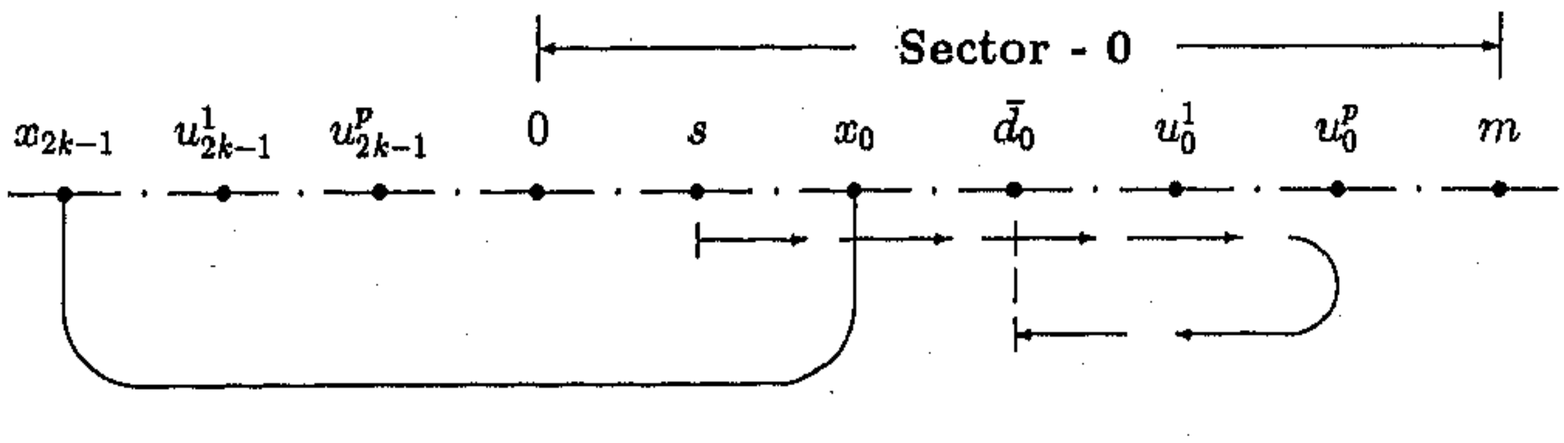
Without loss of generality let us assume that $s \leq \bar{d}_0$.

Case 1 : s and $\bar{d}_0$ are in the different halves of the sector 0. Let us consider the following two subcases separately :

Subcase 1a : $[\ s, \bar{d}_0\ ] \cap [u_0^1, u_0^p] = \phi$.
In this case either $[u_0^1, u_0^p] \subseteq [\bar{d}_0 + 1, m]$ or $[u_0^1, u_0^p] \subseteq [0, s-1]$.
Suppose, $[u_0^1, u_0^p] \subseteq [\bar{d}_0 + 1, m]$. Consider the situation in Fig. 3.3(a).



(a) Traversal 1



(b) Traversal 2

Figure 3.3: Traversal 1 and Traversal 2

For $s + \bar{d}_0 < m$, use of the hops in H_δ requires the traversal from s to x_{2k-1} using

c-edges, from x_{2k-1} to x_0 using an additional hop h_0 and then from x_0 to $\bar{d}_0$ using c-edges. Hence, the number of edges other than those in H_6 is $s+\lfloor\frac{m}{2}\rfloor+1+\bar{d}_0-\lfloor\frac{m}{2}\rfloor$ $= s + \bar{d}_0 + 1 \leq m$. We call this traversal as *Traversal-1*.

For $s + \bar{d}_0 \geq m$, the required traversal is from s to u_0^p and from u_0^p to $\bar{d}_0$, using c-edges only, as shown in Fig. 3.3(b). Thus the number of edges other than those in H_6, is

$$u_0^p - s + u_0^p - \bar{d}_0 \leq 2m - (s + \bar{d}_0) \leq m$$

We call this traversal as *Traversal-2*.

For $[u_0^1, u_0^p] \subseteq [0, s-1]$, it can similarly be shown that the number of c-edges can be at most m.

Subcase 1b : $[\ s, \bar{d}_0\ ] \cap [u_0^1, u_0^p] \neq \phi$. If $[u_0^1, u_0^p]$ is totally contained in $[\ s, \bar{d}_0\ ]$, then the required path is direct from s to $\bar{d}_0$, using c-edges and the path length is $\bar{d}_0 - s$. Otherwise, as shown in Fig. 3.4, it can be tackled in a similar way as we have done in subcase 1a. In both the situations the number of edges other than those in H_6, will be at most m.

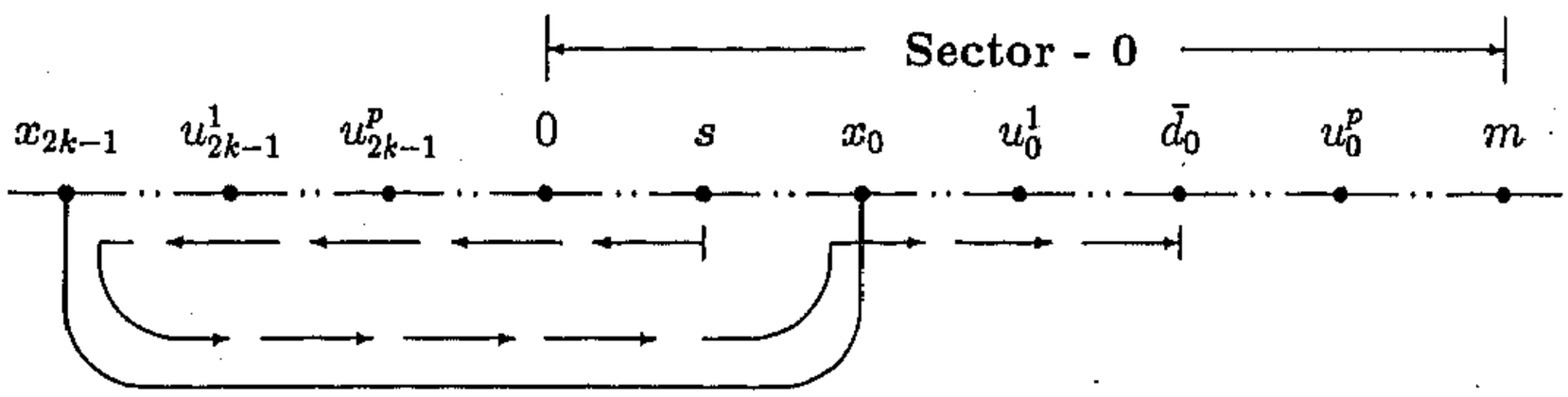


Figure 3.4: Traversal 1 for Case 2

Case 2 : s and $\bar{d}_0$ are in the same half of the sector.

Let us consider the situation when $[u_0^1, u_0^p]$ is also in the same half and $[\ s, \bar{d}_0\ ] \subseteq$ $[u_0^1, u_0^p]$, as shown in Fig. 3.5. Here, the traversal along the c-edges is from s to u_0^1, from u_0^1 to u_0^p and finally from u_0^p to $\bar{d}_0$. The total number of c-edges is thus equal

to $(s - u_0^1) + (u_0^p - u_0^1) + (u_0^p - \bar{d}_0) = 2u_0^p - 2u_0^1 - (\bar{d}_0 - s) \leq m - (\bar{d}_0 - s) \leq m$. Whenever, $[u_0^1, u_0^p]$ and $[\, s, \bar{d}_0\,]$ are in the same half of the sector, the traversal will be same as above.

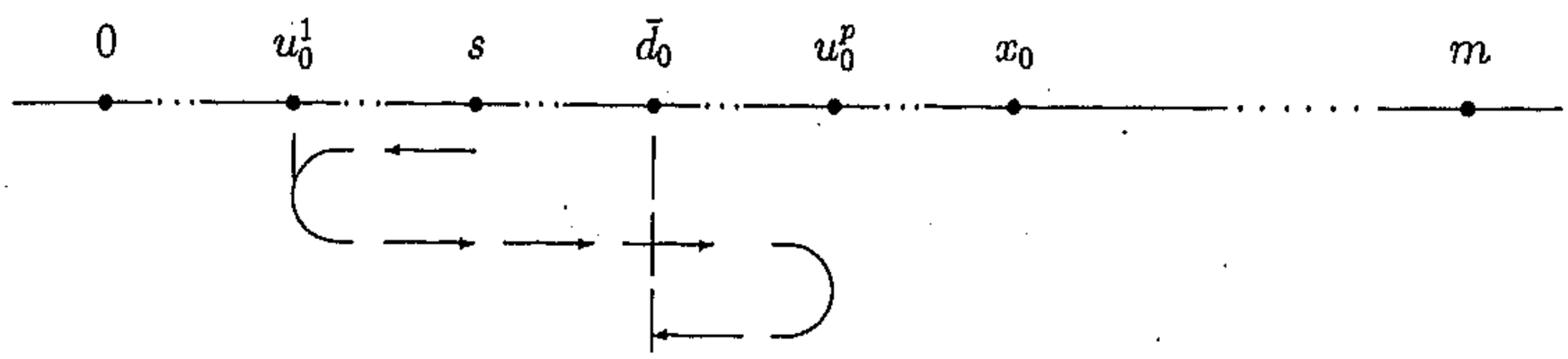


Figure 3.5: Traversal for Case 2

All other cases can be treated in a similar way as discussed in the above two cases.

Combining all the above results, the length $\hat{L}$, of the longest path between any two nodes is given by

$$\hat{L} = \begin{cases} \lfloor \frac{3}{2}(\lfloor \frac{m-2}{4} \rfloor) \rfloor + 1 + m, & \text{when } m \bmod 8 = 2, 3; \\ \\ \lfloor \frac{3}{2}(\lfloor \frac{m}{4} \rfloor - 1) \rfloor + 2 + m, & \text{otherwise.} \end{cases}$$

A little algebraic manipulation yields the following result.

Theorem 3.1 *The diameter D of the graph $G(m, N)$ is given by $D \leq \hat{L} \leq \lfloor \frac{11m}{8} \rfloor + 1$ if m mod 8 is 2, 4 or 5. Otherwise, $D \leq \lfloor \frac{11m}{8} \rfloor$.*

Remark : When s and $\bar{d}_0$ are in the same half of a sector, a path of smaller length, other than that we have established above, can be found between s and d. In this case we will use both even and odd hops corresponding to the RRB representation of δ, except those whose origins correspond to the points in $[s + 1, \bar{d}_0 - 1]$. We will replace each of these hops in $[s + 1, \bar{d}_0 - 1]$ by exactly two hops of smaller length. The total number of hops required is thus at most $\lceil \frac{\lfloor \frac{m}{2} \rfloor}{2} \rceil + \lfloor \frac{\bar{d}_0 - s}{2} \rfloor$. The

corresponding traversal from s to $\bar{d_0}$ is from s to x_{2k-1} using c-edges, from x_{2k-1} to x_0 using h_0 and finally from x_0 to $\bar{d_0}$ using c-edges. Therefore the total path length will be

$$\begin{aligned}
& \lceil \tfrac{\lfloor \frac{m}{2} \rfloor}{2} \rceil + \lfloor \tfrac{\bar{d_0}-s}{2} \rfloor + m - (\bar{d_0} - s) + 1 \\
\leq \quad & \lceil \tfrac{\lfloor \frac{m}{2} \rfloor}{2} \rceil + m - \tfrac{\bar{d_0}-s}{2} + 1 \\
\leq \quad & \lceil \tfrac{\lfloor \frac{m}{2} \rfloor}{2} \rceil + m + 1 \\
\leq \quad & \lfloor \tfrac{5m}{4} \rfloor + 2 \\
\leq \quad & \lfloor \tfrac{11m}{2} \rfloor, \text{ when } m \geq 13
\end{aligned}$$

We call this traversal as *Traversal-3*. Thus we have seen that when s and $\bar{d_0}$ are in the same half of the sector, the path length is upper bounded by $\lfloor \frac{5m}{4} \rfloor + 2$, when $m \geq 13$. Otherwise (when $m < 13$), the length of these paths may exceed the diameter of the topology by at most 1.

3.4 Routing Algorithm

Input : 1) the source node s and the destination node d
2) the two parameters m and N of the topology

Output : A path from s to d

Procedure RRB($sector - diff$, E, O)

/* E and O are two linear arrays of maximum size $\lfloor \frac{m}{4} \rfloor$. These arrays are used to store even and odd bits of the RRB representation separately. 'sector-diff' is the difference between the sector numbers of d and s */

Step 1 : Obtain the RRB representation of sector-diff.
Let it be $(b_L\ b_{L-1} \cdots b_2\ b_1\ b_0)$

Step 2 : **for** $i = 1$ to $\lfloor \frac{m}{4} \rfloor$ **do**
begin

$E[i] = b_{2i}$;

$O[i] = b_{2i-1}$;

end;

$b \leftarrow b_0$;

Example 3.5 *For the source-destination pair (2, 42), the sector-diff is 5 and the values of E and O as computed by the procedure RRB are as follows.* $E[1] = 1$, $E[2] = 0$, $O[1] = 0$, $O[2] = 0$ *and* $b = 1$.

Procedure Hops (s, d, E, O, b, $diag - edge$)

/* Obtain hops in a specific order, in which they will be used to reach d, starting from s and store that order (along with the cyclic edges) in a final array. '$diag - edge$' is a boolean variable which indicates whether a diagonal edge is to be included in the path or not. The final array will be constructed on the basis of E, O, b and $diag - edge$. */

Step 1 : Obtain $w^e_{sector-diff}$ and $w^o_{sector-diff}$.

Step 2 : **if** $(w^e_{sector-diff} > w^o_{sector-diff})$ **then**
convert all the odd hops to even hops and modify the elements of E and O in the way as described in Section 3.3.2
else convert all the even hops to odd hops and modify the elements of E and O accordingly.

Step 3 : /* To find the order in which the hops are to be used */
if $(\bar{s_0} + \bar{d_0} < m)$ **then** use the traversal-1
else use the traversal-2 (as discussed in Section 3.3.2)

Example 3.6 *For the problem addressed in Example 3.5, the values computed by the procedure Hops are* $w^e_5 = 1$, $w^o_5 = 0$.
The order in which the edges will be accessed are as follows :

$-c,\ -c,\ -c,\ h_2,\ -c,\ -c,\ h_0,\ h_0,\ -c,\ -c,\ -c$

Here, $-c$ denotes a cyclic edge in counter-clockwise direction.

Procedure Find-Path (s, d, $sector - diff$)

/* This procedure finds the hops which are to be included in the path and the order H in which they are to be used to reach d starting from s. */

Step 1 : $range \leftarrow \lceil \frac{2}{3}(2^{\lfloor \frac{m}{2} \rfloor} - 1) \rceil$;

if $(0 < sector - diff \leq range)$ **then**

begin

$diag - edge \leftarrow$ 'false';

RRB($sector - diff$, E, O);

end;

else

begin

$diag - edge \leftarrow$ 'true';

/* to indicate whether a diagonal edge is to be included */

$sector - diff \leftarrow -(\frac{N}{2m} - 1 - sector - diff)$;

RRB($sector - diff$, E, O);

end;

Step 2 : $\bar{s}_0 \leftarrow s - \lfloor \frac{s}{m} \rfloor \times m$;

$\bar{d}_0 \leftarrow d - \lfloor \frac{d}{m} \rfloor \times m$;

Step 3 : /* assume that $\bar{s}_0 \leq \bar{d}_0$ */

Hops (s, d, E, O, b, $diag - edge$);

/* Main Program */

begin

$distance \leftarrow (d - s + N) \bmod N$;

$sector - diff \leftarrow (\lfloor \frac{d}{m} \rfloor - \lfloor \frac{s}{m} \rfloor + \frac{N}{m}) \bmod \frac{N}{m}$

if $distance \leq m$ **then**

walk along the c-edges from s to d;

```
else
  if   sector − diff ≤  N/2m − 1 then
  Find-Path (s, d, sector − diff)
  else
    begin
      distance  ←  N - distance;
      if   distance ≤  m   then
      walk along the c-edges from s to d;
      else
        begin
          Find-Path (s, d, N/2m − sector − diff);
          Keeping the order same, change the sign of all the hops obtained;
        end;
    end;
end.
```

Example 3.7 *According to the above algorithm, the following two paths are computed for the source destination pair (2, 42) and (13, 81) in* $G(8,256)$.
Path from 2 to 42 : $2 \rightarrow 1 \rightarrow 0 \rightarrow 255 \rightarrow 31 \rightarrow 30 \rightarrow 29 \rightarrow 37 \rightarrow 45 \rightarrow 44 \rightarrow 43 \rightarrow 42$.
Path from 13 to 81 : $13 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 17 \rightarrow 81$.

Analysis : It can be verified that execution of the above algorithm requires $O(\log N)$ steps. The path is computed once for all at the source node in terms of the hops taken in order and attached to the transmitted message. As there are m different types of hops (including both positive and negative), $\log m$ bits are needed to designate each hop. Hence, the total number of bits required to specify a path is $O(m \log m) \simeq O(\log N \log\log N)$.

3.5 Fault Diameter

It is clear that the graph is biconnected. To find the diameter in presence of a single faulty node, we proceed as follows.

Let P be the path between two nodes s and d, obtained by the method discussed in Section 3.3.2 with the path length $L(P) \leq \lfloor \frac{11m}{8} \rfloor + 1$. In presence of a single faulty node, if the faulty node f lies on the path P, then there may be two possible cases : (A) the faulty node lies on a sector other than the sector 0 and the destination sector, (B) the faulty node is in the destination sector. The case for the faulty node lying in sector 0 can be treated in the same way as the case (B).

Case A :

Let h_q and h_r $(q < r)$ be two consecutive hops to be accessed in the path P and suppose while traversing along P, we enter the sector j by using the hop h_q and leave that sector by using the hop h_r. Let v_q and v_r be the respective nodes in sector j from where h_q and h_r originate.

The fault would affect the path if any one of the following occurs :

Case 1 : The node v_r itself is faulty.

Case 2 : The node v_r is a live node but the hop h_r would take us to such a node which is faulty.

Case 3 : An intermediate node within the range $[v_q, v_r]$ is faulty.

We do not consider the case when the node v_q is faulty. Because in that case, the situation will be identical to the case (2) above for the sector from which we enter sector j. We consider below each of these cases separately.

Case 1 : Instead of using a hop h_r, we can use four hops of type $(r-2)$. Thus, in this case the path length would be increased by three. Therefore, the path length would be at most $L(P) + 3 \leq \hat{L} + 3$.

Case 2 : In this situation, the only restriction is on the hop h_r, originated from v_r in sector j. There is no difficulty in using the hop $-h_r$ from v_r. Now, $2^r = -3 \times 2^r + 2^{r+2}$. Instead of using a hop h_r from sector j, we can use $-h_r$ three

times consecutively, starting from sector j, and then h_{r+2} once. Thus in P, the hop h_r is actually replaced by a sequence of four hops, namely, $\{-h_r, -h_r, -h_r, h_{r+2}\}$. Moreover, if h_r is to be used twice in P, then it will be replaced by six $-h_r$ hops followed by two h_{r+2} hops. Thus, the path length would be at most $L(P) + 6 \leq \hat{L} + 6$.

Case 3 : We consider two subcases.

Subcase 3a : The faulty node is of degree 2.

Let v_l and v_{l+2} be the two neighbors of the faulty node f, and $\pm h_l$ and $\pm h_{l+2}$ be the hops originated from these two nodes respectively. To bypass f, a sequence of five hops, namely, $\{-h_l, -h_l, -h_l, -h_l, h_{l+2}\}$ will be accessed in between h_q and h_r. Thus the path length would be at most $L(P) + 5 \leq \hat{L} + 5$.

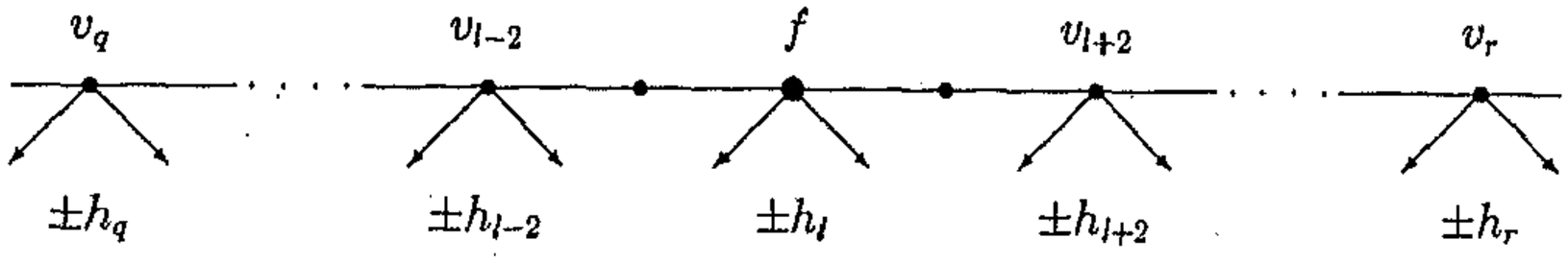


Figure 3.6: Faulty node is of degree 4

Subcase 3b : The faulty node is of degree 4.

This situation is illustrated in Fig. 3.6. Let the hops that are originated from f be of type l. Also, let $q \neq l-2$ and $r \neq l+2$. Hence, to bypass f, a sequence of hops $\{-h_{l-2}, -h_{l-2}, -h_{l-2}, -h_{l-2}, -h_l, -h_l, -h_l, h_{l+2}\}$ can be used in between h_q and h_r. Thus, we are inserting 8 more hops in the original sequence of hops in P. A point should be noted here that the upper bound $\hat{L}$, on the length of the path P is a conservative estimate. While computing the value of $\hat{L}$, we assumed that the path P contains all the even or odd types of hops at least once. But in this particular case, we see that there are at least three types of hops, namely, h_{l-2}, h_l and h_{l+2} which were not included in P. This reduces the upper bound on $L(P)$ to $\hat{L} - 3$. Therefore, in presence of such a faulty node, the path length would be at most $L(P) + 8 \leq \hat{L} + 5$.

If $q = l-2$, then the sequence $\{h_q, h_r\}$ in P will be replaced by the sequence $\{-h_q, -h_q, -h_q, -h_l, -h_l, -h_l, h_{l+2}, h_r\}$. If $r = l+2$, then the sequence $\{h_q, h_r\}$ in P will be replaced by the sequence $\{h_q, h_{l-2}, h_{l-2}, h_{l-2}, h_{l-2}, h_l, h_l, h_l\}$. Thus, the path length will be increased by 6.

Case B :

Let us now consider the case when the faulty node lies in the destination sector. Of course, we assume that $f \neq d$. Actually, the fault would affect the path P only if it lies in that part of the destination sector which is included in P. Let us call that part as the *restricted portion* of the destination sector. These restricted portions will be different for different traversals from s to $\bar{d}_0$ as discussed in the last part of Section 3.3.2. We shall consider those cases separately.

Case 1 : When s and $\bar{d}_0$ are in the different halves of the sector 0.

Subcase 1a : $[\, s, \bar{d}_0\,] \cap [u_0^1, u_0^p] = \phi$.

i) $s + d < m$

Here, in the destination sector (that is, in sector δ) the restricted portion is from x_δ to d.

If d is a degree-4 node, then let us assume that the hops that originating from d be of type q. To bypass the fault in the restricted portion, along with the hops in P, we shall use two additional hops $+h_q$ and $-h_q$. The resulting traversal will be as follows. Start traversing along P, as discussed in Section 3.3.2. During this traversal, we would have passed through a node $\bar{d}_r$, the projection of d on some sector r, $r \neq \delta$. As soon as we reach such a node for the first time, we use the hop $+h_q$, which is not originally in P. Then, again we shall follow the same order of using hops and c-edges as it was in the path P. Because of using an additional hop in the new traversal, after using all the other hops in P we shall reach the sector $(\delta + 2^q)$. Finally, from the node $\bar{d}_{\delta+2^q}$ in that sector, we would take the hop $-h_q$ to come back to the destination sector δ directly through the node d. Thus, in this case, we can bypass the fault at the expense of two additional edges.

If d is not a degree-4 node, then let us assume that the hops originating from the node $d+1$ be of type q. Here also we shall use two additional hops $+h_q$ and $-h_q$ in the new traversal so that as soon as we reach the projection of $d+1$ on some sector r for the first time, $r \neq \delta$, we take $+h_q$ and finally we would enter the destination sector through the node $d+1$ by using a $-h_q$ hop. From there, we have to come back to the node d using a c-edge. Thus the path length will be increased by 4.

ii) $s + d \geq m$

Here the restricted portion is from u_δ^1 to d. This situation can be tackled in a way similar to the case (i) above, by taking two additional hops of type same as that originating from either d or $d-1$. Thus, here we bypass the fault at the expense of at most 4 additional edges.

Subcase 1b : $[u_0^1, u_0^p] \subseteq [s, \bar{d}_0]$

Here, corresponding to the path P, the restricted portion in the destination sector is from u_δ^p to d. Let us consider the following two cases.

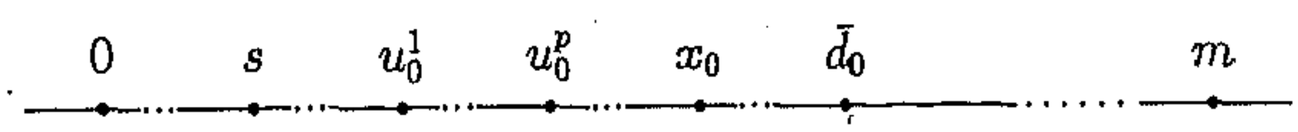


Figure 3.7: The restricted portion corresponds to $[u_0^p, \bar{d}_0]$

Case (a) : If $\bar{d}_0$ and $[u_0^1, u_0^p]$ are in the different halves of the sector, as shown in Fig.3.7, then to avoid the restricted portion in the destination sector, we will choose any one of the following two options :

Option 1 : Without loss of generality, let us assume that the path P contains only even type of hops. Thus, the total number of hops included in P will be at most $2w_\delta^o + w_\delta^e + 1$. The new traversal can be done as follows. Starting from s, use only c-edges to reach x_0. From there use h_0 to reach x_1 in sector 1. From sector 1, start

using hops in increasing order of magnitude. After using all the hops in H_δ, use only c-edges to reach d in sector δ. Thus, in this case, the total path length will be

$$L(P_1) = \frac{m}{2} - s + 1 + \frac{m}{2} + m - \bar{d}_0 + 2w_\delta^o + w_\delta^e + 1$$

Option 2 : In this alternative path, we will use only odd type of hops. As a result, the range of H_δ will now be in the other half of the sector. That is, $[u_0^1, u_0^p]$ and $\bar{d}_0$ are now in the same half of the sector. But at this point $[u_0^1, u_0^p]$ may not be a subset of $[s, \bar{d}_0]$. The number of hops in this case will be $2w_e^0 + w_\delta^o + 1$. Here, we shall bypass the fault by the same technique (at the expense of at most 4 edges) as we have taken in the first part of the subcase 1a. The total path length will be

$$L(P_2) = s + \frac{m}{2} + 1 + \bar{d}_0 - \frac{m}{2} + 2w_\delta^e + w_\delta^o + 1 + 4$$

Now, $\min(L(P_1), L(P_2)) \leq \frac{L(P_1)+L(P_2)}{2} \leq \frac{11m}{8} + 5$. This shows that the path length may increase by at most 4, in presence of such a fault.

Case (b) : If $\bar{d}_0$ and $[u_0^1, u_0^p]$ are in the same half of the sector then if we interchange s and d, the situation will be identical to the case (a) above.

Subcase 1c : $[u_0^1, u_0^p] - [\, s, \bar{d}_0\,] \neq \phi$

This situation can be treated in a similar way as we have done in subcase 1a.

Case 2 : s and $\bar{d}_0$ are in the same half of the sector.

With respect to the path P, discussed in the remark made in section 3, the restricted region in the destination sector will be included in the portion from x_δ to d.

Let the hops that originate from either $\bar{d}_0$ or $\bar{d}_0 - 1$ be of type q according to the situation whether d is a node of degree 4 or not, respectively. The fault in the destination sector can be bypassed by using three additional hops, namely, h_{q-1}, h_{q-1} and h_q. If we traverse along P, we would have passed through a node from which h_{q-1} originates. As soon as we enter such a node we will take h_{q-1} twice so that after using all the hops in P we will reach the sector $\delta + 2^q$. From there

we will take a $-h_q$ hop to reach the destination d. Considering two additional c-edges at most, the fault can be bypassed at the expense of a maximum of five edges.

From the above discussion, we can conclude that in presence of a single fault, the diameter of the topology can be increased at most by 6.

3.6 Implementation of Algorithms

When N is a power of 2, a class of parallel algorithms, known as ASCEND and DESCEND types of algorithms [PV81], can easily be implemented on the proposed network topology.

Suppose, $N = 2^q$ and the input data a_0, a_1, $\cdots$, a_{N-1} are stored in the processors $P[0]$, $P[1]$, $\cdots$, $P[N-1]$, respectively. An algorithm is in the ASCEND class if a sequence of operations is carried out between a pair of data that are successively 2^0, 2^1, $\cdots$, 2^{q-1} processor locations apart. In DESCEND class of algorithms, the operations are carried out just in reverse order. These classes of algorithms have applications in the problems like *cyclic shift*, *bitonic merge*, *odd-even-merge*, *Fast Fourier Transform*, *shuffle*, *matrix transposition*, *bitonic sort* etc. Now, we shall discuss the implementation of such algorithms in our case. For brevity, we shall discuss here only the ASCEND type of algorithms. For the ease of discussion, let us first renumber the nodes of $G(m, N)$.

Renumbering of nodes : Let $N = 2^q$ and $m = 2^r$. Therefore, $q = 2^{r-1} + r + 1$. Since there are $N/m = 2^{q-r}$ sectors, $G(m, N)$ contains 2^{q-r} cycles each of length $m + 1$, containing m c-edges and a single hop h_0. Let us number these cycles as 0, 1, $\cdots$, $2^{q-r} - 1$, so that the cycles i and $(i + 1)$ mod 2^{q-r} have one node in common. The node in cycle 0 from which the hop h_0 is originated, is now renumbered as 0. The remaining nodes are numbered from 1 to $N - 1$ along the largest cycle of length N, so that the cycle i now consists of the renumbered nodes $\{im,\ im + 1,\ \cdots,\ (i + 1)m\}$. We would also represent any such renumbered

node by an ordered pair (l,p), $0 \leq l \leq 2^{q-r}-1, 0 \leq p \leq m-1 = 2^r-1$, where l represents the cycle number which this node belongs to and p represents its distance from the node lm. Note that, since $p < m$, every node will have a unique representation by such an ordered pair. Thus, to address any of the N nodes, we require q bits in which the most significant $q-r$ bits would represent the cycle number and the least significant r bits would represent the position of the node in the corresponding cycle. Also, if a node is renumbered as n, then, $n = l.2^r + p$. An example of this renumbering scheme for $G(5,40)$ is shown in Fig. 3.8.

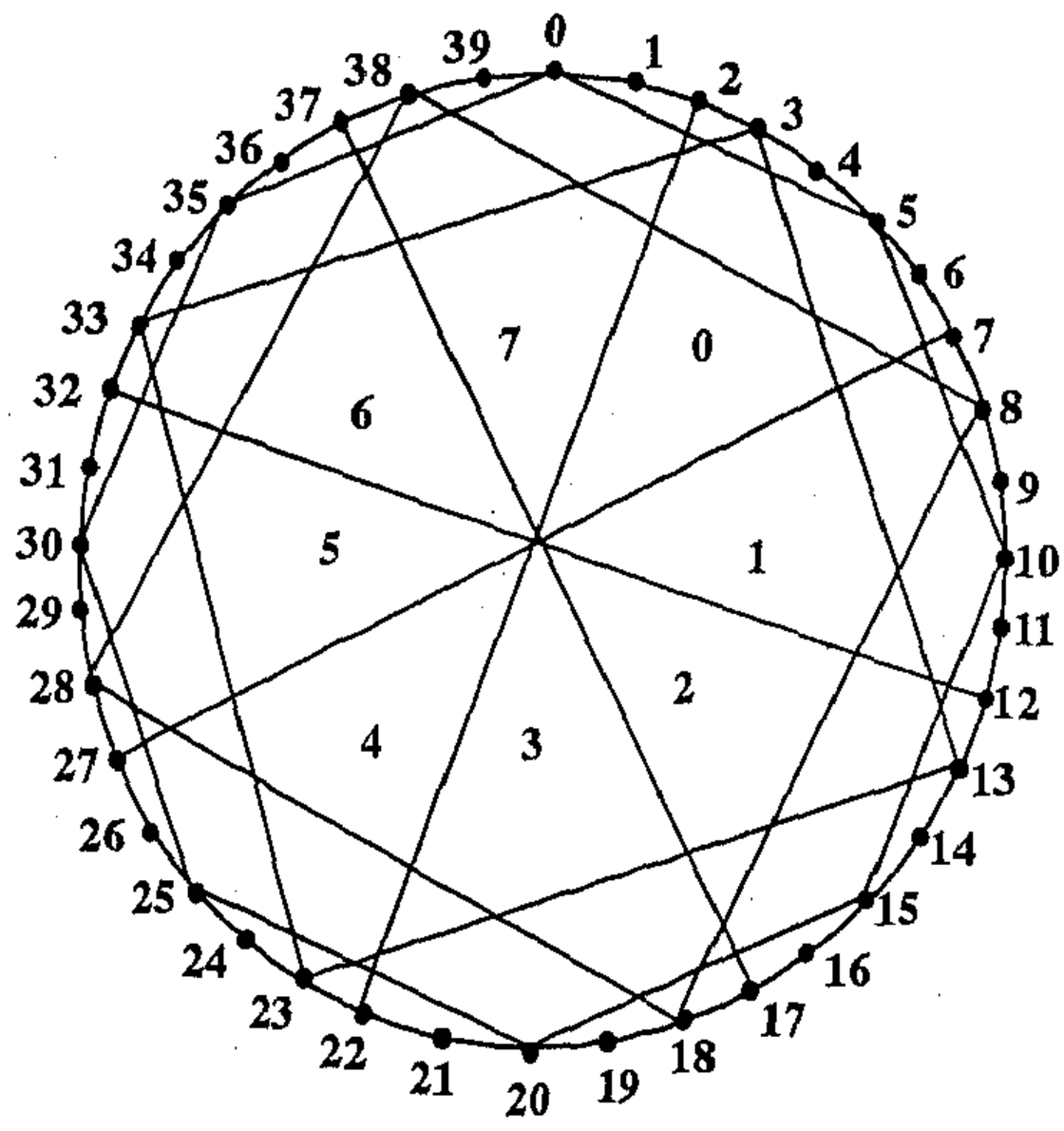


Figure 3.8: The nodes are renumbered in $G(5,40)$

In our later discussions, we refer to a node by this renumbered value.

Before going to discuss the implementation, we now describe an ASCEND type of algorithm in the following two steps, where a basic operation between the two processors $P[i]$ and $P[r]$ has been indicated by OPER(i, j, $P[i]$, $P[r]$), when $r = i + 2^j$. $\text{bit}_j(i)$ denotes the jth least significant bit of the binary representation of i.

Proc ASCEND

Step 1 : /* Process data elements within each cycle of length $m+1$ */

begin

for each l, $0 \leq l \leq 2^{q-r}-1$, **do in parallel**

begin

for $j = 0$ **to** $r-1$ **do**

begin

for each p, $0 \leq p \leq 2^r-1$ **do in parallel**

if $\text{bit}_j(p) = 0$ **then**

OPER(p, j, $P[(l, p)]$, $P[(l+2^j, p)]$)

end;

end;

end;

Step 2 : /* Processes data elements across the cycles */

begin

for $j = r$ **to** $q-1$ **do**

begin

for each i, $0 \leq i \leq N-1$ **do in parallel**

if $\text{bit}_j(i) = 0$ **then**

OPER(i, j, $P[i]$, $P[i+2^j]$)

end;

end;

Step 1 can be implemented in a similar way as it was discussed in [PV81]. This step can be executed on $G(m, N)$ in time linear in cycle length, that is, in $O(m)$ time. We shall now discuss the implementation of step 2 on $G(m, N)$. As an example, we choose $G(2^3, 2^8)$. Fig. 3.9 illustrates the hop distribution in cycle 0 of $G(2^3, 2^8)$.

Initially, let us start with the assumption that the data element a_i is stored in the processor $P[i]$, $\forall\ i, 0 \leq i \leq N-1$. We shall now concentrate only on those

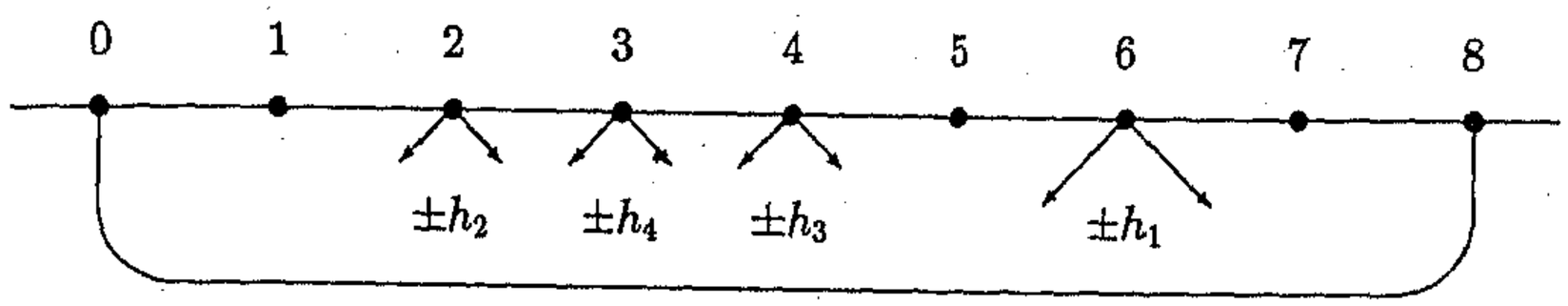


Figure 3.9: Hop distribution

operations which involve a_0, the data element initially stored in the processor $P[0]$. Step 2 of ASCEND algorithm demands that the operations between the pairs of data elements stored in the processors $(P[0], P[8])$, $(P[0], P[16])$, $(P[0], P[32])$, $(P[0], P[64])$, $(P[0], P[128])$ should be carried out successively. At first, using the hop h_0, the processor $P[0]$ and $P[8]$ can communicate with each other so that any operation can be carried out between a_0 and a_8. To perform the later operations, the data elements are to be shifted to some other nodes where the suitable hops are available. This can be made possible by successive shifting of each data element through all the nodes in the cycle, which it belongs to. From this point it is clear that the corresponding positions within a cycle must be same for the data elements among which operations are to be carried out. Now, by three successive shifts, the data element which is actually stored in $P[0]$ can be brought to the processor $P[6]$ from where the hop h_1 (which is actually a direct connection between cycle 0 and cycle 2) can be used to perform an operation with the element which was supposed to be stored in the processor $P[16]$ initially. Then to avail the next hop originating from the node 4, again two successive shifts are needed. The next available hop is h_3 which connects cycle 0 and cycle 8. With the help of this hop, an operation can be carried out between the elements which were initially stored in the processors $P[0]$ and $P[64]$. But, to fulfill the requirement of ASCEND algorithm the third operation must be between the data which were initially stored in the processors $P[0]$ and $P[32]$. Therefore, initially a_{32} must be stored in the processor $P[64]$. Similarly, a_{64} should be placed initially in $P[128]$ and a_{128} in $P[32]$. Thus, we see that some appropriate permutation must be specified for initial distribution

of data elements among the processors. In general, the required permutation is described below in the inputting scheme.

Inputting of data elements : Let the sequence of hops originated from the nodes of the cycle l, $0 \leq l \leq 2^{q-r} - 1$, in the order $(l,1), (l,2), \cdots, (l,m-1)$, $(l+1,0)$ be designated as $< h_{i_1}, h_{i_2}, \cdots, h_{i_\alpha} >$, where $\alpha = 2^{r-1}$. Clearly, $h_{i_\alpha} = h_0$. For example, the sequence of hops in any cycle of $G(2^3, 2^8)$ is $< h_2, h_4, h_3, h_1, h_0 >$ (here, h_4 is the diagonal edge of length 2^7). The sequence of hops $< h_{i_1}, h_{i_2}, \cdots, h_{i_\alpha} >$ can be alternatively designated by a sequence of numbers $< i_1, i_2, \cdots, i_\alpha >$. Thus, for $G(2^3, 2^8)$, the sequence of hops is denoted by $< 2, 4, 3, 1, 0 >$.

We now define a permutation π_m, associated with $G(m, N)$ as follows:

$$\pi_m = \begin{pmatrix} q-r & q-r-1 & \cdots & 0 \\ i_1 & i_2 & \cdots & i_\alpha \end{pmatrix}$$

For $G(2^3, 2^8)$, $$\pi_{2^3} = \begin{pmatrix} 4 & 3 & 2 & 1 & 0 \\ 2 & 4 & 3 & 1 & 0 \end{pmatrix}$$

If we scan the top row of π_m from the right end, we get the figures correspond to the types of the hops to be successively taken for executing step 2 of the ASCEND algorithm. The bottom row of π_m, on the other hand indicates the order of the hop types actually existing in the network $G(m, N)$, when scanned from one end of a cycle of length m.

Let the processor $P[i]$ be placed at the node i. The N data elements $a_0, a_1, \cdots, a_{N-1}$ will be distributed among the processors $P[0], P[1], \cdots, P[N-1]$, in such a way that the element a_i will be stored in the processor $P[j]$, where, $i = (l,p)$, $j = (l',p)$ and the binary representation of l' is obtained by taking the permutation π_m on the binary representation of l.

Let us now discuss the implementation of step 2 on $G(m, N)$:

For a fixed j the computation corresponding to the *for* loop in step 2 can not be executed in one parallel step, because within a cycle only one node is actually connected to a node at 2^j distance apart, $\forall j$, $r \leq j \leq q-1$. Therefore, by means

of successive circular shifts all the data elements in the cycle should be brought to that node, so that OPER(., j, ., .) can be executed. For each j, this step requires $2^r + 1$ $(= m + 1)$ units of time. However, this computation can be pipelined and the total time required to execute step 2 can be reduced to only $O(m)$.

Step 2 can be explained in the following way.

Code section for the processor $(l,\ p)$ in Step 2 :

/* Assume that $\bar{l} = \pi_m^{-1}(l)$, where π_m^{-1} denotes the inverse permutation of π_m. That is, the binary representation of $\bar{l}$ is obtained by taking the permutation π_m^{-1} on the binary representation of l. Moreover, if $(l,\ p)$ is a degree-4 node then assume that $\pm h_k$ originate from that point. Let b represent the k^{th} bit of $\bar{l}$. The value of b can be either 0 or 1. */

for $i = 1$ to $2(m+1)$ **do**

begin

Step (2a) :

if ($(l,\ p)$ is a degree 4 node) **then**
  if $m - p + 2 \leq i \leq 2m - p + 1$ **then**
    if $(b = 0)$ **then**
      use $+h_k$ to communicate with the processor $(l + 2^k,\ p)$ to perform any operation on the data elements stored in these two processors;
      /* if $(p = 0)$ then perform this operation on the data which the processor $(l,\ 0)$ has obtained from $(l,\ 1)$. */
    else
      use -h_k to communicate with the processor $(l - 2^k,\ p)$ to perform any operation on the data elements stored in these two processors;
      /* if $(p = 0)$ then perform this operation on the data which the processor $(l,\ 0)$ has obtained from $(l,\ 1)$. */

Step (2b) :

if ($p \neq 0$) **then**

send the data to (l, $p-1$)

else

send the data which was obtained from (l, 1) to ($l+1$, 0) and that which was obtained from ($l-1$, 0) to the processor ($l-1$, $m-1$);

end

Since the nodes $i.m$ ($0 \leq i \leq 2^{q-r}-1$) are at the joint of two successive cycles, cycle i and $(i-1) \bmod 2^{q-r}-1$, care should be taken for proper pipeline of data elements within a cycle, by keeping two separate registers for the two cycles.

The sequence of events corresponding to the *for* loop in step 2 is illustrated in Table 3.1, for $G(2^3, 2^8)$. In any cycle l of $G(2^3, 2^8)$, $0 \leq l \leq 31$, there are 8 nodes, namely, (l, p), $0 \leq p \leq 7$. For each of these processors, the time units during which they execute step 2a are marked "Y".

The *for* loop in step 2 is executed $2m+2$ times. Moreover, for each i, $(1 \leq i \leq 2m)$, it takes two units of time, one for step 2a and another for step 2b. Therefore, the time required to execute step 2 is 4m + 2 units, that is $O(\log N)$ time. So, we can conclude that the ASCEND class of algorithms can be implemented on $G(m, N)$ in $O(\log N)$ units of time, when N is a power of 2.

Table 3.1: Comparison of $G(m, N)$ with different topologies

Node	Time units															
$(l, .)$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(., 0)	Y	Y	Y	Y	Y	Y	Y	Y								
(., 7)																
(., 6)				Y	Y	Y	Y	Y	Y	Y	Y					
(., 5)																
(., 4)						Y	Y	Y	Y	Y	Y	Y	Y			
(., 3)							Y	Y	Y	Y	Y	Y	Y	Y		
(., 2)								Y	Y	Y	Y	Y	Y	Y	Y	
(., 1)																

3.7 Comparison With Other Topologies

Table 3.2 compares the number of links and diameter of $G(m, N)$ with those of the chordal ring and the distributed loop network, for different values of N.

Table 3.2: Comparison of $G(m, N)$ with different topologies

	Chordal Ring		Distributed Loop Network		Proposed Network Proposed Network	
Total number of nodes (N)	Number of links	Diameter	Number of links	Diameter	Number of links	Diameter
96	144	≥ 9	192	≥ 7	152	≤ 8
256	384	≥ 15	512	≥ 11	400	≤ 11
640	960	≥ 25	1280	≥ 18	992	≤ 14
1536	2304	≥ 38	3072	≥ 28	2368	≤ 17
3584	5376	≥ 59	7168	≥ 42	5504	≤ 19

3.8 Conclusion

A new family of graphs with constant node degree and low diameter has been introduced. With a slight modification in the definition, this family of graphs can be constructed for all most all possible values of N. Given N, if N' is the nearest number to N, $N \leq N'$, on which the graph is defined, then construct a graph with N' nodes. Delete $N' - N$ number of degree 2 nodes, evenly from each sector. The diameter, $D(N)$, would be upper bounded by $D(N')$ and the maximum node degree would remain same as in the original graph.

Chapter 4

Generalized Hypercube-Connected-Cycles

4.1 Introduction

Let us consider the following problems which one may have to face in many practical situations in designing a family of regular network topologies :
(i) Given the size of the network N, and the diameter D (or the node degree δ), design a network with node degree (diameter) as small as possible.
(ii) Given two positive integers δ and D, design a network topology with a suitable number of nodes having node degree δ and diameter D.
In the latter case, the value of N is not fixed *a priori* but is determined by the given values of D and δ.

Thus, in these design problems, we basically aim at finding a suitable family of regular topologies in which any two parameters out of N, δ and D can be fixed according to our choice.

Regarding the first design problem, we note that usually, a given family of network topologies can not be defined for all values of N, i.e., N assumes a value from a definite set of (usually sparse) integers defined by that family of network. There are very few topologies, e.g., ring [L92], chordal ring [AL91], etc., which can be

constructed for all values of N; but a major drawback of these topologies is that their diameters are very large. In view of this observation, we modify the first design problem as follows :

Given a value for N and the diameter D (or the node degree δ), design a network having node degree (diameter) as small as possible with total number of nodes sufficiently close to (greater than or equal to) N.

For the solution of the problems mentioned above, we can first try with the existing families of network topologies as follows.

From the point of view of degree and diameter, the existing topologies can be broadly classified into two categories. Constant degree network topologies form one category. Ring [L92], chordal ring [AL91], distributed loop network [BT91], mesh [AK83], cube-connected cycles [PV81], Moebius graph [LS82], de Bruijn graph [B46], etc., are well known members of this class. In case of these constant degree graphs the network size (N) can be suitably chosen by taking the constraint on the diameter (D) into consideration. Alternatively, if N is given, D will accordingly assume a fixed value. Hence, for every family of such graphs, only one out of the three parameters N, δ and D can be freely chosen.

On the other hand, the second category of the existing topologies includes the hypercubes [L92], [H69], generalized hypercubes (GHC) [BA84], folded hypercubes [AL91], star graphs, pancake graphs [AK89], radix-r de Bruijn graphs [PR82] etc. For these topologies, there exists a functional relationship (explicit or implicit) among the number of nodes N, degree δ and diameter D for each of these families of networks. For example, a hypercube with diameter D can only have node degree D and total number of nodes 2^D. In all these topologies except a few, e.g., GHC, radix-r de Bruijn graph, etc., if we choose a value for one of the three parameters N, δ and D, the other two parameters are automatically fixed. In other words, there is again only one parameter out of N, δ and D which can be freely controlled in these structures.

The generalized hypercube and the radix-r de Bruijn graphs, however, provide a

better option. Here, one can freely handle two of the three parameters N, δ and D in a certain restricted way. That is, for a fixed diameter (degree) one can have multiple choices for the node degree (diameter) and the network size, subject to some restrictions.

In this chapter, we propose a new family of interconnection networks which we refer to as the Generalized Hypercube-Connected-Cycles ($GHCC$). We use this terminology because a generalized hypercube structure can be obtained from it by coalescing certain groups of nodes forming cycles in the structure. This family of topologies provides a way for freely choosing any two of the three parameters N, δ and D.

The proposed topology is a regular graph $GHCC(l, m)$, where l and m are two free parameters of the graph influencing the diameter and the degree of the graph independently. The total number of nodes N in $GHCC(l, m)$ is lm^l. For $l = 1$ and 2, the degree of each node is $m - 1$ and m respectively, while for $l \geq 3$, the network is $(m + 1)$-regular. The diameter of the graph is $\lfloor 5l/2 \rfloor - 2$, for $l > 3$, $m \neq 1$ and $\lfloor 5l/2 \rfloor - 1$, for $l = 1, 2, 3$, $m \neq 1$. In terms of the total number of nodes N, the diameter is $\simeq \frac{5}{2} \log_m N$. We would show that given the two design specifications δ and D, it is possible to find out the appropriate values of m and l to get a network with $N = lm^l$ nodes, node degree δ and diameter in the range $D - 2 \leq$ diameter $\leq D$. Thus, degree and diameter can be freely and independently chosen for such families of networks. Conversely, given the values of N and δ (or D), we can properly choose l and m so that $lm^l \geq N$ with the desired δ (or D). We will show that GHCC competes favorably with GHC, star graphs, etc. in regard to the degree and the diameter for a given number of nodes.

As special cases, GHCC reduces to a ring for $m = 1$, while it reduces to a complete graph for $l = 1$. If m takes the value of 2, then GHCC reduces to cube-connected cycles as well.

The proposed topology is regular, node symmetric, maximally connected and also provides an easy routing scheme. For $l > 3$, in presence of m faulty nodes, the

diameter of the GHCC increases only by a constant amount. We will also show that many useful algorithms can be efficiently implemented on this topology.

4.2 The Topology of GHCC

We describe the topology in terms of the graph $G(l, m)$, having the following characteristics :

a) N is the total number of nodes in the graph.

b) l and m are two parameters of the graph.

c) $N = lm^l$. We may consider that the total number of N nodes are clustered into l levels, referred to as level 0, level 1, $\cdots$, level $l-1$, each of which contains m^l nodes.

d) Node specification :

(i) To identify a node uniquely inside one level, we number each of the m^l nodes in a particular level by an integer ranging from 0 to $m^l - 1$. We call this number as the *index* of a node in a particular level. This index can be represented by a string of l literals, each literal can take a value from 0 to $m-1$ (both inclusive).

(ii) To distinguish among the nodes at different levels, we use another literal to identify the level number, which ranges from 0 to $l-1$ (both inclusive).

Thus, a node in $G(l, m)$ can uniquely be represented using an ordered set of literals $(p; u_{l-1}u_{l-2}\cdots u_1u_0)$, where p signifies the level number, $0 \leq p \leq l-1$, and the literal string $u_{l-1}u_{l-2}\cdots u_1u_0$ is used to identify the index of a specific node within the level p. Note that, p is an l-valued literal while u_i's, $0 \leq i \leq l-1$, are m-valued literals each.

Another equivalent way of representing a node is by a two tuple (p, n), where p represents the level number and n represents the index such that $n = \sum_{i=0}^{l-1} u_i * m^i$.

We will use these two notations interchangeably.

e) There are two types of links among the nodes, **interlevel** and **intralevel** links.

(i) Interlevel links : There are l nodes in the network which have same index but they are distributed in l different levels. That is, they differ only in their p-values. These l nodes are connected in the form of a ring (cycle) through interlevel links. Thus, a node $(p;\ u_{l-1}u_{l-2}\ \cdots\ u_1u_0)$ is connected to two nodes $((p \pm 1);\ u_{l-1}u_{l-2}\ \cdots\ u_1u_0)$, in two adjacent levels through a *cyclic edge*, all additions and subtractions are done in modulo l. These interlevel links will form m^l node-disjoint cycles in the topology.

Any such cycle consisting of the nodes of the form $(*; u_{l-1}u_{l-2}\cdots u_1u_0)$, where $*$ indicates all possible values from 0 to $l-1$, will be denoted by $C_{(u_{l-1}u_{l-2}\cdots u_1u_0)}$.

(ii) Intralevel links : Inside one level, a node $(p;\ u_{l-1}u_{l-2}\ \cdots\ u_{p+1}u_pu_{p-1}\ \cdots\ u_1u_0)$, is connected to $m-1$ nodes $(p;\ u_{l-1}u_{l-2}\ \cdots\ u_{p+1} * u_{p-1}\ \cdots\ u_1u_0)$ through *clique edges*, where $*$ signifies all possible values of the corresponding literal from 0 to $m-1$ excluding the value u_p. These connections form m^{l-1} disjoint cliques of size m within each level.

f) There are $m-1$ intralevel links and two interlevel links emanating from each node when $l \geq 3$. Thus, the graph is regular with node degree $m+1$, when $l \geq 3$. For $l = 1$ and $l = 2$, the graph is regular with degrees $m-1$ and m respectively.

g) From the connection pattern it can be easily verified that the graph is not only regular, but it is node symmetric as well.

An example of this graph is shown in Fig. 4.1 for $m = 3$ and $l = 3$. In Fig. 4.1, the nodes at each level are shown in separate columns. The nodes shown within a dotted box form a complete graph.

If the nodes with the same index (i.e., the nodes lying in the same cycle) are coalesced to form a single node, without violating the adjacency relationship among

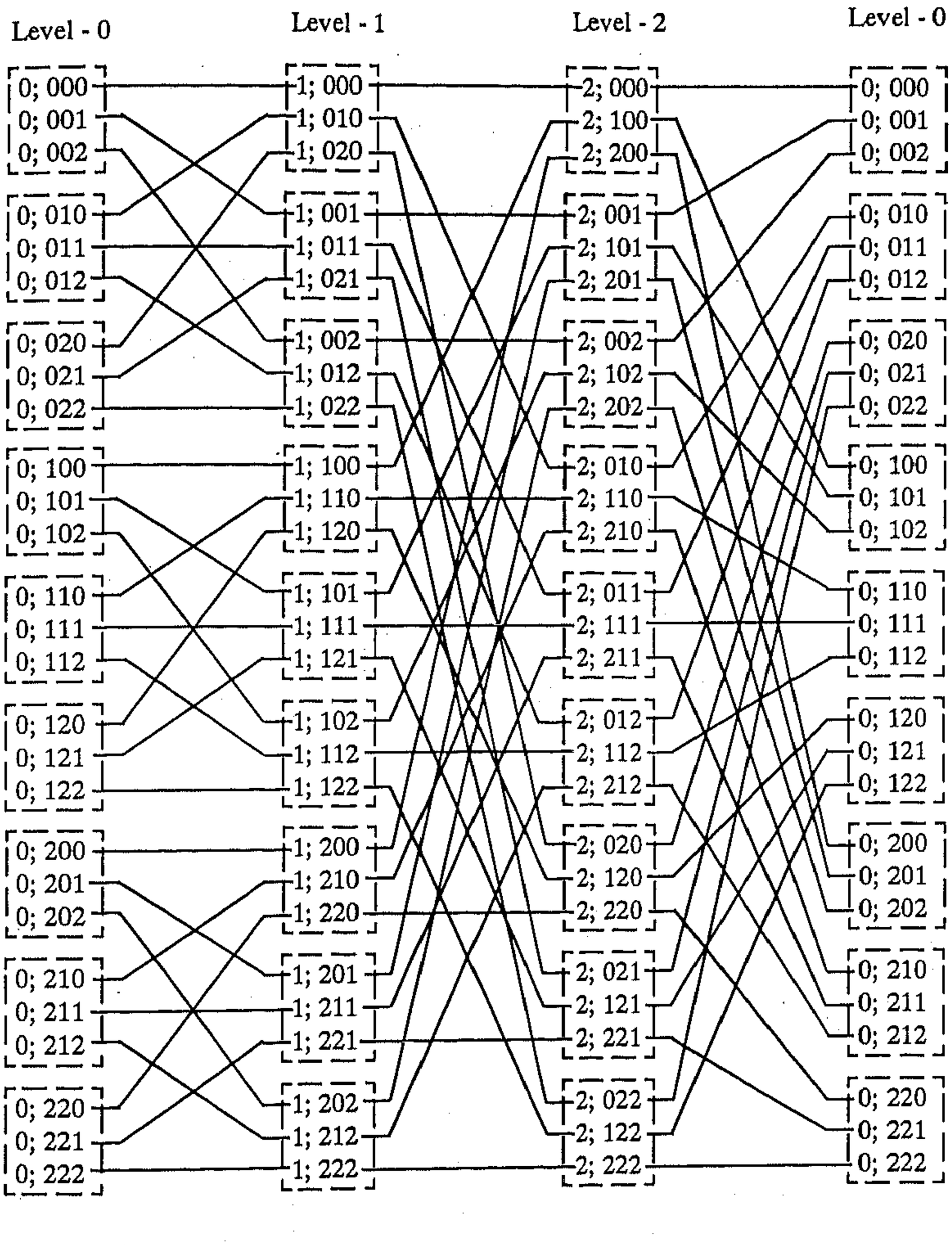


Figure 4.1: The GHCC for $l = 3$, $m = 3$

the nodes, then the proposed topology will reduce to a generalized hypercube with m^l nodes in total (m nodes in each of l directions). This is the reason why we call this topology as Generalized Hypercube-Connected-Cycles (GHCC). Moreover, if we take $m = 2$, then the topology $G(l, m)$ reduces to the cube-connected cycles with $l.2^l$ nodes in all. In our later discussion, we would refer to a GHCC by both $G(l, m)$ and $GHCC(l, m)$ interchangeably.

4.3 Diameter

When $m = 1$ the topology reduces to a ring of length l whose diameter is $\lfloor \frac{l}{2} \rfloor$. We now consider the cases for which $m \neq 1$.

First, let us find a path between any two nodes $s = (p_1, u)$ and $d = (p_2, v)$, where, $0 \leq p_1, p_2 \leq l - 1$ and

$$u = u_0 + u_1.m + u_2.m^2 + \cdots u_{l-1}.m^{l-1}$$

$$v = v_0 + v_1.m + v_2.m^2 + \cdots v_{l-1}.m^{l-1}$$

where, for all i, $0 \leq i \leq l - 1$, $0 \leq u_i, v_i \leq m - 1$.

To find the diameter of the topology, we first show that there is an upper bound on the shortest path length from any source node s to a destination node d. For this, we consider the following traversal from s to d in two steps :

In the first step, after starting from $s = (p_1; u_{l-1}u_{l-2}\cdots u_{p_1+1}u_{p_1}u_{p_1-1}\cdots u_1u_0)$, if $u_{p_1} = v_{p_1}$ then traverse a cyclic edge to move to the node $(p_1+1;\ u_{l-1}u_{l-2}\cdots u_{p_1+1}v_{p_1}$ $u_{p_1-1}\cdots u_1u_0)$ in the next level p_1+1. Otherwise (if $u_{p_1} \neq v_{p_1}$), using a clique edge, first move to the node $(p_1;\ u_{l-1}u_{l-2}\cdots u_{p_1+1}v_{p_1}u_{p_1-1}\cdots u_1u_0)$ in the level p_1 and then from this node move to the node $(p_1 + 1;\ u_{l-1}u_{l-2}\cdots u_{p_1+1}v_{p_1}u_{p_1-1}\cdots u_1u_0)$ in the level $p_1 + 1$. Follow the same sequence of operations also in the subsequent levels $p_1 + 1, p_1 + 2, \cdots, p_1 - 1$ until the node $(p_1 - 1;\ v_{l-1}v_{l-2}\cdots v_{p_1+1}v_{p_1}v_{p_1-1}\cdots v_1v_0)$ is reached. The traversal is thus made through the nodes with non-decreasing level

numbers following the sequence p_1, p_1+1, p_1+2, $\cdots$ upto p_1-1 ($\pm$ operations are done in modulo l). We call this as *clockwise traversal.*

If the traversal of the levels are done in the reverse order, that is, in the direction of non-increasing level numbers, then we will ultimately reach the node $(p_1+1;\ v_{l-1}v_{l-2}\cdots v_{p_1+1}v_{p_1}v_{p_1-1}\cdots v_1v_0)$ after this step and this will be termed as *anti-clockwise traversal.*

In both these cases, the number of cyclic edges used is $l-1$ and the number of clique edges is at most l. Thus, starting from s, at most $2l-1$ edges are required to reach either $(p_1-1;v)$ or $(p_1+1;v)$.

We are then left with the job of changing the level number from either p_1-1 or p_1+1 to p_2 which is done in the second step of the traversal. Since the nodes with the same node index v are connected in the form of a cycle of length l, this part of the traversal can be done using cyclic edges only. It can be easily verified that the number of cyclic edges required in this part is at most $\lfloor\frac{l}{2}\rfloor-1$ (when $l>3$), if in the first part we suitably choose the direction of the traversal. If $l=1,2$ or 3, then the number of required cyclic edges is at most $\lfloor\frac{l}{2}\rfloor$.

Hence, we get the following result.

Lemma 4.1 *Between any two nodes (p_1,u) and (p_2,v), $0\le p_1,\ p_2\le l-1$ and $0\le u,v\le m^l-1$, there always exists a path of length at most $\lfloor\frac{5l}{2}\rfloor-2$, for $l>3$, $m\neq 1$, and $\lfloor\frac{5l}{2}\rfloor-1$, when $l=1,2,$ or 3 and $m\neq 1$.*

Theorem 4.1 *The diameter of the GHCC $G(l,m)$ is $\lfloor\frac{5l}{2}\rfloor-2$, when $l>3$, $m\neq 1$ and $\lfloor\frac{5l}{2}\rfloor-1$, when $l=1,2,$ or 3, $m\neq 1$.*

Proof : From Lemma 4.1, it follows that the diameter is less than or equal to $\lfloor\frac{5l}{2}\rfloor-2$, for $l>3$ and $\lfloor\frac{5l}{2}\rfloor-1$, for $l\le 3$. We will now show that there exists at least a pair of nodes which are exactly at a distance of $\lfloor\frac{5l}{2}\rfloor-2$ ($l>3$) and $\lfloor\frac{5l}{2}\rfloor-1$ ($l\le 3$). Consider the two nodes $s=(0;\ 000\cdots00)$ and $d=(\lfloor l/2\rfloor;\ 111\cdots11)$. Note that the indexes of these two nodes differ in all the l literals. As a result,

the path joining s and d must pass through all the levels and in each level the corresponding literal values should be set accordingly by traversing through a clique edge. Because of the ring connections among the nodes in different levels, we need at least $2l-1$ edges ($l-1$ cyclic and l clique edges) to achieve this. If we traverse the levels in the anti-clockwise direction, then these $2l-1$ edges will take us to the node $(1;\ 111\cdots11)$. From this node, we need another $\lfloor l/2 \rfloor - 1$ edges (for $l > 3$) or $\lfloor l/2 \rfloor$ edges (for $l \leq 3$) along the ring connecting the nodes $(*;\ 111\cdots11)$ (where $*$ indicates all possible values for the level number) to reach the final destination. Thus, the shortest distance between s and d is $\lfloor \frac{5l}{2} \rfloor - 2$, for $l > 3$ and $\lfloor \frac{5l}{2} \rfloor - 1$, for $l \leq 3$. □

For $l = 2r$, $2r+1$, $2r+2$ and $2r+3$, the values of the diameter according to the Theorem 4.1 are $5r-2$, $5r$, $5r+3$ and $5r+5$ respectively. Hence given a value D for the diameter, we can always choose the value of l appropriately so that the actual diameter is in the range $D-2$ through D.

4.4 Comparison With Other Topologies

As a special case, if we take $l = m$ ($l > 2$), then the total number of nodes in $G(l,\ m)$ would be $N = m^{m+1}$ with degree as $m+1$ and diameter as $\lfloor \frac{5m}{2} \rfloor - 2$ ($l > 3$) or $\lfloor \frac{5m}{2} \rfloor - 1$ ($l = 3$). A hypercube with the same number of nodes would have degree and diameter both equal to $(m+1)\log_2 m$. This shows that our proposed topology outperforms the hypercube in regard to the degree and the diameter. Also, it may be noted that with a proper choice of l and m, the degree and the diameter of the graph will be comparable with those of a star graph having number of nodes less than or equal to lm^l. Table 4.1 shows the comparative figures for the degree and the diameter of different topologies with number of nodes N not exceeding lm^l, the number of nodes of the proposed graph with a given choice of l and m. Some of the entries in the column GHC of Table 4.1 show two values of the tuple $< N, \delta, D >$ for two different configurations of the GHC with the same value of N, (i) one with the minimum possible value δ_{min} for the degree δ and (ii)

the other (if possible) with the value of the diameter same as that of the GHCC with the same number of nodes. For the cases when $N = 1029$, 1215 and 2500, the diameters of the GHC are less than those of the GHCC, but the corresponding values of δ_{min} are much larger than the δ values of the GHCC. Another important point to be noted (which is not apparent from the data in Table 4.1) is that, in between two consecutive values of N for which a hypercube or a star graph can be defined, we can construct $G(l,\ m)$'s with many possible values of N, by suitably choosing the values of l and m.

4.5 Routing

4.5.1 Point-to-Point Communication

To understand how node-to-node routing can be implemented, let us first introduce some terminologies.

Definition 4.1 *A binary operation* $\square$ *on the set of nodes in the graph* $G(l, m)$ *is defined as follows:*

$b = (p_1; u) \square (p_2; v) = \{u_{l-1} \bullet v_{l-1},\ u_{l-2} \bullet v_{l-2},\ \cdots,\ u_1 \bullet v_1,\ u_0 \bullet v_0\}$
where, $(p_1; u) = (p_1; u_{l-1}\ u_{l-2} \cdots u_1\ u_0)$, *and* $(p_2; v) = (p_2; v_{l-1}\ v_{l-2} \cdots v_1\ v_0)$ *are two nodes in the graph and b is an ordered binary sequence of length l, such that*

$$u_i \bullet v_i = \begin{cases} 1, & \text{if } u_i \neq v_i \\ 0, & \text{otherwise} \end{cases}$$

Example 4.1 *In* $G(8, 3)$, $(0; 2596) \square (4; 2302) = (0; 10120011) \square (4; 10011021)$
$= \{1 \bullet 1, 0 \bullet 0, 1 \bullet 0, 2 \bullet 1, 0 \bullet 1, 0 \bullet 0, 1 \bullet 2, 1 \bullet 1\} = \{00111010\}$

It may be noted that '$\bullet$' operates on two m-ary digits only. Since the binary string b does not depend on the level numbers of the two nodes, for brevity, we shall denote b by just $u \square v$ instead of $(p_1; u) \square (p_2; v)$ in our later discussions.

Table 4.1: Comparison of $GHCC(l,m)$ with different topologies

$GHCC(l,m)$ $N = lm^l$ $\delta = m+1$, (with $l > 2$) $D = \begin{cases} \lfloor \frac{5l}{2} \rfloor - 2, & l \geq 4; \\ \lfloor \frac{5l}{2} \rfloor - 1, & \text{otherwise} \end{cases}$		Hypercube $N = 2^n \leq lm^l$ $\delta = n$ $D = n$	Star graph $N = n! \leq lm^l$ $\delta = n-1$ $D = \lfloor \frac{3(n-1)}{2} \rfloor$	GHC $N = f_1 * f_2 * \cdots * f_k$ $\delta = \sum f_i - k$ $D = k$
$< l, m >$	$< N, \delta, D >$	$< N, \delta, D >$	$< N, \delta, D >$	$< N, \delta_{min}, D >$ $< N, \delta, D >$
$< 3, 4 >$	$< 192, 5, 6 >$	$< 128, 7, 7 >$	$< 120, 4, 6 >$	$< 192, 8, 7 >$ $< 192, 9, 6 >$
$< 3, 6 >$	$< 648, 7, 6 >$	$< 512, 9, 9 >$	$< 120, 4, 6 >$	$< 648, 11, 7 >$ $< 648, 12, 6 >$
$< 4, 4 >$	$< 2^{10}, 5, 8 >$	$< 2^{10}, 10, 10 >$	$< 720, 5, 7 >$	$< 2^{10}, 10, 10 >$ $< 2^{10}, 12, 8 >$
$< 3, 7 >$	$< 1029, 8, 6 >$	$< 1024, 10, 10 >$	$< 720, 5, 7 >$	$< 1029, 20, 4 >$
$< 5, 3 >$	$< 1215, 4, 10 >$	$< 1024, 10, 10 >$	$< 720, 5, 7 >$	$< 1215, 14, 6 >$
$< 3, 8 >$	$< 1536, 9, 6 >$	$< 1024, 10, 10 >$	$< 720, 5, 7 >$	$< 1536, 11, 10 >$ $< 1536, 15, 6 >$
$< 3, 9 >$	$< 2187, 10, 6 >$	$< 2048, 11, 11 >$	$< 720, 5, 7 >$	$< 2187, 14, 7 >$ $< 2187, 18, 6 >$
$< 4, 5 >$	$< 2500, 6, 8 >$	$< 2048, 11, 11 >$	$< 720, 5, 7 >$	$< 2500, 18, 6 >$
$< 4, 6 >$	$< 5184, 7, 8 >$	$< 2^{12}, 12, 12 >$	$< 5040, 6, 9 >$	$< 5184, 14, 10 >$ $< 5184, 16, 8 >$
$< 5, 10 >$	$< 5 * 10^5, 11, 10 >$	$< 2^{18}, 18, 18 >$	$< 9!, 8, 12 >$	$< 2^5 * 5^6, 29, 11 >$ $< 2^3 * 4 * 5^6, 30, 10 >$

Definition 4.2 *Index difference of two nodes* (p_1, u) *and* (p_2, v) *is defined as the number of non-zero bits in* $u \square v$ *and will be denoted by* $I_d(u, v)$.

Observation : From the connection pattern, it follows that if the j^{th} bit of $u \square v$ is 1, then all paths between $(p_1; u)$ and $(p_2; v)$ must pass through some nodes in the level j, $0 \leq j \leq l-1$. Moreover, the number of such levels that must be visited while traversing from $(p_1; u)$ to $(p_2; v)$ is equal to $I_d(u, v)$. Let us call these levels along with the source and the destination levels as the *necessary-visit-levels* (NVL) for two nodes having index values u and v.

Since the graph is node symmetric, without loss of generality, let us take the source node $s = (0; u) = (0; u_{l-1}u_{l-2} \cdots u_0)$ and the destination node $d = (p; v) = (p; v_{l-1}v_{l-2} \cdots v_1v_0)$. We establish a path from s to d by performing the following sequence of operations successively :

S1. Traverse through an appropriate set of nodes so that the node index is modified from u to v (by changing the value of each literal from u_i to v_i $\forall i$, $0 \leq i \leq l-1$). During this traversal, we positively need to visit the level i if $v_i \neq u_i$. As a result of these operations, we will ultimately reach some node $(p'; v)$, $0 \leq p' \leq l-1$. This would need traversal through some interlevel (cyclic) links and also intralevel (clique) links.

S2. Move through some more nodes to change the level number from p' to p, if required, so as to reach the node (p, v). To effect this, we need to traverse only through some interlevel (cyclic) links.

If L denotes the length of a path from $(0; u)$ to $(p; v)$, then L can be expressed as $L_q + L_c$; where, L_q is the number of intralevel links to be traversed and L_c is that of interlevel links. Given the source and the destination nodes, L_q is fixed and it is equal to $I_d(u; v)$. Thus the problem of minimizing L is now reduced to minimization of L_c. We now formulate the problem of finding the shortest path between $(0; u)$ and $(p; v)$ as follows :

(A) Let $p_0, p_1, \cdots, p_{r-1}$ be the NVLs (not necessarily consecutive levels) including the source and the destination levels, with the ordering $p_0 < p_1 < \cdots < p_{r-1}$. Construct a ring of r nodes $V_0, V_1, \cdots, V_{r-1}$, where,

(i) the node V_i, $0 \leq i \leq r-1$, corresponds to the NVL p_i.

(ii) two nodes V_i and V_{i+1}, $0 \leq i < r-1$, are connected by a weighted edge of weight $w_i = (p_{i+1} - p_i)$. V_{r-1} and V_0 are connected by an edge of weight $l - p_{r-1}$.

Since we have assumed that the source node is at level 0, V_0 corresponds to the source level. Let V_t be the node corresponding to the destination level p, i.e., $p_t = p$. Note that, V_t may also be equal to V_0. The structure of the ring so constructed is shown in Fig. 4.2.

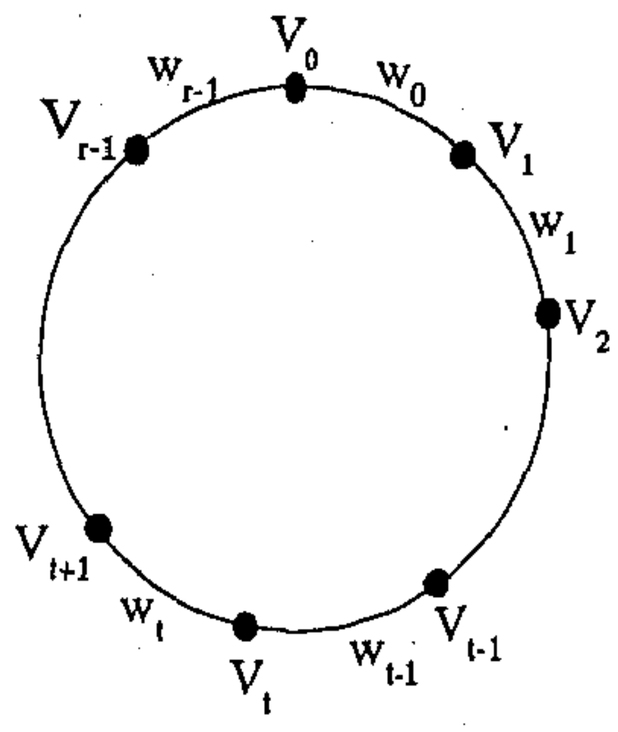


Figure 4.2: The ring constructed with the nodes corresponding to NVLs

(B) Starting from V_0, visit all the nodes in the ring at least once so as to finally reach V_t in the minimum possible distance.

For example, let us consider the ring shown in Fig. 4.3, where, V_3 is the destination level. Here, $V_0\ V_1\ V_2\ V_1\ V_0\ V_4\ V_3$ is the sequence for traversal in a minimum distance. The corresponding minimum distance is 7.

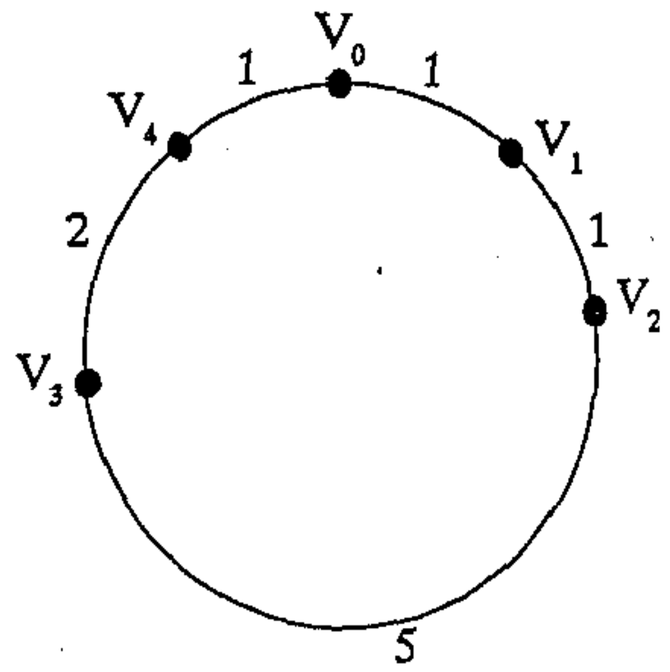


Figure 4.3: Shortest route $V_0\ V_1\ V_2\ V_1\ V_0\ V_4\ V_3$

For the traversal from V_0 to V_t in the minimum distance, we note the following points :

a) Some of the edges may be visited twice.

b) As we have to visit all the nodes in the ring, at most one edge may remain unvisited.

c) For $V_0 \neq V_t$, we must not traverse all the edges for the shortest distance. Hence, in this case, exactly one edge will be excluded while traversing through the shortest route. However, for $V_0 = V_t$, the shortest distance may correspond to the traversal through all the edges.

Let W_j denote the sum of the edge weights in a route, which excludes the edge (V_j, V_{j+1}). Let us first consider that $0 \leq j \leq t-1$. In this case, the possible route excluding the edge (V_j, V_{j+1}) is to traverse in the clockwise direction starting from V_0 upto the node V_j. Then from V_j start traversing in the anti-clockwise direction until we reach V_{j+1} and from V_{j+1} go back to the node V_t. The route is shown in Fig. 4.4. The distance traversed in this route would be .

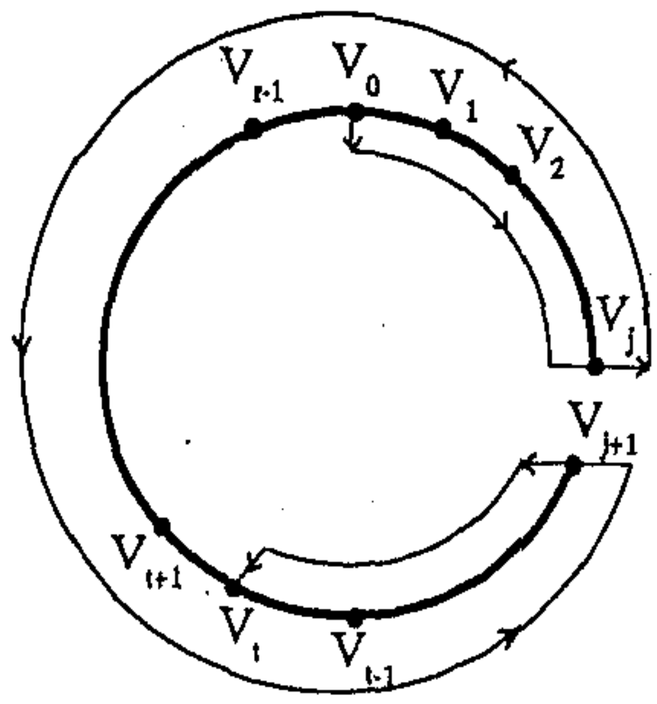


Figure 4.4: The path which does not contain the edge (V_j, V_{j+1})

$$W_j = 2\sum_{\substack{i=0 \\ i\neq j}}^{t-1} w_i + \sum_{i=t}^{r-1} w_i = l - w_j + \sum_{\substack{i=0, \\ i\neq j}}^{t-1} w_i \tag{4.1}$$

Among t such W_j values $(0 \leq j \leq t-1)$, the smallest one will be W_s if $w_s = \max\{w_0, w_1, \cdots, w_{t-1}\}$.

When $t \leq j \leq r-1$, in a similar way it can be shown that the length of the shortest route will correspond to $j = \hat{s}$, where, $w_{\hat{s}} = \max\{w_t, w_{t+1}, \cdots, w_{r-1}\}$ and is given by

$$W_{\hat{s}} = \sum_{i=0}^{t-1} w_i + 2\sum_{\substack{i=t, \\ i\neq \hat{s}}}^{r-1} w_i = l - w_{\hat{s}} + \sum_{\substack{i=t, \\ i\neq \hat{s}}}^{r-1} w_i \tag{4.2}$$

Let, $W_{min} = min\,(W_s,\ W_{\hat{s}})$.

Hence, the minimum value of L_c is given by,

$$min\,(L_c) = \begin{cases} W_{min}, & \text{if } V_0 \neq V_t; \\ min(W_{min}, l), & \text{if } V_0 = V_t. \end{cases}$$

Finally, let us formally describe the algorithm for finding the shortest path.

Algorithm *ShortestPath*

Input : Source $s = (0; u) = (0; u_{l-1}u_{l-2}\cdots u_2u_1u_0)$ and the destination $d = (p; v) = (p; v_{l-1}v_{l-2}\cdots v_2v_1v_0)$

Output : The shortest path between s and d.

Step 1 : By computing $u\Box v$, identify the necessary-visit-levels and also find the value of L_q.

Step 2 : Construct the ring $(V_0, V_1, \cdots, V_{r-1})$ as stated above.

Step 3 : Find $w_s = \max\{w_0, w_1, \cdots, w_{t-1}\}$ and $w_{\hat{s}} = \max\{w_t, w_{t+1}, \cdots, w_{\hat{s}}\}$. Compute W_s, $W_{\hat{s}}$ and $W_{min} = min(W_s, W_{\hat{s}})$ as well as $min(L_c)$.

Step 4 : Identify the path P_{min} in terms of V_0, V_1, $\cdots$, V_{r-1} corresponding to $min(L_c)$ found in step 3.

Step 5 : Corresponding to the path P_{min}, find the path between s and d in terms of the nodes in the original topology. □

If the source level is different from level 0, then we can suitably renumber the levels and then execute the same algorithm. The renumbering scheme is as follows. Suppose $p_1\ (> 0)$ is the source level. Then any level p, $0 \leq p \leq l-1$, should be renumbered as $(p - p_1)\ mod\ l$.

Complexity : Step 1 requires bit operations over a string of length l. In step 2, while constructing the ring, the weights of the edges can also be computed from the string $u\Box v$ of length l. Thus, these two steps require $O(l)$ time. Step 3 takes at most $O(l)$ time. Each of the steps 4 and 5 require at most $O(l)$ time. Hence, the time complexity of this algorithm is $O(l)$.

Let us illustrate the algorithm with an example.

Example 4.2 *Suppose, we want to find the shortest path between the source node* $s = (0; 2596)$ *and the destination node* $d = (4; 2302)$ *in* $G(8, 3)$. *The*

index 2596 of the source node can be equivalently represented in radix-3 system (corresponding to m = 3) as 10120011 and that of the destination node can be represented as 10011021.

1) 2596 □ 2302 = *00111010 (refer to example 4.1).*
The necessary-visit-levels are level 0 (source level), level 4 (destination level) and also the levels 1, 3 and 5 corresponding to the positions of '1' in 2596 □ 2302. $L_q = 4$.

2) A ring with 5 nodes V_0, V_1, V_2, V_3, *and* V_4 *is constructed as shown in the Fig. 4.5. The nodes* V_0, V_1, V_2, V_3, *and* V_4 *correspond to the levels 0, 1, 3, 4 and 5 respectively. The weights assigned to the edges are as follows :*
$w_0 = 1$, $w_1 = 2$ *(for two cyclic edges from level 1 to level 3),* $w_2 = 1$, $w_3 = 1$ *and* $w_4 = 3$ *(for three cyclic edges from level 5 to level 0).*

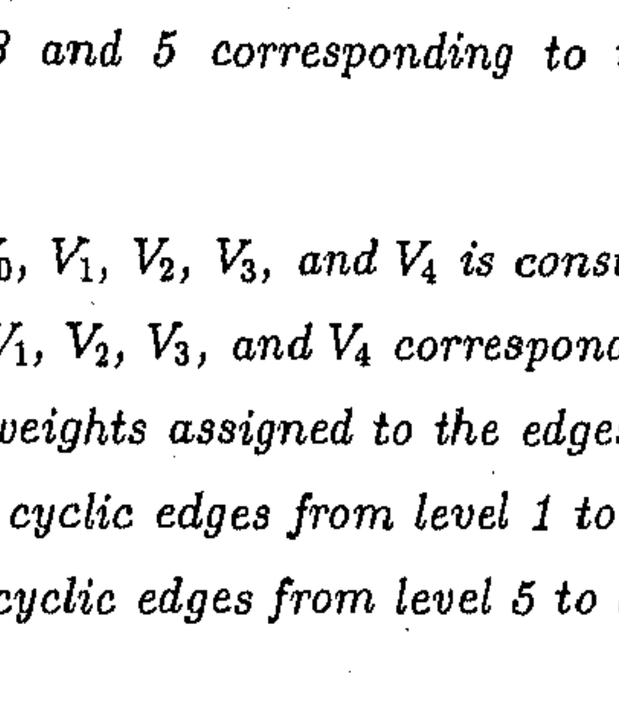


Figure 4.5: The ring constructed for the example 4.2

3) $w_s = max\{w_0, w_1, w_2\} = max\ \{1, 2, 1\} = 2 = w_1$;
$w_{\hat{s}} = max\{w_3, w_4\} = max\ \{1, 3\} = 3 = w_4$.

$W_s = 8 - w_1 + (w_0 + w_2) = 8$ *(from equation 4.1) and* $W_{\hat{s}} = 8 - w_4 + w_3 = 6$ *(from equation 4.2).*
Hence $W_{min} = W_{\hat{s}}$. *Thus,* $min(L_c) = min(8, 6) = 6$.

4) The path P_{min} in the ring corresponding to $W_{\hat{s}}$ is $V_0V_1V_2V_3V_4V_3$ (excluding the edge (V_4, V_0) of weight $w_{\hat{s}} = w_4$).

5) The corresponding shortest path between $(0; 10120011)$ *and* $(4; 10011021)$ *in* $GHCC(8, 3)$ *is*

$s = (0; 10120011) \rightarrow (1; 10120011) \rightarrow (1; 10120021) \rightarrow (2; 10120021) \rightarrow (3; 10120021)$ $\rightarrow (3; 10121021) \rightarrow (4; 10121021) \rightarrow (4; 10111021) \rightarrow (5; 10111021) \rightarrow (5; 10011021)$ $\rightarrow (4; 10011021) = d$. *The shortest distance between these two nodes is given by* $min\ (L_c) + L_q = 6 + 4 = 10$.

4.5.2 One-to-all Broadcast

The connection pattern of $G(l, m)$ shows that it contains m^l node-disjoint cycles, each of length l, so that every node in the topology belongs to one such cycle. These m^l cycles thus cover all the nodes in the topology. If the source node $(p; v)$ could somehow manage to send the message to at least one node in each of these cycles, then the remaining task of transmitting the message to all other nodes in that cycle can be performed in a straight-forward way. During transmission, if at some stage we find that all the nodes in a particular level have got the message, then it is sufficient to say that there exists at least one node in each of these cycles which has already received the message.

The broadcast algorithm which we are going to present consists of two phases. The phase 1 is designed in such a way that by the end of this phase the message would reach all the nodes in the level $(p + 1)$. This guarantees that by the end of this phase each of the m^l cycles would contain at least one node having the message. In each cycle, the nodes which have already received the message in phase 1, would act as source nodes in phase 2. These sources would transmit the message through cyclic edges only, so that all the nodes in that particular cycle can receive the message. Throughout the algorithm, we assume that none of the processors would transmit the message more than once in a particular direction. Moreover, here we assume a single port model, i.e., at a time a processor can send

the message only through one of its links.

The basic step for phase 1 can be described by the following two actions :

[A1] Transmit the message through the cyclic edge in the direction of decreasing level numbers only (anti-clockwise direction). We represent this action by A_c.

[A2] Broadcast the message over the clique which the node belongs to. We represent this action by A_q.

Phase 1 can now be completely described as follows.

Phase 1

Step 1 : The source (p, v) broadcasts the message in parallel to all nodes in the clique which it belongs to.

for $i = 2$ **to** l **do**

/* Steps i(a) and i(b) constitute the basic step of phase 1 as mentioned above. */

Step i(a) : if $i = 2$ **then** the source node and all other nodes which have received the message in step 1, perform A_c in parallel.
else all nodes which have received the message in step $(i-1)$, perform A_c in parallel.

Step i(b) : all nodes which have received the message in step i(a), perform A_q in parallel. □

Step 1 requires $\lceil \log_2 m \rceil$ parallel steps. Step i(a) takes one unit of time, whereas step i(b) requires $\lceil \log_2 m \rceil$ parallel steps. Hence, $l\lceil \log_2 m \rceil + l - 1$ time units are required to complete phase 1.

It follows that after the i^{th} step, $\forall i, 1 \leq i \leq l$, in phase 1, all the members of m^{i-1} cliques in level $(p - i + 1) \bmod l$ would receive the message. Thus, by the end of phase 1, all the members of m^{l-1} cliques in the level $p + 1$ would receive the

message. That is, at the end of this phase all the nodes in the level $p+1$ would receive the message which, in turn, guarantees that for each cycle there will be at least one node which would receive the message at the end of phase 1. Since the transmission through cyclic edges are done only in a particular direction, none of the processors would receive the message more than once in phase 1. We illustrate this phase by the following example.

Example 4.3 *Let us assume that the source node is* $(0;000\cdots00)$. *In step 1, the source node* $(0;000\cdots00)$ *will broadcast the message to the nodes* $(0;00\cdots0*)$ *where * can take any value from* $\{0,1,2,\cdots,m-1\}$. *After this step, one node in each of the cycles* $C_{(00\cdots0*)}$ *would get the message. In step 2(a), these nodes in* $C_{(00\cdots0*)}$ *will transmit the message to their neighbors in the level* $l-1$. *Hence after step 2, exactly one node in each of the cycles* $C_{(*0\cdots0*)}$ *excepting* $C_{(00\cdots0*)}$ *would get this message, while in each of the cycles* $C_{(00\cdots0*)}$ *there will be two nodes who would receive the message.*

The node which receives the message first among all the nodes in any particular cycle will be termed as the *entry-point* corresponding to that cycle.

Referring to the example 4.3, consider two particular cycles say $C_{(00\cdots01)}$ and $C_{(100\cdots02)}$. In $C_{(00\cdots01)}$, the node $(0;00\cdots01)$ will get the message first and after step 2(a) there will be one more node, namely $(l-1;00\cdots01)$ in the same cycle, which will also get the message. On the other hand the node $(l-1;100\cdots02)$ will be the first node in the cycle $C_{(100\cdots02)}$, receiving the message after step 2(b). Thus, the nodes $(0;000\cdots01)$ and $(l-1;100\cdots02)$ are the entry-points of $C_{(000\cdots01)}$ and $C_{(100\cdots02)}$ respectively. The corresponding level of the entry-point will be termed as the *entry-level*. That is, the levels 0 and $l-1$ will be the entry-levels for the cycles $C_{(000\cdots01)}$ and $C_{(100\cdots02)}$ respectively. The following lemma follows from the description of phase 1.

Lemma 4.2 *If* $(p;v)$ *is the source node and the entry-level corresponding to a particular cycle* $C_{(u_{l-1}u_{l-2}\cdots u_1u_0)}$ *is* x, *then all the nodes in* $C_{(u_{l-1}u_{l-2}\cdots u_1u_0)}$ *having*

level numbers $x-1$, $x-2$, $\cdots$, $p+1$ *would receive the message after phase 1.*

Proof : The entry-point of the cycle $C_{(u_{l-1}u_{l-2}\cdots u_1u_0)}$ is $(x;\ u_{l-1}u_{l-2}\cdots u_1u_0)$. Let us assume that this entry-point receives the message at the i^{th} step of phase 1. It follows from the above discussion that $x = (p-i+1)\ mod\ l$ and after the i^{th} step, the message will propagate along the cycle $C_{(u_{l-1}u_{l-2}\cdots u_1u_0)}$ till the end of phase 1. There will be $l-i$ remaining steps in this phase and during these steps, the level numbers of the nodes in this cycle receiving the message, will be decreasing. Therefore, the level number of the node which will receive the message at the last step of phase 1, i.e., at the step l, will be $\{x-(l-i)\}\ mod\ l = (p+1-l)\ mod\ l$ $= p+1$. Hence the proof. □

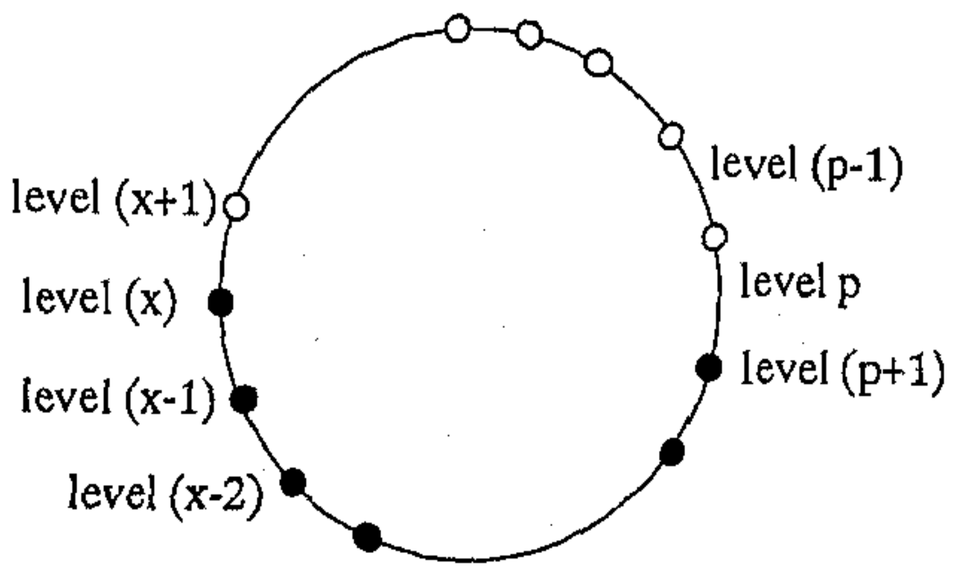


Figure 4.6: Nodes in $C_{u_{l-1}\cdots u_0}$ receiving the message (*solid circles*) in phase 1

Fig. 4.6 shows the situation after phase 1 corresponding to the cycle $C_{u_{l-1}u_{l-2}\cdots u_1u_0}$, where solid circles represent those nodes which receive the message in the first phase.

In phase 2, transmission will be done through cyclic edges only and the message should be sent to the nodes in this cycle which lie in the levels $x+1$, $x+2$, $\cdots$, p. For this, the nodes $(x; u_{l-1}u_{l-2}\cdots u_1u_0)$ and $(p+1; u_{l-1}u_{l-2}\cdots u_1u_0)$ start sending the message in clockwise and anti-clockwise directions respectively along the cycle $C_{u_{l-1}u_{l-2}\cdots,u_1,u_0}$. The number of nodes in this cycle which are going to receive the

message in phase 2 is $i-1$, if the entry-point of the cycle receives the message in the i^{th} step. Thus, after $\lceil \frac{i-1}{2} \rceil$ steps all the nodes in $C_{u_{l-1}u_{l-2}\cdots u_1,u_0}$ will get the message. In the worst case, i takes the value l. In that case, the entry-level of the cycle will be $p+1$ and $(l-1)$ nodes in the cycle are going to receive the message in phase 2. Thus, the required number of steps is $\lceil \frac{l-1}{2} \rceil + 1$. Hence, phase 2 takes at most $\lceil \frac{l-1}{2} \rceil + 1$ steps.

During transmission, the message will be tagged with four extra fields : (1) SOURCE LEVEL (**p**) field (2) WEIGHT (**w**) field (3) PHASE (**t**) field and (4) ENTRY (**en**) field.

PHASE field (t) : It is a one-bit tag which can assume only two values 0 and 1. The value of t is 0 if the message is currently passing through the phase 1, and 1 otherwise.

ENTRY field (en) : This field of the message at any node indicates the entry-level of the cycle containing that node. Thus en ranges from 0 to $l-1$, both inclusive. If a processor receives the message through a clique edge, then this processor will be the first one receiving the message in the cycle where it belongs to, i.e., this receiving processor will be the entry point of the corresponding cycle. In such a case, the ENTRY field of the received message will be changed to the level number of the receiving processor. Otherwise, the ENTRY field would remain unchanged.

WEIGHT field (w) : w keeps track of the number of remaining cyclic edges through which the message is to be transmitted during a particular phase. When the message is transmitted through a cyclic edge, the value of w is always reduced by 1. On the other hand, in case of transmission through the clique edges, the value of this field is kept as it is. Before starting phase 1, the value of w is initialized to $l-1$. As a consequence, w will take the value 0 at the end of the phase 1. At the beginning of the second phase, w will again be set to the value $\lceil \frac{p-en}{2} \rceil$, if $p \geq en$ or $\lceil \frac{l-(en-p)}{2} \rceil$, if $p < en$, so that at the end of this phase as well, w would reduce to 0. Thus the value of w may be used for terminating the algorithm.

We now formally describe the algorithm as follows :

Algorithm *Broadcast*

/* Initial Steps by the source processor */

1) The source $(p; v)$ broadcasts the message to all of its neighbors in the level p with $w = l - 1$, $t = 0$, and $en = p$;

2) $(p; v)$ also transmits the message through the cyclic edge to its neighbor in the level $p - 1$ with $w = l - 2$, $t = 0$, and $en = p$;

/* Job for the processor $(q; u)$ */

3) **if** $(q; u)$ receives the message with four extra fields containing the values p, w, t and en **then**

{

if $t = 0$ **then**

{

if $w \neq 0$ **then**

{

if $(q; u)$ receives the message through a clique edge **then**

{

$en = q$;

send the message to $(q - 1; u)$ with weight $w - 1$, through a cyclic edge in anti-clockwise direction;

/* initialize w and t for starting phase 2 */

if $p < en$ **then**

$w = \lceil \frac{l - en + p}{2} \rceil$

else $w = \lceil \frac{p - en}{2} \rceil$;

$t = 1$;

send the message to $(q + 1; u)$ with weight $w - 1$ through the cyclic edge in clockwise direction;

}

if $(q;u)$ receives the message through a cyclic edge **then**
{
send the message with the same weight to all its neighbors connected through clique edges;
send the message to the node $(q-1;u)$ with weight $w-1$;
}
}
if $w = 0$ **then** /* The node receiving the message is at level $p+1$ */
{
if $(q;u)$ receives the message through a cyclic edge **then**
{
send the message to all of its neighbors connected through clique edges;
/* initialize w and t for starting phase 2 */
if $p < en$ **then**
$w = \lfloor \frac{l-en+p}{2} \rfloor$;
else $w = \lfloor \frac{p-en}{2} \rfloor$;
$t = 1$;
if $w > 0$ then send the message to $(q-1;u)$ through the cyclic edge in anti-clockwise direction with weight $w-1$;
/* This node at level $p+1$ transmits the message in anti-clockwise direction in phase 2 */
}
if $(q;u)$ receives the message through a clique edge **then**
{
$en = q$; /* Here, $q = p+1$ */
$t = 1$;
$w = \lceil \frac{l-en+p}{2} \rceil$;
send the message to $(q+1;u)$ through the cyclic edge in clockwise direction with weight $w-1$;
$w = \lfloor \frac{l-en+p}{2} \rfloor$;
if $w > 0$ then send the message to $(q-1;u)$ through the cyclic edge in

anti-clockwise direction with weight $w-1$;

}

}

}

if $t = 1$ **then**

{

if $w = 0$ **then**

terminate;

else

{

forward the message with weight $w-1$ along a cyclic edge in the same direction (clockwise or anti-clockwise);

}

}

}

Time Complexity : The time taken to complete phase 1 is $l\lceil \log_2 m \rceil + l - 1$ and phase 2 requires at most $\lceil \frac{l}{2} \rceil$ steps. So, in total, the time required to broadcast a message in the $GHCC(l, m)$ is $l\lceil \log_2 m \rceil + \lceil \frac{3l}{2} \rceil$ - 1.

Example 4.4 *Let us illustrate phase 1 of the above algorithm for the structure* $G(3;3)$. *Suppose we want to broadcast a message from the processor (2; 000) to all other processors in the network. In this network the cliques are of size 3. Thus, two steps are required to broadcast the message over a clique.*

Step 1 : *(transmission over a clique)*

(i) (2; 000) $\rightarrow$ *(2; 100) with (w = 2, en = 2, t = 0)*

(ii) (2; 000) $\rightarrow$ *(2; 200) with (w = 2, en = 2, t = 0)*

Step 2(a) : *(transmission through cyclic edges)*

(2; 000) $\rightarrow$ *(1; 000) with (w = 1, en = 2, t = 0)*

(2; 100) $\rightarrow$ *(1; 100) with (w = 1, en = 2, t = 0)*

(2; 200) → (1; 200) with (w = 1, en = 2, t = 0)

Step 2(b) : *(two parallel steps of transmission over cliques)*

(i) (1; 000) → (1; 010) with (w = 1, en = 2, t = 0)

(1; 100) → (1; 110) with (w = 1, en = 2, t = 0)

(1; 200) → (1; 210) with (w = 1, en = 2, t = 0)

(ii) (1; 000) → (1; 020) with (w = 1, en = 2, t = 0)

(1; 100) → (1; 120) with (w = 1, en = 2, t = 0)

(1; 200) → (1; 220) with (w = 1, en = 2, t = 0)

Step 3(a) : *(transmission through cyclic edges)*

(1; 000) → (0; 010) with (w = 0, en = 2, t = 0)

(1; 100) → (0; 100) with (w = 0, en = 2, t = 0)

(1; 200) → (0; 200) with (w = 0, en = 2, t = 0)

(1; 010) → (0; 010) with (w = 0, en = 1, t = 0)

(1; 110) → (0; 110) with (w = 0, en = 1, t = 0)

(1; 210) → (0; 210) with (w = 0, en = 1, t = 0)

(1; 020) → (0; 020) with (w = 0, en = 1, t = 0)

(1; 120) → (0; 120) with (w = 0, en = 1, t = 0)

(1; 220) → (0; 220) with (w = 0, en = 1, t = 0)

Step 3(b) : *(two parallel steps of transmission over cliques)*

(i) (1; 000) → (0; 001) with (w = 0, en = 2, t = 0)

(1; 100) → (0; 101) with (w = 0, en = 2, t = 0)

(1; 200) → (0; 201) with (w = 0, en = 2, t = 0)

(1; 010) → (0; 011) with (w = 0, en = 1, t = 0)

(1; 110) → (0; 111) with (w = 0, en = 1, t = 0)

(1; 210) → (0; 211) with (w = 0, en = 1, t = 0)

(1; 020) → (0; 021) with (w = 0, en = 1, t = 0)

(1; 120) → (0; 121) with (w = 0, en = 1, t = 0)

(1; 220) → (0; 221) with (w = 0, en = 1, t = 0)

(ii) (1; 000) → (0; 002) with (w = 0, en = 2, t = 0)

$(1;\ 100) \rightarrow (0;\ 102)$ *with* $(w = 0,\ en = 2,\ t = 0)$
$(1;\ 200) \rightarrow (0;\ 202)$ *with* $(w = 0,\ en = 2,\ t = 0)$
$(1;\ 010) \rightarrow (0;\ 012)$ *with* $(w = 0,\ en = 1,\ t = 0)$
$(1;\ 110) \rightarrow (0;\ 112)$ *with* $(w = 0,\ en = 1,\ t = 0)$
$(1;\ 210) \rightarrow (0;\ 212)$ *with* $(w = 0,\ en = 1,\ t = 0)$
$(1;\ 020) \rightarrow (0;\ 022)$ *with* $(w = 0,\ en = 1,\ t = 0)$
$(1;\ 120) \rightarrow (0;\ 122)$ *with* $(w = 0,\ en = 1,\ t = 0)$
$(1;\ 220) \rightarrow (0;\ 222)$ *with* $(w = 0,\ en = 1,\ t = 0)$

4.6 Connectivity

If $l = 1$, the $GHCC(1, m)$ is nothing but a clique of size m. A clique is $m - 1$ connected. We will discuss about the connectivity of the $GHCC(l, m)$ for $l = 2$ afterwards. Before that, we consider below the case for $l \geq 3$.

For $l \geq 3$, since the degree of each node in the GHCC is $m+1$, its node-connectivity is at most $m+1$. When $m = 1$, the GHCC is nothing but a ring with connectivity 2. When $m = 2$, the GHCC reduces to a CCC which is triconnected [PV81]. Thus we have the following result.

Lemma 4.3 *The GHCC is* $(m+1)$*-connected for* $m \leq 2$, $l \geq 3$.

For $m > 2$, we show below that between any two nodes of the GHCC, there exist $m + 1$ node-disjoint paths.

Because of the node symmetry of the GHCC, we can assume, without loss of generality, that the source node is $s = (0;\ 000\cdots00)$ and the destination node is $d = (p;\ v_{l-1}v_{l-2}\cdots v_1v_0)$, where $0 \leq p \leq l-1$ and v_j, $0 \leq j \leq l-1$, can take any value between 0 and $m - 1$ (both inclusive).

We shall first give a general construction rule for $m+1$ paths between s and d and then we shall explain why these paths are node-disjoint.

Since our goal is to construct $m+1$ node-disjoint paths from s to d, the only option is to start from s and visit all the $m+1$ distinct neighbors of s, one along each of these $m+1$ paths which we are going to construct. We refer to these $m+1$ paths as P_0, P_1, $\cdots$, P_m such that the path P_i, $1 \leq i \leq m-1$, visits the node $(0;\ 000\cdots0i)$ immediately after starting from the source s, and the paths P_m, P_0 visit the remaining two neighbors $(l-1;\ 000\cdots00)$ and $(1;\ 000\cdots00)$ respectively after starting from s. This scheme is illustrated in Fig. 4.7 for $G(4,5)$.

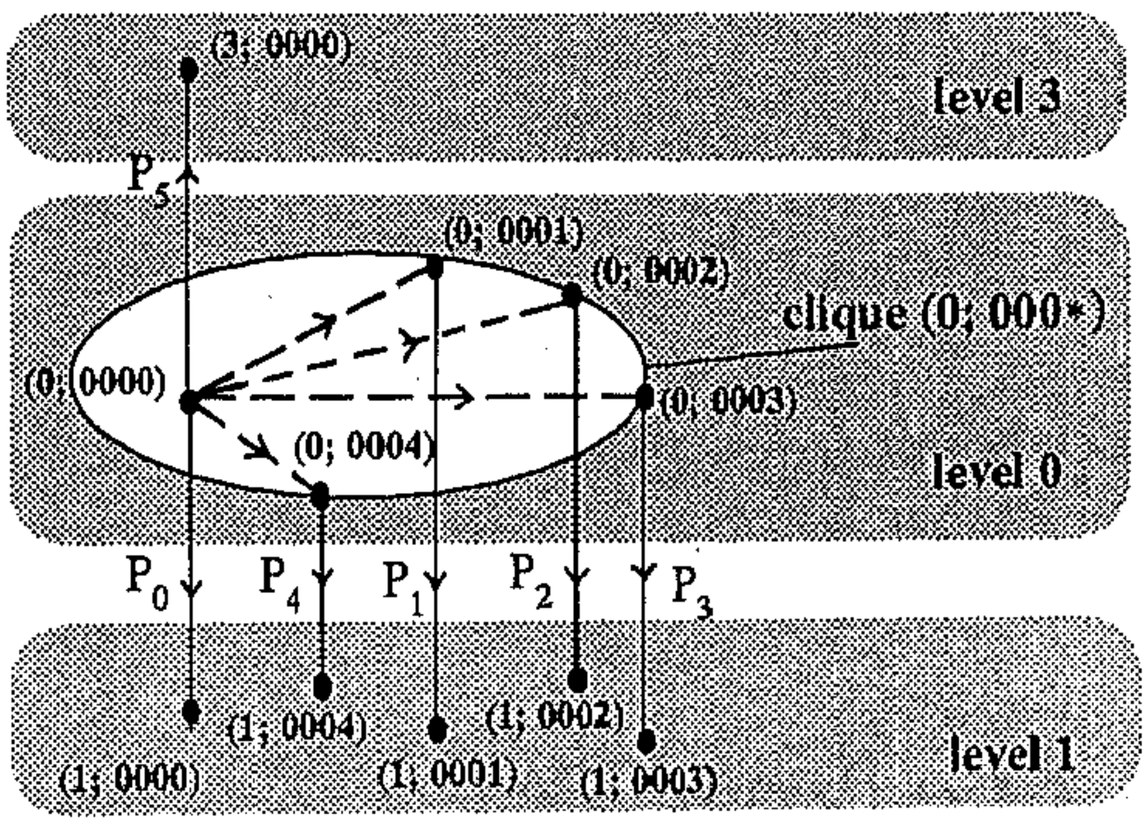


Figure 4.7: Initial portions of node-disjoint paths from (0; 0000) in $G(4,5)$

In our later discussion we shall often use a notation 0^i, while specifying a node index. This notation actually represents a string of i consecutive 0's.

Construction rule for $m+1$ node-disjoint paths :

In what follows, we use the symbols x_i, Q, R, Y and Z in constituting the paths which are defined as follows.

1) The symbol x_i's ($0 \leq i \leq m-1, i \neq v_0$) can take any value from 0 to $m-1$, excluding v_p and $x_i \neq x_j$ whenever $i \neq j$. To be more specific, let us define x_i as follows satisfying the above requirements :

$$x_i = \begin{cases} i+1, & \text{if } i \neq v_p - 1 \\ v_0 + 1, & \text{if } i = v_p - 1 \end{cases}$$

2) $0 \neq Y \neq v_{p-1}$.

3) $0 \neq Z \neq v_{p+1}$.

4) $0 \neq Q \neq v_{l-1}$.

5) $0 \neq R \neq v_1$.

Case I : $2 \leq p \leq l-2$.

Path P_i $(0 \leq i \leq m-1, i \neq v_0)$:

$s \to (0;\ 0^{l-1}i) \to$

$(1;\ 0^{l-1}i) \to (1;\ 0^{l-2}v_1 i) \to$

$(2;\ 0^{l-2}v_1 i) \to (2;\ 0^{l-3}v_2 v_1 i) \to$

$\cdots\cdots\cdots \to$

$(p-1;\ 0^{l-p+1}v_{p-2} \cdots v_2 v_1 i) \to (p-1;\ 0^{l-p}Yv_{p-2} \cdots v_2 v_1 i) \to$

$(p;\ 0^{l-p}Yv_{p-2} \cdots v_2 v_1 i) \to (p;\ 0^{l-p-1}x_i Yv_{p-2} \cdots v_2 v_1 i) \to$

$(p+1;\ 0^{l-p-1}x_i Yv_{p-2} \cdots v_2 v_1 i) \to (p+1;\ 0^{l-p-2}v_{p+1}x_i Yv_{p-2} \cdots v_2 v_1 i) \to$

$\cdots\cdots\cdots \to$

$(l-1;\ 0v_{l-2} \cdots v_{p+1}x_i Yv_{p-2} \cdots v_1 i) \to (l-1;\ v_{l-1} \cdots v_{p+1}x_i Yv_{p-1} \cdots v_1 v_0) \to$

$(0;\ v_{l-1} \cdots v_{p+1}x_i Yv_{p-2} \cdots v_1 i) \to (0;\ v_{l-1} \cdots v_{p+1}x_i Yv_{p-1} \cdots v_1 v_0) \to$

$(1;\ v_{l-1}v_{l-2} \cdots v_{p+1}x_i Yv_{p-2} \cdots v_2 v_1 v_0) \to$

$(2;\ v_{l-1}v_{l-2} \cdots v_{p+1}x_i Yv_{p-2} \cdots v_2 v_1 v_0) \to$

$\cdots\ \cdots \to$

$(p-1;\ v_{l-1} \cdots v_{p+1}x_i Yv_{p-2} \cdots v_1 v_0) \to (p-1;\ v_{l-1} \cdots v_{p+1}x_i v_{p-1}v_{p-2} \cdots v_1 v_0)$

$\to (p;\ v_{l-1} \cdots v_{p+1}x_i v_{p-1} \cdots v_1 v_0) \to (p;\ v_{l-1}v_{l-2} \cdots v_{p+1}v_p v_{p-1} \cdots v_1 v_0) = d$

Path P_{v_0} :

$s \to (0;\ 0^{l-1}v_0) \to$
$(1;\ 0^{l-1}v_0) \to (1;\ 0^{l-2}Rv_0) \to$
$(2;\ 0^{l-2}Rv_0) \to (2;\ 0^{l-3}v_2Rv_0) \to$
$\cdots\ \cdots\cdots \to$
$(p-1;\ 0^{l-p+1}v_{p-2}v_{p-3}\cdots\ v_2Rv_0) \to (p-1;\ 0^{l-p}Yv_{p-2}v_{p-3}\cdots v_2Rv_0) \to$
$(p;\ 0^{l-p}Yv_{p-2}\cdots\ v_2Rv_0) \to (p;\ 0^{l-p-1}v_pYv_{p-2}\cdots\ v_2Rv_0) \to$
$\cdots\ \cdots\cdots \to$
$(l-1;\ 0v_{l-2}v_{l-3}\cdots\ v_pYv_{p-2}\cdots\ v_2Rv_0) \to (l-1;\ v_{l-1}v_{l-2}\cdots\ v_pYv_{p-2}\cdots\ v_2Rv_0)$
$\to (0;\ v_{l-1}v_{l-2}\cdots\ v_pYv_{p-2}\cdots\ v_2Rv_0) \to$
$(1;\ v_{l-1}v_{l-2}\cdots\ v_pYv_{p-2}\cdots\ v_2Rv_0) \to (1;\ v_{l-1}v_{l-2}\cdots\ v_pYv_{p-2}\cdots\ v_2v_1v_0) \to$
$\cdots\ \cdots\cdots \to$
$(p-1;\ v_{l-1}v_{l-2}\cdots\ v_pYv_{p-2}\cdots\ v_1v_0) \to (p-1;\ v_{l-1}v_{l-2}\cdots\ v_pv_{p-1}v_{p-2}\cdots\ v_1v_0) \to$
$(p;\ v_{l-1}v_{l-2}\cdots\ v_2v_1v_0) = d$

Path P_m :

$s \to (l-1;\ 0^l) \to (l-1;\ v_{l-1}0^{l-1}) \to$
$(l-2;\ v_{l-1}0^{l-1}) \to (l-2;\ v_{l-1}v_{l-2}0^{l-2}) \to$
$\cdots\ \cdots\cdots \to$
$(p+1;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}0^{p+2}) \to (p+1;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Z0^{p+1}) \to$
$(p;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Z0^{p+1}) \to (p;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Zv_p0^p) \to$
$\cdots\cdots\cdots \to$
$(0;\ v_{l-1}\cdots\ v_{p+2}Zv_p\cdots\ v_10) \to (0;\ v_{l-1}\cdots\ v_{p+2}Zv_p\cdots\ v_2v_1v_0) \to$
$(1;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Zv_pv_{p-1}\cdots v_1v_0) \to$
$(2;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Zv_pv_{p-1}\cdots\ v_1v_0) \to$
$\cdots\ \cdots\cdots \to$
$(p;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Zv_pv_{p-1}\cdots\ v_0) \to$
$(p+1;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}Zv_pv_{p-1}\cdots\ v_0) \to (p+1;\ v_{l-1}v_{l-2}\cdots\ v_{p+2}v_{p+1}v_p\cdots\ v_0) \to$
$(p;\ v_{l-1}v_{l-2}\cdots\ v_2v_1v_0) = d$

From the above construction rule, we see that following the path P_i, $0 \leq i \leq$

$m-1$, $i \neq v_0$, when we visit the destination level p for the first time, the p^{th} literal is intentionally set to some incorrect value (i.e., other than v_p) x_i so that finally when this literal is set to v_p, we would reach the destination d through a clique edge. On the other hand, in both P_0 and P_m the p^{th} literal is set to v_p but the $(p-1)^{\text{th}}$ literal in P_{v_0} and the $(p+1)^{\text{th}}$ literal in P_m are set incorrectly, while passing through the corresponding levels for the first time. As a result, after setting these literals to their correct values during the second visit to these levels, these paths can enter the destination d directly from the level $p-1$ or $p+1$ through the cyclic edges. We refer to all the paths which enter d through clique edges as of type I; whereas the paths entering d through cyclic edges from levels $p-1$ and $p+1$ are termed as of type II(a) and type II(b) respectively.

Lemma 4.4 *The paths specified by the above construction rules are node-disjoint.*

Proof : The different nodes traversed by each of these paths in the levels 0, k $(1 \leq k \leq p-2)$, $p-1$, p, $p+1$ and r $(p+2 \leq r \leq l-1)$ are listed in the table 4.2. From this table it follows that these paths are node-disjoint due to the restrictions imposed on the values of R, Y, Z and x_i's. □

The length of the paths constructed above are computed as follows.

$$L(P_i) \leq \begin{cases} 2l+p+2, & \text{if } i = 0,\ v_0 \\ 2l+p+3, & \text{if } 1 \leq i \leq m,\ i \neq v_0 \end{cases}$$

Thus, the maximum path length is at most $2l+p+3$. If $p \leq \lfloor \frac{l}{2} \rfloor$, the maximum path length will be bounded by $\lfloor \frac{5l}{2} \rfloor + 3$ which is greater than the diameter by at most 5. If $p > \lfloor \frac{l}{2} \rfloor$, we will construct these node-disjoint paths just in the reverse direction keeping the basic idea unchanged. In that case, the maximum path length will be at most $2l + (l-p) + 3$ which is bounded by $\lfloor \frac{5l}{2} \rfloor + 3$ since $p > \lfloor \frac{l}{2} \rfloor$. Hence, we get the following lemma.

Lemma 4.5 *The maximum length of the paths specified by the above rules is* $\lfloor \frac{5l}{2} \rfloor + 3$.

Case II : When the source and the destination levels are same, i.e., $p = 0$.

In this case $s = (0;\ 000 \cdots 00)$ and $d = (0;\ v_{l-1}v_{l-2} \cdots v_1v_0)$. The construction rules are as follows :

Path P_i $(0 \leq i \leq m-1, i \neq v_0)$:

$s \rightarrow (0;\ 0^{l-1}i) \rightarrow$
$(1;\ 0^{l-1}i) \rightarrow (1;\ 0^{l-2}Zi) \rightarrow$
$(2;\ 0^{l-2}Zi) \rightarrow (2;\ 0^{l-3}v_2Zi) \rightarrow$
$(3;\ 0^{l-3}v_2Zi) \rightarrow (3;\ 0^{l-4}v_3v_2Zi) \rightarrow$
$\cdots\cdots\cdots \rightarrow$
$(l-1;\ 0v_{l-2}v_{l-3} \cdots v_2Zi) \rightarrow (l-1;\ v_{l-1} \cdots v_2Zi) \rightarrow$
$(0;\ v_{l-1} \cdots v_2Zi) \rightarrow$
$(1;\ v_{l-1} \cdots v_2Zi) \rightarrow (1;\ v_{l-1} \cdots v_2v_1i) \rightarrow$
$(0;\ v_{l-1}v_{l-2} \cdots v_2v_1i) \rightarrow (0;\ v_{l-1}v_{l-2} \cdots v_1v_0) = d$

These paths are of type I.

Path P_{v_0} :

$s \rightarrow (0;\ 0^{l-1}v_0) \rightarrow$
$(1;\ 0^{l-1}v_0) \rightarrow (1;\ 0^{l-2}Zv_0) \rightarrow$
$(0;\ 0^{l-2}Zv_0) \rightarrow$
$(l-1;\ 0^{l-2}Zv_0) \rightarrow (l-1;\ v_{l-1}0^{l-3}Zv_0) \rightarrow$
$(l-2;\ v_{l-1}0^{l-3}Zv_0) \rightarrow (l-2;\ v_{l-1}v_{l-2}0^{l-4}Zv_0) \rightarrow$
$\cdots\cdots\cdots \rightarrow$
$(2;\ v_{l-1} \cdots v_30Zv_0) \rightarrow (2;\ v_{l-1} \cdots v_3v_2Zv_0) \rightarrow$
$(1;\ v_{l-1} \cdots v_3v_2Zv_0) \rightarrow (1;\ v_{l-1}v_{l-2} \cdots v_1v_0) \rightarrow$
$(0;\ v_{l-1}v_{l-2} \cdots v_1v_0) = d$

This is a path of type II(b).

Table 4.2: Nodes traversed along different paths for $2 \leq p \leq l-2$

(Entries at any level show only the node-indexes)

	P_i $0 \leq i \leq m-1, i \neq v_0$ $v_{p-1} \neq Y \neq 0$ $x_i \neq v_p$	P_{v_0} $v_{p-1} \neq Y \neq 0$	P_m $v_{p+1} \neq Z \neq 0$
level 0	$(0^{l-1}i)$ $(v_{l-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_1 i)$ $(v_{l-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_1v_0)$	$(0^{l-1}v_0)$ $(v_{l-1}\cdots v_pYv_{p-2}\cdots v_1v_0)$	$(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_10)$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_1v_0)$
level 1	$(0^{l-1}i)$ $(0^{l-2}v_1i)$ $(v_{l-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_0)$	$(0^{l-1}v_0)$ $(0^{l-2}Rv_0)$ $(v_{l-1}\cdots v_pYv_{p-2}\cdots v_2Rv_0)$ $(v_{l-1}\cdots v_pYv_{p-2}\cdots v_1v_0)$	$(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_20^2)$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_10)$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_0)$
level k $1 \leq k \leq p-2$	$(0^{l-k}v_{k-1}\cdots v_1i)$ $(0^{l-k-1}v_kv_{k-1}\cdots v_1i)$ $(v_{l-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_0)$	$(0^{l-k}v_{k-1}\cdots v_2Rv_0)$ $(0^{l-k-1}v_k\cdots v_2Rv_0)$ $(v_{l-1}\cdots v_pYv_{p-2}\cdots v_0)$	$(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_{k+1}0^{k+1})$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_k0^k)$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_0)$
level $p-1$	$(0^{l-p+1}v_{p-2}\cdots v_1i)$ $(0^{l-p}Yv_{p-2}\cdots v_1i)$ $(v_{l-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_0)$ $(v_{l-1}\cdots v_{p+1}x_iv_{p-1}v_{p-2}\cdots v_0)$	$(0^{l-p+1}v_{p-2}\cdots v_2Rv_0)$ $(0^{l-p}Yv_{p-2}\cdots v_2Rv_0)$ $(v_{l-1}\cdots v_pYv_{p-2}\cdots v_0)$ $(v_{l-1}\cdots v_pv_{p-1}v_{p-2}\cdots v_0)$	$(v_{l-1}\cdots v_{p+2}Zv_p0^p)$ $(v_{l-1}\cdots v_{p+2}Zv_pv_{p-1}0^{p-1})$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_0)$
level p	$(0^{l-p}Yv_{p-2}\cdots v_1i)$ $(0^{l-p-1}x_iYv_{p-2}\cdots v_1i)$ $(v_{l-1}\cdots v_{p+1}x_iv_{p-1}\cdots v_0)$	$(0^{l-p}Yv_{p-2}\cdots v_2Rv_0)$ $(0^{l-p-1}v_pYv_{p-2}\cdots v_2Rv_0)$	$(v_{l-1}\cdots v_{p+2}Z0^{p+1})$ $(v_{l-1}\cdots v_{p+2}Zv_p0^p)$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_0)$
level $p+1$	$(0^{l-p-1}x_iYv_{p-2}\cdots v_1i)$ $(0^{l-p-2}v_{p+1}x_iYv_{p-2}\cdots v_1i)$	$(0^{l-p-1}v_pYv_{p-2}\cdots v_2Rv_0)$ $(0^{l-p-2}v_{p+1}v_pYv_{p-2}\cdots v_2Rv_0)$	$(v_{l-1}\cdots v_{p+2}0^{p+2})$ $(v_{l-1}\cdots v_{p+2}Z0^{p+1})$ $(v_{l-1}\cdots v_{p+2}Zv_p\cdots v_0)$ $(v_{l-1}\cdots v_1v_0)$
level r $p+2 \leq r \leq l-1$	$(0^{l-r}v_{r-1}\cdots v_{p+1}x_iYv_{p-2}\cdots v_1i)$ $(0^{l-r-1}v_r\cdots v_{p+1}x_iYv_{p-2}\cdots v_1i)$	$(0^{l-r}v_{r-1}\cdots v_pYv_{p-2}\cdots v_2Rv_0)$ $(0^{l-r-1}v_r\cdots v_pYv_{p-2}\cdots v_2Rv_0)$	$(v_{l-1}\cdots v_{r+1}0^{r+1})$ $(v_{l-1}\cdots v_r0^r)$

Path P_m :

$s \rightarrow (l-1;\ 0^l) \rightarrow (l-1;\ Y0^{l-1}) \rightarrow$
$(0;\ Y0^{l-1}) \rightarrow (0;\ Y0^{l-2}v_0) \rightarrow$
$(1;\ Y0^{l-2}v_0) \rightarrow (1;\ Y0^{l-3}v_1v_0) \rightarrow$
$(2;\ Y0^{l-3}v_1v_0) \rightarrow (2;\ Y0^{l-4}v_2v_1v_0) \rightarrow$
$\cdots\cdots\cdots \rightarrow$
$(l-1;\ Yv_{l-2}v_{l-3}\cdots v_1v_0) \rightarrow (l-1;\ v_{l-1}v_{l-2}\cdots v_1v_0) \rightarrow$
$(0;\ v_{l-1}v_{l-2}\cdots v_1v_0) = d$

This is a path of type II(a).

Lemma 4.6 *When the destination level and the source level are same, the $m+1$ paths constructed by the above rules are node-disjoint.*

Proof : The table 4.3 shows various nodes traversed by the paths, constructed by the above rules. It can be checked from the table that all these nodes are distinct which implies that these paths are node-disjoint. □

The length of the paths P_i can be at most $2l+4$ and that of P_{v_0} and P_m can be at most $2l+3$. Thus, the paths constructed in this case are always smaller than those in case I.

Case III : When level 1 is the destination level, i.e., $p = 1$ or level $l-1$ is the destination level, i.e., $p = l-1$. These two cases can be treated in a similar way. Here we shall discuss only the case when $p = 1$. That is, $s = (0;\ 000\cdots00)$ and $d = (1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$. The paths are constructed as follows.

Path P_i $(0 < i \leq m-1)$:

$s \rightarrow (0;\ 0^{l-1}i) \rightarrow$
$(l-1;\ 0^{l-1}i) \rightarrow (l-1;\ Q0^{l-2}i) \rightarrow$
$(0;\ Q0^{l-2}i) \rightarrow$
$(1;\ Q0^{l-2}i) \rightarrow (1;\ Q0^{l-3}x_ii) \rightarrow$
$(2;\ Q0^{l-3}x_ii) \rightarrow (2;\ Q0^{l-4}v_2x_ii) \rightarrow$

$\cdots\cdots\cdots\rightarrow$

$(l-1;\ Qv_{l-2}v_{l-3}\cdots v_2x_ii)\rightarrow(l-1;\ v_{l-1}v_{l-2}\cdots v_2x_ii)\rightarrow$

$(0;\ v_{l-1}\cdots v_2x_ii)\rightarrow(0;\ v_{l-1}v_{l-2}\cdots v_2x_iv_0)\rightarrow$

$(1;\ v_{l-1}\cdots v_2x_iv_0)\rightarrow d$

These paths are of type I and the maximum length of these paths is at most $2l+6$.

Path P_0 :

$s\rightarrow(1;\ 0^l)\rightarrow(2;\ 0^l)\rightarrow$

$(2;\ 0^{l-3}Z00)\rightarrow$

$(1;\ 0^{l-3}Z00)\rightarrow(1;\ 0^{l-3}Zv_10)\rightarrow$

$(0;\ 0^{l-3}Zv_10)\rightarrow(0;\ 0^{l-3}Zv_1v_0)\rightarrow$

$(l-1;\ 0^{l-3}Zv_1v_0)\rightarrow(l-1;\ v_{l-1}0^{l-4}Zv_1v_0)\rightarrow$

$\cdots\cdots\cdots\rightarrow$

$(2;\ v_{l-1}v_{l-2}\cdots Zv_1v_0)\rightarrow(2;\ v_{l-1}v_{l-2}\cdots v_2v_1v_0)\rightarrow$

$(1;\ v_{l-1}v_{l-2}\cdots v_2v_1v_0)=d$

Path P_m :

$s\rightarrow(l-1;\ 0^l)\rightarrow(l-1;\ Q0^{l-1})\rightarrow$

$(0;\ Q0^{l-1})\rightarrow$

$(1;\ Q0^{l-1})\rightarrow(1;\ Q0^{l-3}v_10)\rightarrow$

$(2;\ Q0^{l-3}v_10)\rightarrow(2;\ Q0^{l-4}v_2v_10)\rightarrow$

$\cdots\cdots\cdots\rightarrow$

$(l-1;\ Qv_{l-2}\cdots v_10)\rightarrow(l-1;\ v_{l-1}\cdots v_10)\rightarrow$

$(0;\ v_{l-1}\cdots v_10)\rightarrow(0;\ v_{l-1}\cdots v_0)\rightarrow$

$(1;\ v_{l-1}v_{l-2}\cdots v_1v_0)=d$

Here, P_0 is of type II(b) and P_m is of type II(a). The length of these two paths can be at most $2l+4$.

Table 4.3: Nodes traversed along different paths for $\mathbf{p} = 0$

(Entries at any level show only the node indexes)

	P_i $0 \leq i \leq m-1, i \neq v_0$ $v_1 \neq Z \neq 0$	P_{v_0} $v_1 \neq Z \neq 0$	P_m $v_{l-1} \neq Y \neq 0$
level 0	$(0^{l-1}i)$ $(v_{l-1}\cdots v_2 Zi)$ $(v_{l-1}\cdots v_2 v_1 i)$	$(0^{l-1}v_0)$ $(0^{l-2}Zv_0)$	$(Y0^{l-1})$ $(Y0^{l-2}v_0)$
level 1	$(0^{l-1}i)$ $(0^{l-2}Zi)$ $(v_{l-1}\cdots v_2 Zi)$ $(v_{l-1}\cdots v_2 v_1 i)$	$(0^{l-1}v_0)$ $(0^{l-2}Zv_0)$ $(v_{l-1}\cdots v_2 Zv_0)$ $(v_{l-1}\cdots v_2 v_1 v_0)$	$(Y0^{l-2}v_0)$ $(Y0^{l-3}v_1 v_0)$
level k $2 \leq k \leq l-2$	$(0^{l-k}v_{k-1}\cdots v_2 Zi)$ $(0^{l-k-1}v_k\cdots v_2 Zi)$	$(v_{l-1}\cdots v_{k+1}0^{k-1}Zv_0)$ $(v_{l-1}\cdots v_k 0^k Zv_0)$	$(Y0^{l-k-1}v_{k-1}\cdots v_1 v_0)$ $(Y0^{l-k-2}v_k\cdots v_1 v_0)$
level $l-1$	$(0v_{l-2}\cdots v_2 Zi)$ $(v_{l-1}v_{l-2}\cdots v_2 Zi)$	$(0^{l-2}Zv_0)$ $(v_{l-1}0^{l-3}Zv_0)$	(0^l) $(Y0^{l-1})$ $(Yv_{l-2}v_{l-3}\cdots v_1 v_0)$ $(v_{l-1}v_{l-2}\cdots v_1 v_0)$

As we have seen earlier, in this case also it can be verified that these paths are node-disjoint except in the following two cases :

(1) The node $(1;\ 0^l)$ in the path P_0 and the node $(1;\ v_{l-1}\cdots v_2x_iv_0)$ in the path P_i will be identical when $v_{l-1} = v_{l-2} = \cdots = v_2 = v_0 = 0$ and $v_1 \neq 0$, i.e., say, $v_1 = b$, $b \neq 0$.

In this case, let us construct $m+1$ node-disjoint paths in a different manner. Here the source node $s = (0;\ 0^l)$ and the destination node $d = (0;\ 0^{l-2}b0)$, where $1 \leq b \leq m-1$.

Path P_a ($a \neq 0$, a can be any integer between 1 and $m-1$) :

$s \rightarrow (0;\ 0^{l-1}a) \rightarrow (1;\ 0^{l-1}a) \rightarrow (1;\ 0^{l-2}ba) \rightarrow (0;\ 0^{l-2}ba) \rightarrow (0;\ 0^{l-2}b0) \rightarrow d$

Path P_i ($0 < i \leq m-1, i \neq a$) :

Here we restrict the value of x_i so that $x_i \neq b$ as well as $x_i \neq 0$ $s \rightarrow (0;\ 0^{l-1}i)$ $\rightarrow (1;\ 0^{l-1}i) \rightarrow (1;\ 0^{l-2}x_ii)$ ($x_i \neq b$ and $x_i \neq 0$) $\rightarrow (0;\ 0^{l-2}x_ii) \rightarrow (0;\ 0^{l-2}x_i0) \rightarrow$ $(1;\ 0^{l-2}x_i0) \rightarrow d$

Path P_0 : $s \rightarrow (1;\ 0^l) \rightarrow d$

Path P_m :

$s \rightarrow (l-1;\ 0^l) \rightarrow (l-2;\ 0^l) \rightarrow (l-3;\ 0^l) \rightarrow \cdots\cdots \rightarrow (2;\ 0^l) \rightarrow (2;\ 0^{l-3}Z00) \rightarrow$ $(1;\ 0^{l-3}Z00) \rightarrow (1;\ 0^{l-3}Zb0) \rightarrow (2;\ 0^{l-3}Zb0) \rightarrow (2;\ 0^{l-3}0b0) \rightarrow d$

These paths are node-disjoint.

(2) The node $(0;\ 0^{l-1}i)$ in the path P_i and the node $(0;\ v_{l-1}\cdots v_2v_1v_0)$ in the path P_m will be identical when $v_{l-1} = v_{l-2} = \cdots = v_2 = v_1 = 0$. Let us construct the paths in the following way when $d = (1;0^{l-1}v_0)$.

Path P_i ($0 \leq i \leq m-1, i \neq v_0$) :

$s \rightarrow (0;\ 0^{l-1}i) \rightarrow (1;\ 0^{l-1}i) \rightarrow (1;\ 0^{l-2}x_ii) \rightarrow (0;\ 0^{l-2}x_ii) \rightarrow (0;\ 0^{l-2}x_iv_0) \rightarrow (1;\ 0^{l-2}x_iv_0)$ $\rightarrow d$

Note that $x_i \neq v_1$, i.e., $x_i \neq 0$ here.

Path P_{v_0} :

$s \rightarrow (0;\ 0^{l-1}v_0) \rightarrow d$

Path P_m :

$s \rightarrow (l-1;\ 0^l) \rightarrow (l-1;\ Q0^{l-1}) \rightarrow (0;\ Q0^{l-1}) \rightarrow (0;\ Q0^{l-2}v_0) \rightarrow (l-1;\ Q0^{l-2}v_0)$
$\rightarrow (l-1;\ 0^{l-1}v_0) \rightarrow (l-2;\ 0^{l-1}v_0) \rightarrow \cdots\cdots \rightarrow (2;\ 0^{l-1}v_0) \rightarrow d$

It should be noted here that if $v_0 = 0$ then some portion of these paths will form a loop. In that case we shall discard that portion. These paths are clearly node-disjoint. Since the case for $p = l-1$ can similarly be dealt with, we have the following lemma.

Lemma 4.7 *For $l \geq 3$, there exist $m+1$ node-disjoint paths between the source and the destination nodes, when $p = 1$ or $p = l-1$.*

Regarding the path length, the longest path in this case can be greater than the diameter of the topology by at most 6 when $2 \leq l \leq 5$. Otherwise, the length of the longest path is greater than the diameter by at most 5.

Because of lemmas 4.3, 4.4, 4.5, 4.6 and 4.7 and the discussions made on the path lengths, we now have the following theorem.

Theorem 4.2 *For $l \geq 3$, the node-connectivity of the GHCC is $m+1$ and in presence of m faulty nodes the diameter of the GHCC can exceed that for the fault free case by at most 6.*

We now consider the GHCC for $l = 2$. In this case the path P_m will not exist, while the other node-disjoint paths can be easily derived from the general rule discussed above. The above results on the lengths of the corresponding paths will also remain valid for $l = 2$. Thus the node connectivity of the $GHCC(2, m)$ is m and the diameter in presence of $(m-1)$ faulty nodes may be more than that for the fault-free case by at most 6.

4.7 Applications

In this section we shall show how some useful algorithms can be implemented on this network topology.

4.7.1 Sum / Average / Maximum / Minimum

Here we shall discuss the problem of summing up lm^l data points stored in lm^l processors. The problem of finding the maximum and minimum of these data can be done in a similar way. Let us first give an outline of this algorithm.

In section 4.2, we have mentioned that the topology has m^l disjoint cycles of length l. These l nodes in a cycle lie in l different levels. In step 1, the values are summed up along the m^l disjoint cycles and these partial sums are then stored in the processors lying in level 0. After this step, the remaining job is to sum m^l values which are now stored in a single level, that is, in level 0. We know that the nodes in a single level are grouped into m^{l-1} node-disjoint cliques of size m. In $\lceil \log_2 m \rceil$ parallel steps the values stored in the processors forming a particular clique are added and then stored in the processor of that clique having the least significant literal (lsl) of the processor ID as 0. Now, only the nodes at the level 0 with the lsl of their ID as 0 contains the values to be added. There are m^{l-1} such nodes in level 0. These values are now sent to the level 1. In this level, m^{l-1} nodes of the form $(1; *^{l-1}0)$ are grouped into m^{l-2} cliques. Values stored in these cliques are again added in $\lceil \log_2 m \rceil$ parallel steps and stored in m^{l-2} nodes of the form $(1; *^{l-2}00)$. These values are then sent to the level 2. This process is continued until the values reach the level $l-1$. In the level $l-1$ the values are distributed over the nodes of the form $(l-1; *0^{l-1})$ which are gathered in only one clique. The final result is thus computed by simply adding these m values in $\lceil \log_2 m \rceil$ parallel steps and finally stored in the node $(l-1; 0^l)$.

Each processor will have two temporary registers, one of which will be referred to as the *SUM* register and the other as R register. The *SUM* registers initially

contain the data values to be added and all R registers are initially reset to 0.

Algorithm SUM

Step 1

do Steps 1.1 and 1.2 in **parallel** for all nodes having node index $(v_{l-1}v_{l-2}\cdots v_1v_0)$

Step 1.1

For $i = 1$ to $\lfloor l/2 \rfloor - 1$ **do**

{

Step 1.1.1 The content of $SUM(\lfloor l/2 \rfloor - i + 1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$ is sent to

$$R(\lfloor l/2 \rfloor - i;\ v_{l-1}v_{l-2}\cdots v_1v_0)$$

Step 1.1.2 $SUM(\lfloor l/2 \rfloor - i;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(\lfloor l/2 \rfloor - i;\ v_{l-1}v_{l-2}\cdots v_1v_0)$
$+ R(\lfloor l/2 \rfloor - i;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

}

Step 1.2

For $i = 1$ to $\lfloor l/2 \rfloor - 1$ **do**

{

Step 1.2.1 The content of $SUM(\lfloor l/2 \rfloor + i;\ v_{l-1}v_{l-2}\cdots v_1v_0)$ is sent to

$$R(\lfloor l/2 \rfloor + i + 1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$$

Step 1.2.2 $SUM(\lfloor l/2 \rfloor + i + 1;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(\lfloor l/2 \rfloor + i + 1;\ v_{l-1}\cdots v_0)$
$+ R(\lfloor l/2 \rfloor + i + 1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

}

Step 1.3

Step 1.3.1 (i) $R(0;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

(ii) $SUM(0;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(0;\ v_{l-1}v_{l-2}\cdots v_1v_0)$
$+ R(0;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

Step 1.3.2 **if** (l is odd) **then**

(i) $R(0;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(l-1;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

(ii) $SUM(0;\ v_{l-1}v_{l-2}\cdots v_1v_0) \leftarrow SUM(0;\ v_{l-1}v_{l-2}\cdots v_1v_0)$
$+ R(0;\ v_{l-1}v_{l-2}\cdots v_1v_0)$

Step 2

For $i = 0$ to $l - 1$ **do**

{

(i) Add the contents of m registers $SUM(i;\ v_{l-1}v_{l-2}\cdots v_{i+1} * 0^i)$ and store the final result in $SUM(i;\ v_{l-1}v_{l-2}\cdots v_{i+1}0^{i+1})$.

(ii) Send the content of $SUM(i;\ v_{l-1}v_{l-2}\cdots v_{i+1}0^{i+1})$ to $SUM(i+1;\ v_{l-1}v_{l-2}\cdots v_{i+1}0^{i+1})$.

}

Time Complexity : Step 2 requires $l\lceil \log_2 m \rceil$ time units. Step 1.1 and step 1.2 are executed in parallel and each of them requires $\lfloor \frac{l}{2} \rfloor - 1$ time units. Hence the algorithm SUM requires $(\lfloor \frac{l}{2} \rfloor - 1) + 1 + l\lceil \log_2 m \rceil = O(l \log m)$ time for lm^l data points.

4.7.2 ASCEND / DESCEND Classes of Algorithms

Ascend / Descend classes of algorithms are highly parallel algorithms. These classes of algorithms have applications in the problems like cyclic shift, bitonic merge, bitonic sort, odd-even merge, fast Fourier transform, shuffle, matrix transposition, etc. Implementation and application of these classes of algorithms are discussed in [PV81] in detail. In this section we shall show how the Ascend class of algorithms can efficiently be implemented on the proposed topology $GHCC(l, m)$. The Descend class of algorithms can also be implemented on a GHCC in a similar fashion.

Let us assume that all the three parameters N, m and l are powers of 2 and $N = 2^q$. With this restriction, a node $(p; n) \equiv (p;\ u_{l-1}u_{l-2}\cdots u_1u_0)$, $0 \leq p \leq l-1$ and $0 \leq u_k \leq m-1$, $0 \leq k \leq l-1$, can also be equivalently represented by a binary string of length $q = \log_2 N = \log_2 l + l \log_2 m$ as $(b_{q-1}b_{q-2}\cdots b_1b_0)$. Let us assume that among these q bits, $\log_2 l$ most significant bits represent the level number p and the bit string $(b_{(i+1)\log m-1}b_{(i+1)\log m-2}\cdots b_{i\log m})$ represents the literal

u_i, $0 \leq i \leq l-1$ (from now on all logarithms in this section will be taken with base 2). In other words, $u_i = \sum_{j=0}^{\log m-1} b_{i \log m+j} * 2^j$, and $p = \sum_{j=1}^{\log l} b_{\log N-j} * 2^{\log l-j}$. Moreover, in general we call a processor $P(b_{q-1}b_{q-2} \cdots b_1 b_0)$ as the r^{th} processor (node) or $P[r]$ if $r = \sum_{j=1}^{q-1} b_j * 2^j$, $0 \leq r \leq N-1$.

Let us assume that the input data $t_0, t_1, t_2, \cdots, t_{N-1}$ are stored in the processors $P[0], P[1], P[2], \cdots, P[N-1]$ respectively. At the i^{th} step of the Ascend algorithm, the operands involved reside in the processors whose identifications differ only in the i^{th} least significant bit, $0 \leq i \leq q-1$. While implementing the Ascend algorithm on the $GHCC(l,m)$, we notice that any two processors in level 0 whose binary string identifiers differ only in the bits $b_0, b_1, \cdots, b_{\log m-1}$ are connected by an edge. Thus, the first $\log m$ steps of the Ascend algorithm can be executed only on the data stored in the processors in level 0. Similarly, the next $\log m$ steps can be carried out only on the data stored in the processors in level 1 and so on. Hence, the initial $\log m$ steps are carried out among the processors in level 0 and then a cyclic shift is given to the data so that the data stored in the processor $P(p;n)$ will move to the processor $P(p+1;n)$. As a result, the data which have already gone through the first $\log m$ steps of the Ascend algorithm are now in level 1 and are ready to go through the next $\log m$ steps of the algorithm. At this stage, the second $\log m$ steps are carried out on the data in the processors in level 1 and simultaneously the first $\log m$ steps of the algorithm are carried out on the new data elements moved to the processors in level 0. After another cyclic shift, first, second and third $\log m$ steps of the Ascend algorithm will be carried out in parallel on the data in the processors in level 0, 1 and 2 respectively. This process will be continued until all the data go through the first $l \log m$ steps of the Ascend algorithm. This would require $2l$ cyclic shifts and in total $2l \log m$ time steps. As the processors whose binary string identifier differ only in $\log l$ most significant bits are connected in a cycle, the last $\log l$ steps of the ascend algorithm will be carried out among the processors which are connected in a cycle of length l. This can be done in $O(l)$ steps [PV81]. Formal description of the algorithm is given below.

In this algorithm, OPER(r, i, $P[r]$, $P[r+2^i]$) represents a basic operation that is carried out on the data stored in the processors $P[r]$ and $P[r+2^i]$ which may depend on the values of r and i. Moreover, by $\text{bit}_j(r)$ we mean the j^{th} bit of the binary string representing r.

Algorithm *ASCEND*

Step 1 :

/* Job for the processor $P[r] \equiv P(p;n) \equiv P(b_{q-1}b_{q-2}\cdots b_1b_0)$, in level p, where $p = \sum_{j=1}^{\log l} b_{q-j} * 2^{\log l - j}$, executes the following statements at this step. */

for $i = 0$ to $(2l-2)$ **do**

{

Step 1.1 :

if $p \le i \le (l+p-1)$ **then**

{

for $j = 0$ to $(\log m - 1)$ **do**

{

if $\text{bit}_{p\log m+j}(r) = 0$ **then**

$OPER(r, j, P[r], P[r+2^{p\log m+j}])$

}

}

Step 1.2 :

Send the data stored in the processor $P(p;n)$ to the processor $P(p+1;n)$.

}

Step 2 : /* Last $\log l$ steps of the Ascend algorithm */

for $j = 0$ to $(\log l - 1)$ **do**

{ **for** each $r : 0 \leq r \leq N-1$ **do** in parallel

{

if $\text{bit}_{l \log m + j}(r) = 0$ **then**

OPER(r, j, $P[r]$, $P[r + 2^{l \log m + j}]$)

/* Here, *OPER* involves the data stored in two processors which are in the same cycle */

}

}

Time Complexity : Step 1 requires $2l \log m$ time. Step 2 can be performed in $O(l)$ time [PV81]. Thus, in total, the Ascend class of algorithms can be implemented on the proposed topology in $O(l + 2l \log m)$ time.

4.8 Conclusion

In this chapter, we have presented a new family of interconnection networks $GHCC(l, m)$, in which any two quantities out of the network size, node degree and diameter can be independently chosen with certain restrictions by controlling the two parameters l and m. The GHCC reduces to a ring for $m = 1$ and a CCC for $m = 2$. For $l = 1$, it reduces to a complete graph. The GHCC competes favorably with GHC, star graphs, etc. in regard to the degree and the diameter for a given number of nodes. For $l \geq 3$, the graph is $(m + 1)$ connected and in presence of m faulty nodes the diameter of the network increases only by 6. The topology has a simple routing scheme to implement. Different useful classes of algorithms can be efficiently implemented on this topology.

Chapter 5

Isomorphism of Maximal Outerplanar Graphs

5.1 Introduction

The general techniques for testing isomorphism for planar [HT73], [HW74] and outerplanar graphs [CB81] are definitely applicable for testing isomorphism of maximal outerplanar graphs (MOP) as well. However, there are some specific properties of maximal outerplanar graphs which can be exploited to design more efficient and simpler algorithms for testing isomorphism of MOPs. Beyer, Jones and Mitchel proposed such an algorithm [BJM79] using the fact that the hamiltonian degree sequence can uniquely characterize a MOP upto isomorphism.

In [JK88], Ja Ja and Kosaraju parallelized the approach due to Hopcroft and Tarjan [HT73] for planar graphs, with a time complexity of $O(\log^3 N)$ when implemented on a CREW PRAM model with $O(N^2)$ processors, and $O(\sqrt{N})$ time when implemented on an $\sqrt{(N)} \times \sqrt{(N)}$ array of processors. In [LLP+90], Levcopoulos et al. have shown that isomorphism of trees and outerplanar graphs can be tested in $O(\log^2 N)$ time with $\frac{N}{\log^2 N}$ processors on an EREW PRAM model. These results also hold for MOPs. Moreover, in case of MOPs, if the hamiltonian degree sequences are compared (as discussed in [BJM79]) by using efficient parallel string matching algorithms, then the time complexity can be reduced to

$O(\log N)$ by using an $O(\log N)$ optimal string matching algorithm presented in [CCG93], which works on an EREW PRAM model. But, in this case, the number of processors used is quite large.

In this chapter, we are going to present a new parallel algorithm for testing isomorphism of maximal outerplanar graphs (MOPs). Here, we have viewed the isomorphism problem of MOPs in terms of the isomorphism of triangulations of a convex polygon. Some interesting properties of triangulation of convex polygons have been unveiled and using these properties, a simple and efficient parallel algorithm has been developed for the MOP isomorphism problem. This algorithm requires $O(\log N)$ time using N processors when implemented on an EREW PRAM model as well as on a hypercube. We assume that the graphs with N vertices will be given in the form of ordered adjacency lists. If, however, the graphs are given in terms of the adjacency matrices, then our algorithm will take same amount of time on N^2 processors.

As a by-product of this work we have been able to introduce a new classification scheme and find various interesting properties regarding triangulation of convex polygons which may be useful for a better understanding of the triangulation problem and in finding efficient solutions to several problems in the area of computational geometry, robotics, etc.

5.2 Preliminaries

Our goal is to design a simple parallel algorithm for testing isomorphism of MOPs. Since there is a one-to-one correspondence between the maximal outerplanar graphs and the planar embedding of triangulated convex polygons, we would view this problem in terms of the isomorphism of triangulated convex polygons. For that we would like to reveal some basic concepts regarding the triangulation problem.

Triangulation of a simple polygon is a classical problem of interest in the field of computational geometry [PS88]. Triangulating a polygon is to partition it into

non-intersecting triangles (they can share common edges and vertices) by inserting internal diagonals [PS88]. To triangulate a polygon of size N, $N-3$ internal diagonals are required. This will generate $N-2$ component triangles in the polygon. These internal diagonals can be chosen in many possible ways and as a result, triangulation of a simple polygon is non-unique. Fig. 5.1 shows two different triangulations of an octagon.

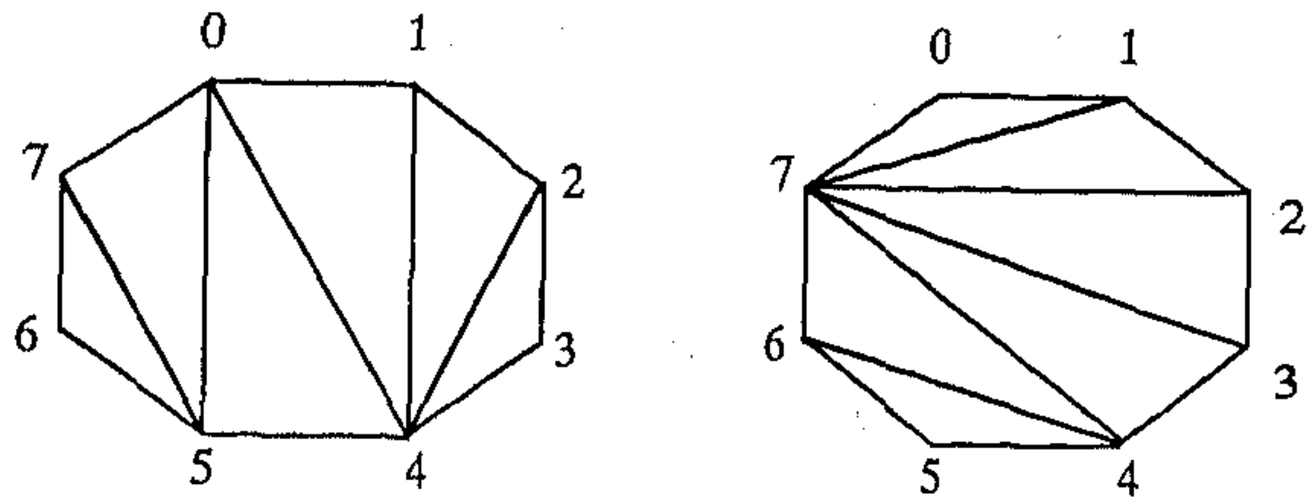


Figure 5.1: Two triangulations of an octagon

In a convex polygon, all of its diagonals are internal diagonals. Thus, the number of triangulations of a convex N-gon is independent of the shape, and therefore, it can be uniquely characterized by the number of vertices N. Let $\delta(N)$ be the total number of triangulations of a convex N-gon. In [GKP88], Knuth et al. have shown that

$$\delta(N) = C_{N-2} = \frac{1}{N-1}\binom{2N-4}{N-2}$$

where, C_N is the N^{th} Catalan number.

Figures 5.1(a) and (b) show that the triangulated polygons are nothing but maximal outerplanar graphs of size 8. In general, each maximal outerplanar graph corresponds to a triangulated convex polygon and vice versa. The MOP corresponding to a triangulation T will be represented by $G_T\left(V, \varepsilon\right)$, where $v_i \in V$ is a vertex of the polygon, and $(v_i, v_j) \in \varepsilon$ if v_i and v_j are joined by either a polygon

edge or a diagonal.

Two triangulations are said to be *isomorphic* to each other if the corresponding maximal outerplanar graphs are isomorphic. Two isomorphic triangulations of an octagon are shown in Fig. 5.2.

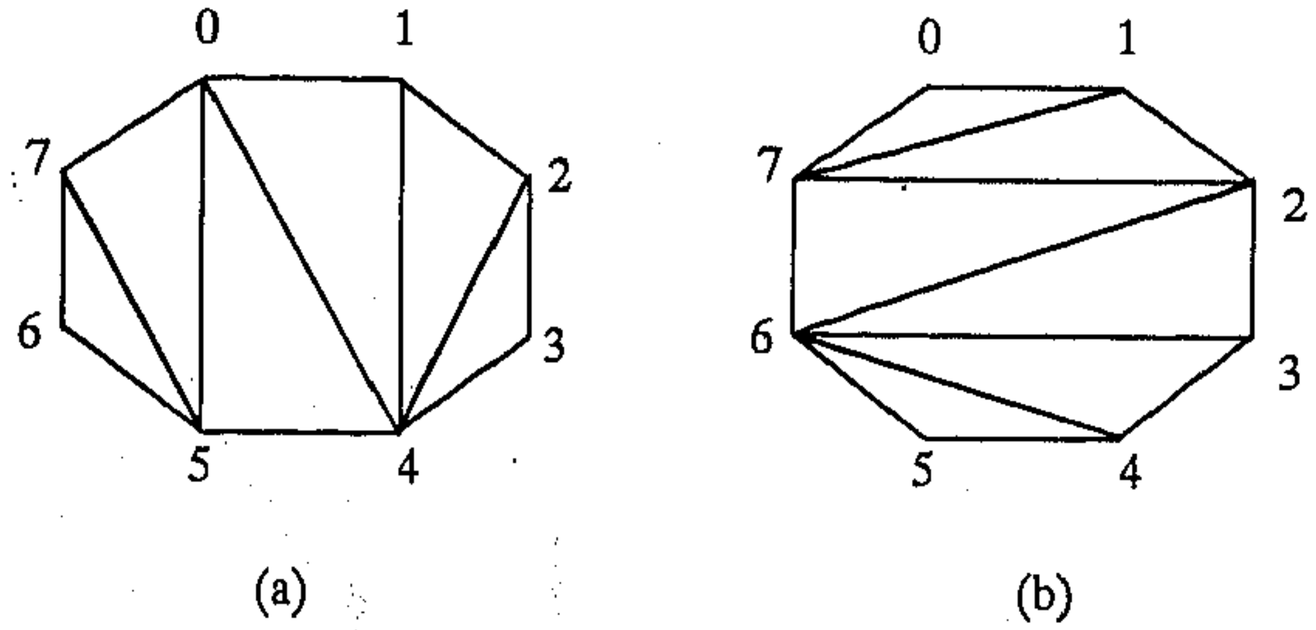


Figure 5.2: Two isomorphic triangulations

5.3 Classification of Polygon Triangulations

In this section we will introduce a new classification scheme for triangulation of convex polygons which will also apply to the characterization of MOPs.

Throughout the chapter, we will consider a convex polygon with N vertices (N is any positive integer, $N \geq 3$). The vertices are numbered from 0 to $N-1$ in the clockwise direction. The polygon will be denoted by $P(0, 1, 2, ..., N-1)$. The notation $T(0, 1, 2, ..., N-1)$ or just T in short, will be used to denote a particular triangulation of the polygon $P(0, 1, ..., N-1)$. The corresponding maximal outerplanar graph will be denoted by G_T. An edge joining two vertices i and j, will be denoted by the pair (i, j). Nodes of the graph G_T are also labeled in the same fashion as in P.

Definition 5.1 *In G_T, the length $d(i,j)$ of an edge (i,j) is defined as the minimum path length between i and j along the outermost cycle. In other words,*

$$d(i,j) = d(j,i) = min\ \{|i-j|, N-|i-j|\}$$

Example 5.1 *In Fig. 5.2(a), $d(1,6) = 3$, $d(1,3) = 2$.*

For any triangulation T, the length of the longest edge in G_T will be denoted by L. It is easy to verify that $L \leq \lfloor N/2 \rfloor$.

5.3.1 Classification

Definition 5.2 *A triangulation T is said to be a bisector triangulation, if $L = \lfloor N/2 \rfloor = \lceil N/2 \rceil$. Otherwise T will be a non-bisector triangulation.*

Thus, for odd values of N, we can not have any bisector triangulation, whereas for even N, we may have both the types.

Definition 5.3 *Since each internal face in G_T is enclosed by three vertices, any two vertices corresponding to an internal face, are said to be the consecutive neighbors of the third vertex.*

Theorem 5.1 *For any triangulation T, $\lceil N/3 \rceil \leq L \leq \lfloor N/2 \rfloor$.*

Proof : For a bisector triangulation, $L = \lceil N/2 \rceil = \lfloor N/2 \rfloor$. So we need to consider only the non- bisector triangulations. Let T be a non-bisector triangulation and without any loss of generality, let us assume that G_T contains the edge $(0,x)$, such that $d(0,x) = x = L < \lceil N/2 \rceil$. Let i and x be two consecutive neighbors of vertex 0, where $i > x$, as shown in Fig. 5.3.

Now, $d(x,i) \leq L$. Therefore,

$$i - x \leq L \tag{5.1}$$

Also, $d(0,i) \leq L$. As $i > x$ and $d(0,i) = N - i$,

$$N - i \leq L \tag{5.2}$$

Adding (5.1) and (5.2) we get, $2L \geq N - x$.

As $x = L$, we have, $3L \geq N$,

i.e., $\lceil N/3 \rceil \leq L$. Hence the proof. □

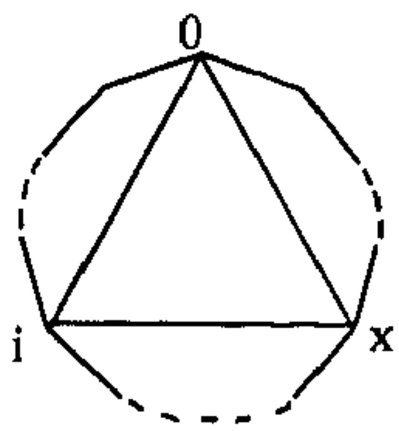


Figure 5.3: Illustration of theorem 5.1

Lemma 5.1 *In any non-bisector triangulation T, if there are more than one longest edge, then every two of them must have one end-point in common.*

Proof : W.l.o.g. let us assume that $(0,x)$ be one such edge, where $d(0,x) = x = L < \lceil N/2 \rceil$. Let there be another longest edge $(i, i+x)$ such that $i > x$ and $i + x \neq 0 \ (mod\ N)$. The situation is shown in Fig. 5.4. For proper triangulation of the original polygon $P(0, 1, ..., N-1)$, we now have to triangulate the polygonal structure $(0, x, x+1, x+2, ..., i, i+x, i+x+1, ..., N-1)$ which is also convex. Thus we have to join at least one vertex from the set $(i+x, i+x+1, ..., N-1, 0)$ to one of $(x, x+1, x+2, ..., i)$ by an edge other than $(0,x)$ and $(i, i+x)$. But the length of that edge will be at least $L+1$. This contradicts our hypothesis. □

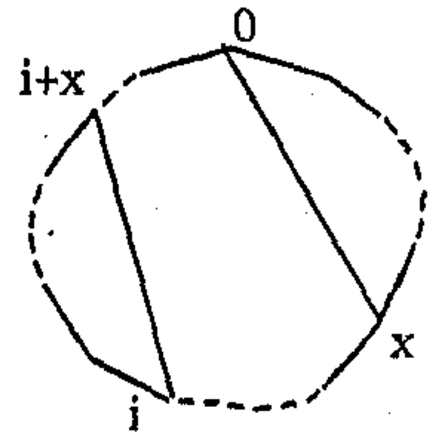


Figure 5.4: Illustration of lemma 5.1

Theorem 5.2 *For any triangulation T, G_T will satisfy the following properties:*

(1) If $L = \lfloor N/2 \rfloor = \lceil N/2 \rceil$, there will be exactly one longest edge in T.

(2) If $L = \lfloor N/3 \rfloor = \lceil N/3 \rceil$, there will be exactly three longest edges in T.

(3) If $\lfloor N/3 \rfloor < L < \lceil N/2 \rceil$, there will be at most two longest edges in T.

Proof : Follows from Theorem 5.1 and lemma 5.1. □

Theorem 5.3 *For any non-bisector triangulation T, there is a unique triangle (face) in G_T whose perimeter is equal to N.*

Proof : W.l.o.g. let us assume that G_T contains the edge $(0, x)$, such that $d(0, x) = x = L < \lceil N/2 \rceil$. Let i and x be two consecutive neighbors of the vertex 0, where $x + 1 \leq i \leq N - 1$.

Now, $d(0, x) = L$. As L is the maximum length, $d(x, i) = i - x = i - L$ and $d(i, 0) = N - i$.

Therefore, the perimeter of the triangle $(0, x, i)$ is N.

To prove uniqueness, let us assume that there exists another triangle in G_T, whose perimeter is also N. From planarity, the second triangle must be contained in either of the three shaded portions of the original polygon (Fig. 5.5). These three portions (regions) are named as α, β, and γ.

Now, the perimeter of any triangle, contained in the portion α, will be at most $2L$, which can not be equal to N, as $L < \lceil N/2 \rceil$. Similarly, the perimeter of any triangle contained in the portion β and γ will be at most $2(i - L)$ and $2(N - i)$, respectively. But none of these can be equal to N, as $L < \lceil N/2 \rceil$. Hence the theorem. □

Corollary 5.1 *The longest edge(s) of the triangulated polygon must be the edge(s) of the unique triangle with perimeter N.*

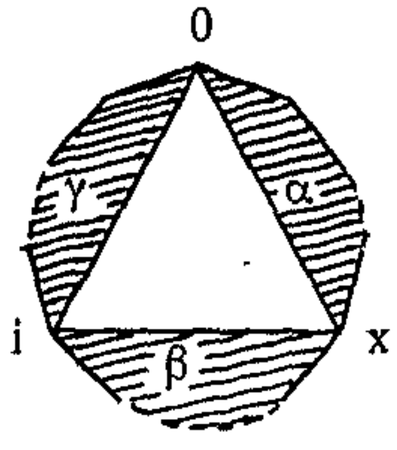


Figure 5.5: A unique face with perimeter N

Definition 5.4 *In any non-bisector triangulation, the unique triangle with perimeter N will be called the central triangle of that triangulation.*

Example 5.2 *The triangulation of the polygon, as shown in Fig. 5.6, has the central triangle (0, 3, 6).*

Note that the concept of central triangle applies only to non-bisector triangulation. The central triangle may be equilateral, isosceles or scalene.

Remark : If $N = 3$, the polygon itself is the central triangle. For $N = 4$, all triangulations are bisector triangulations and hence there is no central triangle. For $N > 4$, the central triangle may involve at most one polygonal edge; otherwise, if the central triangle would have two polygonal edges, then the length of the third edge of this triangle would be $N - 2$. This, in turn, implies that $N - 2 \leq \lfloor \frac{N}{2} \rfloor$,

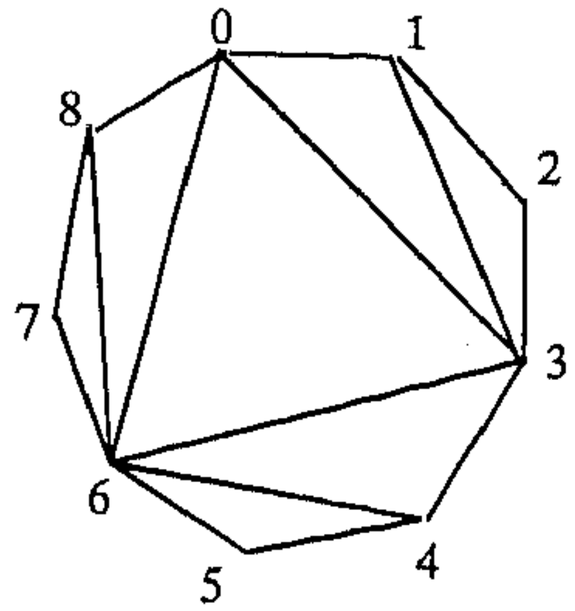


Figure 5.6: The central triangle (0, 3, 6)

which is impossible for $N > 4$.

If N (> 4) is even, then the central triangle (corresponding to a non-bisector triangulation) can not involve any polygonal edge. If N (> 4) is odd, the central triangle may involve a polygonal edge and in that case the length of the other two edges will be $(N-1)/2$.

Remark : Incidentally, the notion of the unique central triangle corresponds to the three *central vertices* introduced in [BJM79], which were obtained by successively deleting the *ears* from the triangulated convex polygon. Moreover, when $L = \lfloor \frac{N}{2} \rfloor = \lceil \frac{N}{2} \rceil$, the end points of the longest edge of the bisector triangulation also correspond to the two *central vertices* defined in [BJM79].

Definition 5.5 *A triangulation will be called scalene or isosceles or equilateral, if the central triangle is scalene or isosceles or equilateral respectively.*

The notions of bisector triangulation and the central triangle allow us to classify the various triangulations of a convex polygon into the following cases, as shown in Fig. 5.7.

Remark : For $N = 6r+1$ or $6r+5$, $(r \geq 0)$, bisector and equilateral triangulations are not possible.

For $N = 6r+2$ or $6r+4$, $(r \geq 0)$, equilateral triangulations are not possible.

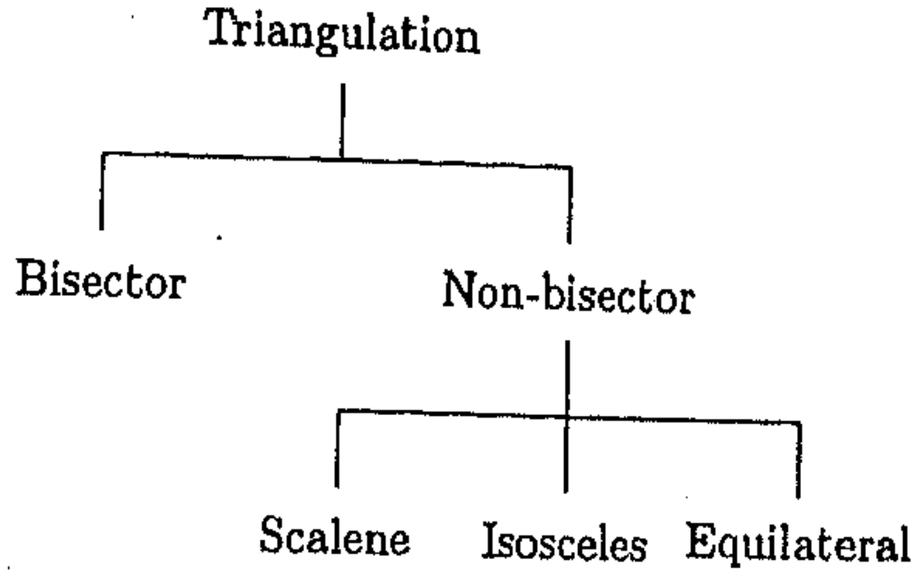


Figure 5.7: Classification of Triangulations

For $N = 6r + 3$, $(r \geq 0)$, bisector triangulations are not possible.

5.3.2 Reflectional Symmetry of Triangulations

In this section we are going to define reflectional symmetry (a) about a vertex, and (b) about the perpendicular bisector of an edge.

Definition 5.6 *The reflection of a triangulation* $T(0, 1, \cdots, N-1)$ *about the vertex 0 denoted by* $T'(0, 1, \cdots, N-1)$, *is defined as,*

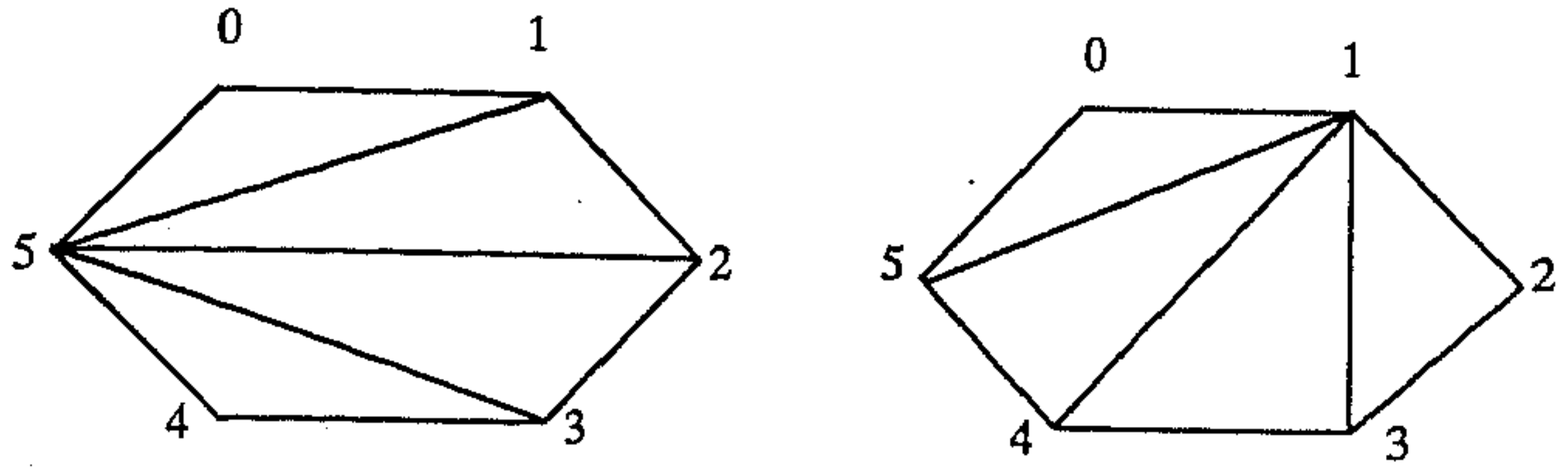
$$T'(0, 1, \cdots, N-1) = T(0, N-1, N-2, \cdots, 1)$$

Fig. 5.8 shows a triangulation of a convex hexagon and also its reflection about the vertex 0.

Remark : $(T')' = T$.

Definition 5.7 *A triangulation* $T(0, 1, ..., N-1)$ *is called V-symmetric triangulation about the vertex 0, if* $T(0, 1, ..., N-1) = T'(0, 1, ..., N-1)$.

Fig. 5.9 shows an example of a V-symmetric triangulation about the vertex 0.



T(0, 1, 2, 3, 4, 5) T '(0, 1, 2, 3, 4, 5)

Figure 5.8: Reflection of a triangulation about the vertex 0

Definition 5.8 *Reflection of the vertices of a convex polygon* $P(0, 1, ..., N-1)$ *about the perpendicular bisector of the edge* $(0, N-1)$ *will be defined by the mapping* $E : V \rightarrow V$, *where* $E(v) = N - v - 1$, $v \in V$.

Definition 5.9 *Reflection of a triangulation* $T(0, 1, \cdots, N-1)$ *about the edge* $(0, N-1)$ *will be denoted by* $T^E(0, 1, \cdots, N-1)$ *and defined as*

$$T^E(0, 1, \cdots, N-1) = T(E(0), E(1), E(2), \cdots, E(N-1)) = T(N-1, N-2, \cdots, 1, 0)$$

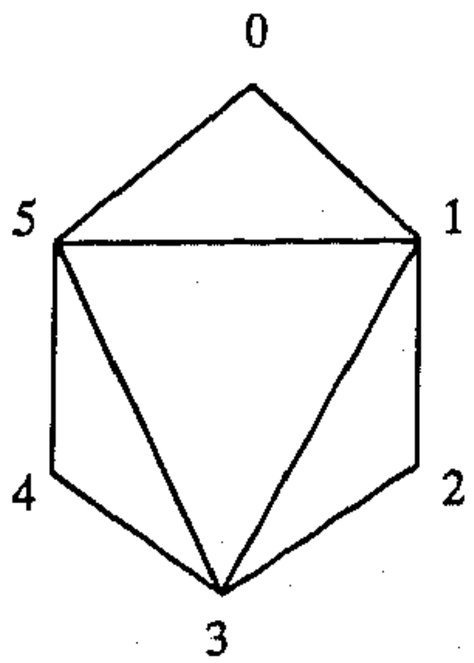


Figure 5.9: V-symmetric triangulation about the vertex 0

Fig. 5.10 shows an example of a triangulation and its reflection about the perpendicular bisector of an edge.

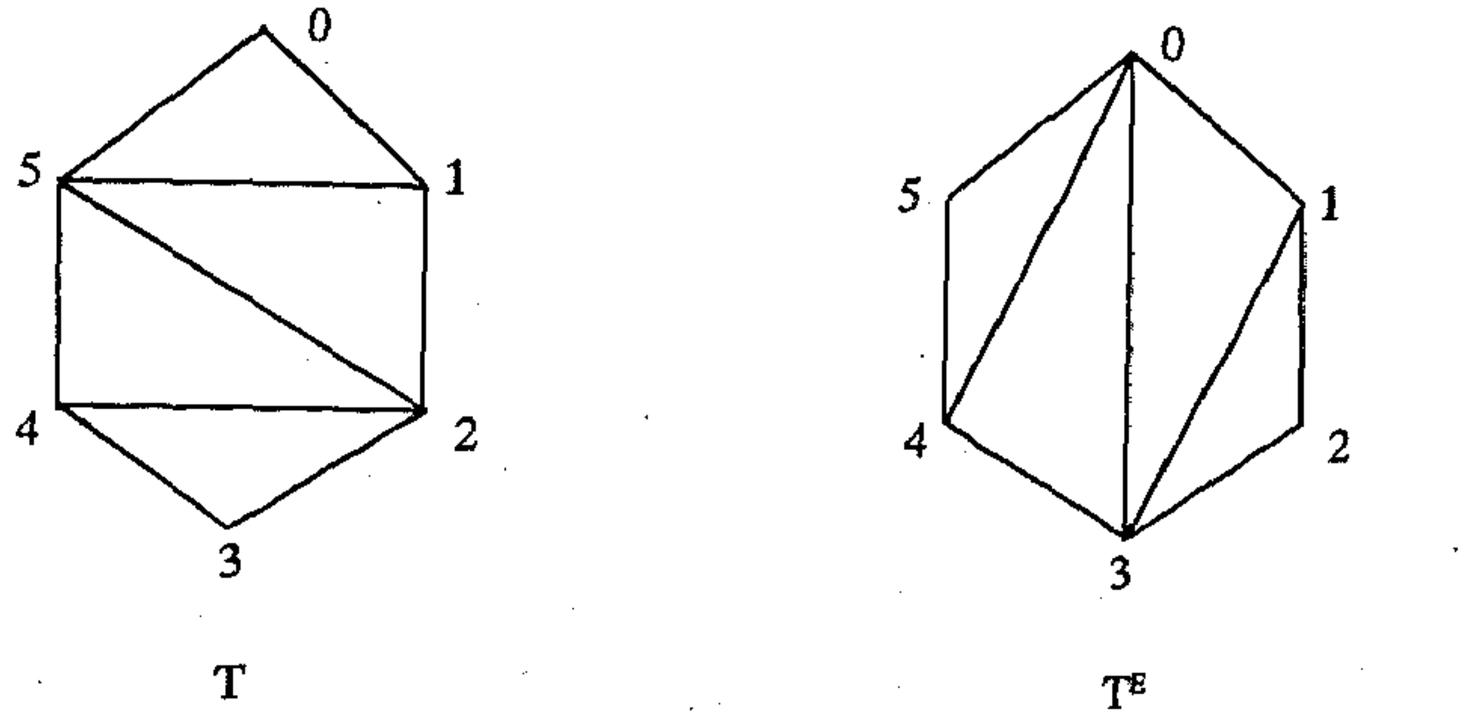


Figure 5.10: Reflection about the perpendicular bisector of the edge (0, 5)

Remark : $(T^E)^E = T$.

Definition 5.10 *A triangulation* $T(0, 1, \cdots, N-1)$ *is called an E-symmetric triangulation if* $T(0, 1, \cdots, N-1) = T^E(0, 1, \cdots, N-1)$.

Fig. 5.11 displays an example of an E-symmetric triangulation.

Lemma 5.2 *Let* $\delta_E(N)$ *denote the total number of E-symmetric triangulations of a polygon* $P(0, 1, ..., N-1)$. *Then,* $\delta_E(2) = 1$, *and for* $N > 2$,

$$\delta_E(N) = \begin{cases} 0, & \text{if } N \text{ is even;} \\ \delta((N+1)/2), & \text{otherwise.} \end{cases}$$

Proof : We prove the result by induction on N. It is easy to verify the result for $N = 3$, 4 and 5. Let the result be true for $N = 1, 2, ..., N-1$. Let $T(0, 1, 2, ..., N-1)$ be any E-symmetric triangulation. Because of E-symmetry, for every edge (i, j) in T, the edge $(N-j-1, N-i-1)$ should also be present in T. But, due to planarity, for $i < (N-1)/2 < j$, it is not possible for any triangulation T to have

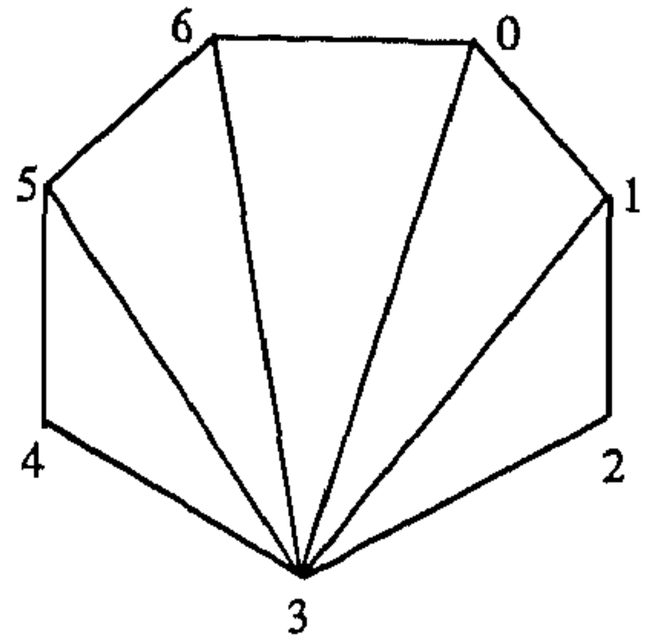


Figure 5.11: E-symmetric triangulation about the perpendicular bisector of (0, 6)

both these edges, unless $j = N-i-1$ (i.e., when (i,j) and $(N-j-1, N-i-1)$ represent the same edge). When $j = N-i-1$, an edge of the form $(i, N-i-1)$, $1 \leq i \leq N-2$, divides a polygon into two polygons of smaller size and both of them have to be triangulated in an E-symmetric fashion to get an E-symmetric triangulation of the original polygon. The polygon containing the edge $(0, N-1)$ and $(i, N-i-1)$ will have even number of vertices. But, by induction hypothesis, it is not possible to triangulate it in an E-symmetric way. So, T cannot have an internal edge of the form $(i, N-i-1)$.

From the above discussion it follows that if N is even, the polygon $(0, N/2 - 1, N/2, N-1)$ will remain intact in T. Thus no E-symmetric triangulation exists, for even values of N.

For odd values of N, there must be a central triangle $(0, (N-1)/2, N-1)$ in T, as edges (i,j) for $i < (N-1)/2 < j$ cannot exist in T. This divides the polygon into two smaller polygons $(0, 1, ..., (N-1)/2)$ and $((N-1)/2, ..., N-2, N-1)$. Referring to Fig. 5.12, for T to be E-symmetric, we must have, $T^E(0, 1, ..., (N-1)/2) = T((N-1)/2, ..., N-1)$. Thus the total number of E-symmetric triangulations will be $\delta_E(N) = \delta((N+1)/2)$. □

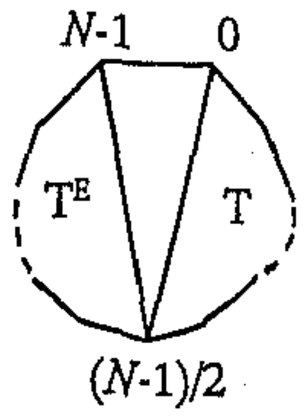


Figure 5.12: Illustration of lemma 5.2

Remark : If we redefine $\delta(N)$ as,

$$\delta(N) = D_{N-2} = \begin{cases} C_{N-2}, & \text{if N} \geq 2 \\ 0, & \text{for fractional and/or negative values of } (N-2) \end{cases}$$

Then the above result can be restated as :

$\delta_E(2) = 1$, and for $N > 2$, $\delta_E(N) = D_{(N-3)/2}$.

5.4 Counting Non-Isomorphic Triangulations of Each Class

Having classified the triangulations of a convex polygon as above, we now show that two triangulations belonging to two different classes cannot be isomorphic. Then we proceed to count the number of non-isomorphic triangulations in each of the above classes. Summing them up, we could also find the overall expression for the total number of non-isomorphic triangulations of a convex polygon.

5.4.1 Properties of Isomorphic Triangulations

For the sake of completeness we would like to reiterate some existing results found in the literature [BJM79].

Result 1 : *The maximal outerplanar graph G_T, corresponding to any triangulation $T(0, 1, ..., N-1)$, has a unique hamiltonian cycle, $(0, 1, ..., N-1, 0)$, along the outermost cycle.*

This result can be proved in the following way.
The sequence of vertices $(0, 1, 2, ..., N-1, 0)$ along the outermost face of G_T gives us one hamiltonian cycle. If possible, let there be another hamiltonian cycle which includes an internal edge (i, j), as in Fig. 5.13. Let us try to traverse this hamiltonian cycle starting from the vertex i and moving towards j along the edge (i, j). The edge divides the polygon into two halves and from vertex j we must go to a vertex in one of the halves. But once we enter one of the halves due to planarity, there is no internal edge, which can take us to the other half. The other two edges into the other half also cannot be used, as i and j both have already been traversed. Thus, we cannot traverse all the vertices. Hence the contradiction.

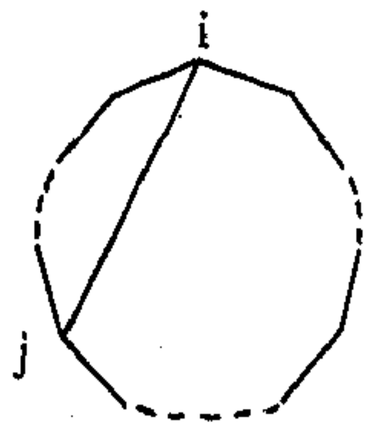


Figure 5.13: Uniqueness of the hamiltonian cycle

Result 2 : *Two triangulations $T_1(0, 1, ..., N-1)$ and $T_2(0, 1, ..., N-1)$ are isomorphic iff one can be obtained from the other by reflection and/or rotation.*

To justify this result we can argue in the following way.

If one of the triangulations can be obtained from the other by reflection and/or rotation, it is easy to see that they are isomorphic.

To prove the necessity, let $T_2 = f(T_1)$. Now, $(0, 1, ..., N-1, 0)$ is a hamiltonian cycle in T_1. As f is an isomorphism, $(f(0), f(1), ..., f(N-1), f(0))$ will also be a hamiltonian cycle in T_2. But, by result 1 given above, the only hamiltonian cycle in T_2 is $(0, 1, ..., N-1, 0)$. So $\{f(0), f(1), ..., f(N-1), f(0)\}$ is nothing but a reflection and/or rotation of $(0, 1, ..., N-1, 0)$. Thus, T_2 can be obtained from T_1 by reflection and/or rotation.

From these two results we derive the following two corollaries.

Corollary 5.2 : *If T_1 and T_2 are two isomorphic triangulations, then the unique central triangle of T_1 is identical to that of T_2.*

Proof : If T_1 and T_2 are isomorphic to each other, then T_1 can be obtained from T_2 by rotation and/or reflection. Such an operation can only affect the orientation of the central triangle of a triangulation, but not its shape and size. Hence the result. □

Corollary 5.3 : *Two triangulations belonging to two different classes cannot be isomorphic to each other.*

Proof : Immediate from corollary 5.2. □

Corollary 5.3 allows us to calculate the number of non-isomorphic triangulations for each class separately.

5.4.2 Counting

We will now derive expressions for enumerating non-isomorphic triangulations in each of the classes.

5.4.2.1 Counting Scalene Triangulations

Theorem 5.4 *The total number of non-isomorphic scalene triangulations of a convex N-gon is*

$$N_{ST} = \sum_{\substack{a+b+c=N \\ a<b<c<\lceil N/2 \rceil}} [\delta(a+1)\delta(b+1)\delta(c+1)]$$

Proof : Assume that the three sides of the central triangle are of length a, b, c (a, b, c are distinct), where $a+b+c=N$. Let us first fix the position of the central triangle in the polygon as $(0, a, a+b)$ as shown in Fig. 5.14. With this fixed position of the central triangle, the total number of possible triangulations is $\delta(a+1)\delta(b+1)\delta(c+1)$.

We claim that the triangulations which are included in the above count are non-isomorphic to each other. To prove this we assume the converse. Let T_1 and T_2 be two isomorphic triangulations present in the above count. Then by the second result mentioned in the subsection 5.4.1, T_1 can be obtained from T_2 by reflection and or rotation. As the central triangle is scalene, reflection or rotation will change the orientation of the central triangle. But this contradicts our hypothesis that the position of the central triangle is fixed at $(0, a, a+b)$.

For a given set of values $\{a, b, c\}$, $a+b+c=N$ (a, b, c are all distinct), it is enough to consider any one of the 6 possible central triangles, viz., $(0, a, a+b)$, $(0, a, a+c)$, $(0, b, a+b)$, $(0, b, b+c)$, $(0, c, a+c)$, $(0, c, b+c)$. All triangulations for the remaining position of the central triangle will be isomorphic to some member of the set already considered. So let us assume $a < b < c$, and consider the central triangle $(0, a, a+b)$. Thus, the total number of non-isomorphic scalene

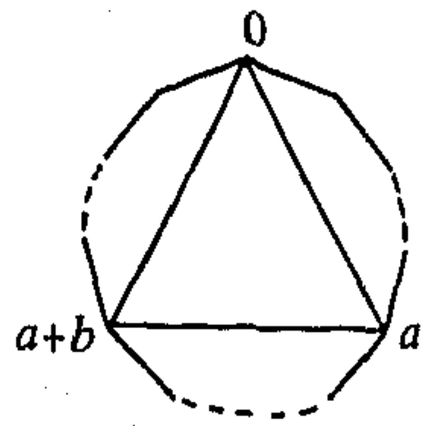


Figure 5.14: Counting scalene triangulations

triangulations of a convex N-gon will be :

$$\sum_{\substack{a+b+c=N \\ a<b<c<\lceil N/2 \rceil}} [\delta(a+1)\delta(b+1)\delta(c+1)]$$

□

Remark : With our notation defined above, the above result can further be simplified as :

$$N_{ST} = (1/3)D_{N-2} - (1/12)D_{N-1} - (1/4){D_{N/2-1}}^2 - (1/6){D_{N/3-1}}^3 - (1/2)\sum_{\substack{2a+b=N \\ a \neq b < \lceil N/2 \rceil}} [{D_{a-1}}^2 D_{b-1}]$$

(For derivation, see appendix at the end of chapter 5)

5.4.2.2 Counting Isosceles Triangulations

Theorem 5.5 *The total number of non-isomorphic isosceles triangulations of a convex N-gon is*

$$N_{IT} = \sum_{\substack{2a+b=N \\ a \neq b, a, b < \lceil N/2 \rceil}} (1/2)\{[\delta(a+1)]^2\delta(b+1) + \delta(a+1)\delta_E(b+1)\}$$

Proof : Let the two equal sides of the central triangle be of length a, and the other side be of length b. Thus, $2a + b = N$, $a \neq b$ and $a, b < \lceil N/2 \rceil$. Let us first fix the

position of the central triangle as $(0, a, a+b)$. With this fixed position of the central triangle, the total number of possible triangulations will be $[\{\delta(a+1)\}^2\delta(b+1)]$.

Since an isosceles triangle has a twofold symmetry, the above triangulations are non-isomorphic upto rotation only. In other words, the above set contains both T and T' (if they are not identical). Let us consider an isosceles triangulation $T\,(0, 1, ..., N-1)$, with central triangle $(0, a, a+b)$. Central triangle divides the original polygon into three smaller polygons, $(0, 1, \cdots, a)$, $(a, a+1, \cdots, a+b)$, and $(a+b, a+b+1, \cdots, N-1, 0)$, as shown in Fig. 5.15.

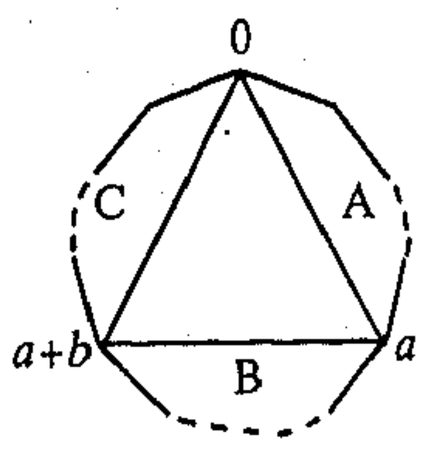


Figure 5.15: Counting isosceles triangulations

In any triangulation these three smaller polygons are also triangulated in some fashion. This is symbolically represented as :
$T\,(0, 1, ..., N-1) = (A(0, 1, ..., a)B(a, a+1, ..., a+b)C(a+b, a+b+1, ..., N-1, 0))$,
where A and C are any two triangulations of a polygon of size $(a+1)$ and B is that of a polygon of size $(b+1)$. In short, we also denote this by $T \equiv (ABC)$.

Now, $T'(0,\ 1,\ \cdots,\ N-1) = T\,(0,\ N-1,\ N-2,\ \cdots,\ 1)$
$= A(0,\ N-1,\ \cdots,\ a+b\,)B(a+b,\ a+b+1,\ \cdots,\ a)C(a, a-1, \cdots, 0)$
$= C^E(0,\ 1,\ \cdots,\ a)B^E(a,\ a+1,\ \cdots,\ a+b)A^E(a+b, a+b+1, \cdots, N-10)$
$\equiv (C^E B^E A^E)$

If T is a V-symmetric triangulation, then
$T\,(0, 1, ..., N-1) = T'(0, 1, ..., N-1) = T\,(0, N-1, N-2, ..., 1)$
Therefore, $C = A^E$ and $B = B^E$.
Thus, for T to be a V-symmetric triangulation, B should be E-symmetric and C

should be A^E.

Therefore, the total number of V-symmetric triangulations will be $\delta(a+1)\delta_E(b+1)$ which can be computed using Lemma 5.2.

Thus for the fixed position of the central triangle, the number of non-isomorphic triangulations will be $1/2\{[\delta(a+1)]^2\delta(b+1)+\delta(a+1)\delta_E(b+1)\}$.

Any change in the position of the central triangle, will generate a triangulation isomorphic to one that has already been considered in the above count.

Therefore, the total number of non-isomorphic isosceles triangulations will be

$$N_{IT} = \sum_{\substack{2a+b=n \\ a\neq b, a,b<\lceil N/2\rceil}} 1/2\{[\delta(a+1)]^2\delta(b+1)+\delta(a+1)\delta_E(b+1)\}$$

□

Remark : The above result can be simplified as :

$$N_{IT} = (1/2) \sum_{\substack{2a+b=N \\ a\neq b, a,b<\lceil N/2\rceil}} \{[D_{a-1}]^2 D_{b-1}\} + (1/2)D_{(N-1)/2-1} + (1/4)D_{N/2-1} - (1/4){D_{N/4-1}}^2 - (1/2)D_{N/3-1}D_{N/6-1}$$

(For proof, see appendix at the end of chapter 5)

5.4.2.3 Counting Equilateral Triangulations

Theorem 5.6 *The total number of non-isomorphic equilateral triangulations of a convex N-gon is*

$$N_{ET} = 1/6\{[\delta(N/3+1)]^3 + 2\delta(N/3+1) + 3\delta(N/3+1)\delta_E(N/3+1)\}$$

Proof : Let N be a multiple of 3. Without loss of generality, we fix the central triangle at $(0, N/3, 2n/3)$. With this central triangle, the total number of possible triangulations will be $[\delta(N/3+1)]^3$. Among these, let us consider a particular triangulation T such that

$$T(0,1,...,N-1) = (A(0,1,...,N/3)B(N/3,N/3+1,...,2n/3)$$
$$C(2n/3,2n/3+1,...,N-1,0)),$$

i.e., $T \equiv (ABC)$, where A, B, C are any three triangulations of a convex polygon of size $(N/3+1)$, as shown in Fig. 5.16.

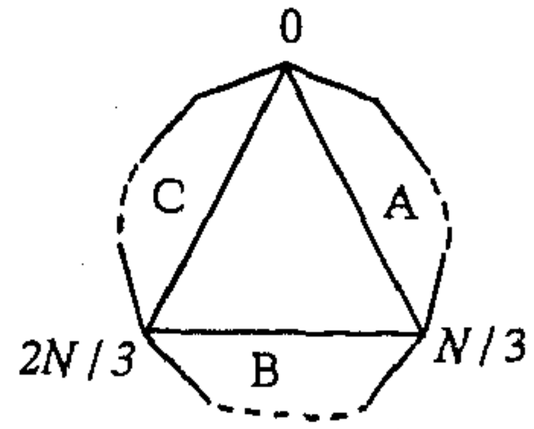


Figure 5.16: Counting equilateral triangulations

As the position of the central triangle is fixed, there are atmost five triangulations, namely, $T_1 = (BCA)$, $T_2 = (CAB)$, $T_3 = (C^E B^E A^E)$, $T_4 = (B^E A^E C^E)$, $T_5 = (A^E C^E B^E)$, all of which are isomorphic to T. But the triangulation T_i $(1 \leq i \leq 5)$ may be identical to T. If so, T_i has no contribution in the above count. Otherwise, it has been considered separately in the count.

If T_1 (or T_2) $= T$, then $A = B = C$. The number of such triangulations, for which T_1 (or T_2) is identical to T, is $\delta(N/3+1)$.

If $T_3 = T$, then $A = C^E$, $B = B^E$, $C = A^E$. The number of such triangulations will be $\delta(N/3+1)\delta_E(N/3+1)$.

Similarly, the number of triangulations for which T is identical to T_4 or T_5 will be $\delta(N/3+1)\delta_E(N/3+1)$.

Thus, the total number of non-isomorphic equilateral triangulations, with the central triangle $(0, N/3, 2n/3)$ will be $1/6\{[\delta(N/3+1)]^3 + 2\delta(N/3+1) + 3\delta(N/3+1)\delta_E(N/3+1)\}$.

For any triangulation generated by changing the position of the central triangle, has already been included in the above count.

Thus, the total number of non-isomorphic equilateral triangulations of $P(0,1,...,N-$

1) will be

$$N_{ET} = 1/6\{[\delta(N/3+1)]^3 + 2\delta(N/3+1) + 3\delta(N/3+1)\delta_E(N/3+1)\}$$

When N is not a multiple of 3, $N_{ET} = 0$. Therefore for any N- gon,

$$N_{ET} = (1/6){D_{N/3-1}}^3 + (1/3)D_{N/3-1} + (1/2)D_{N/3-1}D_{N/6-1}$$

□

5.4.2.4 Counting Bisector Triangulations

Theorem 5.7 *The total number of non-isomorphic bisector triangulations of a convex N-gon is*

$$N_{BT} = 1/4[\{\delta(N/2+1)\}^2 + 2\delta(N/2+1) + \{\delta_E(N/2+1)\}^2]$$

Proof : In this case, the length of the longest edge is $\lfloor N/2 \rfloor = \lceil N/2 \rceil$, that is N must be even.

Let us first fix the position of this edge as $(0, N/2)$.

With this fixed position of the longest edge, the total number of possible triangulations is $[\delta(N/2+1)]^2$.

Let us consider a triangulation T such that

$T(0, 1, ..., N-1) = (A(0, 1, ..., N/2)B(N/2, N/2+1, ..., N-1, 0))$

i.e., $T \equiv (AB)$, where A and B are any two triangulations of a convex polygon of size $(N/2+1)$, as shown in Fig. 5.17.

With the fixed position of the longest edge, the triangulation T has at most three isomers, namely $T_1 \equiv (BA)$, $T_2 \equiv (A^E B^E)$ and $T_3 \equiv (B^E A^E)$. But these triangulations may be identical to T. If they are not, then they must have been considered separately in the above count.

If $T = T_1$, then $A = B$. The number of such triangulations, (AA), is $\delta(N/2+1)$.

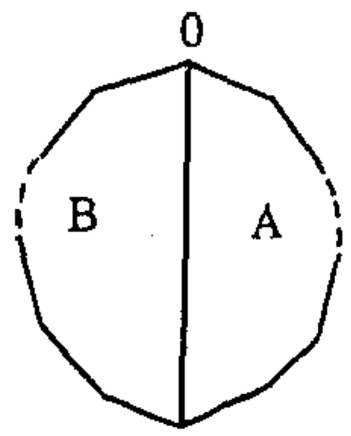


Figure 5.17: Counting bisector triangulations

If $T = T_2$, then $A = A^E$ and $B = B^E$. The number of such triangulations, is $[\delta_E(N/2+1)^2]$.

If $T = T_3$, then $A = B^E$ and $B = A^E$. The number of such triangulations, $(A^E A)$, is $\delta(N/2+1)$.

Thus, the total number of non-isomorphic bisector triangulations, with the fixed position of the longest edge, will be $1/4[\{\delta(N/2+1)\}^2+2\delta(N/2+1)+\delta_E(N/2+1)^2]$.

Triangulations obtained by changing the position of the longest edge will be isomorphic to one which has already been included in the above count.

Therefore, the total number of non-isomorphic bisector triangulations of a convex N-gon will be

$$N_{BT} = 1/4[\{\delta(N/2+1)\}^2 + 2\delta(N/2+1) + \{\delta_E(N/2+1)\}^2]$$

For odd values of N, $N_{BT} = 0$. Therefore, for all values of N

$$N_{BT} = (1/4){D_{N/2-1}}^2 + (1/2)D_{N/2-1} + (1/4){D_{N/4-1}}^2$$

□

5.4.2.5 Counting Total Number of Non-Isomorphic Triangulations

Theorem 5.8 *The total number of non-isomorphic triangulations of a convex polygon* $P(0, 1, ..., N-1)$ *is given by*

$$\delta^{NI}(N) = (1/12)[4D_{N-2} + 9D_{N/2-1} + 4D_{N/3-1} + 6D_{(N-1)/2-1} - D_{N-1}]$$

Proof : See appendix at the end of chapter 5. □

It may be noted here that the problem of counting the total number of triangulations, non-isomorphic upto reflection and rotation, has already been solved in [MM63] and [HPR75] for convex polygons. The following counting formula is due to Moon and Moser [MM63] :

$$\delta^{NI}(N) = \begin{cases} f(N)/2N + f(N/3+1)/3 + 3f(N/2+1)/4, & \text{if } N \text{ is even} \\ f(N)/2N + f(N/3+1)/3 + f((N+1)/2)/2, & \text{otherwise} \end{cases}$$

where, $f(N) = \delta(N)$, if $N \geq 2$

$= 0$, for fractional and/or negative values of $(N-2)$.

It can be verified that this result agrees with that given in the above theorem 5.8. But the approach used in [MM63] is different from that of ours which is based on counting the non-isomorphic triangulations of each class separately.

The number of non-isomorphic triangulations in each class as well as the total number of triangulations upto $N = 20$ are given in Table 5.1.

Table 5.1: Number of triangulations of different type for various values of N

N	N_{ST}	N_{IT}	N_{ET}	N_{BT}	$\delta^{NI}(N)$
3	0	0	1	0	1
4	0	0	0	1	1
5	0	1	0	0	1
6	0	0	1	2	3
7	0	4	0	0	4
8	0	3	0	9	12
9	10	15	2	0	27
10	0	26	0	56	82
11	70	158	0	0	228
12	140	105	25	463	733
13	1008	1274	0	0	2282
14	1176	1930	0	4422	7528
15	12180	12192	462	0	24834
16	20328	17339	0	46231	83898
17	150414	134943	0	0	285357
18	280962	177938	12404	511940	983244
19	1826682	1585738	0	0	3412420
20	3804372	2228001	0	5912241	11944614

5.5 Identification of the Class and the Central Triangle

In this section we look at the problem of identifying the class which a given triangulation belongs to along with the central triangle (for a non-bisector triangulation). For this we first find the longest edge(s) in the triangulated polygon. If the longest edge is of length $\frac{N}{2}$ (which can occur only when N is even), then the given triangulation is a bisector one. Otherwise, we find the unique central triangle in the given non-bisector triangulation by using the fact that the longest edge in G_T must be one of the sides of the central triangle of T (from corollary 5.1).

The complete process can be described by the three steps given below.

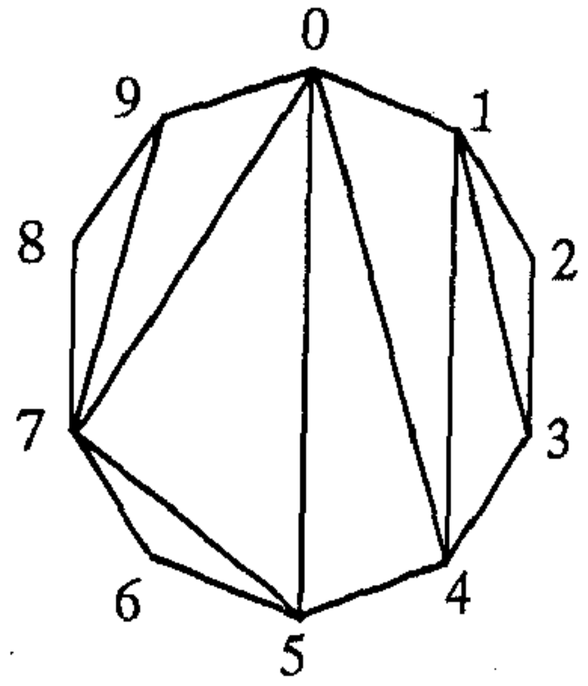
(I) Find the longest edge emanated from the vertices i of G_T, $\forall i,\ 0 \leq i \leq N-1$.

(II) Among N edges obtained in (I), find again the longest one. Let it be (p, q).

(III) If $d(p, q) = \frac{N}{2}$, then the triangulation T is a bisector triangulation; otherwise, for each vertex k, other than p and q, find $d(k, p) + d(k, q) + d(p, q)$. If the sum is N then (k, p, q) is the central triangle.

We assume that the planar straight line embedding of G_T is given in the form of ordered adjacency lists. By ordered adjacency list we mean that the nodes in the list will appear in a specific direction, either in the direction of increasing node number or decreasing node number and for a node i, the list will start either from $i+1$ or $i-1$ ($\pm$ are taken under modulo N). An example is shown in Fig.5.18.

Let us now formally describe the algorithm *ICCT* for identification of the class and the central triangle of the triangulation when implemented with N processors. These processors will be referred to as $P_0, P_1, \ldots, P_{N-1}$ so that the processor P_i will take care of the vertex i.



Ordered adjacency lists for the nodes :

0 : 1, 4, 5, 7, 9

1 : 2, 3, 4, 0

2 : 3, 1

3 : 4, 1, 2

4 : 5, 0, 1, 3

5 : 6, 7, 0, 4

6 : 7, 5

7 : 8, 9, 0, 5, 6

8 : 9, 7

9 : 0, 7, 8

Figure 5.18: An example of representing a graph by ordered adjacency lists

Algorithm *ICCT*

Input : The maximal outerplanar graph $G_T\left(V, \epsilon\right)$ corresponding to the given triangulation T. G_T is given in the form of ordered adjacency lists.

Output : The class which T belongs to and also the central triangle in case of non-bisector triangulation.

Step 1 : for $i = 0$ to $N-1$ **do in parallel**

from the adjacency list of the node i, the processor P_i finds the largest chord (i, k) of length l_i emanating from the vertex i, where $k \in V$ is some vertex in G_T. If there are more than one such edge with the same length l_i, then only that corresponding to the minimum k value will be chosen. P_i generates the triplet $< i, k, l_i >$.

Step 2 : P_i's find the lexicographic minimum of the triplets $< N - l_i, i, k >$. Let this minimum be $< N - l_s, s, t >$. The processor P_0 will store the triplet $< s, t, l_s >$.

Step 3 : If $l_s = \frac{N}{2}$ then output T as a bisector triangulation and stop.

Step 4 : The triplet $< s, t, l_s >$ is broadcast to all the $N-1$ processors P_i, $0 < i \leq N-1$.

Step 5 : for $i = 0$ to $N-1$ do in parallel

if the processor P_i finds that both the vertices s and t are neighbors of i, that is, if both the chords (i, s) and (i, t) exist in T then it will compute the perimeter π_i of the triangle (i, s, t) as $\pi_i = d(i, s) + d(i, r) + l_s$.
If $\pi_i = N$ then P_i will output the corresponding triangle (i, s, t) as the central triangle of the triangulation. Let the central triangle be (p, q, r), $0 \leq p, q, r \leq N-1$.

Step 6 : The processor P_0 will compute the chord lengths $l_1 = d(p, q)$, $l_2 = d(q, r)$ and $l_3 = d(r, p)$. Comparing l_1, l_2 and l_3, P_0 decides the class which T belongs to.

Timing Analysis

(1) When implemented on EREW PRAM model with N processors :

Since the neighbors of the node i, $0 \leq i \leq N-1$, are given in the form of an ordered list, as described above, the corresponding sequence of distances will be bitonic in nature. That is, if the given adjacency list for a node i is $\{v_0, v_1, \cdots, v_\delta\}$ and the corresponding distances are $\{ d_0, d_1, \cdots, d_\delta \}$ where, $d(i, v_j) = d_j$, then we must have $d_0 \leq d_1 \leq d_2 \leq \cdots \leq d_p \geq d_{p+1} \geq \cdots \geq d_\delta$, for some p. This characteristic is true for all nodes in G_T. Due to this characteristic of $\{d_i\}$, the largest among these distances can be found by a single processor in $\lceil \log_2(\delta) \rceil$ time steps through a binary search. Thus, in step 1, the processor P_i can find the largest chord emanated from the node i in at most $\lceil \log_2(N-1) \rceil$ time steps. Clearly steps 2 and 4 require $\lceil \log_2 N \rceil$ time steps. In step 5, each of the N processors would take at most $2\lceil \log_2(N-1) \rceil$ time steps in parallel. Step 6 requires constant time. Hence, in total the algorithm requires $O(\log N)$ time.

(2) When implemented on a hypercube with N processors :

In this case, the required time will be the same as above assuming that the adja-

cency list of the node i will be given as input to the processor P_i.

5.6 Isomorphism Problem for MOPs

We know that there exist a one-to-one correspondence between the MOPs and the planar embedding of triangulated convex polygons. If the nodes in a MOP are numbered in such a way that the sequence of nodes $\{0, 1, \cdots, N-1, 0\}$ describes the unique hamiltonian cycle of the MOP, then a MOP G can be equivalently represented by the triangulated convex polygon $T(0, 1, \cdots, N-1)$. We now recall the notion of *hamiltonian degree sequence* [BJM79] in a MOP as follows.

Definition 5.11 *For a MOP* $T(0, 1, 2, \cdots, N-1)$, *the sequence* $\{d_0, d_1, \cdots, d_{N-1}\}$ *of positive integers will be called the hamiltonian degree sequence of* T *iff* d_i *is the degree of the node* i, $\forall i$, $0 \leq i \leq N-1$.

From the properties of triangulation of a polygon [PS88], it may be noted that

i) $2 \leq d_i \leq N-1$

and ii) there exist at least two elements of the sequence whose values are 2.

For the MOP isomorphism problem we start from the point that two MOPs G_1 and G_2, having N nodes each, are given in terms of two triangulations T_1 $(0, 1, \cdots, N-1)$ and T_2 $(0, 1, \cdots, N-1)$. Hence, deciding whether G_1 and G_2 are isomorphic to each other amounts to testing the isomorphism of the two triangulations T_1 $(0, 1, \cdots, N-1)$ and T_2 $(0, 1, \cdots, N-1)$.

We now state the following result which is also mentioned in [BJM79] in some other form.

Result 3 : Two MOPs $T_1(0, 1, \cdots, N-1)$ and $T_2(0, 1, \cdots, N-1)$ are isomorphic to each other iff their hamiltonian degree sequences are identical under reflection and/or rotation.

For completeness of our discussion, an outline of the proof in terms of the proposed notion of the central triangle is, however, given as follows.

Result 2 given in the subsection 5.4.1 proves the necessity.

For the sufficiency part, we need to prove that given two hamiltonian degree sequences which are identical under reflection and/or rotation, the corresponding triangulations are isomorphic to each other. Since we are dealing with hamiltonian degree sequences which are identical under reflection and/or rotation, it suffices to show that given a hamiltonian degree sequence $\{d_0, d_1, \cdots, d_{N-1}\}$, the way of triangulating a polygon $P(0, 1, \cdots, N-1)$, satisfying the sequence $\{d_i\}$ is unique. For this, we proceed as follows.

In the degree sequence, the elements whose values are 2 represent nodes in the ears [PS88] of the triangulation. Considering the sequence $\{d_i\}$, one can easily place those ears in the polygon and these positions are uniquely identified from the degree sequence. Once these ears are identified, the degree-2 nodes can be deleted from the polygon to get a new polygon of size at most $N-2$. The degree sequence of this new polygon can be obtained from $\{d_i\}$ by removing all 2's and reducing the preceding and the following elements of each of these 2's in the sequence by 1. Let the modified degree sequence be $\{d'_0, d'_1, \cdots, d'_k\}$, where $k \leq N-2$. Now, our problem reduces to triangulating a polygon of size at most $N-2$ satisfying the degree sequence $\{d'_i\}$. Applying the same construction rule for the new polygon, the problem size can again be reduced by at least 2. If we go on repeating the same process, we shall end up either at a single line (for bisector triangulation) or at the unique central triangle. Every step of this process actually fixes up the edges of the triangulated polygon. Thus, the uniqueness of the triangulation satisfying $\{d_i\}$ is established.

The algorithm which we are going to present to test whether two MOPs are isomorphic or not is mainly based on the above result 3. Let the given degree sequences of T_1 and T_2 along their hamiltonian cycles be $\{d^1_0, d^1_1, d^1_2, \cdots, d^1_{N-1}\}$ and $\{d^2_0, d^2_1, d^2_2, \cdots, d^2_{N-1}\}$ respectively. We need to find out whether there exists any reflection and/or rotation under which these degree sequences will be identical. If

there exists one such transformation, for non-bisector triangulations, we can say by the corollary 5.2, that the central triangle of T_1 and T_2 will be identical under the same transformation. Let (i^1, j^1, k^1) and (i^2, j^2, k^2) be the central triangles of T_1 and T_2 respectively. If the central triangles (i^1, j^1, k^1) and (i^2, j^2, k^2) are not congruent to each other, then we can at once conclude that the triangulations are non-isomorphic. But, in case they are congruent to each other, different possible transformations (at most 6 in number) can easily be found out under one of which the degree sequences will be identical when T_1 and T_2 are isomorphic. Let us discuss the latter case in detail.

Without loss of generality (w.l.o.g.), let us assume that $l_1^1 = |i^1 - j^1|$, $l_2^1 = |j^1 - k^1|$, $l_3^1 = |k^1 - i^1|$ and $l_1^2 = |i^2 - j^2|$, $l_2^2 = |j^2 - k^2|$, $l_3^2 = |k^2 - i^2|$.

The triplets $< l_1^1, i^1, j^1 >$, $< l_2^1, j^1, k^1 >$ and $< l_3^1, k^1, i^1 >$ are sorted (say, in non-decreasing order) over the values of l_1^1, l_2^1 and l_3^1. Let the sorted sequence of the triplets be $< l_1, i_1, j_1 >$, $< l_2, j_1, k_1 >$ and $< l_3, k_1, i_1 >$.

Similarly, for T_2, starting with the triplets $< l_1^2, i^2, j^2 >$, $< l_2^2, j^2, k^2 >$ and $< l_3^2, k^2, i^2 >$ and sorting these over the values of l_1^2, l_2^2 and l_3^2, suppose we get the sorted triplets as $< l_1, i_2, j_2 >$, $< l_2, j_2, k_2 >$ and $< l_3, k_2, i_2 >$.

Depending on the values of l_1, l_2 and l_3, there may be three different cases :

Case 1 : When $l_1 \neq l_2 \neq l_3 \neq l_1$.

In this case, the only possible mapping on the set of nodes V_2 of T_2 under which the degree sequences may be identical to each other is, $M_s : V_2 \rightarrow V_2$ such that for $v \in V_2$, $M_s(v) = v + (i_1 - i_2)$.
This M_s actually represents a rotation through $(i_1 - i_2)$ nodes. Note that i_1 can be found from the intersection of the sets of vertices $\{i_1, j_1\}$ and $\{k_1 i_1\}$ corresponding to the triplets $< l_1, i_1, j_1 >$ and $< l_3, k_1, i_1 >$ of T_1. Similarly i_2 can be obtained from the triplets $< l_1, i_2, j_2 >$ and $< l_3, k_2, i_2 >$ of T_2.

The degree sequence $\{d_j^2\}$ will then be modified to $\{\bar{d}_j^2\}$ under M_s such that $\bar{d}_j^2 = d^2_{M_s^{-1}(j)}$.

If the modified degree sequence $\{\bar{d}_j^2\}$ becomes identical to $\{d_j^1\}$, that is, if $d_j^1 = \bar{d}_j^2, \forall j, 0 \leq j \leq N-1$, then T_1 and T_2 are isomorphic to each other. Otherwise they are non-isomorphic. □

Case 2 : When $l_1 = l_3 \neq l_2$.

In this case, there will be two possible mappings on the vertex set V_2 under one of which the degree sequences may be identical.

(i) $M_{i1} : V_2 \rightarrow V_2$, such that for $v \in V_2$, $M_{i1}(v) = v + (i_1 - i_2)$
This mapping represents a rotation through $(i_1 - i_2)$ nodes which will map i_2 to i_1, j_2 to j_1 and k_2 to k_1.

(ii) $M_{i2} : V_2 \rightarrow V_2$, such that for $v \in V_2$, $M_{i2}(v) = i_1 + i_2 - v$.
M_{i2} represents a rotation through $(i_1 - i_2)$ nodes followed by a reflection about the vertex i_1. This will map i_2 to i_1, j_2 to k_1 and k_2 to j_1.

If the modified degree sequence $\{d^2_{M_{i1}^{-1}(j)}\}$ or $\{d^2_{M_{i2}^{-1}(j)}\}$ becomes identical to $\{d_j^1\}$, then T_1 is isomorphic to T_2. Otherwise they are non-isomorphic. □

Case 3 : When $l_1 = l_2 = l_3$.

Here, the number of possible mappings on V_2 is six under any one of which the degree sequences will be identical if T_1 is isomorphic to T_2. These mappings are as follows.

(i) $M_{e1}(v) = v + (i_1 - i_2)$
This will map i_2 to i_1, j_2 to j_1 and k_2 to k_1.

(ii) $M_{e2}(v) = i_1 + i_2 - v$
M_2 will map i_2 to i_1, j_2 to k_1 and k_2 to j_1.

(iii) $M_{e3}(v) = v + (i_1 - j_2)$
This will map j_2 to i_1, k_2 to j_1 and i_2 to k_1.

(iv) $M_{e4}(v) = i_1 + j_2 - v$
M_4 will map j_2 to i_1, i_2 to j_1 and k_2 to k_1.

(v) $M_{e5}(v) = v + (i_1 - k_2)$

This will map k_2 to i_1, i_2 to j_1 and j_2 to k_1.

(vi) $M_{e6}(v) = i_1 + k_2 - v$

M_4 will map k_2 to i_1, j_2 to j_1 and i_2 to k_1.

Here also, the modification rule for the degree sequence of T_2 is same as in the above two cases. If the degree sequences of T_1 and T_2 are identical under any one of the above mappings then T_1 and T_2 are isomorphic to each other. Otherwise they are non-isomorphic. □

In case both T_1 and T_2 are bisector triangulations, there will be four possible mappings on V_2.

Suppose, (i_1, j_1) and (i_2, j_2) are the longest edges of the triangulations T_1 and T_2 respectively. Then, the mappings can be defined as :

(i) $M_{b1}(v) = v + (i_1 - i_2)$

This mapping represents a rotation through $(i_1 - i_2)$ nodes.

(ii) $M_{b2}(v) = i_1 + i_2 - v$

This represents a rotation through $(i_1 - i_2)$ nodes followed by a reflection about the vertex i_1.

Both the above mappings map i_2 to i_1 and j_2 to j_1.

(iii) $M_{b3}(v) = v + (j_1 - i_2)$

This mapping represents a rotation through $(j_1 - i_2)$ nodes.

(iv) $M_{b4}(v) = j_1 + i_2 - v$

This represents a rotation through $(j_1 - i_2)$ nodes followed by a reflection about the vertex j_1.

These two mappings map i_2 to j_1 and j_2 to i_1. □

Let us now present the algorithm in a formal way.

Algorithm *ISO_MOP*

Input : Two triangulations T_1 $(0, 1, \cdots, N-1)$ and T_2 $(0, 1, \cdots, N-1)$ corresponding to two maximal outerplanar graphs G_1 and G_2 along with the respective hamiltonian degree sequences $\{d_0^1, d_1^1, d_2^1, \cdots, d_{N-1}^1\}$ and $\{d_0^2, d_1^2, d_2^2, \cdots, d_{N-1}^2\}$ are given. The Mops are given in the form of the ordered adjacency lists.

Output : Decide whether G_1 and G_2 are isomorphic or not.

Step 1 : Using the algorithm *ICCT*, identify the classes which these two triangulations belong to. If these two triangulations belong to two different classes then the triangulations are non-isomorphic. Otherwise, in case of bisector triangulations, find the longest edges of both T_1 and T_2. In case of non-bisector triangulations, find the central triangles $\{i_1, j_1, k_1\}$ and $\{i_2, j_2, k_2\}$ of T_1 and T_2 respectively.

Step 2 : If the triangulations are bisector triangulations, then check whether the modified hamiltonian degree sequence of T_2 is identical to that of T_1 under any one of the four possible mappings M_b discussed above. If one such mapping is found then the triangulations are isomorphic. Otherwise, these triangulations are non-isomorphic triangulations.

Step 3 : If the triangulations are scalene triangulations, then modify the hamiltonian degree sequence $\{d_j^2\}$ using the rule given above, under the corresponding mapping M_s (case 1 above). If the modified degree sequence $\{\bar{d}_j^2\}$ is identical to $\{d_j^1\}$ then the triangulations are isomorphic. Otherwise, they are non-isomorphic.

Step 4 : If the triangulations are isosceles triangulations then modify $\{d_j^2\}$ using the same rule, under the two possible mappings M_i (case 2) and check whether any one of these modified degree sequences is identical to $\{d_j^1\}$. If these two are identical with respect to any one of the two mappings then the triangulations are isomorphic. Otherwise, they are non-isomorphic.

Step 5 : If the triangulations are equilateral triangulations, then there are six possible mappings M_e as discussed above (case 3). Following the same procedure as in step 4, if the modified degree sequence is found identical to $\{d_j^1\}$ with respect

to any one of the mappings then the triangulations are isomorphic. Otherwise, they are non-isomorphic.

Implementation of the algorithm

(1) On EREW PRAM model with N processors :

Step 1 requires $O(\log N)$ time steps as discussed in the previous section. Steps 2, 3, 4, 5 can be implemented on this model in $O(\log N)$ time. In each of the steps 2, 3, 4 and 5, the processor P_i will compute the inverse of the respective mappings (as discussed above) for i. For example, $M_s^{-1}(i)$ will be computed as $i - (i_1 - i_2)$ (discussed in case 1). Let it be j. P_i will then read the value of d_j^2 from the PRAM and compare it with d_i^1. All the processors will do the same work in parallel. It will take constant amount of time to implement. Finally, the results obtained by comparison will be gathered by the processor P_0 in $O(\log N)$ time and P_0 will take the decision whether the triangulations are isomorphic or not.

Hence, the total time taken to implement this algorithm on EREW PRAM model with N processors is $O(\log N)$.

(2) On a hypercube with N processors :

We assume that the processor P_i will take care of the node i, $0 \leq i \leq N-1$. Initially, the ordered adjacency list for the node i and the i^{th} elements d_i^1 and d_i^2 of the hamiltonian degree sequences of T_1 and T_2 will be given to the processor P_i.

Step 1 can be implemented on the hypercube in $O(\log N)$ time. To implement steps 2, 3, 4, and 5, here also the processor P_i will first compute the respective inverse mapping of i to get the value of j. It will then request the processor P_j to send the value of d_j^2. The value of d_j^2 will be compared with d_i^1 at the processor P_i. These results are gathered by the processor P_0 to take the final decision. These steps can be implemented on a hypercube in $O(\log N)$ time steps [JH89], [NS81]. Hence the required time is $O(\log N)$.

Remark : If instead of ordered adjacency lists, the graphs are given in the form of adjacency matrices along with the hamiltonian degree sequences, then also the algorithm *ICCT* can be implemented with the same time complexity. But, in this case we would need N^2 processors. This is because, we know that the maximum of N numbers can be found in $\lceil \log_2 N \rceil$ time using N processors. Therefore, in step 1 of the algorithm *ICCT*, to find the longest edge emanated from a vertex in $\lceil \log_2 N \rceil$ time steps, N processors will be required for each of the vertices. As a result, we would need N^2 processors altogether. Hence, to keep the time complexity of the algorithm *ISO_MOP* at $O(\log N)$, N^2 processors are required when the graphs are given in the form of adjacency matrices.

5.7 Conclusion

In this chapter, we have presented an algorithm for testing isomorphism of maximal outerplanar graphs (MOPs). If the MOPs are given in the form of ordered adjacency lists, the algorithm will take $O(\log N)$ time steps, when implemented on an EREW PRAM model as well as on a hypercube with N processors. We have solved this problem in the light of triangulation of convex polygons, since there exist a one-to-one correspondence between the MOPs and the planar embedding of triangulated convex polygons. As a by-product, we have studied some interesting properties of triangulated convex polygons which, in turn, hold for MOPs as well. These results may be of use in the field of computational geometry, robotics, etc.

Appendix

From Theorem 5.4,5.5, 5.6 and 5.7, we get,

$$\delta^{NI}(N) = N_{ST} + N_{IT} + N_{ET} + N_{BT} \cdots \qquad \text{(A)}$$

From the recurrence relation [G68] of Catalan numbers we have,

$$D_N = \sum_{k=0}^{N-1} D_{N-k-1} D_k$$

One can easily verify that, $\sum_{k=0}^{\lceil N/2-1 \rceil} D_{N-k-1} D_k = 1/2(D_N + D_{(N-1)/2^2})$ (5.3)

For any three positive integers a, b, c and a function f(a, b, c), we can write :

$$\sum_{a+b+c=N} f = \sum_{\substack{a+b+c=N \\ a,b,c<N/2}} f + 3 \sum_{\substack{a+b+c=N \\ a \geq N/2}} f$$

i.e., $$\sum_{a+b+c=N} f = \sum_{\substack{a+b+c=N \\ a,b,c \text{ are distinct} \\ a,b,c<N/2}} f + 3 \sum_{\substack{a+b+c=N \\ a=b \neq c \\ a,b,c<N/2}} f + \sum_{\substack{a+b+c=N \\ a=b=c=N/2}} f + 3 \sum_{\substack{a+b+c=N \\ a \geq N/2}} f$$

Therefore, $$\sum_{a+b+c=N} f = 6 \sum_{\substack{a+b+c=N \\ a<b<c<N/2}} f + 3 \sum_{\substack{a+b+c=N \\ a=b \neq c \\ a,b,c<N/2}} f + \sum_{\substack{a+b+c=N \\ a=b=c=N/3}} f + 3 \sum_{\substack{a+b+c=N \\ a \geq N/2}} f$$

Therefore, $$\sum_{\substack{a+b+c=N \\ a<b<c<N/2}} [\delta(a+1)\delta(b+1)\delta(c+1)] = 1/6 \sum_{a+b+c=N} [\delta(a+1)\delta(b+1)\delta(c+1)]$$
$$- 1/2 \sum_{\substack{2a+b=N \\ a \neq b<N/2}} [\delta(a+1)^2 \delta(b+1)]$$
$$- 1/6[\delta(N/3+1)]^3$$
$$- 1/2 \sum_{\substack{a+b+c=N \\ a \geq N/2}} [\delta(a+1)\delta(b+1)\delta(c+1)] \qquad (5.4)$$

Now, $$\sum_{a+b+c=N} [\delta(a+1)\delta(b+1)\delta(c+1)] = \sum_{a=0}^{N} \delta(a+1) \sum_{b+c=N-a} \delta(b+1)\delta(c+1)$$
$$= \sum_{a=0}^{N} \delta(a+1) f(N-a),$$

where, $f(k) = \sum_{b+c=k} \delta(b+1)\delta(c+1)$, for k> 1

$= 0,$ for k = 1

$= \sum_{b+c=k} D_{b-1}D_{c-1},$ for k > 1

Therefore, we get

$$f(k) = \begin{cases} D_{k-1}, & \text{for } k > 1; \\ 0, & \text{for } k = 1. \end{cases} \qquad (5.5)$$

Hence, $\sum_{a+b+c=N} [\delta(a+1)\delta(b+1)\delta(c+1)] = \sum_{a=0}^{N-2} [\delta(a+1) f(N-a)] + \delta(N) f(1)$

$= \sum_{a=0}^{N-2} D_{a-1}D_{N-a-1}$ [as f(1)= 0]

$= \sum_{a=0}^{N-1} [D_{a-1}D_{N-a-1}] - D_{N-2}$

$= D_{N-1} - D_{N-2} \qquad (5.6)$

Now, $\sum_{\substack{a+b+c=N \\ a \geq N/2}} [\delta(a+1)\delta(b+1)\delta(c+1)] = \sum_{a=\lceil N/2 \rceil}^{N} [\delta(a+1) f(N-a)]$

$= \sum_{a=\lceil N/2 \rceil}^{N-1} [D_{a-1}D_{N-a-1}] - D_{N-2}$ [using (5.5)]

$= \sum_{k=\lceil (N-1)/2 \rceil}^{N} [D_k D_{N-k-2}] - D_{N-2}$

$= 1/2(D_{N-1} + D^2_{N/2-1}) - D_{N-2}$ [using (5.3)]

$\cdots$ (5.7)

Using (5.4), (5.6) and (5.7) we get

$$N_{ST} = \sum_{\substack{a+b+c=N \\ a<b<c<N/2}} [\delta(a+1)\delta(b+1)\delta(c+1)] = 1/6(D_{N-1} - D_{N-2})$$
$$- 1/2 \sum_{\substack{2a+b=N \\ a\neq b<N/2}} [\delta(a+1)^2\delta(b+1)]$$
$$- 1/6[\delta(N/3+1)]^3$$
$$-1/2[1/2(D_{N-1} + D^2_{N/2-1}) - D_{N-2}]$$

$$\text{Now, } \sum_{2a+b=N} \delta(a+1)\delta_E(b+1) = \sum_{\substack{b\geq 0 \\ N}} \delta(N/2 - b/2 + 1)\delta_E(b+1)$$
$$= \sum_{b=0} D_{N/2-b/2-1}\delta_E(b+1)$$
$$= D_{(N-1)/2-1}\delta_E(2) + \sum_{b/2=1}^{N/2-1} D_{N/2-b/2-1}D_{b/2-1}$$
$$= D_{(N-1)/2-1} + D_{N/2-1} \qquad \cdots \quad (5.8)$$

$$\text{Again, } \sum_{2a+b+N} \delta(a+1)\delta_E(b+1) = \sum_{\substack{a+b/2=N/2 \\ b/2<N/4}} \delta(a+1)\delta_E(b+1) + \sum_{\substack{a+b/2=N/2 \\ b/2\geq N/4}} \delta(a+1)\delta_E(b+1)$$
$$\text{or, } \sum_{2a+b=N} \delta(a+1)\delta_E(b+1) = \sum_{\substack{a+b/2=N/2 \\ b/2<N/4}} \delta(a+1)\delta_E(b+1) + \sum_{\substack{a+b/2=N/2 \\ b/2\geq N/4}} \delta(a+1)\delta(b/2+1)$$
$$= \sum_{\substack{a+b/2=N/2 \\ b/2<N/4}} \delta(a+1)\delta_E(b+1) + \sum_{k=0}^{\lfloor N/4 \rfloor - 1} D_k D_{N/2-k-2}$$
$$= \sum_{\substack{a+b/2=N/2 \\ b/2<N/4}} [\delta(a+1)\delta_E(b+1)] + 1/2(D_{N/2-1} + D^2_{N/4-1})$$

(using (5.3))

$$\cdots \quad (5.9)$$

Using (5.8) and (5.9), we get

$$\sum_{\substack{2a+b=N \\ b<N/2}} \delta(a+1)\delta_E(b+1) = D_{(N-1)/2-1} + D_{N/2-1} - 1/2(D_{N/2-1} + D_{N/4-1}{}^2)$$

Therefore, $\sum_{\substack{2a+b=N \\ b<N/2, a\neq b}} \delta(a+1)\delta_E(b+1) = D_{(N-1)/2-1} + D_{N/2-1} - 1/2(D_{N/2-1} + D^2_{N/4-1})$

$$-D_{N/3-1}\delta_E(N/3+1)$$

Now, $N_{IT} = \sum_{\substack{2a+b=N \\ a\neq b, a,b<\lceil N/2\rceil}} 1/2[\delta(a+1)]^2\delta(b+1) + \delta(a+1)\delta_E(b+1)$

$$= \sum_{\substack{2a+b=N \\ a\neq b, a,b<\lceil N/2\rceil}} [\delta(a+1)^2\delta(b+1) + (1/2)D_{(N-1)/2-1} - 1/4(D_{N/2-1} + D_{N/4-1}{}^2)$$

$$-(1/2)D_{N/3-1}\delta_E(N/3+1)$$

Using simplified expressions obtained for $N_{ST} and N_{IT}$, expression (A) can further be simplified as

$$\delta^N(N) = 1/3D_{N-2} - 1/12D_{N-1} + 3/4D_{N/2-1} + 1/3D_{N/3_1} + 1/2D_{(N-1)/2-1}$$

$$= (1/12)(4D_{N-2} + 9D_{N/2-1} + 3D_{N/3-1} + 6D_{(N-1)/2-1} - D_{N-1})$$

Chapter 6

Lagrange Interpolation

6.1 Introduction

Let $y_1, y_2, \cdots, y_N$ be the given values of a function $F(x)$ at $x_1, x_2, \cdots, x_N$ respectively. Suppose, it is required to evaluate the value of $F(x)$ at the point $x = \overline{x}$.

The N-point Lagrange interpolation formula for this problem is as follows [H56] :

$$F(\overline{x}) = \pi(\overline{x}) \sum_i [y_i / \{(\overline{x} - x_i)\pi'(x_i)\}], \tag{6.1}$$

where, $y_i = F(x_i)$,

$$\pi(\overline{x}) = (\overline{x} - x_1)(\overline{x} - x_2)(\overline{x} - x_3) \cdots (\overline{x} - x_N),$$

and $$\pi'(x_i) = (x_i - x_1)(x_i - x_2) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_N).$$

Let us define $$t_{ij} = \begin{cases} (\overline{x} - x_i)/(x_j - x_i) & \text{if } i \neq j \\ y_i & \text{if } i = j \end{cases}.$$

With this definition, the formula 6.1 can be rewritten as :

$$F(\overline{x}) = \sum_{j=1}^{N} T_j, \tag{6.2}$$

where, $T_j = t_{1j} \times t_{2j} \times \cdots \times t_{Nj}$

A sequential algorithm with $O(N^2)$ time complexity for evaluating $F(\overline{x})$ according to the formula 6.2 is described as follows.

Sequential Algorithm

Input : $\overline{x}$, x_1, x_2, $\cdots$, x_N, y_1, y_2, $\cdots$, y_N

Output : $F(\overline{x})$

1. $F(\overline{x}) \leftarrow 0$;
2. **for** $j = 0$ **to** N-1 **do**
   begin
   prod $\leftarrow$ 1;
   for $i = 0$ **to** N-1 **do**
   begin
   if $i \neq j$ **then**
   prod $\leftarrow$ prod $\times$ $(\overline{x}$ - $x_i)$ / $(x_j$ - $x_i)$;
   end;
   $F(\overline{x}) = F(\overline{x})$ + prod $\times$ y_i;
   end

Goertzel [G94] has given a parallel algorithm for Lagrange interpolation with N points in $\lfloor \frac{N}{2} \rfloor + O(\log N)$ steps using N processors with an AT-value of $O(N^2)$. In this chapter, we propose a new parallel algorithm for Lagrange interpolation with $O(N/\log N)$ time using a mesh of tree architecture having $N(2\log_2 N - 1)$ processors with the same AT-value of $O(N^2)$ as that in [G94]. It will also be shown that the basic idea of this algorithm can be extended to the case when only $p(2\log_2 p - 1)$ processors are available, where $p = \frac{N}{K}$, k being any integer greater than 1, yet keeping the AT-value at $O(N^2)$.

6.2 Computational Model

In this section, we describe the model for the proposed parallel algorithm .

We assume that $N = 2^n$. We would use $N \times (2n - 1)$ processors which are interconnected in the following way :

(1) The processors are arranged in a two-dimensional array having $(2n-1)$ rows and N columns. P(i, j) denotes the processor placed at the intersection of the i^{th} row and the j^{th} column, $1 \leq i \leq (2n-1)$, $1 \leq j \leq N$.

(2) The processors in each column are interconnected in the form of a complete binary tree, rooted at the processor placed in the last row, i.e., in the $(2n-1)^{th}$ row. Thus, the processor $P(2n-m, j)$ is connected to the processors $P(2n-2m, j)$ and $P(2n-2m-1, j)$, when $1 \leq m \leq (n-1)$.

The tree formed by the processors in the j^{th} column is referred to as the j^{th} column tree. The processors in the rows $1, 2, \cdots, n$ are referred to as the leaf processors and those in the next $(n-2)$ rows will be termed as the internal processors of the corresponding column tree.

(3) The processors in the $(2n-1)^{th}$ row are also interconnected in the form of a binary tree, rooted at $P(2n-1, 1)$. The processor $P(2n-1, j)$ is connected to the processors $P(2n-1, 2j)$ and $P(2n-1, 2j+1)$, if they exist.

(4) Data inputting is done only through all the processors placed in the rows 1 through n and also in the $(2n-1)^{th}$ row.

(5) The final result will be outputted by the processor $P(2n-1, 1)$.

(6) All the processors have three local registers A, L and R. Referring to any column tree of processors, the registers L and R of an internal (non - leaf) processor, are used to store the data received from its left and right children respectively. Moreover, each of the processors in the rows $1, 2, \cdots, n$ has one extra register X, which is used to store the value of $\overline{x}$.

An example of the above model is shown in Fig. 6.1 for $N = 2^3$. The interconnected network is, in fact, a subgraph of mesh of trees with $N(2n-1)$ processors.

Let h_m denote the height of the processor $P(2n-m, j)$, with respect to the j^{th} column tree. Therefore,

$$\forall\, m,\ 1 \leq m \leq (n-1), \qquad h_m = \lfloor \log (2n-1) \rfloor - \lfloor \log\ m \rfloor.$$

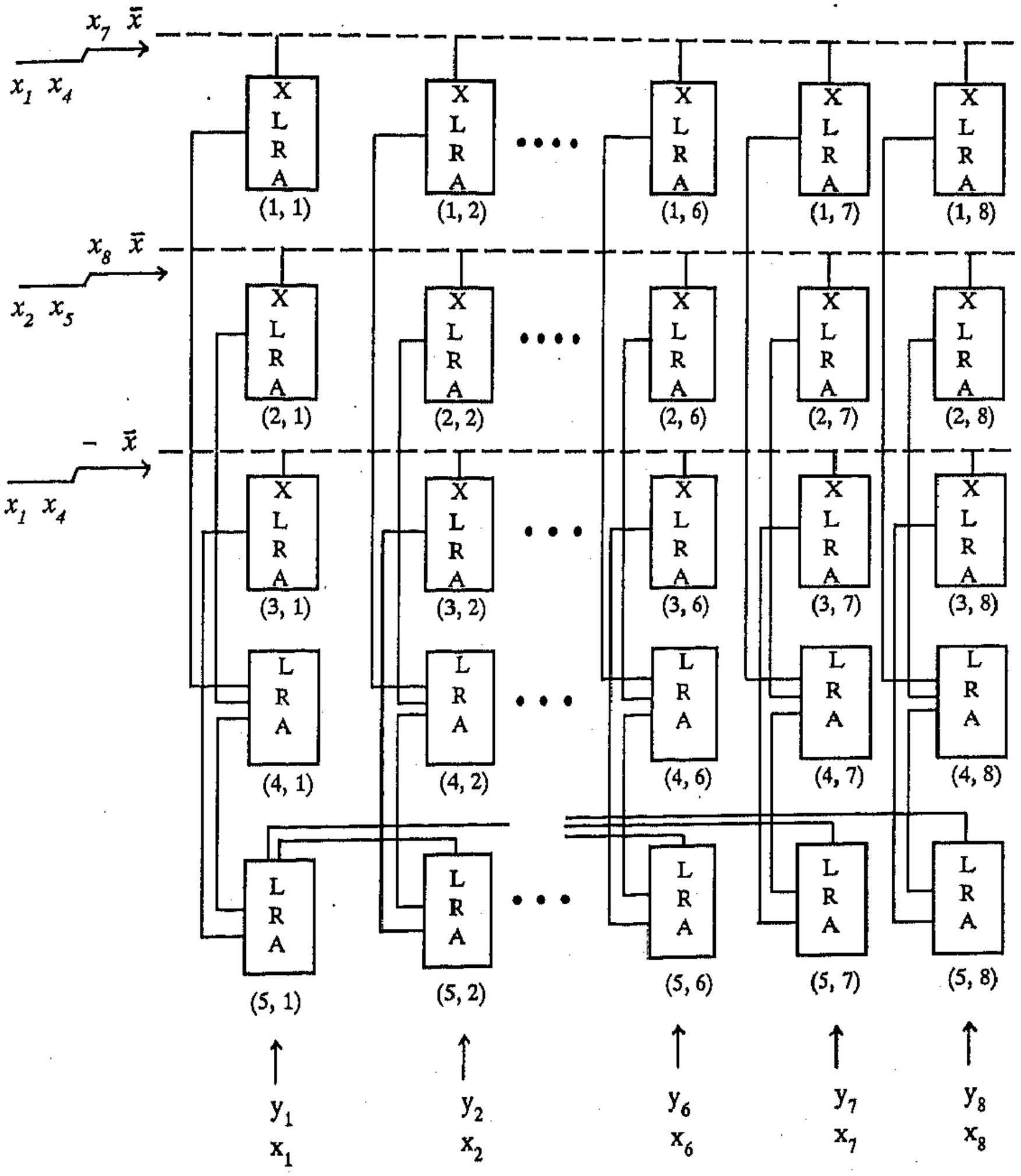


Figure 6.1: Computational model for $N = 2^3$ processors

6.3 Parallel Algorithm

The basic idea of the algorithm is as follows :

• The input data $x_1,\ x_2,\ \cdots,\ x_N$ are grouped in $(\lceil N/n \rceil)$ sets, denoted by S_1, $S_2,\ \cdots,\ S_{(\lceil N/n \rceil)}$. The set S_p, $1 \leq p < \lceil N/n \rceil$, contains n data points, namely, $x_{(p-1)n+1},\ x_{(p-1)n+2},\ \cdots,\ x_{pn}$. The last set may contain fewer elements. The elements of the last set $(S_{\lceil N/n \rceil})$ are $x_{(\lceil N/n \rceil - 1)n+1},\ x_{(\lceil N/n \rceil - 1)n+2},\ \cdots,\ x_N$.

• There are $\lceil N/n \rceil$ number of data inputting stages. In the p^{th} stage, the set $S_{\lceil N/n \rceil - p+1}$ is chosen and each of its data elements are fed in parallel to the leaf processors of each column tree, one input value along one row. That is, in the p^{th} data inputting stage, $1 \leq p \leq \lceil N/n \rceil$, the processor P$(i,\ j)$ receives $x_{(\lceil N/n \rceil - p)n+i}$, $\forall\ i, j,\ 1 \leq i \leq n,\ 1 \leq j \leq N$, whenever $(\lceil N/n \rceil - p)n + i \leq N$.

• The processors of the j^{th} column tree are used to compute the term T_j and the final value of T_j is stored in the root processor P$(2n-1,\ j)$ of that column tree. These T_j values are computed in parallel, $\forall\ j,\ 1 \leq j \leq N$.

Let us explain this step in a little more detail :

By definition, $\quad T_j = t_{1j} \times t_{2j} \times \cdots \times t_{Nj}$.
Thus, T_j is a product of N terms. These N terms are computed by the leaf processors. After receiving the data $x_{(\lceil N/n \rceil - p)n+i}$ in the p^{th} inputting stage, the leaf processor P$(i,\ j)$, $1 \leq i \leq n$, $1 \leq j \leq N$, computes t_{kj}, where $k = (\lceil N/n \rceil - p)n + i$, and sends this value to its parent. The final product is computed with the help of the remaining processors of the j^{th} column tree and eventually stored in the corresponding root processor.

• By using the tree connection among the root processors placed in the last row, these T_j values are summed up to get the final value of $F(\overline{x})$. The final result can be obtained from the processor P$(2n-1,\ 1)$.

Algorithm A :

Step 1 : $\forall\, i,\ j \quad 1 \leq i \leq 2n-1$ and $1 \leq j \leq N$,
initialize (in parallel) the contents of all the registers of
P$(i,\ j)$ to 1.

Step 2 : Do steps 2.1 and 2.2 in parallel

2.1 $\forall\, i,\ 1 \leq i \leq n$, and $\forall\, j,\ 1 \leq j \leq N$, P$(i,\ j)$ receives $\overline{x}$ (in parallel) and stores it in X$(i,\ j)$.

2.2 $\forall\, j,\ 1 \leq j \leq N$, the root processor P$(2n-1,\ j)$ of the j^{th} column tree receives y_j (in parallel) and stores it in A$(2n-1,\ j)$.

Step 3 : $\forall\, j,\ 1 \leq j \leq N$, the root processor P$(2n-1,\ j)$, of the j^{th} column tree receives x_j (in parallel) and broadcasts this value to all the leaf processors in that tree.
The leaf processors store this value in the respective L registers.

Step 4 : /* to compute T_j in the j^{th} column tree */

4.1 p $\leftarrow$ 1;

4.2 **repeat**

Do steps 4.2.1 and 4.2.2 in parallel

4.2.1 /* for leaf processors of each column tree */

if $1 \leq$ p $\leq \lceil N/n \rceil$ **then**

begin

$\forall\, i,\ 1 \leq i \leq n$ and $\forall\, j,\ 1 \leq j \leq N$, P$(i,\ j)$ executes the following in parallel

begin

/* inputting the set $S_{\lceil N/n \rceil - p+1}$ to the rows $1, 2, \cdots$ */

if $((\lceil N/n \rceil - p)n + i) \leq N$ **then**

begin

P(i, j) receives $x_{([N/n]-p)n+i}$;

if $x_{([N/n]-p)n+i} = x_j$ **then**

send 1 to the appropriate register of the parent

else

begin

compute $(\bar{x} - x_{([N/n]-p)n+i})/(x_j - x_{([N/n]-p)n+i})$;

if P(i, j) is the left (right) child of its parent **then**

send this value to the L (R) register of the parent

end;

end;

else send 1 to the appropriate register of the parent

end;

end;

4.2.2 /* for internal processors of each column tree */

$\forall\, m,\ 1 \le m \le n-1$ and $\forall\, j, 1 \le j \le N$, the processor P($2n-m$, j) executes the following in parallel

begin

if $h_m \le$ p $\le \lceil N/n \rceil + h_m$ **then**

/* h_m is the height of P($2n-m$, j) */

begin

if P($2n-m$, j) receives some data from its right child **then**

A($2n-m$, j) $\leftarrow$ L($2n-m$, j) * R($2n-m$, j)

else A($2n-m$, j) $\leftarrow$ L($2n-m$, j);

if $m \neq 1$ **then**

send the value contained in A(2n-m, j) to the L (R) processor of the parent if it is a left (right) child

end;

4.2.3 p $\leftarrow$ p+1;

until (p $> \lceil N/n \rceil + \lfloor \log (2n-1) \rfloor$)

Step 5 :

/* To sum up all T_j's [stored in A(2n - 1, j)] and to store the final result in A(2n - 1, 1) */

begin
for k = log N downto 1 **do**
begin
for j = 2^{k-1} to 2^k - 1 **do in parallel**
begin
if $2j \leq N$ **then**
begin
$L(2n-1,\ j) \leftarrow A(2n-1,\ 2j)$;
$A(2n-1,\ j) \leftarrow A(2n-1,\ j) + L(2n-1,\ j)$;
end;
if $(2j+1) \leq N$ **then**
begin
$R(2n-1,\ j) \leftarrow A(2n-1,\ 2j+1)$;
$A(2n-1,\ j) \leftarrow A(2n-1,\ j) + R(2n-1,\ j)$;
end
end
end
end.

Complexity of Algorithm A :

Step 3 in algorithm A requires $\lceil \log (2n-1) \rceil$ time and step 5 requires $\lceil \log N \rceil$ time. Time required to execute step 4 is $\lceil \lceil N/\log N \rceil + \lfloor \log (2n-1) \rfloor \rceil$ time. Hence the total time required to execute algorithm A is $\lceil (N/\log N) \rceil + 2 \log (2 \log N - 1) + O(1) = O(N/\log N)$.

Example 6.1 *The above algorithm has been illustrated with an example for* $N = 2^3$ *inputs. The arrangement of the 40 processors along with their respective registers is shown in Fig. 6.1. The contents of X, L, R and A*

registers of the processors in column 1, after executing the step 4.2.2 have been shown in Fig. 6.2 for different values of p. The don't care values are represented by dashes (-).

6.4 Scalability of the Algorithm

We would now modify the algorithm A for the case when only $\lceil p \times (2 \log p - 1) \rceil$ number of processors are available, where $p < N$. For simplicity, let us assume that $N = kp$, where k is an integer. The basic idea of this algorithm is as follows.

• The same model as described in section III is used, with $\lceil p \times (2 \log p - 1) \rceil$ number of processors.

• The $N(= kp)$ input values are grouped in $\lceil kp/\log p \rceil$ sets, namely,
$S_s = \{x_{(s-1)\log p+1}, x_{(s-1)\log p+2}, \cdots, x_{s \log p}\}$, $\forall s, 1 \leq s < \lceil kp/\log p \rceil$ and
$S_{\lceil kp/\log p \rceil} = \{x_{(\lceil kp/\log p \rceil - 1)\log p+1}, x_{(\lceil kp/\log p \rceil - 1)\log p+2}, \cdots, x_N\}$.

• As in algorithm A, each of the T_i terms is computed by the processors of a single column tree. But to compute $k \times p$ such terms, only p trees are available. So, each of these column trees is assigned to compute k such terms. Hence, the whole job is done in k stages, and in the r^{th} stage, the j^{th} column tree computes $T_{j+(r-1)p}$, $\forall r, 1 \leq r \leq k$.

The algorithm is formally described below. Only one more local register, named as SUM , is used by the processors in the $(2 \log p - 1)^{th}$ row and the final value can be obtained from the SUM register of P(2 log $p - 1$, 1).

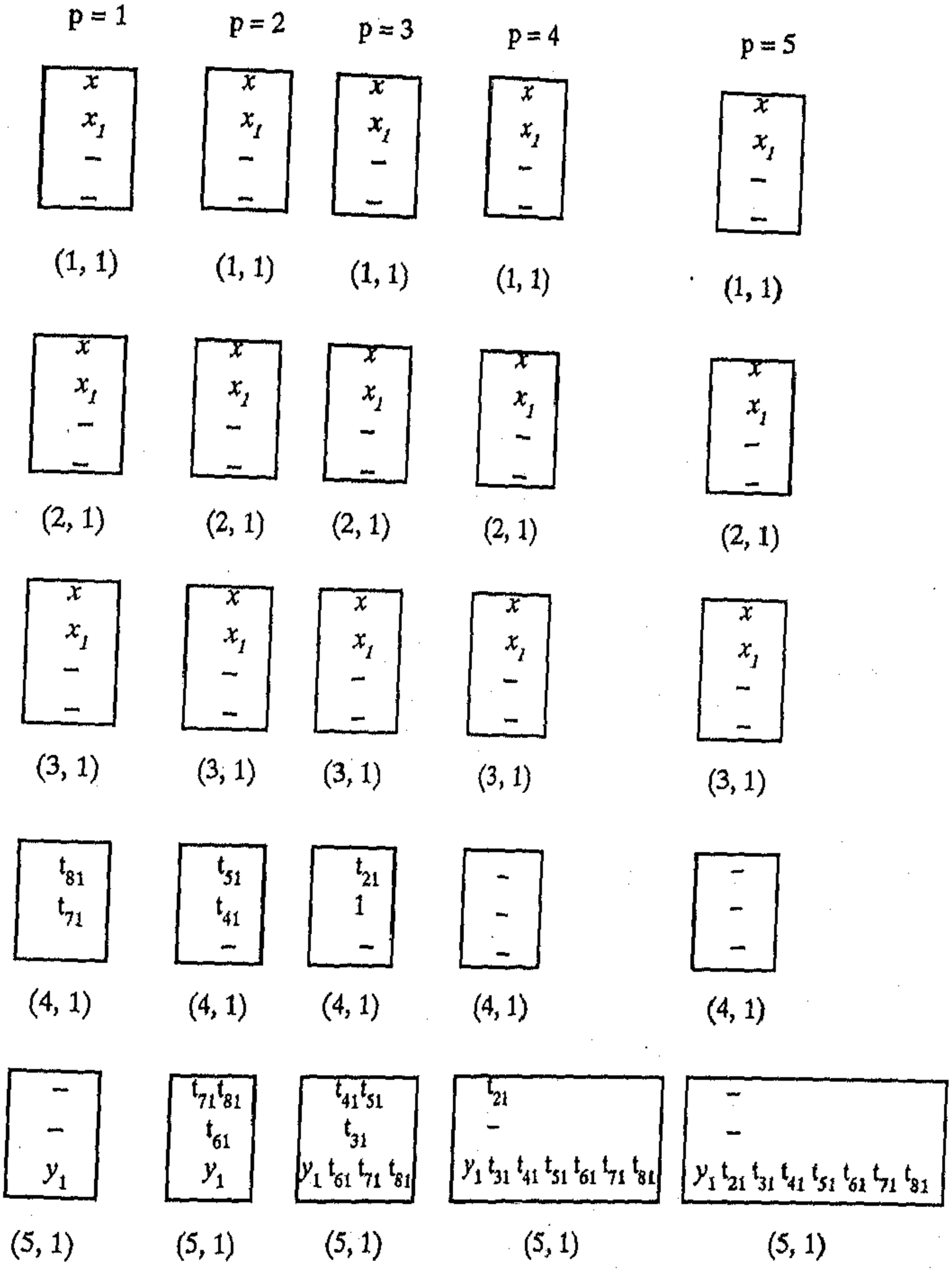


Figure 6.2: Contents of the registers of the processors in column 1 after step 4.2.2

Algorithm B :

Step 1 :

$\forall\, j,\ 1 \leq j \leq p$, initialize SUM$(2 \log\, p - 1,\ j) \leftarrow 0$.

Step 2 :

$\forall\, i,\ 1 \leq i \leq \log\, p$ and $\forall\, j,\ 1 \leq j \leq p$, P$(i,\ j)$ receives $\bar{x}$ (in parallel) and stores it in X$(i,\ j)$.

Step 3 :

for $r = 1$ to k **do**

begin

3.1 $\forall\, i,\ 1 \leq i \leq 2 \log\, p - 1$ and $\forall\, j,\ 1 \leq j \leq p$, initialize L$(i,\ j) \leftarrow 1$; R$(i,\ j) \leftarrow 1$; A$(i,\ j) \leftarrow 1$.

3.2 The input values $y_{(r-1)p+1},\ y_{(r-1)p+2},\ \cdots,\ y_{(r-1)p+p}$ are fed (in parallel) to the root processors of the column trees 1, 2, $\cdots$, p respectively and the values are stored in the A register of the corresponding processors.

3.3 The inputs $x_{(r-1)p+1},\ x_{(r-1)p+2},\ \cdots,\ x_{(r-1)p+p}$ are given to the columns 1, 2, $\cdots$, p respectively, through the corresponding root processors. Broadcast these values to the leaf processors in the respective column tree. The leaf processors store these values in their L registers.

3.4 /* to compute $T_{j+(r-1)p}$ in the j^{th} column tree */

3.4.1 s $\leftarrow 1$;

3.4.2 **repeat**

do steps 3.4.2.1 and 3.4.2.2 **in parallel**

3.4.2.1 /* for leaf processors of each column tree */

if $1 \leq$ s $\leq \lceil kp/\log\, p \rceil$ **then**

begin

$\forall\, i,\ 1 \leq i \leq \log\, p$, $\forall\, j,\ 1 \leq j \leq p$, P$(i,\ j)$ executes the following in parallel

begin

if $((\lceil kp/\log p\rceil - s)\log p + i) \leq N$ **then**

begin

receive $x_{(\lceil kp/\log p\rceil - s)\log p+i}$ and store it in R register;

if $x_{(\lceil kp/\log p\rceil - s)\log p+i} = x_j$ **then**

send 1 to the appropriate register of the parent

else

begin

compute $(\overline{x} - x_{(\lceil kp/\log p\rceil - s)\log p+i})/(x_j - x_{(\lceil kp/\log p\rceil - s)\log p+i})$;

if P(i, j) is the left (right) child of its parent **then**

send this value to the L (R) register of the parent

end

end

else send 1 to the appropriate register of the parent

end;

3.4.2.2 /* for internal and root processors of each column tree */

$\forall\, m, 1 \leq m \leq \log p - 1, \forall\, j, 1 \leq j \leq N,$ processor P$(2 \log p - m, j)$ executes the following in parallel, if it is not a leaf processor

begin

if $h_m \leq s \leq \lceil kp/\log p\rceil + h_m$ **then**

/* h_m is the height of P$(2 \log p - m, j)$ and is stored in P$(2 \log p - m, j)$ */

begin

if P$(2 \log p - m, j)$ receives some data from its right child **then**

A$(2 \log p - m, j) \leftarrow$ L$(2 \log p - m, j) \times$ R$(2 \log p - m, j)$

else A$(2 \log p - m, j) \leftarrow$ L$(2 \log p - m, j)$;

if $m \neq 1$ **then**

send the value contained in A$(2 \log p - m, j)$ to the L (R) register of the parent if it is a left (right)

child of its parent

end

3.4.2.3 $s \leftarrow s+1$;

until $(s > \lceil kp/\log p \rceil + \lfloor \log (2 \log p - 1) \rfloor)$

3.5 /* to store $\sum_{i=1}^{r} T_{j+(i-1)p}$ in SUM(2 log $p-1$, j) */

$\forall\, j,\ 1 \le j \le p$, the processor P(2 log $p-1$, j) executes the following in parallel

SUM(2 log $p-1$, j) $\leftarrow$ SUM(2 log $p-1$, j) + A(2 log $p-1$, j)

end;

Step 4 :

/* to sum up all T_j's, $\forall\, j,\ 1 \le j \le kp$ */

begin

for m = log p **downto** 1 **do**

begin

for j = 2^{m-1} to 2^m - 1 **do in parallel**

begin

if $2j \le p$ **then**

begin

L(2 log $p-1$, j) $\leftarrow$ SUM(2 log $p-1$, $2j$);

SUM(2 log $p-1$, j) $\leftarrow$ SUM(2 log $p-1$, j) + L(2 log $p-1$, j);

end;

if $(2j+1) \le p$ **then**

begin

R(2 log $p-1$, j) $\leftarrow$ SUM(2 log $p-1$, $2j+1$);

SUM(2 log $p-1$, j) $\leftarrow$ SUM(2 log $p-1$, j) + R(2 log $p-1$, j);

end

end

end

end.

Complexity of Algorithm B :

Step 3.3 requires $\lceil \log\ (2\ \log\ p - 1) \rceil$ time and the step 3.4 requires $\lceil kp/\log\ p \rceil$ + $\lfloor \log\ (2\ \log\ p - 1) \rfloor$ time. Hence, the total time required to execute the step 3 is $k[\lceil kp/\log\ p \rceil + 2\ \log\ (2\ \log\ p - 1)]$. Finally, the step 4 requires $\lfloor \log\ p \rfloor$ time. Hence, the total execution time for algorithm B is

$$k[\lceil kp/\log\ p \rceil + 2\ \log\ (2\ \log\ p - 1)] + \lfloor \log\ p \rfloor + O(1)\ =\ O(k^2 p/\log\ p)$$

The number of processors required, to implement this algorithm, is $O(p \log\ p)$. Hence, the AT-cost of this algorithm is $O(k^2\, p^2) = O(N^2)$.

6.5 Conclusion

A parallel algorithm for polynomial interpolation using Lagrange interpolation formula has been developed with $N(2 \log_2 N - 1)$ number of processors and $O(N/\log N)$ time complexity, where N is the number of input data points. It has also been shown how the underlying idea can be extended to the situation when fewer number of processors are available. It is found that in both the cases the AT cost is $O(N^2)$.

Chapter 7

Recursive Matrix Algorithms

7.1 Introduction

In this chapter, we would like to investigate the problem of parallel implementation of a specific class of recursive matrix algorithms on systolic architectures. This particular class consists of those algorithms which are based on partitioning the original matrix into submatrices of equal size and then performing the recursive calls on those smaller submatrices. Strassen's algorithm [S69] for matrix multiplication belongs to this class of recursive algorithms. In this algorithm, for multiplying two $n \times n$ matrices A and B, each of these matrices is first partitioned into four submatrices of size $\frac{n}{2} \times \frac{n}{2}$. The product is then obtained by recursively multiplying some derived submatrices of size $\frac{n}{2} \times \frac{n}{2}$ each and combining the results of these multiplications in a certain way. Another recursive algorithm for matrix inversion, due to Pease [P69], also uses a similar matrix partitioning approach.

The specific details of parallel implementation of such recursive algorithms definitely depends on the particular nature of the algorithm in question. However, as a representative case, we have considered here the Strassen's matrix multiplication algorithm for its parallel implementation on a hypercube architecture.

We will show that the parallel version of the Strassen's algorithm for multiplying two $n \times n$ matrices works in $O(\log n)$ time on a hypercube architecture having n^3

processors. It may be noted that the parallel version of the conventional matrix multiplication algorithm on a hypercube with n^3 processors also needs $O(\log n)$ time [A89]. Thus, although the sequential version of Strassen's algorithm is better than the conventional matrix multiplication algorithm (because of reducing the number of multiplications from 8 to 7 in case of multiplying two 2×2 matrices), the parallel version of both the algorithms have the same order of execution time. However, the main objective of this chapter is not to devise a more efficient matrix multiplication algorithm by parallelizing the Strassen's method; rather to investigate various issues related to the parallelization of such recursive algorithms on systolic architectures, through this example of Strassen's algorithm.

In what follows, we first describe the essential features of the Strassen's recursive method for matrix multiplication and then its parallel implementation on a hypercube architecture.

7.2 Strassen's Algorithm

Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be two $n \times n$ matrices to be multiplied to get the product matrix $C = [c_{ij}]$. To describe the Strassen's recursive method [S69] for matrix multiplication, we first partition the matrices A, B and C as

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}, \quad B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}, C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix},$$

where A_{ij}'s, B_{ij}'s and C_{ij}'s $(0 \leq i, j \leq 1)$ are $\frac{n}{2} \times \frac{n}{2}$ matrices each. To compute C, seven intermediate quantities are computed as follows:

$$
\begin{aligned}
M_1 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\
M_2 &= (A_{10} + A_{11})B_{00} \\
M_3 &= A_{00}(B_{01} - B_{11}) \\
M_4 &= A_{11}(B_{10} - B_{00})
\end{aligned} \qquad \cdots (7.1)
$$

$$
\begin{aligned}
M_5 &= (A_{00} + A_{01})B_{11} \\
M_6 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\
M_7 &= (A_{01} - A_{11})(B_{10} + B_{11})
\end{aligned}
$$

$$
\begin{aligned}
\textit{Then,}\quad C_{00} &= M_1 + M_4 - M_5 + M_7 \\
C_{01} &= M_3 + M_5 \\
C_{10} &= M_2 + M_4 \\
C_{11} &= M_1 + M_3 - M_2 + M_6
\end{aligned} \qquad \cdots (7.2)
$$

$$
\begin{aligned}
\textit{Let,}\quad X_1 &= A_{00} + A_{11}, & X_2 &= B_{00} + B_{11} \\
X_3 &= A_{10} + A_{11}, & X_4 &= B_{00} \\
X_5 &= A_{00}, & X_6 &= B_{01} - B_{11} \\
X_7 &= A_{11}, & X_8 &= B_{10} - B_{00} \\
X_9 &= A_{00} + A_{01}, & X_{10} &= B_{11} \\
X_{11} &= A_{10} - A_{00}, & X_{12} &= B_{00} + B_{01} \\
X_{13} &= A_{01} - A_{11}, & X_{14} &= B_{10} + B_{11}
\end{aligned} \qquad \cdots (7.3)
$$

In terms of these X_i's, we can write,

$$
\begin{aligned}
M_1 &= X_1 \times X_2 \\
M_2 &= X_3 \times X_4 \\
M_3 &= X_5 \times X_6 \\
M_4 &= X_7 \times X_8 \\
M_5 &= X_9 \times X_{10} \\
M_6 &= X_{11} \times X_{12} \\
M_7 &= X_{13} \times X_{14}
\end{aligned} \qquad \cdots (7.4)
$$

Thus, in all, there are seven matrix multiplications of the form $X_{2k+1}\ X_{2k+2}, k = 0, 1, \cdots, 6$ involving reduced size, i.e., $\frac{n}{2} \times \frac{n}{2}$ matrices and eighteen matrix additions/subtractions of reduced size $\frac{n}{2} \times \frac{n}{2}$.

To multiply two $n \times n$ matrices, this technique can be recursively employed by dividing the original $n \times n$ matrix into 4 submatrices of size $\frac{n}{2} \times \frac{n}{2}$. The second recursive call will involve 7 multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices. At the third recursive call, 7^2 multiplications of $\frac{n}{4} \times \frac{n}{4}$ matrices need to be performed and in general, at the l^{th} level of recursion, 7^{l-1} different sets of multiplication of $\frac{n}{2^{l-1}} \times \frac{n}{2^{l-1}}$ matrices are to be computed. The resulting time complexity of this recursive multiplication scheme is $O\ (n^{\ log 7}) = O(n^{\ 2.808})$.

7.3 Parallel Implementation of Strassen's Algorithm

7.3.1 Framework

Let us first estimate the total space required by the Strassen's algorithm. In each recursive call of this algorithm, the required storage space is increased to $\frac{7}{4}$ times of that needed in the previous recursive call. In effect, the storage requirement is finally increased to $(\frac{7}{4})^{\log n}$ of the initial $O(n^2)$ storage space for storing all the elements of the matrices A and B. Thus, we estimate that the total space requirement for parallel implementation is $O((\frac{7}{4})^{\log n} n^2)$, i.e., $o(n^3)$. Due to this estimate, we think of implementing the parallel version of Strassen's algorithm on an architecture having n^3 processors. Moreover, we see that after each recursive call, the storage space is almost getting doubled. This nature of the algorithm motivates us to select the hypercube as a possible architecture for its parallelization.

With this implementation idea in mind, we first give an overview of the proposed parallel algorithm on a hypercube followed by the detailed steps in case of a 2×2 matrix multiplication and also the formal description of the algorithm for $n \times n$

matrices.

7.3.2 Overview of the Algorithm

We note that once X_i's $1 \leq i \leq 14$ are computed, all the seven multiplications to compute M_i's can be carried out in parallel. However, instead of seven multiplications we compute eight matrix multiplications (M_1 is computed twice), although only 14 different X_i's are still used. It will not increase the time as the multiplications are all done in parallel. On the other hand, it will help to reduce the number of data communication steps in the final stage of matrix additions/subtractions. It must be mentioned that by doing so, we would in no way deviate from the essential idea of seven distinct multiplications in the Strassen's algorithm for 2×2 matrix multiplications. The final result is then obtained by performing additions/subtractions on these matrices.

Let us assume that $n = 2^q$. We implement the proposed parallel algorithm on a $3q$-dimensional hypercube having $N = 2^{3q} = n^3$ processors $P_0, P_1, \cdots, P_{n^3-1}$. All links are assumed to be two-way. However, at any instant of time, data will flow in only one direction along any link. Each processor will be designated by a three tuple (d, r, c), where, d, r and c are three integers such that, $0 \leq d, r, c \leq 2^q - 1$. That is, we can visualize that the processors are arranged in the form of an $n \times n \times n$ array, where the processor P_m occupies the position (d, r, c), such that $m = dn^2 + rn + c$. P_m will thus be represented by $P(d, r, c)$. A '$*$' in one or more of these d, r and c components will mean all possible values from 0 through $2^q - 1$ of the corresponding component(s) and this will signify some subcube of the hypercube. The processor P_m can also be equivalently represented by a bit string of length $3q$, obtained by concatenating the q-bit binary representations of each of d, r and c respectively (with no comma (,) separating these bit representations). However, a '$*$' in any bit position in this notation will mean both the values 0 and 1 for this bit. We will use both such notational schemes interchangeably in our later discussions; the specific scheme that has been used will be clear from the context.

We assume that each processor has two registers R_1 and R_2; $R_1(i,j,k)$ and $R_2(i,j,k)$ denote the contents of the registers R_1 and R_2 respectively of the processor $P(i,j,k)$. Initially, the elements of the matrices are stored in the processors in such a way that $R_1(0,r,c)$ contains the element a_{rc} and $R_1(1,r,c)$ contains the element $b_{c,q-r-1}$. In other words, the n^2 elements of the matrix A are initially stored in the $2q$-dimensional subcube $0^{q-1}0*^{2q}$, where 0^{q-1} represents a string of consecutive $q-1$ 0's, $*^q$ denotes a string of q *'s. Similarly, the n^2 elements of the matrix B are initially stored in the $2q$-dimensional subcube $0^{q-1}1*^{2q}$, as shown in Fig. 7.1

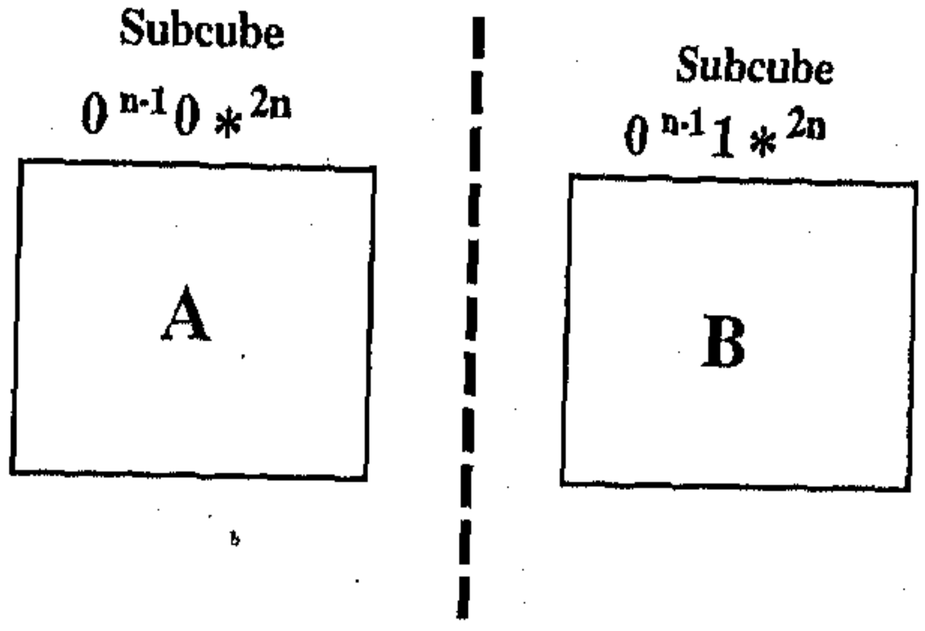


Figure 7.1: Initial distribution of the matrices over the subcubes

In what follows, we describe the parallel algorithm also in a recursive fashion. The overall idea of the algorithm goes in the following way. Every recursive call will consist of two different phases of action. In the first phase, the data elements are to be properly distributed along with some additions/subtractions among the processors to generate the X_i's as mentioned in equation (7.3). In the second phase, these X_i's are to be multiplied in the required manner as in equation (7.4). For this, first the X_i's are further suitably broadcast to some more processors and then the next level of recursive call is made. This is further explained in the following discussion.

Before starting the first level of recursion, the respective matrix elements are stored in the R_1 registers of the subcube $0^{q-1}*^{2q+1}$. During the first phase of this first level of recursion, the X_i values (14 different X_i's) will be computed in three parallel

steps and will be stored in the registers R_1 and R_2 of different processors in the following manner :

X_1 : $R_2(0^{q-1}0 \quad 0*^{q-1} \quad 0*^{q-1})$,
X_3 : $R_1(0^{q-1}0 \quad 1*^{q-1} \quad 0*^{q-1})$,
X_5 : $R_1(0^{q-1}0 \quad 0*^{q-1} \quad 0*^{q-1})$,
X_7 : $R_1(0^{q-1}0 \quad 1*^{q-1} \quad 1*^{q-1})$,
X_9 : $R_1(0^{q-1}0 \quad 0*^{q-1} \quad 1*^{q-1})$,
X_{11} : $R_2(0^{q-1}0 \quad 1*^{q-1} \quad 0*^{q-1})$,
X_{13} : $R_2(0^{q-1}0 \quad 0*^{q-1} \quad 1*^{q-1})$,
X_1 : $R_2(0^{q-1}0 \quad 1*^{q-1} \quad 1*^{q-1})$,

X_2 : $R_2(0^{q-1}1 \quad 0*^{q-1} \quad 0*^{q-1})$
X_4 : $R_1(0^{q-1}1 \quad 1*^{q-1} \quad 0*^{q-1})$
X_6 : $R_1(0^{q-1}1 \quad 0*^{q-1} \quad 0*^{q-1})$
X_8 : $R_1(0^{q-1}1 \quad 1*^{q-1} \quad 1^{q-1})$
X_{10} : $R_1(0^{q-1}1 \quad 0*^{q-1} \quad 1*^{q-1})$
X_{12} : $R_2(0^{q-1}1 \quad 1*^{q-1} \quad 0*^{q-1})$
X_{14} : $R_2(0^{q-1}1 \quad 0*^{q-1} \quad 1*^{q-1})$
X_2 : $R_2(0^{q-1}1 \quad 1*^{q-1} \quad 1*^{q-1})$

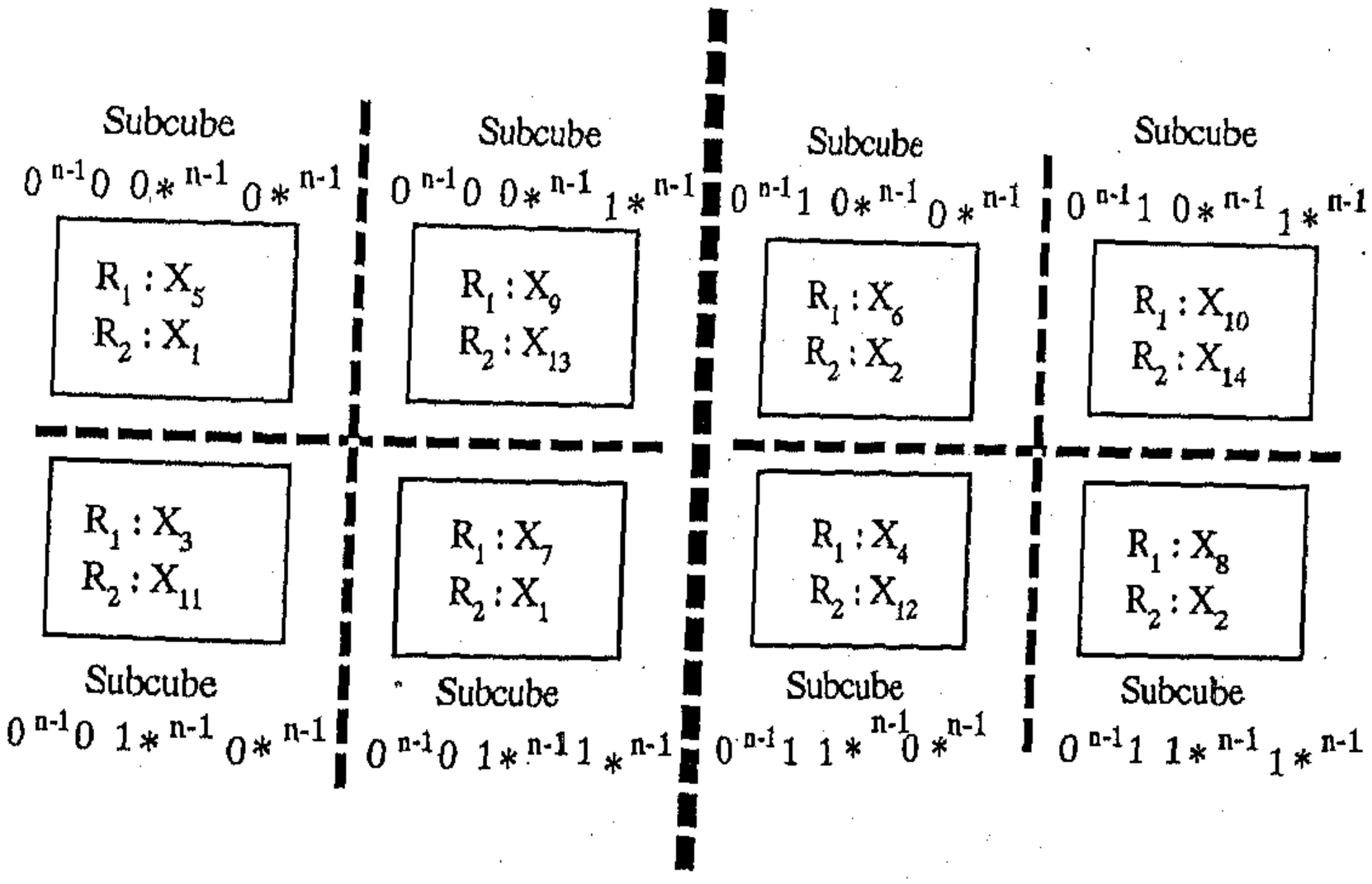


Figure 7.2: Distribution of X_i's over different subcubes
(After the first phase of the first level of recursion)

The scheme is illustrated in Fig. 7.2. Note that each of X_1 and X_2 is stored at two different processors. To perform the multiplications $X_{2k+1}\ X_{2k+2}$, $k = 0, 1, \cdots, 6$, the contents of the registers R_2 (i.e., submatrices $X_1, X_2, X_{11}, X_{12}, X_{13}$, and X_{14}) in the subcube $0^{q-2}0*^{2q+1}$ will be sent to the R_1 registers of those processors in

the subcube $0^{q-2}\ 1\ *^{2q+1}$ whose bit representations differ only in the $(2q+1)^{th}$ bit (the rightmost bit is assumed to be the 0^{th} bit); while the contents of the registers R_1 (i.e., the submatrices $X_3, X_4, X_5, X_6, X_7, X_8, X_9$ and X_{10}) of the subcube $0^{q-2}\ 0\ *^{2q+1}$ are retained in their original positions as shown in Fig. 7.3. Note that, in Fig. 7.3, the subcubes have been designated by following the (d, r, c) notation. Also note that $X_{2k+1}\ X_{2k+2}$, $k = 0, 1, \cdots, 6$ now reside in the subcubes differing only in the $2q^{th}$ bit position. After this, the second recursive call is made to multiply X_{2k+1} by X_{2k+2}, $k = 0, 1, \cdots, 6$. In this second level of recursion eight

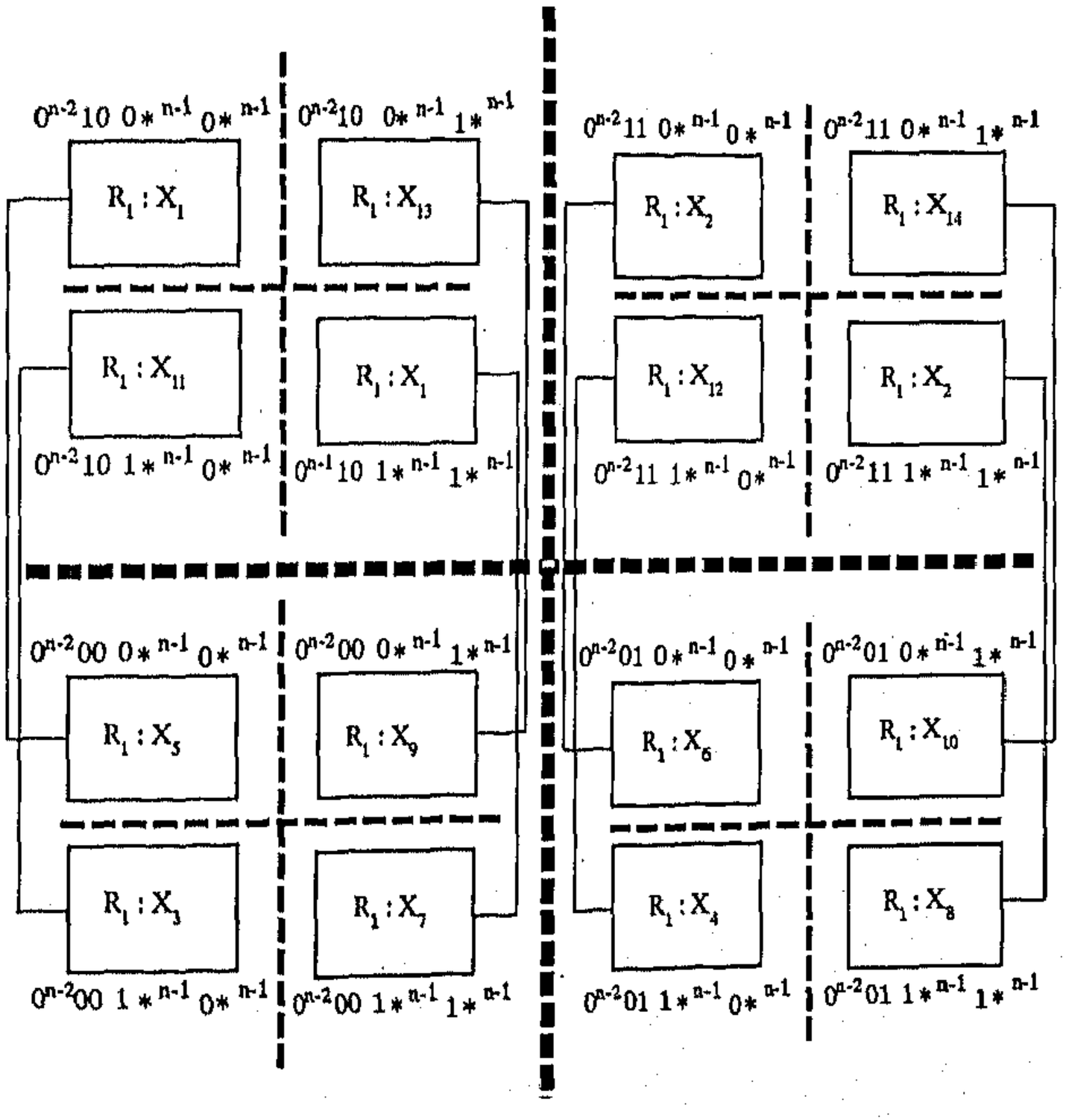


Figure 7.3: Contents of R_1 in different subcubes before second recursive call

matrix multiplications of dimension $\frac{n}{2} \times \frac{n}{2}$ are to be performed in parallel. In general, before the l^{th} recursion ($l \geq 1$), all the elements to be multiplied will

reside in the R_1 registers of the processors $0^{q-l}*^{2q+l}$, and in the l^{th} recursive call 8^{l-1} matrix multiplications of dimension $\frac{n}{2^{l-1}} \times \frac{n}{2^{l-1}}$ are to be performed in parallel. More specifically, the contents of the registers $R_1(0^{q-l}\beta_{2q+l-1} \cdots \beta_{2q+1}0 \quad \beta_{2q-1} \cdots \beta_{2q-l+1}*^{q-(l-1)} \quad \beta_{q-1} \cdots \beta_{q-l+1}*^{q-(l-1)})$ are to be multiplied with the contents of the registers $R_1(0^{q-l}\beta_{2q+l-1} \cdots \beta_{2q+1}1 \quad \beta_{2q-1} \cdots \beta_{2q-l+1}*^{q-(l-1)} \quad \beta_{q-1} \cdots \beta_{q-l+1}*^{q-(l-1)})$, for all possible combinations of β_i's where, $\beta_i = 0$ or 1 for all i; a particular combination of the values of these $3(l-1)$ β_i's correspond to one of these 8^{l-1} matrix multiplications. The overall scheme for distribution of data elements at the various recursive calls is schematically shown in Fig. 7.4 where the subcubes are denoted by (d, r, c) notation. Before the last, i.e., q^{th} recursive call, all the elements to be multiplied will reside in the R_1 registers of the processors $*^{3q}$ and now 8^{q-1} matrix multiplications of dimension 2×2 each are to be performed. At this stage, since each X_i is a single element, 8^{q-1} sets of $X_{2k+1}X_{2k+2}$, $k = 0, 1, \cdots, 6$, can be computed in parallel without further data distribution. Thus, we get 8^{q-1} sets of M_i's, $i = 1, \cdots, 7$ according to equation (7.4). From this 8^{q-1} sets of M_i's, 8^{q-1} sets of C_{ij}'s $(0 \leq i, j \leq 1)$ will be formed (as in equation (7.2)) and will be stored in the processors $*^{q-1}0*^{2q}$, which can also be equivalently represented by $(2k, *, *)$ (in terms of the (d, r, c) notation) where, $k = 0, 1, \cdots, 2^{q-1}-1$. These C_{ij}'s are again the M_i values for bigger matrices of dimension 4×4 in the immediate previous recursive call. From these M_i's, C_{ij}'s corresponding to the recursive call for 4×4 matrix multiplications will be computed and they will be stored in the R_1 registers of the subcube $0*^{q-2}0*^{2q}$. In general, the M_i values corresponding to l^{th} recursion $(1 \leq l \leq q)$, residing in the subcube $(2k, *, *)$ $k = 0, 1, \cdots, 2^l - 1$, will be used to compute the C_{ij}'s and these C_{ij}'s will be stored in the subcubes $(2k', *, *)$, where $k' = 0, 1, \cdots, 2^{l-1}-1$. These C_{ij}'s will, in turn, become the M_i values for the $(l-1)^{th}$ recursion. This scheme of storing C_{ij} values corresponding to a specific level of recursive call, is shown in Fig. 7.5. Finally, the product matrix C of order $n \times n$ will be stored in the R_1 registers of the processors $(0, *, *)$.

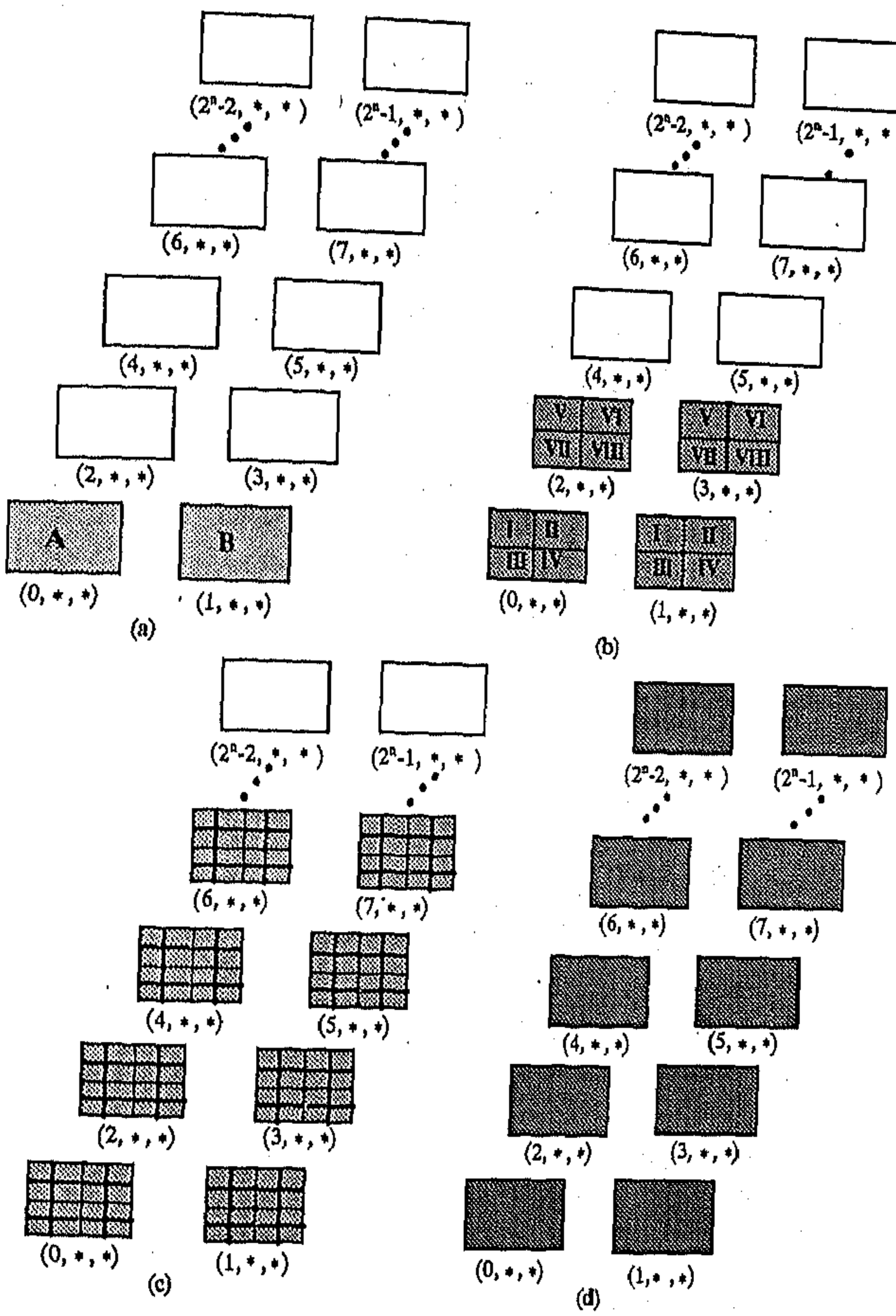


Figure 7.4 : Data elements in the subcubes at different levels of recursion
(a) before the first recursive call
(b) before the second recursive call
(c) before the third recursive call

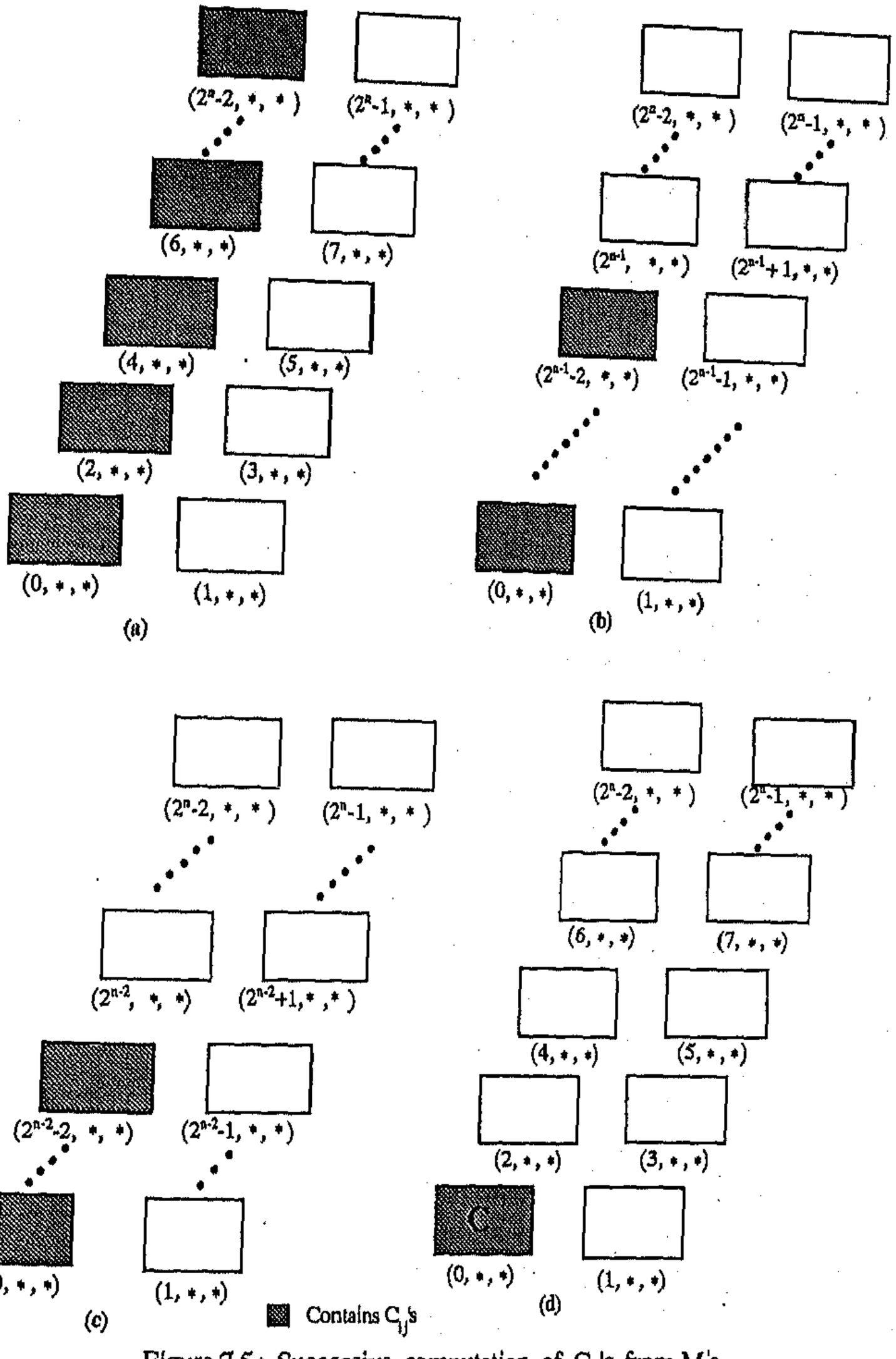


Figure 7.5 : Successive computation of C_{ij}'s from M_i's
(a) C_{ij}'s corresponding to n^{th} recursive call
(b) C_{ij}'s corresponding to $(n-1)^{th}$ recursive call
(c) C_{ij}'s corresponding to $(n-2)^{th}$ recursive call
(d) C_{ij}'s corresponding to 1^{st} recursive call

7.3.3 Detailed Steps for 2×2 Matrices

From the above discussions, it follows that at the q^{th} recursive call, the 2×2 matrix elements in $R_1(\beta_{3q-1} \cdots \beta_{2q+1} 0 \quad \beta_{2q-1} \cdots \beta_{q+1} * \quad \beta_{q-1} \cdots \beta_1 *)$ are to be multiplied by the 2×2 matrix elements in $R_1(\beta_{3q-1} \cdots \beta_{2q+1} 1 \quad \beta_{2q-1} \cdots \beta_{q+1} * \quad \beta_{q-1} \cdots \beta_1 *)$. Let us denote the bit string $\beta_{3q-1} \cdots \beta_{2q+1}$ by v_1, $\beta_{2q-1} \cdots \beta_{q+1}$ by v_2 and $\beta_{q-1} \cdots \beta_1$ by v_3. That is, the elements of the 2×2 matrix, say $A' = [a'_{ij}]$, residing in $R_1(v_1 0 \ \ v_2 * \ \ v_3 *)$ are to be multiplied by those of say, $B' = [b'_{ij}]$, in $R_1(v_1 1 \ \ v_2 * \ \ v_3 *)$. The correspondence between the elements of 2×2 matrices A' and B' with these R_1 registers are as follows: $a'_{00}, a'_{01}, a'_{10}$ and a'_{11} are stored in $R_1(v_1 0 \ \ v_2 0 \ \ v_3 0)$, $R_1(v_1 0 \ \ v_2 0 \ \ v_3 1)$, $R_1(v_1 0 \ \ v_2 1 \ \ v_3 0)$ and $R_1(v_1 0 \ \ v_2 1 \ \ v_3 1)$ respectively. $b'_{00}, b'_{01}, b'_{10}$ and b'_{11} are stored in $R_1(v_1 1 \ \ v_2 1 \ \ v_3 0)$, $R_1(v_1 1 \ \ v_2 0 \ \ v_3 0)$, $R_1(v_1 1 \ \ v_2 1 \ \ v_3 1)$ and $R_1(v_1 1 \ \ v_2 0 \ \ v_3 1)$ respectively. This correspondence is shown diagrammatically in Fig. 7.6.

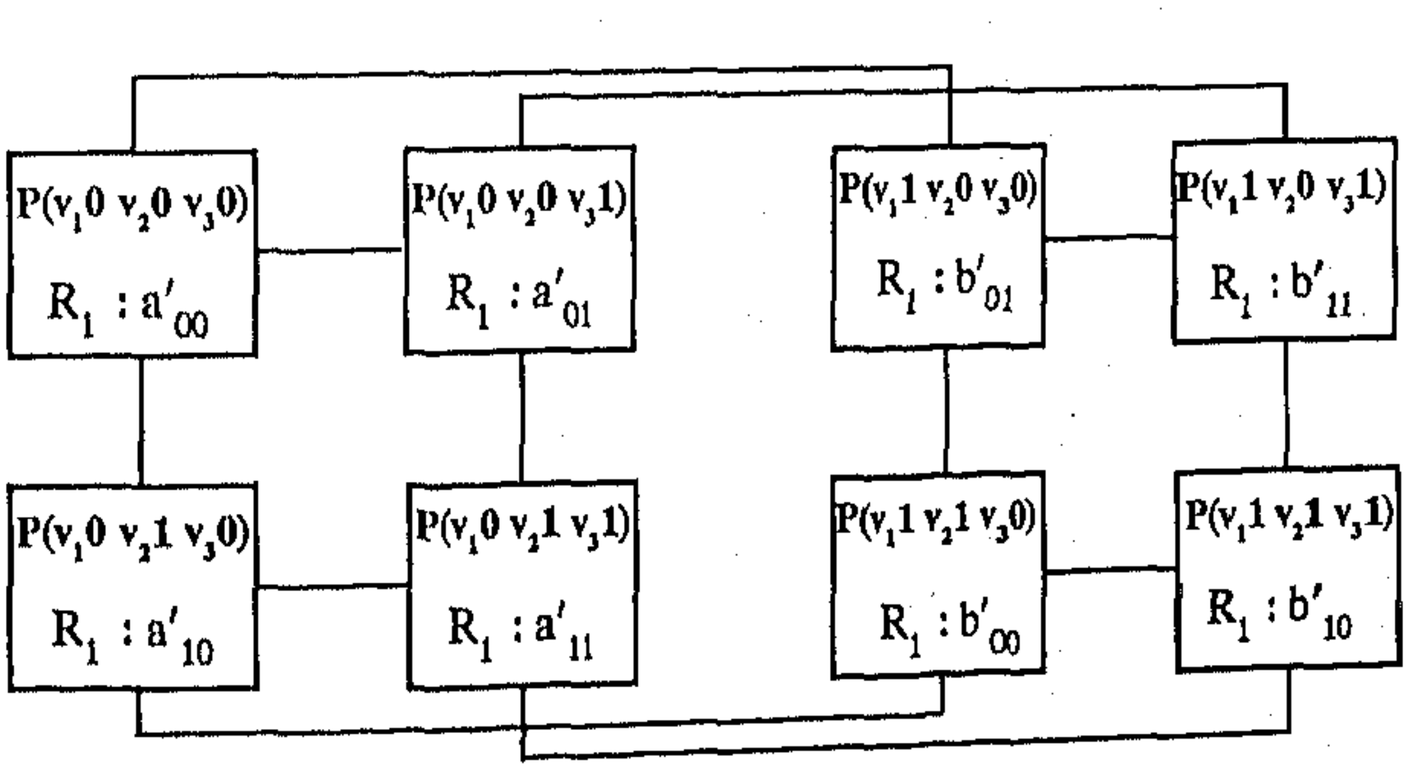


Figure 7.6: Initial contents of registers R_1

The detailed steps to be followed at this recursive call are shown below.

Step 1 : Do the following steps in parallel

$R_2(v_10\ v_20\ v_30) \leftarrow R_1(v_10\ v_21\ v_30)$ - $R_1(v_0\ v_20\ v_30)$;
$R_1(v_10\ v_20\ v_31) \leftarrow R_1(v_10\ v_20\ v_30) + R_1(v_10\ v_20\ v_31)$;
$R_1(v_10\ v_21\ v_30) \leftarrow R_1(v_10\ v_21\ v_30) + R_1(v_10\ v_21\ v_31)$;
$R_2(v_10\ v_21\ v_31) \leftarrow R_1(v_10\ v_20\ v_31)$ - $R_1(v_10\ v_21\ v_31)$;
$R_1(v_11\ v_20\ v_30) \leftarrow R_1(v_11\ v_20\ v_30)$ - $R_1(v_11\ v_20\ v_31)$;
$R_2(v_11\ v_20\ v_31) \leftarrow R_1(v_11\ v_21\ v_31) + R_1(v_11\ v_20\ v_31)$;
$R_2(v_11\ v_21\ v_30) \leftarrow R_1(v_11\ v_20\ v_30) + R_1(v_11\ v_21\ v_30)$;
$R_1(v_11\ v_21\ v_31) \leftarrow R_1(v_11\ v_21\ v_31)$ - $R_1(v_11\ v_21\ v_30)$;

/* The situation after this step is shown in Fig. 7.7. Arrows indicate the direction of data movements */

Step 2 : Do the following steps in parallel

$R_2(v_10\ v_20\ v_31) \leftarrow R_2(v_10\ v_21\ v_31)$;
$R_2(v_10\ v_21\ v_30) \leftarrow R_2(v_10\ v_20\ v_30)$;
$R_2(v_11\ v_20\ v_30) \leftarrow R_2(v_11\ v_21\ v_30)$ - $R_1(v_11\ v_20\ v_30)$;
$R_2(v_11\ v_21\ v_31) \leftarrow R_2(v_11\ v_20\ v_31)$ - $R_1(v_11\ v_21\ v_31)$;

/* Fig. 7.8 shows the situation after this step */

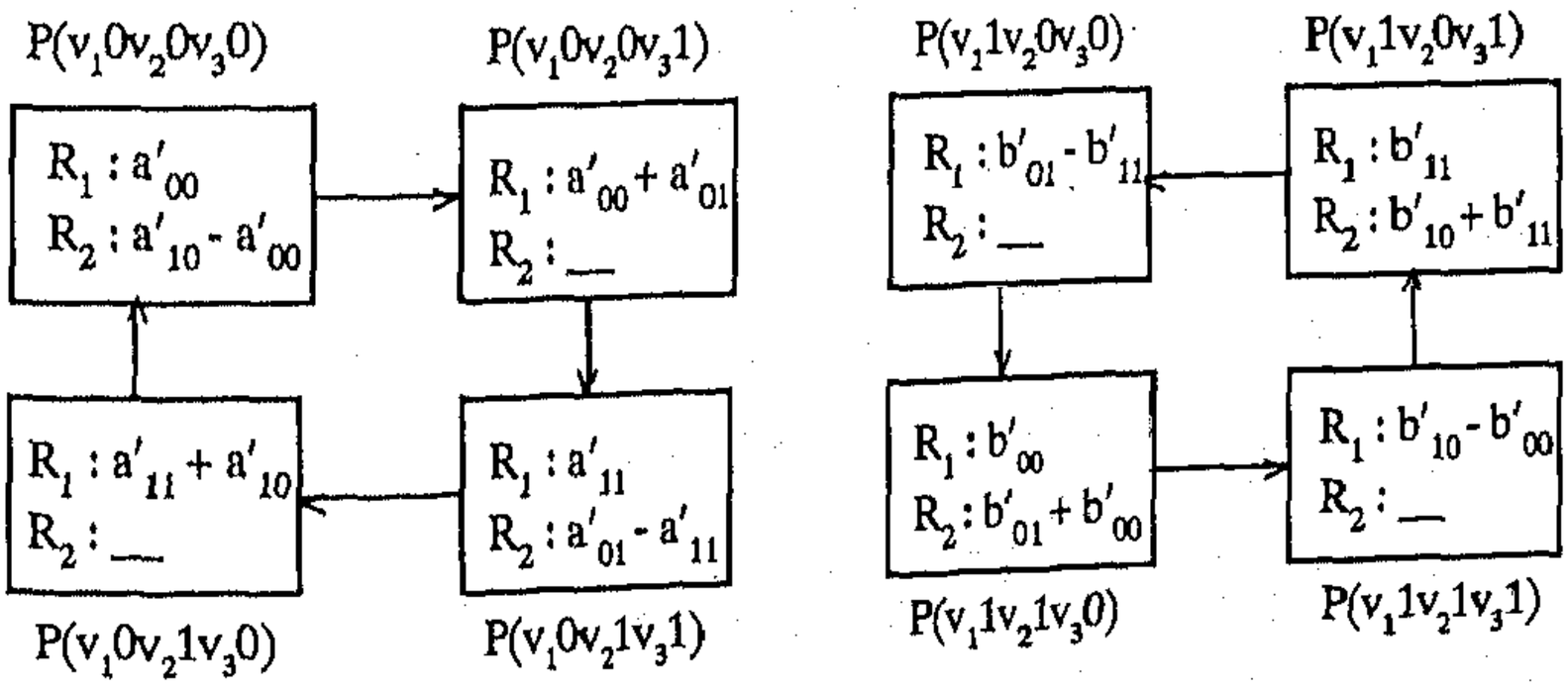


Figure 7.7: Contents of registers R_1 and R_2 after Step 1

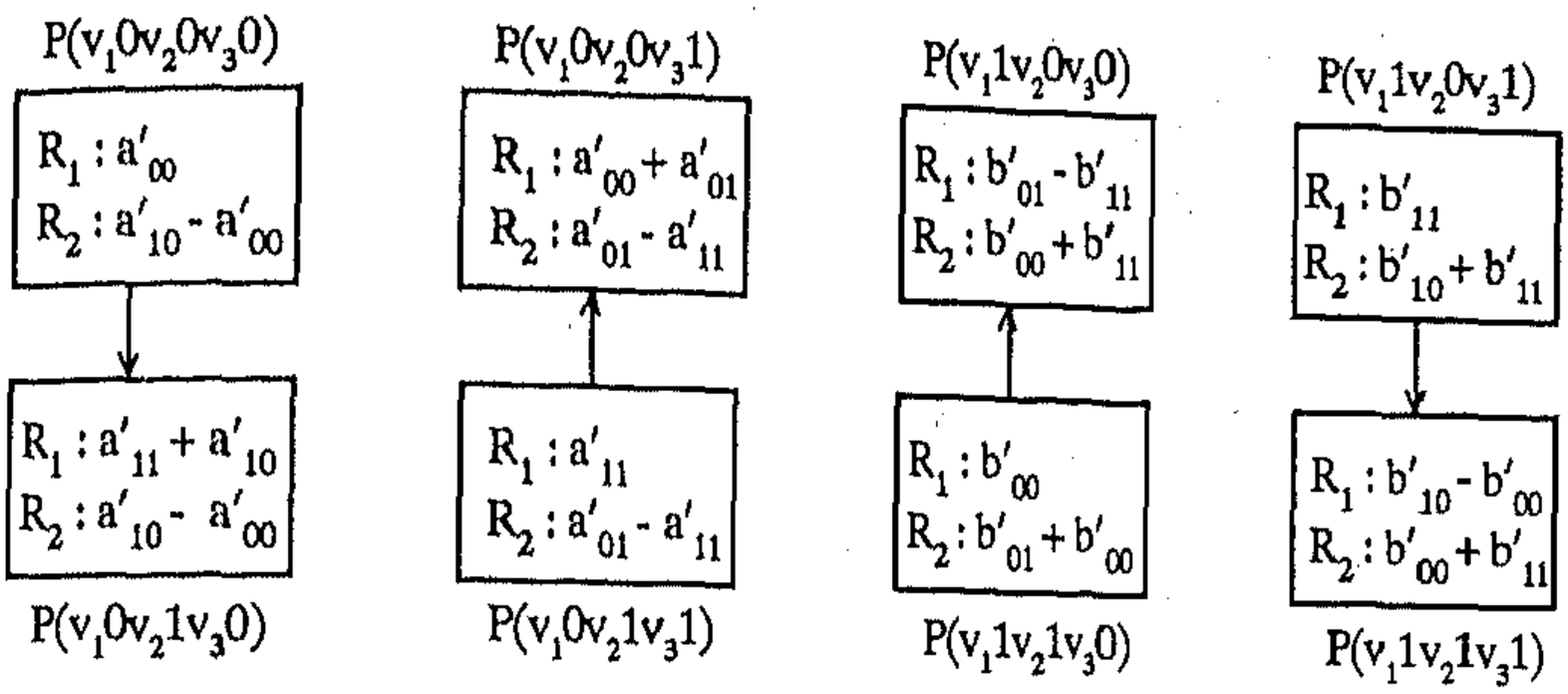


Figure 7.8: Contents of registers R_1 and R_2 after Step 2

Step 3 : Do the following steps in parallel

$R_2(v_1 0\ v_2 0\ v_3 0) \leftarrow R_1(v_1 0\ v_2 1\ v_3 0)$ - $R_2(v_1 0\ v_2 0\ v_3 0)$;

$R_2(v_1 0\ v_2 1\ v_3 1) \leftarrow R_1(v_1 0\ v_2 0\ v_3 1)$ - $R_2(v_1 0\ v_2 1\ v_3 1)$;

/* Fig. 7.9 shows the situation after this step */

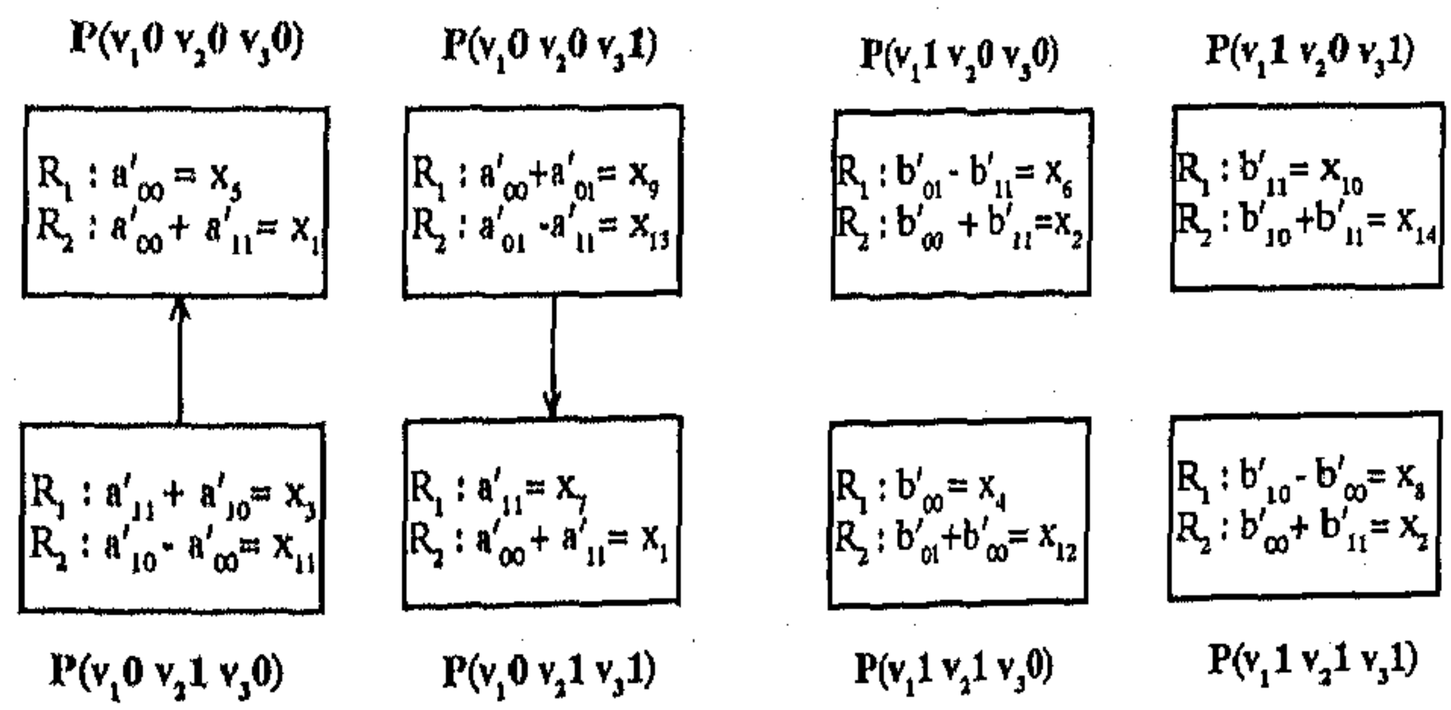


Figure 7.9: Contents of registers R_1 and R_2 after Step 3

Note the correspondence between the X_i values produced after this step and the X_i's stored in different subcubes as shown in Fig. 7.3 after the first phase of the first level of recursion.

Step 4 : Do steps 4.1 and 4.2

Step 4.1 For all v_1, v_2, v_3 do in parallel

$R_1(v_1 0\ v_2 *\ v_3 *) \leftarrow R_1(v_1 1\ v_2 *\ v_3 *) \times R_1(v_1 0\ v_2 *\ v_3 *)$;

/* For a given combination of the values of v_1, v_2 and v_3, all the four multiplications are also done in parallel.

This step actually computes M_3, M_5, M_4 and M_2 which are stored in the R_1 registers of the processors $P(v_1 0\ v_2 0\ v_3 0)$, $P(v_1 0\ v_2 0\ v_3 1)$, $P(v_1 0\ v_2 1\ v_3 1)$ and $P(v_1 0\ v_2 1\ v_3 0)$ respectively. */

Step 4.2 For all v_1, v_2, v_3 do in parallel

$R_1(v_1 1\ v_2 *\ v_3 *) \leftarrow R_2(v_1 0\ v_2 *\ v_3 *) \times R_2(v_1 1\ v_2 *\ v_3 *)$;

/* M_7 and M_6 are stored in the R_1 registers of the processors $P(v_1 1\ v_2 0\ v_2 1)$, $P(v_1 1\ v_2 1\ v_3 0)$ respectively and M_1 is stored in the R_1 registers of both the processors $P(v_1 1\ v_2 1\ v_3 1)$ and $P(v_1 1\ v_2 0\ v_3 0)$ after this step. */

/* Contents of different R_1 registers after this step are shown in Fig. 7.10. */

/* The elements of the 2×2 product matrix c' are computed in steps 5 and 6 */

Step 5 : Do the following steps in parallel

$R_2(v_1 0\ \ v_2 0\ \ v_3 0) \leftarrow R_1(v_1 0\ \ v_2 0\ \ v_3 0) + R_1(v_1 0\ \ v_2 0\ \ v_3 1)$; /* c'_{01} for the q^{th} recursive call */

$R_2(v_1 0\ \ v_2 0\ \ v_3 1) \leftarrow R_1(v_1 0\ v_2 1\ \ v_3 1) - R_1(v_1 0\ \ v_2 0\ v_3 1)$;

$R_2(v_1 0\ \ v_2 1\ \ v_3 1) \leftarrow R_1(v_1 0\ \ v_2 1\ \ v_3 0) + R_1(v_1 0\ \ v_2 1\ \ v_3 1)$; /* c'_{10} for the q^{th} recursive call */

$R_2(v_1 0\ v_2 1\ \ v_3 0) \leftarrow R_1(v_1 0\ \ v_2 0\ \ v_3 0) - R_1(v_1 0\ \ v_2 1\ \ v_3 0)$;

$R_2(v_1 1\ \ v_2 0\ \ v_3 1) \leftarrow R_1(v_1 1\ \ v_2 0\ \ v_3 1) + R_1(v_1 1\ \ v_2 1\ \ v_3 1)$;

$R_2(v_1 1\ \ v_2 1\ \ v_3 0) \leftarrow R_1(v_1 1\ \ v_2 1\ \ v_3 0) + R_1(v_1 1\ \ v_2 0\ \ v_3 0)$;

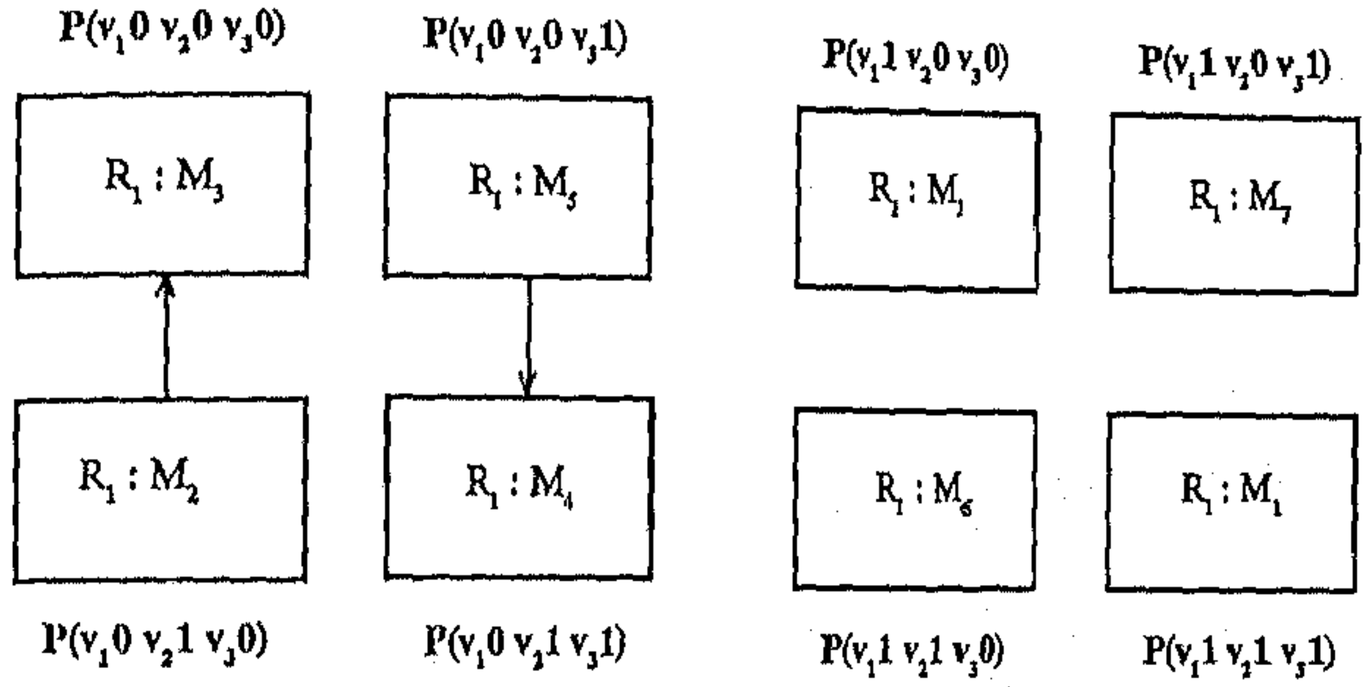


Figure 7.10: Contents of registers R_1 after Step 4

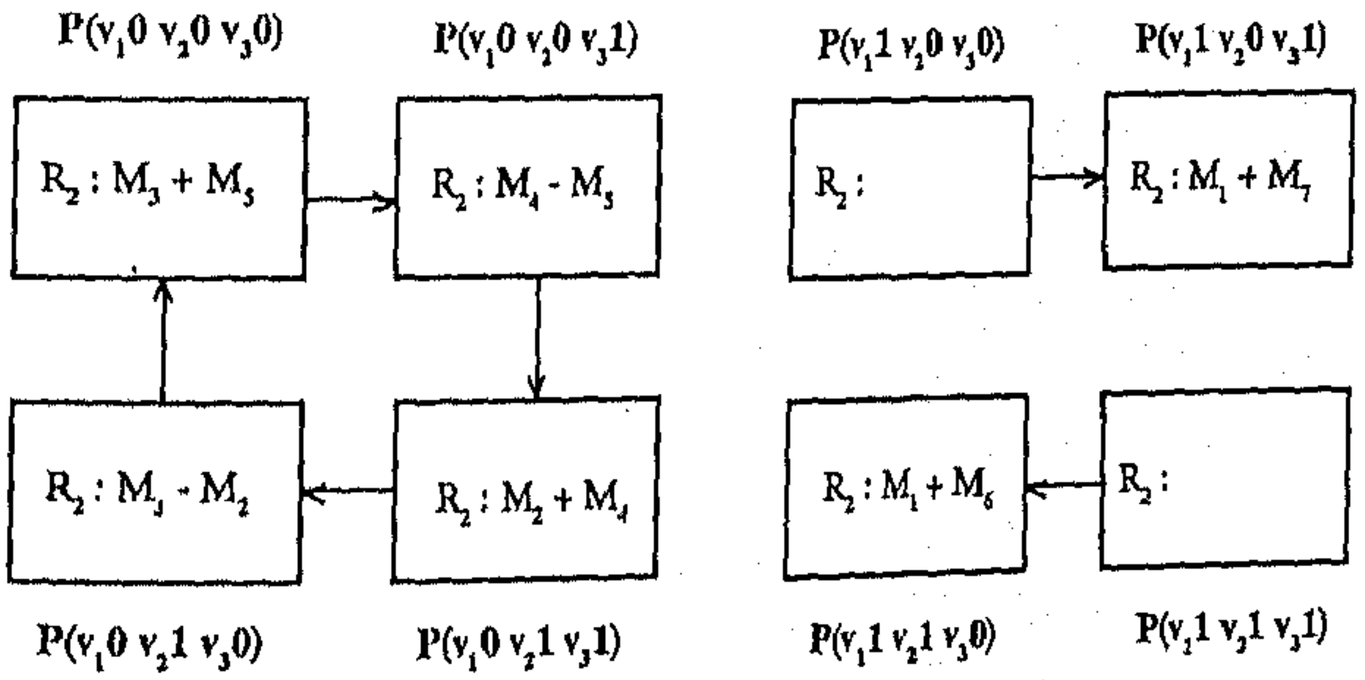


Figure 7.11: Contents of registers R_2 after Step 5

Step 6 : Do the following steps in parallel

$R_1(v_1 0\ \ v_2 0\ \ v_3 0) \leftarrow R_2(v_1 0\ \ v_2 0\ \ v_3 0)$;

$R_1(v_1 0\ \ v_2 0\ \ v_3 1) \leftarrow R_2(v_1 0\ \ v_2 0\ \ v_3 1) + R_2(v_1 1\ \ v_2 0\ \ v_3 1)$;
/* c'_{00} for the q^{th} recursive call */

$R_1(v_1 0\ \ v_2 1\ \ v_3 0) \leftarrow R_2(v_1 0\ \ v_2 1\ \ v_3 0) + R_2(v_1 1\ \ v_2 1\ \ v_3 0)$;
/* c'_{11} for the q^{th} recursive call */

$R_1(v_1 0\ \ v_2 1\ \ v_3 1) \leftarrow R_2(v_1 0\ \ v_2 1\ \ v_3 1)$; □

/* Figs. 7.11 and 7.12 show the contents of the R_1 registers after steps 5 and 6 respectively. */

The product matrix is thus stored in the subcube $(v_1 0\ \ v_2 *\ \ v_3 *)$ in such a way that the R_1 registers of the processors $P(v_1 0\ \ v_2 i\ \ v_3 j)$ $0 \leq i, j \leq 1$, contain the product matrix element $c'_{i,q-j-1}$ corresponding to the q^{th} recursive call.

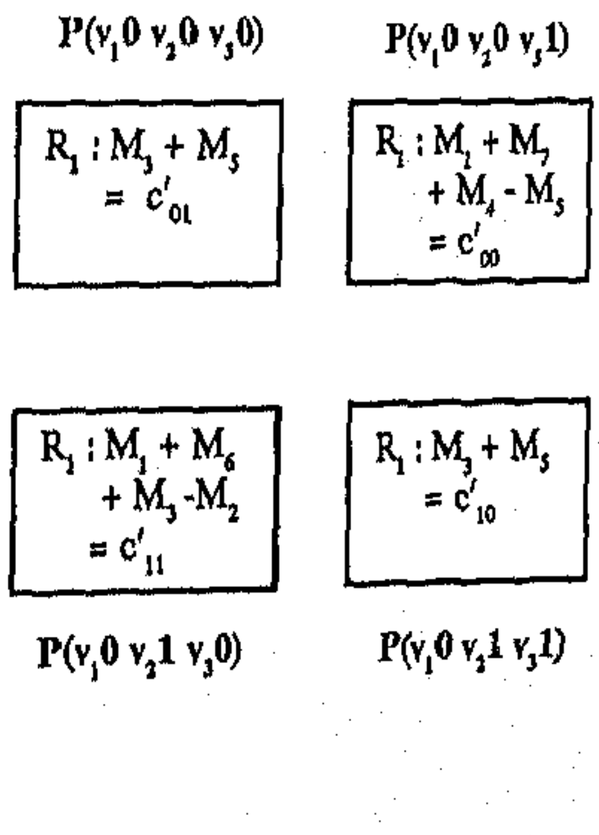


Figure 7.12: Contents of registers R_1 after Step 6

7.3.4 Formal Description of the Algorithm

Algorithm *Par_Strass*

Input : Two matrices $A = [a_{ij}]$ and $B = [b_{ij}]$, each of size $n \times n$, distributed over the processors of a hypercube Q_{3q} such that initially the R_1 register of $P(0,i,j)$ contains the element a_{ij} and the R_1 register of $P(1,i,j)$ contains the element $b_{j,q-i-1}$.

Output : The matrix $C = [c_{ij}] = A \times B$ stored in the hypercube such that the element $c_{i,q-j-1}$ will be obtained from the R_1 register of the processor $P(0,i,j)$.

Main program

{

size $\leftarrow n = 2^q$;

$I \leftarrow 1$; /* I stands for level of recursive call */

$d \leftarrow 0$;

$r \leftarrow 0$;

$c \leftarrow 0$;

Par_mat_mult $(size, I, d, r, c)$;

}

/* *size* = Number of rows/columns of the matrices to be multiplied */

/* (d, r, c) indicates the starting address of the subcube containing A */

Procedure Par_mat_ mult $(size, I, d, r, c)$

/* first phase */

Step 1:

for $i = r$ **to** $(r + 2^{q-I} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-I} - 1)$ **do in parallel**

{ $R_2\ (d,i,j) \leftarrow R_1\ (d, i + \frac{n}{2^I}, j) - R_1(d,i,j)$;

$R_1\ (d,i,j + \frac{n}{2^I}) \leftarrow R_1\ (d,i,j) + R_1\ (d,i,j + \frac{n}{2^I})$;

$R_1\ (d, i + \frac{n}{2^l}, j) \leftarrow R_1\ (d, i + \frac{n}{2^l}, j) + R_1\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l})$;

$R_2\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l}) \leftarrow R_1\ (d, i, j + \frac{n}{2^l}) - R_1\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l})$;

$R_1\ (d+1, i, j) \leftarrow R_1\ (d+1, i, j) - R_1\ (d+1, i, j + \frac{n}{2^l})$;

$R_2\ (d+1, i, j + \frac{n}{2^l}) \leftarrow R_1(d+1, i + \frac{n}{2^l}, j + \frac{n}{2^l}) + R_1(d+1, i, j + \frac{n}{2^l})$;

$R_2\ (d+1, i + \frac{n}{2^l}, j) \leftarrow R_1\ (d+1, i, j) + R_1\ (d+1, i + \frac{n}{2^l}, j)$;

$R_1\ (d+1, i + \frac{n}{2^l}, j + \frac{n}{2^l}) \leftarrow R_1\ (d+1, i + \frac{n}{2^l}, j + \frac{n}{2^l}) - R_1\ (d+1, i + \frac{n}{2^l}, j)$; }

Step 2:

for $i = r$ **to** $(r + 2^{q-l} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-l} - 1)$ **do in parallel**

{ $R_2\ (d, i, j + \frac{n}{2^l}) \leftarrow R_2\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l})$;

$R_2\ (d, i + \frac{n}{2^l}, j) \leftarrow R_2\ (d, i, j)$;

$R_2\ (d+1, i, j) \leftarrow R_2\ (d+1, i + \frac{n}{2^l}, j) - R_1\ (d+1, i, j)$;

$R_2\ (d+1, i + \frac{n}{2^l}, j + \frac{n}{2^l}) \leftarrow R_2\ (d+1, i, j + \frac{n}{2^l}) - R_1\ (d+1, i + \frac{n}{2^l}, j + \frac{n}{2^l})$;

}

Step 3:

for $i = r$ **to** $(r + 2^{q-l} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-l} - 1)$ **do in parallel**

{ $R_2\ (d, i, j) \leftarrow R_1\ (d, i + \frac{n}{2^l}, j) - R_2\ (d, i, j)$;

$R_2\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l}) \leftarrow R_1\ (d, i, j + \frac{n}{2^l}) - R_2\ (d, i + \frac{n}{2^l}, j + \frac{n}{2^l})$;

}

Step 4:

if $size > 2$

/* phase 2 starts here */

{

/* X_i values are broadcast */

$size \leftarrow size/2$

for $i = r$ **to** $(r + 2^{q-(I-1)} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-(I-1)} - 1)$ **do in parallel**

$\{ R_1(d + 2^I, i, j) \leftarrow R_2(d, i, j);$
$R_1(d + 2^I + 1, i, j) \leftarrow R_2(d + 1, i, j);$
$\}$

Step 5 :

/* recursive call */

for $i = 0$ **to** 1 **do in parallel**

for $j = 0$ **to** 1 **do in parallel**

for $k = 0$ **to** 1 **do in parallel**

$Par\_matrix\_mult(size, I + 1, d + i.2^I, r + j.2^{q-I}, c + k.2^{q-I})$

Step 6 :

/* Steps 6 and 7 compute c_{ij}'s for $I = q - 1, q - 2, \cdots, 1$ */

for $i = r$ **to** $(r + 2^{q-I} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-I} - 1)$ **do in parallel**

$\{ R_2(d, i, j) \leftarrow R_1(d, i, j) + R_1(d, i, j + 2^{q-I});$

$R_2\ (d, i, j + 2^{q-I}) \leftarrow R_1\ (d, i + 2^{q-I}, j + 2^{q-I}) - R_1\ (d, i, j + 2^{q-I})$;
$R_2\ (d, i + 2^{q-I}, j + 2^{q-I}) \leftarrow R_1\ (d, i + 2^{q-I}, j) + R_1\ (d, i + 2^{q-I}, j + 2^{q-I})$;
$R_2\ (d, i + 2^{q-I}, j) \leftarrow R_1\ (d, i, j) - R_1\ (d, i + 2^{q-I}, j)$;
$R_2\ (d + 2^I, i, j + 2^{q-I}) \leftarrow R_1\ (d + 2^I, i, j + 2^{q-I}) + R_1\ (d + 2^I, i + 2^{q-I}, j + 2^{q-I})$;
$R_2\ (d + 2^I, i + 2^{q-I}, j) \leftarrow R_1(d + 2^I, i + 2^{q-I}, j) + R_1(d + 2^I, i, j)$;
}

Step 7:

for $i = r$ **to** $(r + 2^{q-I} - 1)$ **do in parallel**

for $j = c$ **to** $(c + 2^{q-I} - 1)$ **do in parallel**

{ $R_1\ (d, i, j + 2^{q-I}) \leftarrow R_2\ (d, i, j + 2^{q-I}) + R_2\ (d + 2^I, i, j + 2^{q-I})$;
$R_1\ (d, i + 2^{q-I}, j) \leftarrow R_2\ (d, i + 2^{q-I}, j) + R_2\ (d + 2^I, i + 2^{q-I}, j)$;
$R_1\ (d, i, j) \leftarrow R_2\ (d, i, j)$;
$R_1\ (d, i + 2^{q-I}, j + 2^{q-I}) \leftarrow R_2\ (d, i + 2^{q-I}, j + 2^{q-I})$;
}

} /* ending of if */

return;

Step 8 :

/* Compute M_i's for 2×2 matrices */

for $i = 0$ **to** 1 **do in parallel**

for $j = 0$ **to** 1 **do in parallel**

$R_1(d, r + i, c + j) \leftarrow R_1(d, r + i, c + j) \times R_1(d + 1, r + i, c + j)$;

for $i = 0$ **to** 1 **do in parallel**

for $j = 0$ **to** 1 **do in parallel**

$R_1(d + 1, r + i, c + j) \leftarrow R_2(d, r + i, c + j) \times R_2(d + 1, r + i, c + j)$;

/* Steps 9 and 10 compute c_{ij}'s for $I = q$ */

Step 9 : do in parallel

{ $R_2(d,r,c) \leftarrow R_1(d,r,c+1) + R_1(d,r,c)$;
$R_2(d,r,c+1) \leftarrow R_1(d,r+1,c+1)$ - $R_1(d,r,c+1)$;
$R_2(d,r+1,c+1) \leftarrow R_1(d,r+1,c+1) + R_1(d,r+1,c)$;
$R_2(d,r+1,c) \leftarrow R_1(d,r,c)$ - $R_1(d,r+1,c)$;
$R_2(d+1,r,c+1) \leftarrow R_1(d+1,r,c+1) + R_1(d+1,r+1,c+1)$;
$R_2(d+1,r+1,c) \leftarrow R_1(d+1,r+1,c) + R_1(d+1,r,c)$;
}

Step 10 : do in parallel

{ $R_1(d,r,c+1) \leftarrow R_2(d,r,c+1) + R_2(d+1,r,c+1)$;
$R_1(d,r+1,c) \leftarrow R_2(d,r+1,c) + R_2(d+1,r+1,c)$;
$R_1(d,r,c) \leftarrow R_2(d,r,c)$;
$R_1(d,r+1,c+1) \leftarrow R_2(d,r+1,c+1)$;
}

return; □

Observation: Steps 8, 9 and 10 of the procedure Par_matrix_mult are executed when $I = n$, i.e., when only 2×2 matrix multiplications are involved. These three steps, in fact, do the same computations as the steps 4, 5 and 6 of section 7.3.2.

Theorem 7.1 *Algorithm Par_Strass computes the product C of the two matrices A and B correctly.*

Proof : It follows from the foregoing discussions in 7.3.1 and 7.3.2 and the above observation that the algorithm Par_matrix_mult works correctly for 2×2 matrix multiplication. That is, the base case of recursion is correct. For $n \times n$ matrix multiplications, $n > 2$, we note that at the I^{th} recursive call $1 \leq I < q$, the distribution

of the X_i submatrices over the subcubes will be exactly similar to that in Fig. 7.3, with each square block in the figure as an $\frac{n}{2^7} \times \frac{n}{2^7}$ submatrix instead of just one single matrix element. The next recursive call in step 5 of Par_matrix_mult should compute the products of the elements of the form $X_{2k+1}X_{2k+2}$, $k = 0, 1, \cdots, 6$. Assuming that these products are computed correctly, the corresponding M_i values should be computed and stored in different subcubes as shown in Fig. 7.10 with the only difference that each block is now an $\frac{n}{2^7} \times \frac{n}{2^7}$ submatrix. Steps 7 and 8 of Par_matrix_mult manipulates these M_i's over different subcubes in the same manner as in Figs. 7.11 and 7.12 with each block now considered as of size $\frac{n}{2^7} \times \frac{n}{2^7}$. Hence, it follows that each of the required submatrices c_{00}, c_{01}, c_{10} and c_{11} of the product matrix corresponding to the I^{th} recursive call is now computed in the original subcubes containing the submatrices X_9, X_5, X_7 and X_3 of Fig. 7.3 respectively.

Hence, it follows that the algorithm Par_Strass computes the product $C = AB$ correctly where elements are stored in the initial position of the A matrix with a lateral inversion (column i of C takes the position of column $n - i$ of A). □

7.3.5 Time Complexity of the Algorithm

For the l^{th} recursive call ($1 \leq l \leq q - 1$), each of the steps 1, 2 and 3 of the algorithm Par_Strass requires one unit of time(including both data communication and addition/subtraction time). Step 4 of the algorithm also requires a single unit of time. Computations of c_{ij}'s in each recursion have been done in steps 6 and 7 of the algorithm which require two more units of time. Therefore, $(q-1)$ recursive calls require $6(q-1) = 6(log\, n - 1)$ parallel steps. q^{th} recursive call requires 7 time units (as indicated in section 7.3.2), assuming that multiplication of two numbers can also be done in one unit of time. Therefore, the overall time complexity of the algorithm comes out to be $(6\, log_2\, n + 2)$ time units.

Theorem 7.2 *The algorithm Par_Strass computes the product of two $n \times n$ matrices in $O(log\, n)$ parallel steps.*

The execution time, including both communication and computation steps, of the above algorithm is of the same order as that of the parallel version of the conventional algorithm on a hypercube with n^3 processors. At the end of the initial data broadcasting (along with the required additions/subtractions of the submatrices) over the n^3 processors, all processors do the scalar multiplications (multiplications of two numbers) in a single parallel step and finally these products are gathered after the required additions to generate the product C. Since all scalar multiplications are done in one parallel step, the reduction in the total number of scalar multiplications in Strassen's algorithm over the conventional algorithm does not really result in any timing advantage due to this parallelization.

7.4 Conclusion

With a view of investigating different aspects of parallel implementation of a class of recursive matrix algorithms on systolic architectures, we have considered here the parallelization of Strassen's algorithm on a hypercube. Other recursive matrix algorithms of similar nature, e.g., matrix inversion algorithm due to Pease [P69], may also be tackled in a similar way. However, a general framework for tackling such recursive algorithms is a topic for future research.

Chapter 8

Conclusion

The problems addressed in this thesis are all related to the static interconnection networks. In the first part of the thesis, we concentrated on the problem of designing network topologies. We have introduced two new topologies and analyzed their properties which may be useful in measuring effectiveness of these topologies. In the second part, we have designed parallel algorithms for three well known problems and analyzed how these algorithms work when implemented on suitable static interconnection networks.

The first structure we have proposed in chapter 3 is an extension of the ring network. The network graph with N nodes is denoted by $G(m, N)$, where m is a parameter of the graph such that $m \geq 3$ and N is an even multiple of m. While designing this structure, we added extra edges over the ring topology to reduce the diameter to $O(\log N)$ at the cost of increased node-degree. The maximum node-degree of the proposed topology is 4 and the average node-degree is 3 when m is odd, and $(3 + \frac{1}{m})$ when m is even. An $O(\log N)$ time algorithm for point-to-point routing on this topology has been discussed. When N is a power of 2, *Ascend* and *Descend* classes of algorithms can be implemented on this topology in $O(\log N)$ time.

In most of the existing network topologies, out of the three design parameters, number of nodes (N), node-degree (δ) and diameter (D), only one can be chosen

independently. There are very few topologies, for example, generalized hypercube, radix-r de Bruijn graphs, etc., in designing of which two of these three parameters can be varied independently. In chapter 4, we have proposed a new family of network topologies which also provides this design flexibility, i.e., any two of the three parameters N, δ and D, can be chosen independently. We call this topology as Generalized Hypercube-Connected-Cycles (GHCC). This topology is presented by a graph $G(l, m)$, where l and m are two integer parameters influencing the diameter and the degree respectively. $G(l, m)$ is a regular graph with node-degree $m + 1$, when $l \geq 3$ and the total number of nodes $N = l.m^l$. The diameter of the topology is $\lfloor \frac{5l}{2} \rfloor - 2$ for $l \neq 1, 2, 3$, $m \neq 1$ and $\lfloor \frac{5l}{2} \rfloor - 1$ for $l \leq 3$, $m \neq 1$. The graph is maximally connected and in the presence of m faulty nodes, the diameter of the network increases by at most 6. Algorithms for point-to-point routing and single node broadcast on this network have been developed. The *Ascend* and *Descend* classes of algorithms can be implemented on this topology in $O(\log N)$ time.

To measure the efficiency of the proposed topologies, comparative studies of these two structures with the existing popular topologies are also included in the respective chapters.

Future work on these structures includes embedding of existing popular topologies on these structures and developing algorithms for communication problems like, scattering, total exchange, multinode broadcast, etc. Implementation of useful existing algorithms on these structures is also an interesting area for further research.

The problems that we have taken up for designing parallel algorithms are (i) testing isomorphism of maximal outerplanar graphs, (ii) numerical interpolation using Lagrange interpolation technique, and finally (iii) the Strassen's matrix multiplication algorithm, as a representative example of a specific class of recursive matrix algorithms.

The graph isomorphism problem for a very special class of graphs (MOPs) called maximal outerplanar graphs, has been solved in chapter 5. The algorithm we have

designed for testing isomorphism of MOPs requires $O(\log N)$ time, where N is the total number of nodes in the graphs to be tested. We have implemented the algorithm on EREW PRAM model with N processors as well as on a hypercube architecture with the same number of processors. We have assumed that the graphs are given in the form of ordered adjacency lists. If the graphs are given in terms of adjacency matrices instead of adjacency lists, $O(N^2)$ processors will be required to keep the time complexity at $O(logN)$.

In chapter 6, we have generated a parallel algorithm for Lagrange interpolation. This algorithm, when implemented on a suitable architecture, requires $O(N/\log N)$ time using $O(N \log N)$ processors for N-point interpolation.

In order to investigate different issues related to parallel implementation of a class of recursive matrix algorithms on systolic architectures, in chapter 7, we have implemented the Strassen's matrix multiplication algorithm on hypercube architecture. The basic philosophy of our technique may be adopted in parallelizing other recursive matrix algorithms of similar nature, although, the specific details of parallel implementation entirely depends on the particular nature of the algorithm in question. However, devising a general framework for recursive matrix algorithms of such nature may be a potential topic for future research.

Bibliography

[A65] S. B. Akers, "On the construction of (d, k) graphs," *IEEE Trans. Electron. Comput.*, pp. 448, June 1965.

[A89] Selim G. Akl, *The Design and Analysis of Parallel Algorithms.* NJ : Prentice-Hall, 1989.

[AD89] L. G. Aleksandrov and H. N. Djidjev, "Solving systems of linear equations on a systolic processor," Technical Report, Center of Informatics And Computer Technology, BAN-Sofia, no. 1-8, 1989.

[AG81] R. Armstrong and F. G. Gray, "Fault diagnosis in a boolean cube of microprocessors," *IEEE Trans. Comput.*, vol. C-30, no. 8, pp. 587 - 590, Aug. 1981.

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, MA : Addison-Wesley, 1974.

[AK83] M. Atallah and R. Kosaraju, "Graph problems on a mesh connected processor array," *J. Assoc. Comp. Mach.*, vol. 31, pp. 649 - 667, 1983.

[AK84] S. B. Akers and B. Krishnamurti, "Group graphs as interconnection networks," in *Proc. Int. Conf. Fault Tolerant Comput.*, 1984, pp. 422 - 427.

[AK89] S. B. Akers and B. Krishnamurti, "A group-theoretic model for symmetric interconnection networks," *IEEE Trans. Comput.*, vol. 38, no. 4, pp. 555 - 566, Apr. 1989.

[AL81] B. W. Arden and H. Lee, "Analysis of chordal ring network," *IEEE Trans. Comput.*, vol. C-30, pp. 291 - 295, Apr. 1981.

[AL90] A. El-Amawy and S. Latifi, "Bridged hypercube networks," *J. Parallel Distributed Comput.*, pp. 90 - 95, Sept. 1990.

[AL91] A. El-Amawy, and S. Latifi, "Properties and performance of folded hypercube," *IEEE Trans. Parallel Distributed Syst.*, vol. 2, no. 1, pp. 31 - 42, 1991.

[AP89] S. Abraham and K. Padmanabhan, "Twisted cube : A study in asymmetry," *J. Parallel Distributed Comput.*, vol. 13, pp. 104 - 110, Nov. 1991.

[AW93] S. Arno and F. S. Wheeler, "Signed digit representation of minimal hamming weight," *IEEE Trans. Comput.*, vol. C-42, no. 8, pp. 1007 - 1010, Aug. 1993.

[B46] N. G. de Bruijn, "A combinatorial problem," *Koninklijke Nederlande Academie Van Waten Schapen Proc.*, no. A49, pp. 758 - 764, 1946.

[B74] N. Biggs, *Algebraic Graph Theory.* London : Cambridge University Press, 1974.

[B78] B. Bollobas, *Extremal Graph Theory.* London Math. Soc. Monograph no. 11, London : Academic Press, 1978.

[B90] H. L. Bodlaender, "Polynomial algorithms for graph isomorphism and chromatic index on partial K-trees," *J. Algorithms,* vol. 11, pp. 631 - 643, 1990.

[BA84] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-33. no. 4, pp. 323 - 333, Apr. 1984.

[BDQ82] J. C. Bermond, C. Delorme and J. J. Quisquater, "Tables of large graphs with given degree and diameter," *Info. Proc. Lett.*, vol. 15, no. 1, pp. 10 - 13, 1982.

[BFM89] J. -C. Bermond O. Favaron and M. Maheo, "Hamiltonian decomposition of cayley graphs of degree four," *J. Combinatorial Theory*, Ser. B 46, pp. 142 - 153, 1989.

[BH94] S. M. Balle and P. C. Hansen, "A Strassen-type matrix inversion algorithm," *Advances in Parallel Algorithms*. Amsterdam : IOS Press, 1994, pp. 22 - 30.

[BI73] E. Bannai and T. Ito, "On finite Moore graphs," *J. Fac. Sci.*, Tokyo Univ., pp. 191 - 208, 1973.

[BIP85] J. -C. Bermond, G. Illiades and C. Peyrat, "An optimization problem in distributed loop computer networks," in *Proc.* 3^{rd} *Int. Conf. Combinatorial Maths.*, 1985.

[BJM79] T. Beyer, W. Jones and S. Mitchel, " Linear algorithms for isomorphism of maximal outerplanar graph," *J. Assoc. Comp. Mach.*, vol. 26, no. 4, pp. 603 - 610, Oct. 1979.

[BOS+91] D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, "Optimal communication algorithm for hypercubes," *J. Parallel Distributed Comput.*, vol. 11, pp. 263 - 275, 1991.

[BR89] A. Benaini and Y. Robert, "An even faster systolic array for matrix multiplication," *Parallel Computer*, vol. 12, pp. 249 - 254, 1989.

[BS78] G. Baudet and D. Stevenson, "Optimal sorting algorithms for parallel computers," *IEEE Trans. Comput.*, vol. C-27, no. 1, pp. 84 - 87, Jan. 1987.

[BT84] F. T. Boesch anf R. Tindell, "Circulants and their connectivities," *J. Graph Theory*, vol. 8, pp. 487 - 499, 1984.

[BT91] J. C. Bermond and D. Tzvieli, "Minimal- diameter double-loop networks : dense optimal families," *Networks* vol. 21, no. 1, pp. 1 - 10, Jan. 1991.

[BW85] F. T. Boesch and J. -F. Wang, "Reliable circulant networks with minimum transmission delay," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 1286-1291, 1985.

[C76] A. K. Chandra, "Maximal parallelism in matrix multiplication, *IBM Technical Report RC6193*, Thomas J. Watson Research centre, Yorktown Heights, N. Y., Sept. 1976.

[C76a] L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM J. Comput.*, vol. 5, pp. 618 - 623, 1976.

[C82] C. J. Colbourn, "Farey series and maximum outerplanar graphs," *SIAM J. Algebraic Discrete Methods*, vol. 3, no. 2, pp. 187 - 189, June 1982.

[CAB94a] C. Chen, D. P. Agrawal and J. R. Bruke, "dBCube : A new class of hierarchical multiprocessor interconnection networks with area efficient layout," *IEEE Trans. Parallel Distributed Syst.*, vol. 4, no. 12, pp. 1332 - 1344, 1994.

[CAB94b] —, "Design and analysis of a class of highly scalable hierarchical networks : PdBCube," *J. Parallel Distributed Comput.*, vol. 22, pp. 555 - 564, 1994.

[CB81a] C. J. Colbourn and K. S. Booth, "Linear time automorphism algorithms for trees, interval graphs and planar graphs," *SIAM J. Comput.*, vol. 10, no. 1, pp. 203 - 225, Feb. 1981.

[CB81b] S. D. Conte and C. de Boor, *Elementary Numerical Analysis*. Tokyo : McGraw-Hill, 1981.

[CC94] S. Y. Cheng and J. H. Chuang, "Varietal hypercube - A new interconnection network topology for large scale multicomputer," in *Proc. Int. Conf. Parallel Distributed Syst.*, 1994, pp. 703 - 708.

[CCG+93] R. Cole, M. Crochemore, Z. Galil, L. Gasieniec, R. Hariharan, S. Muthukrishnan, K. Park and W. Rytter, "Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions," in *Proc. 34th Annu. Symp. Found. Comp. Sc. (FOCS)*, 1993, pp. 248 - 258.

[CG70] D. G. Corneil and C. C. Gotlieb, "An efficient algorithm for graph isomorphism," *J. Assoc. Comput. Mach.*, vol. 17, no. 1, pp. 51 - 64, Jan. 1970.

[CGK90] P. R. Cappelo, E. Gallopoulous and C. K. Koc, "Systolic computation of interpolating polynomials," *Computing*, vol. 45, no. 2, pp. 95 - 117, 1990.

[CLR90] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms.* London : MIT Press, 1990.

[CPH+83] M. Clint, R. H. Perrot, C. M. Holt and A. Stewart, "The influence of hardware and software considerarions on the design of synchronous parallel algorithms," *Software Practice and Experience*, vol. 13, no. 10, pp. 961 - 974, 1983.

[CS87] K. H. Cheng and S. Sahni, "VLSI systems for band matrix multiplication," *Parallel Comput.*, vol. 4, pp. 239 - 258, 1987.

[CW80] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," *SIAM J. Comput.*, vol. 11, pp. 472 - 492, 1980.

[CW87] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 1 - 6.

[D79] P. J. Davis, *Circulant Matrices.* New York : John Wiley, 1979.

[D84] K. W. Doty, "New designs for dense processor interconnection networks," *IEEE Trans. Comput.*, vol. C-33, pp. 447 - 450, May 1984.

[DE91] S. P. Dandamudi and D. L. Eager, "Hierarchical interconnection networks for multicomputer systems," *IEEE Trans. Comput.*, vol. C-39, pp. 786 - 797, 1990.

[DHL+90] D. Du, D. Hsu, Q. Li and J. Xu, "A Combinatorial problem related to distributed loop networks," *Networks*, vol. 20, pp. 173 - 180, 1990.

[DMS94] R. K. Das, K. Mukhopadhyaya and B. P. Sinha, "A new family of bridged and twisted hypercubes," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 1240 - 1247, Oct. 1994.

[DNS81] E. Dekel, D. Nassimi and S. Sahni, "Parallel matrix and graph algorithms," *SIAM J. Comput.*, vol. 10, no. 4, pp. 657 - 675, Nov. 1981.

[DS95] R. K. Das and B. P. Sinha, "Optimal communication algorithms in distributed loop networks," *J. Parallel Distributed Comput.*, vol. 30, pp. 85 - 90, 1995.

[E64] B. Elspas, "Topological constraints on interconnection-limited logics," *Switching Circuit Theory Logical Design*, pp. 133 - 147, Oct. 1964.

[E87] H. Edelsbrunner, *Algorithms in Combinatorial Geometry.* Heidelberg : Springer-Verlag, 1987.

[E89] A. El-Amawy, A systolic architecture for fast dense matrix inversion," *IEEE Trans. Comp.*, vol. 38, pp. 449 - 455, 1989.

[E92] K. Efe, "The crossed cube architecture for parallel computation," *IEEE Trans. Parallel Distributed Syst.*, vol. 3, no. 5, pp. 513 - 524, Sept. 1992.

[ED89] A. El-amawy and K. R. Dharmarajan, "Parallel VLSI algorithm for stable inversion of dense matrices, *IEE Proc., Pt E.* 136, pp. 575 - 580, 1989.

[ENS91] A. Esfahanian, L. M. Ni and B. E. Sagan, "The twisted N-cube with application to microprocessing," *IEEE Trans. Comput.*, vol. 40 no. 1, pp. 88 - 93, Jan. 1991.

[ES83] P. Erdos and J. Spencer, "Evolution of the n-cube," *J. Comput. Math. Appl.*, vol. 5, pp. 307 - 312, Mar. 1983.

[F66] H. Friedman, "A design for (d, k) graphs," *IEEE Trans. Electron. Comput.*, pp. 253 - 254, Apr. 1966.

[F71] —, "On the impossibility of certain Moore graphs," *J. Combinatorial Theory (B)*, pp. 245 - 252, Apr. 1966.

[F77] S. Foldes, "A characterization of hypercubes," *J. Discrete Maths.*, vol. 17, pp. 155 - 159, 1977.

[F92] P. Fraigniaud, "Asymptotically optimal broadcasting and gossiping in faulty hypercube microprocessor," *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1410 - 1419, Nov. 1992.

[FK76] M. J. Flynn and S. R. Kosaraju, "Process and their interactions," *Kibernatics*, vol. 5, pp. 159 - 163, 1976.

[FL72] D. J. Farber and K. C. Larson, "The system architecture of the distributed computer system - the communication systems," in *Proc. Symp. Computer Comm. Networks and Teletraffic*, Apr. 1972, Brooklyn Polytechnic Press, pp. 21 - 27.

[FOH87] G. C. Fox, S. W. Otto and A. J. G. Hey, "Matrix algorithms on a hypercube I : Matrix multiplication," *Parallel Comput.*, vol. 4, pp. 17-31, 1987.

[G91] H. Gazit, " A deterministic parallel algorithm for planar graph isomorphism," in *Proc. 32nd Annu. Symp. Found. Comp. Sc. (FOCS)*, 1991, pp. 723 - 732.

[G94] B. Goertzel, "Lagrange interpolation on a processor tree with ring connections," *J. Parallel Distributed Comput.*, vol. 22, no. 2, pp. 321 - 323, 1994.

[GHV+88] L. Geus, W. Henning, M. Vajteri and J. Volkert, "Matrix inversion algorithms for a pyramidal multiprocessor system," *Computers and Artificial Intelligence*, vol. 7, pp. 65 - 79, 1988.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*. San Francisco : W. H. Freeman & Co., 1979.

[GKP88] R. L. Graham, D. E. Knuth and O. Potashnik, *Concrete Mathematics*. Reading MA : Addison - Wesley, 1988.

[GR90] H. Gazit and J. H. Reif, "A randomized parallel algorithm for planar graph isomorphism," in *Proc. 2nd Annu. ACM Symp. Parallel Algo. Arch.*, July 1990, pp. 210 - 219.

[H56] F. B. Hildebrand, *Introduction to Numerical Analysis*. New York : McGraw-Hill, 1956.

[H69] F. Harary, *Graph Theory*. New York : Addison-Wesley, 1969.

[H76] S. Hart, "A note on the edges of the n-cube," *J. Discrete Maths.*, vol. 14, pp. 157 - 163, 1976.

[HB84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, New York : McGraw-Hill, 1984.

[HC82] K. Hwang and Y. H. Cheng, "Partitioned matrix algorithms for VLSI arithmetic systems," *IEEE Trans. Comput.*, vol. C-31, no. 12, pp. 1215-1224, Dec. 1982.

[HHW88] F. Harary, J. P. Hayes and H. Wu, "A survey of the theory of hypercube graphs," *Comp. Math. Applic.*, vol. 15, no. 4, pp. 277 - 289, 1988.

[HPR75] F. Harary, M. Palmer and R. C. Read, "On the cell-growth problem for arbitrary polygons," *Discrete Maths.*, vol. 11, pp. 371 - 389, 1975.

[HS60] A. J. Hoffman and R. R. Singleton, "On Moore graphs with diameter 2 and 3," *IBM J. Res. Devolop.*, pp. 497 - 504, 1960.

[HT71] J. E. Hopcroft and R. E. Tarjan, "A V^2 algorithm for determining isomorphism of planar graphs," *Info. Proc. Lett.*, vol. 1, pp. 32 - 34, 1971.

[HT72] ——, "Isomorphism of planar graphs," in *Complexity of Computer Computations*. R. E. Miller and J. W. Thatcher, Eds., New York : Plenum Press, pp. 172 - 184, 1974.

[HT73] ——, "Dividing a graph into triconnected components," *SIAM J. Comput.*, vol. 2, pp. 135 - 158, 1973.

[HW74] J. E. Hopcroft and J. K. Wong, "Linear time algorithm for isomorphism of planar graphs," in *Proc. 6th Annual ACM Symp. Theory Comput.*, 1974, pp. 172 - 184.

[HZ83] E. Horowitz and A. Zorat, "Divide-and-conquer for parallel processing," *IEEE Trans. Comput.*, vol. C-32, no. 6, pp. 582 - 585, June 1983.

[J92] J. Ja Ja, *An Introduction to Parallel Algorithm*. Reading, Massachusetts : Addison Wesley, 1992.

[JH89] S. L. Johnson and C. -T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, vol. 38, no. 9, pp. 1249 - 1268, Sept. 1989.

[JK88] J. Ja Ja and S. R. Kosaraju, "Parallel algorithm for planar graph isomorphism and related problems," *IEEE Trans. on Circuits Syst.*, vol. 35, no. 3, pp. 304 - 311, Mar. 1988.

[JK89] H. V. Jagadish and T. Kailath, "A family of new efficient arrays for matrix multiplication," *IEEE Trans. Comput.*, vol. C-38, no. 1, pp. 149 - 155, Jan. 1989.

[JS95] P. K. Jana and B. P. Sinha, "Fast parallel algorithm for polynomial interpolation," *Comput. and Maths. Applic.*, vol. 29, no. 4, pp. 85 - 92, 1995.

[K67] I. Korn, "On (d, K) Graphs," *IEEE Trans. Electron. Comput.*, pp. 90, Feb. 1967.

[K80] H. T. Kung, "The structure of parallel algorithms," *Advances in Computers*, vol. 19, M. C. Yovits, Eds., New York : Academic Press, 1980.

[K88a] S. C. Kak, "A two-layered mesh array for matrix multiplication," *Parallel Comput.*, vol. 6, pp. 383-385, 1988.

[K88b] H. P. Katseff, "Incomplete Hypercubes," *IEEE Trans. Comput.*, vol. C-37, pp. 604 - 608, May 1988.

[K88c] S. Y. Kung, *VLSI array processor*, Englewood Cliffs, NJ: Prentice-Hall, 1988.

[KLY92] J. I. Khan, W. Lin and D. Y. Y. Yun, "A parallel matrix inversion algorithm on torus with adaptive pivoting," in *Proc. Int. Conf. Parallel Processing*, vol. 3, St. Charles, IL, 1992, pp. III-69 - III-72.

[L70] J. Q. Longyear, "Regular d-valent graphs of girth 6 and $2(d^2 - d + 1)$ vertices," *J. Combinatorial Theory*, vol. 9, pp. 420 - 422, 1970.

[L83] F. T. Leighton, *Complexity issues in VLSI*. Cambridge, Mass. : MIT Press, 1983.

[L89] S. Latifi, "Identification of operational subcubes in faulty hypercube," Research report, Univ. Nevada, Las Vegas, 1989.

[L90] B. Louka, "Triangular matrix inversion on systolic arrays," *Parallel Comput.*, vol. 14, no. 2, pp. 223 - 228, 1990.

[L92] F. T. Leighton, *Introduction To Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes.* California : Morgan Kaufmann, 1992.

[LA89] S. Latifi and A. El-Amawy, "On folded hypercubes," in *Proc. Int. Conf. Parallel Processing,* 1989, pp. 180 - 187.

[LB79] G. S. Leuker and K. S. Booth, "A linear time algorithm for deciding interval graph isomorphism," *J. Assoc. Comp. Mach.*, vol. 26, pp. 183 - 195, 1979.

[LB94] S. Latifi and N. Bagherzadeh, "Incomplete Star : An alternative scalable network based on the star graph," *IEEE Trans. Parallel Distributed Syst.*, vol. 5, no. 1, pp. 97 - 102, Jan. 1994.

[LH92] T. -C. Lee and J. P. Hayes, "'A fault-tolerant communication scheme for hypercube computers," *IEEE Trans. Comput.*, vol. 41, no. 10, pp. 1242 - 1256, Oct. 1992.

[LLP+90] C. Levcopoulos, A. Lingas, O. Peterson and W. Rytter, "Optimal parallel algorithms for testing isomorphism of trees and outerplanar graphs," in *Proc. Found. Software Tech. Theoretical Comp. Sc.*, 1990, pp. 204 - 214.

[LO94] M. B. Lin and A. Y. Oruc, "Constant-time inner product/matrix computations on permutation network processors," *IEEE Trans. Comput.*, pp. 1429 - 1434, Dec. 1994.

[LP89] A. Lingas and A. Proskurowski, "On parallel complexity of the subgraph homeomorphism and the subgraph isomorphism problem for

classes of planar graphs," *Theoretical Comp. Sc.*, vol. 68, pp. 155 - 173, 1989.

[LS82] W. E. Leland and M. H. Solomon, "Dense trivalent graphs for processor interconnection," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 219 - 222, Mar. 1982.

[LW85] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, no. 1, pp. 66 - 77, Jan. 1985.

[M82] M. Mulder, "n-cube and median graphs," *J. Graph Theory*, vol. 4, pp. 107 - 110, 1982

[M86] G. P. McKeown, "Iterated interpolation using a systolic array," *ACM Trans. Math. Software*, vol. 12, pp. 162 - 170, 1986.

[MC80] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Massachusetts : Addition-Wesley, 1980.

[MC87] L. Melkemi and. M. T. Chuente, "Complex of matrix product on a class of orthogonally connected systolic arrays," *IEEE Trans. Comput.*, vol. C-36, no. 5, pp. 615 - 619, May 1987.

[MKC92] V. K. Murthy, E. V. Krishnamurthy and P. Chen, "Systolic algorithm for rational interpolation and pade approximation," *Parallel Computing*, pp. 75 - 83, Jan. 1992.

[MM63] J. W. Moon and L. Moser, "Triangular dissection of n-gons," *Canad. Math. Bull.*, vol. 6, pp. 175 - 178, 1963.

[MMS92] E. I. Milovanovic, I. Z. Milovanovic and M. K. Stojcev, "Matrix inversion algorithm for linear array processor," *Lecture Notes in Comp. Sc. 634*, Berlin : Springer-Verlag, 1992, pp. 367 - 372.

[MP70] J. H. Morris (JR.) and V. R. Pratt, "A linear pattern matching algorithm," Tech. Rep. no. 40, Computing Center, Univ. California, Berkley, California, 1970.

[MR82] G. Memmi and Y. Railard, "Some new results about the (d, k) graph problem," *IEEE Trans. Comput.*, vol. C-31, no. 8, pp. 748 - 791, Aug. 1982.

[MR85] G. Miller and J. Reif, "Parallel tree contraction and its application," in *Proc.* 26^{th} *IEEE Symp. on FOCS*, pp. 478 - 489, 1985.

[MS95] K. Mukhopadhyaya and B. P. Sinha, "Fault tolerant routing in distributed loop networks," *IEEE Trans. Computers.*, vol. 44, no. 12, pp. 1452 - 1456, Dec. 1995.

[NMB83] D. Nath, S. N. Maheshwari and P. C. P. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees", *IEEE Trans. Comput.*, vol. C-32, no. 6, pp. 569 - 581, June 1983.

[NS81] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD computers," *IEEE Trans. Comput.*, vol. 30, no. 2, pp. 101 - 106, Feb. 1981.

[P69] M. C. Pease, "Inversion of matrices by partitioning," *J. Assoc. Comp. Mach.*, vol. 16, no. 2, pp. 302 - 314, Apr., 1969.

[P84] V. Y. Pan, "How to multiply matrices faster," *Lecture notes in Computer Science*, vol. 179, New York: Springer-Verlag, 1984.

[PA88] W. A. Porter and J. L. Aravena, "Cylindrical arrays for matrix multiplication," in *Proc.* 24^{th} *Annu. Allerton Conf. Commun. Control Comput.*, Mar. 1988, pp. 595 - 602.

[PC94] J. -H. Park and K. -Y. Chwa, "Recursive Circulant : A new topology for multicomputer networks," in *Proc. Int. Symp. Parallel Arch., Algo. Networks*, 1994, pp. 73 - 80.

[PR82] D. K. Pradhan and S. M. Reddy, "A fault-tolerant communication architecture for distributed systems,"*IEEE Trans. Comput.*, vol. C-31, no. 9, pp. 863 - 870, Sept. 1982.

[PS88] F. P. Preparata and M. I. Shamos, *Computational Geometry - An Introduction.* Berlin : Springer-Verlag, 1988.

[PT89] V. K. Prasanna Kumar and Y. C. Tsai, "Designing linear systolic arrays," *J. Parallel Distributed Comput.*, vol. 7, pp. 441 - 463, 1989.

[PT91] V. K. Prasanna Kumar and Y. C. Tsai, "Synthesizing optimal family of linear systolic arrays for matrix computations," *IEEE Trans. Comput.*, vol. 40, no. 6, pp. 770 - 774, June 1991.

[PV80] F. P. Preparata and J. Vuillemin, "Area-time optimal VLSI networks for multiplying matrices," *Info. Proc. Lett.*, vol. 11, no. 2, pp. 77 - 80, Oct. 1980.

[PV81] F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles : A versatile network for parallel computation," *Comm. Assoc. Comp. Mach.*, pp. 300 - 309, May 1981.

[R60] G. W. Reitweisner, "Binary Arithmetic," *Advances in Computers,* vol. 1, pp. 231 - 308, 1960.

[R85] G. Rote, "A systolic array algorithm for the algebraic path problem (shortest path; matrix inversion)," *Computing,* vol. 34, no. 3, pp. 191 - 219, 1985.

[R93] J. H. Reif, Ed., *Synthesis of Parallel Algorithms.* California : Morgan Kaufmann, 1993.

[RC77] R. Read and D. Corneil, "The graph isomorphism disease," *J. Graph Theory,* I pp. 239 - 363, 1977.

[RS93] Yordon Rouskov and Pradip K. Srimani, "Fault diameter of star graphs," *Info. Proc. Lett.*, vol. 48, pp. 243 - 251, 1993.

[RV84] I. V. Ramakrishnan and P. J. Varman, "Modular matrix multiplications on a linear array," *IEEE Trans. Comput.*, vol. 33, pp. 952-958, 1984.

[S64] H. Sachs, "On regular graphs with given girth," *Theory of Graphs and its Applications.* New York : Academic, pp. 91 - 97, 1964.

[S69] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354-356, 1969.

[S70] R. M. Storwick, "Improved construction technique for (d, k) Graphs," *IEEE Trans. Electron. Comput.*, pp. 1214 - 1216, Dec. 1970.

[S76] L. L. Schumaker, "Fitting surface to scattered data," in *Approximation Theory.* vol. 11, Lorentz, Chui and Schumaker, Eds., pp. 203 - 268, Academic Press, 1976.

[S78] R. Sibson, "Locally equiangular triangulations," *Comput.*, vol. J. 21, pp. 243 - 245, 1978.

[S80] H. S. Stone, Ed., *Introduction to Computer Architecture.* Science Research Associates, Chicago, 1980.

[S85] Q. Stout, "Tree-based graph algorithms for some parallel computers," in *Proc. Int. Conf. Parallel Processing*, 1985, pp. 727 - 733.

[S86] V. Strassen, "The asymptotic spectrum of tensors and the exponent of matrix multiplication," in *Proc. Conf. Found. Comp. Sc. (FOCS)*, 1986, pp. 49 - 54.

[SB93] H. Shen and R. J. Back, "Construction of large- size interconnection networks with high performance," *Networks*, vol. 23, pp. 399 - 414, 1993.

[SMK91] H. Schroeder, V. K. Murthy and E. V. Krishnamurthy, "Systolic algorithm for polynomial interpolation and related problems," *Parallel Computing*, pp. 493 - 503, July 1991.

[SS88] Y. Saad and M. H. Shultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. 37, no. 7, pp. 867 - 872, July 1988.

[T91] D. Tzvieli, "Minimal diameter double-loop networks I. Large infinite families," *Networks*, vol. 21, pp. 387 - 415, Jul. 1991.

[TC95] J. C. Tsay and P. Y. Chang, "Some new designs of 2D array for matrix multiplication and transitive closure," *IEEE Trans. Parallel Distributed Syst.*, vol. 6, no. 4, pp. 351 - 362, Apr. 1995.

[TRS90] S. B. Tien, C. S. Raghavendra and M. A. Sridhar, "Reconfiguring embedded task graphs in faulty hypercubes by automorphism," in *Proc. Hawaii Int. Conf. Syst. Sci.*, Jan. 1990, pp. 91 - 100.

[TS79] S. Touge and K. Steiglitz, "The design of small diameter networks by local search," *IEEE Trans. Comput.*, vol. C-28, pp. 537 - 542, Jul. 1979.

[U84] J. D. Ullman, *Computational Aspects of VLSI*, Rockville, Md. : Computer Science Press, 1984.

[V76] F. L. Van Scoy, "Parallel algorithms in cellular spaces," *Ph. D. thesis*, University of Virginia, Charlotteville, Va., 1976.

[VKV91] N. C. Veeraraghavulu, P. Srenivasa Kumar and C. E. Veni Madhavan, "A linear time algorithm for isomorphism of a subclass of chordal graphs," *Proc. 1st National Seminar Theoretical Comp. Sc.*, 1991, pp. 159 - 168.

[VR86] P. J. Varman and I. V. Ramakrishnan, "Synthesis of an optimal family of matrix multiplication algorithms on linear arrays," *IEEE Trans. Comput.*, vol. 35, pp. 989 - 996, 1986.

[W66] L. Weinberg, "On the maximum order of the automorphism group of a planar triply-connected graph," *SIAM J. Appl. Math.*, vol. 14, pp. 729 - 738, 1966.

[W72] R. S. Wilkov, "Analysis and design on reliable computer networks," *IEEE Trans. Commun.*, C-20, pp. 660 - 678, 1972.

[WC74] C. K. Wong and D. Coppersmith, "A combinatorial problem related to multimodule memory organization," *J. Assoc. Comp. Mach.*, vol. 21, no. 3, pp. 392 - 401, 1974.

[YTR94] P. -J. Yang, S. -B. Tien and C. S. Raghavendra, "Reconfiguration of rings and meshes in faulty hypercubes," *J. Parallel Distributed Comput.*, vol. 22, no. 1, pp. 96 - 106, Jul. 1994.

List of Publications of the Author

[1] S. Sen Gupta, R. K. Das, K. Mukhopadhyaya and B. P. Sinha, "A Family of Network Topologies with Multiple Loops and Logarithmic Diameter," to appear in *Parallel Computing.*

[2] S. Sen Gupta, D. Das and B. P. Sinha, "The Generalized Hypercube - Connected - Cycle : An Efficient Network Topology," to appear in *Proc. International Conference on High Performance Computing (HiPC)*, to be held in December 1996, Trivandrum.

[3] S. Sen Gupta, D. Das and B. P. Sinha, "A Fast Parallel Algorithm for Polynomial Interpolation Using Lagrange's Formula," in *Proc. International Conference on High Performance Computing (HiPC)*, December 1995, New Delhi, pp. 701 - 706.

[4] S. Sen Gupta, K. Mukhopadhyaya, B. B. Bhattacharya and B. P. Sinha, "Geometric classification of triangulations and their enumerations in a convex polygon," *Computers and Mathematics with Applications*, vol. 27, no. 7, pp. 99 - 115, 1994.

[5] S. Sen Gupta, R. K. Das, K. Mukhopadhyaya and B. P. Sinha, "A new family of low-diameter network topologies with multiple loops," in *Proc. First International Workshop on Parallel Processing*, December 1994, Bangalore, pp. 41 - 46.