COLLECTIVE COMMUNICATION PROBLEMS IN MULTIPROCESSORS

by

VASSILIOS V. DIMAKOPOULOS
M.A.Sc, University of Victoria, 1992
Diploma, University of Patras, 1990

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

We accept this dissertation as conforming
to the required standard

Dr. N. J. Dimopoulos, Supervisor, Dept. of Electrical & Computer Engineering

---

Dr. K. F. Li, Member, Dept. of Electrical & Computer Engineering

---

Dr. V. K. Bhargava, Member, Dept. of Electrical & Computer Engineering

---

Dr. W. Myrvold, Outside Member, Dept. of Computer Science

Dr. S. Vassiliadis, External Examiner, Delft University of Technology, The Netherlands

© VASSILIOS V. DIMAKOPOULOS, 1996

University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part by photocopy or other means, without the permission of the author.

Supervisor: Dr. N. J. Dimopoulos

ABSTRACT

Distributed-memory multiprocessors are based on a collection of independent processing nodes integrated through a point-to-point interconnection network. The presence of locally generated or maintained data spawns the need for solving certain information dissemination problems, also known as collective communications. They include: *broadcasting* where one node needs to send one piece of information to all the other nodes; *scattering* where one node needs to send different items of information to different nodes; *gathering*, the dual problem of scattering, where one node collects data from every other node; *multinode broadcasting* where every node needs to broadcast its own data; *total exchange* where every node needs to perform scattering, that is every node has a different message to send to every other node.

In this dissertation we study the above communication problems in packet-switched networks under two capability models: *single-port* and *multiport*. We provide solutions for specific networks as well as results applicable to general settings. In particular, we design optimal scattering algorithms for extended rings and two-dimensional tori, we present optimal total exchange algorithms in linear arrays and rings and we solve optimally the aforementioned problems in fat trees.

For the multiport broadcasting problem we provide a general construction of broadcast trees in multidimensional (cartesian product) networks. Under the single-port model we derive new lower bounds. Known bounds on this problem become special cases of our result.

For the single-port total exchange problem we construct an optimal algorithm for a large number of node symmetric networks, the class of Cayley graphs. Complete graphs, rings, circulants, hypercubes, cube-connected cycles, butterflies, belong to this family and our construction is an optimal solution applicable to all these networks.

A general theory is also developed for total exchange in multidimensional networks. We show that the problem can be decomposed to the simpler problem of performing total exchange in individual dimensions. We provide optimality conditions for any multidimensional network under the single-port model and for homogeneous networks under the multiport model. The analysis is applicable to many popular interconnection networks such as e.g. hypercubes, meshes, tori (including k-ary n-cubes).

Examiners:

Dr. N. J. Dimopoulos, Supervisor, Dept. of Electrical & Computer Engineering

---

Dr. K. F. Li, Member, Dept. of Electrical & Computer Engineering

---

Dr. V. K. Bhargava, Member, Dept. of Electrical & Computer Engineering

---

Dr. W. Myrvold, Outside Member, Dept. of Computer Science

Dr. S. Vassiliadis, External Examiner, Delft University of Technology, The Netherlands

Table of Contents

Abstract ii

Table of Contents iv

List of Figures vii

List of Tables ix

Acknowledgement x

Dedication xi

1 Introduction 1

1.1 Multiprocessors and Interconnection Networks 2

1.2 Communication Modes 5

1.3 About This Work 7

2 Networks and Communication Model 9

2.1 Graph-theoretic Definitions 9

2.1.1 Multidimensional networks 11

2.1.2 Symmetry 12

2.2 Node-disjoint Paths in Multidimensional Networks 13

2.3 Some Networks of Interest 15

2.4 Communication Model 18

3 Single-source Communications 20

3.1 Broadcasting Under the Multiport Model 20

3.1.1 Broadcast trees in multidimensional networks 22

3.2 Broadcasting Under the Single-port Model 25

3.2.1 General networks 26

3.2.2 Trees 29

3.3 Scattering Under the Single-port Model 29
3.4 Scattering Under the Multiport Model 30
3.4.1 Scattering in extended rings 32
3.4.2 Scattering in the 2D torus 32

4 Total Exchange **37**
4.1 Introduction 37
4.1.1 Total exchange in certain networks 39
4.2 Multiport Total Exchange in Linear Arrays 40
4.3 Multiport Total Exchange in Rings 45
4.3.1 Odd number of nodes 47
4.3.2 Even number of nodes 50
4.4 Single-port Total Exchange 52
4.4.1 Cayley graphs 54
4.4.2 An automorphism property of Cayley graphs 54
4.4.3 Optimal total exchange algorithms 56
4.4.4 A simple node-invariant algorithm 60
4.4.5 An example in hypercubes 62

5 Total Exchange in Multidimensional Networks **63**
5.1 Introduction 63
5.2 Status (Total Distance) in Multidimensional Networks 64
5.3 Total Exchange Under the Single-port Model 67
5.3.1 Optimality conditions 71
5.4 Extension to the Multiport Model 73
5.4.1 Optimality conditions 78

6 Fat Trees **80**
6.1 Preliminaries 81
6.2 Communications Under the Single-port Model 82
6.2.1 Broadcasting 82
6.2.2 Scattering 84
6.2.3 Multinode broadcasting 86
6.2.4 Total exchange 89
6.2.5 Discussion 91
6.3 Communications Under the Multiport Model 91

6.3.1 Single-source communications 92
6.3.2 Multinode broadcasting 92
6.3.2.1 Queueing considerations 93
6.3.2.2 Eliminating the queues 95
6.3.3 Total exchange 96
6.3.3.1 A tighter bound for exponential capacities trees 98
6.3.4 Discussion 100

7 Conclusion **102**
7.1 Open Problems and Future Directions 104

Bibliography **106**

List of Figures

Figure 1.1 Multiprocessor interconnections 2
Figure 1.2 Popular interconnection networks 4

Figure 2.1 Trees 10
Figure 2.2 Cartesian product of two graphs 11
Figure 2.3 Paths in multidimensional graphs 14
Figure 2.4 Linear arrays and rings 16
Figure 2.5 Extended ring E_{14}^2 16
Figure 2.6 Meshes and tori 17
Figure 2.7 Hypercubes 18

Figure 3.1 Broadcasting in Q_3 21
Figure 3.2 Broadcasting in E_{14}^2 22
Figure 3.3 Broadcasting in the $3 \times 4 \times 2$ mesh 24
Figure 3.4 32
Figure 3.5 Mapping a torus on the plane 33
Figure 3.6 Four cases of area division in an $n \times m$ grid 35
Figure 3.7 Scattering in a 7×7 torus 36

Figure 4.1 Initial configuration of rightward messages 40
Figure 4.2 A 6-node linear array example 41
Figure 4.3 Clockwise messages in an odd ring 47
Figure 4.4 Examples in a 7-node ring (a) and a 6-node ring (b) 50
Figure 4.5 An optimal single-port total exchange algorithm for Cayley networks 61
Figure 4.6 An optimal single-port total exchange algorithm for hypercubes . . . 62

Figure 5.1 Two views of a 4×3 torus 68
Figure 5.2 Algorithm A1 69
Figure 5.3 Algorithm A2 for the single-port model 71
Figure 5.4 A 3×3 homogeneous mesh 74

Figure 5.5 Algorithm A3 for multiport homogeneous networks 77

Figure 6.1 A complete binary tree with 15 nodes (8 leaves) 81
Figure 6.2 Portion of a k-ary fat tree 82
Figure 6.3 Broadcasting under the single-port model 83
Figure 6.4 Optimal broadcasting algorithm under the single-port model 84
Figure 6.5 85
Figure 6.6 Optimal multinode broadcasting algorithm under the single-port model 87
Figure 6.7 Multinode broadcasting in 4 leaves under the single-port model . . . 88
Figure 6.8 Optimal total exchange algorithm under the single-port model . . . 90
Figure 6.9 Multinode broadcasting in a 4-leaf binary tree 94
Figure 6.10 Multinode broadcasting algorithm that eliminates queues 95
Figure 6.11 A total exchange algorithm with no contention 97
Figure 6.12 Collapsing the last i levels of the tree gives graph G_i 98

List of Tables

Table 5.1 Messages to be transferred from $s = (v_i, u_j)$ 70
Table 5.2 Messages to be transferred from node (1,1) 74

Table 6.1 Time requirements for communications under the single-port model . 91
Table 6.2 Time requirements for communications under the multiport model . . 101

Acknowledgement

I would like to thank all the people that helped me directly or indirectly during these last five years in Victoria. I am grateful most of all to my supervisor, Dr. N. J. Dimopoulos, for all his academic (and not only) guidance and support. My graduate studies could have been quite painful if it was not for him and his attitude towards his students.

I would like to thank the members in my program committee, Dr. K. F. Li, Dr. V. K. Bhargava, and Dr. W. Myrvold for all the useful comments and discussions during this research and during the courses they offered. I am especially indebted to Dr. W. Myrvold for her approach to graph theory, a subject I used to dislike.

All the people in the Laboratory for Parallel and Intelligent Systems in UVic have been great colleagues and friends. A special thanks goes to Dr. Sivakumar Radhakrishnan and Mr. Mahmood Chowdhury, my co-researchers in the parallel processing group.

In the non-academic side, Dimitra has been my moral supporter for all these years. She went through good and bad times but was always there for me till the last day. I hope all these years of waiting were worth it; and I hope I can repay you.

Finally, I would like to acknowledge all the financial support I received from my supervisor, the University of Victoria, and the B.C. Advanced Systems Institute, through research and teaching assistantships, fellowships and awards.

Dedication

This thesis is dedicated to my family.

To my mother, Agathi, who went through difficult times to see me here, today.
You are responsible for everything good in me.
To my father, Dimitri, who has done more than he realizes for us.
I'm proud to be your son.
And to Nick and Thanasi. My brothers and my best friends .. for ever.

Αφιέρωση

Η παρούσα διατριβή αφιερώνεται στην οικογένειά μου.

Στη μητέρα μου, Αγαθή, που πέρασε δύσκολες στιγμές για να με δει εδώ που είμαι σήμερα.
Είσαι υπεύθυνη για ότι καλό υπάρχει μέσα μου.
Στον πατέρα μου, Δημήτρη, που έχει κάνει για εμάς πιο πολλά απότι φαντάζεται.
Είμαι περήφανος που είμαι γυιός σου.
Και στον Νικολή και τον Θανάση. Τα αδέρφια μου και οι καλύτεροί μου φίλοι ..
για πάντα.

Chapter 1

Introduction

There exists a wealth of practical scientific and engineering problems whose solution will have a significant impact in advancing human civilization. These problems, which have been classified as "grand challenges" [14], include weather forecasting, genetic engineering, petroleum exploration, fluid dynamics, aerodynamic simulations, to name just a few. Their solution requires the use of extremely powerful (fast) computational systems.

The need for computational speed was always one of the driving forces that led to technological improvements; one may anticipate that this will always be the case. However, technology has physical limits, e.g. the propagation speed of signals in physical media. There is probably a long way to go before approaching them but, nevertheless, they are there. Apart from future predictions, it is a fact that today's technology alone is not sufficient to satisfy the ever-increasing demand for speed. Technological advances have been necessarily combined with architectural improvements in the design of computing systems.

Advanced computer architectures are centered around the concept of *parallelism*. Parallelism means simply doing more than one thing at a time and clearly there is no limit (in concept) to the number of concurrent actions. That is, if we have n functional components, we can perform n different operations at a time, making thus our system n times faster. Such a performance is, of course, ideal. Although the major portion of an algorithm may be parallelizeable for execution on the n components, there may exist segments of the code for which this is impossible. In such cases, a simple relation known as *Ahmdal's Law* [46] shows that the departure from the ideal performance can be quite significant. Another limiting factor is the time spent on communication between cooperative processes executing on different functional components.

Parallelism can be achieved within any of the SISD, SIMD and MIMD architectural categories [34], in the form of pipelined processors, array processors and multiprocessors respectively. The most general form of parallelism is associated with multiprocessors and this is the class we concern ourselves with here.

1.1 Multiprocessors and Interconnection Networks

Multiprocessors consist of many processing elements (PE's) which operate under the supervision of a single operating system. Depending on the way the PE's communicate with each other, multiprocessors can be further classified as *tightly coupled* if communication is achieved through shared memory modules or *loosely coupled* if communication occurs through a message-passing processor interconnection subsystem. Because the second category is the focus of this work, we consider the term 'multiprocessor' synonymous with 'loosely coupled multiprocessor'.

In a (loosely coupled) multiprocessor, each PE is actually a complete computer module with local memory and possibly an I/O subsystem. The whole system relies heavily on the interconnection structure between the PE's.

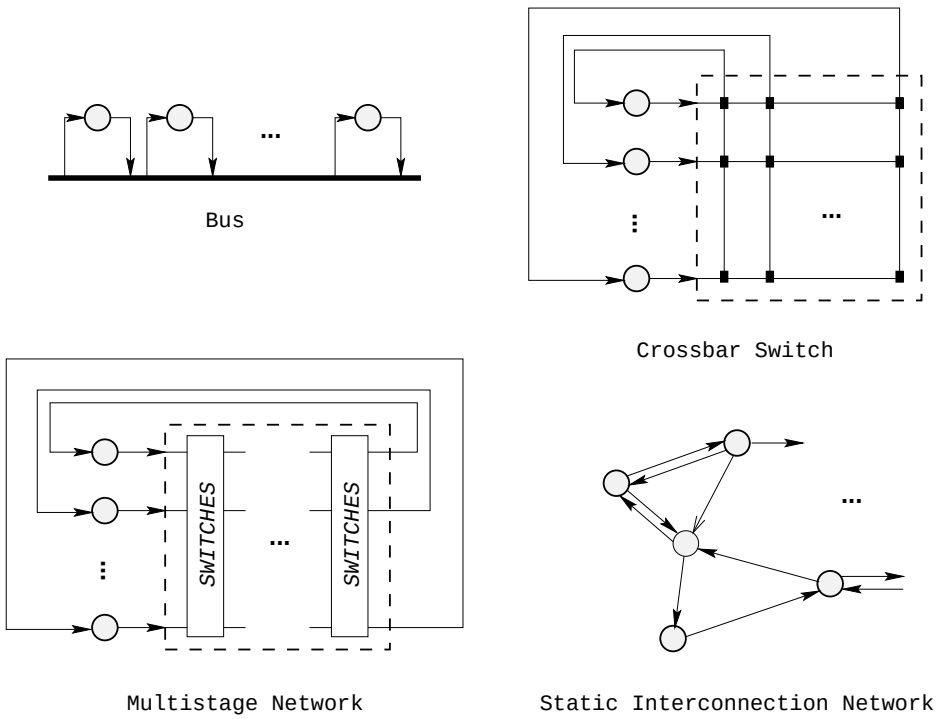

Figure 1.1. *Multiprocessor interconnections*

There have been many interconnection schemes proposed and implemented, including buses, crossbar switches, multistage networks and static networks (Fig. 1.1). Bus structures are attractive because of their low cost. Their performance, though, deteriorates as the number of PE's increase — improvement can be had with the use of multiple buses. Crossbar switches yield the best possible performance because they provide a complete in-

terconnectivity between the PE's. Their cost is prohibitive, though, for more than a few PE's. Multistage networks involve one or more layers of switching elements that can be programmed to provide different paths from their inputs to their outputs. It has been argued that multistage networks do not exploit computational locality. Such networks have been used mainly to interconnect processors and memory modules in tightly coupled multiprocessors and will not concern us here.

Static interconnection networks (or simply *interconnection networks*) have been adopted as a cost-effective communication scheme between the PE's. They consist of dedicated point-to-point links between pairs of PE's and can be circuit or packet switched much like large scale and local area networks. The network is modeled as a graph where the vertices correspond to PE's and the edges correspond to links between PE's. In what follows we use the term *node* to represent a PE or its corresponding vertex in the underlying graph. Popular networks include linear arrays, rings, meshes, tori, hypercubes, etc. (Fig. 1.2). Notice that usually the links between nodes are *bidirectional* so that the graph is undirected; each edge corresponds to two unidirectional links.

For communication to occur between processors, paths should be established over which messages will be exchanged. This is the responsibility of the *router*; given a source node and a destination node, routing policies determine the intermediate nodes and links to be traversed in order for messages to travel from the source to the destination. Routing policies need to be simple and fast. The requirement for simplicity and for inexpensive communication hardware is also responsible for the regular and symmetric nature of most popular interconnection networks.

Apart from its topology and its routing policies, an interconnection network is also characterized by its flow control algorithm and its switching mode [52]. Flow control is responsible for decisions when a resource collision occurs. For example, consider a packet traveling to a destination and assume that an intermediate node that lies on the path finds the next link busy. The flow control algorithm may decide to drop the packet, stop it in place, remove it and buffer it, reroute it, and so on. Switching is the mechanism that forwards packets from an input channel to an output channel.

In *packet switching*, a packet is received in its entirety by an intermediate node and it is buffered immediately. It is placed on the output channel as soon as this channel is free and the next node has space in its packet buffers. Because of the overhead associated with buffering, *virtual cut-through* was proposed whereby the header of the packet is examined before the whole packet is received, and the packet starts immediately flowing to the appropriate output channel, as long as this channel is free. Buffering occurs only if the

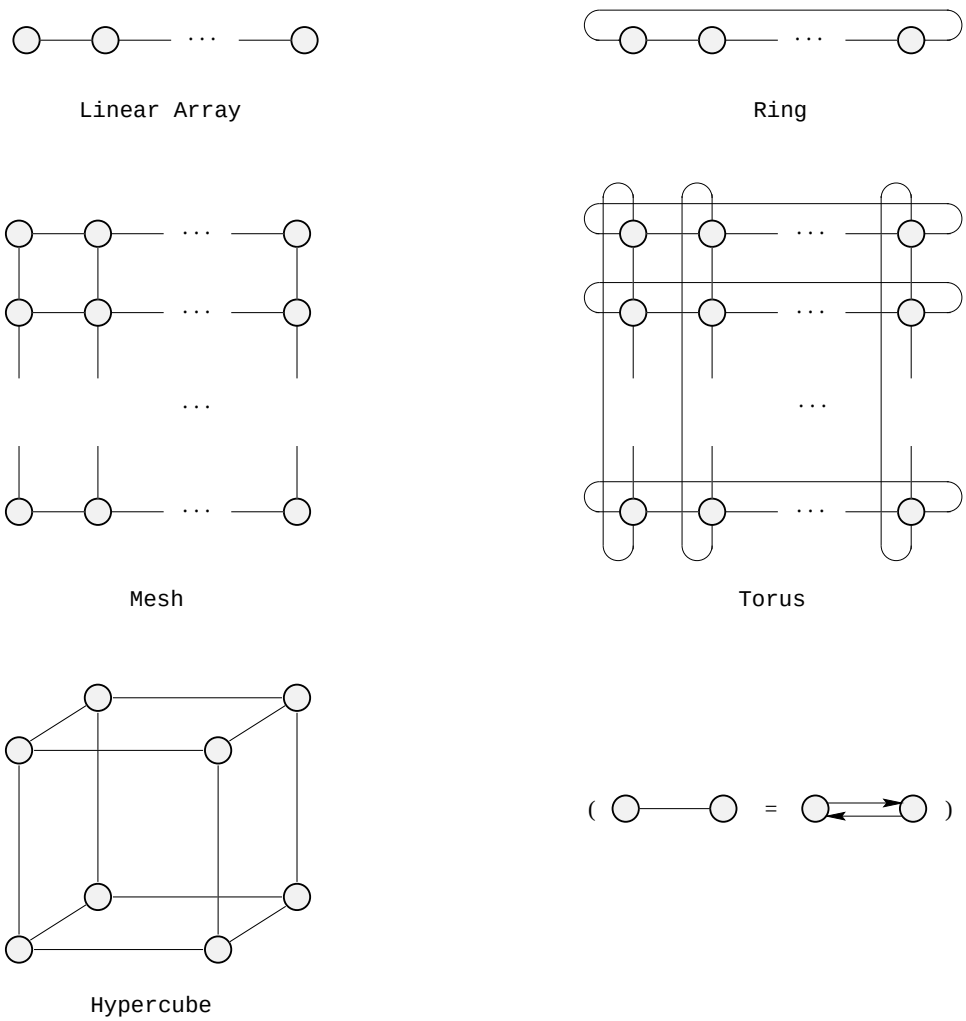

Figure 1.2. *Popular interconnection networks*

output link is busy. In *circuit switching* on the other hand, a physical circuit is constructed between the source and the destination during the 'circuit establishment' phase. In the 'packet transmission' phase the packet is transmitted along the circuit to its destination; the circuit is dedicated to the two nodes for the whole period. After transmission is complete the circuit is released. *Wormhole routing* stands between virtual cut-through and circuit switching. A packet is divided into very small parts called *flits*. These are forwarded in a cut-through manner one after the other, in a pipelined fashion, resembling a moving worm. The head flit determines the direction of the 'worm'. Whenever the head flit is blocked at some node, all other flits are also blocked at their places. A physical link can be multiplexed to accommodate many *virtual channels*. In effect, although some worm may be blocked at some links, another worm may continue moving over the same links.

1.2 Communication Modes

Apart from the necessity of communication between a pair of nodes, there is a need for other forms of communication which we call *communication modes* and which have been identified as follows.

1. *(Single node) broadcasting*, where one specific node has to send the same data to all the other nodes in the network.
2. *Multinode broadcasting*, which involves simultaneous broadcasting from every node.
3. *Gathering*, where a specific node has to receive separate data from each of the other nodes, and *scattering*, the dual problem of gathering, where a certain node needs to send different data to each of the other nodes.
4. *Total exchange*, a multiple scattering/gathering operation, where every node has distinct data to send to every other node.

Notice the difference between broadcasting and scattering: in the second case *different* data are sent to each node.

It is worth noting that in the literature the terminology is not standard yet; we have followed Bertsekas *et al* [5, 4]. Multinode broadcasting is also known as *gossiping* or *all-to-all communication* [65], while Saad and Schultz [59, 60] named it 'total exchange'. Scattering is also known as *one-to-all personalized communication* [36, 6] and total exchange is termed *all-to-all personalized communication* in [36] and *multi-scattering* in [59, 60]. The above communication modes are also known as *collective communications*. It is also important to note that the above list does not exhaust all communication possibilities. Other operations

have also been studied, e.g. *multicasting* [43], a generalized form of broadcasting where the receiving nodes may form a proper subset of the nodes in the network.

The importance of efficient communication algorithms has been realized in the context of linear algebra computations [27, 35]. The aforementioned communication modes arise quite naturally in other contexts as well. For example, broadcasting is essential in many techniques for achieving synchronization among nodes of an asynchronous network [68], for detecting the termination of distributed algorithms [66] and for other system maintenance purposes. We illustrate the need for these communication patterns through an example numerical application on an arbitrary network of n nodes.

Example:

A large number of numerical algorithms for linear and nonlinear problems are centered around an iteration of the form

$$x(t+1) := f\big(x(t)\big) \tag{1.1}$$

where f is a function from $\Re^n$ to $\Re^n$ and $x(t)$, $t = 0, 1, \ldots$, is a sequence of n-dimensional vectors as generated by (1.1). Iteration (1.1) is sometimes referred to as *relaxation* iteration. Jacobi-style, Gauss-Seidel and successive over-relaxation (SOR) methods for solving systems of linear equations [62] can be written in the form of (1.1); in this case f is linear and the iteration can be written as

$$x(t+1) := Ax(t) \tag{1.2}$$

(plus a constant possibly), where A is an $n \times n$ matrix. Statement (1.2) is essentially a matrix-vector multiplication. To execute it we organize the multiprocessor as follows: matrix A is stored in *row-major* manner, i.e. its rows are distributed among the n nodes. It may be the case that $x(t+1)$ is needed in its whole at every node, possibly for subsequent computations of the algorithm. Then, node i, $i = 1, 2, \ldots, n$, computes the inner product of the ith row of A with $x(t)$ and forms the ith component, $x_i(t+1)$, of $x(t+1)$. In order for every node to form $x(t+1)$, node i needs to send $x_i(t+1)$ to every node (for all i). This is immediately recognized as a multinode broadcast problem.

On the other hand, it may be that node i needs only store $x_i(t+1)$ instead of the whole vector $x(t+1)$, as is the case with many distributed asynchronous algorithms [5]. Distributing the components of the initial estimate, $x(0)$, from some node to the other nodes (node i receives $x_i(0)$) requires a scattering operation. After enough applications of (1.1) or (1.2) satisfactory convergence is possibly reached (according to some criterion) so that the algorithm terminates. The result is a vector $x(t)$, for some t, and has its components

distributed among the nodes. To collect the result at a certain node, we need every node i send $x_i(t)$ to the designated node. This is accomplished through a gathering operation.

Sometimes, depending on the sparsity structure of matrix A, it may be advantageous [5] to store A in *column-major* fashion, whereby node i holds the ith column of A. Changing the storage scheme from row-major to column-major (or vice versa) requires the distribution of the ith row to the other nodes, node j receiving the jth entry of row i, A_{ij}. This is to occur for all n rows, leading to an instance of the total exchange problem. Notice that in the above operation we essentially transpose A and this is the reason that total exchange was considered synonymous to matrix transposition in [59, 60]. □

It can be seen [5] that scattering and gathering have increased communication requirements with respect to broadcasting; multinode broadcasting is at least as time consuming as scattering/gathering; and total exchange is the most time demanding of all. It is also worth noting that given an algorithm for the scattering problem, an algorithm for the gathering problem (and vice versa) is produced simply by reversing the data paths. This is to say that the two problems are considered equivalent in terms of time requirements so that we can concentrate only on scattering, without loss of generality.

1.3 About This Work

In this work we focus on the analysis of the communication modes discussed in the previous paragraphs and on the design of algorithms to implement them in specific networks. Some of the contributions of this thesis include: general bounds on the broadcasting problem for arbitrary networks; optimal total exchange algorithms for linear arrays and rings; a theory for broadcasting and total exchange in multidimensional networks; a theory for total exchange in Cayley networks; a complete set of communication algorithms for fat tree networks. The topics are organized as follows:

Chapter 2. Related graph-theoretic terminology and some interconnection networks of interest are introduced formally. Multidimensional networks form an important class of topologies for parallel machines. We review some of their properties and derive new ones related to our study. In addition, we state the major assumptions pertaining to the communication model we are going to follow.

Chapter 3. Here we consider single-source communications (broadcasting and scattering/gathering). First we review general strategies for solving the broadcasting problem and

provide constructions for certain graphs, including multidimensional ones. We then proceed with a derivation of general lower bounds for the problem in arbitrary networks. Certain bounds known from the open literature become special cases of our formulas. The problem of scattering is examined next and we show analytically that the time requirements are independent from the topology of the network under the single-port assumption (introduced in Chapter 2). Optimal scattering schemes are given for extended rings and tori under the multiport model.

Chapter 4. Total exchange is the densest of all the communication problems stated previously. The known general bounds for this problem are given in this chapter. We then proceed to develop optimal total exchange algorithms for two important networks: the linear array and the ring. The algorithm for linear arrays is the only optimal algorithm known in the literature. For rings, optimal algorithms were known for the cases where the number of nodes is odd; our algorithms are optimal for any number of nodes. Under the single-port model, we develop an optimal solution for any Cayley network. Rings, hypercubes, (wrapped) butterflies and cube-connected cycles are only a few of the important networks in the Cayley class that can take advantage of the developed theory. Optimal algorithms were previously known for only two specific Cayley networks.

Chapter 5. Efficient solutions to most communication problems are necessarily topology-specific. Nevertheless, we show here that it is possible to develop a general total exchange theory for important classes of networks such as multidimensional ones (defined in the next chapter). It is seen that lower bounds and algorithm construction can be had based on bounds and algorithms for each dimension separately. The theory is novel and in effect provides solutions for a whole class of graphs.

Chapter 6. Fat trees are networks based on complete trees and seem quite promising candidates for massively parallel computing; they have already been utilized in certain commercial machines. Fat trees differ in many ways from all the other networks we consider in this thesis. We study in detail the communication problems in the context of such networks and we provide algorithms for all problems and for different network configurations.

Chapter 7. This is the final chapter, summarizing the work and concluding this thesis. We review the major contributions and we identify issues that may form subjects for further research.

Chapter 2

Networks and Communication Model

2.1 Graph-theoretic Definitions

We are going to utilize some standard graph terminology which can be found in any text on the subject (e.g. [28, 12]). A *graph* consists of a set of vertices, V, interconnected by a set of edges, E, symbolized as $G = (V, E)$. If the edges have no direction the graph is *undirected* otherwise, it is *directed*. Unless otherwise stated, graphs will be assumed undirected.

The edge e that connects vertices v and u is written as $e = (v, u)$ and is said to be *incident* with v and u. If $(v, u) \in E$ then v and u are *adjacent* to each other. Vertices adjacent to v will also be called *neighbors* of v. A vertex v has *degree* d_v if it is incident with exactly d_v edges. In a *regular* graph G, all vertices have the same degree, equal to d_G.

A *path* from v_1 to v_k is a sequence of distinct vertices $P = v_1, \ldots, v_k$ such that for every i, $1 \leq i < k$, the edge (v_i, v_{i+1}) is in E. Alternatively, a path could be defined as an alternating sequence of vertices and edges such that the vertices are distinct and every edge is incident with the vertex preceding and the vertex following it in the sequence. A *cycle* is a sequence of vertices $C = v_1, \ldots, v_k$ such that all vertices are distinct except $v_1 = v_k$, and $(v_i, v_{i+1}) \in E$ for all i, $1 \leq i < k$. Alternatively, a cycle can be viewed as a path plus an additional edge joining the first and the last vertices of the path. The *length* of a path is equal to the number of edges it contains, which is equal to the number of its vertices minus one. We should note that we will always consider graphs in which there exists a path between every pair of vertices in V; that is we consider only *connected* graphs.

The *distance* between v and u, $dist(v, u)$, is the minimum length of a path between v and u. Consider the maximum distance from vertex v: let u be a vertex such that $dist(v, u) = \max_{w \in V} dist(v, w)$. Vertex u is an *eccentric* vertex for v and the *eccentricity* of

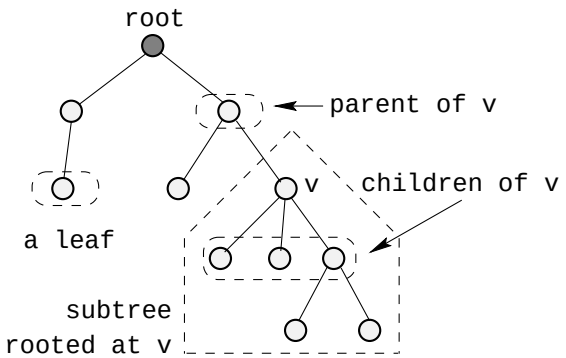

Figure 2.1. *Trees*

v is $e(v) = dist(v, u)$. The maximum eccentricity among all vertices is the *diameter* of the graph.

A *subgraph* of $G = (V, E)$ is a graph $H = (U, F)$ such that $U \subseteq V$ and $F \subseteq E$. A *spanning* subgraph of G has $U = V$. A *tree* is a connected graph that has no cycles. Finally, a *spanning tree* of G is a spanning subgraph of G that is a tree.

Let G be a tree. Any vertex with degree equal to one will be called a *leaf* vertex and it is known that there exist at least two such vertices in any tree [12]. In most cases we will talk about a certain vertex in the tree, called the *root*. The tree will be drawn in a top-down manner with the root being the top vertex as in Fig. 2.1. The *height* of the tree is equal to the eccentricity of the root vertex. Vertices with the same distance from the root are said to belong to the same *level* of the tree. In trees there exists a unique path between any pair of vertices. Consider this unique path from the root to a vertex v in the tree. The vertex preceding v in the path is the *parent* of v. All the other neighbors (if any) of v are its *children* and are usually depicted below vertex v. Finally, if v is a vertex other than the root, the tree induced by vertices whose unique path from the root passes through v will be called the *subtree* rooted at vertex v. One fact that will be useful in later sections is that any tree with n vertices has $n - 1$ edges. Also, any connected graph with n vertices and $n - 1$ edges is a tree.

In this thesis we use the terms 'graph' and 'network' interchangeably. We are going to use the term 'node' to denote either a processing element in the multiprocessor or its corresponding vertex in the underlying graph. The term 'link' is taken as synonymous to 'edge'. In most cases we will have each node labeled by a unique name, called the *address* of the node. The set of nodes will be equivalent to the set of their addresses.

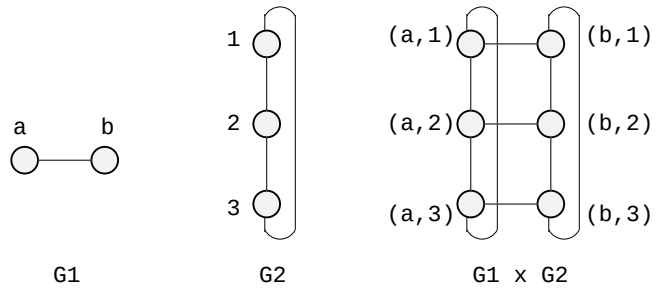

Figure 2.2. *Cartesian product of two graphs*

2.1.1 Multidimensional networks

For our purposes, it is also useful to define the (*cartesian*) *product* of graphs [12]. Given k graphs $G_i = (V_i, E_i)$, $i = 1, \ldots, k$, their product is defined as the graph $G = G_1 \times \cdots \times G_k = (V, E)$ whose vertices are labeled by a k-tuple $(v_1, \ldots, v_k)$ and

$$V = \Big\{(v_1, \ldots, v_k) \mid v_i \in V_i, i = 1, \ldots, k\Big\}$$
$$E = \Big\{\big((v_1, \ldots, v_k), (u_1, \ldots, u_k)\big) \mid \exists j \text{ s.t. } (v_j, u_j) \in E_j \text{ and } v_i = u_i \text{ for all } i \neq j\Big\}.$$

Such products of graphs are also termed *multidimensional* graphs here and G_i is called the ith *dimension* of the product. The ith component of the address tuple of a node will be called the ith *address digit* or the ith *coordinate*. An example is given in Fig. 2.2. Dimension 1 is a two-node graph with $V_1 = \{a, b\}$ while dimension 2 consists of a three-node cycle with $V_2 = \{1, 2, 3\}$. Their product has the node set given by:

$$V = \Big\{(a,1), (a,2), (a,3), (b,1), (b,2), (b,3)\Big\}.$$

According to the definition, node $(a, 1)$ has the following neighbors: since node a is adjacent to node b in the first dimension, node $(a, 1)$ will be adjacent to node $(b, 1)$; since node 1 is adjacent to both nodes 2 and 3 in the second dimension, node $(a, 1)$ will also be adjacent to nodes $(a, 2)$ and $(a, 3)$.

For multidimensional graphs it is known that if v_i has degree d_{v_i} and eccentricity $e_i(v_i)$ in G_i then the degree of $v = (v_1, \ldots, v_k)$ and its eccentricity in G are given by

$$d_v = \sum_{i=1}^{k} d_{v_i} \tag{2.1}$$

$$e(v) = \sum_{i=1}^{k} e_i(v_i). \tag{2.2}$$

Also, if $dist_i(v_i, u_i)$ is the distance between v_i and u_i in G_i then the distance between $v = (v_1, \ldots, v_k)$ and $u = (u_1, \ldots, u_k)$ in G is

$$dist(v, u) = \sum_{i=1}^{k} dist_i(v_i, u_i). \tag{2.3}$$

In the context of multidimensional graphs it will be convenient to use the *don't care* symbol '$*$' as a shorthand notation for a set of addresses. An appearance of this symbol at an element of an address tuple represents all legal values of this element. In the previous example, $(a, *) = \{(a, 1), (a, 2), (a, 3)\}$, $(*, 1) = \{(a, 1), (b, 1)\}$ while $(*, *)$ denotes the entire node set of the graph.

The efficiency of a multiprocessor interconnection structure is closely related to the characteristics of the underlying graph. For example, the degree is in direct correspondence with the amount of communication hardware required at the nodes. On the other hand, the diameter determines the maximum delay that a message has to suffer when traveling between two nodes. Loosely speaking, multidimensional graphs have the desired feature of relatively small diameters but low-dimensioned graphs, on the other hand, yield better performance when it comes to VLSI implementation [15], due to their smaller degrees.

2.1.2 Symmetry

In most cases, especially for general-purpose machines as opposed to machines optimized for a specific problem, the interconnection possesses some type of symmetry. This symmetry is usually expressed as "every node has the same view of the network". Such a characteristic is quite desirable since the implementation of the network is based on only one design: a single type of node with a single type of communication hardware. No node is "special" or different than the others.

A graph is *node symmetric* (or *vertex transitive*) [8, 12] if there exists a mapping of any vertex to any other vertex such that the edges are preserved. Formally, an *automorphism* α of a graph is a one-to-one mapping of the vertices to the vertices such that edges are mapped to edges. That is, $(\alpha(v), \alpha(u)) \in E$ iff $(v, u) \in E$. A graph G is node symmetric if for any two vertices v and u there exists an automorphism α of G such that $\alpha(v) = u$.

It is easy to see that multidimensional graphs consisting of node symmetric dimensions are also node symmetric. Let $G = G_1 \times \cdots \times G_k$. Assuming that every dimension G_i is node symmetric, we will show that G is also node symmetric. Consider any node $v = (v_1, \ldots, v_k)$ and pick any other node $v' = (v'_1, \ldots, v'_k)$. Let α_j be an automorphism in the jth dimension that maps v_j to v'_j. Such an automorphism exists since G_j is node symmetric. Consider

the one-to-one mapping

$$\alpha(v_1, \ldots, v_k) = (\alpha_1(v_1), \ldots, \alpha_k(v_k)).$$

Clearly this maps v to v'. We only need to show that the mapping preserves the edges. If (u, w) was an edge in G then by definition all coordinates of u and w are the same except the jth, for some j, where $(u_j, w_j) \in E_j$. Since $u_i = w_i$ for all $i \neq j$, we have that $\alpha_i(u_i) = \alpha_i(w_i)$. Since α_j preserves the edges in G_j, we have that $(u_j, w_j) \in E_j \Rightarrow (\alpha_j(u_j), \alpha_j(w_j)) \in E_j$. This shows that $(\alpha(u), \alpha(w)) \in G$, proving the claim.

Node symmetric graphs are regular (every node has the same degree) and if $n_j(v)$ is the number of nodes at distance j from v then $n_j(v) = n_j(v')$ for all v' in G and for all j. A direct consequence is that the eccentricities of the nodes are the same, all equal to the diameter of the graph.

2.2 Node-disjoint Paths in Multidimensional Networks

The most basic form of communication in interconnection networks is communication between a pair of nodes. This is accomplished by constructing a path between the two nodes of interest. In any (connected) k-dimensional graph, a path between two nodes $v = (v_1, \ldots, v_k)$ and $u = (u_1, \ldots, u_k)$ can be constructed as follows. Since dimension i (G_i) is connected, there exists a path between nodes v_i and u_i in G_i, denoted as $v_i \rightarrow u_i$. Then the following is a path from v to u in G:

$$(v_1, v_2, \ldots, v_k) \rightarrow (u_1, v_2, \ldots, v_k) \rightarrow (u_1, u_2, \ldots, v_k) \rightarrow \cdots \rightarrow (u_1, u_2, \ldots, u_k).$$

This method of path construction is sometimes referred to as coordinate correction since the path traverses dimensions sequentially, and in dimension i it aims at "correcting" the ith coordinate of v to the ith coordinate of u. As an example, consider the graph in Fig. 2.3(a). A path from node $(1, 1)$ to node $(2, 3)$ is

$$(1, 1) \rightarrow (2, 1) \rightarrow (2, 3),$$

where $(2, 1) \rightarrow (2, 3)$ is the path $((2, 1), (2, 2), (2, 3))$. Notice also that "partial" corrections are possible where, while traversing dimension i for example we do not reach up to node u_i but rather reach an intermediate node x_i, then traverse other dimensions and return later to the ith dimension to correct from x_i to u_i. Fig. 2.3(b) shows an example. We first correct partially the second dimension (vertical) from $(1, 1)$ to $(1, 2)$, then correct the

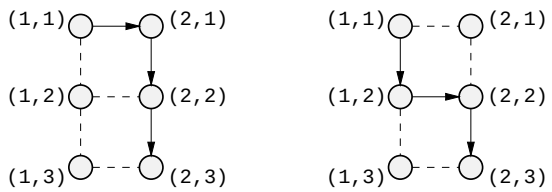

Figure 2.3. *Paths in multidimensional graphs*

first dimension (horizontal) from $(1,2)$ to $(2,2)$ and complete the correction in the second dimension from $(2,2)$ to $(2,3)$.

Reliability and speed requirements make multiple node-disjoint paths (i.e. paths that do not share any node except the first and the last one) between a pair of nodes quite desirable. If there was only a unique path between two nodes then failure in any of the intermediate nodes of the path would result in an inability to communicate. The presence of more than one path that utilizes different intermediate nodes offers thus improved reliability. In another context, when moving large amounts of data between a pair of nodes and there exist p node-disjoint paths between them, the most efficient scheme is to partition the data into p equal parts; sending each part over a different path completes the transfer in $1/p$th the amount of time it would take if we utilized only one path. We are thus interested in constructing as many node-disjoint paths as possible between any pair of vertices.

We show here that in a multidimensional graph $G = G_1 \times \cdots \times G_k$ where in the ith dimension there exist p_i node-disjoint paths between v_i and u_i, there exist $\sum_{i=1}^{k} p_i$ node-disjoint paths from $v = (v_1, \ldots, v_k)$ to $u = (u_1, \ldots, u_k)$.

Let $x_{i1}, x_{i2}, \ldots, x_{ip_i}$ be the "penultimate" nodes in the paths of the ith dimension, i.e. the nodes which are adjacent to u_i in each of the p_i node-disjoint paths between v_i and u_i. The ith class of paths we will construct consists of first correcting partially the ith coordinate up to the penultimate nodes, then the $(i+1)$th, ..., kth, 1st, ..., $(i-1)$th dimension in sequence and finally completing the correction of the ith dimension from the penultimate nodes to u_i:

$$\begin{aligned}
(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k) &\rightarrow (v_1, \ldots, v_{i-1}, x_{ij}, v_{i+1}, \ldots, v_k) \\
&\rightarrow (v_1, \ldots, v_{i-1}, x_{ij}, u_{i+1}, \ldots, v_k) \\
&\rightarrow \cdots \\
&\rightarrow (u_1, \ldots, u_{i-1}, x_{ij}, u_{i+1}, \ldots, u_k) \\
&\rightarrow (u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_k),
\end{aligned}$$

for all $j = 1, 2, \ldots, p_i$. Notice that since the paths from v_i to x_{ij} do not share any intermediate nodes with paths from v_i to $x_{ij'}$ ($j' \neq j$), the p_i paths in this ith class are node-disjoint just before they leave the ith dimension. After we leave the ith dimension, *every* node in the jth path has its ith digit equal to x_{ij} until the very last edge is traversed in order to correct the ith dimension to u_i. Consequently, since the x_{ij} are distinct, there is no node in common between the p_i paths in the the ith class. Furthermore, it is not very hard to see that there are no nodes in common between the ith class and the i'th class for any $i' \neq i$, due to the different sequence of corrections. The conclusion is that there exist $\sum p_i$ node-disjoint paths from v to u as claimed.

There is only one more case to consider. If $v_i = u_i$ then clearly p_i should be zero. Nevertheless, we can construct $\sum p'_i$ node-disjoint paths between v and u where $p'_i = p_i$ if $v_i \neq u_i$ and $p'_i = d_{v_i}$ (the degree of v_i in G_i) if $v_i = u_i$, as follows. Use the construction given above for all classes i such that $v_i \neq u_i$. For a class i for which $v_i = u_i$, let the "penultimate" nodes x_{ij}, $j = 1, \ldots, d_{v_i}$, be the d_{v_i} neighbors of v_i in G_i and follow the same construction. We have thus proven the following:

Theorem 2.1 *There exist* $\sum_{i=1}^{k} p_i$ *node-disjoint paths between nodes* $(v_1, \ldots, v_k)$ *and* $(u_1, \ldots, u_k)$ *in* $G_1 \times \ldots \times G_k$, *where* p_i *is the number of node-disjoint paths between* v_i *and* u_i *in* G_i *if* $v_i \neq u_i$, *and* $p_i = d_{v_i}$ *if* $v_i = u_i$. □

2.3 Some Networks of Interest

Linear Array

A linear array is one of the simplest interconnection networks; it is a graph L_n consisting of a path on n nodes, which are labeled 1 to n as shown in Fig. 2.4. Nodes i and $i+1$ are adjacent for all $1 \leq i \leq n-1$. The diameter is clearly equal to $n-1$. The degree of each node is 2 except for nodes 1 and n which have degree 1.

Ring

A ring R_n is an n-node graph consisting simply of a cycle. Nodes are labeled in a clockwise manner from 0 to $n-1$, and node i is adjacent to nodes $i \pm 1 \bmod n$ as in Fig. 2.4. Rings are regular graphs with degree 2 and diameter equal to $\lfloor n/2 \rfloor$.

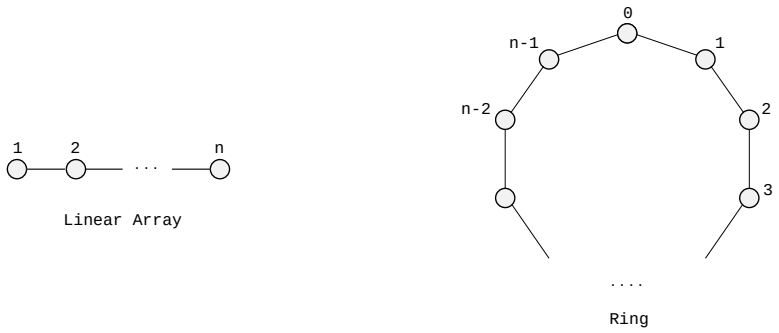

Figure 2.4. *Linear arrays and rings*

Extended Ring

An extended ring E_n^ρ consists of a ring enriched with additional edges which in effect give a graph with a smaller diameter. Specifically, if the ring has n nodes, then in E_n^ρ node i is adjacent to nodes $i \pm 1 \bmod n$, $i \pm 2 \bmod n$, ..., $i \pm \rho \bmod n$, where $\rho \leq n/2$ is the *connectivity parameter*. The ring in Fig. 2.4 has $\rho = 1$. Fig. 2.5 shows E_{14}^2. These graphs are a special case of certain node symmetric graphs called *circulants* [10] and have degree equal to 2ρ if $\rho < n/2$ ($2\rho - 1$ if $\rho = n/2$) and diameter equal to $\lceil \lfloor n/2 \rfloor / \rho \rceil$.

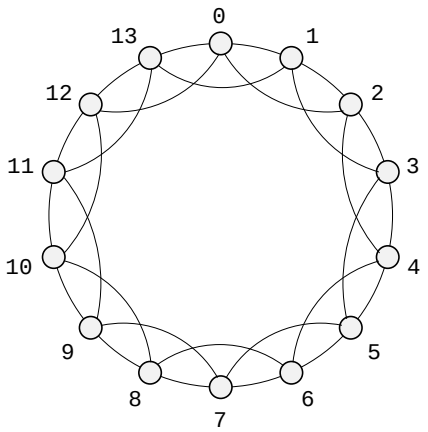

Figure 2.5. *Extended ring* E_{14}^2

Mesh

Meshes are cartesian products of linear arrays, see Fig. 2.6. One can see that in the $4 \times 3 \times 2$ mesh, the first dimension (vertical) is a linear array of 4 nodes, the second dimension (horizontal) is a linear array of 3 nodes and the third dimension is a 2-node linear array.

For illustration purposes, we concentrate on square two-dimensional meshes, i.e. $n \times n$ meshes. In such a graph, vertices are labeled as (i, j), $1 \leq i \leq n$, $1 \leq j \leq n$. Vertices (i, j) and (k, l) are adjacent if $|i - k| = 1$ and $j = l$, or $|j - l| = 1$ and $i = k$, that is, without considering the boundary nodes, node (i, j) is adjacent to nodes $(i+1, j)$, $(i-1, j)$, $(i, j + 1)$, $(i, j - 1)$. On the other hand, a boundary node has degree 3 and the four corner nodes have degree 2. A minimum length path from node (i, j) to node (k, l) can be constructed by traveling in the ith row till we meet column l and then in the lth column till row k. Consequently, the diameter of the mesh is equal to $2n - 2$, since this is the distance between nodes $(1, 1)$ and (n, n).

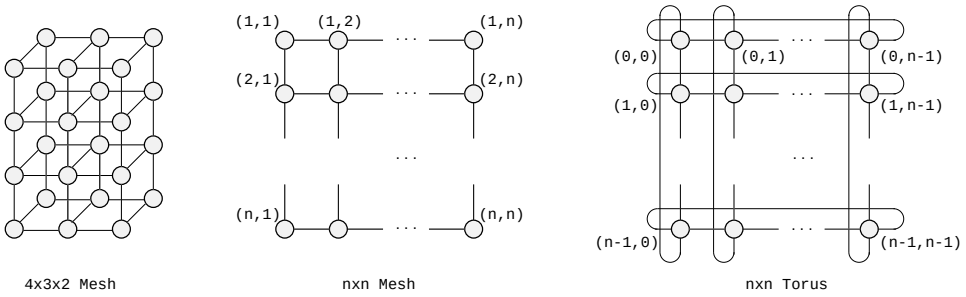

Figure 2.6. *Meshes and tori*

Torus

Tori are products of rings, see Fig. 2.6. Tori with m dimensions and n nodes per dimension are also known as n-ary m-cubes. We concentrate on the $n \times n$ torus (or n-ary 2-cube). Vertices are labeled by (i, j) where $0 \leq i \leq n-1$ and $0 \leq j \leq n-1$. Vertex (i, j) is adjacent to the four vertices $(i \pm 1 \bmod n,\ j)$ and $(i,\ j \pm 1 \bmod n)$, for all i and j. A minimum length path is constructed as in the case of meshes but when moving in a row or a column we select the shortest of the two available paths. The diameter is equal to $2\lfloor n/2 \rfloor$.

Hypercube

There are several ways to define a hypercube, Q_d, consisting of $n = 2^d$ nodes. One is to view it as a cartesian product of d linear arrays (or rings) of two nodes each. A more informative way though is to represent each node with an address between 0 and $2^d - 1$. Two vertices are adjacent if the binary representations of their addresses differ in exactly one bit, see also Fig. 2.7. Q_d is also called binary d-cube or simply d-cube. The degree and the diameter of Q_d are equal to $d = \log_2 n$.

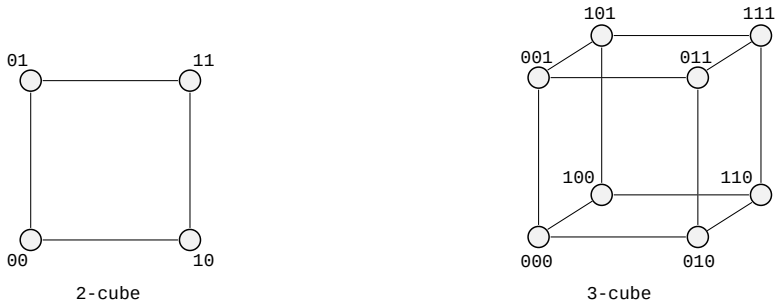

Figure 2.7. *Hypercubes*

Hypercycle

Hypercycles are products of extended rings. Each extended ring has its own connectivity parameter ρ_i, $i = 1, 2, \ldots, d$, in an d-dimensional hypercycle. The degree and the diameter of the graph are the sum of the degrees and of the diameters, correspondingly, of its dimensions. Tori and hypercubes are hypercycles with $\rho_i = 1$, for all i.

2.4 Communication Model

After defining the networks of interest, we now describe the assumptions we are going to make in order to model communications among the nodes. Nodes are supposed to exchange information in the form of messages. Messages traverse a path from a *source* node to one or many *destination* nodes. We will assume that the networks are packet switched. As explained in Chapter 1, this means that messages are split in packets of fixed size and a node must receive a packet in its entirety before using it or forwarding it to another node. In this thesis we will follow the common approach of treating the terms 'message' and 'packet' as synonymous; in other words, for the information exchange in the communication problems we consider, we will take the packet size to be equal to the size of the message(s) exchanged.

One may model the time taken for transferring a message between two nodes as follows: if t_s is the *set-up* time (preparing a link for transfer) and t_p is the propagation time of a single bit of data over a physical link, then a message (or packet) consisting of L bits is transferred in time

$$t = t_s + Lt_p.$$

This is referred to as the *linear model*. In this thesis we will follow the *constant model* where t does not depend on the message length. This is a usual assumption in the literature when studying theoretical properties of communications in networks. It is valid if the message

sizes are small so that the term Lt_p is negligible as compared to t_s. The set-up time in most of the commercial machines is known to be quite large as compared to the propagation delay [24, 21]. We are thus going to assume that transferring a message over a link requires constant time, and we will normalize this time to unity

$$t = 1 \quad \text{time unit.}$$

We will also refer to one time unit as one *step*.

Usually, we will be interested in optimizing the time needed for a communication operation. Such optimizations under the linear model are difficult (if not impossible) in many cases and the common practice is to optimize with respect to the total number of transmission set-ups and/or propagation delays separately. In view of this, optimization under the constant model which we are going to assume here actually results in optimization of the number of set-ups under the linear model. We should lastly point out that there have been some authors who considered different time models as discussed in [25].

The next issue concerns the bidirectional nature of the links. We have already mentioned that we deal mainly with undirected graphs where an edge between two nodes v and u actually corresponds to two links, one from v to u and one from u to v. Thus each edge can accommodate two directions of movement. An edge is *half-duplex* if only one direction can be accommodated at a time or *full-duplex* if both directions can be used simultaneously. Since a half-duplex edge can emulate a full-duplex one in two time steps, we will only consider full-duplex links. Any algorithm we will present can be trivially executed over half-duplex links with a slowdown factor of at most two.

The final parameter to our model is port availability. A node of degree d has d neighbors, hence d communication ports. Depending on the implementation of the machine, such a node may be able to send messages only to one neighbor at a time. This will be referred to as the *single-port* assumption and is suited for many early and current parallel machines. Unless otherwise stated, we will assume that each node can also receive at most one message in each step. Actually, our arguments will also work in the case where more than one reception is allowed (but only one message can be sent at a time). The *multiport* assumption relaxes these restrictions, that is, a node can communicate (send and receive messages) with all its neighbors simultaneously. The nCUBE-2 is one example of a machine that allows this overlapping of ports [47]. In many cases the design and the performance of communication algorithms is heavily dependent on the port availability assumption in effect.

Chapter 3

Single-source Communications

In this chapter we consider communication patterns for which one particular node needs to communicate with all the other nodes in the network. First we take a closer look at broadcasting where one node must disseminate a unique message. Under the multiport model we review the known results and we give a general construction for multidimensional networks which can be viewed as a generalization of the binomial tree for hypercubes. If the single-port assumption is in effect, there is no known result that determines the time needed to perform broadcasting in arbitrary graphs. Studies in the literature deal primarily with specific networks. If the network is not known one may only determine lower bounds on broadcasting time based on given properties of the network. For this case, we derive analytically new lower bounds which include some known bounds as special cases.

The problems of scattering and gathering are examined next. Gathering algorithms can be had from scattering algorithms (and vice versa) by reversing the directions of the paths traversed [5] so there will not be a special treatment of gathering here. We show first that under the single-port model scattering is always completed in $|V|-1$ steps where $|V|$ is the number of nodes in the network. It is seen that this may be achieved by any spanning tree of the network. For the multiport model there exists only a straightforward lower bound which may not always be tight. Through counterexamples it is seen that scheduling over a spanning tree is not always optimal. We finally examine scattering in the context of extended rings and tori.

3.1 Broadcasting Under the Multiport Model

It is known that if nodes can communicate with all their neighbors simultaneously then broadcasting from a node v requires $e(v)$ steps where $e(v)$ is the eccentricity of v. A simple algorithm is the following: once a node receives the message it sends it to all its neighbors. Of course, many nodes will receive the message more than once. One is thus interested

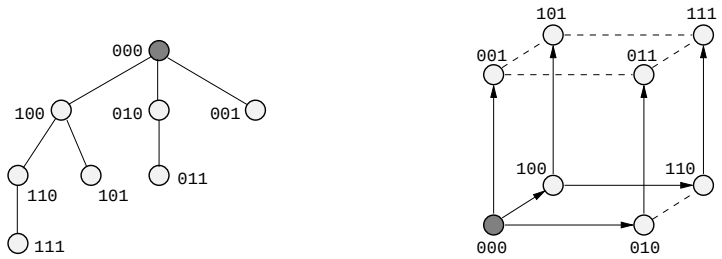

Figure 3.1. *Broadcasting in Q_3*

in eliminating redundancies. This may be achieved with any tree of height $e(v)$ rooted at v, e.g. a shortest-paths spanning tree of the graph such as one constructed by Dijkstra's algorithm or by a breadth-first search of the graph [28].

For hypercubes, a tree suitable for broadcasting is the well-known *binomial tree* [63, 36]. Assuming that we broadcast from node $00\cdots0_2$ in Q_d, the binomial tree is constructed as follows: the root node sends the message to its d neighbors $10\cdots0_2$, $01\cdots0_2$, ..., $00\cdots1_2$. When a node with binary address $b_{d-1}\cdots b_k100\cdots0$ receives the message, it sends it to nodes $b_{d-1}\cdots b_k110\cdots0$, $b_{d-1}\cdots b_k101\cdots0$, ..., $b_{d-1}\cdots b_k100\cdots1$. The general principle is that an intermediate node receiving the message through a link in dimension k informs its neighbors in all dimensions lower than k. An example in Q_3 is given in Fig. 3.1.

For other topologies, constructions of spanning trees with various properties have been given in [5, 25]. We have given an optimal spanning tree for broadcasting in extended rings elsewhere [17] (a short description follows shortly). An example is given in Fig. 3.2 for E_{14}^2. Notice that this is not a shortest-paths spanning tree since node 8 is reached through a path of length four although its distance from node 0 is three. Nevertheless the height of the tree is equal to the diameter of E_{14}^2 which is enough to guarantee optimality. More importantly, the tree can be generated "on the fly", that is, when the message reaches an intermediate node, the node can find out to which neighbors it should be sent (i.e. find its children in the tree) without any global information. This leads to a fully distributed broadcasting algorithm that uses only local information at each node.

Below we outline the broadcasting algorithm that implicitly constructs such a spanning tree for E_n^ρ. For integers i, j, let $i \oplus j$ and $i \ominus j$ be their sum and difference modulo n. Let

$$\begin{aligned} D &= \left\lceil \frac{\lfloor n/2 \rfloor}{\rho} \right\rceil \quad \text{(the diameter)} \\ w &= \left\lfloor \frac{n-1}{\rho} \right\rfloor - D \\ k &= n - (w+D)\rho - 1. \end{aligned}$$

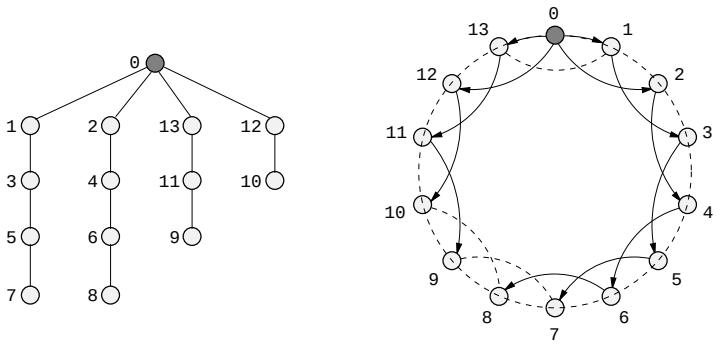

Figure 3.2. *Broadcasting in* E_{14}^2

The source node attaches a "weight" field to the message. If node i is the source node, then it sends the message with weight D to its neighbors in the clockwise direction, i.e. nodes $i \oplus 1, \ldots, i \oplus \rho$. It also sends the message to the first k counterclockwise neighbors (nodes $i \ominus 1, \ldots, i \ominus k$) with weight $w+1$ and to the remaining counterclockwise neighbors (nodes $i \ominus (k+1), \ldots, i \ominus \rho$) with weight w. Any node j receiving the message follows the algorithm:

1. decrease weight by one

2. if weight is zero then stop
   else send message to node $j \oplus \rho$ if it was received from the counterclockwise neighbor or to node $j \ominus \rho$ otherwise.

Without going into more detail, we mention some of the properties of the generated spanning tree [17]:

- the root has 2ρ children,
- each subtree rooted at a child of the root is a path,
- ρ of these subtrees have height $D-1$, k have height w and $\rho - k$ have height $w-1$,
- a furthest node from the root is at distance D.

The last property shows that broadcasting with this scheme takes the minimum possible time.

3.1.1 Broadcast trees in multidimensional networks

Consider a graph $G = G_1 \times \cdots \times G_k$ and assume that we want to broadcast from a node $v = (v_1, \ldots, v_k)$. We will provide a recursive construction of a spanning tree for G, rooted at node v. This tree can be seen as a generalization of the binomial tree for hypercubes in the

sense that when a node receives the message from a neighbor in dimension i it broadcasts in all dimensions lower than i. The only difference is that it also takes part in broadcasting within dimension i.

Let $\mathbf{T}^{(k-1)}$ be a spanning tree for broadcasting from node $(v_1, \ldots, v_{k-1})$ in $G_1 \times \cdots \times G_{k-1}$ and let $T^{(i)}_{v_i}$ be a spanning tree for broadcasting from node v_i in G_i, $i = 1, 2, \ldots, k$. Then a spanning tree $\mathbf{T}^{(k)}$ for G is derived as follows:

> Construct $|V_k|$ copies of $\mathbf{T}^{(k-1)}$ and attach a kth digit x_j to the address of all vertices in the jth copy, where $x_j \in G_k$, $j = 1, 2, \ldots, |V_k|$.
> Interconnect nodes $(v_1, \ldots, v_{k-1}, *)$ using the edges of $T^{(k)}_{v_k}$, i.e. if $(x_j, x_{j'}) \in T^{(k)}_{v_k}$ then $\big((v_1, \ldots, v_{k-1}, x_j), (v_1, \ldots, v_{k-1}, x_{j'})\big) \in \mathbf{T}^{(k)}$.

An example is shown in Fig. 3.3 for broadcasting from node $(2, 3, 1)$ in the three-dimensional $3 \times 4 \times 2$ mesh. A broadcast tree for the first dimension (horizontal) is constructed ($\mathbf{T}^{(1)} = T^{(1)}_2$). Then four copies of $\mathbf{T}^{(1)}$ are made, a spanning tree for the second dimension (vertical) is constructed ($T^{(2)}_3$), and vertices $(2, *)$ are interconnected according to $T^{(2)}_3$; this results in $\mathbf{T}^{(2)}$. The procedure is repeated once more for the third dimension to obtain the final tree $\mathbf{T}^{(3)}$.

Theorem 3.1 *The above construction yields a spanning tree of $G = G_1 \times \cdots \times G_k$ rooted at $(v_1, \ldots, v_k)$. Moreover, if the height of $T^{(i)}_{v_i}$ in G_i is equal to $e_i(v_i)$, the height of the above tree is equal to $e(v)$.*

Proof. If there exists only one dimension then trivially $\mathbf{T}^{(1)} = T^{(1)}_{v_1}$, a spanning tree of G_1 with the claimed properties. Assuming as an induction hypothesis that the theorem is true for $k-1$ dimensions, we will show it holds for k dimensions.

From the hypothesis it is seen than $\mathbf{T}^{(k-1)}$ is a spanning tree of $G' = G_1 \times \cdots \times G_{k-1}$. Thus the node set of $\mathbf{T}^{(k-1)}$ is the $(k-1)$-tuple $(*, \ldots, *)$, i.e. the node set of G'. $\mathbf{T}^{(k)}$ is constructed from $|V_k|$ copies of $\mathbf{T}^{(k-1)}$ and the jth copy has node set described by the k-tuple $(*, \ldots, *, x_j)$, $x_j \in V_k$. Since x_j takes all possible values in dimension k, the node set of $\mathbf{T}^{(k)}$ is the k-tuple $(*, \ldots, *, *)$ which is equal to the node set of G. Also, $\mathbf{T}^{(k)}$ only uses edges in G as seen easily from the construction. The conclusion is that $\mathbf{T}^{(k)}$ is a spanning subgraph of G.

We only need to show that $\mathbf{T}^{(k)}$ is a tree. $\mathbf{T}^{(k)}$ is connected since there is a path from $(v_1, \ldots, v_k)$ to any node $(u_1, \ldots, u_k)$: correct first the kth dimension to u_k by following edges of $T^{(k)}_{v_k}$ and then follow the (unique) path in $\mathbf{T}^{(k-1)}$ to correct the rest of the dimensions. Since $\mathbf{T}^{(k-1)}$ is a spanning tree of G', it has $|V'|-1 = |V_1| \cdots |V_{k-1}|-1$ edges (see Section 2.1). $\mathbf{T}^{(k)}$ has $|V_k|$ copies of $\mathbf{T}^{(k-1)}$ plus the edges of $T^{(k)}_{v_k}$ which must be $|V_k|-1$ in number. Thus

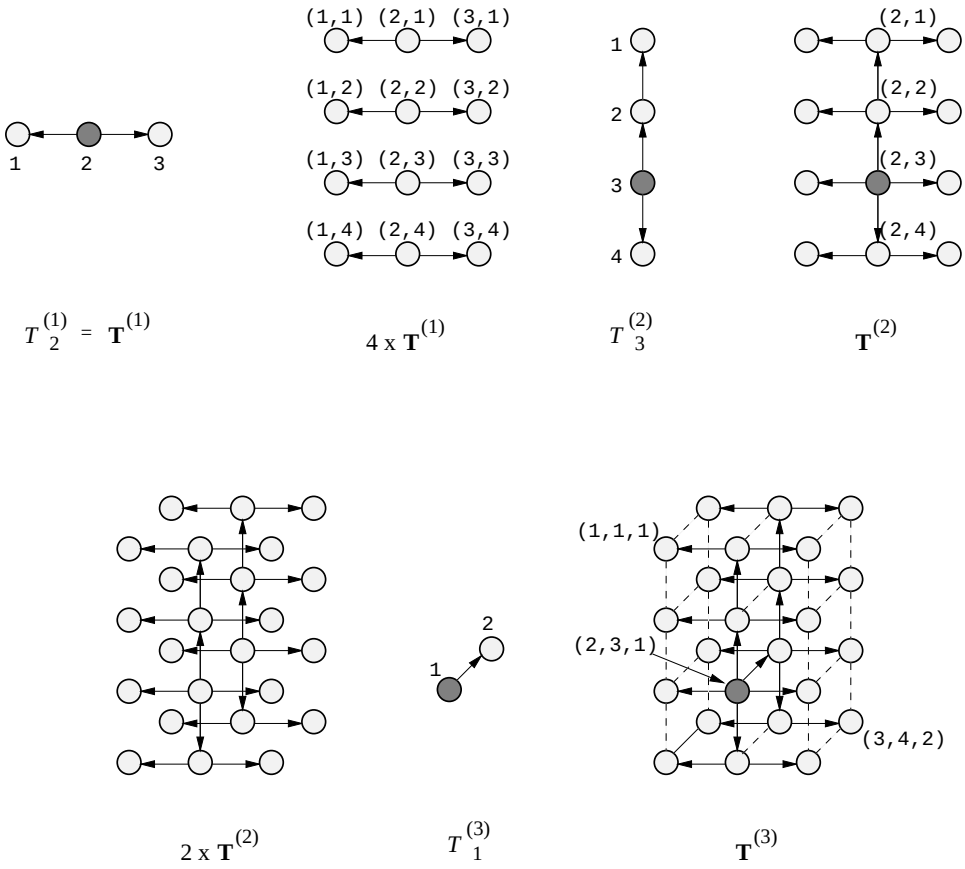

Figure 3.3. *Broadcasting in the* $3 \times 4 \times 2$ *mesh*

$\mathbf{T}^{(k)}$ has in total $|V_k| - 1 + |V_k|(|V'| - 1) = |V| - 1$ edges. Since $\mathbf{T}^{(k)}$ is a connected graph on $|V|$ vertices with $|V| - 1$ edges it must be a spanning tree.

If the height of $\mathbf{T}^{(k-1)}$ is equal to the eccentricity of $(v_1, \ldots, v_{k-1})$ in G', which is $\sum_{i=1}^{k-1} e_i(v_i)$ according to (2.2), and the height of $T_{v_k}^{(k)}$ is equal to $e_k(v_k)$ then the height of $\mathbf{T}^{(k)}$ is clearly $\sum_{i=1}^{k} e_i(v_i)$, the eccentricity of $(v_1, \ldots v_k)$ in G. ■

Consider some node $u = (u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_k)$ other than the root node, and its parent in $\mathbf{T}^{(k)}$ $w = (u_1, \ldots, u_{i-1}, w_i, u_{i+1}, \ldots, u_k)$ for some i. Because the edges of the tree are derived only from edges of the trees $T_{v_j}^{(j)}$, $j = 1, 2, \ldots, k$, it is seen that w_i was the parent of u_i in $T_{v_i}^{(i)}$. Moreover, since only nodes $(v_1, \ldots, v_{i-1}, *)$ were interconnected when constructing $\mathbf{T}^{(i)}$, it must be the case that $u_1 = v_1$, ..., $u_{i-1} = v_{i-1}$. Since u_i has a parent in $T_{v_i}^{(i)}$, it means that $u_i \neq v_i$. Consequently, no node $(v_1, \ldots, v_{i-1}, u_i, *, \ldots, *)$ is incident with edges from trees $T_{v_j}^{(j)}$ for $j > i$. The conclusion is that node u has no neighbors in

dimensions $j > i$ in the tree. If its parent is in dimension i then all its children lie in dimensions lower or equal to i; it will have a child in dimension i only if u_i is not a leaf in $T_{v_i}^{(i)}$. This is exactly the behavior of the binomial tree in hypercubes. Only in that case, each dimension has only two nodes; u_i is always a leaf in $T_{v_i}^{(i)}$ and thus it never has a child in dimension i in the final broadcast tree.

3.2 Broadcasting Under the Single-port Model

Consider a graph G and assume that a vertex can only communicate with one neighbor at a time. Let $B(v)$ be the (minimum) number of time units required to complete broadcasting from node v, called the broadcast time of v. One of the first known results is that $B(v) \geq \log_2 n$, where $n = |V|$. This can be seen as follows: if at some step m nodes know the message, they can inform at most another m nodes in the next step. In other words, the number of informed nodes can at most double after each step; hence the bound.

There has been a substantial amount of work on the broadcasting problem. Hedetniemi, Hedetniemi and Liestman [30] and Fraigniaud and Lazard [25] gave two excellent surveys on the subject. A large portion of the work has been concentrated on constructing efficient broadcast graphs, i.e. graphs that allow for broadcasting in the minimum possible time and contain as few edges as possible; see for example [29, 3] and the references therein. The rest of the work on the problem concentrates mostly on developing minimum-time algorithms or determining bounds on the broadcasting time of particular classes of graphs. In [42] bounds on $B(v)$ were given for graphs with maximum vertex degree three or four. The results were generalized in [3] for arbitrary maximum vertex degree.

Here we derive general lower bounds for the broadcasting problem in arbitrary graphs. Except for $B(v) \geq \log_2 n$, the only other general bound known is that $B(v) \geq e(v)$ where $e(v)$ is the eccentricity of the source node; this is easy to see since nodes at distance $e(v)$ from the source cannot be reached in less than $e(v)$ steps. Fraigniaud and Lazard [25] showed that if there exist at least 2 nodes at distance $e(v)$ from the source node, then at least $e(v) + 1$ steps are required. We show here that a general formula can be derived for the case where the number of nodes at distance greater than a given constant is known.

Consider a graph G and let v be the source node. We will assume that there exist n_i nodes in G which are at distance i from v, for all $i = 1, 2, \ldots, e(v)$. The sequence $\{n_1, n_2, \ldots, n_{e(v)}\}$ is known as the *distance degree sequence* of node v [12]. Based on this sequence, we are going to derive a lower bound for the broadcasting time, $B(v)$, of node v.

Let $r(\ell, t)$ be the maximum number of nodes that can be reached through paths of length

ℓ from v and which get informed exactly at time t. Notice that if during broadcasting a node is reached through a path of length ℓ, it is not implied that its distance from node v is ℓ; there may exist shorter paths, too. A node informed through a path of length ℓ at time t receives the message from a node lying on a path of length $\ell-1$ from v who got informed some time before t. Therefore $r(\ell,t)$ is described by the following recursion:

$$r(\ell,t) = r(\ell-1,t-1) + r(\ell-1,t-2) + \cdots + r(\ell-1,\ell-1). \tag{3.1}$$

Notice that if $\ell > t$ then no node ℓ links away from v has been informed as of time t. Thus $r(\ell,t) = 0$ for $\ell > t$ and this is why the above recursion stops at the term $r(\ell-1,\ell-1)$. The recursion thus holds for any $t \geq \ell$ and $\ell \geq 1$. Since the root can inform at most one of its neighbors at a time, only one node at distance 1 from the root can be informed at any step $t \geq 1$, leading to the boundary condition $r(1,t) = 1$. We finally define $r(0,0) = 1$; the source is assumed to become aware of the message at time 0.

Lemma 3.2

$$r(\ell,t) = \binom{t-1}{\ell-1}.$$

Proof. Eq. (3.1) can be written in a more familiar form by observing that $r(\ell-1,t-2) + r(\ell-1,t-3) + \cdots + r(\ell-1,\ell-1) = r(\ell,t-1)$. Consequently we have

$$r(\ell,t) = r(\ell-1,t-1) + r(\ell,t-1),$$

with $r(1,t) = 1$ for $t \geq 1$. The solution of the above recursion is $r(\ell,t) = \binom{t-1}{\ell-1}$ [44]. ■

3.2.1 General networks

Based on the above, we may express a general lower bound on broadcasting from node v. As we already mentioned, nodes that receive the message through paths of length ℓ are not necessarily at distance ℓ from v. In general, if a vertex is at distance ℓ' from v, it may be informed through any path of length $\ell \geq \ell'$. Concentrate on distance d and assume that broadcasting finishes at time $B(v) = T$. Then by time T all nodes at distance d or more must have been informed. According to the distance degree sequence, the number of nodes at distance at least d from v is

$$R_d = \sum_{i=d}^{e(v)} n_i.$$

Since such nodes must be informed through paths of length at least d, we must have

$$\sum_{\ell=d}^{T}\sum_{t=\ell}^{T} r(\ell,t) \geq R_d.$$

From Lemma 3.2 we obtain

$$\sum_{\ell=d}^{T}\sum_{t=\ell}^{T} r(\ell,t) = \sum_{\ell=d}^{T}\sum_{t=\ell}^{T} \binom{t-1}{\ell-1}.$$

The sum on the right-hand side evaluates to $\sum_{\ell=d}^{T}\binom{T}{\ell}$ [44]. Hence,

$$\sum_{\ell=d}^{T}\sum_{t=\ell}^{T} r(\ell,t) = \sum_{\ell=d}^{T} \binom{T}{\ell} \geq R_d. \tag{3.2}$$

We have thus proven the following:

Theorem 3.3 *If R_d is the number of nodes at distance d or more from v and T is the minimum t such that $\sum_{\ell=d}^{t}\binom{t}{\ell} \geq R_d$ then $B(v) \geq T$, i.e. broadcasting from v requires at least T steps.* □

Notice that if $d = 0$, $R_d = n$, i.e. all nodes in the network are included. In this case, using standard results, (3.2) reduces to

$$\sum_{\ell=0}^{T} \binom{T}{\ell} = 2^T \geq n,$$

which gives $T \geq \log_2 n$, the well-known bound for any network.

At the other extreme, consider the case of $d = e(v)$. If T is equal to $e(v)$ then (3.2) shows that $R_{e(v)} = n_{e(v)} \leq 1$. This was a result derived in [25]. We may obtain similar results for other values of $T \geq e(v)$. For example, if $T = e(v)+1$ then (3.2) gives $\binom{e(v)+1}{e(v)} = e(v)+1 \geq n_{e(v)}$. Consequently, if there exist more than $e(v)+1$ nodes at distance $e(v)$ from the source node, broadcasting requires at least $e(v)+2$ steps.

We next give an approximate formula for the minimum T due to the difficulty associated with handling inequality (3.2).

Corollary 3.4 *Broadcasting from v requires time at least equal to*

$$T = \begin{cases} 2d-1 & \text{if } R_d = 4^{d-1} \\ \log_2(4^{d-1}+R_d) & \text{if } R_d > 4^{d-1} \\ \dfrac{d}{2} + [d!(R_d-1)]^{1/(d+1)} & \text{if } R_d < 4^{d-1}. \end{cases}$$

Proof. Setting $T = 2d-1$, (3.2) evaluates to

$$\sum_{\ell=d}^{2d-1} \binom{2d-1}{\ell} = \frac{1}{2} \sum_{\ell=0}^{2d-1} \binom{2d-1}{\ell} = \frac{1}{2} 2^{2d-1} = 4^{d-1} \geq R_d,$$

hence the first branch of the result.

If $R_d > 4^{d-1}$ then $T > 2d-1$. We obtain

$$\begin{aligned} \sum_{\ell=d}^{T} \binom{T}{\ell} &= \sum_{\ell=0}^{T} \binom{T}{\ell} - \sum_{\ell=0}^{d-1} \binom{T}{\ell} \\ &\leq \sum_{\ell=0}^{T} \binom{T}{\ell} - \sum_{\ell=0}^{d-1} \binom{2d-1}{\ell} \\ &= 2^T - \frac{1}{2} 2^{2d-1}. \end{aligned}$$

Hence, $2^T \geq 4^{d-1} + R_d$, or $T \geq \log_2(4^{d-1} + R_d)$.

If $R_d < 4^{d-1}$ then $T < 2d-1$. In this case the maximum term of the sum in (3.2) is $\binom{T}{d}$. Consequently,

$$\sum_{\ell=d}^{T} \binom{T}{\ell} \leq (T-d) \binom{T}{d} + 1.$$

By standard combinatorial properties, $(T-d)\binom{T}{d} = (d+1)\binom{T}{d+1}$. Hence,

$$\sum_{\ell=d}^{T} \binom{T}{\ell} \leq (d+1) \frac{(T-d)(T-d+1)\cdots T}{(d+1)!} + 1.$$

It is well known that given m numbers $a_1, \ldots, a_m$, their geometric mean $(a_1 \cdots a_m)^{1/m}$ is less or equal to their arithmetic mean $(a_1 + \cdots + a_m)/m$ [51]. Thus, we seek the minimum T such that

$$\frac{[T - \frac{d}{2}]^{d+1}}{d!} \geq R_d - 1.$$

Taking the $(d+1)$th root of both sides yields the desired result (last branch of the equation). As a final note, for large m, Stirling's approximation gives $m! \approx \sqrt{2\pi m}(m/e)^m$, where e is the base of the natural logarithms. Consequently, if d is large and $R_d \leq 4^{d-1}$, the last branch of the corollary can be written as

$$T \geq \frac{d}{2} + \frac{d+1}{e} \left(\frac{(R_d - 1)\sqrt{2\pi}}{\sqrt{d+1}} \right)^{\frac{1}{d+1}}.$$

■

3.2.2 Trees

Trees are of particular interest since a broadcasting algorithm actually defines a spanning tree of the underlying network. The interested reader is referred to [56]. If the network is a tree, the path between any two nodes is unique and as a consequence, nodes informed through a path of length ℓ are at distance exactly ℓ from the root; (3.2) may thus take the simpler form

$$\sum_{t=d}^{T} r(d,t) = \binom{T}{d} \geq n_d,$$

which leads to the following corollary.

Corollary 3.5 *If the network is a tree rooted at v and T is the minimum t such that $\binom{t}{d} \geq n_d$ then $B(v) \geq T$, i.e. broadcasting from v requires at least T steps.* □

It is interesting to note that if $h = e(v)$ is the height of the tree, according to the above corollary, if broadcasting is to be completed in the minimum number of steps (i.e. $T = h$) then there must exist *at most* $\binom{h}{\ell}$ nodes in any level ℓ of the tree. Trees with exactly $\binom{h}{\ell}$ nodes at every level $\ell = 0, 1, \ldots, h$ and which achieve this minimum broadcasting time are unique [56]; they are the binomial trees we encountered in Section 3.1 which are used for broadcasting in hypercubes.

In conclusion, we derived lower bounds on the time needed to broadcast from a vertex in arbitrary networks. The bounds may be viewed as a generalization of a result in [25] and they are based on the distance degree sequence of the source node. The known bounds from the literature, $B(v) \geq \log_2 n$ and $B(v) \geq e(v)$ become special cases of our formulas. Tighter bounds may be derived if it is known that the network is of bounded degree, i.e. no node has degree greater than a given constant. In this case one may consider a recursion similar to (3.1) but with limited history. However to the best of our knowledge no closed-form solution can be obtained for such a recursion. In [38] an approximation was given for the case where the maximum degree is four. The only other known result for bounded-degree graphs is due to J.-C. Bermond *et al* [3] and is based only on the number of nodes in the network.

3.3 Scattering Under the Single-port Model

The problem of scattering involves sending $n-1$ distinct messages from a source node to the $n-1$ other nodes in the network. Since the source can send only one message at a time, any algorithm requires at least $n-1$ steps. In fact, there exists a simple solution to

the problem that requires exactly $n-1$ steps showing that the time needed for scattering under the single-port model is always $n-1$ time units, independently of the topology of the network. This fact was first observed by Bertsekas and Tsitsiklis [5, p. 81] but to the best of our knowledge no formal proof has appeared in the literature. Here we provide a simple proof through the next theorem.

Theorem 3.6 *In a network with n nodes, scattering can be performed in $n-1$ steps. Furthest-first scheduling of the messages over any spanning tree of the network always achieves the optimum time.*

Proof. Consider any spanning tree of the network rooted at the source node v and let $\{n_1, n_2, \ldots, n_{e(v)}\}$ be the distance degree sequence of v in the tree, that is, the number of nodes at distance 1, 2, ..., $e(v)$ from the root. Notice that none of the n_i are zero: $n_i \geq 1$ for all $i = 1, 2, \ldots, e(v)$, since the existence of a node in distance i implies the existence of a node at distance $i-1$. Now consider a furthest-first scheduling of the $n-1$ messages: messages meant for nodes at distance i leave before messages for nodes at distance $i' < i$. Then, the last message for a node at distance i will leave v at time $n_{e(v)} + n_{e(v)-1} + \cdots + n_i$. Because in a tree there are no cycles, this message can not collide with any other messages and will thus arrive exactly $i-1$ time units later, at time

$$T_i = n_{e(v)} + n_{e(v)-1} + \cdots + n_i + i - 1.$$

Since $T_{i+1} = n_{e(v)} + \cdots + n_{i+1} + (i+1) - 1 = T_i - n_i + 1$, and $n_i \geq 1$, we conclude that $T_{i+1} \leq T_i$ and T_i is a non-increasing function of i. Consequently, the last node to receive a message will do so at time

$$\max\{T_1, \ldots, T_{e(v)}\} = T_1 = n_{e(v)} + n_{e(v)-1} + \cdots + n_1 = n - 1.$$

■

In the light of the last theorem, it is seen that the broadcast tree constructed in Section 3.1.1 can also be used for an optimal single-port scattering algorithm in any multidimensional network.

3.4 Scattering Under the Multiport Model

Assuming that the source node v is able to communicate with all its d_v neighbors simultaneously, the time required for scattering from v, $S(v)$, is bounded below by

$$S(v) \geq \left\lceil \frac{n-1}{d_v} \right\rceil. \tag{3.3}$$

This is because out of the $n-1$ messages to be sent in total, at most d_v can leave node v at each step. A sufficient condition for attaining the lower bound has been given in [4].

Theorem 3.7 (Bertsekas *et al*, [4])
Let T be a spanning tree of a graph, rooted at node v. Let c be the number of children of v in T and let T_i be the subtree rooted at the ith child of v, $i = 1, 2, \ldots, c$. If N_i is the number of nodes in T_i then scattering from v requires $\max\{N_1, N_2, \ldots, N_c\}$ steps. The lower bound in (3.3) is achieved if the tree is balanced and $c = d_v$. □

The theorem is seen easily by observing that the ith neighbor of v receives one message at every step so that scattering in T_i follows basically the single-port model of the previous section. The term "balanced" refers to the number of nodes in each subtree. A tree is balanced if the c subtrees rooted at the children of the root have approximately the same number of nodes. A more precise statement is the following: none of the c subtrees has more than $\lceil (n-1)/c \rceil$ nodes. This is because the average number of nodes in the subtrees is $(n-1)/c$, thus all subtrees must have a population as close as possible to this average. Since the average is not always an integer, the ceiling operation is needed which also allows for some of the trees to have a small amount of nodes below the average. If in addition $c = d_v$ then (3.3) is tight.

Theorem 3.7 shows that if we are able to find a balanced spanning tree of a graph such that v has d_v children then we will be able to achieve the lower bound of (3.3). It is important to observe that this bound is not always tight. A simple example is shown in Fig. 3.4(a) where scattering needs three steps, one more than the time predicted in (3.3). Another observation is that the converse of Theorem 3.7 is not true. If there exists no balanced spanning tree in a graph, it is not implied that the bound of (3.3) cannot be achieved. In fact, the optimal schedule need not be a tree. We demonstrate this with a counterexample in Fig. 3.4(b). There exist only two spanning trees rooted at v, and such that v has two children. In both trees one subtree has four nodes (both trees are unbalanced) so that scattering over either of them needs four steps. The lower bound can nevertheless be achieved through a non-tree schedule: in the first step send the messages for nodes d and e through the two paths leading to node c. In the second step send the message for node c through any path and in the last step inform nodes a and b.

In conclusion, the lower bound on scattering time may not always be tight and the best possible scattering time is not necessarily attained by scheduling over a spanning tree of the graph. If there exists a balanced spanning tree such that the number of children of the root is equal to the node's degree in the graph, then scattering can be performed in time

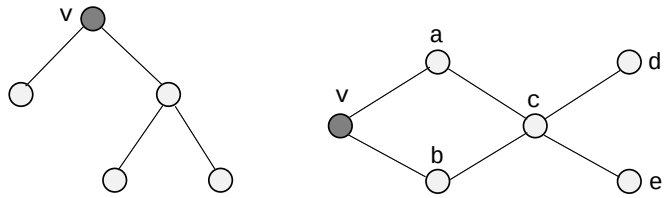

Figure 3.4.

equal to (3.3).

Scattering in linear arrays was considered in [5, p. 44]. Scattering in rings was considered in [60]; the algorithm was shown to be optimal under the linear time model in [26]. Bhatt *et al* [6] studied scattering and gathering in wormhole routed trees. For hypercubes, balanced tree constructions have been given in [32, 4]. We provide balanced trees for extended rings and two-dimensional tori in the next paragraphs.

3.4.1 Scattering in extended rings

We show here that the broadcasting tree given in Section 3.1 for E_n^ρ can also be used for scattering.

Theorem 3.8 *The spanning tree in Section 3.1 is an optimal scattering tree for* E_n^ρ.

Proof. According to Theorem 3.7, in order to prove the optimality of the tree it is enough to show that the tree is balanced. From the properties of the tree, we know that the root node has 2ρ children, equal to the degree of the graph. Each subtree is a path so that the number of nodes in the subtree is equal to its height plus one. The maximum height is $D-1$ thus D is the maximum subtree size. All we have to show is that

$$D \leq \left\lceil \frac{n-1}{2\rho} \right\rceil .$$

We already know that $D = \lceil \lfloor n/2 \rfloor / \rho \rceil$. If n is odd $\lfloor n/2 \rfloor = (n-1)/2$ and the inequality holds. Otherwise, $D = \lceil n/2\rho \rceil$, which is equal to $\lceil (n-1)/2\rho \rceil$ for even n. ■

3.4.2 Scattering in the 2D torus

Spanning trees for two-dimensional $n \times m$ tori have appeared in [5, 25, 53]. Some constructions have interesting properties but were designed for the broadcasting problem, they work for special values of n and m and they are not always balanced. The construction in

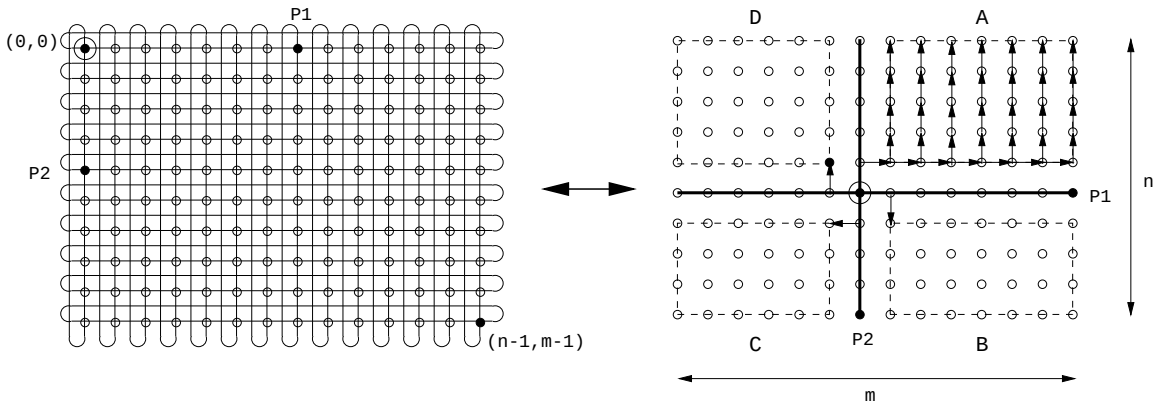

Figure 3.5. *Mapping a torus on the plane*

[5, pp. 81–82] is for square $n \times n$ tori and is also suitable for the multinode broadcasting problem. In the case of odd n this construction yields balanced trees, suggesting an optimal scheme for scattering. However according to the suggested extension for the even n case, the balanced-tree property fails and it is unclear how the construction can be generalized to an $n \times m$ torus where $n \neq m$.

An $n \times m$ torus is node symmetric, every vertex has degree four and the number of vertices is equal to nm. According to (3.3), the scattering time from any vertex is

$$S_{n\times m} \geq \left\lceil \frac{nm-1}{4} \right\rceil .$$

We will construct a spanning tree which yields scattering time equal to $\lceil (nm-1)/4 \rceil$ and is therefore optimal, for $n, m \geq 4$.

Let (c, r) be the source node. We first map the nodes of the torus to points in a cartesian coordinate system such that node (c, r) is identified with the center of the axes. A point on the plane with coordinates $[i, j]$ (i and j integers) corresponds to node

$$[i, j] \longleftrightarrow (r - j \bmod n,\ c - i \bmod m) \tag{3.4}$$

in the torus, where $-\lfloor (m-1)/2 \rfloor \leq i \leq \lceil (m-1)/2 \rceil$, and $-\lfloor (n-1)/2 \rfloor \leq j \leq \lceil (n-1)/2 \rceil$, as shown in Fig. 3.5. It is straightforward to see that this is a bijective mapping and that the neighbors of a node mapped to point $[i, j]$ on the plane are mapped to the neighbors of $[i, j]$.

Excluding the origin, the sizes of the positive y-axis ($y+$), the negative y-axis ($y-$), the positive x-axis ($x+$) and the negative x-axis ($x-$) are, correspondingly, $\lceil (n-1)/2 \rceil$, $\lfloor (n-1)/2 \rfloor$, $\lceil (m-1)/2 \rceil$ and $\lfloor (m-1)/2 \rfloor$ points. The four quadrants, not including the

axes, have sizes:

$$\begin{aligned}
|A| &= \left\lceil \frac{n-1}{2} \right\rceil \left\lceil \frac{m-1}{2} \right\rceil \quad \text{north-east region} \\
|B| &= \left\lfloor \frac{n-1}{2} \right\rfloor \left\lceil \frac{m-1}{2} \right\rceil \quad \text{south-east region} \\
|C| &= \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor \quad \text{south-west region} \\
|D| &= \left\lceil \frac{n-1}{2} \right\rceil \left\lfloor \frac{m-1}{2} \right\rfloor \quad \text{north-west region.}
\end{aligned}$$

Regions A, B, C and D will be 'owned' by points $[0,1]$, $[1,0]$, $[0,-1]$ and $[-1,0]$ correspondingly.

Inside a region a spanning tree can be constructed easily. In region A, starting from point $[0,1]$ we form a horizontal path spanning the width of the region. From each point in the path we form vertical segments spanning the height of the region, as in Fig. 3.5. For regions B, C and D we follow an analogous procedure.

Depending on the values of n and m, the four regions may not have the same number of points. The idea is, then, to expand the regions appropriately in order to obtain a balanced division of the area. Expansions are performed by either giving a whole axis to a region or by sharing points on the axes among adjacent regions. In the second case we follow an 'odd-even' rule: axis points with odd coordinate are given to the region that is owned by a point on the axis in question. For example, if we need to give half of the $y+$ axis to region A and the other half to (adjacent) region D we give all points $[0,j]$, where j is odd, to region A since A is owned by point $[0,1]$ which is on the $y+$ axis; all points $[0,j]$, where j is even, are given to region D. Notice that splitting an axis in two equal halves is not always possible. The size of the $y+$ axis is $\lceil (n-1)/2 \rceil$ points. The two halves of $y+$ have sizes

$$\left\lceil \frac{\lceil (n-1)/2 \rceil}{2} \right\rceil, \quad \left\lfloor \frac{\lceil (n-1)/2 \rceil}{2} \right\rfloor$$

correspondingly. It is easily seen that the region that gets the 'odd' half of the axis (region A in the above example) actually gets the bigger of the two halves. The four possible divisions of the plane are shown in Fig. 3.6. The spanning trees in each region are expanded in a straightforward manner as shown in the figure. For example, a point added to the left of a region becomes a child of the leftmost point in this region which has the same y coordinate as the added point.

Case 1: both n and m are odd.

In this case $(n-1)/2$ and $(m-1)/2$ are integers so that all four regions have the same size. Then all axes are shared among adjacent regions. Region A gets half of the $y+$ and

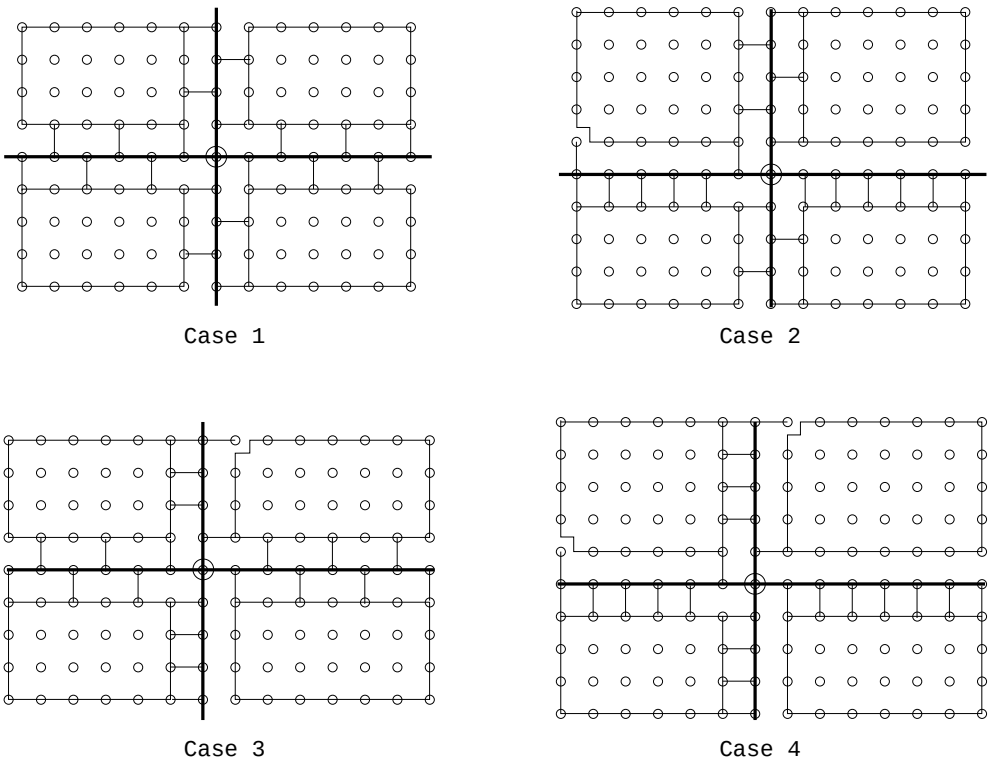

Figure 3.6. *Four cases of area division in an $n \times m$ grid*

half of the $x+$ axis. Region B gets half of the $x+$ and half of the $y-$ axis. Region C gets half of the $y-$ and half of the $x-$ axis and region D gets the remaining points on the axes.

Case 2: n is even and m is odd.

In this case the regions are expanded as follows: region A gets half of the $y+$ axis; region B gets the whole $x+$ axis and half of $y-$; the other half goes to region C along with the whole of $x-$. Finally, region D gets half of the $y+$ axis. In this case, however, point $[-1, 0]$ owns region D and lies on the $x-$ axis given to region C. To correct the imbalance, point $[-(m-1)/2, 1]$ is removed from region D and is given to region C.

Case 3: n is odd and m is even.

In this case the regions are expanded as follows: regions A and B share the $x+$ axis; region C gets the whole $y-$ and half of the $x-$ axis; region D gets half of $x-$ plus the $y+$ axis. Finally, because the owner of region A is on the $y+$ axis (point $[0, 1]$), the top-left corner of region A (point $[1, (n-1)/2]$) is removed from A and given to region D.

Case 4: both n and m are even.

In this case region A is not expanded. Region B is given the $x+$ axis, region D is given the $y+$ axis and region C gets $x-$ and $y-$. Since point $[0, 1]$ on $y+$ owns region

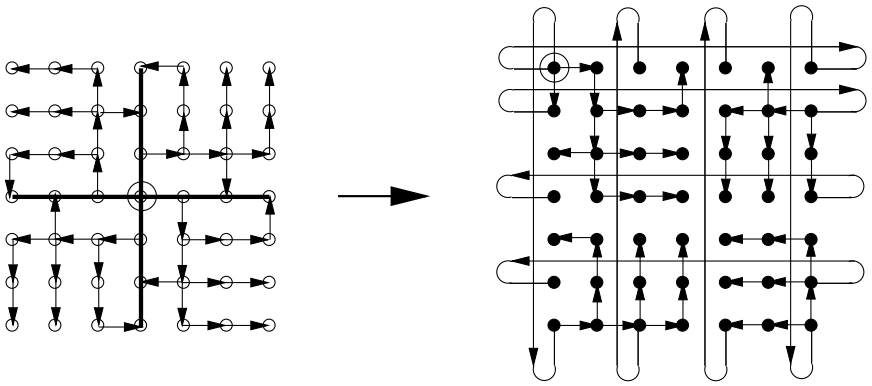

Figure 3.7. *Scattering in a* 7×7 *torus*

A, point $[1, \lceil (n-1)/2 \rceil]$ from region A is given to region D. For the same reason point $[-\lfloor (m-1)/2 \rfloor, 1]$ from region D is attached to region C.

We now show that the division in Case 2 is balanced. Quite similar arguments can be used to show that the division is balanced in the other three cases, too, so we will omit the details. Expressing n as 2κ and m as $2\lambda + 1$, we find that

$$\left\lceil \frac{nm-1}{4} \right\rceil = \kappa\lambda + \left\lceil \frac{2k-1}{4} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil \frac{m-1}{2} + \left\lceil \frac{n-1}{4} \right\rceil .$$

The sizes of the four expanded regions are

$$\begin{aligned}
|D| \le |A| &= \left\lceil \frac{n-1}{2} \right\rceil \frac{m-1}{2} + \left\lceil \frac{\lceil (n-1)/2 \rceil}{2} \right\rceil \\
&= \left\lceil \frac{n-1}{2} \right\rceil \frac{m-1}{2} + \left\lceil \frac{n}{4} \right\rceil \\
&= \left\lceil \frac{nm-1}{4} \right\rceil \\
|B| \le |C| &= \left\lfloor \frac{n-1}{2} \right\rfloor \frac{m-1}{2} + \left\lceil \frac{\lfloor (n-1)/2 \rfloor}{2} \right\rceil + \frac{m-1}{2} \\
&= \left(\left\lceil \frac{n-1}{2} \right\rceil - 1 \right) \frac{m-1}{2} + \left\lceil \frac{n-2}{4} \right\rceil + \frac{m-1}{2} \\
&\le \left\lceil \frac{nm-1}{4} \right\rceil ,
\end{aligned}$$

which shows that no region has more than $\lceil (nm-1)/4 \rceil$ nodes.

We conclude the section with an example on the 7×7 torus in Fig. 3.7. Both n and m are odd and the regions are expanded according to Case 1 above. In this case the four regions have identical shape and the spanning trees for regions B, C and D can be obtained by rotating the spanning tree in region A by 90°, 180° and 270°, correspondingly. The corresponding spanning tree in the torus can be obtained through the mapping in (3.4).

Chapter 4

Total Exchange

4.1 Introduction

In the total exchange problem, every node performs a scattering operation; in total $n(n-1)$ messages have to be sent in a network with n nodes. Clearly, the time required for total exchange is bounded below by the time needed for scattering. However this bound is not tight. Tighter bounds can be obtained for both the single-port and the multiport model.

First consider total exchange under the multiport model in $G = (V, E)$. Partition the vertex set V in two disjoint sets V_1 and V_2 such that $V_1 \cup V_2 = V$. Let $C_{V_1 V_2}$ be the number of edges in E joining the two parts, i.e. edges $e = (v, u)$ such that $v \in V_1$ and $u \in V_2$. Messages from nodes in V_1 destined for nodes in V_2 must cross these $C_{V_1 V_2}$ edges. The total number of such messages is $|V_1||V_2|$. Since only $C_{V_1 V_2}$ messages are able to pass from V_1 to V_2 at a time, we obtain the following lower bound for total exchange time:

$$TE_{\text{multiport}} \geq \frac{|V_1||V_2|}{C_{V_1 V_2}}. \tag{4.1}$$

We are of course interested in maximizing the fraction in the right-hand side by selecting V_1 and V_2 appropriately so that the tightest possible bound results. In many cases a *bisection* of the graph is the most appropriate choice, that is, a division of the nodes in two parts with equal (within one) number of nodes. Then $|V_1||V_2| = \lfloor n/2 \rfloor \lceil n/2 \rceil = \lceil (n^2-1)/4 \rceil$, maximizing the numerator of the fraction in (4.1). The *bisection width*, BW_G, of a graph is the minimum number of edges whose removal induces a bisection. Thus, (4.1) gives

$$TE(n) \geq \frac{\lceil (n^2-1)/4 \rceil}{BW_G}.$$

Determining the bisection width of a graph is an open research problem even for relatively simple networks such as d-dimensional meshes [40], so in most cases we just select a sensible partition of V such that the two parts are relatively close in size.

The bound of (4.1) is not appropriate for the single-port model though since each of the nodes can send at most one message. If some node of V_1 is incident with more than one of the $C_{V_1V_2}$ edges then not all these edges can be used to transfer messages simultaneously.

The *status* or *total distance* of a node v, $s(v)$, is defined as [12] the sum of the distances of all the other nodes in the network from node v. Let $\{n_1, n_2, \ldots, n_{e(v)}\}$ be the distance degree sequence of v. Then clearly,

$$s(v) = \sum_{i=1}^{e(v)} in_i. \tag{4.2}$$

The *average status* of the network is defined as

$$AS(G) = \frac{\sum_{v \in V} s(v)}{n}. \tag{4.3}$$

Since node v has to send a message to every other node in the network, it has n_i messages for nodes at distance i, each of which must travel over i links in order to reach its destination. Thus, the status of v also determines the total distance of messages of node v from their destinations when total exchange commences. Similarly, $\sum_{v \in V} s(v)$ represents the total distance for all messages of all nodes involved in the total exchange operation. Every time a message traverses a link its distance from its destination is reduced by one (at best). Total exchange is complete if every message is within distance zero from its destination. In effect, during total exchange the initial total distance $\sum_{v \in V} s(v)$ is reduced gradually to zero. If the single-port model is in effect, then only one message can be sent by a node at a time. Consequently, at most n messages can get closer to their destination; at most n units can be subtracted from $\sum_{v \in V} s(v)$ at a time. We thus obtain the lower bound

$$TE_{\text{single-port}} \geq \frac{\sum_{v \in V} s(v)}{n} = AS(G). \tag{4.4}$$

In conclusion, total exchange under the single-port model is bounded below by the average status of the network. If the graph is node symmetric then every vertex has the same distance degree sequence, hence the same status. Let s_G be the status of the vertices of a node symmetric graph G. Then clearly, (4.3) gives $AS(G) = s_G$ and

$$TE_{\text{single-port}}^{\text{node symmetric}} \geq s_G. \tag{4.5}$$

Notice that the number of message receptions does not play any role. Whether a node is allowed to receive one or more than one messages at a time is irrelevant to the derivation of (4.4) as long as one message departure per step is allowed from each node.

4.1.1 Total exchange in certain networks

Total exchange algorithms for various networks and under a variety of assumptions have appeared in [59, 60, 36, 4, 37, 33, 11, 61, 67, 24, 39, 64]. Under the packet-switched model, references [59, 36, 4, 37, 67] deal with hypercubes. Saad and Schultz [59] were among the first to study the communication problems we consider in this thesis. They gave algorithms to solve the problems in hypercubes, including an algorithm for the total exchange problem. This algorithm improved on a matrix-transposition algorithm which appeared in [35]. Johnsson and Ho [36] presented more efficient algorithms which were based on scheduling over certain spanning trees. The algorithms were close but not exactly optimal under the constant model. An optimal recursive algorithm was given by Bertsekas *et al* [4].

Algorithms for other networks have appeared in [60, 5, 67]. Rings and tori were considered in [60, 67]. The authors in [60] suggested a simple total exchange algorithm for rings whereby data are rotated around the ring, dropping a message at each stop. In [67] a different algorithm was constructed based on a matrix decomposition scheme associated with the total exchange problem. Both algorithms are optimal only when the ring has an odd number of nodes, as we will show later. Finally, linear arrays, among other networks, were considered in [5] where optimal algorithms for all but the total exchange problem were given.

In the following sections we will construct optimal total exchange algorithms under the multiport model for two simple networks: the linear array and the ring. Their importance is well-known [40] both for special-purpose as well as general-purpose computing (for example, the structure of the Homogeneous Multiprocessor [19] is based on a linear array). As was implied by the above survey, our algorithms are the only known optimal algorithms for the two networks of interest.

With few exceptions, most of the work on total exchange has been centered around the multiport model. In this chapter, in addition to the multiport model, we will consider total exchange under the single-port model. We provide a general construction for a certain type of networks (Cayley networks) and analytically prove its optimality. Optimal algorithms under the single-port model have appeared in the literature only for two such networks: the hypercube [5, pp. 81–83] and the star graph [50]. We instead give an optimal solution applicable to the entire class. Finally, our work on total exchange is continued in Chapter 5 where we will provide solutions for multidimensional networks.

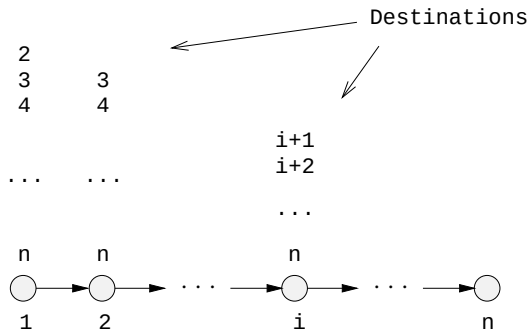

Figure 4.1. *Initial configuration of rightward messages*

4.2 Multiport Total Exchange in Linear Arrays

For a linear array L_n we may derive a lower bound on the total exchange operation using (4.1). We recall that nodes are labeled 1 to n from left to right (see Fig. 2.4 and 4.1). Let $V_1 = \{1, 2, \ldots, \lfloor n/2 \rfloor\}$ and $V_2 = V \setminus V_1$ where $\setminus$ is the set-theoretic subtraction. There is only one link separating the two halves, namely the link from node $\lfloor n/2 \rfloor$ to node $\lfloor n/2 \rfloor + 1$ so that $C_{V_1 V_2} = 1$. Since $|V_1||V_2| = \lfloor n/2 \rfloor (n - \lfloor n/2 \rfloor) = \lceil (n^2 - 1)/4 \rceil$, (4.1) gives

$$TE(L_n) \geq \left\lceil \frac{n^2 - 1}{4} \right\rceil = T_{opt}.$$

A simple algorithm was suggested in [5, p. 44] in order to determine the complexity of the operation: perform n consecutive scatterings, one from each node. The authors showed that an optimal scattering from node i can be completed in $\max\{i - 1, n - i\}$ steps. After some algebra, one can see that the time needed for this total exchange scheme is $3T_{opt} - \lfloor n/2 \rfloor$ which is about three times slower than the optimum. In the next paragraphs we present an algorithm that runs in time equal to T_{opt}.

Consider the messages that travel towards the right end of the array, i.e. messages originating from a node i and destined to a node j where $j > i$. These messages do not interfere with the ones traveling towards the left end since they use links of opposite direction. Contention occurs only between messages traveling in the same direction. Consequently the problem can be thought as two subproblems, each involving message transmissions in one direction, and under the multiport model they can be solved simultaneously. Without loss of generality, we can concentrate only on the rightward messages.

The initial configuration is shown in Fig. 4.1. Node i, $1 \leq i \leq n - 1$, has $n - i$ messages to send to its right. In addition, there are $i - 1$ messages in total meant for node i, residing at nodes $1, 2, \ldots, i - 1$. The following algorithm will be shown to deliver all messages in the

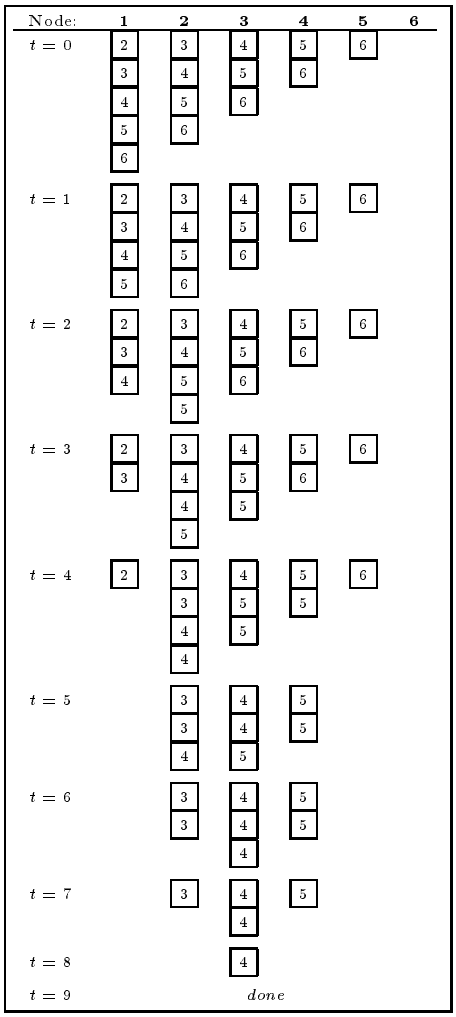

Figure 4.2. *A 6-node linear array example (only message destinations are shown)*

minimum number of steps:

> Do in parallel for all nodes $i = 1, 2, \ldots, n-1$, all the time:
> *if node i has any messages in its queue, it selects the message that has to travel the longest distance and sends it to its right.*

That is, messages are always scheduled using a *furthest-first* discipline. Notice that there is no synchronization required among nodes, although for our purposes we will assume that all nodes start at the same time and are synchronized by a common clock. In Fig. 4.2 we give an example for the case of $n = 6$ nodes. Notice also that when a message arrives at its destination it is immediately forwarded to the local processor, i.e. it does not join the node's message queue.

Although in the case of ties (two or more messages in the same node have to travel the same distance) all tie-breaking protocols will yield the same running time, we concentrate in a specific protocol. The protocol we will assume breaks ties in favor of the message that has *already* traveled the furthest distance, or in other words, in favor of the source with the smallest address.

Let $m_k(i)$ stand for the message of node k addressed to node i. Node k is the source and node i is the destination of the message. Based on our scheduling rules, if $m_k(i)$ competes with any message $m_l(j)$ at a node, then $m_l(j)$ will have priority over $m_k(i)$ if $j > i$, or if $j = i$ and $l < k$.

For illustration purposes, we view the evolution of the algorithm as a sequence of logical phases. Phase i starts after node $i-1$ gets emptied, i.e. has no more messages to send. Our basic argument is that by the time node i gets emptied, i.e. by the end of phase i, all messages addressed to node $n-i+1$ will have reached their destination. By the end of every phase the leftmost and the rightmost of the active nodes become idle, reducing the number of active nodes by two.

Message $m_k(i)$ has to go through all nodes j, $1 \leq k \leq j \leq i-1$, before it reaches node i. Let $t^{(j)}\big(m_k(i)\big)$ be the time $m_k(i)$ leaves node j; this is also the time $m_k(i)$ arrives at node $j+1$. For the special case of messages destined to node n observe that $m_j(n)$ leaves node j at time 1, for all $j \leq n-1$ and it moves towards n without any delays since it will have priority over all other messages residing at the intermediate nodes. Message $m_1(n)$ is the last message to reach node n (it arrives at time $t = n-1$) and in general $t^{(j)}\big(m_1(n)\big) = j$. We have the following lemma.

Lemma 4.1 *If* $1 \leq j < i \leq n-1$, *then*

$$t^{(j)}\big(m_1(i)\big) = \begin{cases} t^{(j)}\big(m_j(i+1)\big)+1, & i < n-1 \\ j+1, & i = n-1 \end{cases} \tag{4.6}$$

$$t^{(j)}\big(m_{k+1}(i)\big) = t^{(j)}\big(m_k(i)\big)+1, \quad 1 \leq k \leq j-1. \tag{4.7}$$

Proof. We prove the lemma using double induction on i and j.

(Basis 1: $i = n-1$). Messages destined for node n are not delayed anywhere, so that between time 1 and j, node j receives and sends messages $m_j(n)$, $m_{j-1}(n)$, ..., $m_2(n)$, $m_1(n)$ correspondingly. Message $m_1(n-1)$ leaves node 1 right after $m_1(n)$ and follows behind it without being delayed anywhere, as well. Hence it leaves node j right after $m_1(n)$ does, i.e. at time $j+1$, proving thus (4.6) for $i = n-1$.

We prove (4.7) by induction on j. Clearly, $m_2(n-1)$ leaves node 2 immediately after

$m_1(n-1)$. If we assume as a hypothesis that (4.7) holds for all $j = 1, 2, \ldots, J-1$, then

$$t^{(J-1)}\big(m_{k+1}(n-1)\big) = t^{(J-1)}\big(m_k(n-1)\big) + 1. \tag{4.8}$$

We will show that it also holds for $j = J$. We saw above that the first message for node $n-1$ to leave node J was $m_1(n-1)$, at time $J+1$, immediately after all messages for node n were dispatched from this node. Equation (4.8) shows that all other messages for $n-1$ arrive at node J one after the other with no delay, and consequently they will leave node J one after the other, proving thus (4.7) for $j = J$, and $i = n-1$.

(Basis 2: $i = n-2$). We prove (4.6) for $i = n-2$ since $t^{(j)}\big(m_k(i)\big)$ is given by a different formula when $i < n-1$. The proof is by induction on j. Notice that the formula is true for $j = 1$. If it holds for $j = 1, 2, \ldots, J-1$, then

$$t^{(J-1)}\big(m_1(n-2)\big) = t^{(J-1)}\big(m_{J-1}(n-1)\big) + 1,$$

that is, $m_1(n-2)$ arrives at node J right after $m_{J-1}(n-1)$. Since for $i = n-1$ (4.7) holds, it is seen that $m_J(n-1)$ has not left yet (because it has to be the last message for node $n-1$ to leave node J). Based on our scheduling rules, $m_1(n-2)$ will be the first message for node $n-2$ to leave J, immediately after $m_J(n-1)$ leaves. Hence,

$$t^{(J)}\big(m_1(n-2)\big) = t^{(J)}\big(m_J(n-1)\big) + 1,$$

concluding the induction on j and proving (4.6) for $i = n-2$.

(Hypothesis). Assume that (4.6) and (4.7) hold for all $i = n-1, n-2, \ldots, I+1$. We will show that they also hold for $i = I$, using separate inductions on j.

For (4.6), notice that it trivially holds for $j = 1$. Assuming as an induction hypothesis on j that it holds for all $j = 1, 2, \ldots, J-1$, we will show that it also holds for $j = J$. For $j = J-1$ we obtain

$$t^{(J-1)}\big(m_1(I)\big) = t^{(J-1)}\big(m_{J-1}(I+1)\big) + 1,$$

that is, $m_1(I)$ arrives at node J right after $m_{J-1}(I+1)$. But from the induction hypothesis on i, for $i = I+1$, message $m_J(I+1)$ has not left yet since from (4.7) it is the last message for node $I+1$ to leave node J. In other words, $m_1(I)$ arrives at node J before message $m_J(I+1)$ leaves. Based on our scheduling discipline, $m_1(I)$ will be the first message for node I to be scheduled after the last message for node $I+1$ ($m_J(I+1)$), proving thus (4.6) for $j = J$.

A similar induction on j can be used to prove (4.7) for $i = I$. For, since $m_2(I+1)$ leaves node 2 one time unit after $m_1(I+1)$ does, message $m_1(I)$ has arrived at node 2 before

$m_2(I)$ leaves; it will thus be scheduled to leave right before $m_2(I)$, proving thus (4.7) for $j = 2$. Assuming that the equation holds for all $j = 1, 2, \ldots, J-1$, we will prove that it also holds for $j = J$. For $j = J-1$, we get

$$t^{(J-1)}\big(m_{k+1}(I)\big) = t^{(J-1)}\big(m_k(I)\big) + 1.$$

Thus $m_{k+1}(I)$ leaves node $J-1$ right after $m_k(I)$, i.e. it arrives at node J just after $m_k(I)$ does. Because our scheduling regimen breaks ties in favor of the smallest source address, $m_{k+1}(I)$ will leave node J immediately after $m_k(I)$. ■

Equations (4.6) and (4.7) state that there is no idle time between message dispatches from node j, for all $j < i$. In particular, (4.7) guarantees that all messages destined for node i have arrived at node j at appropriate times so as to be scheduled one after the other. The last message, hence, to leave node j is message $m_j(i)$ which leaves $j-1$ time units after $m_1(i)$. Equation (4.6) serializes the phases; only after *all* messages for node $i+1$ have left node j, will messages for node i start departing from node j.

Equations (4.6) and (4.7) can be used to derive the exact expression for $t^{(j)}\big(m_k(i)\big)$ for any i, j, k, $1 \le k \le j < i < n-1$:

$$t^{(j)}\big(m_1(i)\big) \overset{(4.6)}{=} t^{(j)}\big(m_j(i+1)\big) + 1 \overset{(4.7)}{=} t^{(j)}\big(m_1(i+1)\big) + j.$$

Hence

$$t^{(j)}\big(m_1(i)\big) = t^{(j)}\big(m_1(n-1)\big) + (n-i-1)j \overset{(4.6)}{=} jn - ji + 1,$$

and from (4.7),

$$t^{(j)}\big(m_k(i)\big) = t^{(j)}\big(m_1(i)\big) + k - 1 = jn - ji + k. \tag{4.9}$$

If $i = n-1$ then (4.6) gives $t^{(j)}\big(m_1(n-1)\big) = j+1$, and (4.7) shows that $t^{(j)}\big(m_k(n-1)\big) = t^{(j)}\big(m_1(n-1)\big) + k - 1 = j + k$, which agrees with (4.9) for $i = n-1$. Thus (4.9) holds for $1 \le k \le j < i \le n-1$.

Lemma 4.2 *Phase $j \le n/2$ ends at time $T_j = jn - j^2$ at which time node j sends its last message and node $n-j+1$ receives its last message.*

Proof. Phase j ends when node j gets emptied. This occurs at time $T_j = t^{(j)}\big(m_j(j+1)\big)$ since $m_j(j+1)$ is the very last message to leave node j. This can be seen from Lemma 4.1 as follows. Any message for a destination with a larger address leaves earlier (from (4.6)). Also, from (4.7), $m_j(j+1)$ will be the last among the messages destined for node $j+1$ to leave node j.

Equation (4.9) gives

$$T_j = t^{(j)}(m_j(j+1)) = jn - j^2.$$

Also, node $n-j+1 \leq n-1$ (which is symmetric to node j) receives its last message at time $t^{(n-j)}(m_{n-j}(n-j+1))$ since, again, this is the last message to leave node $n-j$. Working exactly as above, we get

$$t^{(n-j)}(m_{n-j}(n-j+1)) = jn - j^2 = T_j.$$

The lemma also holds for the case of node n since it receives its last message ($m_1(n)$) at time $n-1$, at the end of phase 1. ■

Theorem 4.3 *The algorithm terminates at time T_{opt}.*

Proof. After each phase the number of active nodes is reduced by two as Lemma 4.2 shows. The algorithm terminates at time T, after phase $n/2$ or phase $(n-1)/2$ depending on whether n is even or odd. Using Lemma 4.2, in the first case

$$T = T_{n/2} = \frac{n^2}{4}$$

and in the second case

$$T = T_{(n-1)/2} = \frac{n^2-1}{4},$$

or, for any n, $T = \lceil (n^2-1)/4 \rceil = T_{opt}$. ■

4.3 Multiport Total Exchange in Rings

We can obtain a lower bound on total exchange for a ring R_n using (4.1). As shown in Fig. 2.4 and 4.3, nodes are labeled 0 to $n-1$ in a clockwise manner. Let $V_1 = \{0, 1, \ldots, \lfloor n/2 \rfloor - 1\}$ and $V_2 = V \setminus V_1$. In this case $C_{V_1 V_2} = 2$ since there are two edges separating the two halves: edge $(0, n-1)$ and edge $(\lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor)$. Since $|V_1||V_2| = \lfloor n/2 \rfloor (n - \lfloor n/2 \rfloor) = \lceil (n^2-1)/4 \rceil$, we obtain

$$TE(R_n) \geq \frac{\lceil (n^2-1)/4 \rceil}{2},$$

which gives easily:

$$TE(R_n) \geq \left\lceil \frac{n^2-1}{8} \right\rceil. \tag{4.10}$$

A total exchange algorithm for rings was suggested in [60] where messages are rotated around the ring, dropping one message at each stop. The algorithm was intended for rings

where communication capabilities resemble the single-port model. Indeed, under such a model the algorithm is optimal. Adopting the algorithm to the multiport model, it can be seen that it remains optimal only when the ring has an odd number of nodes. Varvarigos and Bertsekas [67] have studied a class of symmetric communication problems which they named *isotropic*, and which include total exchange. Their results were based on a correspondence between such problems and decompositions of certain matrices. They proposed general methods for hypercubes and tori, including one-dimensional tori (rings). Apart from the fact that the design of algorithms, as implied by their analysis, is quite complicated, they achieve the lower bound of (4.10) only for odd n. When n is even their results yield algorithms that require $n(n+2)/8$ steps, i.e. they are suboptimal by an additive term of $\Theta(n)$.

In the following we will present algorithms that solve the total exchange problem optimally for any ring size. The algorithms are quite simple and in addition we show that they are not unique. By the last statement we mean that there exist more than one algorithms to solve the total exchange problem optimally, providing a range of choices. Such algorithms will later be proven important for the single-port model, too.

In a ring there are two paths available between any two nodes, one in the clockwise and one in the counterclockwise direction. It is beneficial, in terms of speed, to send each message to its destination through the shortest of the two paths available. As in the case of linear arrays we will concentrate only in one direction, specifically the clockwise one, as counterclockwise messages do not interfere. The difference here is that every node has at most $\lfloor n/2 \rfloor$ messages to send in each direction. Because of some irregularities, we will discuss the cases of odd n and even n separately.

The following notation for operations on message addresses will be useful. Let x and y be integers. Recall that

$$\begin{aligned} x \oplus y &\stackrel{\text{def}}{=} x + y \bmod n \\ x \ominus y &\stackrel{\text{def}}{=} x - y \bmod n. \end{aligned}$$

We again let $m_j(i)$ stand for the message of node j destined for node i. In addition, we extend the operators $\oplus$ and $\ominus$ to messages. Specifically, $m_j(i) \oplus x$ and $m_j(i) \ominus x$ define two new messages with source and destination other than j and i as follows:

$$\begin{aligned} m_j(i) \oplus x &\stackrel{\text{def}}{=} m_{j \oplus x}(i \oplus x) \\ m_j(i) \ominus x &\stackrel{\text{def}}{=} m_{j \ominus x}(i \ominus x). \end{aligned}$$

Finally, let Q be a set of messages. The new sets of messages $Q \oplus x$ and $Q \ominus x$ are defined

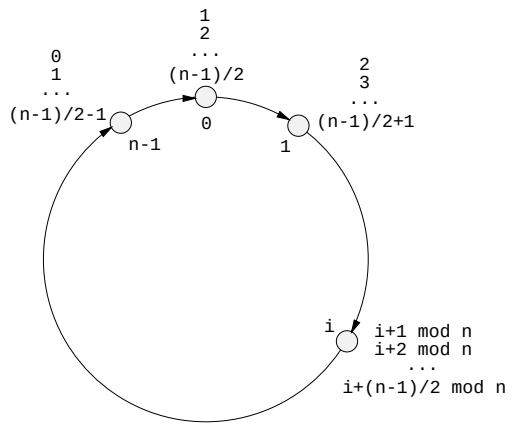

Figure 4.3. *Clockwise messages in an odd ring*

as follows:

$$Q \oplus x \stackrel{\text{def}}{=} \{m_j(i) \oplus x \mid m_j(i) \in Q\}$$
$$Q \ominus x \stackrel{\text{def}}{=} \{m_j(i) \ominus x \mid m_j(i) \in Q\}.$$

4.3.1 Odd number of nodes

In the case of odd n, every node has $(n-1)/2$ messages to send both clockwise and counterclockwise. Concentrating on the clockwise direction, node i has messages for nodes $i \oplus 1$, $i \oplus 2$, ..., $i \oplus (n-1)/2$ as shown in Fig. 4.3. As a result, the messages of node i have to travel a total distance of

$$S_i = \sum_{j=1}^{(n-1)/2} j = \frac{n^2-1}{8}.$$

In the whole ring the total distance for all clockwise messages is

$$S = \sum_{i=0}^{n-1} S_i = \frac{n(n^2-1)}{8}.$$

In a manner similar to the derivation of bound (4.4) for the single-port model, we see that total exchange will be completed when S is reduced to zero. In the case of odd n the lower bound of (4.10) is exactly $T_{opt} = (n^2-1)/8$ steps. An algorithm is thus optimal if it causes a decrease of $S/T_{opt} = n$ in total distance S at *every* step. Such a decrease in S is possible if and only if all n nodes are busy, all the time. Any algorithm achieving this is thus optimal. We show next that there exists a class of algorithms, which we will call node-invariant algorithms, that meet the above requirement.

Let $Q_i(t)$ be the message queue of node i at time $t \geq 0$. Consider a distributed algorithm $f = (f_0, f_1, \ldots, f_{n-1})$ such that f_i is the algorithm running on node i. This local algorithm f_i, at any time t, is given the message queue $Q_i(t)$ and decides that the next message to leave node i is message $f_i(Q_i(t))$.

Definition 4.1 A distributed algorithm $f = (f_0, f_1, \ldots, f_{n-1})$, where f_i is the local algorithm at processor i, is called *node-invariant* if for any non-empty set of messages Q, each local algorithm selects a message of Q, i.e. $f_i(Q) \in Q$, and

$$f_i(Q \oplus i) = f_0(Q) \oplus i,$$
$$f_i(Q \ominus i) = f_0(Q) \ominus i,$$

for all $i = 0, 1, \ldots, n-1$.

In essence, the above defines algorithms that possess a certain type of symmetry. For every message dispatched at node 0, there is a corresponding message dispatched at every other node (examples of node-invariant algorithms will follow shortly). The optimality of these algorithms is derived from the following lemma.

Lemma 4.4 *Any node-invariant algorithm guarantees that, for all* $t \geq 0$,

$$Q_i(t) = Q_0(t) \oplus i.$$

Proof. Initially $(t = 0)$ we have

$$\begin{aligned} Q_i(0) &= \{m_i(i \oplus 1), m_i(i \oplus 2), \ldots, m_i(i \oplus (n-1)/2)\} \\ &= \{m_0(1) \oplus i, m_0(2) \oplus i, \ldots, m_0((n-1)/2) \oplus i\} \\ &= \{m_0(1), m_0(2), \ldots, m_0((n-1)/2)\} \oplus i \\ &= Q_0(0) \oplus i. \end{aligned}$$

If we assume as an induction hypothesis that $Q_i(t) = Q_0(t) \oplus i$, we obtain

$$Q_i(t+1) = Q_i(t) \cup \{f_{i\ominus 1}(Q_{i\ominus 1}(t))\} \setminus \{f_i(Q_i(t))\},$$

since $f_i(Q_i(t))$ is the message leaving node i and $f_{i\ominus 1}(Q_{i\ominus 1}(t))$ is the message received by node i from its counterclockwise neighbor $(i \ominus 1)$ at time t. From the induction hypothesis,

$$Q_i(t+1) = (Q_0(t) \oplus i) \cup \{f_{i\ominus 1}(Q_{0\ominus 1}(t) \oplus i)\} \setminus \{f_i(Q_0(t) \oplus i)\}.$$

Using Definition 4.1, the above can be written as

$$\begin{aligned} Q_i(t+1) &= (Q_0(t) \oplus i) \cup \{f_{0\ominus 1}(Q_{0\ominus 1}(t)) \oplus i\} \setminus \{f_0(Q_0(t)) \oplus i\} \\ &= \Big(Q_0(t) \cup \{f_{0\ominus 1}(Q_{0\ominus 1}(t))\} \setminus \{f_0(Q_0(t))\}\Big) \oplus i \\ &= Q_0(t+1) \oplus i, \end{aligned}$$

concluding the induction. ■

Theorem 4.5 *Any node-invariant algorithm is an optimal total exchange algorithm for odd rings.*

Proof. Consider the first time that some node was observed to be idle, i.e. its queue was empty, under some node-invariant algorithm. Because the algorithm is node-invariant, Lemma 4.4 applies to show that all nodes have the same queue size, i.e. they are all empty. Consequently all nodes become idle at the same time. In our earlier discussion, we saw that any algorithm for odd rings which keeps all nodes busy all the time is optimal. Hence, this is the case for our node-invariant algorithm as well. ■

Most reasonable algorithms are node-invariant, e.g. the simple furthest-first and closest-first types of scheduling with any consistent (across the ring) tie-breaking protocol. Consider for example a closest-first algorithm $f = (f_0, f_1, \ldots, f_{n-1})$ where

f_i : *select the message with the closest destination to node i. Break ties in favor of the source in largest distance from node i,*

for all $i = 0, 1, \ldots, n-1$. Consider a set of messages

$$Q = \{m_{k_1}(i_1), m_{k_2}(i_2), \ldots, m_{k_q}(i_q)\}.$$

Notice that if node x is at distance d_x from node 0 then node $x \oplus i$ is at distance d_x from node i. In other words, any node i_j is in the same distance from node 0 as node $i_j \oplus i$ is from node i, and consequently, if node i_c is the closest destination to node 0 in Q then $i_c \oplus i$ is the closest destination to node i in $Q \oplus i$. Similarly, it is seen that if $m_{k_c}(i_c)$ is the message node 0 selects among tied messages in Q, then $m_{k_c \oplus i}(i_c \oplus i)$ is the message chosen by i from $Q \oplus i$. Hence, if $f_0(Q) = m_{k_c}(i_c)$, for some $1 \le c \le q$, then $f_i(Q \oplus i) = m_{k_c \oplus i}(i_c \oplus i)$, i.e. $f_i(Q \oplus i) = f_0(Q) \oplus i$. According to Definition 4.1 the algorithm is node invariant.

Another algorithm of interest is the *message-shift* algorithm where at every node messages join and leave the queue in a FIFO manner. In effect this is a rotation of the messages around the ring as was initially proposed in [60]. An example utilizing this algorithm is

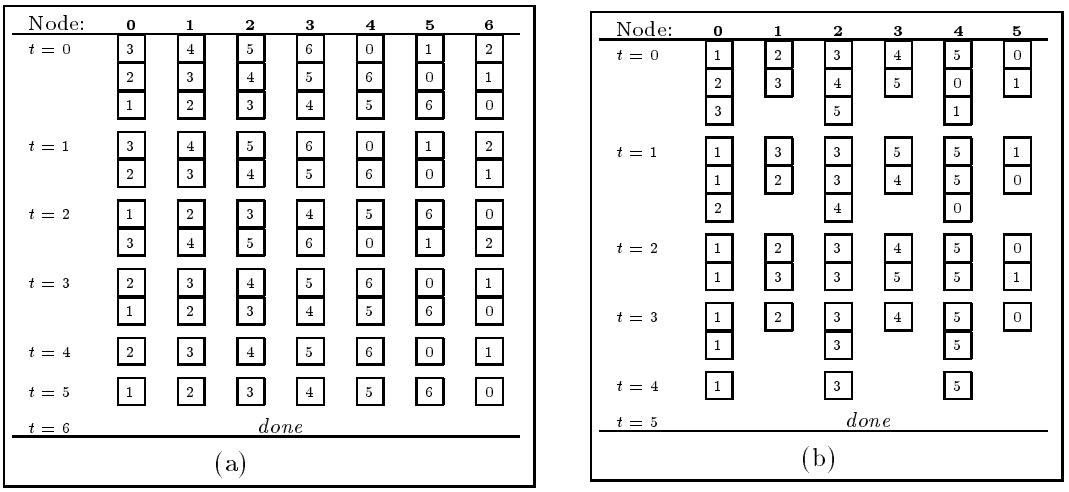

Figure 4.4. *Examples in a 7-node ring (a) and a 6-node ring (b)*

given in Fig. 4.4(a). If the message queues are considered as *ordered* sets of messages, it is a simple matter to show that Lemma 4.4 still holds, and as a result the algorithm is optimal.

It is interesting to note that although all node-invariant algorithms have the same completion time and achieve 100% utilization of the links, the closest-first type of algorithms achieve the minimum possible average packet delay [67]. On the other hand, such algorithms have a significant overhead at every step since the message queues must constantly be sorted in order to determine the message with the closest destination. The message-shift algorithm requires no such actions, allowing for simple and fast processing at every step.

4.3.2 Even number of nodes

The peculiarity of this case arises from the imbalance between clockwise and counterclockwise messages. Specifically, the furthest node from node i is node $i \oplus n/2$, so that if $m_i(i \oplus n/2)$ is scheduled for the clockwise direction, in the counterclockwise direction there is one less message to be sent. If we let every node send $n/2$ messages in the clockwise direction, then the total distance for clockwise messages is

$$S = \sum_{i=0}^{n-1} \sum_{j=1}^{n/2} j = \frac{n^2(n+2)}{8}.$$

Any node-invariant algorithm would reduce S by n units at each step, requiring time equal to $n(n+2)/8$; this is suboptimal by $\Theta(n)$ steps from the bound of (4.10). It is thus necessary to have some of the nodes send $n/2 - 1$ messages clockwise.

We show here that a slightly modified message-shift algorithm achieves the lower bound of (4.10) for even n. The modification requires that all nodes with an even address send $n/2$ messages and all nodes with an odd address send $n/2-1$ messages clockwise. In the counterclockwise direction the situation is reversed. We concentrate on the clockwise direction without loss of generality. Initially, the message queues are organized in a *furthest-first* order. The transmissions then proceed in the normal FIFO manner.

Assuming that messages leave the queue from the right end and join the queue at the left end, the algorithm can be stated as follows, for all $i = 0, 1, \ldots, n-1$:

f_i : Set

$$Q_i(0) = \begin{cases} \{m_i(i \oplus 1), m_i(i \oplus 2), \ldots, m_i(i \oplus n/2)\} & \text{if } i \text{ is even} \\ \{m_i(i \oplus 1), m_i(i \oplus 2), \ldots, m_i(i \oplus (n/2-1))\} & \text{if } i \text{ is odd.} \end{cases}$$

While queue is nonempty do:
send the rightmost message in the queue to node $i \oplus 1$ *and add the incoming message from node* $i \ominus 1$ *to the left end of the queue.*

Theorem 4.6 *The algorithm terminates at time* T_{opt}.

Proof. It is a simple matter to show that for all even i, $Q_i(t) = Q_0(t)$ and for all odd i, $Q_i(t) = Q_1(t)$, for $t \geq 0$. The proof is similar to the proof of Lemma 4.4. We can thus concentrate on nodes 0 and 1 without loss of generality. Initially,

$$\begin{aligned} Q_0(0) &= \Big\{m_0(1), m_0(2), \ldots, m_0(n/2)\Big\} \\ Q_1(0) &= \Big\{m_1(2), m_1(3), \ldots, m_1(n/2)\Big\}. \end{aligned}$$

After $2|Q_1(0)| = n-2$ steps, all messages from nodes $n-2$ and $n-1$ will have been transferred to nodes 0 and 1 respectively, with some of these messages being delivered to the appropriate destinations on their way. Hence,

$$\begin{aligned} Q_0(n-2) &= \Big\{m_{n-2}(1), m_{n-2}(2), \ldots, m_{n-2}(n/2-2)\Big\} \\ Q_1(n-2) &= \Big\{m_{n-1}(2), m_{n-1}(3), \ldots, m_{n-1}(n/2-2)\Big\}. \end{aligned}$$

In general, we shall say that "phase" j has been completed when all messages originating at nodes $n-2j$ and $n-2j+1$ have been transferred to node 0 and 1 respectively. If phase j ends at time T_j then

$$\begin{aligned} Q_0(T_j) &= \Big\{m_{n-2j}(k) \;\mid\; k = 1, 2, \ldots, n/2-2j\Big\} \\ Q_1(T_j) &= \Big\{m_{n-2j+1}(k) \;\mid\; k = 2, 3, \ldots, n/2-2j\Big\}. \end{aligned}$$

Notice that $Q_0(T_{j+1})$ and $Q_1(T_{j+1})$ will be formed after $2|Q_1(T_j)| = n - 4j - 2$ steps. Consequently,

$$T_j = \sum_{k=0}^{j-1}(n - 4k - 2) = jn - 2j^2.$$

If $n/2$ is even, then $|Q_0(T_j)|$ (hence $|Q_1(T_j)|$ as well) will be zero for $j = n/4$. Otherwise, for $j = (n-2)/4$, $|Q_0(T_j)| = 1$ and all queues become empty in the next step since the last message in Q_0 will be destined for node 1. In the first case, the algorithm terminates at time

$$T = T_{n/4} = \frac{n^2}{8}$$

and in the second case,

$$T = T_{(n-2)/4} + 1 = \frac{n^2+4}{8}.$$

In both cases, it is easily seen that $T = \lceil (n^2-1)/8 \rceil = T_{opt}$. ■

4.4 Single-port Total Exchange

In this section we extend the idea of node-invariant algorithms to total exchange under the single-port model. We necessarily deal with node symmetric networks, so that the lower bound of (4.5) is our optimality criterion

$$TE^{\text{node symmetric}}_{\text{single-port}} \geq s_G.$$

We start by showing how to determine the status of a few graphs of interest. In Chapter 5, we elaborate more on such calculations. First, we consider the *complete graph* K_n on n vertices. In such a graph every vertex is adjacent to every other vertex so that easily

$$s_{K_n} = n - 1. \tag{4.11}$$

In a ring, nodes i and $n - i$ are at distance i from node 0, for $i < \lfloor n/2 \rfloor$. If n is odd then nodes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ are at distance $\lfloor n/2 \rfloor = (n-1)/2$ from node 0 so that

$$s^{\text{odd } n}_{R_n} = \sum_{i=1}^{(n-1)/2} 2i = \frac{n^2-1}{4}.$$

If n is even the $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ coincide so there is only one node at distance $\lfloor n/2 \rfloor = n/2$. Thus

$$s^{\text{even } n}_{R_n} = \sum_{i=1}^{(n/2)-1} 2i + \frac{n}{2} = \frac{n^2}{4},$$

or, for any n,

$$s_{R_n} = \left\lceil \frac{n^2-1}{4} \right\rceil . \tag{4.12}$$

In extended rings E_n^ρ it is seen that there exist $n_i = 2\rho$ nodes at distance i from node 0, for all $i = 1, 2 \ldots, q$, where q is the largest integer such that $2\rho q \leq n-1$; q is actually the quotient of the division $(n-1)/2\rho$. If p is the remainder of this division then $n-1 = 2\rho q + p$, and there exist p more nodes at distance $q+1$ from node 0. Notice that $q = \lfloor (n-1)/2\rho \rfloor$ and $p = n-1 \bmod 2\rho$. We obtain

$$\begin{aligned} s_{E_n^\rho} &= \sum_{i=1}^{q} 2\rho i + p(q+1) \\ &= (q+1)(\rho q + p) \\ &= \left(\left\lfloor \frac{n-1}{2\rho} \right\rfloor + 1 \right) \left(n - 1 - \rho \left\lfloor \frac{n-1}{2\rho} \right\rfloor \right) . \end{aligned} \tag{4.13}$$

Equations (4.11)–(4.13) constitute lower bounds for single-port total exchange in the corresponding graphs. Rather than designing specific algorithms for specific networks we find that a whole family of graphs can be considered under the same setting and a general theory can be applied based on the idea that algorithms behave "uniformly" at every node. This family of graphs is introduced in the next section (Section 4.4.1).

Based on the observations made in Sections 4.1 and 4.3.1, if each of the n nodes is busy all the time, then (at best) n units can be subtracted from the total initial distance $\sum_{v \in V} s(v) = n s_G$ at each step, leading to a running time equal to the lower bound of (4.5). Requiring that all nodes are busy all the time is not sufficient though in a general setting. We must in addition require that at each step *all* n transmitted messages get closer to their destination. Consequently, no messages can be *derouted* at any step, meaning that every message must travel on a shortest path towards its destination. In summary, a single-port total exchange algorithm will achieve the bound of (4.4) if:

$$\text{all nodes are busy all the time, and,} \tag{4.14}$$

$$\text{every transmitted messages gets closer to its destination.} \tag{4.15}$$

Conditions (4.14)–(4.15) are sufficient but it is easy to see that they are also necessary for node symmetric graphs. More generally, they are necessary in any graph for which the average status is an integer (see (4.3)). If they are not met, then there will exist at least one step for which the total distance is not reduced by n, hence never achieving the bound of (4.4).

4.4.1 Cayley graphs

Let $V = \{v_i \mid i = 0, 1, \ldots, n-1\}$ be the node set; it will be convenient here to let subscript i take values from 0 to $n-1$.

Cayley graphs constitute a large family of node symmetric networks [8, 1]. Hypercubes belong to this family. The class also includes other important networks such as the (wrapped) *butterfly* and the *cube-connected cycles* [2, 55, 13] which are bounded-degree "approximations" of the hypercube. Connected circulant graphs (which include the complete graphs, the rings, the extended rings) are Cayley graphs, too [8]. Some elementary group-theoretic terminology is used in this section, which should be part of any text on the subject (e.g. see [9] for a comprehensive introduction).

Given a set $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_d\}$ of generators for a group $\mathcal{G}$, a Cayley graph has vertices corresponding to the elements of $\mathcal{G}$ and edges corresponding to the action of the generators. That is, if $v_i, v_j \in \mathcal{G}$, the edge (v_i, v_j) exists if and only if there is a generator $\gamma \in \Gamma$ such that $v_i \cdot \gamma = v_j$, where '$\cdot$' is the multiplicative operation in $\mathcal{G}$. There exists the restriction that the identity element of $\mathcal{G}$ does not belong to Γ (in order to avoid arcs from a node to itself) and that Γ is closed under inverses (so that the graph is in effect undirected).

4.4.2 An automorphism property of Cayley graphs

Let G be a node symmetric network with node set $V = \{v_0, v_1, \ldots, v_{n-1}\}$, and let $\Pi(G)$ be its automorphism group, i.e. the set of all automorphisms of G. Denote by $\Pi_{v_i}(G)$ the subset of $\Pi(G)$ consisting of all automorphisms that map v_0 to v_i:

$$\Pi_{v_i}(G) = \{\sigma \mid \sigma(v_0) = v_i,\ \sigma \in \Pi(G)\}.$$

Notice that $\Pi_{v_i}(G)$ is nonempty since G is node symmetric. From each set $\Pi_{v_i}(G)$ we select one automorphism σ_{v_i} and form the set

$$\Sigma(G) = \{\sigma_{v_i} \mid \sigma_{v_i} \in \Pi_{v_i}(G),\ i = 0, 1, \ldots, n-1\}.$$

In particular, we select σ_{v_0} to be the *identity* mapping. Let $\sigma\sigma'$ be the composition of mappings σ and σ'. We insist that the selected mappings have the following property: for every neighbor v_a of node v_0 and for every $i = 0, 1, \ldots, n-1$,

$$\sigma_{\sigma_{v_i}(v_a)} = \sigma_{v_i}\sigma_{v_a}. \tag{4.16}$$

In simple terms, requirement (4.16) means that if v_a is mapped, through σ_{v_i}, to some neighbor v of v_i then σ_v can be written as the composition of σ_{v_i} and σ_{v_a}. The implications of (4.16) will be seen shortly; but first we have the following result.

Lemma 4.7 *Every Cayley network has a set $\Sigma(G)$ of automorphisms that satisfy (4.16).*

Proof. Consider any Cayley graph based on a group $\mathcal{G}$, and the mapping

$$\sigma_{v_i}(v_x) = v_i \cdot v_0^{-1} \cdot v_x,$$

where v_0^{-1} is the inverse element of v_0 in $\mathcal{G}$. This mapping is easily seen to be an automorphism of the graph [1] and clearly maps v_0 to v_i. We obtain

$$\begin{aligned}
\sigma_{\sigma_{v_i}(v_a)}(v_x) &= \sigma_{v_i}(v_a) \cdot v_0^{-1} \cdot v_x \\
&= v_i \cdot v_0^{-1} \cdot v_a \cdot v_0^{-1} \cdot v_x \\
&= \sigma_{v_i}(v_a \cdot v_0^{-1} \cdot v_x) \\
&= \sigma_{v_i}(\sigma_{v_a}(v_x)),
\end{aligned}$$

which satisfies (4.16). ■

Notice that the set of automorphisms given in the proof of Lemma 4.7 may not be the only one which satisfies (4.16). Also, if the network is known, the automorphisms may have a (computationally) simpler form. As an example, consider a ring with n nodes. Node v_i is adjacent to nodes $v_{i\oplus 1}$ and $v_{i\ominus 1}$ where $\oplus$ and $\ominus$ denote addition and subtraction modulo n. A set $\Sigma(G)$ of automorphisms with the desired properties consists of the following mappings:

$$\sigma_{v_i}(v_x) = v_{i\oplus x},$$

$i = 0, 1, \ldots, n-1$. Clearly,

$$\sigma_{\sigma_{v_i}(v_a)}(v_x) = \sigma_{v_{i\oplus a}}(v_x) = v_{i\oplus a\oplus x} = \sigma_{v_i}(v_{a\oplus x}) = \sigma_{v_i}(\sigma_{v_a}(v_x)),$$

satisfying (4.16). Actually, the above mappings work for any (connected) circulant graph.

Lemma 4.8 *Let $\Sigma(G)$ be a set of automorphisms satisfying (4.16). If v_0 "picks" one of its neighbors, v_a, and every node v_i, $i = 1, 2, \ldots, n-1$, "picks" neighbor $\sigma_{v_i}(v_a)$ then* **(a)** *every node is picked by exactly one other node and* **(b)** *if v_b is the node that picks v_0 then $\sigma_{v_i}(v_b)$ is the node that picks v_i (that is, $\sigma_{\sigma_{v_i}(v_b)}(v_a) = v_i$).*

Proof.

(a) For the first part, all we have to show is that $\sigma_{v_i}(v_a) \neq \sigma_{v_j}(v_a)$ for $i \neq j$. That is, no node is picked by more than one other node. Then, since there are n "picks" in total, each of the n nodes is picked by exactly one other node. Let us say that there exists some i and some $j \neq i$ for which $\sigma_{v_i}(v_a) = \sigma_{v_j}(v_a) = v_k$, for some k. Then $\sigma_{v_k} = \sigma_{\sigma_{v_i}(v_a)}$, and from (4.16), $\sigma_{v_k} = \sigma_{v_i}\sigma_{v_a}$. Similarly, $\sigma_{v_k} = \sigma_{v_j}\sigma_{v_a}$. Consequently, $\sigma_{v_i}\sigma_{v_a} = \sigma_{v_j}\sigma_{v_a}$, or $\sigma_{v_i} = \sigma_{v_j}$, which is absurd.

(b) Let v_b be the node that picks v_0, that is $v_0 = \sigma_{v_b}(v_a)$. Since $v_i = \sigma_{v_i}(v_0)$, we obtain $v_i = \sigma_{v_i}(\sigma_{v_b}(v_a))$. From (4.16), $v_i = \sigma_{\sigma_{v_i}(v_b)}(v_a)$. This means that node $\sigma_{v_i}(v_b)$ picked v_i.

■

4.4.3 Optimal total exchange algorithms

Every node v_i in the network maintains a *message queue*, Q_{v_i}, where incoming messages from neighbors are deposited until they are scheduled for transfer to some other node. If an incoming message is destined for v_i it is assumed that it does not join the message queue but is rather forwarded to the local processor for consumption. At node v_i some local algorithm $\mathcal{A}_{v_i}$ operates in order to schedule the message transfers. Whenever there exist messages in Q_{v_i}, $\mathcal{A}_{v_i}$ is responsible for selecting the message to leave in the next time unit. Because of condition (4.15) we see that apart from the decision of which message to send next, a node must also decide to which neighbor to send it to so that the message is not derouted.

Definition 4.2 A distributed total exchange algorithm $\mathcal{A} = (\mathcal{A}_{v_0}, \mathcal{A}_{v_1}, \ldots, \mathcal{A}_{v_{n-1}})$ is a collection of local algorithms, algorithm $\mathcal{A}_{v_i}$ running on node v_i, $i = 0, 1, \ldots, n-1$. Algorithm $\mathcal{A}_{v_i}$ is written as $\mathcal{A}_{v_i} = (f_{v_i}, w_{v_i})$, where, given a message queue Q_{v_i}, f_{v_i} selects a message $f_{v_i}(Q_{v_i}) = m$ and w_{v_i} selects a neighbor $w_{v_i}(m)$ of v_i.

Now we need to configure the two parameters f_{v_i} and w_{v_i} of each local algorithm in such a way that uniformity is guaranteed. The idea is to let every node v_i select a message "corresponding" to the message selected by v_0. Intuitively we expect that condition (4.14) will be guaranteed, as in the case of rings. In general however, node v_0 may have more than one neighbor and it may be the case that more than one of them is a suitable receiver to satisfy condition (4.15). The same will hold for any other node v_i. Consequently, there must also exist a "correspondence" between the neighbor selected by v_0 and the neighbor selected by v_i. For example, in a ring we may require that whenever node v_0 selects its clockwise neighbor then every node v_i selects its own clockwise neighbor, too. In a general node symmetric network, such correspondences can be established by automorphisms. An automorphism σ_{v_i} that maps v_0 to v_i also maps all neighbors of v_0 to neighbors of v_i. If v_0 communicates with one of its neighbors, v_a, then we require that the node with which v_i communicates be node $\sigma_{v_i}(v_a)$, the image of v_a under σ_{v_i}.

We are now ready to define node-invariant algorithms. The following notations will be needed. Let $m_{v_x}(v_y)$ be the message of node v_x (source) meant for node v_y (destination).

For an automorphism $\sigma \in \Pi(G)$, let $\sigma\big(m_{v_x}(v_y)\big)$ be the message of node $\sigma(v_x)$ destined for node $\sigma(v_y)$, i.e.

$$\sigma\big(m_{v_x}(v_y)\big) \stackrel{\text{def}}{=} m_{\sigma(v_x)}\big(\sigma(v_y)\big). \tag{4.17}$$

Finally, let Q be a set of messages. We define

$$\sigma(Q) \stackrel{\text{def}}{=} \Big\{\sigma\big(m_{v_x}(v_y)\big) \;\mid\; m_{v_x}(v_y) \in Q\Big\}. \tag{4.18}$$

Definition 4.3 Let G be a node symmetric network that has a set $\Sigma(G)$ of automorphisms that satisfy (4.16). A total exchange algorithm $\mathcal{A} = (\mathcal{A}_{v_0}, \ldots, \mathcal{A}_{v_{n-1}})$ where $\mathcal{A}_{v_i} = (f_{v_i}, w_{v_i})$, $i = 0, 1, \ldots, n-1$, will be called *node-invariant* if for any message queue Q and any message m it satisfies

$$\begin{aligned} f_{v_i}\big(\sigma_{v_i}(Q)\big) &= \sigma_{v_i}\big(f_{v_0}(Q)\big) \\ w_{v_i}\big(\sigma_{v_i}(m)\big) &= \sigma_{v_i}\big(w_{v_0}(m)\big). \end{aligned}$$

Lemma 4.9 *If $Q_{v_i}(t)$ is the queue of node v_i at time t, $i = 0, 1, \ldots, n-1$, then any node-invariant algorithm guarantees that*

$$Q_{v_i}(t) = \sigma_{v_i}\big(Q_{v_0}(t)\big),$$

for all $t \geq 0$.

Proof. The proof is by induction on t. Initially ($t = 0$) we have that

$$Q_{v_0} = \{m_{v_0}(v_j) \;\mid\; j = 1, 2, \ldots, n-1\}.$$

Because automorphisms are bijections $\sigma_{v_i}(v_k) \neq \sigma_{v_i}(v_\ell)$ if $k \neq \ell$. Consequently, the set $\{\sigma_{v_i}(v_j) \;\mid\; j = 1, 2, \ldots, n-1\}$ contains all nodes of G except node v_i (since $\sigma_{v_i}(v_0) = v_i$ and $j \neq 0$). Thus the message set $S = \{m_{v_i}\big(\sigma_{v_i}(v_j)\big) \;\mid\; j = 1, 2, \ldots, n-1\}$ is the same as the set $S' = \{m_{v_i}(v_k) \;\mid\; k = 0, 1, \ldots, n-1,\ k \neq i\}$. Notice that $S' = Q_{v_i}(0)$. If we write v_i as $\sigma_{v_i}(v_0)$, we obtain

$$\begin{aligned} S &= \{m_{\sigma_{v_i}(v_0)}\big(\sigma_{v_i}(v_j)\big) \;\mid\; j = 1, 2, \ldots, n-1\} \\ &\stackrel{(4.17)}{=} \{\sigma_{v_i}\big(m_{v_0}(v_j)\big) \;\mid\; j = 1, 2, \ldots, n-1\} \\ &\stackrel{(4.18)}{=} \sigma_{v_i}\Big(\{m_{v_0}(v_j) \;\mid\; j = 1, 2, \ldots, n-1\}\Big) \\ &= \sigma_{v_i}\big(Q_{v_0}(0)\big), \end{aligned}$$

showing that $Q_{v_i}(0) = \sigma_{v_i}(Q_{v_0}(0))$.

Next, assume as an induction hypothesis that for some $t \geq 0$

$$Q_{v_i}(t) = \sigma_{v_i}(Q_{v_0}(t)). \tag{4.19}$$

For time $t+1$ we proceed as follows. Let for simplicity $m_{s(v_i)} = f_{v_i}(Q_{v_i}(t))$ and $v_{s(v_i)} = w_{v_i}(m_{s(v_i)})$. That is, $m_{s(v_i)}$ is the message selected by v_i, and $v_{s(v_i)}$ is the neighbor of v_i to which the selected message is *sent*. From (4.19) and the definition of node-invariant algorithms it is easily seen that

$$m_{s(v_i)} = \sigma_{v_i}(m_{s(v_0)}), \tag{4.20}$$

$$v_{s(v_i)} = \sigma_{v_i}(v_{s(v_0)}). \tag{4.21}$$

Now notice that $v_{s(v_0)}$ is the neighbor v_0 "picked" to send the message to. From (4.21) it is seen that Lemma 4.8 applies to show that every node receives exactly one message, and that, if $v_{r(v_0)}$ is the neighbor from which v_0 *receives* a message then

$$v_{r(v_i)} = \sigma_{v_i}(v_{r(v_0)}) \tag{4.22}$$

is the neighbor from which v_i receives its (unique) message. Moreover, if $m_{r(v_i)}$ is the message received by v_i, we obtain

$$\begin{aligned} m_{r(v_i)} &= m_{s(v_{r(v_i)})} \\ &= \sigma_{v_{r(v_i)}}(m_{s(v_0)}) \\ &= \sigma_{\sigma_{v_i}(v_{r(v_0)})}(m_{s(v_0)}) \\ &= \sigma_{v_i}\big(\sigma_{v_{r(v_0)}}(m_{s(v_0)})\big), \end{aligned}$$

and since $m_{r(v_0)} = m_{s(v_{r(v_0)})} = \sigma_{v_{r(v_0)}}(m_{s(v_0)})$,

$$m_{r(v_i)} = \sigma_{v_i}(m_{r(v_0)}). \tag{4.23}$$

To recapitulate, any node v_i selects a message $m_{s(v_i)}$ given by (4.20), sends it to some node $v_{s(v_i)}$ given by (4.21) and receives a message $m_{r(v_i)}$ given by (4.23) from some node $v_{r(v_i)}$ given by (4.22). If the destination of $m_{r(v_0)}$ is node v_0, then from (4.23) it is seen that the destination of $m_{r(v_i)}$ is node v_i. Conversely, if $m_{r(v_0)}$ is not meant for v_0 then $m_{r(v_i)}$ is not meant for v_i. In the first case at node v_0 we will have

$$Q_{v_0}(t+1) = Q_{v_0}(t) \setminus \{m_{s(v_0)}\},$$

since $m_{r(v_0)}$ does not join the queue, and in the second case,

$$Q_{v_0}(t+1) = Q_{v_0}(t) \cup \{m_{r(v_0)}\} \setminus \{m_{s(v_0)}\}, \tag{4.24}$$

where '\' is the set-theoretic difference. In the second case (the first case is treated identically), for node v_i we have

$$Q_{v_i}(t+1) = Q_{v_i}(t) \cup \{m_{r(v_i)}\} \setminus \{m_{s(v_i)}\}.$$

Using (4.19), (4.20), (4.23) and (4.24),

$$\begin{aligned} Q_{v_i}(t+1) &= \sigma_{v_i}\big(Q_{v_0}(t)\big) \cup \{\sigma_{v_i}(m_{r(v_0)})\} \setminus \{\sigma_{v_i}(m_{s(v_0)})\} \\ &= \sigma_{v_i}\Big(Q_{v_0}(t) \cup \{m_{r(v_0)}\} \setminus \{m_{s(v_0)}\}\Big) \\ &= \sigma_{v_i}\big(Q_{v_0}(t+1)\big), \end{aligned}$$

concluding the induction. ■

Lemma 4.10 *If node v_0 never deroutes a message then the same is true for every other node v_i, $i = 1, 2, \ldots, n-1$.*

Proof. If at some time t node v_0 selects message $m_{v_x}(v_y)$ out of its queue and sends it to some neighbor v_s, then any node v_i selects message $\sigma_{v_i}\big(m_{v_x}(v_y)\big)$ and sends it to neighbor $\sigma_{v_i}(v_s)$ as we have already seen (equations (4.20)–(4.21)). All we have to show is that if v_s is on a shortest path from v_0 to v_y (i.e. v_0 does not deroute the message) then $\sigma_{v_i}(v_s)$ is on a shortest path from v_i to $\sigma_{v_i}(v_y)$.

This is easy to do because automorphisms preserve distances [8]. That is, if σ is an automorphism of a graph G then $dist(v,u) = dist\big(\sigma(v), \sigma(u)\big)$ for any two vertices v and u of G. If v_0 does not deroute then $dist(v_0, v_y) = dist(v_s, v_y) + 1$. Then, we must have $dist\big(v_i = \sigma_{v_i}(v_0), \sigma_{v_i}(v_y)\big) = dist\big(\sigma_{v_i}(v_s), \sigma_{v_i}(v_y)\big) + 1$ and $\sigma_{v_i}(v_s)$ indeed lies on a shortest path from v_i to $\sigma_{v_i}(v_y)$. ■

Theorem 4.11 *Any node-invariant algorithm for which function w_0 selects shortest paths is an optimal total exchange algorithm for Cayley graphs.*

Proof. From Lemma 4.9 it is seen that all nodes have the same queue size at any step. Thus all nodes become idle (all queues are empty, hence total exchange is completed) at the same time. From Lemma 4.10 no message is derouted if w_0 selects shortest paths. Consequently, both conditions (4.14) and (4.15) are satisfied and the algorithm solves the problem optimally. ■

Summarizing, we just showed that there exists a class of algorithms, called node-invariant algorithms, which are able to solve the total exchange problem optimally in any Cayley network. In each step, every node sends one message and also receives exactly one message. Optimal algorithms were previously known only for two Cayley networks: the hypercube [5, pp. 81–83] and the star graph [50].

As in the case of rings, most reasonable algorithms such as furthest-first, closest-first, etc. schemes are valid candidates, as long as they do not stay idle when a queue contains messages and they are replicated "consistently" at all nodes in the network. In the next section, we provide a particularly simple node-invariant algorithm and we give a complete example in the context of hypercubes.

Notice that all our arguments and all our results where based on the requirement that the graph possesses a set of automorphisms that satisfy (4.16). In any network which has this property (Cayley graphs do), node invariant algorithms can be defined and utilized for the total exchange problem. We do not know, however, whether (4.16) is a property or not of every node symmetric network.

4.4.4 A simple node-invariant algorithm

Assume that we have an algorithm $\mathcal{W}$ which takes a message, looks at its destination and picks a neighbor of v_0 which lies on a shortest path from v_0 to the destination of the message. We will use the same algorithm (with different inputs however) at every node. It is always possible to construct such an algorithm $\mathcal{W}$ for any network, e.g. using a table look-up procedure. More efficient schemes of course are possible if the structure of the network is known. For example, in a ring R_n we can have

$$\mathcal{W}(m_{v_x}(v_y)) = \begin{cases} v_1 & \text{if } y \leq n/2 \\ v_{n-1} & \text{otherwise} \end{cases}$$

(nodes v_1 and v_{n-1} are the two neighbors of node v_0).

Let us treat a message queue as a set of messages that behaves as a FIFO queue. At node v_0, we initially sort destinations in any desired order. For instance,

$$Q_{v_0}(0) = \{m_{v_0}(v_1), m_{v_0}(v_2), \ldots, m_{v_0}(v_{n-1})\}.$$

Suppose that the right end is the head of the FIFO queue and the left end is its tail. Departing messages will leave from the head of the queue. Arriving messages will join at the tail of the queue as long as they are not destined for the current node; otherwise they are immediately forwarded to the local processor. We have to guarantee that initially $Q_{v_i}(0)$ is

$\mathcal{A}_{v_i}$: $(i = 0, 1, \ldots, n-1)$

At $t = 0$ set

$$Q_{v_i} = \Big\{ m_{v_i}(\sigma_{v_i}(v_1)), m_{v_i}(\sigma_{v_i}(v_2)), \ldots, m_{v_i}(\sigma_{v_i}(v_{n-1})) \Big\},$$

and let

$f_{v_i}(Q_{v_i})$: select the message at the head of the queue Q_{v_i},

$w_{v_i}(m)$: if $m = f_{v_i}(Q_{v_i})$, select neighbor $\sigma_{v_i}\Big(\mathcal{W}(\sigma_{v_i}^{-1}(m))\Big)$,

Figure 4.5. *An optimal single-port total exchange algorithm for Cayley networks. The queues are FIFO. Messages join at the left end and depart from the right end of the queue.*

equal to $\sigma_{v_i}(Q_{v_0}(0))$, so we let

$$Q_{v_i}(0) = \{m_{v_i}(\sigma_{v_i}(v_1)), m_{v_i}(\sigma_{v_i}(v_2)), \ldots, m_{v_i}(\sigma_{v_i}(v_{n-1}))\}.$$

The local algorithm $\mathcal{A}_{v_i} = (f_{v_i}, w_{v_i})$ is defined as follows:

$$f_{v_i}(Q): \quad \text{select the message at the head of the queue } Q.$$

It is trivial to see that $f_{v_i}(\sigma_{v_i}(Q)) = \sigma_{v_i}(f_{v_0}(Q))$: if m is the message at the head of Q then $\sigma_{v_i}(m)$ is obviously the message at the head of $\sigma_{v_i}(Q)$. Since $m = f_{v_0}(Q)$ and $\sigma_{v_i}(m) = f_{v_i}(\sigma_{v_i}(Q))$, it is derived that $\sigma_{v_i}(f_{v_0}(Q)) = f_{v_i}(\sigma_{v_i}(Q))$.

The only parameter remaining is w_{v_i}. Let σ^{-1} be the inverse mapping of $\sigma \in \Pi(G)$. The existence and the uniqueness of σ^{-1} is guaranteed by the fact the $\Pi(G)$ is a group [9, pp. 300–301]. Given $\mathcal{W}$ we define

$$w_{v_i}(m): \quad \text{for message } m \text{ select neighbor } \sigma_{v_i}\Big(\mathcal{W}(\sigma_{v_i}^{-1}(m))\Big).$$

We only have to show that $w_{v_i}(\sigma_{v_i}(m)) = \sigma_{v_i}(w_{v_0}(m))$, for any message m. Notice that σ_{v_0} is taken to be the identity mapping so that w_{v_0} is actually the same as $\mathcal{W}$. Thus we have to show that $w_{v_i}(\sigma_{v_i}(m)) = \sigma_{v_i}(\mathcal{W}(m))$. Indeed, from the description of w_{v_i} above, we have

$$w_{v_i}(\sigma_{v_i}(m)) = \sigma_{v_i}\Big(\mathcal{W}(\sigma_{v_i}^{-1}(\sigma_{v_i}(m)))\Big) = \sigma_{v_i}(\mathcal{W}(m)),$$

since $\sigma_{v_i}^{-1}\sigma_{v_i}$ is the identity.

In summary, the algorithm shown in Fig. 4.5 is, based on Definition 4.3, node-invariant. Therefore, it is an optimal total exchange algorithm for any Cayley network, according to Theorem 4.11.

$\mathcal{A}_i$: $(i = 0, 1, \ldots, 2^d - 1)$

At $t = 0$ set

$$Q_i = \Big\{ m_i(i \oplus_b 1), m_i(i \oplus_b 2), \ldots, m_i(i \oplus_b (2^d - 1)) \Big\}.$$

At any step $t \geq 0$,

- select the message at the head of Q_i (say $m_x(y)$)
- send it to node $i \oplus_b 2^k$ where k is the leftmost non-zero bit position of $i \oplus_b y$.

Figure 4.6. *An optimal single-port total exchange algorithm for d-dimensional hypercubes. The standard e-cube routing paths are followed at every transmission.*

4.4.5 An example in hypercubes

To illustrate the theory developed in the previous sections we construct an algorithm for hypercubes, based on the algorithm in Fig. 4.5. An optimal algorithm was given in [5, pp. 81–83] but it is not in explicit form, and it is based on a rather involved algorithm for the multiport model.

Let $\oplus$ be the exclusive-or (addition modulo 2) operation. If the binary representation of x is $(x_{d-1}, \ldots, x_1, x_0)$ then the bitwise exclusive-or operation, $\oplus_b$, is defined as

$$x \oplus_b y = (x_{d-1} \oplus y_{d-1}, \ldots, x_1 \oplus y_1, x_0 \oplus y_0).$$

Dropping 'v' from the name of node v_i, a hypercube Q_d has node set $V = \{0, 1, \ldots, 2^d - 1\}$. As we already know, a node i has neighbors $i \oplus_b 2^0$, $i \oplus_b 2^1$, $\ldots$, $i \oplus_b 2^{d-1}$, i.e. nodes whose binary representation differs in exactly one bit position from the binary representation of i. The following is an automorphism of the hypercube [40] that maps node 0 to node i:

$$\sigma_i(x) = i \oplus_b x. \tag{4.25}$$

Because of the associativity of the exclusive-or, it is seen that

$$\sigma_{\sigma_i(a)}(x) = i \oplus_b a \oplus_b x = \sigma_i(\sigma_a(x)),$$

for any node a, so that the set of automorphisms given by (4.25) for $i = 0, 1, \ldots, 2^d - 1$ satisfies (4.16). Because $i \oplus_b i = 0$, it is seen that $\sigma_i^{-1} = \sigma_i$. Finally, it is known that if in the binary representation of y, y_k equals one for some k, then neighbor 2^k of node 0 lies on a shortest path from 0 to y, that is $\mathcal{W}(m_x(y)) = 2^k$. Usually, k is selected to be the leftmost non-zero bit position of y in order to comply with the standard e-cube routing [63, 16]. Consequently, the algorithm of the last section takes the simple form shown in Fig. 4.6.

Chapter 5

Total Exchange in Multidimensional Networks

5.1 Introduction

Total exchange algorithms that have appeared so far have been topology-specific, that is, they were designed specifically for a particular network. Algorithms for hypercubes are not directly applicable to tori for example. These two networks have actually monopolized the research interests of many authors. Under our communication model the major portion of the work is centered around hypercubes [59, 36, 4, 37]. Tori were considered in [67] where the authors derived a relationship between total exchange algorithms and matrix decompositions. No explicit algorithms were given however and in addition only specific types of tori (the class of k-ary n-cubes) were examined. Finally the works in [33, 11, 61, 39, 64] deal with circuit switched or wormhole routed hypercubes and tori under a variety of communication models.

Multidimensional networks have been accepted as efficient interconnection schemes for multiprocessors as a result of their nice graph-theoretic properties. In addition to hypercubes and tori though, other interesting multidimensional networks have been proposed and implemented, including meshes, hypercycles [20] and generalized hypercubes [7] to name a few. No network has been proven optimal for every task and each one can exhibit superior behavior than the others in certain situations.

Our purpose in this chapter is to study the total exchange problem in any arbitrary multidimensional network. We seek to show that total exchange algorithms for these networks can be synthesized out of total exchange algorithms for the individual dimensions. We have witnessed such decompositions of problems or quantities throughout this thesis. For example, formula (2.2) determines the eccentricity of a vertex based only on the ec-

centricities of its coordinates in the corresponding dimensions. The broadcasting problem under the multiport model was solved in Section 3.1 by constructing a spanning tree which is determined solely from spanning trees of each dimension.

Multidimensional networks are not simple networks (at least visually) and calculations or algorithms related to them can be very complex. Breaking a calculation or an algorithm to parts applicable to only one dimension provides major simplifications. We show that this is also possible for the total exchange problem. We develop a general algorithm that under the single-port model can solve the problem in any multidimensional network given that we have devised total exchange algorithms for each of the dimensions. We determine the time requirements of the algorithm and provide conditions under which optimality is achieved. It is seen that these conditions are met for many interesting networks and consequently the total exchange problem can be optimally solved with the given algorithm. In the course of our analysis the status (or total distance) of vertices is required. We derive a novel formula for calculating the status of a vertex in multidimensional networks in the next section. For the multiport model the algorithm is modified in order to improve its time requirements. We again provide conditions under which this extension is optimal and we apply the algorithm to networks with certain characteristics.

In conclusion, in this chapter we present solutions to the total exchange problem for an entire class of graphs. The problem is solved by decomposing it to the simpler problems of performing total exchange in each dimension. As a byproduct of the analysis, it is seen that under the single-port model it is possible to perform total exchange optimally in many networks of interest including hypercubes, and general tori. In certain networks (homogeneous ones) the algorithm also behaves optimally under the multiport model.

5.2 Status (Total Distance) in Multidimensional Networks

We recall that the status (or total distance) $s(v)$ of a vertex v is the sum of the distances from v to all the other nodes in the network. If n is the number of nodes, the quantity $s(v)/(n-1)$ is known as the *average distance* of node v, giving the average number of links that have to be traversed by a message departing from v. It is an important performance measure of the network since under uniform addressing distributions, it is directly linked with the average delay a message experiences before reaching its destination [58, 57].

Calculating $s(v)$ or equivalently each term n_i, $i = 1, 2, \ldots, e(v)$, in (4.2) for not-so-simple networks is usually a challenging combinatorial problem that most often than not does not yield usable results (if any). We calculate the status of vertices in two graphs in order to

show the difficulty associated with the problem.

To calculate the status of a vertex in a hypercube Q_d we observe that a node is at distance i from node 0 if it has i ones in its binary address. Since there are $\binom{d}{i}$ ways of placing i ones in d bit positions, there exist $n_i = \binom{d}{i}$ nodes at distance i from node 0. The status of node 0 (or any other node) is then seen to be equal to $d2^{d-1}$.

Generalized hypercubes are products of complete graphs. A complete graph K_n has all its n vertices adjacent to each other. In the graph $K_{m_1} \times K_{m_2} \times \cdots \times K_{m_k}$ the status of a vertex can be calculated as follows. In each dimension every node is at distance one from every other node. Thus, there exist $m_j - 1$ nodes which differ in the jth coordinate from some node v. In a manner analogous to hypercube calculations, we can find that [7]

$$n_i = \sum (m_{j_1} - 1) \cdots (m_{j_i} - 1)$$

where $j_1, \ldots, j_i \in \{1, 2, \ldots, k\}$, $j_1 \neq j_2 \neq \cdots \neq j_i$, and the sum includes $\binom{k}{i}$ such terms. If $m_j = m$ for all dimensions j, we may derive n_i easily but in any other case the argument fails. It should be apparent that the situation is worse in any other multidimensional network since hypercubes and generalized hypercubes consist of the simplest dimensions possible.

Theorem 5.1 *Let $G = G_1 \times G_2 \times \cdots \times G_k$. If $s_i(v_i)$ is the status of v_i in G_i, $i = 1, 2, \ldots, k$, then the status of $v = (v_1, v_2, \ldots, v_k)$ in G is*

$$s(v) = n \sum_{i=1}^{k} \frac{s_i(v_i)}{|V_i|},$$

where $n = |V_1||V_2| \cdots |V_k|$ is the number of nodes in G.

Proof. We are not going to utilize (4.2) but rather the following equivalent formula which is derived from the definition of the status of v,

$$s(v) = \sum_{u \in G} dist(v, u), \tag{5.1}$$

where $dist(v, u)$ is the distance between v and u. Hence, the status of v_i in G_i can be written as

$$s_i(v_i) = \sum_{u_i \in G_i} dist(v_i, u_i). \tag{5.2}$$

We know that in a multidimensional network the distance between two vertices is equal to the sum of distances between the corresponding coordinates (Eq. (2.3)). Consequently,

from (5.1) we obtain

$$\begin{aligned}
s(v) &= \sum_{(u_1,\ldots,u_k)\in G} dist\big((v_1,\ldots,v_k),(u_1,\ldots,u_k)\big) \\
&= \sum_{u_1\in G_1}\cdots\sum_{u_k\in G_k}\sum_{i=1}^{k} dist(v_i,u_i) \\
&= \sum_{i=1}^{k}\Big\{\sum_{u_1\in G_1}\cdots\sum_{u_{i-1}\in G_{i-1}}\sum_{u_{i+1}\in G_{i+1}}\cdots\sum_{u_k\in G_k}\sum_{u_i\in G_i} dist(v_i,u_i)\Big\} \\
&\overset{(5.2)}{=} \sum_{i=1}^{k}\Big\{\sum_{u_1\in G_1}\cdots\sum_{u_{i-1}\in G_{i-1}}\sum_{u_{i+1}\in G_{i+1}}\cdots\sum_{u_k\in G_k} s_i(v_i)\Big\} \\
&= \sum_{i=1}^{k}\Big\{\frac{n}{|V_i|}s_i(v_i)\Big\},
\end{aligned}$$

as claimed. ■

The status of a vertex in a multidimensional graph is thus equal to the sum of the status of its coordinates, weighted by a factor $n/|V_i|$ for the corresponding dimension. Theorem 5.1 can be used now to calculate the total or the average distance of vertices in many graphs for which no closed-form formula has appeared up to now. As an example, generalized hypercubes $K_{m_1}\times\cdots\times K_{m_k}$ have $|V_i| = m_i$, $n = m_1 m_2\cdots m_k$, and nodes in dimension i have status $s_i = m_i - 1$, as (4.11) shows. Consequently, the average distance of any vertex in the graph is given by

$$\frac{n}{n-1}\sum_{i=1}^{k}\frac{m_i-1}{m_i} = \frac{n}{n-1}\left(k - \sum_{i=1}^{k}\frac{1}{m_i}\right).$$

In the context of the total exchange problem we are interested in the average status of the nodes in the network. Let $AS(G_i)$ be the average status of G_i, defined as $AS(G_i) = \sum_{v_i\in G_i} s_i(v_i)/|V_i|$ (see (4.3)). We have the following corollary.

Corollary 5.2 *Let $G = G_1\times G_2\times\cdots\times G_k$. If $AS(G_i)$ is the average status of G_i, $i = 1,2,\ldots,k$, then the average status of G is given by*

$$AS(G) = n\sum_{i=1}^{k}\frac{AS(G_i)}{|V_i|}.$$

Proof. From Theorem 5.1 we obtain

$$\sum_{v\in G} s(v) = n\sum_{(v_1,\ldots,v_k)\in G}\sum_{i=1}^{k}\frac{s_i(v_i)}{|V_i|}$$

$$
\begin{aligned}
&= n \sum_{v_1 \in G_1} \cdots \sum_{v_k \in G_k} \sum_{i=1}^{k} \frac{s_i(v_i)}{|V_i|} \\
&= n \sum_{i=1}^{k} \sum_{v_1 \in G_1} \cdots \sum_{v_{i-1} \in G_{i-1}} \sum_{v_{i+1} \in G_{i+1}} \cdots \sum_{v_k \in G_k} \sum_{v_i \in G_i} \frac{s_i(v_i)}{|V_i|} \\
&= n \sum_{i=1}^{k} \sum_{v_1 \in G_1} \cdots \sum_{v_{i-1} \in G_{i-1}} \sum_{v_{i+1} \in G_{i+1}} \cdots \sum_{v_k \in G_k} AS(G_i) \\
&= n \sum_{i=1}^{k} \frac{n}{|V_i|} AS(G_i),
\end{aligned}
$$

which, divided by n, gives the required result. ■

5.3 Total Exchange Under the Single-port Model

Let $G = A \times B$. A k-dimensional network $G_1 \times \cdots \times G_k$ can also be expressed as the product of two graphs by taking $A = G_1 \times \cdots \times G_{k-1}$ and $B = G_k$, so we may consider two dimensions without loss of generality. Let $A = (V_A, E_A)$, $B = (V_B, E_B)$, $G = (V, E)$, $n_1 = |V_A|$, $n_2 = |V_B|$ and $n = n_1 n_2$. Finally, let

$$
\begin{aligned}
V_A &= \{v_i \mid i = 1, 2, \ldots, n_1\} \\
V_B &= \{u_i \mid i = 1, 2, \ldots, n_2\}.
\end{aligned}
$$

Graph G consists of n_2 (interconnected) copies of V_A. Let A_j be the jth copy of A with node set $(*, u_j)$, where $*$ takes all values in V_A. Similarly, G can be viewed as n_1 copies of B, and we let B_i be the ith copy of B with node set $(v_i, *)$. An example is shown in Fig. 5.1.

We will develop the basic idea behind our algorithm through the example in Fig. 5.1. Consider the top node of A_1. This node belongs to A_1 as well as B_1. All nodes in A_1 have, among other messages, messages destined for the rest of the nodes in A_1. These messages can be distributed by performing a total exchange within A_1. In addition, nodes in A_1 have messages for all nodes in A_2, A_3 and A_4. Somehow, these messages have to travel to their appropriate destinations. What we will do is the following: all messages of the top node of A_1 meant for the nodes in A_2 will be transferred to the top node of A_2. All messages of the middle node of A_1 destined for the nodes in A_2 will be transferred to the middle node of A_2. Similar will be the case for the bottom node of A_1. Once all these messages have arrived in A_2, the only thing remaining is to perform a total exchange within A_2 and all these messages will be distributed to the correct destinations.

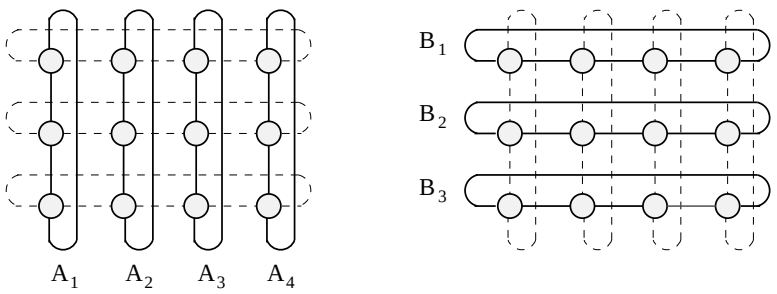

Figure 5.1. *A* 4×3 *torus as (a) four copies of a three-node ring or (b) three copies of a four-node ring*

Next, nodes of A_1 have to transfer their messages meant for A_3 to nodes of A_3. The procedure will be identical to the procedure we followed for messages meant for A_2. Finally, the remaining messages in A_1 are destined for A_4 and one more repetition of the above procedure completes the task. Notice that what we did for messages originating at nodes of A_1 has to be done also for messages originating at the other copies of A, i.e. A_2, A_3 and A_4. We are now ready to formalize our arguments.

We are going to adopt a message notation similar to that of the previous chapter: $m_{(v_i,u_j)}(v_k,u_l)$ will denote the message of node (v_i,u_j) destined for node (v_k,u_l). We will also introduce the '$*$' symbol to denote a corresponding set of messages. For example, $m_{(v_i,u_j)}(*,u_l)$ denotes all messages of node (v_i,u_j) destined for the nodes of A_l, and $m_{(v_i,*)}(v_k,u_l)$ denotes all messages of B_i destined for node (v_k,u_l). Similarly, $m_{(v_i,u_j)}(*,*)$ denotes all messages of (v_i,u_j). Notice that this last set normally includes $m_{(v_i,u_j)}(v_i,u_j)$ since $(*,*)$ covers all nodes. Since no node sends messages to itself, it is always implied that *from any set of messages, we have removed every message whose source and destination are the same.*

Consider the set of messages $m_{(*,*)}(*,*)$. This set represents our total exchange problem: every node has one message for every other node. Next consider the set $m_{(*,u_j)}(*,u_j)$. This is the set of messages of nodes in A_j destined for the other nodes in A_j: they can be distributed by a total exchange operation within A_j. Finally, consider the set $m_{(v_i,u_j)}(*,u_k)$ of node (v_i,u_j) meant for the nodes of A_k. This set will be transferred to node (v_i,u_k). Thus, after such transfers, node (v_1,u_k) will have received $m_{(v_1,u_j)}(*,u_k)$, node (v_2,u_k) will have received $m_{(v_2,u_j)}(*,u_k)$, and so on. Notice that every node in A_k will have received messages meant for every node in A_k: these messages clearly can be distributed to the appropriate

```
1  For every i = 1, 2, ..., n1
2      For every j = 1, 2, ..., n2
3          For every k = 1, 2, ..., n2, k ≠ i
4              Transfer messages m_(v_i,u_j)(*, u_k) to node (v_i, u_k);
5  For every k = 1, 2, ..., n2
6      Do in parallel for all A_j, j = 1, 2, ..., n2
7          In A_j perform Total Exchange with node (v_i, u_j)
           sending messages m_(v_i,u_k)(*, u_j);
```

Figure 5.2. *Algorithm A1*

destinations through a total exchange operation within A_k.

To recapitulate, we can solve the total exchange problem in $G = A \times B$ using Algorithm A1 shown in Fig. 5.2. First we perform *all* the transfers we described above and then we perform the total exchanges within each A_j. The transfers correspond to lines 1–4 in Algorithm A1. After they are completed, every node (v_i, u_j), for every i, j, will have received all messages meant for the jth copy of A originating at nodes (v_i, u_k), $k = 1, 2, \ldots, n_2$, i.e. all messages $m_{(v_i,u_k)}(*, u_j)$. Lines 5–7 of the algorithm distribute these messages to the correct vertices of A_j in n_2 rounds. In the kth round a total exchange is performed and the exchanged messages have originated from A_k.

Algorithm A1 solves the total exchange problem but lines 1–4 do not show how the transfer of messages is exactly implemented. First of all, there may exist path collisions between transfers from (v_i, u_j) to (v_i, u_k) and transfers from $(v_{i'}, u_j)$ to $(v_{i'}, u_k)$, $i \neq i'$, if we try to do them simultaneously. Let us consider again the example in Fig. 5.1. At some point all nodes in A_1 want to transfer their messages, say, for nodes in A_4. We make the observation that these transfers can indeed be done in parallel. That is, the top node of A_1 can transfer its messages to the top node in A_4, the middle node of A_1 can transfer its own messages to the middle node of A_4 and so on, without any interference between them. The trick is to use only paths in the second dimension (B). That is, all the transfers of the top node of A_1 use links in B_1, all transfers from the bottom node of A_1 use links in B_3, etc.

Consequently, a straightforward way of parallelizing line 1 is the following: when transferring messages from (v_i, u_j) to (v_i, u_k) we only allow use of links in the second dimension. In other words, the allowable paths $(v_i, u_j) \rightarrow (v_i, u_k)$ involve only nodes $(v_i, *)$ of B_i. Then if $v_{i'} \neq v_i$, paths $(v_i, u_j) \rightarrow (v_i, u_k)$ and $(v_{i'}, u_j) \rightarrow (v_{i'}, u_k)$ have no node in common. Consequently, lines 1–4 can be rewritten in the improved form:

	For A_1	$\cdots$	For A_j	$\cdots$	For A_k	$\cdots$	For A_{n_2}
R_1	$m_s(v_1, u_1)$	$\cdots$	$m_s(v_1, u_j)$	$\cdots$	$m_s(v_1, u_k)$	$\cdots$	$m_s(v_1, u_{n_2})$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
R_{i-1}	$m_s(v_{i-1}, u_1)$	$\cdots$	$m_s(v_{i-1}, u_j)$	$\cdots$	$m_s(v_{i-1}, u_k)$	$\cdots$	$m_s(v_{i-1}, u_{n_2})$
R_i	$m_s(v_i, u_1)$	$\cdots$	—	$\cdots$	$m_s(v_i, u_k)$	$\cdots$	$m_s(v_i, u_{n_2})$
R_{i+1}	$m_s(v_{i+1}, u_1)$	$\cdots$	$m_s(v_{i+1}, u_j)$	$\cdots$	$m_s(v_{i+1}, u_k)$	$\cdots$	$m_s(v_{i+1}, u_{n_2})$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
R_{n_1}	$m_s(v_{n_1}, u_1)$	$\cdots$	$m_s(v_{n_1}, u_j)$	$\cdots$	$m_s(v_{n_1}, u_k)$	$\cdots$	$m_s(v_{n_1}, u_{n_2})$

Table 5.1. *Messages to be transferred from node $s = (v_i, u_j)$. The jth column is actually unused since messages of (v_i, u_j) for A_j are not to be transferred to any other copy of A.*

```
1  Do in parallel for all v_i ∈ V_A (i = 1, 2, ..., n_1)
2      For every j = 1, 2, ..., n_2
3          For every k = 1, 2, ..., n_2, k ≠ i
4              Transfer messages m_(v_i,u_j)(*, u_k) to node (v_i, u_k) using links in B_i;
```

We may still improve matters by further parallelizing lines 1–3. Within B_i we need to transfer messages $m_{(v_i,u_j)}(*, u_k)$ from *every* vertex u_j to *every* other vertex u_k. In Table 5.1 we list the messages to be transferred by some vertex (v_i, u_j) of A_j. Notice that we do not have to transfer messages meant for A_j anywhere, so the jth column of the table is actually unused (it will only be used for a total exchange within A_j). Column k contains all messages of (v_i, u_j) meant for A_k, to be transferred first to node (v_i, u_k).

Instead of transferring the messages column by column (i.e. transfer all messages in column 1 to A_1, then all messages in column 2 to A_2, etc.) we transfer them horizontally (row by row). The batch R_r of messages in row r contains all messages $m_{(v_i,u_j)}(v_r, *)$. We will transfer all of them, except of course for $m_{(v_i,u_j)}(v_r, u_j)$ in column j which is meant for a node of A_j. Let us consider again the network in Fig. 5.1 and assume that the bottom nodes of A_1, A_2, A_3 and A_4 want to transfer their first batch, R_1. The batch of the bottom node of A_1 contains one message for each of the bottom nodes of A_2, A_3 and A_4. Similarly, batch R_1 for the bottom node of A_2 contains one message for the other three nodes in question. It should be immediately clear that these messages constitute an instance of the total exchange problem in B_1: every node has one message for every other node in B_1.

In general, when every node (v_i, u_1), (v_i, u_2), ..., (v_i, u_{n_2}) in B_i transfers its own batch R_r of Table 5.1, a total exchange within B_i can distribute the messages appropriately. Consequently, all rows of Table 5.1 of every node will be transferred where they should by performing n_1 total exchanges in B_i: at the rth exchange all nodes $(v_i, *)$ transfer their rth

```
1  For r = 1, 2, ..., n1
2      Do in parallel for all B_i, i = 1, 2, ..., n1
3          In B_i perform Total Exchange with node (v_i, u_j)
           sending messages m_(v_i,u_j)(v_r, *), j = 1, 2, ..., n2;
4  For every k = 1, 2, ..., n2
5      Do in parallel for all A_j, j = 1, 2, ..., n2
6          In A_j perform Total Exchange with node (v_i, u_j)
           sending messages m_(v_i,u_k)(*, u_j), i = 1, 2, ..., n1;
```

Figure 5.3. *Algorithm A2 for the single-port model*

batch of messages (rth row of the corresponding tables).

Based on the above discussion, and recalling that transfers within B_i do not interfere with transfers within $B_{i'}$, $i' \neq i$, we may express our total exchange algorithm in its final form, Algorithm A2, appearing in Fig. 5.3. Algorithm A2 is a general solution to the total exchange problem for any multidimensional network. If the network has $k > 2$ dimensions, $G = G_1 \times \cdots \times G_k$, Algorithm A2 can be used recursively, by taking $A = G_1 \times \cdots \times G_{k-1}$ and $B = G_k$. The total exchanges in A_j (lines 4–6) can be performed by invoking the algorithm with $A = G_1 \times \cdots \times G_{k-2}$ and $B = G_{k-1}$ and so forth.

The algorithm is in a highly desirable form: it only utilizes total exchange algorithms for each of the dimensions. The problem of total exchange in a complex network is now reduced to the simpler problem of devising total exchange algorithms for single dimensions. For example, we are in a position to systematically construct algorithms for tori, based on algorithms we presented in Chapter 4 for rings.

We now proceed to determine the time requirements of the algorithm and the conditions under which it behaves optimally.

5.3.1 Optimality conditions

It is not very hard to calculate the time required for Algorithm A2. This is because it is written in a form suitable for the single-port model: every node participates in one total exchange operation at a time. When each total exchange is performed under the single-port model, in effect no node sends more than one message at a time. If in addition each total exchange does not allow nodes to receive more than one message at each step, then the same is clearly true for Algorithm A2.

Theorem 5.3 *If single-port total exchange algorithms for graphs A and B take T_A and T_B steps correspondingly then Algorithm A2 for $G = A \times B$ requires*

$$T = n_1 T_B + n_2 T_A$$

time units.

Proof. The result is straightforward: lines 1–3 perform n_1 total exchanges within B_i (for all $i = 1, 2, \ldots, n_1$ in parallel), each requiring T_B steps. Similarly, lines 4–6 perform n_2 total exchanges within A_j (for all $j = 1, 2, \ldots, n_2$ in parallel), each requiring T_A steps. ■

Corollary 5.4 *If $G = G_1 \times G_2 \times \cdots \times G_k$ and a single-port total exchange algorithm for G_i takes T_i time units, $i = 1, 2, \ldots, k$, total exchange in G under the single-port model can be performed in*

$$T = n \sum_{i=1}^{k} \frac{T_i}{|V_i|}$$

steps, where $n = |V_1||V_2| \cdots |V_k|$.

Proof. The proof is by induction. If we only have one dimension then the corollary is trivially true. Assume as an induction hypothesis that it holds for up to $k-1$ dimensions. Then we must have

$$T' = n' \sum_{i=1}^{k-1} \frac{T_i}{|V_i|},$$

where T' is the time needed for total exchange in $G' = G_1 \times \cdots \times G_{k-1}$ and $n' = |V_1| \cdots |V_{k-1}| = n/|V_k|$. If we let $A = G'$ and $B = G_k$, $n_1 = n'$ and $n_2 = |V_k|$, Theorem 5.3 gives

$$\begin{aligned} T &= n' T_k + |V_k| T' \\ &= \frac{n}{|V_k|} T_k + |V_k| \frac{n}{|V_k|} \sum_{i=1}^{k-1} \frac{T_i}{|V_i|} \\ &= n \sum_{i=1}^{k} \frac{T_i}{|V_i|}, \end{aligned}$$

as required. ■

Theorem 5.5 *If single-port total exchange for every dimension $i = 1, 2, \ldots, k$ of $G = G_1 \times G_2 \times \cdots \times G_k$ can be performed in time equal to the lower bound of (4.4) then the same is true for G.*

Proof. If in G_i total exchange can be performed in time equal to the lower bound of (4.4) then $T_i = AS(G_i)$. From Corollary 5.4, we must have

$$T = n \sum_{i=1}^{k} \frac{AS(G_i)}{|V_i|},$$

which, combined with Corollary 5.2, shows that $T = AS(G)$ and the algorithm is thus optimal. ■

Based on the last theorem and the analysis in Chapter 4 we immediately see that in products of Cayley graphs total exchange is performed optimally with Algorithm A2. For example, since rings, extended rings and complete graphs are Cayley networks, it is derived that Algorithm A2 is an optimal total exchange algorithm for tori (products of rings), hypercubes, k-ary n-cubes (subclasses of tori), hypercycles (products of extended rings) and generalized hypercubes (products of complete graphs).

5.4 Extension to the Multiport Model

In this section, we modify Algorithm A2 to work better under the multiport model. In its present form, Algorithm A2 is not particularly efficient under this model. This is because lines 4–6 are executed after lines 1–3 have finished. During execution of lines 1–3, only edges of the second dimension (B) are used while lines 4–6 use only edges of the first dimension (A). In the multiport model, we try to keep as many edges busy as possible and the behavior of Algorithm A2 does not contribute to that effect. We seek, consequently, to transfer messages in both dimensions simultaneously. In other words we reconstruct the algorithm such that lines 1–3 overlap in time as much as possible with lines 4–6.

The theory we present here applies to *homogeneous* networks. A multidimensional network is called homogeneous when all its dimensions are identical. Thus, $G = H \times H \times \cdots \times H = H^k$ for some graph H. We will only consider the two-dimensional case, i.e. $G = H^2$, but it will also be seen that the algorithm we derive is applicable when the dimensionality of the graph is in general a power of 2, i.e. $G = H^{2^k}$.

Let $G = A \times B = (V, E)$ where $A = B = H$. Also, let $n = |V_H|$. The network in Fig. 5.4 will be used as an example for our arguments. For node $(1, 1)$, we give the messages it has to distribute in Table 5.2. The messages in the first column are meant for the other nodes in A_1. A total exchange within A_1 may thus begin immediately to distribute such messages. Since this total exchange uses only links in the first dimension, node (1,1) is also available

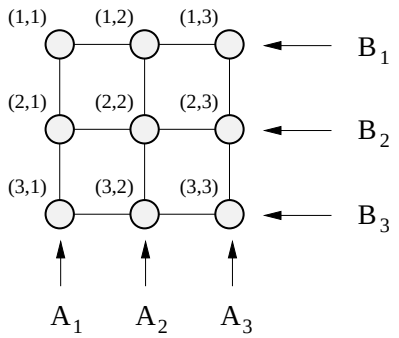

Figure 5.4. *A 3×3 homogeneous mesh*

For A_1	For A_2	For A_3
–	$m_{(1,1)}(1,2)$	$m_{(1,1)}(1,3)$
$m_{(1,1)}(2,1)$	$m_{(1,1)}(2,2)$	$m_{(1,1)}(2,3)$
$m_{(1,1)}(3,1)$	$m_{(1,1)}(3,2)$	$m_{(1,1)}(2,3)$

Table 5.2. *Messages to be transferred from node (1,1)*

to participate in some total exchange in the second dimension (i.e. in B_1). In a general network, node (v_i, u_j) in A_j can participate in a total exchange within B_i as soon as the first total exchange in A_j starts. Within A_j the transferred messages are $m_{(v_i,u_j)}(*, u_j)$, as given in column j of Table 5.1.

Let us see what messages will be involved in the first total exchange within B_i. Our objective is the following: we want every node (v_i, u_j) in B_i to receive $n-1$ appropriate messages so that after this total exchange in B_i is done, another total exchange can be initiated within A_j. Consequently, we seek to arrange the transfers so that (v_i, u_j) receives one message for each node in A_j, i.e. it receives messages with destinations $(*, u_j)$. Notice that any node (v_i, u_j) receives $n-1$ messages through a total exchange in B_i: since A_j has n nodes (including (v_i, u_j)), all the $n-1$ receptions of (v_i, u_j) should be meant for nodes other than (v_i, u_j) itself.

In the network in Fig. 5.4, we let for example node (1,1) send $m_{(1,1)}(2,2)$. This message is at some point receiveded by node (1,2) and it provides one message for the forthcoming total exchange in A_2. If (1,2) sends $m_{(1,2)}(2,3)$, then node (1,3) is also provided with one message for total exchange in A_3. Similarly, (1,3) sends $m_{(1,3)}(2,1)$, needed by node (1,1).

Let us define the following operators:

$$x \oplus_n y \stackrel{\text{def}}{=} [(x + y - 1) \bmod n] + 1$$
$$x \ominus_n y \stackrel{\text{def}}{=} [(x - y - 1) \bmod n] + 1.$$

These operators work like addition/subtraction modulo n but produce numbers ranging from 1 to n instead of 0 to $n-1$, and are better suited for our purposes here. Based on this operator and the preceding discussion, we see that one effective scheduling is to let node (v_i, u_j) (for all i and all j) send, among other messages, message $m_{(v_i,u_j)}(v_{i\oplus_n 1}, u_{j\oplus_n 1})$. Hence, this node will also receive $m_{(v_i,u_{j\ominus_n 1})}(v_{i\oplus_n 1}, u_j)$ from node $(v_i, u_{j\ominus_n 1})$ which it will use for the next total exchange in A_j.

Let us see what other messages are sent during this first total exchange in B_i. In our example, it is seen that since node (1,1) decided to send $m_{(1,1)}(2,2)$, it cannot send another message to node (1,2). Thus it has to send a message to node (1,3). Since this node receives $m_{(1,2)}(2,3)$, which covers one destination in A_3, the only choice for (1,1) is to send $m_{(1,1)}(3,3)$. This message completes the set of messages needed by (1,3) for the next total exchange in A_3 since all other vertices in A_3 are now covered. Similarly, (1,2) and (1,3) must send $m_{(1,2)}(3,1)$ and $m_{(1,3)}(3,2)$ and all three nodes will have a complete set of messages, suitable for total exchanges within A_1, A_2 and A_3.

In general, the second message that node (v_i, u_j) sends is $m_{(v_i,u_j)}(v_{i\oplus_n 2}, u_{j\oplus_n 2})$. This provides node $(v_i, u_{j\oplus_n 2})$ with a second message for the total exchange in $A_{j\oplus_n 2}$. The pattern should now be clear: during the first total exchange in B_i, every node (v_i, u_j), $j = 1, 2, \ldots, n$, sends the following messages:

$$m_{(v_i,u_j)}(v_{i\oplus_n 1}, u_{j\oplus_n 1}),\ m_{(v_i,u_j)}(v_{i\oplus_n 2}, u_{j\oplus_n 2}),\ \ldots,\ m_{(v_i,u_j)}(v_{i\oplus_n (n-1)}, u_{j\oplus_n (n-1)}),$$

or, in a compact form:

$$\Big\{m_{(v_i,u_j)}(v_{i\oplus_n \ell}, u_{j\oplus_n \ell}) \ \mid\ \ell = 1, 2, \ldots, n-1\Big\}.$$

This node provides node $(v_i, u_{j\oplus_n \ell})$ with the ℓth message it needs (i.e. a message destined for node $(v_{i\oplus_n \ell}, u_{j\oplus_n \ell})$ in $A_{j\oplus_n \ell}$). Notice that the above set contains one message to be received by each node $(v_i, u_{j'})$, $j' \neq j$, i.e. it is a perfect set for participation in the first total exchange in B_i. Also, it should be clear that (v_i, u_j) receives the following messages:

$$\Big\{m_{(v_i,u_{j\ominus_n \ell})}(v_{i\oplus_n \ell}, u_j) \ \mid\ \ell = 1, 2, \ldots, n-1\Big\}.$$

Again notice that this set contains one message for each node $(v_{i'}, u_j)$, $i' \neq i$, in A_j. Thus we achieved our goal: every node in B_i receives a full set of messages to be used for the subsequent total exchange in A_j.

Since $A = B$, the first total exchange in A_j finishes exactly when the first total exchange in B_i finishes. Thus the second total exchange in A_j can start immediately using the newly acquired (through the exchange in B_i) messages. Then the story repeats itself: a second

total exchange in B_i can be performed simultaneously with the second total exchange in A_j. Our goal for this total exchange in B_i remains the same: to distribute messages that can be used for a third total exchange in A_j.

The idea behind selecting a group of messages for this second total exchange in B_i is similar to the one in the first total exchange we saw above. Now, we let (v_i, u_j) send messages

$$m_{(v_i,u_j)}(v_{i\oplus_n 2}, u_{j\oplus_n 1}),\ \ldots,\ m_{(v_i,u_j)}(v_{i\oplus_n (n-1)}, u_{j\oplus_n (n-2)}),\ m_{(v_i,u_j)}(v_{i\oplus_n 1}, u_{j\oplus_n (n-1)}).$$

The situation is repeated continuously. While the rth total exchange within A_j is in progress, the rth total exchange in B_i is also performed in order to provide nodes with messages for the next total exchange in A_j. During the rth exchange in B_i a node (v_i, u_j) sends the following messages:

$$\begin{aligned}
&m_{(v_i,u_j)}(v_{i\oplus_n r}, \quad u_{j\oplus_n 1})\\
&m_{(v_i,u_j)}(v_{i\oplus_n (r+1)}, u_{j\oplus_n 2})\\
&\qquad \ldots\\
&m_{(v_i,u_j)}(v_{i\oplus_n (n-1)}, u_{j\oplus_n (n-r)})\\
&m_{(v_i,u_j)}(v_{i\oplus_n 1}, \quad u_{j\oplus_n (n-r+1)})\\
&\qquad \ldots\\
&m_{(v_i,u_j)}(v_{i\oplus_n (r-1)}, u_{j\oplus_n (n-1)}).
\end{aligned}$$

Observe that the destinations $v_{i\oplus_n \ell}$ are in the order given by $\ell = r, r+1, \ldots, n-1, 1, \ldots, r-1$. That is, the natural sequence $1, 2, \ldots, n-1$ which we used in the first total exchange in B_i is left-rotated by r positions. Based on this observation, it is easy to verify that the above set of messages can be given in the compact form:

$$\Big\{ m_{(v_i,u_j)}(v_{i\oplus_n ((r-1)\oplus_{n-1}\ell)}, u_{j\oplus_n \ell}) \ \mid\ \ell = 1, 2, \ldots, n-1 \Big\}. \tag{5.3}$$

Similarly, it is seen that after the rth exchange in B_i, node (v_i, u_j) will have received messages

$$\Big\{ m_{(v_i,u_{j\ominus_n ((r-1)\oplus_{n-1}\ell)})}(v_{i\oplus_n \ell}, u_j) \ \mid\ \ell = 1, 2, \ldots, n-1 \Big\}, \tag{5.4}$$

which can be used during the $(r+1)$th total exchange in A_j.

Let us recapitulate. During the first total exchange in A_j, (v_i, u_j) uses $m_{(v_i,u_j)}(*, u_j)$. Simultaneously, total exchanges in B_i start. During the rth exchange in B_i the same node sends the set of messages given in (5.3), and receives the set given in (5.4). This set will be

```
1   Do in parallel
2       - TEA();
3       - TEB();

    TEB()
4       For r = 1, 2, ..., n − 1
5           Do in parallel for all B_i, i = 1, 2, ..., n
6               Perform Total Exchange in B_i with node (v_i, u_j) sending
                {m_(v_i,u_j)(v_{i⊕_n((r−1)⊕_{n−1}ℓ)}, u_{j⊕_n ℓ}) | ℓ = 1, 2, ..., n − 1};
7       Do in parallel for all B_i, i = 1, 2, ..., n
8           Perform Total Exchange in B_i with node (v_i, u_j) sending m_(v_i,u_j)(v_i, ∗);

    TEA()
9       Do in parallel for all A_j, j = 1, 2, ..., n
10          Perform Total Exchange in A_j with node (v_i, u_j) sending m_(v_i,u_j)(∗, u_j);
11      For r = 2, ..., n − 1
12          Do in parallel for all A_j, j = 1, 2, ..., n
13              Perform Total Exchange in A_j with node (v_i, u_j) sending
                the messages received from the second dimension (B_i);
```

Figure 5.5. *Algorithm A3 for multiport homogeneous networks:* $G = A \times B$ *and* $A = B = H$.

used for the $(r+1)$th exchange in A_j. This will occur for all $r = 1, 2, \ldots, n-1$. All total exchanges in B_i are performed in parallel with the total exchanges in A_j.

The last (nth) total exchange in A_j involves the messages received during the $(n-1)$th total exchange in B_i. It can be noticed that (v_i, u_j) has sent all its messages meant for nodes in all other copies of A, A_k $(k \neq j)$, *except* for nodes (v_i, u_k). In the example of Fig. 5.4, we saw that during the first two exchanges in B_1, node (1,1) sent all its messages with the exception of messages $m_{(1,1)}(1,2)$ and $m_{(1,1)}(1,3)$ which are destined for node (1,2) and (1,3). The situation is similar for nodes (1,2) and (1,3) also: all messages of nodes $(1, *)$ meant for nodes in B_1 are the only messages remaining to be sent.

In conclusion, messages $m_{(v_i,u_j)}(v_i, *)$ of node (v_i, u_j) are the only messages remaining to be sent. Observe that this is a perfect set of messages for a (final) total exchange in B_i. This nth exchange can be performed while the nth exchange in A_j occurs.

What we have described up to now is formulated as Algorithm A3 in Fig. 5.5. The total exchanges in the copies of A and B are completely parallelized, hence lines 1–3. Lines 4–8 perform the transfers we described above in B_i. Lines 9–13 perform the total exchanges in A_j. Notice how simple lines 11–13 are: whatever was sent through the rth exchange in B_i is used during the $(r+1)$th exchange in A_j.

As it is, the algorithm works for any two-dimensional homogeneous network. Extension to more than two dimensions seems rather difficult because the homogeneity will be lost, in the sense that A could be different than B. For example, if $G = H^3$, G can be written as $G = A \times B$ only if $A = H^2$ and $B = H$ or vice versa.

However it is easy to see that the algorithm is applicable if the dimensionality is *a power of 2*. If $G = H^{2^k}$ then we let $A = H^{2^{k-1}}$ and $B = H^{2^{k-1}}$. The algorithm can then be applied recursively for A and B, by e.g. setting $A = H^{2^{k-2}} \times H^{2^{k-2}}$, and so on.

We proceed now to determine the time requirements of Algorithm A3 and to give optimality conditions.

5.4.1 Optimality conditions

Theorem 5.6 *If H has n nodes and total exchange in H requires T_H steps then Algorithm A3 in $G = H \times H$ requires time equal to:*

$$T = nT_H.$$

Proof. Procedure TEA() performs n total exchanges in A_j (for all $j = 1, 2, \ldots, n$ in parallel), thus requiring nT_H steps. Similarly, TEB() also requires nT_H steps. The algorithm finishes when both procedures have finished, i.e. at time $T = nT_H$. ■

By the recursive application of the algorithm for networks where the dimensionality is a power of 2 we have the following corollary.

Corollary 5.7 *Let $G = H^d$, where $d = 2^k$. If total exchange in H requires T_H time units, then total exchange in G can be performed in*

$$T = n^{d-1}T_H$$

steps.

Proof. The proof is by induction. The case of two dimensions was covered in Theorem 5.6. If, as an induction hypothesis, for $G' = H^{d/2}$ we need time $T' = n^{d/2-1}T_H$ then set

$G = G' \times G'$ and apply Theorem 5.6 with G' treated as H, T' treated as T_H, and $n^{d/2}$ treated as n. It is then seen that $T = n^{d/2}T' = n^{d-1}T_H$, as claimed. ■

Theorem 5.8 *Let $G = H^d$, where $d = 2^k$. If total exchange in H can be performed in time equal to the lower bound of (4.1) then the same is true for G.*

Proof. From Corollary 5.7, total exchange in G requires $T = n^{d-1}T_H$ time units, where $n = |V_H|$. If T_H achieves the lower bound in (4.1) then there exists a partition V_{H_1}, V_{H_2} of the node set of H such that

$$T_H = \frac{|V_{H_1}||V_{H_2}|}{C_{V_{H_1}V_{H_2}}},$$

where $C_{V_{H_1}V_{H_2}}$ is the number of links separating the two parts.

Consider the following partition of V, the node set of G:

$$\begin{aligned} V_1 &= \bigcup_{v_i \in V_{H_1}} (v_i, *, \ldots, *) \\ V_2 &= \bigcup_{v_i \in V_{H_2}} (v_i, *, \ldots, *). \end{aligned}$$

Then clearly, $|V_1| = |V_{H_1}|n^{d-1}$ and $|V_2| = |V_{H_2}|n^{d-1}$. Notice that G contains n^{d-1} copies of H and that in order to separate the two parts we only need to disconnect each copy of H, by removing links only in the first dimension. Since $C_{V_{H_1}V_{H_2}}$ links are needed to disconnect each copy of H, we obtain

$$C_{V_1V_2} = n^{d-1}C_{V_{H_1}V_{H_2}}.$$

Thus, V_1 and V_2 is a partition of G such that

$$\frac{|V_1||V_2|}{C_{V_1V_2}} = \frac{n^{d-1}|V_{H_1}|n^{d-1}|V_{H_2}|}{n^{d-1}C_{V_{H_1}V_{H_2}}} = n^{d-1}T_H,$$

which is equal to T, the time needed for total exchange in G. Thus the bound in (4.1) is tight for G, too. ■

Summarizing, Algorithm A3 is a multiport total exchange algorithm for homogeneous networks whose dimensionality is a power of two. If total exchange in H can be performed in time equal to the lower bound of (4.1), then Algorithm A3 optimally solves the problem in G. From the results in Chapter 4, we have the following direct consequence. If H is a ring or a linear array, then Algorithm A3 is an optimal total exchange algorithm for homogeneous meshes and tori with 2^k ($k \geq 1$) dimensions.

Chapter 6

Fat Trees

In this chapter, we will study the communication problems in the context of *fat tree* networks. Fat trees constitute a promising interconnection structure for parallel processing and in fact they have been utilized in a commercial multiprocessor, Thinking Machines' CM-5 [41]. They are based on complete trees and it is this hierarchical structure that seems to make them suitable for many scientific problems as well as for general applications.

As it will be seen shortly, fat trees differ in many ways from all the other networks we have considered up to now. First of all, they do not possess any symmetry. The most important difference though, which will affect the analysis of our communication problems, is that only certain nodes constitute processing elements. The rest of the nodes in the network are only used for routing. This characteristic, in effect, renders most of the analysis in the previous chapters inapplicable to fat trees. For example, the total exchange bounds in Section 4.1 are no longer valid since in their derivation we assumed that all nodes participate in the operation by producing and consuming messages.

Some experimental results on communications in Thinking Machines' CM-5 were given in [54]. Also, related but not exactly applicable to fat trees are the works in [23, 6]. The authors in [23] studied among other problems the problem of broadcasting in complete trees, while in [6], scattering under the single-port model in arbitrary trees was considered. In both cases the network was assumed to be wormhole routed and, unlike fat trees, all nodes were assumed to generate and consume messages.

We introduce complete trees and fat trees formally in the next section. One parameter in fat trees that was not present in networks of the previous chapters is the *capacity* of channels. In graph-theoretic terminology, there may exist multiple edges between adjacent nodes. This will affect the performance of the algorithms since it allows multiple messages to be exchanged between neighbors. We discuss communications under the single-port model in Section 6.2. If a processing node is not allowed to send more than one message at a time, then capacities play no role and the network behaves like a complete tree. Capacities

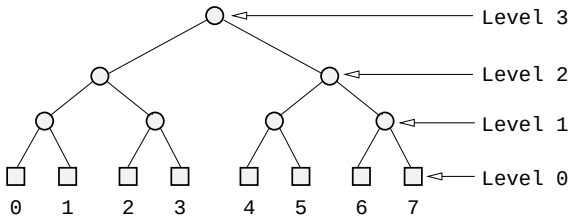

Figure 6.1. *A complete binary tree with 15 nodes (8 leaves)*

certainly affect multiport communications, which will be examined in Section 6.3.

6.1 Preliminaries

A *complete k-ary tree* with height h consists of $(k^{h+1}-1)/(k-1)$ nodes and has the following properties: every node except the leaves has exactly k children and all leaves are at the same level (i.e. at the same distance from the root). A complete binary tree with 15 nodes is shown in Fig. 6.1.

We will number the levels from the bottom to the top. The leaf level is level 0 and the level of the children of the root is level $h-1$. We will also say that the root belongs to level h. A node in level i is at distance $h-i$ from the root of the tree. Since every "internal" node (i.e. a non-leaf node) has k children, level $h-1$ has k nodes, level $h-2$ has k^2 nodes, and in general the ith level has k^{h-i} nodes. The number of leaves is thus $n = k^h$; conversely, if the tree has n leaves then it has height $h = \log_k n$. The leaves are numbered from left to right as $0, 1, \ldots, n-1$ as shown in Fig. 6.1. In what follows we will assume that $h \geq 2$, or alternatively, $n \geq k^2$ (if the height is one then the network degenerates to a star graph which consists of n vertices adjacent only to an additional central vertex).

A k-ary *fat tree* is based on the same topology only branches[1] get thicker as one moves from the leaves to the root of the tree. That is, branches closer to the root may include more physical links than branches further from the root. The *capacity* of a branch is the number of links it includes. A branch between levels $i-1$ and i is called a *level-i branch* and has capacity $c_i \geq 1$. Since branches increase in capacity towards the root, we have $c_i \geq c_j$ if $i > j$. A portion of a k-ary fat tree is shown in Fig. 6.2.

In a fat tree, all *processing* nodes are confined to the leaf level; they are able to generate and consume messages. All the other nodes are used for establishing paths between

[1] The channel joining two adjacent nodes in a tree will be termed "branch" to differentiate with the terms "edge" and "link" which refer to a single physical link.

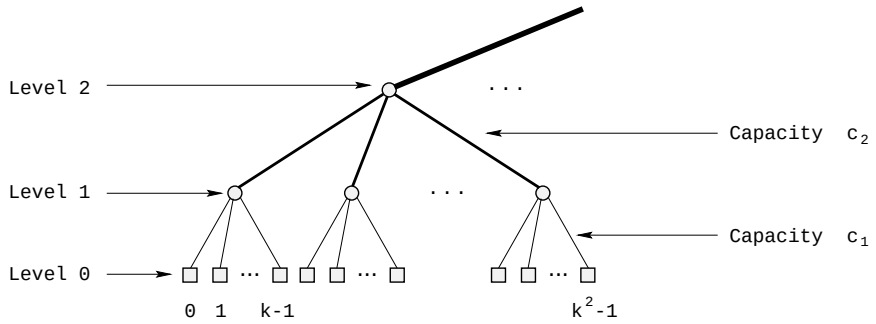

Figure 6.2. *Portion of a k-ary fat tree*

appropriate leaves and are thus called *routing* nodes.

Although the results presented here cover any capacity arrangement, two capacity patterns will be of more interest in later sections: constant and exponential. These two patterns represent two extremes in interconnection size. In the first case $c_i = 1$ for all levels $i = 1, 2, \ldots, h$, and the resulting tree is the complete k-ary tree. In the *exponential* capacities case, $c_i = k^{i-1}$. It is easily seen that in this case the total number of links at every level is equal to $n = k^h$, being thus able to carry messages from all leaves simultaneously.

In what follows, we denote by $B(n)$, $S(n)$, $G(n)$, $MB(n)$ and $TE(n)$ the time needed for broadcasting, scattering, gathering, multinode broadcasting and total exchange correspondingly.

6.2 Communications Under the Single-port Model

It should be clear that if every node is incapable of sending more than one message at every time unit, branch capacities have no effect at all. Any fat tree behaves exactly like the corresponding complete k-ary tree. In this section, we derive lower bounds for the communication problems we consider and provide algorithms that achieve the lower bounds. Notice that we allow any node to receive messages from *all* its neighbors simultaneously but it can only send one message at each step.

6.2.1 Broadcasting

Let us first consider broadcasting from the root v of an arbitrary tree T, where v has d children v_i, $i = 1, 2, \ldots, d$. Let T_{v_i} be the subtree rooted at v_i and assume that broadcasting in T_{v_i} needs $b(T_{v_i})$ steps. Let us further assume for convenience that the trees have indices reversely to their broadcast times, i.e. if $i \leq j$ then $b(T_{v_i}) \geq b(T_{v_j})$. Since the root can send

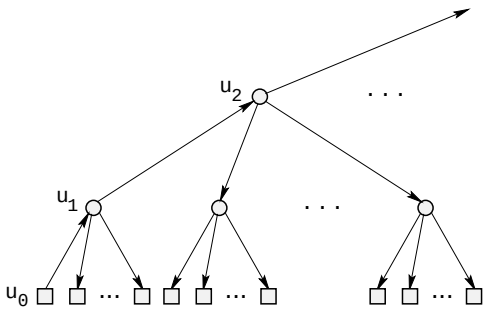

Figure 6.3. *Broadcasting under the single-port model*

the message to only one child at a time, it is clear that the best plan is to send the message to v_1 first, then to v_2, etc. As a result, the time needed to broadcast from v in T is

$$b(T) = \max_{1 \le i \le d} \{b(T_{v_i}) + i\}. \tag{6.1}$$

If T is a complete k-ary tree with height ℓ, then the k subtrees rooted at the children of root v are identical: they are all complete k-ary trees with height $\ell - 1$. Let $B_r(\ell)$ be the broadcast time from the root in T. Then the broadcast time for any of the k subtrees is $B_r(\ell - 1)$, and from (6.1), setting $d = k$, $b(T) = B_r(\ell)$, $b(T_{v_i}) = B_r(\ell - 1)$, we obtain

$$B_r(\ell) = B_r(\ell - 1) + k \Rightarrow B_r(\ell) = k\ell. \tag{6.2}$$

since $B_r(1) = k$.

Our problem, however, is to broadcast from a leaf of a complete k-ary tree. Let, without loss of generality, the leftmost leaf be the source node, labeled u_0 in Fig. 6.3. Starting from u_0's only parent (labeled u_1) and following the upward edges, we observe the following: below the ith node u_i in this path there are $k - 1$ complete k-ary trees of height $i - 1$ (except the one rooted at u_{i-1}). If u_i gets informed about the message, then all the leaves in these $k - 1$ subtrees can be informed in $B_r(i - 1) + k - 1$ steps at the earliest. Of course, u_i also has to inform its parent. In order to determine the exact broadcast time for v_0 we start from the end of the upward path, calculate the minimum time needed for broadcasting from that node, and then work our way down to u_0 using (6.1) at each step. We let $b(T_{u_i})$ be the broadcast time remaining after u_i receives the message.

Node u_h in the path is the root node of our original tree. It can only be informed through its leftmost child. When u_h receives the message, it has to inform its $k-1$ remaining children as we mentioned above and for this $B_r(h - 1) + k - 1$ steps are needed. Consequently,

$$b(T_{u_h}) = B_r(h - 1) + k - 1.$$

Source Node:

1 Send message to parent;

Every Routing Node:

1 Send message to parent (if any);

2 Send message to each child in turn, except to the one the message came from;

Figure 6.4. *Optimal broadcasting algorithm under the single-port model*

Next, consider u_{h-1}. This node has to inform its $k-1$ subtrees lying below plus its parent, u_h. For the subtrees below $B_r(h-2)+k-1$ steps are needed, while for broadcasting from u_h, $B_r(h-1)+k-1$ steps are necessary as we just saw. According to (6.1), the best plan is to send the message to u_h first and to the subtrees below next. Consequently, we need time:

$$b(T_{u_{h-1}}) = \max\{b(T_{u_h})+1, B_r(h-2)+k\} = b(T_{u_h})+1.$$

Proceeding downwards in this manner, it is seen that any node u_i must first inform its parent (u_{i+1}) and then its $k-1$ remaining children, and

$$b(T_{u_i}) = \max\{b(T_{u_{i+1}})+1, B_r(i-1)+k\} = b(T_{u_h})+h-i,$$

giving

$$b(T_{u_0}) = b(T_{u_h})+h = B_r(h-1)+k-1+h.$$

Since from (6.2) $B_r(h-1) = k(h-1)$, we have the following result (recall that $h = \log_k n$):

Theorem 6.1 *Broadcasting in fat trees under the single-port model requires* $(k+1)\log_k n - 1$ *steps.*

Fig. 6.4 shows an algorithm that achieves this lower bound; it is directly derived from the preceding analysis.

6.2.2 Scattering

Scattering and gathering in general trees of processors were considered in [6]. Although related, our case differs in that the significant nodes are confined to the leaves of the tree. The observation that the source node has to send or receive $n-1$ different messages over its incident link leads to a simple lower bound of $S(n) = G(n) \geq n-1$ steps. As a matter

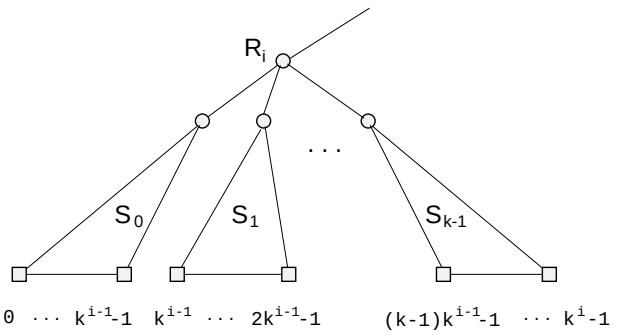

Figure 6.5.

of fact, the exact bound is n or $n+1$ steps (depending on k) and can be seen as follows. In gathering $n-1$ messages at processor 0, in the first two steps we can at most receive one message since the closest leaf is at distance two. Consequently there will be at least one step with no message reception by node 0, i.e. $S(n) = G(n) \geq n$. If the tree is binary ($k = 2$), there will be one extra step of no reception since the next closest leaves (2 and 3) are both at distance four from node 0, i.e. $S(n) = G(n) \geq n+1$ if $k = 2$.

Since a gathering algorithm can be had from a scattering algorithm (and vice versa), we only consider scattering here. The lower bound of n or $n+1$ steps can be achieved using the *furthest-first* scheduling discipline whereby the source node gives priority to messages that have to travel the furthest.

Theorem 6.2 *The furthest-first discipline results in an optimal scattering algorithm.*

Proof. Assuming without loss of generality that leaf node 0 is the source node, consider the routing node R_i at level i which is the root of the subtree containing nodes 0 to $k^i - 1$, as shown in Fig. 6.5. Let S_j be the subtree rooted at the jth child of R_i from the left (the source node belongs to S_0). The last message destined for any of the subtrees $S_1, S_2, \ldots, S_{k-1}$ leaves node 0 before any message destined for a leaf within S_0 does, according to the furthest-first regimen; that is, it leaves at time $n - k^{i-1}$. Since this message is destined for a node at distance $2i$ from node 0, the last message for any of the aforementioned $k-1$ subtrees of R_i is received at time

$$T_i = n - k^{i-1} + 2i - 1.$$

Consequently, the algorithm finishes at time

$$T = \max_{1 \leq i \leq h} \{T_i\}.$$

It can be seen that T_i is a decreasing function of i for $i \geq 2$. The maximum value of T_i for

integer i can thus occur only for $i = 1$ or $i = 2$. Since $T_1 = n$ and $T_2 = n - k + 3$, we see that $T = n$ if $k \geq 3$ and $T = n + 1$ if $k = 2$, as claimed. ■

6.2.3 Multinode broadcasting

In multinode broadcasting, every node broadcasts its own message. If all nodes start broadcasting at the same time, contention will be observed sooner or later at the routing nodes. An optimal multinode algorithm schedules the traffic in each node so that all leaves receive all messages at the minimum possible time. The algorithm we present next will be proven to be optimal. First however, we derive a lower bound for multinode broadcasting algorithms.

Theorem 6.3 *Any multinode broadcasting algorithm under the single-port model requires at least* $kn + (k+1)(\log_k n - 2) + 1$ *steps.*

Proof. Consider the k children v_i, $i = 1, 2, \ldots, k$, of the root node. Node v_1 is at level $h - 1$ and consequently has k^{h-1} leaves lying below. Each of the leaves will generate a broadcast message and this message must be sent through a child of v_1 to v_1 itself. Then v_1 must send it to its remaining $k - 1$ children and to the root of the tree. Thus v_1 must make $kk^{h-1} = k^h$ transmissions in total.

In addition, v_1 will receive all messages coming (through the root) from v_2, v_3, ..., v_k. Each of these messages must be broadcast by v_1 to all its k children. Thus each of these messages also forces v_1 to make k transmissions. Since each v_i will send k^{h-1} messages, v_1 will receive $(k-1)k^{h-1}$ messages from the other children of the root. The total number of transmissions for v_1, including the k^h transmissions from the previous paragraph, will thus be $k(k-1)k^{h-1} + k^h = k^{h+1} = kn$. Under the single-port model, v_1 can only send one message at a time. Thus it must be busy for at least kn steps.

The *first* message will arrive at v_i at time $h - 1$ at the earliest since the closest leaf is at distance $h - 1$. The *last* message to leave v_i will be sent to one of its children. But this child will have to broadcast this message to the subtree it roots which is a complete k-ary tree with height $h - 2$. In other words, after the last message leaves v_i it will need another $B_r(h-2)$ steps (as given by (6.2)) in order for the last leaf to receive it. Consequently, any multinode broadcasting algorithm needs time

$$\begin{aligned} MB(n) &\geq kn + h - 1 + B_r(h-2) \\ &= kn + h - 1 + k(h-2) \\ &= kn + (k+1)(\log_k n - 2) + 1. \end{aligned}$$

■

Every Leaf Node:

Send message to parent;

Every Routing Node: (except the root)

A Receive all messages from children, storing them and sending a copy of them upwards, one-by-one;

B Until the last message arrives from parent

Send one message to each child in turn;

Root Node:

B While receiving all messages from children, keep sending one message to each child in turn;

Figure 6.6. *Optimal multinode broadcasting algorithm under the single-port model*

In Fig. 6.6, we give an algorithm for the multinode broadcasting problem and in Fig. 6.7 we demonstrate its operation for $n = 4$ leaves. After some initial delay, a routing node starts receiving messages from its subtrees and from its parent. During "phase" A in the algorithm, a node keeps forwarding all messages from the children to the parent but also retains a copy of them in its queue (in Fig. 6.7, phase A for level-1 nodes ends in the 3rd step). In phase B, messages in the queue (and any other messages coming from the parent) are transmitted to the appropriate children. Notice that the children *take turns* in receiving messages, i.e. each child receives one message per k steps.

Lemma 6.4 *By the time a routing node sends the last message to its parent (i.e. phase A ends), the queues of all its k children will be empty.*

Proof. Let R_i be a node at any level $i \leq h-2$ and let R_{i+1} be its parent. Node R_i sends the last message to its parent at time $i + k^i$ since it receives k^i messages from the leaves below and the first message arrives at the ith step. Similarly, R_{i+1} sends the last message to its parent at time $i + 1 + k^{i+1}$. Now, R_i needs exactly $(k-1)k^i$ time units to exchange the messages between its children (since it has received k^i messages and each one must be given to $k-1$ children). Consequently, its queue becomes empty at time $i + k^i + (k-1)k^i = i + k^{i+1}$. That is, R_i's queue becomes empty one step before R_{i+1} finishes sending messages to its parent. ■

Theorem 6.5 *The algorithm presented in Fig. 6.6 is optimal.*

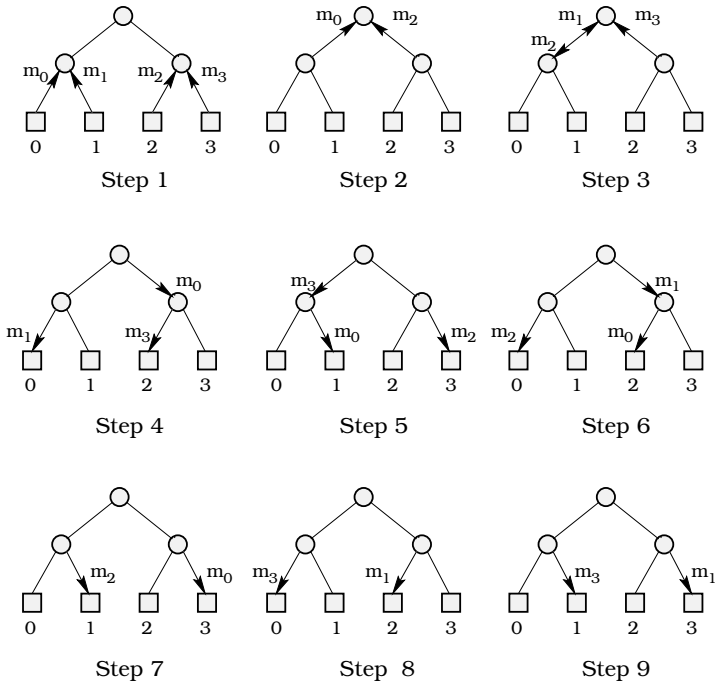

Figure 6.7. *Multinode broadcasting in 4 leaves under the single-port model*

Proof. Consider any routing node R_i at level i and let T_A and T_B be the time this node needs for phases A and B correspondingly. Notice that since R_i is in level i, phase A starts i steps after the time multinode broadcast commences.

During phase B, each message sent to a child of R_i has to be broadcast to the corresponding subtree which has height $i-1$. Consider R_{i-1}, a child of R_i. When R_{i-1} receives the first message from R_i, its queue is empty as Lemma 6.4 shows. For every received message, R_{i-1} needs k steps to broadcast it to its k children. From the way phase B is described above, R_{i-1} receives one message every k steps from R_i. Consequently, at the time R_{i-1} finishes broadcasting a message, it receives a new one from its parent. As a result, at the moment any message arrives from R_i, it finds R_{i-1}'s queue empty. The conclusion is that there is *no contention* at any node during phase B. Thus, after R_i sends the last message to its children (i.e. phase B ends), we only need $B_r(i-1)$ steps to finish.

Concentrating on a node R_{h-1}, it is seen that the time needed for the algorithm is $T = (h-1) + T_A + T_B + B_r(h-1)$. This is because R_{h-1} receives its first message at time $h-1$; it is busy for $T_A + T_B$ steps, and $B_r(h-1)$ steps is sufficient to deliver the last message it sends in phase B since, according to the above discussion, this message does not compete with any other message. Clearly, $T_A = k^{h-1}$ since k^{h-1} messages are received

from its children. These messages must also be exchanged among R_{h-1}'s children and this takes $(k-1)k^{h-1}$ steps. In addition, R_{h-1} receives $(k-1)k^{h-1}$ messages from the root node that have to be sent to all its children. Since only one transmission occurs at a time, $T_B = (k-1)k^{h-1} + k(k-1)k^{h-1}$. Finally, $B_r(h-1) = 2(h-2)$. The total time needed is seen to be $T = k^{h+1} + (k+1)(h-2) + 1$ which is the best possible according to the lower bound derived previously. ■

One undesirable characteristic of the above algorithm is that it requires routing nodes to be equipped with large queues to buffer the incoming messages. We defer a detailed analysis of queueing for the section dealing with the multiport model of communication since the problem arises there too.

6.2.4 Total exchange

Total exchange causes the densest form of contention on the network. Consider the subtree rooted at a routing node R_i at level i. A leaf u in this subtree will generate $n-1$ messages in total, out of which $n-k^{i-1}$ will pass through R_i as follows. There will be $n-k^i$ "upward" messages, that is messages to be sent through the parent of R_i to a subtree other than the one R_i roots. There will also exist messages to be given from the subtree u lies in, to the other $k-1$ subtrees of R_i. Those messages will be $(k-1)k^{i-1}$ in number, giving a total of $n-k^{i-1}$ messages of u that go through R_i. Under the single-port model we may easily determine the lower bound of total exchange algorithms for messages generated at the leaves.

Theorem 6.6 *Any total exchange algorithm under the single-port model requires at least* $n^2(2k+1)(k-1)/k^3 + 2\log_k n - 3$ *steps.*

Proof. Consider any total exchange algorithm and concentrate on some child v of the root node as we did in Theorem 6.3. Node v receives $k^{h-1}(n-k^{h-2})$ different messages from its k^{h-1} leaves below. In addition, v receives $(k-1)k^{h-1}k^{h-1}$ messages from the root of the tree, originating at the other subtrees of the root. All these messages must be transmitted to the appropriate neighbors of v and under the single-port model only one transmission is possible at a time. Accounting for the initial delay for the first message to arrive at v ($h-1$ steps) and the smallest possible delay for the last message leaving v to arrive to its destination (at best $h-2$ steps *after* the departure instant), we obtain

$$TE(n) \quad \geq \quad k^{h-1}(n-k^{h-2}) + (k-1)k^{h-1}k^{h-1} + h - 1 + h - 2$$

Every Leaf Node:

Send the $n-1$ messages in a furthest-first order;

Every Routing Node: (except the root)

A While there exist upward messages from children
    Send one message to parent;

B Until the last message arrives from parent
    Send one message to any (appropriate) child;

Root Node:

B While receiving all messages from children, keep sending one message to any (appropriate) child;

Figure 6.8. *Optimal total exchange algorithm under the single-port model*

$$= n^2\frac{(2k+1)(k-1)}{k^3} + 2h - 3.$$

■

We now present an algorithm that achieves the lower bound of Theorem 6.6. Every leaf node transmits its $n-1$ messages under the *furthest-first* discipline as in the case of scattering and every routing node sends all possible messages to its parent before it starts sending any messages to its children. The algorithm is given in Fig. 6.8.

Since leaves follow a furthest-first rule, "upward" messages arrive in routing nodes first, followed by messages destined to within the subtrees rooted at the routing nodes. In a similar fashion to Lemma 6.4 we have the following.

Lemma 6.7 *By the time a routing node sends the last message to its parent (i.e. phase A ends), the queues of all its k children will be empty.*

Proof. Let R_i be a node at any level $i \leq h-2$, and let R_{i+1} be its parent. Since node R_i receives $k^i(n-k^i)$ "upward" messages, it sends the last message to its parent at time $i+k^i(n-k^i)$; similarly, R_{i+1} sends the last message to its parent at time $t = i+1+k^{i+1}(n-k^{i+1})$. Now, R_i receives an additional $k^i(k-1)k^{i-1}$ messages to be exchanged between its k subtrees. Thus, its queue becomes empty at time $i + k^i(n-k^i) + k^i(k-1)k^{i-1} = i + k^i(n-k^{i-1})$. It is easily seen that $i + k^i(n-k^{i-1}) < t$ for $k \geq 2$. Thus R_i's queue becomes empty before R_{i+1} finishes sending messages to its parent. ■

Problem	Time
Broadcasting	$(k+1)\log_k n - 1$
Scattering/Gathering	n (if $k \geq 3$) , $n+1$ (if $k=2$)
Multinode Broadcasting	$kn + (k+1)(\log_k n - 2) + 1$
Total Exchange	$n^2(2k+1)(k-1)/k^3 + 2\log_k n - 3$

Table 6.1. *Time requirements for communications under the single-port model*

Theorem 6.8 *The algorithm presented above is optimal.*

Proof. The proof is similar to the proof of Theorem 6.5 so the details are omitted. Using Lemma 6.7 it is seen that during phase B in any routing node's schedule there is no contention of messages. Calculating the time needed for phases A and B leads to the desired result. ■

6.2.5 Discussion

We summarized the results of the previous analyses in Table 6.1. It is interesting to compare the relative performance of different trees given that the number of leaves (n) is fixed. The broadcasting time is a decreasing function of k for $2 \leq k \leq 4$ and increases with k for $k \geq 4$; quaternary trees need thus the minimum time. For multinode broadcasting though, binary trees give the best performance as $MB(n)$ is an increasing function of k for $k \geq 2$. Finally, for the total exchange problem (ignoring the logarithmic term) it is seen that larger values of k are needed; at the extreme where $k = \sqrt{n}$ the network consists of only three levels and total exchange can be performed in $\Theta(n\sqrt{n})$ steps.

6.3 Communications Under the Multiport Model

We now assume that a node is able to utilize all its incident links simultaneously. This model of communication is affected by the capacity arrangement on the branches of the tree since now the number of messages allowed to cross a link can be greater than one. In order to compare the performance of fat trees with that of the simple complete k-ary tree, we will assume that the processor (leaf) connections to/from the routing network are fixed, and consist of exactly one link. In other words, the capacity of level-1 branches will be $c_1 = 1$.

6.3.1 Single-source communications

We have already seen that if $e(v)$ is the eccentricity of node v in any network, then broadcasting from v under the multiport model takes time equal to $e(v)$. In our case, since the furthest node from a source is a leaf at distance $2h$, broadcasting requires $B(n) = 2h = 2\log_k n$ steps. It is easily accomplished by setting the routing nodes to a broadcast mode whereby the received message is replicated towards all directions.

Scattering and gathering, under our assumptions, is governed by the same bounds as in the single-port case; there is only one link available from a leaf to a routing node, forcing only one message to be send or received at a time by a leaf. Consequently, $S(n) = G(n) \geq n$ (or $n+1$ if $k = 2$). Had we allowed $c_1 > 1$, other lower bounds could have been derived in an obvious way.

6.3.2 Multinode broadcasting

The same argument used for deriving bounds for scattering/gathering algorithms can be used to determine lower bounds for multinode broadcasting in the multiport model since every leaf has to receive $n-1$ different broadcast messages. The exact bounds are $MB(n) \geq n$ if $k \geq 3$ or $MB(n) \geq n+1$ if $k = 2$. It is seen from this bound (and from the fact that the bound will be shown to be tight) that capacities within the routing network make no difference at least in terms of speed.

We show next that in a complete k-ary tree where all nodes begin broadcasting at the same time and all nodes have enough buffers, all messages will have been received at time n, i.e. multinode broadcasting can be performed in time equal to the lower bound. The algorithm we follow is in effect a *flooding* procedure [65] where received messages are replicated to all possible directions (expect the one they came from) when received by intermediate nodes.

Theorem 6.9 *Multinode broadcasting can be performed in time equal to the lower bound.*

Proof. The proof is by induction on the height h of a complete k-ary tree. We will first assume that $k \geq 3$. Consider only two levels (i.e. $h = 1$ and there exist k leaves in total). It is clear that after the first step the root node knows all messages. Furthermore it is easy to see that the root may distribute them to the appropriate leaves in another $k-1$ steps using the following schedule: during step $i = 2, 3, \ldots, k$ it sends the message of leaf $i-1$ to leaves 0, 1, ..., $i-2$ and the message from leaf $i-2$ to leaves $i-1$, i, ..., $k-1$. The algorithm is thus completed in k steps.

Assume now as an induction hypothesis that the theorem holds for height $h-1 \geq 1$ (i.e. for k^{h-1} leaves we need k^{h-1} steps). We will show it holds for h levels, that is the algorithm requires $n = k^h$ steps.

A complete tree with height h consists of k trees $T_1, T_2, \ldots, T_k$ each with height $h-1$, rooted at nodes $v_1, v_2, \ldots, v_k$, plus an additional node v adjacent to $v_1, v_2, \ldots, v_k$. Concentrate on T_j without loss of generality. Let t_f be the time the first message(s) arrive at v_j. The rest of the messages from the leaves of T_j are pipelined below v_j and reach v_j in subsequent steps. Let also t_l be the time the last message from a leaf in T_j leaves v_j downwards, that is, the last message to complete multinode broadcasting within T_j. The first message from v_j will be available at some other node v_m at time $t_f + 2$ since it has to go through the root v. This also means that v_j itself will start receiving messages from other trees T_m $(m \neq j)$ continuously beginning at time $t_f + 2$. According to the induction hypothesis the algorithm operates in k^{h-1} steps within T_j. Since v_j will receive a total of $(k-1)k^{h-1}$ messages from other trees, and since these messages will leave v_j downwards after the k^{h-1} messages from within T_j do, the algorithm will operate in $k^{h-1} + (k-1)k^{h-1} = k^h = n$ steps; the only requirement is that the last message of T_j leaving v_j downwards is *immediately* followed by the additional messages from the other trees. In other words, all we have to prove is that $t_f + 2 \leq t_l$.

Clearly, the first message arrives in v_j at time $t_f = h - 1$. Node v_j will receive in total k^{h-1} messages from the leaves of T_j, each of which has to be transmitted to $k-1$ children of v_j; that is, there must be $(k-1)k^{h-1}$ transmissions towards its k children. Since only k transmissions may occur at any step, the last such message to leave v_j will do so at time $t_l = t_f + (k-1)k^{h-1}/k$. In order to show that $t_f + 2 \leq t_l$, all we have to show is that $(k-1)k^{h-2} \geq 2$, which is true since we have assumed that $k \geq 3$ and $h - 1 \geq 1$. This completes the induction.

Quite similar arguments can be used to prove that for $k = 2$ the algorithm takes $n + 1$ steps. The only difference is that the induction basis is for a tree with 3 levels. Fig. 6.9 shows that the algorithm takes 5 steps in this case. ■

6.3.2.1 Queueing considerations

The flooding algorithm analyzed above achieves the lower bound but bares the cost of excessive queueing requirements. Concentrate on a routing node other than the root node, i.e. a node R_i at level i, $1 \leq i \leq h-1$, of a simple k-ary tree. The node must maintain a queue at each outgoing link, for a total of $k+1$ queues. When a message is received,

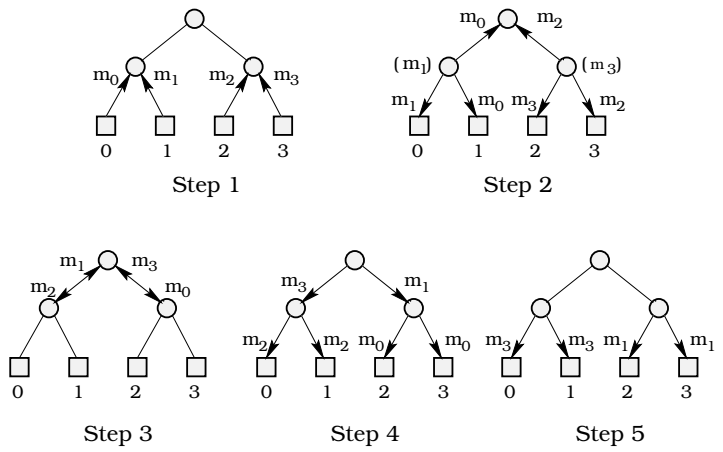

Figure 6.9. *Multinode broadcasting in a 4-leaf binary tree*

it is deposited to all the output queues (except one). Consider first the queue of the link between R_i and its parent. Node R_i is continuously receiving k messages from its children but can only forward one of them at a time to its parent. Since this occurs for k^{i-1} units and R_i has to send k^i messages to its parent, the queue of the link reaches the size of $k^i - k^{i-1} = (k-1)k^{i-1}$ messages.

On the other hand, if $i \geq 3$, there will be $n - k^i$ consecutive messages between (and including) steps $i+2$ and $i+n-k^i+1$, coming from the parent of R_i and intending to go through the link between R_i and one of its children. The same link is used by the $(k-1)k^{i-1}$ messages of the other $k-1$ children, between steps i and $i+(k-1)k^{i-1}-1$. As a result, the queue of the link reachs the size of

$$\min\{i+(k-1)k^{i-1}-1,\ i+n-k^i+1\} - (i+2) + 1$$

messages; since $i < h$, it is seen that the exact size will be $(k-1)k^{i-1}-2$ messages.

Although increasing the branch capacities intuitively seems to reduce the queueing requirements, this is not the case for all links. Assume that the last message to be received by node 0 is the message m from a node at distance $2i$ away. Normally m would arrive at time $2i$ but due to contention it arrives at time n ($n+1$ if the tree is binary) instead; hence it was delayed for $n-2i$ steps. Assuming FIFO queues, this means that the *sum* of the queue sizes along its path reached the value $n-2i$; on the average the queue size per node was $(n-2i)/2i$. The minimum value for the last expression occurs for $i = h$, and it shows that at best the average queue size is $O(n/\log_k n)$. In conclusion, increased capacity allows more messages to move upwards simultaneously but does not reduce queueing on their way down to the leaves.

```
1   For all i = 0, 1, ..., h − 1
        /* Phase i */
2       Do in parallel for all level h − i nodes
3           Broadcast the messages from each of the k subtrees to
            the other k − 1 subtrees;
```

Figure 6.10. *Multinode broadcasting algorithm that eliminates queues*

It should be clear that the queueing requirements of the algorithm are excessive. The routing nodes should be kept as low in complexity and cost as possible; the presence of queueing plus the large size of the queues do not contribute to that effect. Next we present an $(n + 2\log_k n - 2)$-step algorithm that eliminates the queues completely.

6.3.2.2 Eliminating the queues

We show here that for multinode broadcasting it is possible to eliminate the queues at every node at the expense of $2\log_k n - 2$ extra steps. The algorithm induces no contention between messages; nodes should only be capable of buffering one message from every incoming link. It operates as follows. Initially, every node in the k subtrees of the root node broadcasts its message *only* to the other $k - 1$ subtrees. In the normal broadcast algorithm we discussed in Section 6.3.1, when a message is received by a routing node it is replicated towards all directions. Now, however, we insist that such a node sends it only to its parent; only if the message comes from the parent is it broadcast to the children.

The k^{h-1} broadcastings from each subtree of the root do not start simultaneously but in different time steps to avoid contention. After all messages have been transferred, we are left with k subtrees which need to perform a multinode broadcasting within themselves. The algorithm proceeds recursively in the same manner until we are left with one-node subtrees. Iteratively, the algorithm can be stated as shown in Fig. 6.10.

During phase i the routing nodes involved are roots of trees with k^{h-i} leaves, and consequently each one receives k^{h-i} messages from the leaves below. In each steps, such a node receives k messages from its children. It is possible to exchange them among the children in $k - 1$ steps as was shown in the proof of Theorem 6.9. Consequently to avoid contention, there must be only k leaves transmitting every $k - 1$ steps. This means that phase i of the algorithm requires time

$$T_i = (k-1)k^{h-i-1} + 2(h-i) - 1,$$

accounting for the fact that the first message(s) arrives in the routing node at step $h-i$ and that the last message(s) to leave this node needs $h-i$ steps to reach the appropriate leaves. In total, the algorithm requires time

$$T = \sum_{i=0}^{h-1} T_i = k^h + h^2 - 1. \tag{6.3}$$

Instead of executing the phases serially, it is advantageous to pipeline them appropriately. Consider phase i exactly $h-i-1$ steps before it finishes. This is the time when the last message from a level $h-i$ node is transferred to one of its children. Phase $i+1$ could have safely started $h-i-2$ steps before that time instant and no contention would occur since no message of that phase would have reached the children of the level $h-i$ node yet. Consequently, phases i and $i+1$ can have $2h-2i-3$ steps overlapping, leading to a savings of

$$\sum_{i=0}^{h-2}(2h-2i-3) = h^2 - 2h + 1 \quad \text{steps.}$$

Using (6.3), we conclude that the total number of steps for the new algorithm is

$$T = k^h + 2h - 2 = n + 2\log_k n - 2.$$

6.3.3 Total exchange

Lower bounds for the total exchange problem can be derived by considering the messages that go through a level $h-1$ routing node. A particular node at this level receives $k^{h-1}(n-k^{h-1}) = n^2(k-1)/k^2$ "upward" messages from the leaves of the subtree it roots; those messages go through the other nodes in level $h-1$. If the branch capacities are c_h, then only c_h messages can be transferred at a time from our node to the root of the tree. Hence,

$$TE(n) \geq n^2 \frac{k-1}{k^2 c_h} + 2\log_k n - 1. \tag{6.4}$$

A simple total exchange algorithm that works for any capacity pattern and induces no queueing can be derived from the multinode broadcasting algorithm of the previous section. Initially, the k subtrees of the root node exchange their $n(n-k^{h-1})$ messages meant for each other. Then, the k subtrees perform internally a total exchange in parallel. Iteratively, the algorithm can be stated as in Fig. 6.11.

During the ith phase a node at level $h-i$ has to pass $k^{h-i}(k^{h-i}-k^{h-i-1})$ messages over its k incident branches which have capacity c_{h-i}. This means that a maximum of kc_{h-i} messages can cross towards the node at a time. To avoid contention while maintaining

```
1    For all i = 0 to h − 1
         /* Phase i */
2        Do in parallel for all level h − i nodes
3            Transfer all messages from each of the k subtrees
             to the other k − 1 subtrees;
```

Figure 6.11. *A total exchange algorithm with no contention*

maximum speed, exactly kc_{h_i} leaves should dispatch messages at a single step, and the messages should be appropriately chosen so that their destinations are distinct. We omit any implementation details (see [18]) since we are only interested in the time requirements of the algorithm. Phase i clearly requires

$$T_i = \left\lceil \frac{k^{h-i}(k^{h-i} - k^{h-i-1})}{kc_{h-i}} \right\rceil + 2(h-i) - 1,$$

where we also considered the delays for the first message to arrive at a level $h-i$ node and for the last message leaving this node to arrive at the appropriate leaf. The total time needed for the algorithm is

$$T = \sum_{i=0}^{h-1} T_i = \sum_{i=1}^{h} \left\lceil \frac{(k-1)k^{2i-2}}{c_i} \right\rceil + h^2.$$

Pipelining the phases as in the case of multinode broadcasting saves in total $h^2 - 2h + 1$ steps, giving a final number of steps

$$T = \sum_{i=1}^{h} \left\lceil \frac{(k-1)k^{2i-2}}{c_i} \right\rceil + 2h - 1. \tag{6.5}$$

In particular, if the network is the simple k-ary tree ($c_i = 1$), the algorithm takes

$$T = (n^2 - 1)\frac{k-1}{k^2-1} + 2h - 1.$$

If the capacities follow the exponential rule, i.e. $c_i = k^{i-1}$, we obtain from (6.5)

$$T = k^h + 2h - 2 = n + 2h - 2.$$

It is seen from (6.4) that the time needed for our algorithm is slightly suboptimal by a small constant factor. For the exponential capacities case, the bound of (6.4) is not tight since it predicts $TE(n) \geq n - n/k + 2h - 2$. It should be clear that $TE(n) \geq n$ (or $TE(n) \geq n+1$ if the tree is binary) is a tighter bound, since multinode broadcasting can be performed in

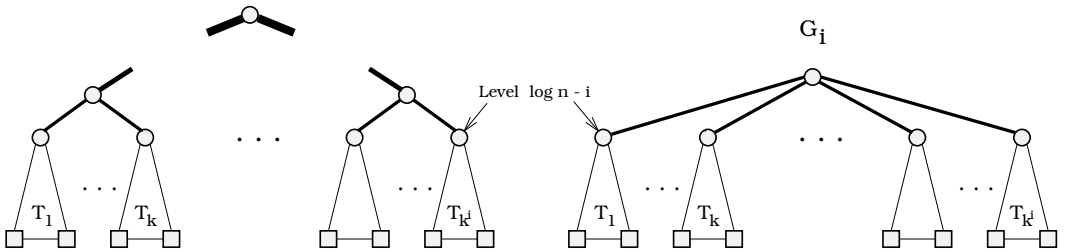

Figure 6.12. *Collapsing the last i levels of the tree gives graph G_i*

at most as many steps as total exchange [5]. According to this last bound, the algorithm we presented above is suboptimal by $2\log_k n - 2$ steps. It is interesting to see whether the last bound is tight or not. In the next section we show that actually there exists a tighter lower bound for trees with exponential capacities.

6.3.3.1 A tighter bound for exponential capacities trees

We show here that for the case of trees with exponential capacities arrangement, a bound tighter than $TE(n) \geq n$ exists which guarantees that the total exchange algorithm we presented is very close to optimal (within $2\log_k \log_k n$ steps) for such trees.

If at any given step of a total exchange algorithm a leaf did not receive a message, we say that a *hole* occurred in its receptions. Notice that since each leaf must receive $n-1$ messages, if the average number of holes per leaf node was w then the algorithm takes time $T \geq n - 1 + w$; the equality holds if all leaves had the same number of holes (w).

Consider the graph G_i which is derived from our original tree by collapsing its last i levels into a single vertex (see Fig. 6.12). We then have a collection of k^i subtrees $T_1, T_2, \ldots, T_{k^i}$ each having n/k^i leaves and their roots have a single common parent. The height of the new graph is $h - i + 1$. In addition, we introduce new edges between every pair of leaves within each of the k^i subtrees so that messages within a subtree can be transferred in a single step. In order to avoid those edges begin used simultaneously, we also impose the restriction that *only one message may be received by a leaf at any time* as in the original tree. It is seen that any total exchange algorithm for the original tree is a total exchange algorithm for G_i (but not vice versa) — the message transfers within the last i levels of our tree are substituted by no-ops in G_i and the additional edges in G_i are not utilized. As a result, the graph G_i can be used to derive a lower bound on total exchange algorithms for our original tree.

A message is called *internal* to subtree T_j if both the source and the destination lie in

T_j; otherwise the message is *external*. The crucial observation is that holes start appearing in G_i after the first external message is dispatched. Assume that T_1 is the one to send the first e external messages at time t. Then at time $t+1$ there will be e holes in the leaves of T_0. In fact, due to the $2(h-i+1)$-step delay external messages suffer before reaching their destinations, no messages arrive from any other subtree before time $t+2(h-i+1)$. Consequently, the number of holes in T_0 between times t and $t+2(h-i+1)$ is equal to the total number of external messages it sent during this period.

We now find the minimum number of holes occur in every leaf in G_i. Consider any total exchange algorithm for G_i and partition time in $2(h-i+1)$-step periods. If there are p such periods (plus possibly a last, shorter one), the algorithm needs time $T \geq 2p(h-i+1)$. Assume that at the sth period there are $w_s^{(j)}$ holes in T_j. Hence in the first period the total number of holes is

$$w_1 = w_1^{(1)} + w_1^{(2)} + \cdots + w_1^{(k^i)}.$$

This is exactly the number of external messages sent (in total) during the first period. During the second period the total number of holes is

$$w_2 = w_2^{(1)} + w_2^{(2)} + \cdots + w_2^{(k^i)}$$

and as a result the total number of external messages sent during the second period is *at most* equal to $w_1 + w_2$. In general, during the jth period the maximum number of external messages is $w_1 + w_2 + \cdots + w_j$. Hence after p periods the total number of external messages observed is at most

$$\begin{aligned}
\sum_{j=1}^{p}(w_1 + w_2 + \cdots + w_j) &= \sum_{j=1}^{p}(p-j+1)w_j \\
&= p\sum_{j=1}^{p} w_j - \sum_{j=1}^{p}(j-1)w_j \\
&= pW - Q
\end{aligned}$$

where $W = \sum w_j$ is the total number of holes and Q is a non-negative term.

Each leaf node must send in total $n-1$ messages out of which $n - n/k^i$ are external- Consequently, the total number of external messages is $n(n-n/k^i)$. This means that

$$pW \geq n(n-n/k^i).$$

Notice that the average number of holes per node is $w = W/n$ and since $T \geq 2p(h-i+1)$, the last inequality yields

$$wT \geq 2n(1-\frac{1}{k^i})(h-i+1). \tag{6.6}$$

From our earlier discussion, if w is the average number of holes per leaf then

$$T \geq n - 1 + w, \tag{6.7}$$

with equality holding if all leaves have exactly w holes. Inequalities (6.6) and (6.7) can be combined to determine the minimum value of T. Since (6.6) holds for any value of i, we can choose i so that we obtain the best bound on T. For example if $i = 1$, it can be seen that T cannot be less than $n+h-1$. A tighter bound can be had if we select $i = \log_k h + 1$; we obtain from (6.6):

$$\begin{aligned} wT &\geq 2n(1 - \frac{1}{kh})(h - \log_k h) \\ &\geq 2n(1 - \frac{1}{2h})(h - \log_k h) \\ &= 2nh - 2n\log_k h - n + n\frac{\log_k h}{h} \\ &\geq 2nh - 2n\log_k h - n. \end{aligned} \tag{6.8}$$

If $w < q = 2h - 2\log_k h - 1$, then (6.8) gives

$$\begin{aligned} T &\geq \frac{2nh - 2n\log_k h - n}{2h - 2\log_k h - 2} \\ &= n + \frac{n}{2h - 2\log_k h - 2}. \end{aligned}$$

Noting that $h = \log_k n$, and after a bit of algebra we obtain $T > n - 1 + q$. On the other hand, if $w > q$ then from (6.7) $T \geq n - 1 + w > n - 1 + q$. Consequently, the minimum time can be had only for $w = q$ and in that case

$$T = n - 1 + q = n + 2h - 2\log_k h - 2 \text{ steps.} \tag{6.9}$$

Theorem 6.10 *An optimal total exchange algorithm for exponential capacity trees needs at least* $n + 2\log_k n - 2\log_k \log_k n - 2$ *steps.*

Proof. This is an immediate consequence of the facts that a total exchange algorithm for the trees of interest is a total exchange algorithm for G_i and that for G_i a lower bound is given by (6.9). ■

6.3.4 Discussion

The results for the multiport model are summarized in Table 6.2. The preceding analyses were based on the assumption that $c_1 = 1$. It is expected that some of the problems require

Problem	**Time**
Broadcasting	$2\log_k n - 1$
Scattering/Gathering	n (if $k \geq 3$) , $n+1$ (if $k = 2$)
Multinode Broadcasting	n (if $k \geq 3$) , $n+1$ (if $k = 2$)
Total Exchange*	$\geq n^2(k-1)/(k^2 c_h) + 2\log_k n - 1$

*see text

Table 6.2. *Time requirements for communications under the multiport model*

less steps if the first level branches consist of more than one link; it would be interesting to see what the exact effect of c_1 is in this case.

For the total exchange problem, the bound given in Table 6.2 is general but provably not tight in some cases. What we have shown here is that for complete k-ary trees, total exchange can be performed in

$$n^2\frac{k-1}{k^2} + 2\log_k n - 1 \leq TE(n) \leq (n^2-1)\frac{k-1}{k^2-1} + 2\log_k n - 1,$$

and for trees with exponentially growing capacities,

$$n + 2\log_k n - 2\log_k \log_k n - 2 \leq TE(n) \leq n + 2\log_k n - 2.$$

In conclusion, in this chapter we studied the implementation and the performance of communication operations in k-ary fat trees where the processing nodes are confined to the leaf level, considering both the single-port and the multiport models. The results can actually be generalized to the case where nodes in different levels can have a different number of children, that is level i nodes having k_i children, where $n = k_1 k_2 \cdots k_h$. Under the multiport model, branch capacities as used in fat tree networks have a beneficial role in high-volume communications such as total exchange. If the processing nodes are connected through a single link with the routing network, capacities are of no importance (at least with respect to speed) in the rest of the communication problems we studied.

There are still a number of issues to be considered. Multinode broadcasting has excessive queueing requirements if it is to be performed in the minimum number of steps. The second algorithm we presented for this problem avoids contention but requires $2\log_k n - 2$ more steps. It is still an open question whether this is optimal or contention can be avoided in less steps. Another issue for consideration is total exchange under the multiport model. The algorithm we presented is close to optimal especially in the case of exponential capacities, where we showed that it is within $2\log_k \log_k n$ steps from the lower bound. Improved bounds though for the total exchange problem need to be found as the straightforward one does not seem to be tight.

Chapter 7

Conclusion

The subject of this thesis has been certain information dissemination problems related to multiprocessor interconnection networks. Apart from the need for one node to communicate with one other node, it has been observed that real-world applications require communications of the one-to-all and all-to-all type. Broadcasting, multinode broadcasting, scattering, gathering and total exchange constitute a set of representative problems for such communications, appearing quite frequently in practice.

In distributed-memory multiprocessors, cooperation between processes residing in different nodes occurs through messages passed over the interconnection network. The time spent communicating among processors can be substantial. Efficient solution of the communication problems we have considered is thus of paramount importance to system performance. This is also evident by the inclusion of collective communication operations in the Message Passing Interface [49, 22], an emerging standard for communication routines in message passing programs, and by their role in supporting various programming constructs in High Performance Fortran [31, 45].

Studies on many of the problems we considered have appeared relatively recently [59]. However, multinode broadcasting in the form of *gossiping* appeared as a problem in 1950, while broadcasting under the single-port model was introduced in 1977 (see [30] and the references therein). These early studies, especially for gossiping, were based on a model not particularly suited for multiprocessors. The usual objective was to minimize the number of communication "rounds", independently of the number of messages transferred between neighbors in a single round. In our case however we are interested in determining or minimizing the *time* required to complete an operation.

Under this objective, we saw that in the multiport model the time required for the broadcasting problem is governed by the diameter of the network. If the single-port model is in effect, there has been found no single characteristic of the network that determines the time needed for broadcasting. If the network has n nodes and diameter D then all that was

known is that the broadcasting time T is bounded below by $T \geq \log_2 n$ and $T \geq D$. If at least two nodes exist at distance D from the source node then $T \geq D+1$ was an additional result in [25]. Here, we offered a general bound based of the number of nodes R_d in distance d or more form the source node. Specifically, we showed that $\sum_{\ell=0}^{T} \binom{T}{\ell} \geq R_d$. The other known bounds were seen to become special cases of our result.

Scattering (and its dual problem, gathering) was the second single-source (or one-to-all) problem we studied. In this problem, one node has to send distinct items of information to every other node in the network. We surveyed the known results in the literature and we provided time-optimal scattering constructions for extended rings and two-dimensional tori. Under the single-port model, the required time is governed by the number of nodes in the network. In particular, we provided a formal proof of the fact that $n-1$ steps are always enough to complete scattering in an n-node network.

In the total exchange problem, every node performs a scattering. Under the multiport model, we provided the first known algorithms to provably achieve the lower bound on total exchange time in linear arrays and rings. Other algorithms have appeared for rings in the literature but were optimal only for odd rings while ours work for every ring.

Based on a type of algorithms which we called *node-invariant* algorithms, and which we initially introduced for rings, we were able to solve the single-port total exchange problem in a large class of node symmetric graphs. This is the class of Cayley networks which includes important topologies such as rings, extended rings, complete graphs, hypercubes, cube-connected cycles, butterflies. We only know of two other optimal single-port algorithms; one designed solely for hypercubes [5, pp. 81–83] and the other for star graphs [50].

In this thesis, we took a particular interest in multidimensional networks, including hypercubes, meshes, tori and hypercycles, since they have been prevailing in the interconnection network theory for multiprocessors, and because many of the commercially available parallel machines are based on them [69]. We developed multiple node-disjoint paths between two pairs of vertices in Section 2.2 and we derived a broadcast tree in Section 3.1.1, applicable to any multidimensional network. Total exchange in such networks was the sole subject of Chapter 5. We derived a formula for the status of vertices and we used it to obtain optimality conditions for a general total exchange algorithm we developed. This algorithm decomposed the problem of total exchange in multidimensional networks to that of total exchange in single dimensions. One of the conclusions was that as long as we have single-dimension total exchange algorithms that achieve the lower bound of the average status, then we can synthesize an optimal algorithm for the whole network (under the single-port model). A corresponding conclusion was reached when we extended the algorithm to ho-

mogeneous graphs with 2^k dimensions and multiport capabilities. In effect, the theory we developed provided optimal solutions for many popular interconnection networks.

We finally took a close look at fat tree networks, which constitute a quite promising interconnection structure. Fat trees were found to be different in many aspects from all the other networks we considered. We thus derived specialized bounds and algorithms for them. An optimal set of algorithms for broadcasting, scattering, multinode broadcasting and total exchange was designed for the single-port model. Under the multiport model we solved optimally the broadcasting, scattering and multinode broadcasting problems. We also suggested a total exchange algorithm that runs in time very close to a certain (non-tight) lower bound, especially for trees with exponentially growing capacities.

We would like to finish this section with a statement by Hedetniemi, Hedetniemi and Liestman [30]:

"... First the subjects of broadcasting, gossiping and related information dissemination processes are terribly rich in real-world applications.

... And finally, as the results in this survey suggest, our understanding of this subject [information dissemination] at present is at best primitive."

It is the hope of this author that this thesis will contribute to the understanding of information dissemination, at least for the five communication problems considered.

7.1 Open Problems and Future Directions

There are a number of problems related to the subject of this thesis which we believe will lead to fruitful future research. Starting with single-port broadcasting, it should be clear that the general insight is very limited. Actually, there is no known result which gives the optimal broadcast time even for well-studied networks such as tori (with more than two dimensions).

Little is known for scattering under the multiport model, too. For many interesting networks the sufficient condition we mentioned in Section 3.4 seems enough. Two solutions are known for hypercubes. For tori we provided a general construction in two dimensions. We also have some evidence that balanced trees can be constructed for d-dimensional tori ($d \geq 2$) but only if they are homogeneous and consist of an odd number of nodes (i.e. the class of k-ary n-cubes with odd k). However, constructions for general multidimensional tori are lacking and constitute an interesting subject for research.

Also, we seek to determine new sufficient or necessary conditions for the optimal scattering time. One observation we made during the course of this research is that the presence of

cut-nodes/edges (i.e. nodes/edges whose removal disconnects the network) may limit substantially the number of messages that can be simultaneously released from the source node and can also make the use of spanning trees a suboptimal technique. We are interested in the exact connection between scattering and cut-nodes/edges in the graph.

Total exchange in many networks is still an open problem. We actually provided optimal solutions for a large number of interesting networks but not every possible topology was covered. The most prominent problem in our opinion is total exchange in multidimensional networks under the multiport model. Our theory for this communication model applies only to power-of-two-dimensional networks. It is not clear yet how the theory can be extended to networks with any number of dimensions. It is one of our future plans to continue our work on this problem.

Finally, it would be interesting to see if the multiport total exchange algorithm we presented for fat trees is optimal or not. We recall that it was shown that its running time is within $2\log_k \log_k n$ steps from a certain lower bound. We conjecture that we cannot do better than that and as part of our future work, we hope it can be actually proven. Also, another interesting issue is the development of exact bounds and algorithms for the case where the capacities of the first-level links are greater than unity.

The analysis in this thesis was based on the assumption that the network is packet-switched. Recently, some works have appeared on communication problems as applied to wormhole-routed/circuit-switched machines (a survey is given in [48]). Under wormhole routing and circuit switching most of our analysis is not applicable since in these cases the distance between two nodes plays no significant role (as long as there is no link contention). Most results are however on the preliminary stage and only for special topologies (mainly hypercubes and two-dimensional tori). This is one particularly appealing area of future research.

Bibliography

[1] S. B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," *IEEE Transactions on Computers*, Vol. 38, No. 4, pp. 555–566, Apr. 1989.

[2] F. Annexstein, M. Baumslag and A. L. Rosenberg, "Group action graphs and parallel architectures," *SIAM Journal of Computing*, Vol. 19, No. 3, pp. 544–569, June 1990.

[3] J. - C. Bermond, P. Hell, A. L. Liestman and J. G. Peters, "Broadcasting in bounded degree graphs," *SIAM Journal of Discrete Mathematics*, Vol. 5, No. 1, pp. 10–24, Feb. 1992.

[4] D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng and J. N. Tsitsiklis, "Optimal communication algorithms for hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 11, pp. 263–275, 1991.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewoods Cliffs, N.J.: Prentice - Hall, 1989.

[6] S. N. Bhatt, G. Pucci, A. Ranade and A. L. Rosenberg, "Scattering and gathering messages in networks of processors," *IEEE Transactions on Computers*, Vol. 42, No. 8, pp. 938–949, Aug. 1993.

[7] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, Vol. C-33, No. 4, pp. 323–333, Apr. 1984.

[8] N. Biggs, *Algebraic Graph Theory (2nd edition)*. Cambridge, G.B.: Cambridge University Press, 1993.

[9] N. L. Biggs, *Discrete Mathematics (revised edition)*. New York: Oxford University Press, 1989.

[10] F. Boesch and R. Tindell, "Circulants and their connectivities," *Journal of Graph Theory*, Vol. 8, pp. 487–499, 1984.

[11] S. H. Bokhari, "Multiphase complete exchange on a circuit switched hypercube," in *Proc. 1991 International Conference on Parallel Processing*, Aug. 1991, pp. I-525 – I-529.

[12] F. Buckley and F. Harary, *Distance in Graphs*. Reading, Mass.: Addison - Wesley, 1990.

[13] G. E. Carlsson, J. E. Cruthirds, H. B. Sexton and C. G. Wright, "Interconnection networks based on a generalization of cube-connected cycles," *IEEE Transactions on Computers*, Vol. C-34, No. 8, pp. 769–772, Aug. 1985.

[14] Committee on Physical, Mathematical and Engineering Sciences Federal Coordinating Council for Science, Engineering and Technology, "High-performance computing and communications, Grand Challenges 1993 report," Washington, D.C., 1993.

[15] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, Vol. 39, No. 6, pp. 775–785, June 1990.

[16] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547–553, May 1987.

[17] V. V. Dimakopoulos, "Processor allocation and message broadcasting in hypercycle interconnection networks," Master of Applied Science thesis, Department of Electrical and Computer Engineering, University of Victoria, 1992.

[18] V. V. Dimakopoulos and N. J. Dimopoulos, "Leaf communications in complete trees," Technical Report ECE-95-6, University of Victoria, Oct. 1995.

[19] N. J. Dimopoulos, "On the structure of the Homogeneous multiprocessor," *IEEE Transactions on Computers*, Vol. C-34, No. 2, pp. 141–150, Feb. 1985.

[20] N. J. Dimopoulos, R. Sivakumar, V. V. Dimakopoulos, M. Chowdhury and D. Radvan, "Hypercycles: A Status Report," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing,*, Victoria, B.C., Canada, May 1991, pp. 111–114.

[21] J. J. Dongarra and T. Dunigan, "Message-passing performance of various computers," Technical Report CS-95-299, University of Tennessee, Aug. 1995.

[22] J. J. Dongarra, S. W. Otto, M. Snir and D. Walker, "An introduction to the MPI standard," Technical Report CS-95-274, University of Tennessee, Apr. 1995.

[23] R. Feldmann, J. Hromkovic, S. Madhavapeddy, B. Monien and P. Mysliwietz, "Optimal algorithms for dissemination of information in generalized communication modes," in *Proc. 4th PARLE, Parallel Architectures and Languages Europe*, Paris, France, June 1992, pp. 115–130.

[24] P. Fraigniaud, "Complexity analysis of broadcasting in hypercubes with restricted communication capabilities," *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 15–26, 1992.

[25] P. Fraigniaud and E. Lazard, "Methods and problems of communication in usual networks," *Discrete Applied Mathematics*, Vol. 53, pp. 79–133, 1994.

[26] P. Fraigniaud, S. Miguet and Y. Robert, "Scattering on a ring of processors," *Parallel Computing*, Vol. 13, No. 3, pp. 377–383, 1990.

[27] D. B. Gannon and J. van Rosendale, "On the impact of communication complexity on the design of parallel numerical algorithms," *IEEE Transactions on Computers*, Vol. C-33, No. 12, pp. 1180–1194, Dec. 1984.

[28] A. Gibbons, *Algorithmic Graph Theory*. Cambridge, G.B.: Cambridge University Press, 1985.

[29] M. Grini and D. Peleg, "Tight bounds on minimum broadcast networks," *SIAM Journal of Discrete Mathematics*, Vol. 4, No. 2, pp. 207–222, May 1991.

[30] S. M. Hedetniemi, S. T. Hedetniemi and A. L. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, Vol. 18, pp. 319–349, 1988.

[31] S. Hiranandani, K. Kennedy and C. - W. Tseng, "Compiling Fortran D for MIMD distributed-memory machines," *Communications of the ACM*, Vol. 35, No. 8, pp. 66–80, Aug. 1992.

[32] C. - T. Ho and S. L. Johnsson, "Spanning balanced trees in boolean cubes," *SIAM Journal of Scientific and Statistical Computing*, Vol. 10, No. 4, pp. 607–630, July 1989.

[33] C. - T. Ho and M. T. Raghunath, "Efficient communication primitives on circuit-switched hypercubes," in *Proc. 6th Distributed Memory Computing Conference*, Portland, Oregon, Apr. 1991, pp. 390–397.

[34] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.

[35] S. L. Johnsson, "Communication efficient basic linear algebra computations on hypercube architectures," *Journal of Parallel and Distributed Computing*, Vol. 4, pp. 133–172, 1987.

[36] S. L. Johnsson and C. - T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, Vol. 38, No. 9, pp. 1249–1268, 1989.

[37] S. L. Johnsson and C. - T. Ho, "Optimal all-to-all personalized communication with minimum span on boolean cubes," in *Proc. 6th Distributed Memory Computing Conference*, Portland, Oregon, Apr. 1991, pp. 299–304.

[38] R. Klasing, B. Monien, R. Peine and E. A. Stöhr, "Broadcasting in butterfly and deBruijn networks," *Discrete Applied Mathematics*, Vol. 53, pp. 183–197, 1994.

[39] S. H. C. Kosak, D. R. O'Hallaron, T. M. Stricker and R. Take, "An architecture for optimal all-to-all personalized communication," in *Proc. 6th ACM Symposium on Parallel Algorithms and Architectures*, New Jersey, June 1994, *to appear.*

[40] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Diego, CA: Morgan Kaufmann, 1992.

[41] C. E. Leiserson *et al*, "The network architecture of the Connection Machine CM-5," in *Proc. 4th ACM Symposium on Parallel Algorithms and Architectures*, June 1992, pp.

272–285.

[42] A. L. Liestman and J. G. Peters, "Broadcast networks of bounded degree," *SIAM Journal of Discrete Mathematics*, Vol. 1, No. 4, pp. 531–540, Nov. 1988.

[43] X. Lin and L. M. Ni, "Multicast communication in multicomputer networks," in *Proc. 1990 International Conference on Parallel Processing*, 1990, pp. 114–118.

[44] C. L. Liu, *Introduction to combinatorial mathematics*. New York: McGraw-Hill, 1968.

[45] D. B. Loveman, "High Performance Fortran," *IEEE Parallel and Distributed Technology*, Vol. 1, pp. 25–42, Feb. 1993.

[46] O. M. Lubeck, "Supercomputer performance: the theory, practice and results", in *Advances in Computers, Vol. 27*, M. C. Yovits ed., Academic Press, pp. 309–362, 1988.

[47] P. K. McKinley, H. Xu, A. - H. Esfahanian and L. M. Ni, "Unicast-based multicast communication in wormhole-routed networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 12, pp. 1252–1264, Dec. 1994.

[48] P. K. McKinley, Y. - J. Tsai and D. F. Robinson, "Collective communication in wormhole-routed massively parallel computers," *IEEE Computer*, Vol. 28, No. 12, pp. 39–50, Dec. 1995.

[49] Message Passing Interface Forum, "MPI: A message-passing interface standard," Technical Report CS-94-230, University of Tennessee, Apr. 1994.

[50] J. Misŝić and Z. Jovanović, "Communication aspects of the star graph interconnection network," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 7, pp. 678–687, July 1994.

[51] D. S. Mitrinović, *Elementary inequalities*. Groningen, Netherlands: P. Noordhoff, 1964.

[52] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, Vol. 26, No. 2, pp. 62–76, Feb. 1993.

[53] J. G. Peters and M. Syska, "Circuit-switched broadcasting in torus networks," Technical Report CMPT TR 93-04, Simon Fraser University, May 1993.

[54] R. Ponnusamy, R. Thakur, A. Choudhary and G. Fox, "Scheduling regular and irregular communication patterns on the CM-5," in *Proc. Supercomputing '92*, Minneapolis, Nov. 1992, pp. 394–402.

[55] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Communications of the ACM*, Vol. 24, No. 5, pp. 300–309, May 1981.

[56] A. Proskurowski, "Minimum broadcast trees," *IEEE Transactions on Computers*, Vol. C-30, No. 5, pp. 363–366, May 1981.

[57] D. A. Reed and D. C. Grunwald, "The performance of multicomputer interconnection networks," *IEEE Computer*, Vol. 20, No. 6, pp. 63–73, June 1987.

[58] D. A. Reed and H. D. Schwetman, "Cost-performance bounds for multicomputer networks," *IEEE Transactions on Computers*, Vol. C-32, No. 1, pp. 83–95, Jan. 1983.

[59] Y. Saad and M. H. Schultz, "Data communications in hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 6, pp. 115–135, 1989.

[60] Y. Saad and M. H. Schultz, "Data communications in parallel architectures," *Parallel Computing*, Vol. 11, pp. 131–150, 1989.

[61] D. S. Scott, "Efficient all-to-all communication patterns in hypercube and mesh topologies," in *Proc. 6th Distributed Memory Computing Conference*, Portland, Oregon, Apr. 1991, pp. 398–403.

[62] G. Strang, *Linear Algebra and Its Applications*. San Diego, CA: Harcourt Brace Jovanovich, 1988 (3rd ed.).

[63] H. Sulivan and T. R. Bashkow, "A large scale, homogenous, fully distributed parallel machine, I," in *Proc. 4th International Symposium on Computer Architecture*, Apr. 1977, pp. 105–117.

[64] R. Thakur and A. Choudhary, "All-to-all communication on meshes with wormhole routing," in *Proc. 8th International Parallel Processing Symposium*, Cancun, Mexico, Apr. 1994, pp. 561–565.

[65] D. M. Topkins, "Concurrent broadcast for information dissemination," *IEEE Transactions on Software Engineering*, Vol. 11, No. 10, pp. 1107–1112, Oct. 1985.

[66] R. W. Topor, "Termination detection for distributed computations," *Information Processing Letters*, Vol. 18, pp. 33–36, Jan. 1984.

[67] E. A. Varvarigos and D. P. Bertsekas, "Communication algorithms for isotropic tasks in hypercubes and wraparound meshes," *Parallel Computing*, Vol. 18, pp. 1233–1257, 1992.

[68] H. Xu, P. K. McKinley and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 172–184, 1992.

[69] G. Zorpette, "Supercomputers: reinventing the machine," *IEEE Spectrum*, Vol. 29, No. 9, pp. 28–41, Sept. 1992.

VITA

Surname: Dimakopoulos ***Given Names:*** Vassilios, Vlassios
Place of Birth: Patras, Greece ***Date of Birth:*** Oct. 23, 1967

Educational Institutions Attended

University of Victoria	1990 to 1996
University of Patras, Greece	1985 to 1990

Degrees Awarded

M.A.Sc.	University of Victoria	1992
Diploma	University of Patras	1990

Honors and Awards

University of Victoria Fellowship	1990 to 1995
Charles S. Humphrey Graduate Award	Oct. 1994
Howard Petch Graduate Scholarship	Oct. 1993
B.C. Advanced Systems Institute (ASI) Graduate Scholarship	Oct. 1992
University of Victoria Teaching Assistant Award	May 1991

Journal Publications

1. V.V. Dimakopoulos, G. Sourtziotis, A. Paschalis and D. Nikolos, "On TSC checkers for m-out-of-n codes", *IEEE Transactions on Computers*, Vol. 44, No. 8, pp. 1055–1059, Aug. 1995.

2. N.J. Dimopoulos and V.V. Dimakopoulos, "Optimal and suboptimal processor allocation for hypercycle-based multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 2, pp. 175–185, Feb. 1995.

3. V.V. Dimakopoulos and N.J. Dimopoulos, "Broadcasting in hypercycles", *accepted*, *Computer Systems Science and Engineering* (to appear).

4. V.V. Dimakopoulos and N.J. Dimopoulos, "Optimal total exchange in linear arrays and rings", *accepted subject to modifications*, Parallel Computing.

Conference Publications

1. V.V. Dimakopoulos and N.J. Dimopoulos, "Communications in binary fat trees", *Proceedings PDCS '95, International Conference on Parallel and Distributed Computing Systems*, Florida, pp. 383–388, Sept. 1995.

2. R. Sivakumar, V.V. Dimakopoulos and N.J. Dimopoulos, "Design of a programmable controller for hypercycle based interconnection networks", *FPD'95, the 3rd Canadian Workshop on Field-Programmable Devices*, Montreal, Quebec, Canada, pp. 212–217, May 1995.
3. V.V. Dimakopoulos and N.J. Dimopoulos, "Optimal total exchange in linear arrays and rings", *Proceedings ISPAN '94, International Symposium on Parallel Architectures, Algorithms and Networks*, Kanazawa, Japan, pp. 230–237, Dec. 1994.
4. N.J. Dimopoulos, M. Chowdhury, R. Sivakumar and V.V. Dimakopoulos, "Routing in hypercycles: deadlock free and backtracking strategies", *Proceedings PARLE '92, Parallel Architectures and Languages Europe*, Paris, France, pp. 973–974, June 1992.
5. R. Sivakumar, N.J. Dimopoulos, V.V. Dimakopoulos, M. Chowdhury and D. Radvan, "Implementation of the routing engine for hypercycle based interconnection networks", *Proceedings CCVLSI '91, Canadian Conference on VLSI*, Kingston, Ont., pp. 6.4.1–6.4.7, Aug. 1991.
6. N.J. Dimopoulos, R. Sivakumar, V.V. Dimakopoulos, M. Chowdhury and D. Radvan, "Hypercycles: a status report", *Proceedings of the 1991 IEEE Pacific Rim Conference*, Victoria, B.C., pp. 111–114, May 1991.

Work Submitted for Publication

1. V.V. Dimakopoulos and N.J. Dimopoulos, "Optimal total exchange in Cayley graphs", *submitted to* SIAM Journal on Computing.
2. V.V. Dimakopoulos and N.J. Dimopoulos, "A theory for total exchange in multidimensional networks", *submitted to* IEEE Transactions on Parallel and Distributed Systems.
3. V.V. Dimakopoulos and N.J. Dimopoulos, "Leaf communications in complete trees", *submitted to* Journal of Parallel and Distributed Computing.
4. V.V. Dimakopoulos and N.J. Dimopoulos, "On broadcasting time", *submitted to* Parallel Processing Letters.

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my dissertation to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this dissertation for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this dissertation for financial gain shall not be allowed without my written permission.

Title of Dissertation: COLLECTIVE COMMUNICATION PROBLEMS IN MULTIPROCESSORS.

Author: _____

VASSILIOS V. DIMAKOPOULOS

March 1, 1996