2000

Multiple Bus Networks for Binary -Tree Algorithms.

Hettihewage Prasanna Dharmasena
Louisiana State University and Agricultural & Mechanical College

MULTIPLE BUS NETWORKS FOR BINARY-TREE ALGORITHMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
H. P. Dharmasena
B.S., University of Moratuwa, 1983
M.S., Louisiana State University, 1987
May 2000

UMI Number: 9979254

UMI®

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgments

I would like to express my sincere gratitude to Dr. R. Vaidyanathan for his guidance, wisdom and especially his patience during the course of this research. I would also like to thank members of my committee Dr. S. Kundu, Dr. A. El-Amawy, Dr. J. L. Trahan, Dr. K. Zhou and Dr. L. J. Smolinsky.

This work would have been impossible without the support of various individuals at my work. I wish to express my sincere appreciation to the members of the instrument development group for their understanding and support.

Table of Contents

ACKNOWLEDGMENTS ii

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT x

CHAPTER

1 INTRODUCTION 1

1.1 MBNs and Binary-Tree Algorithms 4

1.2 Scope of the Dissertation 6

1.3 Contribution of this Work 9

1.4 Organization o: the Dissertation 10

2 PRELIMINARIES 11

2.1 Binary Tree Algorithms 11

2.2 Multiple Bus Networks 13

2.3 Running Binary Tree Algorithms on MBNs 14

2.3.1 Direct and Indirect Mapping 16

2.4 Prefix Computations on Binary-Tree MBNs 17

3 DEGREE, LOADING, TIME TRADE-OFFS 20

3.1 Preliminaries 21

3.2 Lower Bound for Direct Mapping 22

3.3 An $\Omega(\sqrt{n})$ Lower Bound 24

3.3.1 Strategy and Definitions 24

3.3.2 Basic Results 26

3.3.3 The Accounting Scheme 27

3.3.4 Non-Uniform Bus Usage 30

3.3.5 The Lower Bound 31

3.4 An $\Omega\left(n^{\frac{2}{3}}\right)$ Lower Bound 32

3.4.1 Additional Results 32

3.4.2 Tighter Lower Bound 35

3.5 An $\Omega\left(\frac{n}{\log n}\right)$ Lower Bound 37

3.5.1 Initial Condition 38

3.5.2 The New Accounting Scheme 39
3.5.3 Tighter Lower Bound 43
3.6 The Tree MBN 44
3.7 Loading-Speed Tradeoff 47
3.7.1 Lower Bound 47
3.7.2 Upper Bound 49
3.8 Extension to k-ary Tree Algorithms 52
3.9 Concluding Remarks 58

4 Multiple-Bus Enhanced Meshes 59
4.1 Preliminaries 63
4.1.1 MBN Measures 64
4.1.2 Multiple-Bus Enhanced Meshes 65
4.2 Binary-Tree MBN Extensions 66
4.3 Meshes with Tree MBNs 69
4.4 MBEMs with Segment Switches 74
4.4.1 Binary-Tree MBNs with Segment Switches 74
4.4.2 Meshes with Tree MBNs and Segment Switches 80
4.5 Results and Discussion 83

5 Fault Tolerance 84
5.1 Fault Model 85
5.2 Replication 86
5.2.1 Adding Redundant Connections 87
5.2.2 Definition of $\mathcal{R}_k$ 87
5.2.3 The Designated Set 89
5.2.4 Fault Tolerance Properties of $\mathcal{R}_k$ 95
5.2.5 Processor Faults 96
5.2.6 Fault Tolerant Binary-Tree MBNs 97
5.3 Recursive Scheduling 101
5.3.1 An MBN, $\mathcal{M}'_1$, with $2^m - 1$ Buses 102
5.3.2 Recursive Scheduling with 2^s Bus Faults 110
5.3.3 Putting it All Together 111
5.4 Comparison of Results 112
5.5 Concluding Remarks 113

6 VLSI Layout Lower Bound 115
6.1 Preliminaries 118
6.1.1 VLSI Model 118
6.1.2 Definitions and Figure Conventions 119
6.2 Towards the Lower Bound 124
6.2.1 Minimum Communication Structure 124
6.2.2 Labeling Links 129

6.2.3 Collapsing Links 130

7 Summary and Future Work 136

Bibliography 140

Vita 147

List of Tables

4.1 Some results for meshes with Tree MBN 73

4.2 Some results for meshes with Tree MBN and Segment Switches 82

5.1 Summary of results 114

List of Figures

2.1 Running $Bin(3)$ on an 8×4 MBN 12

2.2 A 16×8 MBN and its matrix 14

2.3 Steps of running prefix computation on binary-tree MBNs 18

3.1 $\mathcal{F}(n)$ with a direct mapping 23

3.2 Step 1 of $X_1(n)$ 25

3.3 Processors and buses in the neighborhood of bus b_0 36

3.4 Subintervals of $[L, n]$ 37

3.5 The connections on bus b_0 44

3.6 Running $Bin(4)$ on $\mathcal{T}(4)$ 46

3.7 Path with t delays 48

3.8 Running $Bin(3)$ on 8×4 MBN, $\mathcal{D}(3)$ 50

3.9 MBN for ternary tree algorithms 58

4.1 A 32×8 Tree MBN 68

4.2 Structure of a mesh with binary-tree MBN 71

4.3 Steps of method 1 76

4.4 Steps of the method 2 77

4.5 Construction 1 78

4.6 Construction 2 79

5.1 The MBN of Figure 2.2 augmented to handle 3 bus faults 88

5.2 Graph $\mathcal{G}_{3,8}$. 90

5.3 An illustration of the proof of Lemma 5.1. 92

5.4 Node disjoint correspondences for an example 95

5.5 Regions of $\mathcal{F}(n)$ 103

5.6 Recursive decomposition of $\mathcal{F}(n)$ 104

5.7 Running the first step of Region 3 105

5.8 An example of a 8 × 4 MBN 105

5.9 Connections of processors and buses with one bus fault 106

5.10 Connection of processors and buses in Region 3 108

5.11 Regions of $\mathcal{F}(n)$ for k bus faults 110

6.1 H-Tree layout of a 31-processor binary tree 116

6.2 7-node binary tree layout 116

6.3 8-processor MBN layout 117

6.4 8 processor MBN running $Bin(3)$ 117

6.5 A perimeter layout 119

6.6 Links between processors 120

6.7 View from processor 1 120

6.8 Subset view I 121

6.9 Subset view II 122

6.10 Subset view III 122

6.11 Subset view from processor 1 122

6.12 Subset view from processor 4 123

6.13 Equivalent views 123

6.14 Symbolic representation of a subset view 123

6.15 Communication Structure for $\mathcal{F}(3)$ 124

6.16 Subcase 1(a) 125

6.17 Subcase 1(b): p_2-p_1-p_0 link 126

6.18 Subcase 1(b): p_2-p_3-p_0 link 126

6.19 Case 2 127

6.20 Case 2: p_2-p_1-p_0 link 127

6.21 Case 2: p_2-p_3-p_0 link 127

6.22 Subcase 3(a) 128

6.23 Subcase 3(b): p_2-p_1-p_0 link 128

6.24 Subcase 3(b): p_2-p_3-p_0 link 129

6.25 $\mathcal{F}(n)$ 130

6.26 View from final result processor 130

6.27 A maximal collapse of a structure with 4 levels 132

6.28 A different collapse 134

Abstract

Multiple bus networks (MBN) connect processors via buses. This dissertation addresses issues related to running binary-tree algorithms on MBNs. These algorithms are of a fundamental nature, and reduce inputs at leaves of a binary tree to a result at the root. We study the relationships between running time, degree (maximum number of connections per processor) and loading (maximum number of connections per bus). We also investigate fault-tolerance, meshes enhanced with MBNs, and VLSI layouts for binary-tree MBNs.

We prove that the loading of optimal-time, degree-2, binary-tree MBNs is non-constant. In establishing this result, we derive three loading lower bounds $\Omega(\sqrt{n})$, $\Omega(n^{\frac{2}{3}})$ and $\Omega(\frac{n}{\log n})$, each tighter than the previous one. We also show that if the degree is increased to 3, then the loading can be a constant. A constant loading degree-2 MBN exists, if the algorithm is allowed to run slower than the optimal.

We introduce a new enhanced mesh architecture (employing binary-tree MBNs) that captures features of all existing enhanced meshes. This architecture is more flexible, allowing all existing enhanced mesh results to be ported to a more implementable platform.

We present two methods for imparting tolerance to bus and processor faults in binary-tree MBNs. One of the methods is general, and can be used with any MBN and for both processor and bus faults. A key feature of this method is that it permits the network designer to designate a set of buses as “unimportant” and consider all faulty buses as unimportant. This minimizes the impact of faulty elements on the MBN. The second method is specific to bus faults in binary-tree MBNs, whose features it exploits to produce faster solutions.

We also derive a series of results that distill the lower bound on the perimeter layout area of optimal-time, binary-tree MBNs to a single conjecture. Based on this we believe that optimal-time, binary-tree MBNs require no less area than a balanced tree topology even though such MBNs can reuse buses over various steps of the algorithm.

Chapter 1

Introduction

In a parallel processing system, the interprocessor communication network plays a very important role. In this dissertation, we deal with one class of such networks called *multiple bus networks* (*MBNs*). An MBN consists of a set of processors and a set of buses, with each processor connected to at least one bus. Any processor connected to a bus can access the bus. However, the bus can convey only one piece of information at a time.

MBNs have several advantages over traditional point-to-point networks (such as the ring, mesh, torus and hypercube). In a point-to-point network, each communication link is dedicated to a pair of processors. In an MBN, on the other hand, the communication medium (bus) is shared among several processors and could, therefore, be used more efficiently. This sharing of the communication medium also allows for a graceful degradation of performance in the presence of faults. Because the communication medium is shared, MBNs lend themselves to easy broadcasting. An MBN can be used to emulate several point-to-point topologies or set of interconnection functions [26, 27, 33, 47, 48] as each bus could serve as a communication link between different processor pairs at different times. An MBN is representative of any hypergraph based system [6], and a bus can be viewed as an abstraction of any shared resource, for example a memory module in shared memory systems, or a transmission frequency in systems with frequency division multiplexing (such as wireless [18, 22, 35, 52, 61] and optical [5, 20, 65]). Therefore, this work may find applicability in other settings as well.

Traditionally, MBNs have been used in an asynchronous environment with relatively few processors. Most of this work has been on analyzing data throughput of multiprocessor systems under various traffic models, arbitration schemes, and relationship between numbers of processors/buses [12, 14, 21, 28, 32, 36, 55, 56, 66, 94]. Work also exists on variations on the basic MBN model [9, 16, 37, 39, 43, 53, 90] and on the pattern of connections between processors and buses [12, 31, 39, 42, 54, 81]. Traditionally, MBNs have been used in asynchronous systems with a small number of processors partly because of the fact that physical loading due to capacitive coupling limits the number of connections to a bus. In an optical bus, loading is caused by a receiver on the bus drawing some of the available power, thus limiting the number of receivers that can be connected to the bus. The asynchronous bus model also requires a complex arbitration scheme to resolve bus contention.

In this dissertation we primarily consider a synchronous bus model, though most results apply to asynchronous MBNs as well. Technological advances have made it feasible to connect more loads on a bus. This, in turn, makes fine-grained synchronous MBNs (with a large number of processors) possible. The synchronous environment also removes the need for arbitration. Feldman *et al.* [29] recently proposed an optical slab waveguide bus capable of connecting a large number of processors at very high data rates. Qiao and Melhem [70] proposed a communication scheme called time-division source-oriented multiplexing (TDSM) for synchronous optical buses that can be used for large systems. Their method takes advantage of unidirectional propagation and predictable delay of optical fibers to achieve reliable communication among a large number of processors. Lin *et al.* [51] have proposed "precharged" buses to facilitate concurrent broadcasts.

Much work on synchronous MBNs has centered around topologies (primarily the two-dimensional mesh) enhanced with buses (for example [1, 4, 10, 17, 19, 64, 69, 71, 75]). There has also been some work on running algorithm classes and implementing interconnection functions [2, 23, 24, 25, 26, 40, 46, 47, 50, 63, 83, 85]. Another class of MBNs that uses synchronous buses is reconfigurable models, that allow the connection pattern between processors to change dynamically (Nakano [59] provides an extensive bibliography). Under current technological constraints, however, fixed connection MBNs, such as those considered in this work, are easier to implement than reconfigurable networks. Commercially available field programmable gate arrays (FPGAs) have also been proposed as reconfigurable computational platforms [34, 41, 58, 74, 88, 91, 92]. The programmable interconnections between "configurable logic blocks" in FPGAs show some features of MBNs in that they are often implemented as wires with taps (buses) [89, 93].

In this dissertation we address various issues related to running a well-known class of algorithms called *binary-tree algorithms* on MBNs. (Other researchers have also studied algorithm classes on MBNs and other networks [2, 40, 47, 62, 63, 73, 82].) A binary-tree algorithm reduces N inputs to a single result. The computation performed by such an algorithm can be represented as a balanced binary tree with the inputs at the leaves and the result at the root. Several fundamental algorithms involving semigroup operations and prefix computations such as maximum/minimum, parity, polynomial evaluation and barrier synchronization can be implemented as binary-tree algorithms. Binary-tree algorithms require a rich communication pattern, so a network suitable for running binary-tree algorithm is likely to be suitable for many other applications as well. Because of the fundamental nature of binary-tree algorithms, a dedicated hardware module to run these algorithms could aid solution to a large number of problems. Any insights gained by studying binary-tree algorithms will be useful in designing such modules. In the past, binary-tree algorithms and MBNs for them have been in setting of enhanced meshes [1, 4, 10, 19, 64, 69, 75, 76]. Other work on binary-tree MBNs addresses issues such as design, fault-tolerance and VLSI layouts [2, 25, 26, 27, 57, 63, 83, 84, 85].

1.1 MBNs and Binary-Tree Algorithms

In this section we present a broad picture of the issues related to MBNs running binary-tree algorithms (or "binary tree MBNs"). An $N \times M$ *Multiple Bus Network* MBN has N processors and M buses. Each processor is connected to a subset of the set of buses. Two processors may communicate in one unit of time, provided they are connected to a common bus. However, the bus may carry only one piece of information on it at any given point in time. Two important parameters of an MBN are its degree (maximum number of buses connected to a processor) and loading (maximum number of processors connected to a bus). These parameters determine the cost and implementability of the MBN. A large degree requires a processor to have a large number of input/output ports, while a large loading can reduce the data rate of the system.

One direction of research on MBNs considers a given pattern of interconnections between processors and buses and investigates the capabilities of the resulting MBN architecture. Often this takes the form of emulating other architectures (for example [26, 27, 40]) or developing algorithms on models ([52, 60, 61]); the enhanced mesh results cited earlier also fall in this category The second direction considers the problem of designing an MBN suited to a particular interconnection requirement. This dissertation and others [48, 57, 63] represent work in this direction.

As mentioned earlier, degree and loading are important considerations for MBNs. Clearly, the MBN should also be evaluated on how well it provides the interconnection requirements in question; this would consider issues such as number of hops and congestion on buses. Constructing an "optimal" MBN to run a given set of interconnection functions is a non-trivial task. Kulasinghe and El-Amawy [46] showed that the general problem of designing an optimal interconnecting network for a given set of interconnection functions is NP-Hard. The criteria they used for measuring the cost is the of number of buses and interfaces (connections between processors and buses). They showed [47] that this problem can be solved in polynomial time for certain "symmetric" interconnections, and presented a methodology for such implementations. Though such symmetries exist in interconnection topologies, it is not the case for many algorithm classes. Moreover, their analysis does not address the interplay between speed, degree and loading of the MBN.

With a single bus, the solution is simple as the only possibility is to connect all processors to the bus; this approach is used in most enhanced meshes and traditional multiprocessor systems. This method has the disadvantage of high loading and bus contention, limiting the size of the network. At the other extreme, all the processors could be connected to all buses. The Broadcast Communication Model (BCM) [60, 61] adopts this approach. This increases the loading and degree, and consequently, the cost of the MBN.

Thus an intermediate solution (that connects each processor to a subset of buses) is important. Optimal MBNs for binary-tree algorithms are particularly challenging to design. On one hand, for an N input algorithm the MBN needs sufficient bandwidth to sustain $\Theta(N)$ simultaneous communications; the lower (near leaf) levels of the tree involve a large number of simultaneous communications. On the other hand, because of its similarity to a binary tree the MBN should be fairly sparse. Thus, a small number of connections needs to be distributed over a large number of buses, lowering the acceptable values for both degree and loading. Most previous results have completely ignored degree and loading, or have reduced one at the expense of the other. For example Vaidyanathan and Padmanabhan [85] have proposed an N-input binary-tree algorithm that runs optimally in $\log N$ steps. Though the degree of this MBN is 2, its loading is $\Theta(\log N)$. On the other hand, Ragavendra [71] proposed a mesh with a hierarchy of broadcast buses in each row and column. For a given parameter k, this MBN has a loading k, but the degree of $O\left(\frac{\log N}{\log k}\right)$. Thus if the degree is small, the loading is large and vice versa. In this dissertation we construct an MBN that runs binary-tree algorithms optimally and which has both constant degree and loading. We now describe results of this dissertation in more detail.

1.2 Scope of the Dissertation

Because of its fundamental nature, binary-tree algorithms have been studied in almost all facets of computing. As mentioned earlier, most previous work on binary-tree MBNs has focused on enhanced mesh architectures. Very little work has been done on identifying fundamental properties of binary-tree MBNs and to establish relationships between running time, loading and the degree. In Chapter 3, we study these relationships, and establish lower-bounds on degree-2, binary-tree MBNs. We identify two important mappings and establish that it is essential to have a mapping called indirect mapping to achieve low loading. We do this by establishing a series of lower bounds on loading, each one tighter than the previous bound. Specifically, for a 2^n-input, optimal-time, binary-tree MBN we first prove the loading to be $\Omega(\sqrt{n})$. We then improve this bound to $\Omega(n^{\frac{2}{3}})$ by deriving some additional results. Finally the lower bound is further tightened to $\Omega(\frac{n}{\log n})$ by refining the method used to count connections on buses. (The lower bound restriction requires the MBN to have at least 2^{n-1} buses.) These lower bound results (and indeed most other results of this work) are general and apply to <u>any</u> binary-tree MBN satisfying the conditions of the problem, rather than a given MBN instance. Although a degree of 2 necessitates non-constant loading, this is not the case for degree 3. We construct a binary-tree MBN called the Tree MBN that has degree 3 and loading 3, which is the best possible.

Also in Chapter 3 we investigate trade-offs between the loading and running time. We show that if the running time is allowed to increase by a factor of 2, then a degree-2, binary-tree MBN with constant loading exists. We establish that if the additional time (beyond the optimal) used by the MBN is t, and if the largest problem that can be solved on a degree-2, loading-L, optimal-time MBN has size $2^{\tau(L)}$, then $t \geq \left\lfloor \frac{n}{\tau(L)+1} \right\rfloor$. We present an example of a degree-2, loading-4, $(2n-3)$-step binary-tree MBN that matches this bound for constant L.

Chapter 4 explores the idea of using binary-tree MBNs to enhance meshes. Here we show that an architecture using multiple buses has significant advantages over traditional enhanced meshes that employ single-bus networks to connect processor sets. We also study buses with segment switches (each of which can break a bus into two) and use it to reduce the loading. Other parameters of the proposed architecture can be selected for various trade-offs between the cost and performance. (Performance measures include running time, degree, loading, VLSI area and the aspect ratio; many exiting architectures require highly elongated rectangular layouts that are difficult to implement on a chip.) The architecture we propose improves on all previous results in at least one of the measures. It provides more choices to the network designer than any other architecture in the literature. Tables 4.1 and 4.2 (pages 73 and 82) summarize the results of this chapter.

In Chapter 5 we study methods for imparting fault tolerance to binary-tree MBNs. This complements the use of binary-tree MBNs as building blocks for general-purpose computing platforms (described in Chapter 4). Redundant connections can also be used to increase the yield for chips with binary-tree MBNs. In Chapter 5, we present two methods for constructing fault tolerant MBNs from any given binary-tree MBN. One of these methods (replication) is a general method that can be applied to processor and bus faults on any type of MBN. The other method (recursive scheduling) exploits features particular to binary-tree MBNs to produce better results, but handles only bus-faults. The general results of this chapter are too involved to state here; we state results for some particular cases instead. Replication constructs a binary-tree MBN that requires at most 5 extra steps, even if half the buses fail. Also, even if half the processors fail, the number of the extra steps required is at most 2.

In Chapter 6 we investigate the VLSI area required for optimal-time, binary-tree MBNs. The corresponding problem for the balanced tree topology is well studied [80]. The binary-tree algorithm is different from a balanced tree topology in that only one level of the tree is active (or used) in any step of a binary-tree algorithm. Therefore, binary-tree MBNs can reuse the same buses or wires at different levels. This is not possible in a balanced tree topology, where all edges could be active simultaneously. This raises the possibility that the VLSI area for a binary-tree MBN is less than that required for a balanced tree topology. We specifically consider the "perimeter layout" case where all the processors of the MBN are laid out on the periphery of the layout;

allowing the processors to be placed in the interior can trivially use a solution for the balanced tree topology. Our work on this topic leads us to conjecture that the perimeter area required for optimal-time N-input binary-tree MBNs is $\Omega(N \log N)$. Simulations seem to indicate that this conjecture is true.

1.3 Contribution of this Work

This dissertation studies various facets of running binary-tree algorithms on MBNs, providing a better understanding of the abilities and limitations of binary-tree MBNs. Most of our results are general in nature, applicable to any binary-tree MBN rather than particular cases. Many of these results extend to k-ary tree algorithms (for $k > 2$) as well.

Chapter 3 establishes important relationships between key parameters, namely running time, loading and degree. We develop a novel accounting scheme to keep track of the connections on a bus. It is possible that this method of counting connections may be useful in other algorithms as well. We also identify two mappings (called direct and indirect) of binary-tree algorithms on MBNs that impact the loading and degree of binary-tree MBNs. We show that indirect mapping is essential to achieving constant loading. Considering that indirect mapping increases the amount of communication, this result is rather counter-intuitive. Equally surprising is the result of Section 3.7 that shows that by increasing the running time by a constant factor, loading can be reduced by a non-constant factor.

In Chapter 4 we provide a general framework for connecting processors in a 2-dimensional mesh that, among other things, captures all the features of previous enhanced-mesh architectures but with a more realistic loading. Thus, our work provides the means to automatically translate all existing algorithms on enhanced meshes to a more implementable platform. In addition, our approach affords much more flexibility to the network designer than traditional methods.

The contribution of Chapter 5 is in providing a framework that adds redundancy in a controlled manner to convert any binary-tree MBN to one that is resilient to processor and bus faults. In particular, one of the methods, replication, works for any MBN (not just binary-tree MBNs) and uses an approach to rename elements and convert faulty components into ones that have the least impact on performance.

Although Chapter 6 does not derive a lower bound on the area, it distills the argument to a single conjecture. It also develops some satellite results (such as an 8-processor, optimal-time MBN with "one layer" of buses) that may have independent significance.

1.4 Organization of the Dissertation

In the next chapter we discuss some preliminary ideas and introduce some definitions. Chapter 3 deals with loading, running time and degree trade-offs. In Chapter 4 we describe meshes enhanced with binary-tree MBNs. Chapter 5 deals with fault-tolerant MBNs. In Chapter 6 we describe the basis of our conjecture on the lower bound of the area required for a "perimeter layout" of optimal-time binary-tree MBNs. Finally in Chapter 7 we summarize this work, and identify areas for future research.

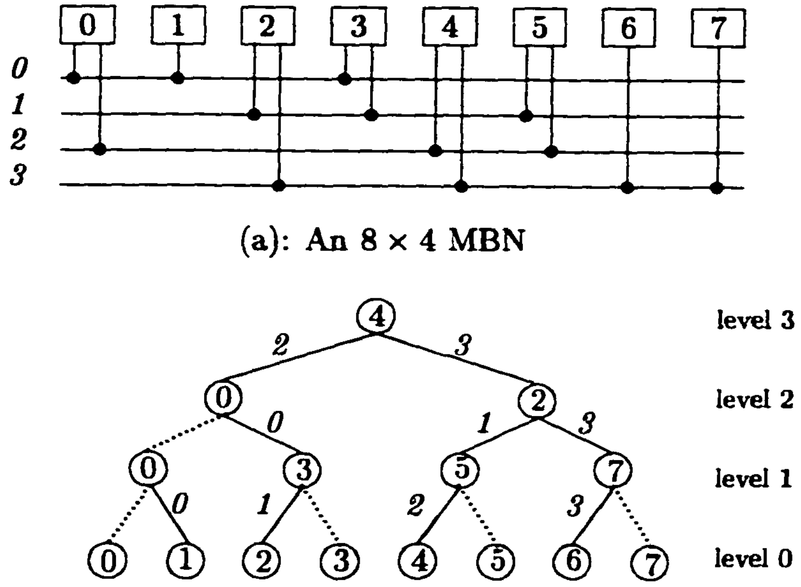
Chapter 2

Preliminaries

In this chapter we discuss some basic ideas used in the rest of the dissertation. We define binary-tree algorithms in Section 2.1 and multiple bus networks (MBNs) in Section 2.2. In Section 2.3 we discuss issues related to running binary-tree algorithms on MBNs. In particular, Section 2.3.1 identifies two types of mappings of MBN processors to "nodes" of binary-tree algorithms. These mappings are important factors, determining the loading of MBNs that run binary-tree algorithms. Finally, in Section 2.4, we prove that an MBN running a binary-tree algorithm can also perform prefix computations in the same order of time.

2.1 Binary Tree Algorithms

A *binary-tree algorithm*, $Bin(n)$, reduces 2^n inputs to a single result. The computation performed by a binary-tree algorithm can be represented as a complete binary tree. For integer $n \geq 1$, and any associative binary operation $\circ$, a binary-tree algorithm, $Bin(n)$ accepts $N = 2^n$ inputs $a_0, a_1, \cdots, a_{N-1}$ at the leaves of a complete binary tree (denoted by $\mathcal{F}(n)$) and produces one output, $a_0 \circ a_1 \circ \cdots \circ a_{N-1}$, at the root of $\mathcal{F}(n)$. The algorithm proceeds level by level from the leaves to the root, applying the



(a): An 8 × 4 MBN

(b): $\mathcal{F}(3)$

Figure 2.1: Running $Bin(3)$ on an 8 × 4 MBN

operation $\circ$ at each internal node to the ***partial results*** at its children. Figure 2.1(b) shows $\mathcal{F}(3)$; the numbers associated with nodes and edges are explained later.

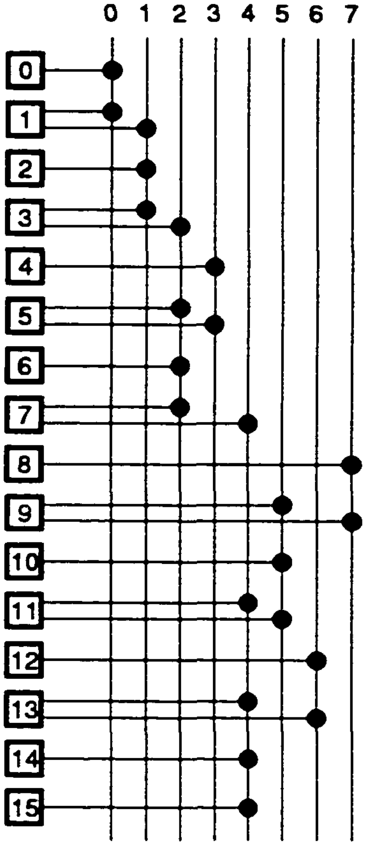
The tree $\mathcal{F}(n)$ has n levels, and at level ℓ (where $0 \leq \ell \leq n$), there are $2^{n-\ell}$ nodes. Clearly, $Bin(n)$ can be used to apply a semigroup operation on a set of 2^n inputs. Any network that runs $Bin(n)$ in $T(n)$ steps can also be used to perform a prefix computation on 2^n inputs in $O(T(n))$ steps (see Section 2.4). It must be noted that $Bin(n)$ is a description of a class of algorithms, rather than the solution to a particular problem (such as a reduction operation) that can be implemented as a binary-tree algorithm. Thus $Bin(n)$ requires at least n steps as the height of $\mathcal{F}(n)$ is n; on the other hand, particular reduction problems such as finding the OR of N bits can be solved on some models in $O(1)$ time [51, 77]. To run $Bin(n)$ on a network with 2^n processors, each of the $2^{n+1}-1$ nodes of $\mathcal{F}(n)$ is mapped to one of the 2^n processors of the network (Figure 2.1(b)). We elaborate further on running binary-tree algorithms on MBNs in Section 2.3.

2.2 Multiple Bus Networks

An $N \times M$ *Multiple Bus Network* (*MBN*) has N processors and M buses. Each processor is connected to a subset of the set of buses. Figure 2.2(a) shows a 16×8 MBN. Two processors connected to the same bus can communicate with each other in one unit of time. A bus can carry only one piece of information at any given point in time.

The number of buses to which a processor is connected is called the *degree of the processor*. The largest of the degrees of all processors is called the *degree of the MBN*. The number of processors connected to a bus is called the *loading of the bus*. The largest of the loadings of all the buses is called the *loading of the MBN*. The MBN of Figure 2.1(a) has a degree of 2 and a loading of 4, while that of Figure 2.2(a) has a degree of 2 and a loading of 5. The degree and loading are important parameters that determine the cost, speed of operation and implementability of an MBN. The degree of an MBN is analogous to the degree of a graph representing a point-to-point network and is indicative of the number of input/output ports needed per processor. A large loading can introduce a significant delay or attenuation of the signal. High loading in electrical buses introduces capacitive coupling that limits the rate at which data can be transmitted. In an optical bus with high loading, the signal is excessively attenuated by power drawn by photodetectors connected to the bus [29]. Therefore, an MBN implementation should attempt to minimize both degree and loading.

An $N \times M$ MBN can be represented as an $N \times M$ Boolean matrix that has a 1 in row p and column b iff processor p is connected to bus b. Figure 2.2(b) shows the matrix representation of the 16 × 8 MBN of Figure 2.2(a). Observe that the rows and columns of the matrix can be permuted without affecting the connectivity properties of the MBN. This is because permuting amounts to just renumbering processors and buses. We use this fact later, when doing so is advantageous.



(a)

	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2		1						
3		1	1					
4				1				
5			1	1				
6			1					
7			1		1			
8								1
9						1		1
10						1		
11					1	1		
12							1	
13					1		1	
14					1			
15					1			

(b)

Figure 2.2: A 16 × 8 MBN and its matrix; blank entries in the matrix represent 0's.

2.3 Running Binary Tree Algorithms on MBNs

We assume that $Bin(n)$ is run on a $2^n \times M$ MBN. Using more than 2^n processors has no advantage. If the number of processors is $2^{n'} < 2^n$, then the 2^n inputs can be divided among the available $2^{n'}$ processors, so that there are $2^{n-n'}$ inputs per processor. Each processor then sequentially reduces the $2^{n-n'}$ inputs to one result in $2^{n-n'}-1$ steps. The reminder of the algorithm is run as a $Bin(n')$ on a $2^{n'} \times M$ MBN.

To run a binary-tree algorithm, $Bin(n)$, on a $2^n \times M$ MBN, each node of $\mathcal{F}(n)$ is mapped to a processor. Each edge of $\mathcal{F}(n)$ that connects nodes mapped to distinct processors represents a communication; such edges are called ***non-trivial edges*** [27]. Consider the example in Figure 2.1(b). Here the nodes of $\mathcal{F}(3)$ are labeled with (mapped to) processor indices $0, 1, \cdots, 7$. This indicates the processor responsible for the action (if any) at a node. Consider the node labeled 0 at level 1 (call it node v for this discussion). Its two children are labeled 0 and 1. The edge from node v to its left child has end vertices, both of which are labeled by the same processor index (0 in this case). Therefore, this edge does not represent a communication and is called a ***trivial edge*** (shown dotted in the figure). On the other hand, the edge from node v to its right child is ***non-trivial*** as its end points have different labels (0 and 1 in this case); hence, the edge represents a communication between processors 0 and 1. Figure 2.1(b) shows non-trivial edges as solid lines; the remaining trivial edges are shown dotted. Each non-trivial edge of $\mathcal{F}(n)$ is mapped to a bus of the MBN.

Conversely, an MBN to run $Bin(n)$ can be specified by mapping nodes and non-trivial edges of $\mathcal{F}(n)$ to processors and buses, respectively, of the MBN. Thus $\mathcal{F}(n)$ (with nodes and non-trivial edges appropriately labeled) completely specifies a $2^n \times M$ MBN and the method used to run $Bin(n)$ on it. Figure 2.1(b) shows $\mathcal{F}(3)$ corresponding to the MBN in Figure 2.1(a). We will loosely use the term "binary-tree MBNs" to refer to MBNs suitable for running binary-tree algorithms.

In running a binary-tree algorithm on an MBN, we assume that in one "step" a processor can read from or write on each bus it is connected to and perform an internal operation using operands from its local memory or input ports. This assumption is reasonable when the number of ports in a processor is small—all of the MBNs considered in this work have a (small) constant degree. Since the focus of this work is on the network connecting processors, there is no advantage in separately considering the time required for internal operations. The following restrictions apply, however: (*i*) Each value sent or received by a processor during a step uses a different bus, and (*ii*) the pair of processors sending and receiving a value must be connected to a common bus. Under these assumptions, a processor is permitted to (a) send a partial result of the binary-tree algorithm, (b) receive two partial results, and (c) perform the operation $\circ$ (associated with the binary-tree algorithm) on the partial results received, all in one step. This is not very different from the usual assumption that a processor can access operands from its local memory and perform an operation on them, all in one step.

2.3.1 Direct and Indirect Mapping

As noted earlier, running $Bin(n)$ on an MBN requires mapping nodes to processors. In this section we identify two types of mappings, direct and indirect, that greatly impact the degree and loading of binary-tree MBNs.

For any node u of $\mathcal{F}(n)$, let $\mu(u)$ denotes the processor to which u is mapped. Let u be an internal node of $\mathcal{F}(n)$ with children v and w. Node u is said to be a *direct* node iff $\mu(u) = \mu(v)$ or $\mu(u) = \mu(w)$; otherwise, node u is said to be *indirect*. A direct node is mapped to the same processor as one of its children while an indirect node is mapped to a processor which is different from both of its children. This implies that an indirect node is connected to each of its children by non-trivial edges, whereas a direct node is connected to one of its children by a trivial edge. A processor mapped to a direct node is called a *direct processor* of the step in question; otherwise, it is called an *indirect processor* of the step. Since a processor may be mapped to more than one node of $\mathcal{F}(n)$, it is possible for the same processor to be a direct processor at one step and an indirect processor at another. Any mapping that has an indirect node is called an *indirect mapping*; otherwise it is called a *direct mapping*.

As an example, in Figure 2.1(b) all nodes except the root and its right child are direct. Therefore the entire $Bin(3)$ or $\mathcal{F}(3)$ uses an indirect mapping (as there is an indirect node). On the other hand, the $\mathcal{F}(2)$ consisting of the left subtree of the root represents a direct mapping.

Observe that an indirect node involves two communications (one from each child), whereas a direct node requires only one. Thus a direct mapping minimizes the number of communications. Notwithstanding the fact that an indirect mapping entails more communications, we show in Section 3.2 that this mapping is necessary for constant loading.

2.4 Prefix Computations on Binary-Tree MBNs

Given N inputs, $a_1, a_2, \cdots, a_N$, and an associative operation $\circ$, the i^{th} prefix (where $1 \leq i \leq N$) is the quantity $a_1 \circ a_2 \circ \cdots \circ a_i$. A prefix computation for the above inputs and operation computes the i^{th} prefix for each $1 \leq i \leq N$. The relationship between reduction algorithms and prefix computations is well known in the context of a PRAM [38, pp. 44–49] and a fixed-degree topology [49, pp. 37–43]. This relationship has not been studied for binary-tree MBNs, however. We prove here that a binary-tree MBN is suitable for prefix computations as well.

Theorem 2.1 *If $X(n)$ is an MBN that runs $Bin(n)$ in $T(n)$ steps, then $X(n)$ can run a prefix algorithm in $2T(n)$ steps.*

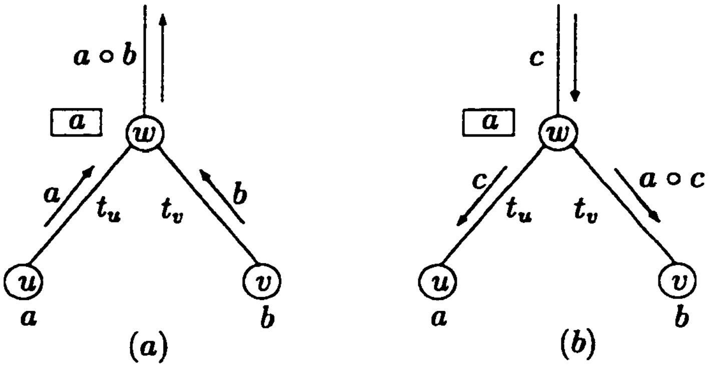


Figure 2.3: Steps of running prefix computation on binary-tree MBNs

Proof: When an MBN runs a binary-tree algorithm, the nodes of $Bin(n)$ are executed in a manner that respects the precedence relationship described by the tree $\mathcal{F}(n)$. Figure 2.3(a) shows three nodes of $\mathcal{F}(n)$ (corresponding to two levels) where nodes u and v are children of node w. Let the two partial results (or inputs) held by nodes u and v be a and b respectively, and let the associative operation performed by the binary-tree algorithm be $\circ$.

The prefix computation runs on the binary-tree MBN in two phases. The first phase runs the binary-tree algorithm from the leaves to the root. The only difference here (from running a regular binary-tree algorithm) is that a node saves the value it receives from the left child, unlike the usual form of the algorithm that simply computes the partial result. For example, node w receives a and b from nodes u and v and computes $a \circ b$. In addition to computing this quantity, node w also saves the value "a" (shown in a box in Figure 2.3). Node w sends partial result $a \circ b$ to the next higher level in the next step (Figure 2.3(a)). The time to run this phase is clearly the same as that of the binary-tree algorithm, namely $T(n)$.

The second phase of the prefix computation also proceeds level by level, starting from the top most level (level n) down to the leaves. This can be viewed as reversing the binary-tree algorithm where value(s) from top of the tree propagates to the leaves. Figure 2.3(b) describes the action at each node during this phase. The root of $\mathcal{F}(n)$ sends the value it stored (one received from its left child in the first phase) to the right child. It sends the identity[1] of operation "$\circ$" to the left child. A processor with stored value a and that receive value c from its parent (*i*) sends c unaltered to its left child and (*ii*) sends $a \circ c$ to the right child. This phase mimics the binary-tree algorithm (phase 1) in reverse, so its time is $T(n)$ as well. Therefore, the time required to run a prefix computation on a binary-tree MBN is twice as much as the time required for running a binary-tree algorithm. (The correctness of this method follows from the results in [38, 49].) ■

[1] The identity ι has the property that for any value x from the domain of $\circ$, $\iota \circ x = x \circ \iota = x$. If $\circ$ does not have an identity, then the root could simply send a special signal indicating to its left child that it need not apply $\circ$ to the value received.

Chapter 3

Degree, Loading, Time Trade-Offs

This chapter establishes non-trivial relationships between the degree, loading and running time of binary-tree MBNs. We first show by a trivial connectivity argument that any binary-tree MBN has a degree of at least 2 and a loading of at least 2; the loading is at least 3 if no more than 2^{n-1} buses are used for 2^n processors. Next we show that for a direct mapping, constant degree can never yield constant loading, and vice-versa. We then establish a series of results that successively bound the loading of degree-2, optimal-time binary-tree MBNs for $Bin(n)$ to first $\Omega(\sqrt{n})$, then to $\Omega(n^{\frac{2}{3}})$ and finally to $\Omega(\frac{n}{\log n})$, where 2^n is the size of the problem. These results make no assumptions about the type of mapping (direct or indirect) and the number of buses, although the optimal-time restriction indirectly requires the MBN to have at least 2^{n-1} buses. Considering that increasing the degree by just 1 can yield a constant loading MBN (see Section 3.6), these lower bound results are quite surprising.

Further, if we relax the optimal-time requirement, then we show the existence of a degree-2, loading-4 MBN that runs $Bin(n)$ in $2n-3$ steps. However, the extra time needed (beyond n) still bounds the loading. We show that if a degree-2 MBN runs $Bin(n)$ in $n+t$ steps, for some $0 < t < n$, then the loading is $\Omega\left(\frac{n}{t\log(\frac{n}{t})}\right)$.

In Section 3.1 we introduce some preliminary ideas used in this chapter and Section 3.2 bounds the loading for binary-tree MBNs with a direct mapping. In Section 3.3 we derive the first of the general lower bounds and lay most of the ground work necessary for the tighter lower bounds of Sections 3.4 and 3.5. We explore loading-time tradeoffs in Section 3.7. We extend the lower bound results of Sections 3.3, 3.4 and 3.5 to k-ary tree algorithms in Section 3.8.

3.1 Preliminaries

As mentioned in Section 2.3, we will consider a 2^n-processor MBN to run $Bin(n)$. An optimal-time MBN requires at least 2^{n-1} buses. If the number of buses is less than 2^{n-1}, at least the first level of $\mathcal{F}(n)$ requires more than one step to schedule, so optimal time is not possible. Therefore, we consider optimal-time $2^n \times M$ binary-tree MBNs for $Bin(n)$ with $M \geq 2^{n-1}$ buses. If such an MBN has degree 2, then it has at most 2^{n+1} connections (at most 2 per processor). If these connections are evenly distributed among the buses, then the loading would be $\left\lceil \frac{2^{n+1}}{M} \right\rceil \leq 4$. In this chapter we show that such a uniform distribution of connections is not possible and that a large number of connections is concentrated on a small number of buses resulting in a large non-constant loading.

An MBN is said to be *connected* iff there is a path (possibly via several buses and processors) between any pair of processors. We now derive trivial lower bounds on the degree and loading of a connected binary-tree MBN.

Lemma 3.1 *For $n \geq 1$ and $M \geq 2$, any connected $2^n \times M$ binary-tree MBN has a degree of at least 2 and a loading of at least* $\max\left(2, \left\lceil \frac{2^n+1}{M} \right\rceil\right)$.

Proof: If the degree is 1 and if the MBN has connections to each of its $M \geq 2$ buses, then the MBN cannot be connected; each processor connected to a bus b

can only communicate with other processors connected to bus b. Thus at least one processor must be connected to 2 or more buses. This implies that the total number of connections in the MBN is at least $2^n + 1$. These connections are distributed over M buses, so the loading is at least $\left\lceil \frac{2^n+1}{M} \right\rceil$. Since each bus must have at least 2 connections (otherwise it cannot be used for a communication), the loading is $\max\left(2, \left\lceil \frac{2^n+1}{M} \right\rceil\right)$.

■

Remark: If $M = 2^m$ for some $1 \leq m < n$, then the minimum loading is $2^{n-m} + 1$. The $2^n \times 2^m$ Tree MBN of Section 4.2 has an optimal loading of 3.

3.2 Lower Bound for Direct Mapping

To run a binary-tree algorithm on an MBN, the nodes of the tree $\mathcal{F}(n)$ are mapped to processors, and non-trivial edges are mapped to buses. Recall the definitions of direct and indirect mapping (see Section 2.3.1, page 16). In a direct mapping, each internal node of $\mathcal{F}(n)$ is mapped to the same processor as one of its children; that is, a processor applying the operation $\circ$ (associated with a binary-tree algorithm) holds one of the operands as a partial result from the previous step. On the other hand, in an indirect mapping, two processors with partial results may send them to a third processor that applies $\circ$ on these. The direct mapping may appear to be a better choice as it reduces communication requirements by maximizing the number of trivial edges. This is not true for the loading of the MBN, as we show below. Indeed, the MBN proposed by Vaidyanathan and Padmanabhan [85] uses a direct mapping and has a non-constant loading.

Lemma 3.2 *For any $n \geq 1$, an MBN with degree d that runs $Bin(n)$ optimally in n steps using a direct mapping has a loading of at least $\left\lceil \frac{n}{d} \right\rceil + 1$.*

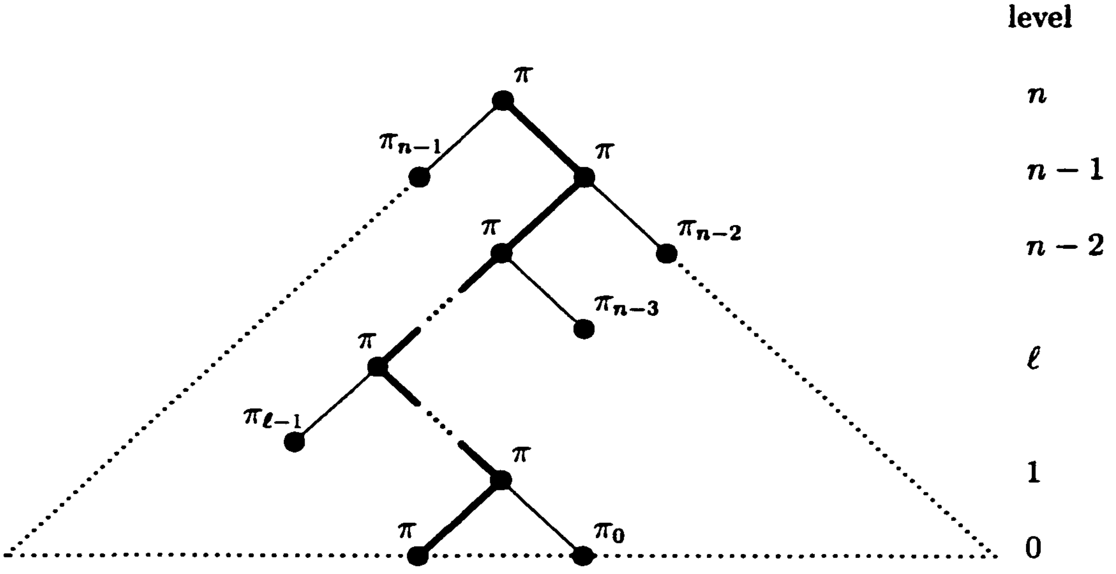


Figure 3.1: $\mathcal{F}(n)$ with a direct mapping

Proof: Consider MBN $\mathcal{M}$ with nodes labeled by a direct mapping. Observe first that for any node u that is mapped to some processor $\mu(u)$, there exists a path from u to a leaf, such that all nodes on the path are mapped to $\mu(u)$. (This follows from the definition of direct mapping.) Let the root of $\mathcal{M}$ be mapped to processor π. From the above observation, there is a path from the root to a leaf such that all nodes on this path are mapped to π (see Figure 3.1). Clearly, there are n internal nodes on this path, each of which has one of its two children also on the path. Let the children not included in the above path be mapped to processors π_ℓ (where $0 \leq \ell < n$) as shown in Figure 3.1. Each leaf of $\mathcal{M}$ is mapped to a different processor (as each input of $Bin(n)$ is in a different processor). This coupled with the observation at the beginning of this proof, establishes that all of $\pi_0, \pi_1, \cdots, \pi_{n-1}$ are distinct. Thus processor π is required to communicate with n different processors.

Let processor π be connected to buses b_i (where $0 \leq i < d' \leq d$). Since the MBN runs $Bin(n)$ in n steps, each of the processors $\pi_0, \pi_1, \cdots, \pi_{n-1}$ must also be connected to at least one bus b_i (where $0 \leq i < d'$). Thus the total number of connections to all buses b_i is at least $n + d'$. This implies that the loading of the MBN is at least $\left\lceil \frac{n+d'}{d'} \right\rceil = \left\lceil \frac{n}{d'} \right\rceil + 1 \geq \left\lceil \frac{n}{d} \right\rceil + 1$. ■

Remark: Lemma 3.2 implies that an indirect mapping is essential for any optimal-time binary-tree MBN with constant degree and loading.

3.3 An $\Omega(\sqrt{n})$ Lower Bound

In this section, we develop the first of a series of non-trivial lower bounds on the loading of degree-2, optimal-time, binary-tree MBNs. Here we will prove that if an MBN runs $Bin(n)$ in n steps and if its degree is 2, then its loading is $\Omega(\sqrt{n})$.

3.3.1 Strategy and Definitions

We prove this lower bound result by showing that the connections in any degree-2, optimal-time binary-tree MBN are distributed unevenly over the buses. Our strategy here (and to a large extent in Sections 3.4 and 3.5 as well) is to identify (or prove the existence of) a small number, β, of buses that collectively have a large number, γ, of connections. This will establish that the loading is at least $\frac{\gamma}{\beta}$.

For $M \geq 2^{n-1}$, consider a $2^n \times M$ MBN, $X(n)$, that runs $Bin(n)$ in n steps (numbered $1, 2, \cdots, n$) and whose degree and loading are 2 and L, respectively. We will use the terms "end of step s" and "beginning of step $s + 1$" synonymously. For any $1 \leq s \leq n$, let $X_s(n)$ denote a $2^n \times M$ MBN that includes only those connections of $X(n)$ that are used in at least one of steps $1, 2, \cdots, s$. Then $X_0(n)$ is a $2^n \times M$ MBN with no connections and $X_n(n) = X(n)$.

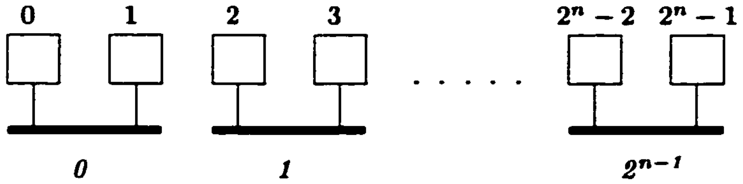


Figure 3.2: Step 1 of $X_1(n)$

At each step s we will consider an $2^n \times 2^{n-1}$ "sub-MBN", $Y_s(n)$, of $X_s(n)$; i.e., connections of $Y_s(n)$ are also connections of $X_s(n)$. Sub-MBN $Y_s(n)$ consists of those connections of $X_s(n)$ whose existence has been established. We say that a connection is *added* to mean that a previously unaccounted for connection has been detected. Therefore, the degree and the loading of the MBN changes from step to step. Running $Bin(n)$ on an MBN can be viewed as a step by step construction of the MBN with the counted connections added at each step.

An intermediate result of $Bin(n)$ (value at any non-root or non-leaf node of tree $\mathcal{F}(n)$) is called a *partial* result. A processor p holding a partial result or an input at the end of step s (where $0 \leq s \leq n$) is called a *result processor* of step s. Otherwise p is a *non-result processor* of step s. If the degree of the processor p at the end of step is 2, then it is called a *full processor* of step s; otherwise, p is a *non-full processor* of step s.

Clearly, all 2^n processors are non-full, result processors of step 0; i.e., at the start of the algorithm. Step 1 (the first step) requires at least 2^{n-1} communications (exactly 2^{n-1}, if all 2^{n-1} partial results generated at the end of step 1 are obtained by a direct mapping). Therefore $X_1(n)$ is isomorphic to the MBN shown in Figure 3.2. Thus for $1 \leq s \leq n$, the terms "full" and "non-full" are synonymous with "degree-2" and "degree-1," respectively.

3.3.2 Basic Results

We now list four simple consequences of $X(n)$ being a degree-2, optimal-time, binary-tree MBN; these facts are used, often without explicit mention, in subsequent discussion.

1. All partial results received in a step are used in the same step, and a partial result generated in a step is used up in the next step. This is because the algorithm runs optimally, so partial results cannot idle.
2. A direct (resp., indirect) processor of a step receives one (resp., two) partial results in that step; this follows from 1 above.
3. A processor receiving two partial results at a step must do so from different processors; otherwise the step will not be executed in unit time.
4. A processor sending a partial result cannot receive one at the same step. This is because it will have to receive two partial results and send one partial result as a processor can hold only one partial result. This is not possible on a degree-2, optimal-time MBN.

Lemma 3.3 *For any* $1 < s \leq n$*, if* p *is a non-full, result processor of step* s*, then* p *is a non-full, result processor of steps* $1, 2, \cdots, s$*.*

Proof: It suffices to prove that if p is a non-full, result processor of step s, then it is a result processor of step $s-1$. If p holds a result at the end of step s, but not at the end of step $s-1$, then it must have obtained two partial results during step s. This requires p to have two connections and be a full processor of step s. ■

Corollary 3.4 *For any* $L \leq s \leq n$*, each result processor of step* s *is a full processor of step* s*.*

Proof: Let p be a non-full, result processor of step s. Then by Lemma 3.3, it is also a result processor of steps $1, 2, \cdots, s$. Therefore to prove the lemma, it is sufficient to prove that $s < L$. Let p receive a partial result for step t (where $1 \leq t \leq s$) from processor p_t, via the only bus b (say) to which p is connected. Therefore, bus b is connected to processors in the set $\{p\} \cup \{p_t : 1 \leq t \leq s\}$.

Consider processor p_t that sends a partial result to p during step t. If p_t is a result processor of step t, then it must receive two partial results from processors different from p (in addition to sending a partial result to p). This is not possible as one of the (at most 2) buses to which p_t is connected is used by p. Since this bus (bus b) is used by p during steps $1, 2, \cdots, s$, processor p_t cannot be a result processor of steps $t, t+1, \cdots, s$. Therefore, $p_t \notin \{p_x : t < x \leq s\}$ and so $\{p\} \cup \{p_t : 1 \leq t \leq s\}$ has $s+1$ processors, all of which are connected to bus b. Since the loading of $X(n)$ is L, we have $s + 1 \leq L$ (or $s < L$). ■

From this point on, we will only consider step $s \geq L$. Since our aim is to prove that $L = \Omega(\sqrt{n})$ ($\Omega(\frac{n}{\log n})$ in Section 3.5), we may assume that $L < n$. By Corollary 3.4, all result processors (of any step) can be assumed to be connected to 2 buses.

3.3.3 The Accounting Scheme

In determining a lower bound on the loading L of $X(n)$, we will count connections between processors and buses of $X(n)$. Let $\langle p, b\rangle$ denote a connection between processor p and bus b. In our analysis, we will consider only those connections $\langle p, b\rangle$ for which p is a full processor, and which participates in some step $s \geq L$. Since a lower bound on the loading is sought, some connections can be ignored.

To account for the connections considered, we now associate each such connection with a processor. For each processor p and step $s \geq L$, define a set, $\Gamma_s(p)$, of connections *owned by processor p in step s*. (We will show later that if $p_1 \neq p_2$, then $\Gamma_s(p_1)$ and $\Gamma_s(p_2)$ are disjoint.) We now define $\Gamma_s(p)$.

1. If p is a result processor of step L, then it is also a full processor of step L (by Corollary 3.4). Let p be connected to buses b_1 and b_2. For each such p, define $\Gamma_L(p) = \{\langle p, b_1\rangle, \langle p, b_2\rangle\}$. If p is not a result processor of step L, then define $\Gamma_L(p)$ to be empty.

2. For $s > L$, let p be a result processor of step s that receives partial result(s) from (not necessarily distinct) processor(s) p' and p'' via bus(es) b' and b'', respectively. Define $\Gamma_s(p)$, $\Gamma_s(p')$ and $\Gamma_s(p'')$ as follows.

$$\begin{aligned}
\Gamma_s(p) &= \Gamma_{s-1}(p) \cup \{\langle p_1, b'\rangle, \langle p_2, b''\rangle\} \\
\Gamma_s(p') &= \Gamma_{s-1}(p') - \{\langle p_1, b'\rangle\} \\
\Gamma_s(p'') &= \Gamma_{s-1}(p'') - \{\langle p_2, b''\rangle\}
\end{aligned}$$

   where $\langle p_1, b'\rangle \in \Gamma_{s-1}(p')$ and $\langle p_2, b''\rangle \in \Gamma_{s-1}(p'')$; since we are interested primarily in the cardinality, $|\Gamma_s(p)|$, of $\Gamma_s(p)$, $\langle p_1, b'\rangle$ (resp., $\langle p_2, b''\rangle$) can be any element of $\Gamma_{s-1}(p')$ (resp., $\Gamma_{s-1}(p'')$). Note that if p receives only one value in step s, then $p' = p''$, $b' = b''$ and $p_1 = p_2$.

   In summary, for each partial result received by processor p from processor p' via bus b, processor p' transfers ownership of a connection on bus b to processor p. If processor p does not send or receive any partial result in step s, then $\Gamma_s(p) = \Gamma_{s-1}(p)$.

Lemma 3.5 *For any $s \geq L$,*

(*i*) *For distinct processors p_1, p_2, $\Gamma_s(p_1)$ and $\Gamma_s(p_2)$ are disjoint.*

(*ii*) *For any processor p, if $\langle p', b\rangle \in \Gamma_s(p)$, then processor p is connected to bus b.*

(*iii*) *If p is a result processor of step s, then $\Gamma_s(p)$ has a connection of the form $\langle p', b\rangle$, for each bus b to which p is connected.*

Proof: At step L, by definition of Γ_L, all result processors own their connections to the two buses to which they are connected. Therefore, Lemma 3.5 holds at step L. Observe that in part 2 of the definition of $\Gamma_s(p)$, the sets $\Gamma_s(p)$ and $(\Gamma_s(p'), \Gamma_s(p''))$ are disjoint, and the connections added to $\Gamma_s(p)$ are $\langle p_1, b'\rangle$ and $\langle p_2, b''\rangle$, where b' and b'' are buses to which p is connected. These observations, coupled with the fact that Lemma 3.5 holds for step L, completes the proof. ■

Remarks: If the sets $\Gamma_s(p)$ are used to count the number of connections in $X_s(n)$, then part (*i*) of Lemma 3.5 ensures that no connection is counted more than once. However, some connections may not be counted at all. Part (*ii*) is used later in Theorem 3.9. Part (*iii*) ensures that the transfer of ownership in part 2 of the definition of $\Gamma_s(p)$ is always possible.

Lemma 3.6 *For any step $s > L$ and any processor p, $|\Gamma_s(p)| = |\Gamma_{s-1}(p)| + \delta$, where*

$$\delta = \begin{cases} 1, & \textit{if } p \textit{ is a result processor of steps } s \textit{ and } s-1. \\ 2, & \textit{if } p \textit{ is a result processor of step } s \textit{ and a non-result processor} \\ & \textit{of step } s-1. \\ -1, & \textit{if } p \textit{ is a non-result processor of step } s \textit{ and a result processor} \\ & \textit{of step } s-1. \\ 0, & \textit{if } p \textit{ is a non-result processor of steps } s \textit{ and } s-1. \end{cases}$$

Proof: Observe that δ is the number of partial results received by processor p in step s; $\delta = -1$ indicates that p sends a partial result. The lemma now follows from this observation and part 2 of the definition of $\Gamma_s(p)$. ■

Corollary 3.7 *For any step $s \geq L$, if p is a result processor of α of the steps $L, L+1, \cdots, s$, then $|\Gamma_s(p)| \geq \alpha$.*

Proof: The first time p becomes a result processor at step s_0, say, (even if $s_0 = L$), $|\Gamma_{s_0}(p)| = 2$. For each of the remaining $\alpha - 1$ times it is a result processor in some step $s' > s_0$, we consider two cases:

Case 1: Suppose p is a result processor of steps $s' - 1$ and s'. Here $|\Gamma_{s'}(p)| = |\Gamma_{s'-1}(p)| + 1$ (Lemma 3.6).

Case 2: Suppose p is a result processor of step s' and a non-result processor of step $s' - 1$. Since $s' > s_0$, there is a step s'' ($s_0 < s'' < s'$) such that p is a result processor of step $s'' - 1$ and a non-result processor of steps $s'', s'' + 1, \cdots, s' - 1$. Here $|\Gamma_{s'}(p)| = |\Gamma_{s'-1}(p)| + 2$, $|\Gamma_{s'-1}(p)| = |\Gamma_{s''}(p)|$, and $|\Gamma_{s''}(p)| = |\Gamma_{s''-1}(p)| - 1$ (again by Lemma 3.6). Therefore, $|\Gamma_{s'}(p)| = |\Gamma_{s''-1}(p)| + 1$.

In any case, for each step $s' > s_0$ of which p is a result processor, $\Gamma_{s'}(p)$ increases by one. Thus at the last step $t \leq s$ of which p is a result processor, $|\Gamma_t(p)| = 2 + (\alpha - 1) = \alpha + 1$. If $t < s$, then $|\Gamma_{t+1}(p)| = |\Gamma_s(p)| = \alpha$. ■

3.3.4 Non-Uniform Bus Usage

In this section we show that as the binary-tree algorithm proceeds towards the root of $\mathcal{F}(n)$, most of the activity in the MBN centers around few buses that ultimately incur a high loading.

For any step $s \geq L$, a bus b of $X(n)$ is said to be *active in step s* iff it is connected to at least one result processor of step s. If a bus is used to carry a partial result in step s, then it must be active in step s. However, a bus that is active in step s need not be used in step s. In the following lemma, we prove that the pool of buses that could be active at a step shrinks with each step, thereby forcing a few buses to have a large number of connections.

Lemma 3.8 *For any step $s > L$, bus b is active in step s, then it is also active in step $s-1$.*

Proof: Let bus b not be active in step $s-1$. Then by definition of a non-active bus, all the processors connected to b are non-result processors of step $s-1$. Suppose at step s, processor p connected to b becomes a result processor of step s, thereby making b active in step s. Since p cannot be a result processor of step $s-1$ (otherwise b would be an active bus in step $s-1$), p must receives two results in step s from distinct processors p' and p''. One of these partial results must be via bus b. Clearly the two sending processors p' and p'' are full (Corollary 3.4), result processors of step $s-1$. Thus, one of them must be connected to b in step $s-1$, which contradicts the assumption that b is not active in step $s-1$. ■

3.3.5 The Lower Bound

We are now in a position to prove the main result of this section.

Theorem 3.9 *For any $n \geq 2$, if a 2^n-processor MBN with degree 2 and loading L runs $Bin(n)$ optimally in n steps, then $L = \Omega(\sqrt{n})$*

Proof: From Lemma 3.8, there exists a bus, b_0, that is active in steps $L, L+1, \cdots, n$. Let b_0 be connected to $\ell \leq L$ full processors, $p_1, p_2, \cdots, p_\ell$. For $1 \leq i \leq \ell$, let the two buses to which processor p_i is connected be b_0 and b_i. Also let processor p_i be a result processor α_i times from step L to step n.

From Lemma 3.5(ii), each element of $\Gamma_n(p_i)$ is a connection to either b_0 or b_i. Since the loading of the MBN is L, $\sum_{i=1}^{\ell} |\Gamma_n(p_i)| \leq \ell + \ell L \leq L^2 + L$. From Corollary 3.7 we also have $\sum_{i=1}^{\ell} |\Gamma_n(p_i)| \geq \sum_{i=1}^{\ell} \alpha_i$. Since b_0 is an active bus of steps $L, L+1, \cdots, n$, $\sum_{i=1}^{\ell} \alpha_i \geq n - L + 1$. Thus, $n - L + 1 \leq \sum_{i=1}^{\ell} \alpha_i \leq \sum_{i=1}^{\ell} |\Gamma_n(p_i)| \leq L^2 + L$, which implies that $n \leq L^2 + 2L - 1$ or $L = \Omega(\sqrt{n})$. ■

Remark: Theorem 3.9 proves that for large problem sizes, the product of the degree and loading of any MBN that runs a binary-tree algorithm in optimal time is at least 9, thereby establishing that the MBN, $\mathcal{T}(n)$, proposed in Section 3.6 has the best possible "degree-loading" product.

3.4 An $\Omega\left(n^{\frac{2}{3}}\right)$ Lower Bound

In the lower bound of Section 3.3, we selected a bus b_0 and proved that its neighborhood (consisting of processors on b_0 and buses connected to these processors) had a large number of connections. In restricting our consideration to the neighborhood of bus b_0, the technique used undercounted the number of connections in the neighborhood. Here we develop additional results that provide a more accurate count of connections, even though the consideration is expanded to a larger neighborhood.

3.4.1 Additional Results

Recall the definitions of direct and indirect nodes and processors (Section 2.3.1, page 16).

Lemma 3.10 *For any $s \geq L$, let p be a result processor of step $s+1$.*

(*i*) *If p is a result processor of step s, then it is a direct processor of step $s+1$.*

(*ii*) *If p is a non-result processor of step s, then the following assertions hold:*

(*a*) *Processor p is an indirect processor of step $s+1$.*

(*b*) *The two buses to which processor p is connected are active in step s.*

(*c*) *For each bus b to which processor p is connected, a result processor of step s (that is also connected to b) becomes a non-result processor of step $s+1$.*

Proof: If p is a result processor of both steps s and $s+1$, then the result it holds from step s must be used to obtain the result of step $s+1$. (Otherwise, the processor will have to receive two new values, while sending the result of step s to another processor; this is not possible on a degree-2 MBN.) Also, since $X(n)$ runs $Bin(n)$ in n steps, partial results cannot be saved to be used at a later step.

If, on the other hand, p does not hold a result at step s, it must receive two partial results during step $s+1$, and is, therefore, an indirect processor of step $s+1$. Since these results (of step s) arrive through the two buses b_1 and b_2 (say) to which p is connected, there must be result processors p_1 and p_2 of step s that are connected to buses b_1 and b_2, respectively; that is, the buses b_1 and b_2 are active in step s. The result processors p_1 and p_2 of step s cannot be result processors of step $s+1$ as they send their results to processor p. (A result processor sending its value to another processor must receive two values to remain a result processor of the next step; this is not possible on a degree-2 MBN.) ■

The following corollary is a generalization of Lemma 3.8.

Corollary 3.11 *For any $s \geq L$, if ρ result processors of step s are connected to bus b, then for any $s \geq s' \geq L$, at least ρ result processors of step s' must be connected to bus b.*

Proof: It is sufficient to prove that the number of result processors connected to bus b cannot increase after step L. Let processors $p_1, p_2, \cdots, p_x$ be result processors of step s' that are connected to bus b. Let processor q connected to bus b not be a result processor of step s', and let it become a result processor of step $s' + 1$. Since processor q is connected to bus b, one of the partial results must come from one of the processors $p_1, p_2, \cdots, p_x$ via bus b. The processor that is sending the partial result becomes a non-result processor of step $s'+1$, and the total number of result processors connected to bus b does not increase. Therefore, at least a total of ρ result processors must be connected to bus b in all the steps $s \geq s' \geq L$. ■

Remark: It is important to note that the processors holding the results may change from one step to another, while the number of result processors is non-increasing.

Two processors are said to be *neighbors* iff they are connected to a common bus. For integers a, b with $a \leq b$, let *interval* $[a, b]$ denote the set $\{a, a+1, \cdots, b-1, b\}$.

Lemma 3.12 *For $s > L$, if p is not a result processor of step $s - 1$, and is a result processor of steps $s, s+1, \cdots, s+x-1$ (for some $x > 0$), then the following assertions hold:*

(i) For each step s' in the interval $[L, s-1]$, at least $x + 1$ neighbors of p are result processors of step s'.

(ii) At the end of step $s + x - 1$, the neighbors of p collectively own at least $\left((x+1)(s-L) + \frac{x(x-1)}{2}\right)$ connections.

Proof: When processor p becomes a result processor of step s, it must be an indirect processor of step s (Lemma 3.10), and it consumes two results from its neighbors. Processor p also consumes a result from one of its neighbors in each of the steps $s+1$, $s+2, \cdots, s+x-1$, (during which it is a direct, result processor). By Corollary 3.11, for the $2 + (x - 1) = x + 1$ results consumed by p in steps $s, s + 1, \cdots, s + x - 1$, there must be $x + 1$ results among the neighbors of p in each of steps $L, L + 1, \cdots, s - 1$.

Of the $x + 1$ results consumed by processor p, two are consumed in step s. Therefore for the remaining $x - 1$, there are $x - 1$ result processors among the neighbors of p in each of steps $L, L + 1, \cdots, s - 1, s$. In general, for any $1 \leq i < x$, there are $x - i$ result processors among the neighbors of p in each of the steps $L, L + 1, \cdots, s + i - 1$. If N is the set of neighbors of p, then by Corollary 3.7 we have $\sum_{p' \in N} |\Gamma_{s+x-1}(p')| = (s - L)(x + 1) + \sum_{i=1}^{x-1} (x - i) = (s - L)(x + 1) + \frac{x(x-1)}{2}$. ■

3.4.2 Tighter Lower Bound

Now we are in a position to prove the main result of this section.

Theorem 3.13 *For any $n \geq 2$, if a 2^n-processor MBN with degree of 2 and loading of L runs a Bin(n) optimally in n steps, then $L = \Omega(n^{\frac{2}{3}})$.*

Proof: Let the loading of the MBN be $L < n$. By Lemma 3.8, there is a bus that is active during each step in the interval $[L, n]$; let this bus be b_0. Let full processors $p_1, p_2, \cdots, p_\ell$ (for some $\ell \leq L$) be connected to bus b_0. For each i (where $1 \leq i \leq \ell$), let processor p_i be connected to bus b_i (in addition to bus b_0). Besides processor p_i, let bus b_i be connected to $m_i < L$ processors (see Figure 3.3).

For any given step $s \in [L, n]$, at least one of $p_1, p_2, \cdots, p_\ell$ is a result processor (as b_0 is active during each step of $[L, n]$). Therefore, the interval $[L, n]$ can be partitioned into k subintervals, $I_1, I_2, \cdots, I_k$, as follows (see Figure 3.4):

(i) In each step of subinterval I_j, processor $\pi_j \in \{p_i \; : \; 1 \leq i \leq \ell\}$ is a result processor.

(ii) For $j > 1$, $\pi_j \neq \pi_{j-1}$.

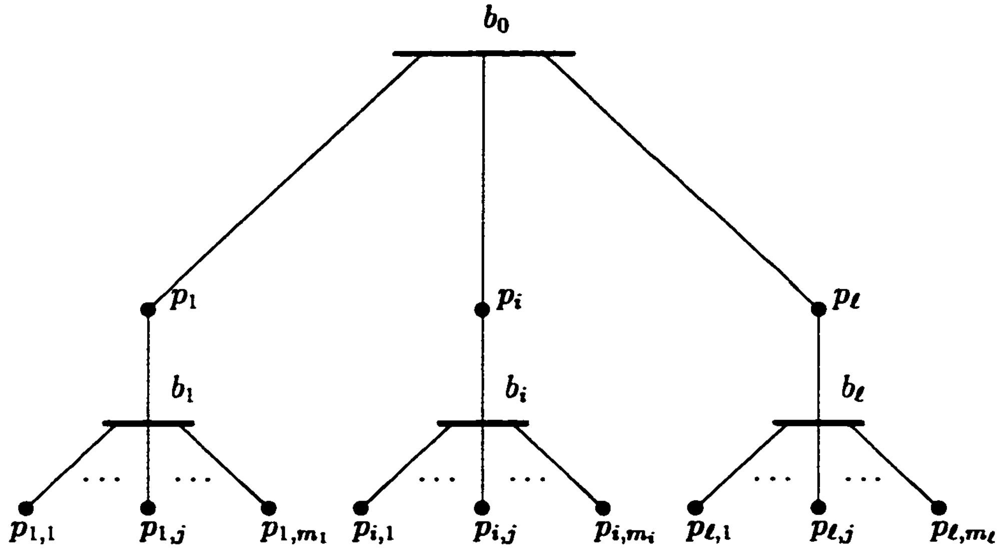


Figure 3.3: Processors and buses in the neighborhood of bus b_0

Let interval I_j be of length x_j (where $1 \leq x_j \leq n-L+1$). Clearly, $\sum_{j=1}^{k} x_j = n - L + 1$. Also $I_1 = [L, L+x_1-1]$ and for $j > 1$, $I_j = \left[L + \sum_{r=1}^{j-1} x_r\ ,\ L - 1 + \sum_{r=1}^{j} x_r\right] = [s_j, s_j + x_j - 1]$ (say).

Applying Lemma 3.12 to I_j, the number of connections owned by neighbors of π_j (at the end of step $s_j + x_j - 1$) is $(s_j - L)(x_j + 1) + \frac{x_j(x_j-1)}{2}$. Summing this for all k intervals, we can assert that the number of connections collectively owned by processors $p_1, p_2, \cdots, p_\ell$ and their neighbors is at least $\sum_{j=1}^{k}\left((s_j - L)(x_j+1) + \frac{x_j(x_j-1)}{2}\right) = \frac{x_1(x_1-1)}{2} + x_1(x_2+1) + \frac{x_2(x_2+1)}{2} + \cdots + (x_1 + x_2 + \cdots + x_{k-1})(x_k+1) + \frac{x_k(x_k-1)}{2} > \frac{(x_1^2+x_2^2+\cdots+x_k^2)}{2} + (x_1x_2 + x_1x_3 + \cdots + x_1x_k + x_2x_3 + x_2x_4 + \cdots + x_2x_k + \cdots + x_{k-1}x_k) = \frac{(n-L)^2}{2} = \Theta(n^2)$. These connections are distributed among the $1 + \ell + \sum_{i=1}^{\ell} m_i \leq 1 + L + L(L-1) = L^2 + 1$ buses connected to the above processors (see Figure 3.3). Since each bus can have at most L connections we have $L(1 + L^2) = L^3 + L = \Omega(n^2)$, which implies that $L = \Omega(n^{\frac{2}{3}})$. ■

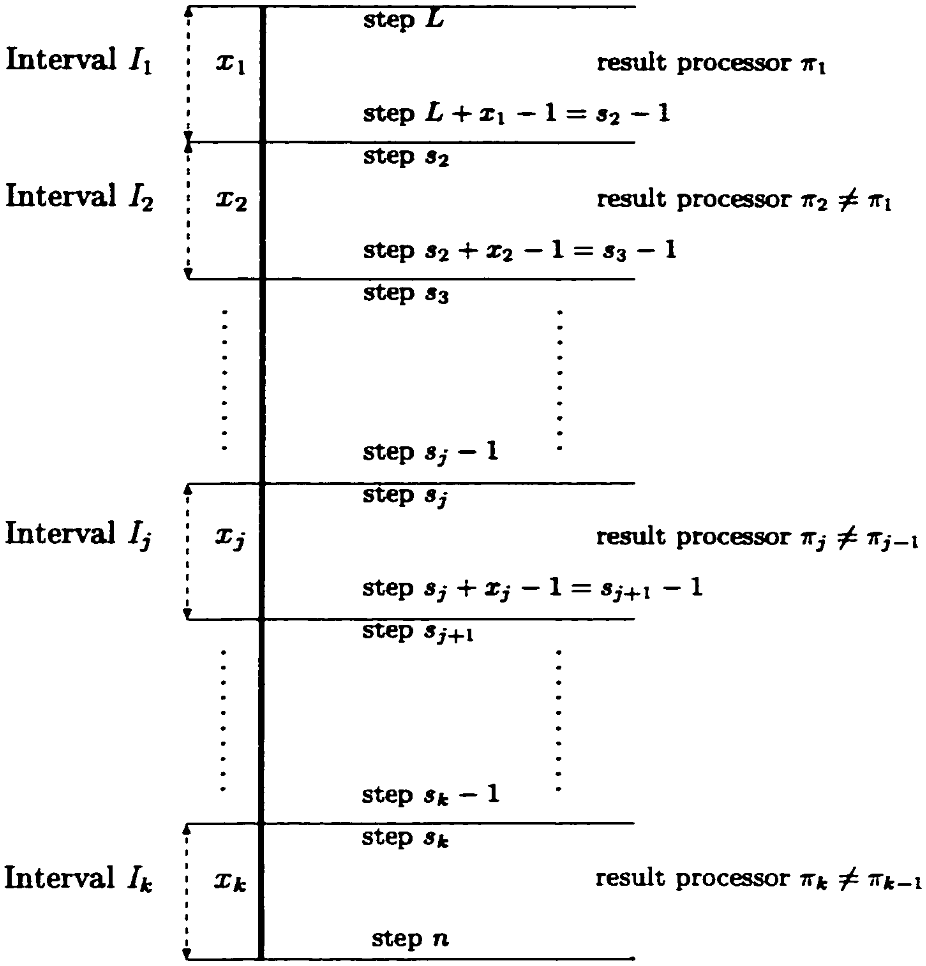

Interval I_1
x_1
step L
result processor π_1
step $L + x_1 - 1 = s_2 - 1$
Interval I_2
x_2
step s_2
result processor $\pi_2 \neq \pi_1$
step $s_2 + x_2 - 1 = s_3 - 1$
step s_3
step $s_j - 1$
Interval I_j
x_j
step s_j
result processor $\pi_j \neq \pi_{j-1}$
step $s_j + x_j - 1 = s_{j+1} - 1$
step s_{j+1}
step $s_k - 1$
Interval I_k
x_k
step s_k
result processor $\pi_k \neq \pi_{k-1}$
step n

Figure 3.4: Subintervals of $[L, n]$

3.5 An $\Omega\left(\frac{n}{\log n}\right)$ Lower Bound

In the last two lower bound derivations (Sections 3.3 and 3.4), we used an accounting scheme to count the number of connections in the neighborhood of a bus b_0 that was active at steps $L, L+1, \cdots, n$. This accounting scheme transferred ownership of one connection for each partial result sent/received by a processor. This scheme assumes the existence of only one transferable connection with each result processor, even though the result processors could possibly own many more connections. In this section we develop a modification to this accounting scheme that allows many more connections to be counted. The new accounting scheme permits the ownership of more than one connection to be transferred between processors, whenever possible. We do this by breaking the interval $[L, n]$ into smaller segments and establishing the existence of more than one transferable connection with result processors in each segment. This is the key to tightening the lower bound on the loading to $\Omega\left(\frac{n}{\log n}\right)$.

In Section 3.5.1 we derive the basic results needed for the new accounting scheme. In Section 3.5.2 we describe the new accounting scheme. Finally in Section 3.5.3 we derive the new lower bound.

3.5.1 Initial Condition

For some integer $d \geq 1$, partition the interval $[L, n]$ into y segments $\underbrace{[L, 3L+1]}_{I_0}$, $\underbrace{[3L+2,\ 3L+1+d]}_{I_1}$, $\underbrace{[3L+2+d,\ 3L+1+2d]}_{I_2}$, $\cdots$, $\underbrace{[3L+2+(y-1)d,\ n]}_{I_y}$. Denote these segments by $I_0, I_1, \cdots, I_y$. Segment I_0 contains $2L+2$ steps, and segments $I_1, I_2, \cdots, I_{y-1}$ each contains d steps. The last segment, I_y, contains $(n-3L-2) - d(y-1) \leq d$ steps. In this section we develop a relationship between the number of result processors and the total number of connections on an active bus at the end of step $3L+1$ (end of interval I_0). Without loss of generality, assume $L \leq \frac{n-2}{3}$.

Lemma 3.14 *Let b be any active bus of steps $L, L+1, \cdots, 3L+1$. Let there be ρ result processors connected to bus b at the end of step $3L+1$. Let there be $\xi \leq L$ full processors (including the ρ result processors) connected to bus b at step $3L+1$. Then $\xi \geq 2\rho$.*

Proof: At step $3L+1$ there are ρ result processors connected to bus b. Therefore, there must be at least ρ (not necessarily the same) result processors connected to bus b all the way from step L (Corollary 3.11). In this interval, ownership of only one connection is transferred with each partial result. Therefore, during this interval of $2L + 2$ steps, the total number of connections owned by processors connected to bus b is increased by at least $(2L + 2)\rho$ (Corollary 3.7, page 30). There is a total of ξ full processors connected to bus b at step $3L + 1$. Each of these ξ processors is connected to a bus different from b. Collectively, the number of connections owned by all ξ processors on bus b is $\xi L + \xi$, where ξL is the connection to the buses in the neighborhood of b, and ξ is the number of connections to b itself. (Recall that a processor can own connections only to the buses it is connected to and all detected connections are owned.) Since the loading of any of these ξ buses in the "neighborhood of bus b" cannot exceed L, we have $\xi(1+L) \geq 2\rho(1+L)$, which implies that $\xi \geq 2\rho$. ■

3.5.2 The New Accounting Scheme

Notice that the result of Lemma 3.14 shows that at the end of step $3L + 1$ there are twice as many connections as result processors to any active bus. However, some result processors may still own only one connection on each of its buses, while other processors may own many more connections. At the end of step $3L+1$, if all the known connections are redistributed among the α result processors, then each processor will own at least 2 connections to each (active) bus b to which it is connected. Therefore, beyond this point ownership of 2 connections could be transacted for each partial result sent/received. This change alone with the previous method of counting the connections will raise the lower bound on L by a factor of about 2. However, if we proceed for another d steps, we can show that the number of connections owned by each result processor is greater that 2. This can be used to evenly redistribute connections so that each result processor now owns more than 2 connections. This in turn allows more connections to be transacted with each partial result. In fact, such a redistribution of the known connections can be carried out at the end of each of the intervals $I_0, I_1, I_2, \cdots I_{y-1}$. The last steps of these intervals (at which known connections are redistributed) are called ***transition points***. In general, if there are α result processors connected to bus b, and a total of γ connections to an active bus b at a transition point, then after redistributing connections, each result processor is guaranteed to own $\lfloor \frac{\gamma}{\alpha} \rfloor$ connections to bus b. The accounting scheme used in Sections 3.3 and 3.4 can still be used with suitable modifications. We now outline these modifications.

1. For $1 \leq i \leq y$, consider interval $I_i = [s_i, s_{i+1} - 1]$, where $s_i = 3L + 2 + (i-1)d$ is the first step of I_i. Clearly $s_i - 1$ is a transition point, so at the end of step S_i the accounting scheme redistributes known connections of each active bus evenly among its result processors. Let each result processor own w connections at step s_i. More precisely, for result processor p with connections to buses b' and b'', let its set of owned connections be $\Gamma_{s_i}(p) = \{\langle p'_1, b'\rangle, \langle p'_2, b'\rangle, \cdots, \langle p'_w, b'\rangle, \langle p''_1, b''\rangle, \langle p''_2, b''\rangle, \cdots, \langle p''_w, b''\rangle\}$ where p'_j and p''_j $(1 \leq j \leq w)$ are some processors connected to bus b' and b'', respectively. If p is not a result processor of step s_i, then $\Gamma_{s_i}(p)$ is empty. Since the set of known connections is partitioned among the result processors, no connection is owned by more than one processor.

2. For a step s (where $s_i < s \leq s_{i+1} - 1$), let p be a result processor of step s that receives partial result(s) from (not necessarily distinct) processor(s) p' and p'' via bus(es) b' and b'', respectively. The sets $\Gamma_s(p)$, $\Gamma_s(p')$ and $\Gamma_s(p'')$ change as

follows.

$$\begin{aligned}
\Gamma_s(p) &= \Gamma_{s-1}(p) \cup \{\langle p'_1, b'\rangle, \langle p'_2, b'\rangle, \cdots, \langle p'_w, b'\rangle\} \cup \\
&\quad \{\langle p''_1, b''\rangle, \langle p''_2, b''\rangle, \cdots, \langle p''_w, b''\rangle\} \\
\Gamma_s(p') &= \Gamma_{s-1}(p') - \{\langle p'_1, b'\rangle, \langle p'_2, b'\rangle, \cdots, \langle p'_w, b'\rangle\} \\
\Gamma_s(p'') &= \Gamma_{s-1}(p'') - \{\langle p''_1, b''\rangle, \langle p''_2, b''\rangle, \cdots, \langle p''_w, b''\rangle\}
\end{aligned}$$

For $1 \leq j \leq w$, $\langle p'_j, b\rangle \in \Gamma_{s-1}(p')$ and $\langle p''_j, b''\rangle \in \Gamma_{s-1}(p'')$; since we are interested primarily in the cardinality, $|\Gamma_s(p)|$, of $\Gamma_s(p)$, the w connections transferred ($\langle p'_j, b'\rangle$ and $\langle p''_j, b''\rangle$, for $1 \leq j \leq w$) can be any w element of $\Gamma_{s-1}(p')$ and $\Gamma_{s-1}(p'')$. Note that if p is a direct processor that receives only one partial result in step s, then $p' = p''$, $b' = b''$ and $p'_j = p''_j$.

In summary, for each partial result received by processor p from processor p' via bus b, processor p' transfers ownership of w connections on bus b to processor p. We call w the *transaction weight* of interval I_i. If processor p does not send or receive any partial result in step s, then $\Gamma_s(p) = \Gamma_{s-1}(p)$. The facts stated in Lemma 3.5 also hold when the transaction weight is more than one. We restate Lemma 3.5 modified to accommodate the idea of transaction weight; its proof is the same as that of Lemma 3.5.

Lemma 3.15 *Let I be an interval with transaction weight w. For any step $s \in I$, the following statements hold.*

(*i*) *For distinct processors p_1, p_2, $\Gamma_s(p_1)$ and $\Gamma_s(p_2)$ are disjoint.*

(*ii*) *For any processor p, if $\langle p', b\rangle \in \Gamma_s(p)$, then processor p is connected to bus b.*

(*iii*) *If p is a result processor of step s, then $\Gamma_s(p)$ has w connections of the form $\langle p'_1, b\rangle, \langle p'_2, b\rangle, \cdots, \langle p'_w, b\rangle$ for each bus b to which p is connected.*

■

Lemma 3.6 tracks the size of $\Gamma_s(p)$. We now restate this lemma with appropriate modifications for the new accounting scheme. Again its proof parallels that of Lemma 3.6.

Lemma 3.16 *Let I be an interval with transaction weight w. For any step $s \in I$, $|\Gamma_s(p)| = |\Gamma_{s-1}(p)| + \delta$, where*

$$\delta = \begin{cases} w, & \textit{if } p \textit{ is a result processor of steps } s \textit{ and } s-1, \\ 2w, & \textit{if } p \textit{ is a result processor of step } s \textit{ and a non-result processor} \\ & \textit{of step } s-1, \\ -w, & \textit{if } p \textit{ is a non-result processor of step } s \textit{ and a result processor} \\ & \textit{of step } s-1, \\ 0, & \textit{if } p \textit{ is a non-result processor of steps } s \textit{ and } s-1. \end{cases}$$

■

Corollary 3.7 (page 30) relates the number of connections owned by a processor and the number of steps for which it holds a partial result. We now restate Corollary 3.7 with suitable modifications to accommodate a transaction weight $w \geq 1$.

Corollary 3.17 *Let I be an interval with transaction weight w. For any step $s \in I = [s_1, s_2]$, if p is a result processor of α of the steps of subinterval $[s_1, s]$ of I, then $|\Gamma_s(p)| \geq w\alpha$.*

Proof: The proof follows along the same lines as the proof of Corollary 3.7 (page 30) with each connection transferred replaced by a group of w connections. With each partial result, w connections are transferred. The remaining steps of the proof are the same as in Corollary 3.7. ■

3.5.3 Tighter Lower Bound

We now use the results developed so far to derive a tighter lower bound on the loading of degree-2, optimal-time, binary-tree MBNs. Recall that the initial segment $I_0 = [L,\ 3L+1]$, the last segment is $I_y = [3L+2+(y-1)d,\ n]$ and for $1 \leq i \leq y$, $I_i = [3L+2+(i-1)d,\ 3L+1+id]$.

Lemma 3.18 *Let b_0 be an active bus of steps $L, L+1, \cdots, n$. For $0 \leq i < y$, let w_i be the transaction weight of segment I_i. Then, $w_0 = 2$ and $w_{i+1} = \left\lfloor \frac{dw_i}{L+1} \right\rfloor$.*

Proof: Lemma 3.14 proves that $w_0 = 2$. Let there be ρ_{i+1} result processors connected to bus b_0 at the beginning of interval I_{i+1}. Let ξ_{i+1} be the total number of processors (including the ρ_{i+1} result processors) connected to bus b_0 at the beginning of the interval I_{i+1}. Since there are ρ_{i+1} result processors connected to bus b_0 at the beginning of the interval I_{i+1}, there must be at least ρ_{i+1} (possibly different) result processors connected to bus b_0 at each of the steps of interval I_i (Lemma 3.11). The number of connections collectively owned by these processors at the beginning of the interval I_{i+1} is at least $dw_i\rho_{i+1}$ (Corollary 3.17). These connections must be on bus b_0 or buses in its neighborhood (that are connected to a processor with a connection to bus b_0) as shown in Figure 3.5. The number of connections on bus b_0 is ξ_{i+1}. Each of the ξ_{i+1} has at most L connections. Therefore, the number of connections on b_0 and buses in its neighborhood is at most $\xi_{i+1} + L\rho_{i+1} = (1+L)\rho_{i+1}$. Since the number of connections owned by the ρ_{i+1} processors on b_0 cannot exceed this quantity, we have $w_i\rho_{i+1}d \leq \xi_{i+1}L + \xi_{i+1} = (1+L)\xi_{i+1}$. By definition, $w_{i+1} = \left\lfloor \frac{\xi_{i+1}}{\rho_{i+1}} \right\rfloor = \left\lfloor \frac{dw_i}{1+L} \right\rfloor$. ■

Lemma 3.19 *For $0 \leq i < y$, let w_i be the transaction weight for segment I_i. If $\frac{d}{1+L}$ is an integer, then $w_{i+1} = 2\left(\frac{d}{L+1}\right)^i$.*

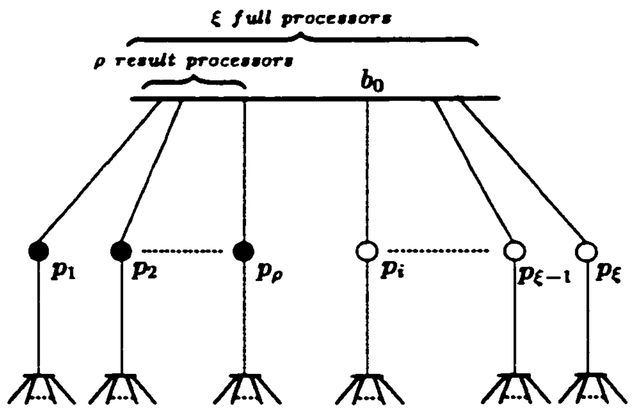


Figure 3.5: The connections on bus b_0

Proof: If $\frac{d}{1+L}$ is an integer, then $\left\lfloor \frac{dw_i}{1+L} \right\rfloor = \frac{dw_i}{1+L}$. Then from Lemma 3.18, $w_{i+1} = \frac{dw_i}{1+L}$, and $w_0 = 2$. Solving this recurrence will yield $w_{i+1} = 2\left(\frac{d}{1+L}\right)^i$. ■

Now we are in a position to prove the main theorem of this section.

Theorem 3.20 *For $n \geq 2$, if a degree-2, loading-L, 2^n-processor MBN runs a $Bin(n)$ in n steps, then $L = \Omega(\frac{n}{\log n})$.*

Proof: Let $d = 2(1 + L)$. Using the same notation as in the proof of Lemma 3.18, this gives $w_y = 2^{y+1} = \lfloor \frac{\xi_y}{\rho_y} \rfloor$. Since $\rho_y \geq 1$, we have $\xi_y \geq 2^{y+1}$. That is, there are at least 2^{y+1} connection to bus b_0 that is active in each of the steps of I_y. Therefore $2^{y+1} \leq L$, hence, $y + 1 = \left\lfloor \frac{n-(3L+2)}{2(1+L)} \right\rfloor + 1 \leq \log L$. Thus $L \log L = \Omega(n)$ which implies that $L = \Omega(\frac{n}{\log n})$. ■

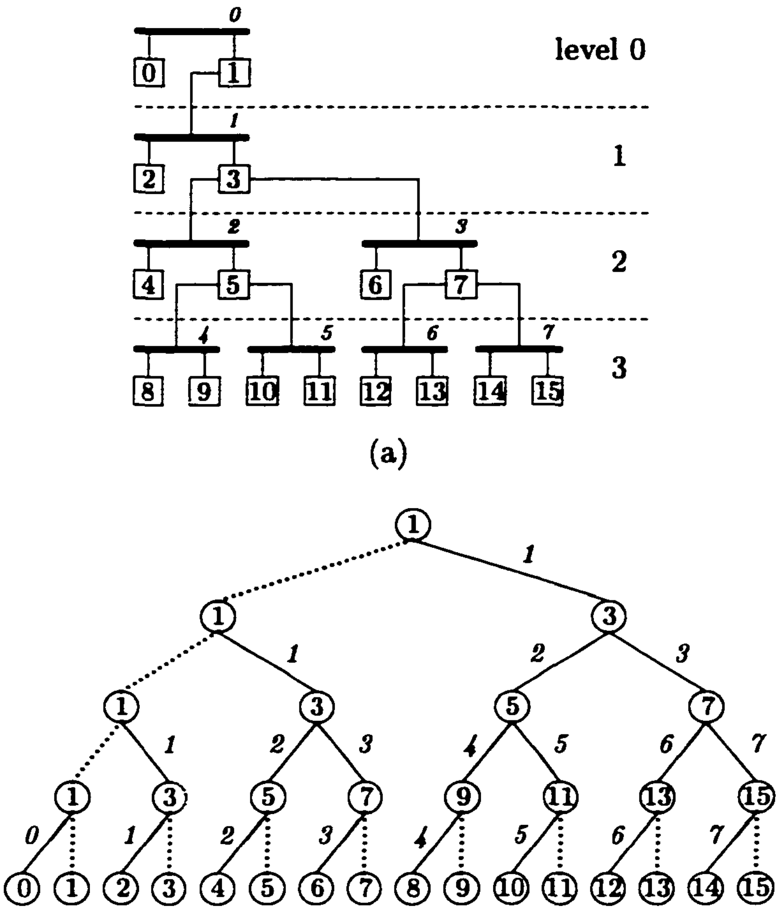
3.6 The Tree MBN

We showed in the previous sections that for $n \geq 1$, any MBN with at least 2^{n-1} buses running $Bin(n)$ has a degree of at least 2 and a loading of at least 3. Next we proved that if an MBN runs $Bin(n)$ in n steps and if its degree is 2, then its loading is

$\Omega\left(\frac{n}{\log n}\right)$. This raises the question, what if the degree is 3? In this section we answer this question positively by constructing a binary-tree MBN with the least possible degree-loading product. For $n \geq 1$, we present a $2^n \times 2^{n-1}$ MBN $\mathcal{T}(n)$, called the *Tree MBN*, that runs $Bin(n)$ in n steps. The degree and loading of $\mathcal{T}(n)$ are each 3. Lemma 3.1 and Theorem 3.20 prove that $\mathcal{T}(n)$ has the best possible degree-loading product of 9. Figure 3.6(a) shows a $\mathcal{T}(4)$ and Figure 3.6(b) shows how it runs $Bin(4)$. We now formally describe $\mathcal{T}(n)$. For $n = 1$, each of the two processors of $\mathcal{T}(1)$ is connected to the only bus. For the remaining description, we assume that $n \geq 2$.

Let the processors and buses of $\mathcal{T}(n)$ be indexed $0, 1, \cdots, 2^n - 1$ and $0, 1, \cdots, 2^{n-1} - 1$, respectively. Group processors and buses into 2^{n-1} clusters, C_i, where $0 \leq i < 2^{n-1}$. Cluster C_i consists of bus i and processors $2i$ and $2i+1$, both of which are connected to bus i. Arrange the 2^{n-1} clusters into n levels (see Figure 3.6(a)). Level 0 contains only cluster C_0. For $0 < \ell < n$, level ℓ contains clusters C_x, for $2^{\ell-1} \leq x < 2^{\ell}$. In addition to connections from processors $2i$ and $2i+1$ to bus i (where $0 \leq i < 2^{n-1}$), $\mathcal{T}(n)$ has the following connections between clusters. Processor 1 is connected to bus 1; for $1 \leq i < 2^{n-2}$, processor $2i+1$ of cluster C_i is connected to buses $2i$ and $2i+1$. It is easy to see that for $0 \leq i < 2^{n-1}$, processor $2i$ is connected only to bus i, and processor $2i+1$ is connected only to buses i, $2i$ and $2i+1$ (if they exist). Similarly for $0 \leq i < 2^{n-1}$, bus i is connected only to processors $2i$, $2i+1$ and $\left(2\left\lfloor\frac{i}{2}\right\rfloor + 1\right)$. Thus the degree and loading of $\mathcal{T}(n)$ are each 3.

Figure 3.6(b), with nodes and non-trivial edges labeled with processor and bus indices, respectively, shows how $\mathcal{T}(4)$ runs $Bin(4)$. The general case is a straightforward extension of this, so we will keep this description brief. In running $Bin(n)$ on $\mathcal{T}(n)$, each processor initially holds an input. The first step consists of a communication within each cluster, with processor $2i+1$ receiving an input from processor $2i$



(a)

(b)

Figure 3.6: Running $Bin(4)$ on $\mathcal{T}(4)$

(via bus i) and performing the operation $\circ$. Subsequent steps involve communication between clusters. In step s (where $2 \leq s \leq n$), processor $2i + 1$ (where $0 \leq i < 2^{n-s}$) sends a partial result to processor $\left(2\left\lfloor \frac{i}{2} \right\rfloor + 1\right)$, receives partial results from processors $4i + 1$ and $4i + 3$, and applies the operation $\circ$ on the partial results received. At the end of step n, processor 1 holds the result of $Bin(n)$.

Theorem 3.21 *For any $n \geq 1$, the $2^n \times 2^{n-1}$ MBN, $\mathcal{T}(n)$, runs $Bin(n)$ optimally in n steps. The degree and loading of $\mathcal{T}(n)$ are each* 3. ■

3.7 Loading-Speed Tradeoff

In the preceding sections we proved that an optimal-time, degree-2 binary-tree MBN cannot have constant loading. In Section 3.6 we showed that this lower bound on loading does not hold if the degree is permitted to increase to 3. This section deals with the "optimal-time" restriction on these results. We prove that if the algorithm is permitted to run a little more slowly, then a constant loading is possible, even with a degree of 2. Specifically we show the existence of a degree-2, loading-4 MBN that runs $Bin(n)$ in $n + t$ time when t is $\Theta(n)$. When t is constant, however, the loading cannot be constant. We derive a lower bound that relates the loading with t, the amount of time beyond the optimal that the MBN is permitted to execute the binary-tree algorithm.

3.7.1 Lower Bound

Let $2^{\tau(L)}$ be the size of the largest instance of a binary-tree algorithm that can be run optimally in $\tau(L)$ steps on a degree-2, loading-L MBN. From Theorem 3.20 we know that $\tau(L) = O(L \log L)$. The existence of a degree-2, $\Theta(n)$-loading, optimal-time MBN for $Bin(n)$ [85] gives the bound $\tau(L) = \Omega(L)$.

Theorem 3.22 *For any degree-2, loading-L, 2^n-processor MBN that runs $Bin(n)$ in $n + t$ steps, $t \geq \left\lfloor \frac{n}{\tau(L)+1} \right\rfloor$.*

Proof: Since the MBN runs $Bin(n)$ suboptimally, there are some nodes of $\mathcal{F}(n)$ with "delays" in them. A node with delay δ passes its value (input/partial result) to its parent δ steps after this value is available to it. This delay may be used to transfer the value to a different processor with less demands in its buses.

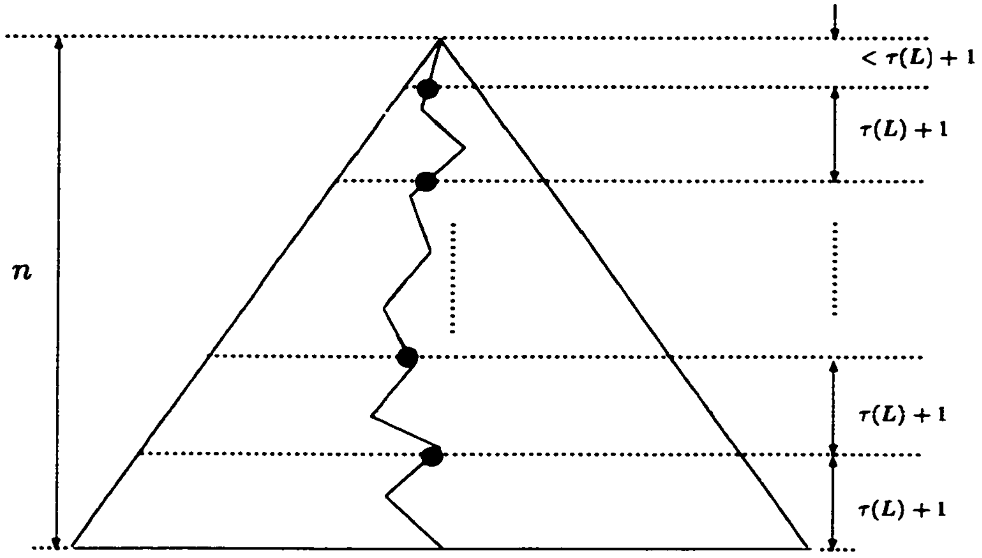


Figure 3.7: Path with t delays

The definition of $\tau(L)$ ensures that for $x > \tau(L)$, the tree $\mathcal{F}(x)$ has at least one delayed node in it; therefore $Bin(x)$ requires at least $x + 1$ steps on such an MBN. Divide $\mathcal{F}(n)$ into $\left\lceil \frac{n}{\tau(L)+1} \right\rceil$ parts (see Figure 3.7), P_i, where $1 \leq i \leq \left\lceil \frac{n}{\tau(L)+1} \right\rceil$. Each part, P_i, (except perhaps the last one) consists of $\tau(L) + 1$ contiguous levels of $\mathcal{F}(n)$. That is, these parts contain trees isomorphic to $\mathcal{F}(\tau(L) + 1)$ that must contain at least one delayed node (with delay $\delta \geq 1$). For part P_1 that starts from the leaves, the roots of the $\mathcal{F}(\tau(L) + 1)$s in this part obtain the values no earlier than at step $\tau(L) + 2$. As a result, the roots of the $\mathcal{F}(\tau(L) + 1)$s in the next part P_2 obtain their values no earlier than at step $2(\tau(L) + 2)$. In general for $1 \leq i \leq \left\lfloor \frac{n}{\tau(L)+1} \right\rfloor$, the roots of the $\mathcal{F}(\tau(L) + 1)$s of part P_i obtain their values no earlier than at step $i(\tau(L) + 2)$. Without the delayed nodes, these roots would have obtained their values at step $i(\tau(L) + 1)$, so the additional time taken is at least i. Therefore, the additional time (due to delayed nodes) taken before the root of $\mathcal{F}(n)$ obtains its value is at least $\left\lfloor \frac{n}{\tau(L)+1} \right\rfloor$. ■

3.7.2 Upper Bound

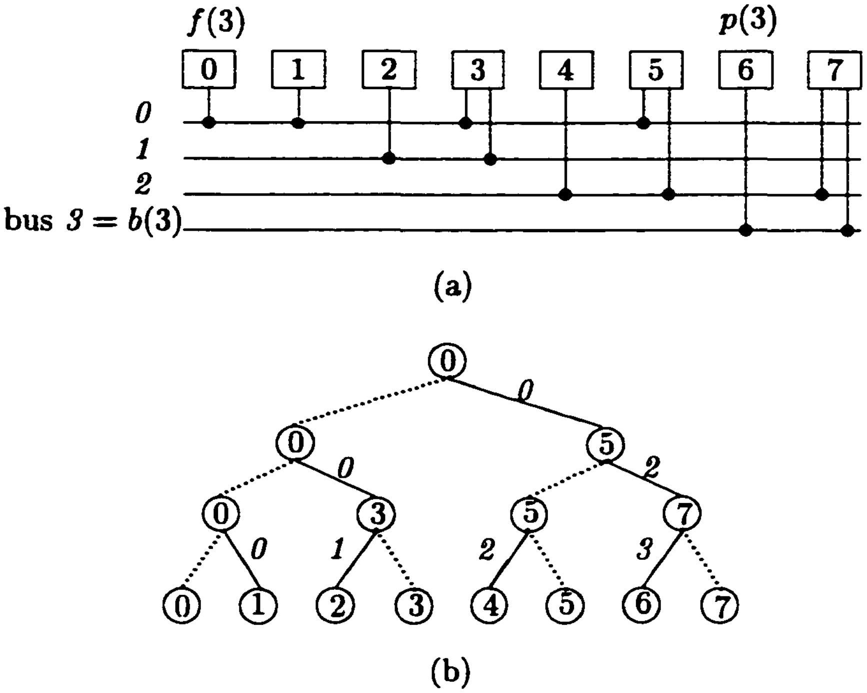
When the loading L is a constant, Theorem 3.22 requires the additional time (beyond the minimum required n steps) to run $Bin(n)$ to be $\Omega(n)$. This is because $\tau(L) = O(L \log L) = O(1)$, for constant L. We now show that this lower bound on the additional time is tight by presenting a degree 2, loading-4 MBN that runs $Bin(n)$ in $2n-3$ steps (that is, with $n-3$ additional steps).

Consider a degree-2, constant loading-ℓ, $2^n \times 2^{n-1}$ *delayed root* MBN, $\mathcal{D}(n)$, that has the following properties:

- The processor, $f(n)$, that holds the final result of $Bin(n)$ has a degree of 1; that is, $f(n)$ is connected to only one bus.
- There is a bus $b(n)$ with loading $\ell - 2$; that is, two more processors could be connected to bus $b(n)$ without increasing the loading of $\mathcal{D}(n)$.
- One of the processors, $p(n)$, connected to bus $b(n)$ has degree 1; that is, processor $p(n)$ is not connected to any bus other than $b(n)$.

Processors $f(n)$ and $p(n)$ and bus $b(n)$ will be called the *special elements* of $\mathcal{D}(n)$. An example of such an MBN is the 8×4 MBN shown in Figure 3.8(a). For this MBN the loading ℓ is 4, and $f(3)$, $p(3)$ are processors 0 and 6, respectively, while bus *3* is $b(3)$. It is easy to verify, that $\mathcal{D}(3)$ possesses the above properties. Also note that $\mathcal{D}(3)$ runs $Bin(3)$ in 3 steps.

We now show how two copies of $\mathcal{D}(n)$ can be used to construct the $2^{n+1} \times 2^n$ MBN, $\mathcal{D}(n+1)$. To distinguish these copies, we name them $\mathcal{D}'(n)$ and $\mathcal{D}''(n)$ and refer to



(a)

(b)

Figure 3.8: Running $Bin(3)$ on 8 × 4 MBN, $\mathcal{D}(3)$

their special elements as $f'(n)$, $p'(n)$ $b'(n)$, $f''(n), p''(n)$ and $b''(n)$. To construct $\mathcal{D}(n+1)$ from $\mathcal{D}'(n)$ and $\mathcal{D}''(n)$, all that needs to be done is to connect processors $f'(n)$ and $f'(n)$ to bus $b'(n)$. Designate $p'(n)$ to be $f(n+1)$, $p''(n)$ to be $p(n+1)$ and $b''(n)$ to be $b(n+1)$.

MBN $\mathcal{D}(n+1)$ has a loading of ℓ as the only two added connections are to bus $b'(n)$ that has only $\ell-2$ connections to start with. Its degree is 2 as the added connections are one each from processors $f'(n)$ and $f''(n)$, that had only one connection each to begin with. It is easy to verify that each of the processors $f(n+1) = p_1(n)$ and $p(n+1) = p''(n)$ has degree 1 and that bus $b(n+1) = b''(n)$ has loading $\ell - 2$. If we can now establish that $f(n+1)$ holds the final result, then $\mathcal{D}(n+1)$ will satisfy the three properties stated above for $\mathcal{M}(n)$.

When $Bin(n+1)$ is run on $\mathcal{D}(n+1)$, the first n levels (starting from the leaves) are run simultaneously on $\mathcal{D}'(n)$ and $\mathcal{D}''(n)$; the results of these steps are in processors $f'(n)$ and $f''(n)$. These processors use bus $b'(n)$ to send their partial results to $f(n+1) = p'(n)$. (Recall that processors $f'(n)$ and $f''(n)$ have been connected to bus $b'(n)$, while $f(n+1) = p'(n)$ is already connected to this bus.) Processor $f(n+1)$ now computes the final value of $Bin(n+1)$. From this discussion it should also be evident that if $\mathcal{D}(n)$ runs $Bin(n)$ in $T(n)$ steps, then $\mathcal{D}(n+1)$ runs $Bin(n+1)$ in $T(n+1) = T(n) + 2$ steps. This is because both processors $f'(n)$ and $f''(n)$ use the same bus, $b'(n)$, to send their partial results to processor $f(n+1)$; this introduces a delay in computing the root. Coupled with the fact that $T(3) = 3$ (see Figure 3.8), this gives $T(n) = 2n - 3$, for $n \geq 3$.

Thus we have the following result.

Lemma 3.23 *For any $n \geq 3$, the degree-2, loading-4, $2^n \times 2^{n-1}$ delayed root MBN, $\mathcal{D}(n)$, runs $Bin(n)$ in $2n-3$ steps.* ■

The generalization of the above result to non-constant loading $L > 4$ is straightforward. First construct a $2^L \times 2^{L-1}$ degree-2, loading-L MBN, by the method proposed by Vaidyanathan and Padmanabhan [85]). Denote this MBN by $\mathcal{D}_L$. An important feature of the MBN $\mathcal{D}_L$ is that the result processor has only one connection, and the bus that is connected to the final result processor has less than L connections. We can now use two copies of $\mathcal{D}_L(n-1)$ and follow the construction described in the upper bound section to obtain $\mathcal{D}_L(n)$. It is easy to see that the time it takes to run $Bin(n)$ on $\mathcal{D}_L(n)$ is $L - 1 + 2(n - L + 1) = 2n - L + 1$ steps.

Theorem 3.24 *For any $n \geq 3$, the degree-2, loading-L, $2^n \times 2^{n-1}$ MBN, $\mathcal{D}_L(n)$, runs $Bin(n)$ in $2n - L + 1$ steps.* ■

The results of this section show that a degree-2, constant loading MBN exists for $Bin(n)$ if and only if it is permitted to take $\Theta(n)$ more steps than the optimal.

3.8 Extension to k-ary Tree Algorithms

In this section, we extend the lower bound results of this chapter to k-ary tree algorithms. In a k-ary algorithm, a node (or a processor) receives k-inputs/partial results and reduces them to one result by performing an associative k-ary operation on them. Binary-tree algorithms form a special case of k-ary tree algorithms, with $k = 2$. All the results so far have been established for binary-tree algorithms.

In general, for $k \geq 2$, a k-ary tree algorithm reduces k^n inputs to one result, and can be represented as a balanced (n-level) k-ary tree. Clearly, the optimal time for this algorithm is n steps. In this section we extend the lower bound results of Sections 3.3, 3.4 and 3.5 to k-ary tree algorithms. Most of the basic results needed to establish the lower bounds for k-ary tree algorithms are very similar to the binary tree case. We will therefore keep our discussion brief and only point out places where the k-ary case differs from the binary case.

Let $X(n)$ be an optimal-time, loading-L, k-ary tree MBN with k^n processors and $M \geq (k-1)k^{n-1}$ buses. We start by stating the basic results of Section 3.3.2 (page 26) extended to k-ary tree MBNs. We list four simple consequences of $X(n)$ being a degree-k, optimal-time, k-ary tree MBN that are used often without explicit mention, in subsequent discussion.

1. A full processor has k connections, while a non-full processor has less than k connections.

2. All partial results received in a step are used in the same step, and a partial result generated in a step is used up in the next step. This is because the algorithm runs optimally, so partial results cannot idle.

3. A processor using a direct (resp., indirect) mapping receives $k-1$ (resp., k) partial results; this follows from 2 above.

4. A processor receiving k partial results at a step must do so from different processors. Otherwise the step will not be executed in unit time.

5. A processor sending a partial result cannot receive one at the same step. This is because it will have to receive k and send one partial result. This is not possible on a degree-k, optimal-time MBN.

It is easy to show that Lemma 3.3 and Corollary 3.4 hold for the k-ary case. Define ownership as in Section 3.3.3 with one connection being transacted for each partial result sent/received. Clearly Lemma 3.5 still holds. Lemma 3.6 clearly extends to the following:

Lemma 3.25 *For any step $s > L$ and any processor p, $\Gamma_s(p) = \Gamma_{s-1}(p) + \delta$, where*

$$\delta = \begin{cases} k-1, & \textit{if } p \textit{ is a result processor of steps } s \textit{ and } s-1 \\ k, & \textit{if } p \textit{ is a result processor of step } s \textit{ and a non-result processor} \\ & \textit{of step } s-1 \\ -1, & \textit{if } p \textit{ is a non-result processor of step } s \textit{ and a result processor} \\ & \textit{of step } s-1 \\ 0, & \textit{if } p \textit{ is a non-result processor of steps } s \textit{ and } s-1 \end{cases}$$

■

As a result, we have the following corollary using the reasoning of Corollary 3.7.

Corollary 3.26 *For any step $s \geq L$, if p is a result processor of α of the steps $L, L+1, \cdots, s$, then $|\Gamma_s(p)| \geq \alpha(k-1)$.* ■

Observe that Lemma 3.8 is independent of the k-ary case.

Theorem 3.27 *For any $n \geq 2$, if a k^n-processor MBN with of degree-k and loading of L, runs k^n-input, k-ary tree algorithm optimally in n steps, then $L = \Omega(\sqrt{n})$.*

Proof: From Lemma 3.8, there exists bus, b_0, that is active in steps $L, L+1, \cdots, n$. Let b_0 be connected to $\ell \leq L$ processors, $p_1, p_2, \cdots, p_\ell$, all of which are full processors of step n. For $1 \leq i \leq \ell$, let the k buses to which processor p_i is connected be b_0 and $b_{i,1}, b_{i,2}, \cdots, b_{i,k-1}$. Also let processor p_i be a result processor α_i times from step L to step n.

From Lemma 3.5(ii), each element of $\Gamma_n(p_i)$ is a connection to either b_0 or b_i. Since the loading of the MBN is L, $\sum_{i=1}^{\ell} |\Gamma_n(p_i)| \leq \ell + \ell(k-1)L \leq L^2(k-1) + L$. From Corollary 3.26 we also have $\sum_{i=1}^{\ell} |\Gamma_n(p_i)| \geq (k-1)\sum_{i=1}^{\ell} \alpha_i$. Since b_0 is an active bus of steps $L, L+1, \cdots, n$, $\sum_{i=1}^{\ell} \alpha_i \geq n - L + 1$. Thus, $(k-1)(n-L+1) \leq L^2(k-1) + L$, which implies that $n \leq L^2 + \frac{Lk}{k-1} - 1$ or $L = \Omega(\sqrt{n})$. ■

To obtain the second lower bound from the first, observe that Lemma 3.10 and Corollary 3.11 hold for the k-ary case as stated. Lemma 3.12 changes slightly as shown below.

Lemma 3.28 *For $s > L$, if p is not a result processor of step $s-1$, and is a result processor of steps $s, s+1, \cdots, s+x-1$ (for some $x > 0$), then the following assertions hold:*

(*i*) *For each step s' in the interval $[L, s-1]$, at least $x+1$ neighbors of p are result processors of step s'.*

(*ii*) *At the end of step $s+x-1$, the neighbors of p collectively own at least $(k-1)\left((x+1)(s-L)+\frac{x(x-1)}{2}\right)$ connections.*

Proof: Is the same as in Lemma 3.12 except that the number of connection owned by a processor increases by a factor of $k-1$ for every result holding step (Corollary 3.26).

■

Theorem 3.29 *For any $n \geq 2$, if a k^n-processor MBN with degree of k and loading of L runs a k-ary tree algorithm optimally in n step, then $L = \Omega\left(\frac{n^2}{k}\right)^{\frac{1}{3}}$.*

Proof: This proof is the same as in Lemma 3.13 with some modifications as indicated below. The number of connections collectively owned by processors $p_{1,1}$, $p_{1,2}$, $\cdots$, $p_{1,k-1}$ $\cdots$, $p_{\ell,1}$, $p_{\ell,2}$, $\cdots$, $p_{\ell,k-1}$ and their neighbors is at least

$$(k-1)\sum_{j=1}^{k}\left((s_j-L)(x_j+1)+\frac{x_j(x_j-1)}{2}\right) = \Theta((k-1)(n-L)^2) = \Theta((k-1)n^2).$$

These connections are distributed among $1+(k-1)\ell+ (k-1)^2(L-1)\ell \leq 1+L(k-1)+L(L-1)(k-1)^2$ buses connected to the above processors. Since each bus can have at most L connections we have

$$L\left[1+L(k-1)+L(L-1)(k-1)^2\right] = \Theta((k-1)^2L^3) = \Omega((k-1)n^2)$$

, which implies that $L = \Omega\left(\frac{n^2}{k}\right)^{\frac{1}{3}}$. ■

Remark: For constant k, L is still $\Omega(n^{\frac{2}{3}})$.

The new accounting scheme developed for proving the $\Omega(\frac{n}{\log n})$ lower bound of Section 3.5 is also valid for the k-ary tree algorithms. We state results from Section 3.5 that require some changes.

Lemma 3.30 *For $k > 2$, let b be any active bus of steps $L, L+1, \cdots, 3L+2$. Let there be ρ result processors connected to bus b at the end of step $3L+2$. Let there be $\xi \leq L$ full processors (including the ρ result processors) connected to bus b at step $3L+2$. Then $\xi \geq 2\rho$.*

<u>Proof:</u> At step $3L+2$ there are ρ result processors connected to bus b_0. Therefore, there must be at least ρ (not necessarily the same) result processors connected to bus b_0 all the way from step L (Corollary 3.11). In this interval, ownership of only one connection is transferred with each partial result. Therefore, during this interval of $2L+2$ steps, the total number of connections owned by processors connected to bus b_0 is increased by at least $(2L+2)(k-1)\rho$ (Corollary 3.25). There is a total of ξ full processors connected to bus b_0 at step $3L+2$. Each of these ξ processors is connected to $(k-1)$ buses different from b_0. Collectively, the number of connections owned by all ξ processors on bus b_0 is $\xi + \xi(k-1)L$, where $\xi(k-1)L$ is the connection on the buses in the neighborhood of b_0 and ξ is the number of connections on b_0 itself. Since the loading of any of these ξ buses in the "neighborhood of bus b_0" cannot exceed L, $\xi(1+(k-1)L) \geq \rho(k-1)(2L+2)$, which implies that $\xi \geq 2\rho$, or $w_0 \geq 2$. ■

For the same partitioning of the interval $[L, n]$ in Section 3.5.1, we have the following.

Lemma 3.31 *Let b_0 be an active bus of steps $L, L+1, \cdots, n$. For $0 \leq i < w$, let w_i be the transaction weight for segment I_i. Then, $w_0 = 2$ and $w_{i+1} = \left\lfloor \frac{d}{(k-1)L+1} \right\rfloor w_i$.*

Proof: Lemma 3.30 proves that $w_0 = 2$. Rest of the proof is the same as in Lemma 3.18. However, notice that each processor in the neighborhood of the bus is now connected to $k-1$ other buses. Therefore,

$w_i \rho_{i+1} d \leq \xi_{i+1}(k-1)L + \xi_{i+1} = (1 + L(k-1))\xi_{i+1}$, so $\frac{\xi_{i+1}}{\rho_{i+1}} \geq \frac{w_i d}{1+L(k-1)}$

By definition, $w_{i+1} = \left\lfloor \frac{\xi_{i+1}}{\rho_{i+1}} \right\rfloor = \left\lfloor \frac{d}{1+L(k-1)} w_i \right\rfloor$. ■

Lemma 3.32 *Let b_0 be an active bus of steps $L, L+1, \cdots, n$. For $0 \leq i < w$, let w_i be the transaction weight for segment I_i. If $\frac{d}{1+L(k-1)}$ is an integer, then $w_{i+1} = 2\left(\frac{d}{(k-1)L+1}\right)^i$.*

Proof: If $\frac{d}{1+L(k-1)}$ is an integer, then $\left\lfloor \frac{d}{1+L(k-1)} \right\rfloor = \frac{d}{1+L(k-1)}$. Then from Lemma 3.31, $w_{i+1} = w_i \frac{d}{1+L(k-1)}$, and $w_0 = 2$. Solving this recurrence will yield $w_{i+1} = 2\left(\frac{d}{1+L(k-1)}\right)^i$.

■

Theorem 3.33 *For $n \geq 2$, if a degree-k, loading-L, k^n-processor MBN runs k^n-input k-ary tree algorithm in n steps, then $L = \Omega(\frac{\frac{n}{k}}{\log n - \log k})$.*

Proof: Let $d = 2(1+L(k-1))$. This gives $w_y = 2^{y+1} = \lfloor \frac{\xi_y}{\rho_y} \rfloor$. Since $\rho_y \geq 1$, $\xi_y \geq 2^{y+1}$. That is 2^{y+1} connection to a bus. Therefore, $2^{y+1} \leq L$, and $\left\lfloor \frac{n-3L-\frac{2}{k-1}}{2(1+L(k-1))} \right\rfloor < \log L$. Thus $kL \log L = \Omega(n)$ which implies that $L = \Omega(\frac{\frac{n}{k}}{\log n - \log k})$. ■

Remark: For constant k, L is still $\Omega(\frac{n}{\log n})$.

We now outline the construction of a Tree MBN for k-ary tree algorithms. Figure 3.9 shows an example when $k = 3$. The degree of this MBN is 4 and its loading is 3. In general, the k-ary tree MBN has k^n processors, $(k-1)k^{n-1}$ buses, and a degree of $k+1$. The loading is always 3.

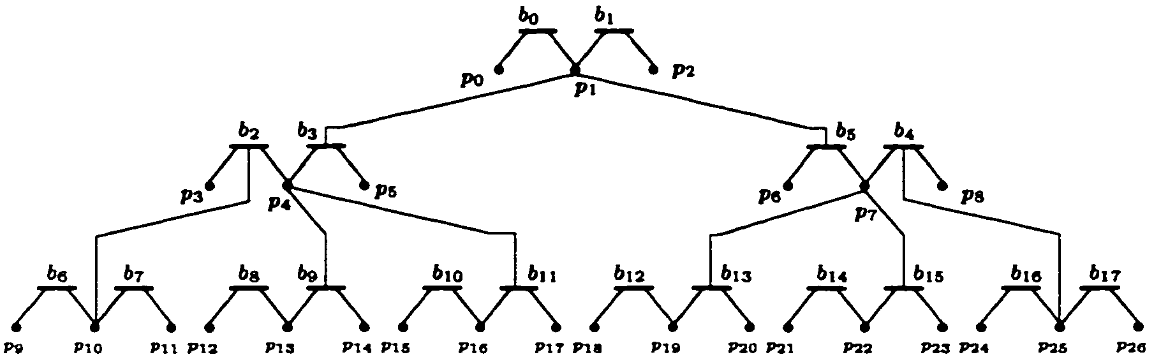


Figure 3.9: MBN for ternary tree algorithms

3.9 Concluding Remarks

We have proved that for any degree-2 MBN that runs $Bin(n)$ optimally, its loading is $\Omega(\frac{n}{\log n})$. If the MBN uses a direct mapping, then we have proved that the bound on the loading is $\Omega(n)$. This is a tight bound as there exists an MBN with such a loading [85]. We have also shown a tradeoff between the speed and loading of degree-2 MBNs. In particular, we have proved that a degree-2, constant loading MBN can run $Bin(n)$ if and only if it is permitted to take $\Theta(n)$ steps more than the optimal. We conjecture that an optimal-time, degree-2, binary-tree MBN has an $\Omega(n)$ loading lower bound. If this is the case, then the $\Omega(n)$ MBN of Vaidyanathan and Padmanabhan [85] is optimal for degree-2 MBNs.

Chapter 4

Multiple-Bus Enhanced Meshes

Over the last decade several topologies have been proposed for connecting processors in parallel systems. Of these, the two dimensional mesh has emerged as one of the most widely studied, due, in part, to its regular structure and simple layout in two dimensions. A disadvantage of the mesh is its large diameter, which is often the bottleneck of many fundamental algorithms. To circumvent this problem, while building on the advantages of the mesh, researchers have proposed meshes enhanced with buses (for example [1, 7, 8, 11, 13, 19, 27, 30, 33, 51, 71, 72, 75]). Most such enhancements carefully select the set of processors to connect, but employ an overly simple method to connect elements of that set. In this chapter we demonstrate the advantage of connecting these sets using MBNs in general, and binary-tree MBNs in particular.

A general idea in enhanced meshes is to identify (not necessarily disjoint) sets of processors of the mesh, and then connect processors in each set by a single bus. We will refer to these sets as *connect-sets*. Typically, a connect-set consists of processors in a row (or column) of the mesh [1, 7, 10, 30, 51], or variants of this idea such as every x^{th} processor of a row (or column), for some integer x [4, 17, 19, 75]. Hierarchical approaches [71, 64] have also been suggested for selecting connect-sets.

Regardless of the method used, each connect-set is connected by a single bus. (A single "segmentable bus" [72, 75] can also used in this context.) We will refer to such architectures as *Single-Bus Enhanced Meshes* (*SBEMs*). With increase in the network size, and consequently the connect-set sizes, SBEMs require buses with high loading. Another problem of connecting a large number of processors to a single bus is an increase in the demand for the bus, resulting in communication bottlenecks. One advantage, however, is the ease of broadcasting (provided the loading is not excessive).

In this chapter, we propose *Multiple-Bus Enhanced Meshes* (*MBEMs*) that allow the use of multiple buses to connect processors in connect-sets. In particular, binary-tree MBNs are very well suited for this purpose as they are designed to facilitate the two most widely studied applications of enhanced meshes, namely, semigroup operations and broadcasting. An MBEM with binary-tree MBNs along rows/columns (or their subsets) can be viewed as being similar to the mesh of trees, a very versatile topology [49]. A mesh of trees has mesh processors arranged in a grid, and additional processors and links that connect each row and column as a complete binary tree. This network has the desirable features of both the tree and the mesh architecture such as a small diameter and a large bisection width. Variations of this idea, such as mesh with trees along diagonals, have also been proposed [49, page 295].

MBEMs have three important advantages over SBEMs. First, the loading of MBEMs can be limited (often to a constant), regardless of the network size. Second, the network for connecting elements of a connect-set can be tailored by the network designer to obtain various trade-offs between network cost and performance. Third, an MBEM is a generalization of the SBEM; therefore MBEMs can capture the features of SBEMs.

Almost all previous enhanced mesh architectures are SBEMs, focusing on identifying connect-sets rather than the method for connecting elements within them. Consequently, they have the disadvantage of high loading associated with SBEMs. For example Stout [76] and Bokhari [10] added a single global bus to the basic 2-dimensional mesh to facilitate broadcasting. Since all the processors are connected to this single bus, the loading is very high ($\Theta(N)$). However, Aggarwal [1] added k global buses to a d-dimensional array. This architecture has a degree k and also a high $\Theta(N)$ loading, as each processor is connected to all k global buses. Another model that has drawn considerable interest is the mesh with multiple broadcasting buses [7, 8, 30, 51, 76]. In this model, each row and the column of the mesh is connected to a single bus. Consequently, the loading is $\Omega(\sqrt{N})$. A partial solution to the high loading problem has been to connect only a selected subset of row/column processors to a bus. For example Chen *et al.* [17], Bar-Noy and Peleg [4], and Serrano and Parhami [75] connect every $N^{\frac{1}{8}}$ processor on each row and column to a bus. These methods still have a high loading ($\Omega(N^{\alpha})$, for $\alpha \geq \frac{1}{2}$). Raghavendra [71] has proposed the HMESH, a hierarchical architecture that reduces loading, but only at the cost of a large non-constant degree. Pan *et al.* [64] proposed the IMMB architecture, which uses a multi-level mesh hierarchy. Even though its degree is small, it has a high $O(\sqrt{N})$ loading. Serrano and Parhami [75] used buses with segment switches in each row/column of the mesh. Although this reduces the loading somewhat, the model still has a $\Theta(\sqrt{N})$ loading.

A large portion of results on enhanced meshes has been performing semigroup operation (reductions and semigroup operations). Stout [76] showed that finding maximum/minimum, median, and sorting can be done in $O(\sqrt{N})$, $O(\sqrt{N \log N})$, and $O(N)$ time with a single global bus. Bokhari [10] improved the time required for finding the maximum/minimum to $O(N^{\frac{1}{3}})$ steps. Aggarwal [1] added k global buses to a d-dimensional array, and showed that finding maximum/minimum requires $\Omega\left(\frac{N^{\frac{d}{2}}}{k}\right)^{\frac{1}{d}+1}$ time. Prasanna-Kumar and Raghavendra [69] derived an $O(N^{\frac{1}{6}})$ time algorithm for running semigroup operations on a $\sqrt{N} \times \sqrt{N}$ square mesh with broadcast buses. Chen *et al.* [17] and Bar-Noy and Peleg [4] have shown that the running time can be reduced to $O(N^{\frac{1}{8}})$, if a rectangular $N^{\frac{3}{8}} \times N^{\frac{5}{8}}$ mesh augmented with broadcast buses is used. Serrano and Parhami [75] added segment switches to the buses and achieved the same running time, while reducing the loading. Chung [19] reduced the time to $O(N^{\frac{1}{10}})$ while Pan *et al.* [64] achieved better time on an enhanced mesh of low aspect ratio.

In addition to semigroup operations, enhanced meshes have also been used to solve other classes of problems as well. For example, Bhagavathi *et al.* [7] have shown that many visibility problems, such as convex hull, can be computed in $O(\log N)$ time on a $\sqrt{N} \times \sqrt{N}$ mesh with multiple broadcast buses. In a different paper, Bhagavathi *et al.* [8] established that selecting the k^{th} smallest element in a rectangular $N^{\frac{3}{8}} \times N^{\frac{5}{8}}$ enhanced mesh can be done in $O(N^{\frac{1}{16}}(\log N)^{\frac{3}{4}})$ time. The batched searching and ranking problem (which is fundamental to many algorithms including database querying, pattern recognition, robotics and VLSI), where m values stored in a $\sqrt{N} \times \sqrt{N}$ mesh with multiple broadcast buses, has been be solved in $O(\log N + \sqrt{m})$ time [11].

Some architectures using multiple buses for connect-set have also been proposed. The GMCCMB [19] allows several buses to connect elements of a connect-set, it represents a particular situation that can be viewed as a further refinement of the connect-set itself, rather than employing a specialized MBN. Indeed, our results improve on those of the GMCCMB. The TBN [25], BBT [26] and BRT [27] are MBEMs, that also use binary-tree MBNs to enhance the mesh. However, these results do not provide a general treatment of the topic or relate cost and performance issues as is done in this work.

Given any N-processor, $\frac{N}{2}$-bus, binary-tree MBN, we first develop a framework for deriving other binary-tree MBNs that provide different cost/performance trade-offs. The primary parameters considered are time for reduction and broadcasting, loading, degree, and layout area. We also study binary-tree MBNs with "segment switches" on buses. (A segment switch allows a bus to be cut into several parts that can be used simultaneously as independent bus segments.)

We use the binary-tree MBN derivatives mentioned above to construct MBEMs and show how the network parameters can be adjusted to obtain various trade-offs. Tables 4.1 and 4.2 (pages 73 and 82) show some parameter choices, with interesting possibilities. All MBEMs in the table have optimal area and constant degree. Although our discussion on semigroup operations focuses on reduction, the results can also be extend to prefix computations (see Section 2.4, page 17).

In the next section we briefly discuss the parameters used to evaluate MBEMs. In Section 4.2 we use a given binary-tree MBN to derive other binary-tree MBNs. We put these results together in Section 4.3 to construct enhanced meshes with a wide range of cost/performance trade-offs. Section 4.4 deals with similar ideas for MBEMs with segment switches. Finally in Section 4.5 we summarize our results and make some concluding remarks.

4.1 Preliminaries

In this section we discuss some preliminary ideas and define some terms used in rest of the chapter.

4.1.1 MBN Measures

We will quantify the notion of cost and performance of an MBN (and an MBEM based on it) using five measures: reduction time, broadcast time, degree, loading, and layout area. These quantities are not entirely independent of each other, but collectively serve to measure the network's cost and performance. The number of buses has also been used in previous results to reflect the sparseness of the network. We ascribe less importance to this quantity in this work, as the layout area captures the notion of network sparseness. The degree and loading have been discussed in Section 2.2 (page 13). We now describe the remaining parameters.

Reduction time: This is the time required for the MBN to reduce inputs at its processors to a single result, using a binary-tree algorithm.

Broadcast time: This is the time required for the MBN to broadcast a piece of information from one processor to all other processors. A broadcast (from a fixed source) can be viewed as a traversal of a binary tree from root to leaves, so the broadcast time is upper bounded by the reduction time for the MBN. If the broadcast can originate from any processor, then the broadcast time is at most twice the reduction time (corresponding to a traversal to the root of a tree, and a broadcast down to the leaves).

Layout: An $X \times Y$ layout of an MBEM or MBN is a placement of its processors and buses in two layers within an $X \times Y$ rectangle. A "word-model" is assumed in which buses and connections between processors and buses are of unit width. Processors themselves are assumed to be of constant area; this is reasonable for constant degree MBNs, such as those considered in this chapter. The layout is assumed to be rectilinear; that is, all buses and connections consist of horizontal and/or vertical line segments (wires). The two-layer layout places horizontal and vertical wires on two different layers with "vias" connecting layers when needed. The *area* of an $X \times Y$ layout is XY and its aspect ratio is $\frac{\max(X,Y)}{\min(X,Y)}$. Clearly, a low area is desirable, while a low aspect ratio facilitates easier implementation.

As mentioned earlier, we will use MBNs to construct MBEMs. Since these MBNs will be used with connect-sets, each of which spans a single row or column of the mesh, we will only consider MBN layouts in which all processors are placed in a line along one side of the rectangle enclosing the layout. Such a layout is called a "perimeter" layout. Since a perimeter layout typically has one side of size $\Theta(P)$, where P is the number of processors, we will specify a high aspect ratio, perimeter layout by its *layout height* H; this represents a $P \times H$ layout (see Figure 6.5, page 119). The layout of the MBEMs, however, will be *dense* and place processors throughout the enclosing rectangle to obtain a constant aspect ratio. Here the layout must be specified by both dimensions of the enclosing rectangle; i.e., as an $X \times Y$ layout.

4.1.2 Multiple-Bus Enhanced Meshes

A *Multiple-Bus Enhanced Mesh* (*MBEM*) has a set of processors connected by an underlying mesh topology. The processors are grouped into (not necessarily disjoint) sets $C_0, C_1, \cdots, C_x$, called *connect-sets.* The selection of these connect-sets is an architectural choice, made with an eye on the application domain, cost and desired performance. A typical connect-set includes an entire row/column [1, 10, 51] or subsets of a row or column [4, 17, 19, 75]. Other more complex methods have also been proposed [33, 64, 71].

The processors of each connect-set are connected via one or more buses. Up to this point, there is no difference between MBEMs, and traditional Single-Bus Enhanced Meshes (SBEMs). The difference between them is in the manner in which processors of a connect-set are connected. In an SBEM all elements of a connect-set are connected to a single bus, dedicated to that connect-set. In an MBEM, however, any MBN could be used to connect elements of a connect-set. As a particular case, if MBNs, each with one bus are used, then the MBEM becomes an SBEM. Therefore, MBEMs can be viewed as generalizations of SBEMs. In general, a different type of MBN may be used for different connect-sets. They could even be different for the same connect-set in two different MBEMs. Thus the idea of decomposing the mesh into connect-sets is independent of the method used to connect each connect-set. It is the method of connecting them that distinguishes SBEMs (that use single buses) from MBEMs (that use multiple buses). We show in this chapter that there are significant advantages to connecting elements of connect-sets by multiple buses.

4.2 Binary-Tree MBN Extensions

In this section, we present a method to construct a $2^n \times 2^m$ MBN from any given $2^n \times 2^{n-1}$ binary-tree MBN (where $0 \leq m < n$). We apply this general result to the Tree MBN (Section 3.6, page 44) and then use the resulting $2^n \times 2^m$ binary-tree MBN to enhance 2-dimensional meshes (Sections 4.3 and 4.4).

Lemma 4.1 *Let $X(n)$ be a $2^n \times 2^{n-1}$ MBN with degree and loading of d_n and ℓ_n, that runs $Bin(n)$ in t_n steps and that performs broadcast in q_n steps. Let $X(n)$ have a layout height of h_n. Then for any $0 \leq m < n$, there exists a $2^n \times 2^m$ MBN, $X'(n,m)$, with degree and loading d_{m+1} and $\ell_{m+1} + 2^{n-m} - 2$, that runs $Bin(n)$ in $2^{n-m} + t_{m+1} - 2$*

steps. The MBN $X'(n,m)$ has an h_{m+1} layout height and a broadcast time of $q_{m+1}+2$ steps.

Proof: Consider running $Bin(m+1)$ on $X(m+1)$. Clearly the first step divides the 2^{m+1} processors into 2^m pairs, each of which uses a distinct bus. For any $0 \leq i < 2^{m+1}$, let processor pair $(i, \pi(i))$ use bus $b(i)$ in the above step. Construct the $2^n \times 2^m$ MBN, $X'(n,m)$, as follows. Divide its 2^n processors into 2^{m+1} sets, S_i (where $0 \leq i < 2^{m+1}$); each set consists of 2^{n-m-1} contiguous processors of $X'(n,m)$. Designate one of the processors, of each set as its "leader." Connect each non-leader processor of $S_i \cup S_{\pi(i)}$ to bus $b(i)$ and connect the 2^{m+1} leaders to the 2^m buses as in $X(m+1)$. Note that the leader of S_i is also connected to bus $b(i)$.

The MBN $X'(n,m)$ runs $Bin(n)$ by first combining the inputs in S_i and $S_{\pi(i)}$ (via bus $b(i)$) into the leaders of S_i and $S_{\pi(i)}$ in $2(2^{n-m-1}-1) = 2^{n-m}-2$ steps. This effectively reduces $Bin(n)$ into $Bin(m+1)$, which is next run as on $X(m+1)$ in t_{m+1} steps. The processors of $X(m+1)$ have at most d_{m+1} connections. All the other processors have only one connection each. Therefore, the degree of $X'(n,m)$ is d_{m+1}. Each bus of $X'(n,m)$ is connected to 2^{n-m} processors. Two of these 2^{n-m} processors are part of $X(m+1)$. Therefore, the loading of $X'(n,m)$ is $2^{n-m}-2+\ell_{m+1}$. Since the layout height of $X(n)$ is h_n, the layout height of $X(m+1)$ is h_{m+1}. Any processor in $X'(n,m)$ can be reached from another processor by traversing in MBN $X(m+1)$ and along two additional bus (within a group). Since the broadcast time of $X(n)$ is q_n, the broadcast time of $X'(n,m)$ is $q_{m+1}+2$ steps. ■

We now discuss the implication of Lemma 4.1. Observe that if $t_n = n$ (which is optimal), then the time for $X'(n,m)$ to run $Bin(n)$ is $2^{n-m}+m-1$, which has been shown to be optimal for any $2^n \times 2^m$ MBN [2]. If d_n is a constant, then so is d_{m+1}, the degree of $X'(n,m)$. If $\ell_n = 3$, then the loading of $X'(p,m)$ is $2^{n-m}+1$, the best

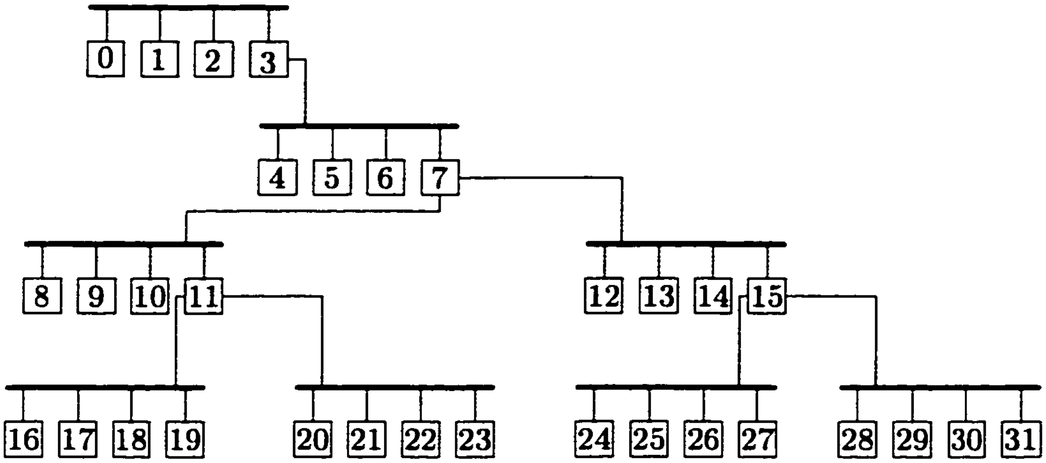


Figure 4.1: A 32 × 8 Tree MBN

possible for any connected $2^n \times 2^m$ MBN (see Lemma 3.1, page 21). Thus, $X'(p, m)$ inherits optimal features of $X(p)$. In particular, the $2^n \times 2^{n-1}$ Tree MBN, $\mathcal{T}(n)$ (see Section 3.6, page 44), runs $Bin(n)$ optimally and has degree and loading of 3 each. Lemma 4.1 allows $\mathcal{T}(n)$ to be extended to a $2^n \times 2^m$ MBN (for any $0 \leq m < n$) that runs $Bin(n)$ in optimal time, whose degree is 3 (same as that of $\mathcal{T}(n)$), and whose loading is $2^{n-m}+1$ (which is optimal). Figure 4.1 shows the structure of a 32×8 Tree MBN. This MBN has a loading of 5, and a degree of 3. Its time for running $Bin(5)$ is 6 steps.

The selection of an appropriate MBN to connect the processors in the connect-set is crucial as it determines all the important network parameters (running time, degree, loading, etc.). For this work we will use the Tree MBN. (Other MBNs such as the delayed root MBN (Section 3.7.2, page 49) or those of [27, 24] could also be used.) Though the Tree MBN is defined for 2^n inputs, it is easily modified to handle

N (not necessarily a power of 2) inputs. It can also be verified (see Figure 3.6(a)) that a $2^n \times 2^{n-1}$ Tree MBN has $\Theta(n)$ layout height. For brevity, we will express the performance measures (time, loading, degree etc.) of the Tree MBN in terms of orders, rather than exact values. We now summarize the properties of the above Tree MBN in the following theorem using the order notation.

Theorem 4.2 *For $1 \leq M \leq \frac{N}{2}$, there exists an $N \times M$ Tree MBN with constant degree and $O(\frac{N}{M})$ loading. It runs an N-input, binary-tree algorithm in $O\left(\frac{N}{M} + \log M\right)$ steps. This MBNs has a $O(\log M)$ layout height and $O(\log M)$ broadcast time.*

4.3 Meshes with Tree MBNs

In this section we construct an MBEM called the *Mesh with Tree MBNs* that uses the Tree MBN to connect processors of connect-sets. This structure has several advantages over other enhanced meshes proposed in the literature. We derive some results to highlight these advantages. Our description here uses the least number of parameters to describe the idea. This idea can be generalized to include different sized and dimensioned meshes, MBNs other than the Tree MBN, and different sizes/types of MBNs for different connect-sets.

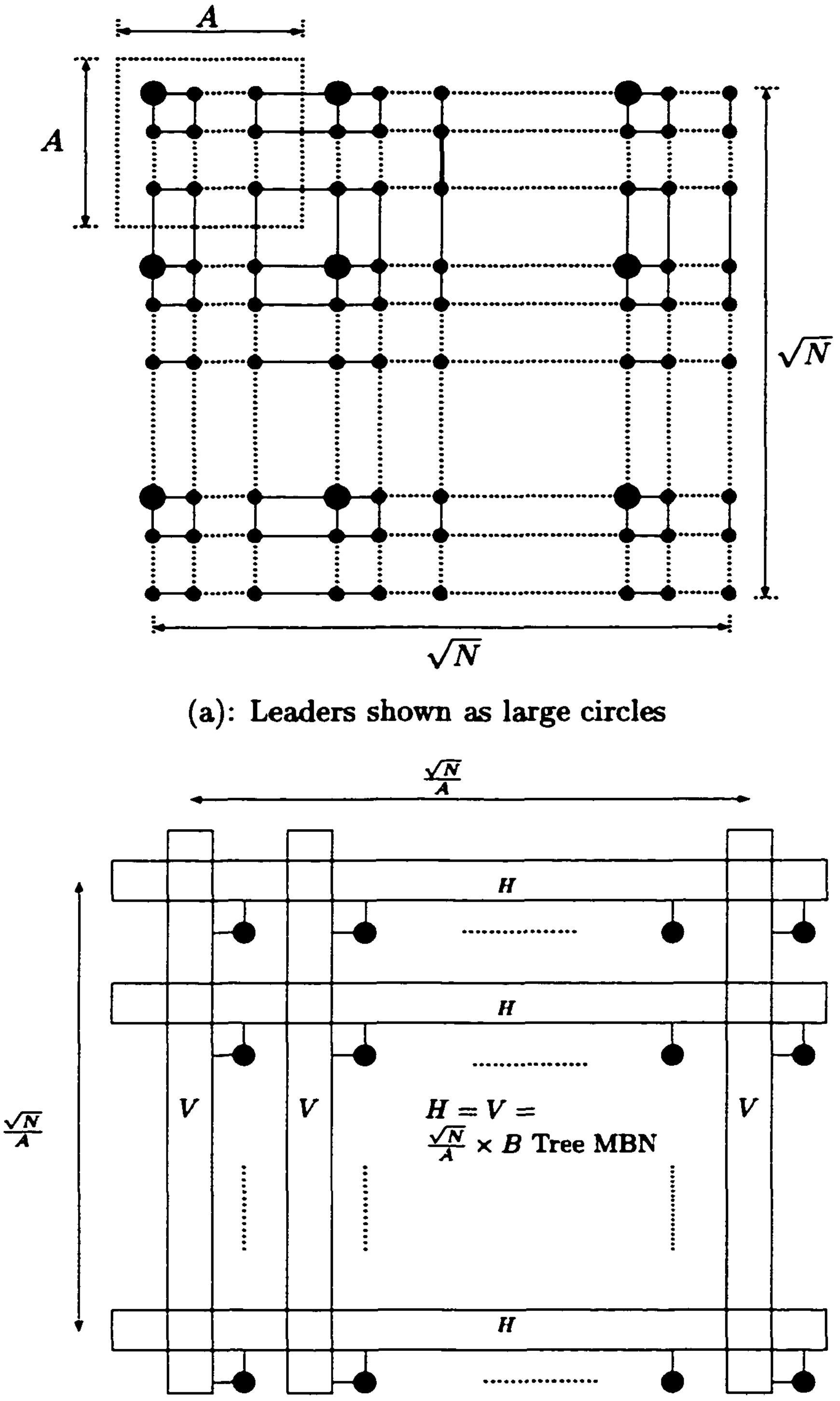
For integers $N, A, B \geq 1$ define the mesh with Tree MBNs, $\mathcal{MT}(N, A, B)$, as follows. Arrange N processors as a $\sqrt{N} \times \sqrt{N}$ mesh. Divide this mesh into $A \times A$ submeshes (Figure 4.2(a)), and designate one processor (the top, left processor $p_{i,j}$ say) of each submesh as the "leader." The leaders form a $\frac{\sqrt{N}}{A} \times \frac{\sqrt{N}}{A}$ array. For $0 \leq i < \frac{\sqrt{N}}{A}$, processors $\{p_{i,j} : 0 \leq j < \frac{\sqrt{N}}{A}\}$, form the *horizontal connect-sets* and processors $\{p_{j,i} : 0 \leq j < \frac{\sqrt{N}}{A}\}$ form the *vertical connect-sets.* In other words, rows and columns of leaders form connect-sets. Each connect-set is connected via B buses to form a $\frac{\sqrt{N}}{A} \times B$ instance of the Tree MBN (Figure 4.2(b)). If $B = 1$, then there will be a single bus in each Tree MBN connecting the horizontal and vertical connect-sets and the resulting SBEM structure will be the model of Chen *et. al* [17], if $A = N^{\frac{1}{8}}$ and that of Bhagavathi *et. al* [10], if $A = 1$. We now outline the three major phases involved in running an N-input binary-tree algorithm on this structure.

1. Local reduction: Reduce the $A \times A$ inputs in each submesh to a single partial result in its leader using the local mesh links. This requires $2(A-1)$ steps and the problem size reduces to $\frac{N}{A^2}$.

2. Horizontal reduction: Use all $\frac{\sqrt{N}}{A} \times B$ horizontal Tree MBNs (shown as H in Figure 4.2(b)), in parallel to reduce $\frac{\sqrt{N}}{A}$ partial results in leaders of each horizontal connect-set to a single partial result. The time and loading for this phase are $O(\frac{\sqrt{N}}{AB} + \log B)$ and $O(\frac{\sqrt{N}}{AB})$, respectively (Theorem 4.2). The problem size is now reduced to $\frac{\sqrt{N}}{A}$.

3. Vertical reduction: All partial results of the horizontal reduction step are in one connect-set. This phase is performed on the vertical $\frac{\sqrt{N}}{A} \times B$ Tree MBN of this connect-set. The time and loading for this phase are again $O(\frac{\sqrt{N}}{AB} + \log B)$ and $O(\frac{\sqrt{N}}{AB})$. The inputs are now reduced to one result.

The overall time for running an N-input binary-tree algorithm on $\mathcal{MT}(N, A, B)$ is the sum of time required for each of the three phases; that is, the reduction time is $O(A + \frac{\sqrt{N}}{AB} + \log B)$. The broadcast time is the time it takes for a piece of information to travel from one processor to any other processor. In this MBEM, it is equal to the broadcast time of two $\frac{\sqrt{N}}{A} \times B$ Tree MBNs and time required to reach any processor in an $A \times A$ submesh. Therefore, the broadcast time is $O(A + \log B)$. The loading is $O(\frac{\sqrt{N}}{AB})$ and the degree (including the local mesh connections) is $4 + 2 \times 3 = O(1)$. The total number of buses (not including local mesh connections) MBEM is $O(\frac{B\sqrt{N}}{A})$. The



(a): Leaders shown as large circles

(b): Binary tree MBNs connecting the leaders

Figure 4.2: Structure of a mesh with binary-tree MBN

VLSI area required for the $\sqrt{N} \times \sqrt{N}$ processor array alone is N. Therefore, the VLSI area of $\mathcal{MT}(N, A, B)$ is $\Omega(N)$. Each of the horizontal and vertical MBNs has $\frac{\sqrt{N}}{A}$ processors and, as shown in Figure 4.2(a), these processors are separated by a distance of $A + \log B$ units. Therefore, each of these MBNs has an $O\left((\frac{\sqrt{N}}{A}(A + \log B)) \times \log B\right)$ VLSI layout. Since the $\frac{\sqrt{N}}{A}$ vertical and horizontal MBNs have symmetric layouts, $\mathcal{MT}(N, A, B)$ has a constant aspect ratio $O\left((\sqrt{N}(1 + \frac{\log B}{A})\right) \times \left(\sqrt{N}(1 + \frac{\log B}{A})\right)$ layout. Therefore, its layout area is $O(N + \frac{N \log B}{A})$. In summary, $\mathcal{MF}(N, A, B)$ has the following:

$$\begin{array}{lcl}
\text{Degree} & = & 10 \\
\text{Loading} & = & \Theta(\frac{\sqrt{N}}{AB}) \\
\text{Broadcast time} & = & \Theta(A + \log B) \\
\text{Reduction time} & = & \Theta(A + \frac{\sqrt{N}}{AB} + \log B) \\
\text{Number of buses} & = & \Theta(\frac{B\sqrt{N}}{A}) \\
\text{Area} & = & \Theta(N + \frac{N \log B}{A}) \\
\text{Aspect ratio} & = & \Theta(1)
\end{array}$$

Since the reduction time is $\Omega(A)$ and $\Omega(\log N)$ (N inputs cannot be reduced in less than $\log N$ steps), we choose $A = \Omega(\log N)$. Since $B = O(N)$, $\log B = O(\log N)$. Therefore, $A = \Omega(\log B)$. Then $\mathcal{MT}(N, A, B)$ has $\Theta(N)$ area, which is optimal. With $L = \Theta(\frac{\sqrt{N}}{AB})$ denoting the loading, we have

$$\begin{array}{lcl}
\text{Degree} & = & 10 \\
\text{Loading} & = & L \\
\text{Broadcast time} & = & \Theta(A) \\
\text{Reduction time} & = & \Theta(A + L)
\end{array}$$

$$\begin{aligned} \text{Number of buses} &= \Theta(B^2L) \\ \text{Area} &= \Theta(N) \\ \text{Aspect Ratio} &= \Theta(1) \end{aligned}$$

Notice that it is possible to make trade-offs between loading and the number of buses, while keeping the time and VLSI area unchanged. Table 4.1 shows results for various choices of A and L. The last two entries of the table also show reduction results on other (SBEM) architectures in the literature. Notice that our method matches that of the GMCCB [19], while having a better aspect ratio and providing additional possibilities for the same time, or for the same loading. Compared to IMMB [64] our method has a better loading for $z < \frac{1}{2}$ and provides more possibilities for the loading (given a fixed time).

Table 4.1: Some results for meshes with Tree MBN

Architectures	Time	Loading	No of Buses	Aspect Ratio
$A = N^{\frac{1}{8}}$	$N^{\frac{1}{8}}$	constant	$N^{\frac{6}{8}}$	constant
$A = N^{\frac{1}{8}}$	$N^{\frac{1}{8}}$	$N^{\frac{1}{8}}$	$N^{\frac{5}{8}}$	constant
$A = N^{\frac{1}{10}}$	$N^{\frac{1}{10}}$	constant	$N^{\frac{8}{10}}$	constant
$A = N^{\frac{1}{10}}$	$N^{\frac{1}{10}}$	$N^{\frac{1}{10}}$	$N^{\frac{7}{10}}$	constant
$A = N^{\frac{1}{16}}$	$N^{\frac{1}{16}}$	constant	$N^{\frac{14}{16}}$	constant
$A = N^{\frac{1}{16}}$	$N^{\frac{1}{16}}$	$N^{\frac{1}{16}}$	$N^{\frac{13}{16}}$	constant
for $z > 0$, $A = \log^z N$	$\log^z N$	L	$\frac{N}{L \log^{2z} N}$	constant
for $z > 0$, $A = N^z$	N^z	3	N^{1-2z}	constant
for $z > 0$, $A = N^z$	N^z	N^z	N^{1-3z}	constant
GMCCMB [19]	$N^{\frac{1}{10}}$	$N^{\frac{1}{10}}$	$N^{\frac{7}{10}}$	$N^{\frac{1}{5}}$
IMMB [64], for $z > 0$	N^z	$N^{\frac{1}{2}}$	N^{1-3z}	constant

In all of the cases VLSI area is $\Theta(N)$ and the degree is constant.
All the entries show orders rather than absolute values.

4.4 MBEMs with Segment Switches

A *segment switch* placed on a bus can break (when opened) the bus into two independent buses. When the switch is closed, the two segments are fused together and they function as a single bus. It has been well studied in the context of dynamically reconfigurable architectures [59, 77] and other bus-based networks [72, 75]. Since segment switches allow a bus to be configured to suit a particular step of an algorithm, they could be used to reduce the loading as shown in this section.

4.4.1 Binary-Tree MBNs with Segment Switches

In this section we derive results similar to Theorem 4.2 for binary-tree MBNs with segment switches. We then use these derivative binary-tree MBNs to enhance the mesh.

Addition of segment switches requires further development of some of the ideas used. These ideas are in the setting of a given computation (such as reduction or broadcasting). We assume that a segment switch changes state at least once during the course of the computation. A *bus* may be defined as a maximally connected segment resulting from all segment switches being in the closed state.

The state of segment switches in an MBN may change during the execution of an algorithm. It is therefore possible to define two types of loadings for MBNs with segment switches, *absolute* and *relative*. The *absolute loading* is the largest possible loading of a bus (when all segment switches on it are closed); this matches the conventional idea of loading. The *relative loading of a bus segment* at some step s of a computation is the number of connections and (closed) segment switches on that segment during step s. The *relative loading of a step* s is the largest of the relative loadings of all the bus segments of step s. The *relative loading of a computation* is the largest of the relative loadings of all steps of the computation. Notice that the effect of bus loading (whether optical or electrical) is restricted to within a segment. Therefore, relative loading is indicative of the demands on the system for that computation. On the other hand, absolute loading is useful for the worst case scenario in a general purpose environment. Naturally, the absolute loading is no smaller than the relative loading of any computation.

We now describe two methods for running binary-tree algorithms on MBNs with segment switches.

Consider a bus with K segment switches numbered $0, 1, \cdots, K-1$. When all the switches are open, the bus is broken into K independent segments[1]. Let these *atomic segments* be $S_0, S_1, \cdots S_{K-1}$. In the context of a binary-tree algorithm, let there be one result processor holding a partial result per segment. (There may be many non-result processors connected to each bus segment.) The aim is to reduce the K partial results to one final result.

Method 1: Without loss of generality, let $K = 2^k$ for integer k. This method performs the reduction in $\log K$ steps (optimal time) as follows (see also Figure 4.3). First, close the segment switches $0, 2, 4, \cdots, K-2$. This will fuse segments $\{S_0, S_1\}$, $\{S_2, S_3\}$, $\cdots$, $\{S_{K-2}, S_{K-1}\}$. Reduce the partial results of the two fused segments to one result. Next, close the segment switches $1, 5, 9, \cdots, K-3$ and reduce the two partial results of the fused segments to one result. At this step, four original bus segments $\{S_0, S_1, S_2, S_3\}$, $\{S_4, S_5, S_6, S_7\}$, $\cdots$,$\{S_{K-4}, S_{K-3}, S_{K-2}, S_{K-1}\}$ are fused together. This process is carried out doubling the number of segments fused, until all the segments are fused together and the last two partial results are reduced to one

[1] Actually ther are K+1 segments, but we will keep the discussion simple by leaving one unutilized.

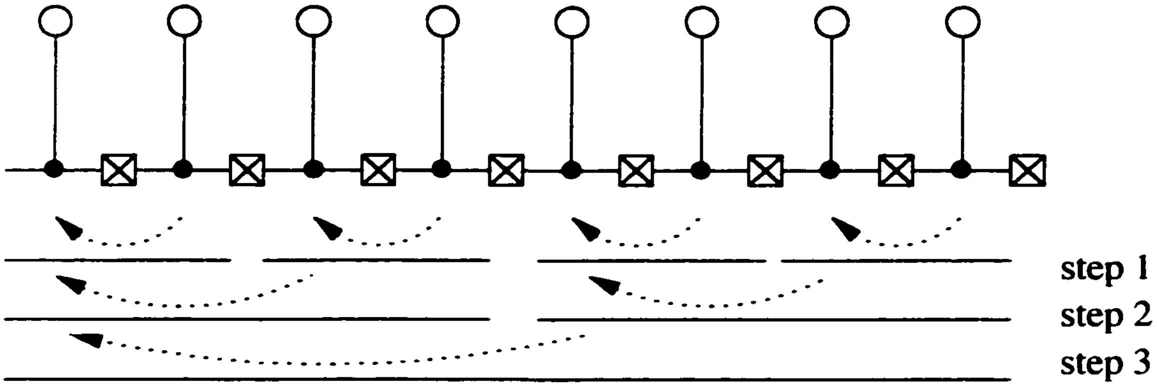


Figure 4.3: Steps of method 1

final result. The time for this reduction is clearly $\log K$ steps, and loading is equal to the number of processors connected to the entire bus (or at least half this number). Since all the processors are connected via the bus when all the segment switches are closed, the broadcast time of this method is only one step.

Method 2: This method performs the reduction in K steps as illustrated in Figure 4.4. First close segment switch $K-2$, and reduce the inputs in segments S_{K-2} and S_{K-3} to a result in segment S_{K-3}. Next open segment switch $K-2$ and close the segment switch $K-3$. This will fuse segments S_{K-3} and S_{K-4}. Reduce the partial results in segments S_{K-3} and S_{K-4} into a result in segment S_{K-3}. This reduction process is carried out, until the last two segments S_1 and S_0 are fused together and their results/inputs are reduced to one result in segment S_0. The time for this reduction is clearly $K-1$ steps. Since at most two segments are fused at any given time, the loading of this method is equal to twice the number of processors in each segment. For this loading, the broadcast time is the same as the reduction time.

We now present two methods to construct an S-segment switch, $N \times M$ binary-tree MBN from any given $N \times \frac{N}{2}$ binary-tree MBN. We propose two constructions, each of which uses the two methods discussed above.

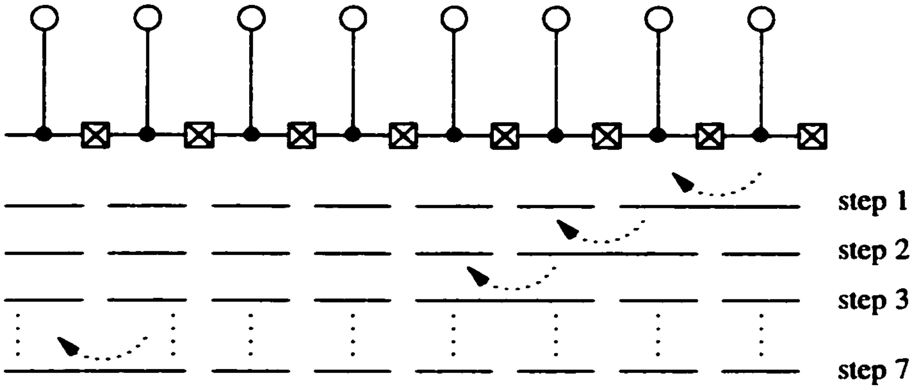


Figure 4.4: Steps of the method 2

Construction 1: Let $X(N)$ be an $N \times \frac{N}{2}$ MBN. The aim is to use $X(N)$ to construct an $N \times M$ MBN $X_1(N, M, S, K)$ with S segment switches and a parameter K that controls its running time and relative loading. Let these parameters satisfy $1 \leq \frac{MK}{2} \leq S \leq N$. For brevity and without loss of generality, assume that quantities such as $\frac{N}{M}$, $\frac{S}{MK}$ and $\frac{N}{S}$ are integers.

The idea is to first use $\frac{M}{2}$ buses with segment switches to reduce the N inputs to M partial results. Next use the remaining $\frac{M}{2}$ buses (without segment switches) as an $M \times \frac{M}{2}$ instance $X(M)$ the given MBN. We now describe the reduction of N inputs to M partial results. Each of $\frac{M}{2}$ buses used for this part is used to obtain 2 partial results. Since all buses proceed identically, we describe the activity of only one.

Of the N processors and S segment switches, $\frac{2N}{M}$ processors and $\frac{2S}{M}$ switches are assigned to each bus. These processors and switches are arranged on the bus as shown in Figure 4.5. The processors are divided into $\frac{N}{S}$ *segments*, with a segment switch between adjacent segments. There are $\frac{2N}{M}/\frac{N}{S} = \frac{2S}{M}$ segments, one per switch. Cluster K contiguous segments into a *group*. Each group spans K segments and there are $\frac{2S}{MK}$ groups. The reduction proceeds as follows.

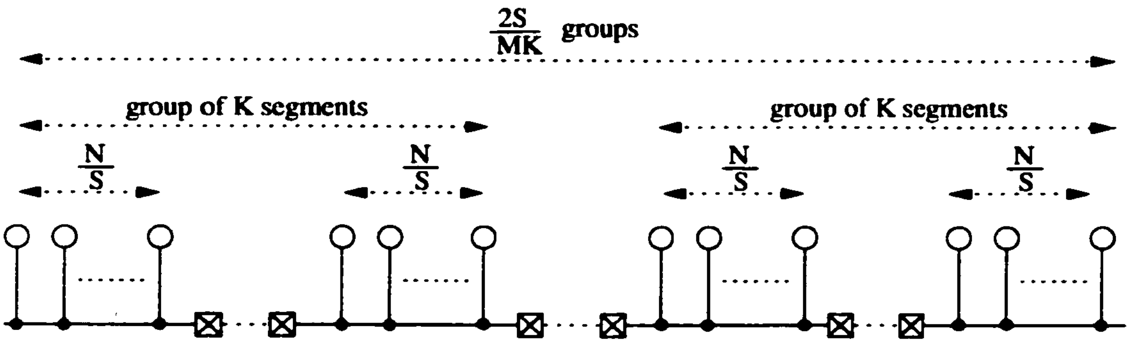


Figure 4.5: Construction 1

1. Open all the segment switches, and sequentially reduce the $\frac{N}{S}$ inputs in each segment. This takes $\frac{N}{S} - 1$ steps, and the relative loading of each bus segment is $\frac{N}{S}$.

2. Use Method 1 to reduce each group of K segments in $O(\log K)$ time with a relative loading of $\frac{NK}{S}$.

3. Use Method 2 to reduce group partial results to the two partial results for the bus. This takes $\frac{S}{MK}$ steps with relative loading $\frac{NK}{S}$.

4. Reduce the M results on an $M \times \frac{M}{2}$ instance $X(M)$ of the given MBN.

If $X(N)$ runs in t_N steps with relative loading ℓ_N, this step runs in t_M steps with ℓ_M relative loading. If the given MBN $X(N)$ has degree d_N, broadcast time q_N and layout height h_N, then the total time required to reduce N inputs on $X_1(N, M, S, K)$ is $O(\frac{N}{S} + \log K + \frac{S}{MK} + t_M)$. The relative loading and degree are $O(\frac{NK}{S} + \ell_M)$ and d_M respectively. The layout height and the absolute loading of $X_1(N, M, S, K)$ are $O(h_M)$ and $O(\frac{N}{M} + \ell_M)$. It is easy to see that the broadcast time of $X_1(N, M, S, K)$ is determined by step 3 and the broadcast time on $X(M)$. Therefore, the broadcast time of $X_1(N, M, S, K)$ is $O(q_M + \frac{S}{MK})$ steps.

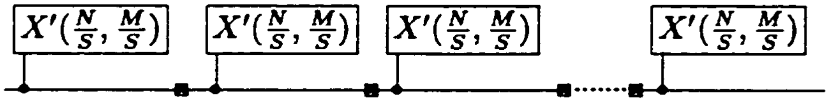

$X'(\frac{N}{S}, \frac{M}{S})$
$X'(\frac{N}{S}, \frac{M}{S})$
$X'(\frac{N}{S}, \frac{M}{S})$
$X'(\frac{N}{S}, \frac{M}{S})$

Figure 4.6: Construction 2

Lemma 4.3 *Let $X(N)$ be an $N \times \frac{N}{2}$ MBN with degree and loading d_N and ℓ_N. Let $X(N)$ run an N-input binary-tree algorithm in t_N steps and have a layout height of h_N. Let the broadcast time of this MBN be q_N. For $1 \leq \frac{MK}{2} \leq S \leq N$, there exists an S-segment switch, $N \times M$ MBN $X_1(N, M, S, K)$ with degree d_M, $O(\frac{NK}{S} + \ell_M)$ relative loading, and $O(\frac{N}{M} + \ell_M)$ absolute loading that runs an N-input binary-tree algorithm in $O(\frac{N}{S} + \frac{S}{MK} + \log K + t_M)$ steps. The VLSI height of $X_1(N, M, S, K)$ is $O(h_M)$ and the broadcast time is $O(q_M + \frac{S}{MK})$ steps.* ∎

If the given MBN $X(N)$ is the Tree MBN, then we have the following result.

Theorem 4.4 *For $1 \leq \frac{MK}{2} \leq S \leq N$, there exists an MBN with constant degree, $\Theta(\frac{NK}{S})$ relative loading and $\Theta(\frac{N}{M})$ absolute loading that runs an N-input binary-tree algorithm in $\Theta(\frac{N}{S} + \frac{S}{MK} + \log K + \log M)$ steps. This MBN has a layout height of $\Theta(\log M)$ and a broadcast time of $\Theta(\log M + \frac{S}{MK})$ steps.*

Construction 2: Again let $X(N)$ be the given MBN. This construction uses a small number of segment switches $S < M$. Use Lemma 4.1 to construct $X'(\frac{N}{S}, \frac{N}{S})$, an $\frac{N}{S} \times \frac{N}{S}$ derivative of $X(N)$. Use S such $X'(\frac{N}{S}, \frac{N}{S})$s to reduce the N inputs to S partial results. Now the problem is that of reducing S partial results on a bus (see Figure 4.4). With a parameter K (where $1 \leq K \leq S$), this can be done with Method 1 first and then Method 2 in $\Theta(\log K + \frac{S}{K})$ steps with a relative loading of $\Theta(K)$.

Construct $X_2(N, M, S, K)$ as follows. Distribute all S segment switches along one bus. Divide the N processors and M buses into S equal parts. Construct S copies of MBN $X'(\frac{N}{S}, \frac{M}{S})$ and connect one MBN each to the S bus segments.

With the same notation as in Construction 1, we have the following result.

Lemma 4.5 *Let $X(N)$ be an $N \times \frac{N}{2}$ MBN with degree and loading d_N and ℓ_N. Let $X(N)$ run an N-input binary-tree algorithm in t_N steps and have a layout height of h_N. Let the broadcast time of this MBN be q_N. For $1 \leq K \leq S \leq M \leq \frac{N}{2}$, there exists an S-segment switch, $N \times (M+1)$ MBN $X_2(N, M, S, K)$ with $d_{\frac{2M}{S}}$ degree, $\Theta(2K + \ell_{\frac{2M}{S}})$ relative loading and $\Theta(S + \frac{N}{M} + \ell_{\frac{2M}{S}})$ absolute loading that runs an N-input binary-tree algorithm in $\Theta(\frac{N}{M} + \frac{S}{K} + \log K + t_{\frac{2M}{S}})$ steps. The layout height of $X_2(N, M, S, K)$ is $Oh(\frac{2M}{S})$ and the broadcast time is $O(q_{\frac{2M}{S}} + \frac{S}{K})$.* ■

Remark: Note that $X''(N, M, S, K)$ is an $M \times (M+1)$ MBN.

If the given MBN $X(n)$ is the Tree MBN, then we have the following result.

Lemma 4.6 *For $1 \leq K \leq S \leq M \leq \frac{N}{2}$, there exists an MBN with constant degree, $\Theta(K + \frac{N}{M})$ relative loading and $\Theta(S + \frac{N}{M})$ absolute loading that runs an N-input binary-tree algorithm in $O(\frac{N}{M} + \frac{S}{K} + \log K + \log(\frac{M}{S}))$ steps. The layout height of the MBN is $O(\log(\frac{M}{S}))$ and the broadcast time is $O\left(\log(\frac{M}{S}) + \frac{S}{K}\right)$ steps.* ■

4.4.2 Meshes with Tree MBNs and Segment Switches

The idea of Section 4.3 readily extends to binary-tree MBNs with segment switches. Here the MBN connecting row and column connect-sets is a $\frac{\sqrt{N}}{A} \times B$, S segment-switch extension of the Tree MBN, obtained from Construction 1 (Theorem 4.4). The steps involved in running a binary-tree algorithm with segment switches are the same as in Section 4.3. Therefore, we simply state the results.

$$\begin{array}{lcl}
\text{Time} & = & \Theta(A + \frac{\sqrt{N}}{AS} + \frac{S}{BK} + \log K + \log B) \\
\text{Relative loading} & = & \Theta(\frac{\sqrt{N}K}{AS}) \\
\text{Absolute loading} & = & \Theta(\frac{\sqrt{N}}{AB}) \\
\text{Number of buses} & = & \Theta(\frac{\sqrt{N}B}{A}) \\
\text{Number of segment switches} & = & \Theta(\frac{\sqrt{N}S}{A}) \\
\text{Broadcast time} & = & \Theta(\frac{S}{BK} + \log B) \\
\text{VLSI area} & = & \Theta(N + \frac{N \log B}{A})
\end{array}$$

Let $A' = \frac{\sqrt{N}}{AS}$ and $A'' = \frac{S}{BK}$. If we let $1 \leq \frac{BK}{2} \leq S \leq \frac{\sqrt{N}}{A} \leq AS$, then the conditions for Lemma 4.5 are satisfied by row and column MBNs. Also $1 \leq \frac{S}{A''} \leq S \leq A'S \leq AS$ which implies that $A' \leq A$. If we also set $S \leq ABK$ then $A'' \leq A$, so we have the following:

$$\begin{array}{lcl}
\text{Reduction time} & = & (A) \\
\text{Relative loading} & = & \Theta(A'K) \\
\text{Absolute loading} & = & \Theta(A'A''K) \\
\text{Number of segment switches} & = & \Theta(\frac{N}{A^2A'}) \\
\text{Number of buses} & = & \Theta(\frac{N}{A^2A'A''K}) \\
\text{Broadcast time} & = & \Theta(A'' + \log B) \\
\text{VLSI area} & = & \Theta(N)
\end{array}$$

Using the above set of equations, we can compute various network parameters for different values of running time, segment switches and K. We show some results in Table 4.2. This table clearly shows the effect of segment switches on loading. Compared to Table 4.1, for the same running time, a lower relative loading can be

Table 4.2: Some results for meshes with Tree MBN and Segment Switches

Case	Time	Relative Loading	Absolute Loading	Aspect Ratio	Number of Buses	Number of Switches
$A = A' = N^{\frac{1}{8}}$ $A'' = K = N^{\frac{1}{8}}$	$N^{\frac{1}{8}}$	$N^{\frac{2}{8}}$	$N^{\frac{3}{8}}$	$O(1)$	$N^{\frac{3}{8}}$	$N^{\frac{5}{8}}$
$A = A' = N^{\frac{1}{8}}$ $A'' = N^{\frac{1}{8}}, K = N^{\frac{3}{8}}$	$N^{\frac{1}{8}}$	$N^{\frac{4}{8}}$	$N^{\frac{5}{8}}$	$O(1)$	$N^{\frac{1}{8}}$	$N^{\frac{5}{8}}$
$A = N^{\frac{1}{16}}, A' = 1$ $A'' = K = 1$	$N^{\frac{1}{16}}$	$O(1)$	$O(1)$	$O(1)$	$N^{\frac{14}{16}}$	$N^{\frac{14}{16}}$
$A = A' = N^{\frac{1}{16}}$ $A'' = K = N^{\frac{1}{16}}$	$N^{\frac{1}{16}}$	$N^{\frac{2}{16}}$	$N^{\frac{3}{16}}$	$O(1)$	$N^{\frac{11}{16}}$	$N^{\frac{13}{16}}$
$A = N^z, A' = 1$ $A'' = K = 1$	N^z	$O(1)$	$O(1)$	$O(1)$	N^{1-2z}	N^{1-2z}
$A = \log^z N, A' = L$ $A'' = K = 1$	$\log^z N$	L	$L \log^z N$	$O(1)$	$\frac{N}{L \log^{2z} N}$	$\frac{N}{L \log^{2z} N}$
$A = A' = N^z$ $A'' = N^z, K = 1$	N^z	N^z	N^{2z}	$O(1)$	N^{1-4z}	N^{1-3z}
$A = A'' = N^z$ $A' = K = 1$	N^z	$O(1)$	N^z	$O(1)$	N^{1-3z}	N^{1-2z}
Segmented Bus Enhanced Mesh [75]	$N^{\frac{1}{8}}$	$N^{\frac{4}{8}}$	$N^{\frac{4}{8}}$	$N^{\frac{2}{8}}$	$N^{\frac{4}{8}}$	$N^{\frac{5}{8}}$

achieved. For example with time $O(n^z)$ and N^{1-3z} buses, the loading is constant. For the same case in Table 4.1 this loading is N^z. Moreover, the results here are better then that has been achieved by Serrano and Parhami [75]. For a running time of $O(N^{\frac{1}{8}})$, we have a much wider choice of parameters, and the resulting MBN is superior in all respects to that in [75]. Specifically, we can achieve a much lower loading, while improving on aspect ratio, absolute loading and number of buses. Also compared to the IMMB [64] we now have for $O(N^z)$ time a constant relative loading with the same number of buses.

4.5 Results and Discussion

Traditionally, buses have been used in a variety of ways to enhance the basic mesh. In all these schemes, a single bus is used to connect processors. In this chapter we have explored the use of multiple buses to connect processors together to form meshes enhanced with MBNs. The traditional single bus approach is a special case of our framework.

The IMMB can also achieve $O(\log N)$ time, but to reduce its loading to constant, a high-dimensional, sub-optimal area structure [86] is needed. The methods presented in this chapter can be applied with any MBN. Therefore, these methods can be adopted for use with other algorithms by suitably selecting MBNs in horizontal and vertical dimensions. If each processor (or some of the processors) is connected to several buses through segment switches, then it is possible to "dynamically reconfigure" the MBNs. In that case, the same MBEM can be used to run different algorithms or different steps of the same algorithm optimally by reconfiguring the processor bus interconnection pattern.

Chapter 5

Fault Tolerance

In Chapter 4, we showed that binary-tree MBNs can be used for general purpose enhanced meshes with properties similar to the mesh of trees [49]. This chapter deals with the construction of fault-tolerant binary-tree MBNs. Such MBNs can be used as fault-tolerant building blocks for enhanced meshes. We present two methods for constructing such MBNs. One of the methods is more general in that it can also be used for any MBN (not just binary-tree MBNs). It is particularly useful in situations where the MBN uses resources (buses and processors) non-uniformly. In other words, if a given algorithm uses some of the resources most of the time, and the rest not that often, then this method can exploit this situation to produce better results; binary-tree algorithms represent one such situation. The second method applies only to binary-tree algorithms.

Specifically, we present two methods called *replication* and *recursive scheduling* that add connections in a systematic and controlled manner to transform any given binary-tree MBN into a fault-tolerant one. Given any $N \times M$ binary-tree MBN, $\mathcal{M}$, and an integer $1 \leq k \leq \frac{M}{2}$, we derive a $N \times M$ MBN $\mathcal{M}'$ that can tolerate the failure of any set of k buses. The performance of the fault-tolerant MBN, $\mathcal{M}'$, is measured in terms of (i) the time to run a set of computations designed for $\mathcal{M}$, (ii) its degree, and (*iii*) its loading. These attributes of $\mathcal{M}'$ depend on the corresponding attributes of $\mathcal{M}$. Replication can also be used for handling processor faults. The methods we propose in this chapter accept any MBN as input, so the approaches for bus and processor faults are independent; that is, tolerance to processor faults can be imparted to an MBN that is already tolerant to bus faults and vice versa.

Most previous work [3, 12, 15, 31, 45, 67, 68] on fault-tolerance in MBNs has focused primarily on issues related to connectivity/topology (number of failures to disconnect network, average distance between processors, etc.) and performance in a general purpose setting (such as throughput under various traffic models). Nadella and Vaidyanathan [57, 84] have considered the design of a specific fault-tolerant binary-tree MBN. The methods we present here are a generalization of that work in that it can be applied to any binary-tree MBN.

In Section 5.1 we state the assumptions used in the chapter. In Sections 5.2 and 5.3, we detail replication and recursive scheduling. The extension of replication to processor faults is discussed in Section 5.2.5. In Section 5.2.6 we tailor the replication results specifically to binary-tree MBNs. We compare the two methods in Section 5.4 and make some concluding remarks in Section 5.5.

5.1 Fault Model

We assume here that a faulty bus or processor is entirely faulty and completely unusable. We also assume that the faulty or fault-free status of each processor and bus is known before the MBN begins its computation and does not change during the computation. If a bus b (or processor p) is faulty, then a fault-free bus b' (or processor p') will be assigned to perform the functions of bus b (or processor p). We assume that bus b' (or processor p') has all the information necessary to perform these functions. The above assignment of responsibilities to fault-free elements is done before the computation commences.

The focus of this work is on designing an MBN that has the required redundant connections available, while the actual fault handling procedure (including fault identification and processing) is not considered. This may also be useful in improving the yield of a chip with binary-tree MBNs in which faults during manufacture can be bypassed by a one-time reconfiguration [44, 87].

5.2 Replication

In this section we develop the results for bus faults first, then extend them to include processor faults in Section 5.2.5. Replication is a general method that can be applied to any MBN. A key feature of replication is that it permits a set of k buses to be designated as "less important," and the failure of an arbitrary set of k buses can be treated as the failure of these less-important buses. In cases where not all resources are used equally, replication constructs a fault-tolerant MBN that is better tuned to the given computational setting. Binary-tree algorithms is a good example of uneven resource use, where the number of processors and buses required decreases by a factor of 2 with each level. We establish that for any $2^n \times 2^{n-1}$ binary-tree MBN, replication gives a fault-tolerant MBN that requires at most 5 (resp., 2) extra steps if as many as 2^{n-2} buses (resp., 2^{n-1} processors) fail. Such a result would not be possible without considering the fact that some buses/processors are used for only a few steps. Even with this consideration, one cannot guarantee that the faulty elements would be the ones used lightly. Replication provides the effect of this guarantee by allowing the failure of any set of buses/processors to be treated as the failure of a fixed set of less important elements. This flexibility lends itself to designing a fault-tolerant MBN that is better tuned to perform a given set of computations. Since we assume that faults occur before the start of the computation, resilience to processor faults can be obtained as a dual of the bus faults case. Therefore, replication can be used for processor faults as well.

5.2.1 Adding Redundant Connections

Given any $N \times M$ MBN, $\mathcal{M}$, and an integer $1 \leq k \leq \frac{M}{2}$, replication constructs a $N \times M$ MBN, $\mathcal{R}_k$, that can emulate $\mathcal{M}$, even if any set of (at most) k buses fail. The idea is to select k replacements for each bus b and copy the connections of b to each of the k replacement buses. We first define $\mathcal{R}_k$ and then establish that $\mathcal{R}_k$ can treat any set of faulty buses as a designated set of less important buses. This is followed by the derivation of the fault-tolerance properties of $\mathcal{R}_k$. Finally, we discuss tolerance to processor faults, as the dual of the bus fault case.

5.2.2 Definition of $\mathcal{R}_k$

Let the buses of the given $N \times M$ MBN, $\mathcal{M}$, (and the generated $N \times M$ fault-tolerant MBN, $\mathcal{R}_k$) be $0, 1, \cdots, M-1$. For any bus b of $\mathcal{M}$, let $Proc[b : \mathcal{M}]$ denote the set of processors of $\mathcal{M}$ that are connected to bus b. For any $0 \leq b < M$, let $R(b) = \{(b-i) \bmod M \ : \ 0 \leq i \leq k\}$ be the *replacement set* for bus b. Now define the fault-tolerant MBN, $\mathcal{R}_k$, as follows: For any $0 \leq b < M$, $Proc[b : \mathcal{R}_k] = \bigcup_{b' \in R(b)} Proc[b' : \mathcal{M}]$.

Each bus b of $\mathcal{R}_k$ has all the connections of bus b of $\mathcal{M}$, and the additional connections needed to replace buses $(b-i)(\text{mod } M)$ of $\mathcal{M}$, where $1 \leq i \leq k$.

In the following example, we have arranged (permuted) the buses so that the connections of a bus replicated on the k other buses overlap with existing connections. This can reduce the degree from $(k+1)d$ to kd. Permuting the buses in this manner to

	0	1	2	3	4	5	6	7
0	1	○	○	○				
1	1	●	○	○	○			
2		1	○	○	○			
3		1	●	○	○	○		
4				1	○			
5			1	●	○	○	○	
6			1	○	○	○		
7			1	○	●	○	○	○
8	○	○	○					1
9	○	○	○			1	○	●
10	○					1	○	○
11	○				1	●	○	○
12	○	○					1	○
13	○	○			1	○	●	○
14					1	○	○	○
15	○	○	○		1	○	○	●

Figure 5.1: The MBN of Figure 2.2 augmented to handle 3 bus faults

reduce degree may not be possible in all situations. With $k = 3$, Figure 5.1 shows the MBN $\mathcal{R}_3$ corresponding to the 16×8 MBN, $\mathcal{M}$, of Figure 2.2 (page 14). A connection between a processor and a bus of $\mathcal{M}$ is indicated by a "1," and a connection added for fault-tolerance is indicated by a "○." Entries where an existing connection ("1") and an added connection ("○") overlap are indicated by "●." In this example, nearly half the buses are permitted to be faulty. Therefore, a dense MBN, $\mathcal{M}_3$, is to be expected. The observations below show that this is not the case in general.

The following observations about $\mathcal{R}_k$ are straightforward.

1. If none of the buses are faulty, then $\mathcal{R}_k$ can emulate $\mathcal{M}$ without any overhead. This is because for each bus b, $Proc[b : \mathcal{M}] \subseteq Proc[b : \mathcal{R}_k]$. In other words, the set of connections of $\mathcal{M}$ is a subset of those of $\mathcal{R}_k$.

2. If the degree of $\mathcal{M}$ is d, then the degree of $\mathcal{R}_k$ is at most $(k+1)d$. This is because each connection to a bus b of $\mathcal{M}$ is copied over to k other buses of $\mathcal{R}_k$. Thus a processor connected to $d' \leq d$ buses of $\mathcal{M}$ is connected to at most $(k+1)d' \leq (k+1)d$ buses of $\mathcal{R}_k$.

   By arranging buses to maximize connection overlaps (shown as "•" in Figure 5.1), it may be possible to reduce the degree to kd. Notice that the degree of the MBN of Figure 5.1 is 7, rather than $(3+1)2 = 8$.

3. If the loading of $\mathcal{M}$ is ℓ, then the loading of $\mathcal{R}_k$ is at most $(k+1)\ell$. This is because each bus b of $\mathcal{R}_k$ is connected[1] to

   $$\left|Proc[b : \mathcal{R}_k]\right| \quad \leq \quad \sum_{b' \in R(b)} \left|Proc[b' : \mathcal{M}]\right| \quad \leq \quad (k+1)\ell \text{ processors.}$$

   Since not all buses have ℓ connections to start with, the loading of $\mathcal{R}_k$ is usually much smaller than $(k+1)\ell$.

5.2.3 The Designated Set

Often some buses of an MBN, $\mathcal{M}$, are more critical than others. This may be due to connectivity and/or usage in a particular set of computations. Failure of these "critical buses" impacts the network performance more severely. By the same measure, failure of "non-critical buses" does not degrade the performance to the same extent. In this section we first prove that there is no loss of generality in assuming that a fixed set of k buses is faulty (regardless of which k buses of $\mathcal{R}_k$ are actually faulty). That is, $\mathcal{R}_k$ can treat the failure of an arbitrary set of k buses as the failure of a fixed set of k designated non-critical buses. Next we show how this fixed faulty set can be emulated by the fault-free buses of $\mathcal{R}_k$. This has the benefit of allowing the network

[1] We denote the cardinality of a set S by $|S|$.

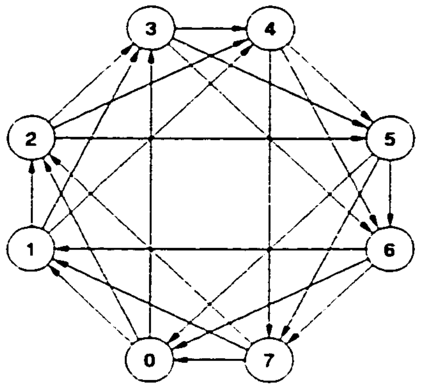


Figure 5.2: Graph $\mathcal{G}_{3,8}$.

designer to designate a suitable set of less important buses that can be treated as faulty.

For any $1 \leq k < M$, define a directed graph $\mathcal{G}_{k,M}$ with nodes $\{0, 1, 2, \cdots, M-1\}$ as follows. There is a directed edge $\langle b, b' \rangle$ from node b to node b' iff $b' = (b+i) \bmod M$, for some $1 \leq i \leq k$. Figure 5.2 shows $\mathcal{G}_{3,8}$. In the context of the $N \times M$ MBN, $\mathcal{R}_k$, each node of $\mathcal{G}_{k,M}$ represents a bus of $\mathcal{R}_k$. Node b has a directed edge to each node that can replace it; that is, $\langle b, b' \rangle$ is an edge iff $b \in R(b')$.

Let $\mathcal{G} = (V, E)$ be any directed graph and let $U, W \subseteq V$ with $|U| \subseteq |W|$. An injective[2] function $\rho : U \longrightarrow W$ is called a *node disjoint correspondence* iff

1. For each $u \in U$, there is a directed path in $\mathcal{G}$ from u to $\rho(u)$, and

2. For distinct $u_1, u_2 \in U$, the paths from u_1 to $\rho(u_1)$ and u_2 to $\rho(u_2)$ are node disjoint (that is, the paths have no nodes is common).

By establishing that $\mathcal{G}_{k,M}$ has a node disjoint correspondence from the set of faulty buses to the designated set of less important buses, we will show that the faulty buses can be treated as less important.

[2] A function $\rho : U \longrightarrow W$ is *injective* iff $u_1 \neq u_2$ implies that $\rho(u_1) \neq \rho(u_2)$; that is, distinct elements of U are mapped to distinct elements of W.

In the following we assume that $\frac{M}{k}$ is an integer. This assumption greatly simplifies the discussion while the extension to arbitrary values of B (relative to k) is quite straightforward.

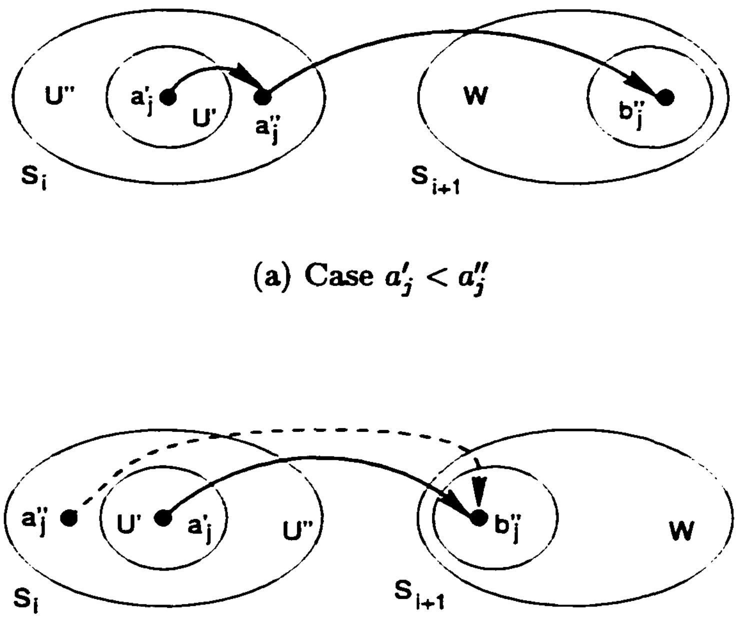
Divide the vertex set, $\{0, 1, \cdots, M-1\}$, of $\mathcal{G}_{k,M}$ into $\frac{M}{k}$ segments $S_0, S_1, \cdots, S_{\frac{M}{k}-1}$, each consisting of k contiguous buses. For $0 \leq i < \frac{M}{k}$, let $S_i = \{ik + j : 0 \leq j < k\}$; that is $S_0 = \{0, 1, 2, \cdots, k-1\}$, $S_1 = \{k, k+1, \cdots, 2k-1\}$, and so on.

Lemma 5.1 *For any* $0 \leq i < \frac{M}{k} - 1$, *let* $U \subseteq S_i$ *and* $W \subseteq S_{i+1}$ *so that* $|U| \leq |S_{i+1} - W|$. *Graph* $\mathcal{G}_{k,M}$ *admits a node-disjoint correspondence* $\rho : U \longrightarrow (S_{i+1} - W)$.

Proof: For each $u \in U$ we will construct a node disjoint path in reverse. That is, starting from $S_{i+1} - W$, we will trace the path back to U. Let $a_j = ik + j$ and $b_j = (i+1)k + j$ (where $0 \leq j < k$) so that $S_i = \{a_j : 0 \leq j < k\}$ and $S_{i+1} = \{b_j : 0 \leq j < k\}$. (Observe that a_j has edges to elements $a_{j+1}, a_{j+2}, \cdots, a_k, b_1, b_2, \cdots, b_j$.) From each $b_j \in S_{i+1} - W$, trace edge $\langle a_j, b_j \rangle$ back to a_j. If $a_j \in U$, then let $\rho(a_j) = b_j$, and edge $\langle a_j, b_j \rangle$ is the required node disjoint path from a_j to $S_{i+1} - W$. We now consider the remaining elements of U and $S_{i+1} - W$ (that is, those not mapped as discussed above). Let sets U' and U'' be as follows:

$$\begin{aligned} U' &= \{a_j \in U : b_j \notin S_{i+1} - W\} \\ &= \text{elements of } U \text{ with no path established to } S_{i+1} - W \text{ as yet;} \\ U'' &= \{a_j \notin U : b_j \in S_{i+1} - W\} \\ &= \text{elements of } S_j \text{ that are not in } U\text{, but have been arrived at} \\ & \text{from } S_{i+1} - W. \end{aligned}$$

Since $|U| \leq |S_{i+1} - W|$, it is easy to see that $|U'| \leq |U''|$. Also note that U' and U'' are disjoint. Let $U' = \{a'_0, a'_1, \cdots, a'_x\}$ and $U'' = \{a''_0, a''_1, \cdots, a''_y\}$, where $y \geq x \geq 1$. Assume that element $a''_j \in U''$ was arrived at from element $b''_j \in S_{i+1} - W$. We will



(a) Case $a'_j < a''_j$

(b) Case $a'_j > a''_j$

Figure 5.3: An illustration of the proof of Lemma 5.1.

now attempt to establish a node disjoint path from a'_j to a''_j. If this can be done, then we have established a node disjoint path from a'_j to b''_j (via a''_j). We consider two cases.

Case 1: $(a'_j < a''_j)$ Since $a'_j, a''_j \in S_i$, there is an edge $\langle a'_j, a''_j \rangle$ in $\mathcal{G}_{k,M}$. (This is because $\mathcal{G}_{k,M}$ has an edge from an element of S_i to all larger elements of S_i). Let $\rho(a'_j) = b''_j$ with the path being $\langle a'_j, a''_j, b''_j \rangle$ (see Figure 5.3(a)).

Case 2: $(a'_j > a''_j)$ Since $a''_j < a'_j < b''_j$ and $\mathcal{G}_{k,M}$ has edge $\langle a''_j, b''_j \rangle$, the edge $\langle a'_j, b''_j \rangle$ also exists. Therefore let $\rho(a'_j) = b''_j$ with the path being edge $\langle a'_j, b''_j \rangle$ (see Figure 5.3(b)).

Since U' and U'' are disjoint, the case $a'_j = a''_j$ is not possible. It is clear that $\rho : U \longrightarrow (S_{i+1} - W)$ is an injection, and that for each $u \in U$, there is a path from u

to $\rho(u)$. To see that these paths are node disjoint observe that paths consisting of a single edge pose no problem. The remaining 2-edge paths (due to Case 1) have the form $\langle a'_j, a''_j, b''_j\rangle$. The only danger is of a''_j being in the path of an element of U that is different from a'_j. Since $a''_j \notin U$ and a''_j is unique for a given a'_j, this is not possible and hence, ρ is a node-disjoint correspondence. ■

Lemma 5.2 *Let $F \subseteq \{b : 0 \leq b < M\}$ be the set of (at most k) faulty buses of $\mathcal{R}_k$. Graph $\mathcal{G}_{k,M}$ admits a node-disjoint correspondence $\rho : F \longrightarrow S_{\frac{M}{k}-1}$.*

Proof: For $0 \leq i < \frac{M}{k}$, let F_i be the set of faulty buses in S_i and let $X_i = F_0 \cup F_1 \cup \cdots \cup F_i$. We will now prove by induction on i (where $0 < i < \frac{M}{k}$) that $\mathcal{G}_{k,M}$ admits a node-disjoint correspondence $\rho_i : X_{i-1} \longrightarrow (S_i - F_i)$. Clearly this will prove the lemma.

For $i = 1$, let $U = F_0 = X_0$ and $W = F_1$. Since $|F_0| + |F_1| \leq |F| \leq k = |S_1|$, we have $|U| = |F_0| \leq |S_1 - F_1| = |S_1 - W|$, so Lemma 5.1 guarantees a node-disjoint correspondence $\rho_1 : X_0 \longrightarrow (S_1 - F_1)$.

Let $\rho_i : X_{i-1} \longrightarrow (S_i - F_i)$ be a node-disjoint correspondence. Let $Y_{i-1} \subseteq S_i - F_i$ be the set to which elements of X_{i-1} have been mapped by ρ_i. Since ρ_i is an injection, $|Y_{i-1}| = |X_{i-1}|$. Notice that $|Y_{i-1}| + |F_i| + |F_{i+1}| = |X_{i-1}| + |F_i| + |F_{i+1}| = |F_0| + |F_1| + \cdots + |F_{i-1}| + |F_i| + |F_{i+1}| \leq |F| \leq k = |S_{i+1}|$.

Therefore $|F_i \cup Y_{i-1}| \leq |S_{i+1} - F_{i+1}|$. With $U = F_i \cup Y_{i-1}$ and $W = F_{i+1}$, Lemma 5.1 gives us a node-disjoint correspondence $\rho : (F_i \cup Y_{i-1}) \longrightarrow (S_{i+1} - F_{i+1})$. Now define $\rho_{i+1} : X_i \longrightarrow (S_{i+1} - F_{i+1})$ as follows.

$$\rho_{i+1}(b) = \begin{cases} \rho(\rho_i(b)), & \text{if } b \in X_{i-1} \\ \rho(b), & \text{if } b \in F_i \end{cases}$$

Since ρ_i and ρ are-node disjoint correspondences and since X_{i-1} and Y_{i-1} are disjoint, ρ_{i+1} is a node-disjoint correspondence. ■

We say that bus b *replaces bus* b' to mean that bus b assumes the identity of bus b', in the process losing its own.

Theorem 5.3 *MBN $\mathcal{R}_k$ can treat the failure of any set, F, of k buses as the failure of a designated set, D, of k buses of $\mathcal{M}$.*

Proof: Without loss of generality, let $D = S_{\frac{M}{k}-1} = \{M-k, M-k+1, \cdots, M-1\}$. By Lemma 5.2, there is a node-disjoint correspondence $\rho : F \longrightarrow D$. For any faulty bus $b \in F$, let the path from b to $\rho(b)$ be $\langle b = b_0, b_1, \cdots, b_x = \rho(b)\rangle$. Since $\langle b_i, b_{i+1}\rangle$ (where $0 \leq i < x$) is an edge of $\mathcal{G}_{k,M}$, bus b_{i+1} of $\mathcal{R}_k$ can replace bus b_i of $\mathcal{M}$. In other words, the faulty bus $b = b_0$ can indirectly be replaced by bus $b_x \in D$, via buses $b_{x-1}, b_{x-2}, \cdots, b_1$. The node disjoint correspondence guarantees that no bus is called upon to replace more than one other bus. The only buses that replace other buses, but are themselves not replaced, are those of D.

Thus the buses can assume new identities so that, regardless of the set, F, of faulty buses, the MBN can treat the designated set, D, as faulty. We now show an example.

Let the set of buses be $\{0, 1, \cdots, 7\}$ and let $k = 3$ with $D = \{5, 6, 7\}$ and $F = \{1, 2, 4\}$. Then the node disjoint correspondence is given by the directed paths $\langle 1, 3, 6\rangle$,$\langle 2, 5\rangle$, and $\langle 4, 7\rangle$. These paths are shown in bold in Figure 5.4. The new identities assumed by buses are as follows:

Original bus	0	1	2	3	4	5	6	7
Replaced by	0	3	5	6	7	–	–	–

Notice that buses of the designated set $D = \{5, 6, 7\}$ are not replaced, and are therefore considered faulty.

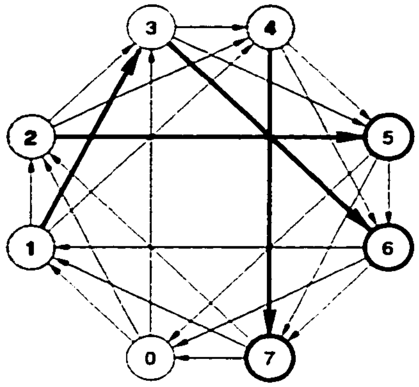


Figure 5.4: Node disjoint correspondences for an example

We now describe how the MBN copes with the loss of buses from the designated set. We say that fault free bus b *emulates* faulty bus b' to mean that b assumes the work of b', in addition to its own. (This is different from the notion of bus b "replacing" bus b', where b loses its identity to b'.) Each faulty bus $b \in D$ (the designated set) is emulated by a fault-free bus $b' \in S_0$. Since there is an edge in $\mathcal{G}_{k,M}$ from bus $M - k + b \in S_{\frac{M}{k}-1}$ to bus $b \in S_0$, for $0 \leq b < k$, such an emulation is always possible. We will refer to the set S_0 as the *emulating set.*

5.2.4 Fault Tolerance Properties of $\mathcal{R}_k$

Earlier in Section 5.2.2 we established that if the degree and loading of $\mathcal{M}$ are d and ℓ, respectively, then the degree and loading of $\mathcal{R}_k$ are $(k+1)d$ and $(k+1)\ell$, respectively. We now derive the time needed for $\mathcal{R}_k$ to emulate a computation on $\mathcal{M}$.

The time required for $\mathcal{R}_k$ to run a computation of $\mathcal{M}$ depends on the choice of the designated set D and the emulating set S_0 (in addition to the computation to be performed). For example, if the buses of S_0 are never used concurrently with those of D, then $\mathcal{R}_k$ emulates $\mathcal{M}$ without any loss of speed, even when k buses fail. At the other extreme, if buses of S_0 and D are used simultaneously for t steps in a computation, then these t steps now run in $2t$ steps (each bus of S_0 does the work of two buses). In general, if a computation on $\mathcal{M}$ has t steps in which a bus of D and its replacement are both used, then $\mathcal{R}_k$ requires t extra steps to emulate $\mathcal{M}$. In particular, if a T-step computation on $\mathcal{M}$ uses the buses of D for at most t steps, then $\mathcal{R}_k$ performs this computation in at most $T + t$ steps. This view permits the performance to be bounded by the set D alone. If D is the set of least used buses, then we have the following result.

Theorem 5.4 *For any $N \times M$ MBN, $\mathcal{M}$, and an integer $0 < k \leq M - 1$, the $N \times M$ MBN, $\mathcal{R}_k$, has the following properties:*

(i) If no bus is faulty, then $\mathcal{R}_k$ can emulate $\mathcal{M}$ without overhead.

(ii) A T-step computation on $\mathcal{M}$ that uses a set of k buses for at most $t \leq T$ steps can be run on $\mathcal{R}_k$ in $T + t$ steps, even if any set of (at most) k buses of $\mathcal{R}_k$ fail.

(iii) If the degree of $\mathcal{M}$ is d, then the degree of $\mathcal{R}_k$ is at most $(k + 1)d$.

(iv) If the loading of $\mathcal{M}$ is ℓ, then the loading of $\mathcal{R}_k$ is at most $(k + 1)\ell$.

■

5.2.5 Processor Faults

Since the fault model assumes an off-line fault processing scheme, the ideas developed so far for bus faults apply to processor faults as well. All that this requires is transposing the $N \times M$ MBN matrix into a $M \times N$ matrix; this interchanges the roles of processors and buses. Therefore, Theorem 5.4 can be restated as follows.

Theorem 5.5 *For any $N \times M$ MBN, $\mathcal{M}$, and an integer $0 \leq k \leq \frac{N}{2}$, the fault-tolerant $N \times M$ MBN, $\mathcal{R}^k$, has the following properties:*

(*i*) *If no processor is faulty, then* $\mathcal{R}^k$ *can emulate* $\mathcal{M}$ *without overhead.*

(*ii*) *A* T*-step computation on* $\mathcal{M}$ *that uses a set of* k *processors for at most* $t \leq T$ *steps can be run on* $\mathcal{R}^k$ *in* $T+t$ *steps, even if any set of* (*at most*) k *processors of* $\mathcal{R}^k$ *fail.*

(*iii*) *If the degree of* $\mathcal{M}$ *is* d*, then the degree of* $\mathcal{R}^k$ *is at most* $(k+1)d$.

(*iv*) *If the loading of* $\mathcal{M}$ *is* ℓ*, then the loading of* $\mathcal{R}^k$ *is at most* $(k+1)\ell$.

■

The MBN, $\mathcal{M}$, could already be one that is resilient to bus faults. In that case, $\mathcal{R}^k$ is resilient to both processor and bus faults. We combine the Theorems 5.4 and 5.5 and the fact that loading and degree cannot exceed N and M respectively, to obtain the main result of this section.

Theorem 5.6 *Given any* $N \times M$ *MBN* $\mathcal{M}$*, and an integers* $1 \leq q \leq \frac{N}{2}$ *and* $1 \leq k \leq \frac{M}{2}$*, the* $N \times M$ *MBN* $\mathcal{R}_k^q$ *has the following properties.*

(*i*) *If no processor or bus is faulty, then* $\mathcal{R}_k^q$ *can emulate* $\mathcal{M}$ *without overhead.*

(*a*) *A* T*-step computation on* $\mathcal{M}$ *that uses a set of* k *buses for* $t_b \leq T$ *steps and* q *processors for* $t_p \leq T$ *steps can be run on* $\mathcal{R}_k^q$ *in* $T + t_b + t_p$ *steps even if any set of* (*at most*) k *buses and* (*at most*) q *processors of* $\mathcal{R}_k^q$ *fail.*

(*b*) *If the degree of* $\mathcal{M}$ *is* d*, then the degree of* $\mathcal{R}_k^q$ *is* $\max(M, (k+1)(q+1)d)$.

(*c*) *If the loading of* $\mathcal{M}$ *is* ℓ*, then the loading of* $\mathcal{R}_k^q$ *is* $\max(N, (q+1)(k+1)\ell)$.

■

5.2.6 Fault Tolerant Binary-Tree MBNs

In this section, we use replication to derive results specific to binary-tree MBNs. We first derive bounds on processor and bus usages in binary-tree MBNs and then use these bounds with Theorems 5.4 and 5.5 to derive results specific to fault-tolerant binary-tree MBNs. Recall the following facts (Chapter 2, page 11) regarding binary-tree algorithms.

1. With the root of $\mathcal{F}(n)$ at level n and the leaves at level 0, for any $0 \leq i \leq n$, there are $2^{n-\ell}$ nodes at level ℓ.

2. Call a communication (non-trivial edge of $\mathcal{F}(n)$) that brings partial results to a level-ℓ node as a *level-ℓ communication*. For $0 < \ell \leq n$, there are at least $2^{n-\ell}$, and at most $2^{n-\ell+1}$, level-ℓ communications; this is because each internal node has at least 1 and at most 2 non-trivial edges from its children.

Consider the problem of running $Bin(n)$ on a $2^n \times 2^m$ MBN, where $0 \leq m < n$. For $1 \leq \ell \leq n-m$, there are at least $2^{n-\ell} \geq 2^m$ level-ℓ communications. For these levels, the number of communications exceeds the number of available buses, so it is reasonable to assume that the MBN minimizes the number of communications (and hence the running time). Therefore for $1 \leq \ell \leq n-m$, there are exactly $2^{n-\ell}$ level-ℓ communications, that are performed on the 2^m buses in $2^{n-\ell-m}$ steps. The total number of steps for levels $1, 2, \cdots, n-m$ is $\sum_{\ell=1}^{n-m} 2^{n-\ell-m} = 2^{n-m} - 1$.

Consider the next step, that executes nodes at level $n-m+1$. This level has at most 2^m communications and potentially uses all the buses. Level $n-m+2$ has at most 2^{m-1} communications and so uses at most 2^{m-1} buses. Similarly, at most 2^{m-2} buses are used at level $n-m+3$. As a result, at least $2^m - 2^{m-2}$ buses are not used at level $n-m+3$. In the same way, at most 2^{m-3} buses are used at level $n-m+4$ and at least $2^m-(2^{m-2}+2^{m-3}) = 2^{m-1}+2^{m-3}$ buses are not used at levels $n-m+3$ and $n-m+4$. Similarly, at most 2^{m-4} buses are used at level $n-m+5$ and at least $2^m - (2^{m-2}+2^{m-3}+2^{m-4}) = 2^{m-1}+2^{m-4}$ buses are not used at levels $n-m+3$,

$n - m + 4$ and $n - m + 5$. In general, we have the following lemma, whose proof is straightforward by induction on $\ell \geq n - m + 3$.

Lemma 5.7 *For $n - m + 3 \leq \ell \leq n$, any $2^n \times 2^m$ MBN running Bin(n) at least $2^{m-1} + 2^{n-\ell+1}$ buses are not used in communications at levels $n - m + 3, n - m + 2, \cdots, \ell$.*

■

A direct consequence of this result is the following Theorem.

Theorem 5.8 *For $0 \leq m \leq n$, any $2^n \times 2^m$ MBN running Bin(n) has at least 2^{m-1} buses, each of which is used for at most $2^{n-m} + 3$ steps.*

Proof: Communications at levels $1, 2, \cdots, n - m$ require $2^{n-m} - 1$ steps, and use all the buses. Since there are at most 2^m (resp., 2^{m-1}) communications at levels $n - m + 1$ (resp., $n - m + 2$), these levels can each take at most 2 steps (even if both communications to a node use the same bus). Lemma 5.7 implies that at least $2^{m-1} + 2 > 2^{m-1}$ buses are not used at levels $n - m + 3, n - m + 2, \cdots, n$. These 2^{m-1} buses are used for at most $2^{n-m} - 1 + 2 + 2 = 2^{n-m} + 3$ steps. ■

We now outline the derivation of similar results for processor usage. We assume that a step is required for a processor to send/receive partial results and perform an internal computation. Clearly there are $2^{n-\ell}$ active processors (nodes) at level ℓ of $\mathcal{F}(n)$.

As explained earlier, assume that *Bin(n)* is run on a 2^n-processor MBN. Suppose, we use a $2^n \times 2^m$ MBN where $m \leq n - 2$. Divide the input into 2^{m+1} groups, each with 2^{n-m-1} inputs. Here it is reasonable to (sequentially and optimally) reduce a group of 2^{n-m-1} inputs to one partial result; 2^{n-m-1} processors of a group are connected to a bus and take turns to send their input to a fixed processor (leader) of the group. All 2^{n-m-1} processors of the group, except the leader, work for only one of the

$2^{n-m-1} - 1$ steps needed to reduce the group. Since there are 2^{m+1} groups, there are at least $(2^{n-m-1} - 1)2^{m+1} = 2^n - 2^{m+1} \geq 2^n - 2^{n-1} = 2^{n-1}$ processors that are used for only one step.

When $m \geq n - 1$, there are enough buses to accommodate all the communications at each level. At level $n \leq \ell < 1$, there are $2^{n-\ell}$ "active" processors. Therefore the number of processors not used in any of the levels $2, 3, \cdots, \ell$ is $2^n - \sum_{i=2}^{\ell} 2^{n-i} = 2^{n-1} + 2^{n-\ell}$ unused processors. Each of levels 0 and 1 can use a processor only once. This is because at most half the processors are used in level 1. Thus, we have the following result.

Lemma 5.9 *For $0 \leq m \leq n$, any $2^n \times 2^m$ MBN running Bin(n) has at least 2^{n-1} processors, each of which is used for at most 2 steps.* ■

Then, by Theorem 5.6 we have the following result.

Theorem 5.10 *For $0 \leq m \leq n$, and any given $2^n \times 2^m$ MBN, $\mathcal{M}$, there is a fault-tolerant $2^n \times 2^m$ MBN, $\mathcal{M}'$, that runs Bin(n) in at most $2^{n-m} + 5$ additional steps, with 2^{n-1} faulty processors and 2^{m-1} faulty buses.* ■

Remarks: There are $2^{n-\ell}$ communications at level ℓ of $\mathcal{F}(n)$. Therefore, until level $n - m$, the number of communications per level exceeds the available buses. Thus, the 2^{n-m} additional steps cannot be avoided. The remaining 5 additional steps are only upper bounds. For existing networks [23, 24], the corresponding number is only 2. In particular, when $n = m - 1$, these MBNs require only 1 extra step to tolerate the failure of half the buses (or processors).

5.3 Recursive Scheduling

In this section we present a second method for converting any given binary-tree MBN into one that is resilient to bus faults. Replication (Section 5.2) works for any MBN, not just binary-tree MBNs. Consequently, it does not exploit features particular to binary-tree algorithms. For example, consider an MBN that, with no faulty buses, executes each level of $\mathcal{F}(n)$ in one step. If this MBN now has one or few faulty buses, then each level with even one faulty bus now requires two steps under replication. In other words, replication fails to exploit the possibility of executing nodes at higher levels (closer to the root) before all lower level nodes have been executed. (Notice that the only requirement in $\mathcal{F}(n)$ is for a node to be executed after all its descendants. It is not required to wait for lower-level non-descendent nodes.) Recursive scheduling exploits the features of binary-tree algorithms to construct fault-tolerant MBNs that run faster than their replication counterparts. However, the loading of the fault-tolerant MBN is somewhat higher, and the method itself is less general, being applicable only to bus faults in binary-tree MBNs.

For $1 \leq m < n$, given any $2^n \times 2^m$ binary-tree MBN $\mathcal{M}$ and integer k (where $1 \leq k = 2^s < 2^m$), recursive scheduling produces a $2^n \times 2^m$ MBN $\mathcal{S}_k$that is resilient to the failure of an arbitrary set of at most k buses. The restriction that $k = 2^s$ admits $k = 1$ and 2, the most probable fault situations. We now outline the major steps in the construction of MBN $\mathcal{S}_k$.

1. Use the given binary-tree MBN $\mathcal{M}$ to construct a $2^n \times (2^m - k)$ MBN $\mathcal{M}'_k$. MBN $\mathcal{M}'_k$ has k fewer buses than $\mathcal{M}$ and is not tolerant to bus failures.

2. Add k buses to $\mathcal{M}'_k$ to convert it into a $2^n \times 2^m$ binary-tree MBN $\mathcal{M}''_k$; the k added buses have no connections at this point.

3. Use replication (Section 5.2) to transform $\mathcal{M}_k''$ into a $2^n \times 2^m$ binary-tree MBN $\mathcal{M}_k'''$, that is resilient to k arbitrary bus faults.

4. Finally, superimpose $\mathcal{M}$ on $\mathcal{M}_k'''$ to obtain $\mathcal{S}_k$. This last step ensures that $\mathcal{S}_k$ behaves exactly as $\mathcal{M}$ in the absence of bus faults.

All of the above steps, except the construction of $\mathcal{M}_k'$ (Step 1), are straightforward; most of the remainder of this section is devoted to the construction of $\mathcal{M}_k'$. We first consider the case where $k = 1$, as the construction of $\mathcal{M}_k'$ can be expressed in terms of $\mathcal{M}_1'$.

5.3.1 An MBN, $\mathcal{M}_1'$, with $2^m - 1$ Buses

In this section we consider the case where $k = 1$ and construct $\mathcal{M}_1'$, a $2^n \times (2^m - 1)$ binary-tree MBN. This MBN is used to define $\mathcal{M}_k'$ for $k = 2^s > 1$ (Section 5.3.2).

For $x > 0$, let $\mathcal{M}_x$ be the $2^x \times 2^{x-1}$ instance of the given MBN $\mathcal{M}$. We will use instances $\mathcal{M}_{m-1}$ and $\mathcal{M}_m$ (among others) in the construction of $\mathcal{M}_1'$.

Recall that a binary-tree MBN can be defined by the manner in which it "schedules" the tree $\mathcal{F}(n)$; i.e., by the labeling of the nodes and non-trivial edges by buses (see Section 2.3, page 14). Here we will define how $\mathcal{F}(n)$ is scheduled by $\mathcal{M}_1'$. Decompose $\mathcal{F}(n)$ into three regions as shown in Figure 5.5. The $2^n \times (2^m - 1)$ MBN $\mathcal{M}_1'$ schedules Regions 1, 2 and 3 in succession (in that order). For each region it uses all $2^m - 1$ buses available to it. This approach is different from that of replication which would have caused each level of the entire tree $\mathcal{F}(n)$ to be executed in succession using at most $2^m - 1$ buses. The 2^n leaves of $\mathcal{F}(n)$ are labeled with the 2^n processors of $\mathcal{M}_1'$. In executing $\mathcal{F}(n)$, an internal node u at level ℓ is labeled only with one of the $2^{n-\ell}$ levels at the subtree rooted at u. Thus, Regions 1 and 2 use disjoint sets of processors of $\mathcal{M}_1'$. The roots of the trees at Regions 1 and 2 are leaves of the tree at

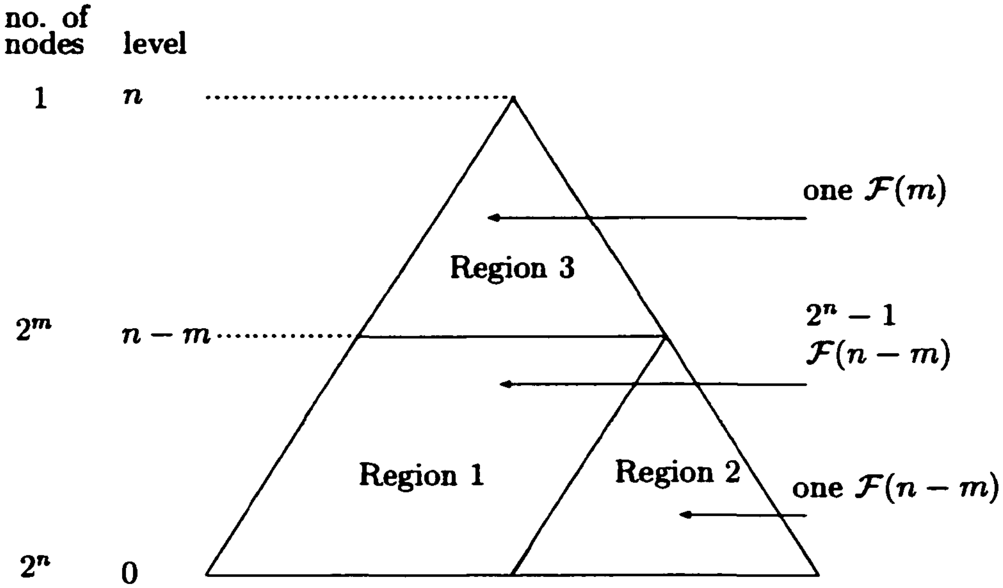


Figure 5.5: Regions of $\mathcal{F}(n)$

Region 3. Region 3 uses only processors at these leaves (level $n-m$ nodes of $\mathcal{F}(n)$). We now describe the three regions in detail and the method used to schedule them on $\mathcal{M}'_1$.

Region 1: This region lies between levels $n-m$ and 0 of the tree $\mathcal{F}(n)$. It consists of $2^m - 1$ trees, each an $\mathcal{F}(n-m)$ rooted at a level $n-m$ node of $\mathcal{F}(n)$. (Of the 2^m such subtrees of $\mathcal{F}(n)$, any $2^m - 1$ may be selected for Region 1.) Each $\mathcal{F}(n-m)$ is scheduled with a single bus. That is, all 2^{n-m} processors at the leaves of the $\mathcal{F}(n-m)$ are connected to a single bus, and their values sequentially reduced to one leader processor assigned to the root of the $\mathcal{F}(n-m)$. Clearly this requires $2^{n-m} - 1$ steps. Since there are $2^m - 1$ buses available, all the $\mathcal{F}(n-m)$s of Region 1 can be scheduled as discussed simultaneously. (Notice that this is a very efficient use of the buses as each of the $2^m - 1$ buses is utilized in all $2^{n-m} - 1$ steps.) Also observe that

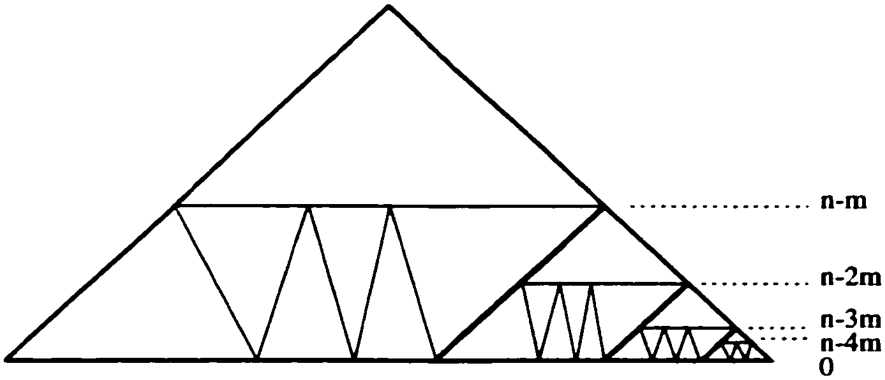


Figure 5.6: An example showing 4 levels of recursive decomposition of $\mathcal{F}(n)$ with $m = 2$ and $k = 1$

in running Region 1, each processor is connected to only one bus, and each bus is connected to 2^{n-m} processors.

Region 2: This region consists of a single $\mathcal{F}(n-m)$ rooted at a level $n-m$ node of tree $\mathcal{F}(n)$. (Of the 2^m such subtrees of $\mathcal{F}(n)$, Region 2 has one, while Region 1 has $2^m - 1$ such subtrees.) If $n - m \leq m$, then Region 2 is scheduled on an instance $\mathcal{M}_{n-m}$ of the given MBN $\mathcal{M}$. Notice that $\mathcal{M}_{n-m}$ uses $2^{n-m-1} \leq 2^{m-1} \leq 2^m - 1$ buses as $m > 0$, so sufficient buses are available.

On the other hand, if $n-m > m$, then the tree $\mathcal{F}(n-m)$ of Region 2 is scheduled recursively on a $2^{n-m} \times (2^m - 1)$ MBN. That is, this $\mathcal{F}(n-m)$ is divided into three regions, each scheduled in sequence (see Figure 5.6).

Region 3: This region consists of levels n to $n-m$ of the tree $\mathcal{F}(n)$. Consequently, it comprises of a single $\mathcal{F}(m)$. Notice that level $n-m$ of $\mathcal{F}(n)$ is shared between Regions 1, 2, and Region 3. The leaves of the $\mathcal{F}(m)$ of Region 3 are the roots of the $\mathcal{F}(n-m)$s of Regions 1 and 2. By virtue of the fact that each tree of Regions 1 and

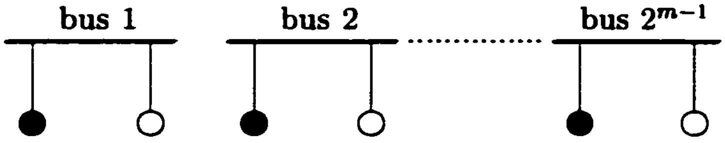


Figure 5.7: Scheduling the lowest level of communications of Region 3. Processors shown in dark hold partial results and move to higher levels of Region 3.

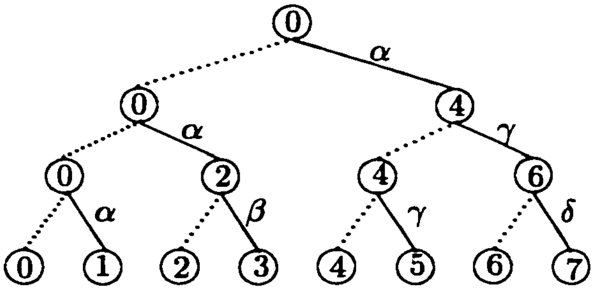


Figure 5.8: An example of a 8 × 4 MBN

2 use disjoint sets of processors, the leaves of the $\mathcal{F}(m)$ of Region 3 are labeled with distinct processors.

The first step of Region 3 schedules the lowest level of communications of the $\mathcal{F}(m)$. This involves reducing 2^m inputs to 2^{m-1} partial results and can be done with $2^{m-1} \leq 2^m - 1$ buses as shown in Figure 5.7.

If $m = 1$, then this completes the execution of Region 3. Otherwise the 2^{m-1} processors holding partial results, along with $2^{m-2} < 2^m - 1$ buses of $\mathcal{M}'_1$, are used to schedule the remainder of Region 3, as a $2^{m-1} \times 2^{m-2}$ instance, $\mathcal{M}_{m-1}$, of the given MBN $\mathcal{M}$.

We now illustrate these ideas using an example, where $n = 4, m = 2$ and $k = 1$. Number the $2^4 = 16$ processors $0, 1, \cdots, 15$ and call the $2^m - k = 3$ available buses α, β, γ. Let the given MBN use a direct mapping (see Figure 5.8). For replication,

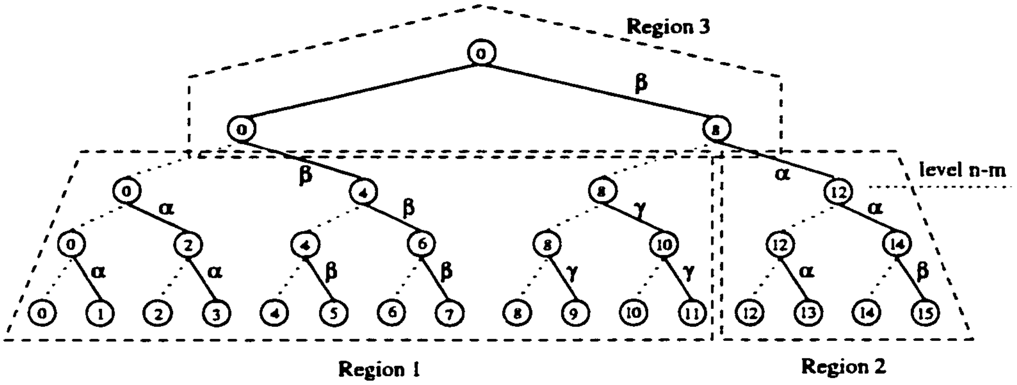


Figure 5.9: Connections of processors and buses with one bus fault

each of $2^2 = 4$ buses are used for first 3 levels. That is, replication uses at least 3 extra steps, for a total running time of 8 steps. The optimal running time for this MBN is only 5 steps. Figure 5.9 illustrates recursive scheduling. In this example, Regions 1, 2 and 3 run in 3, 2 and 2 steps respectively, for a total of 7 steps. In contrast, replication requires 8 steps. This difference will be magnified for large problems.

Running Time: Let $T_1(n,m)$ denote the time to run $Bin(n)$ on the $2^n \times (2^m - 1)$ MBN $\mathcal{M}'_1$. For $x > 0$, let t_x denote the time to run $Bin(x)$ on $\mathcal{M}_x$, a $2^x \times 2^{x-1}$ instance of the given MBN, $\mathcal{M}$; let $t_0 = 0$.

Clearly, $T_1(n,m)$ is the sum of the times needed to run all three regions. Region 1 runs in $2^{m-m} - 1$ steps, and Region 3 in $t_{n-m} + 1$ steps. If $n - m \leq m$, then Region 2 also runs in t_{n-m} steps. Otherwise Region 2 runs (recursively) in $T_1(n-m,m)$ steps. Thus we have the following recurrence.

$$T_1(n,m) = \begin{cases} 2^{n-m} + t_{m-1} + t_{n-m}, & \text{if } n - m \leq m \\ 2^{n-m} + t_{m-1} + T_1(n-m,m), & \text{if } n - m > m \end{cases}$$

Let $i_1 = \lceil \frac{n}{m} \rceil - 1$. It can be verified that $n - i_1 m \leq m < n - (i_1 - 1)m$. Therefore,

$$\begin{aligned}
T_1(n,m) &= 2^{n-m} + t_{m-1} + T_1(n-m,m) \\
&= 2^{n-m} + t_{m-1} + 2^{n-2m} + t_{m-1} + T_1(n-2m,m). \\
&\vdots \\
&= (2^{n-m} + 2^{n-2m} + \cdots + 2^{n-i_1 m}) + i_1 t_{m-1} + T_1(n-i_1 m,m) \\
&= 2^{n-m}\left(\tfrac{1-2^{-i_1 m}}{1-2^{-m}}\right) + i_1 t_{m-1} + t_{n-i_1 m}
\end{aligned}$$

Degree: The degree, $D_1(n,m)$, of $\mathcal{M}_1'$ depends on the manner in which processors are brought together on Region 3. For our initial discussion, if d_x is the degree of a $2^x \times 2^{x-1}$ instance $\mathcal{M}_x$ of the given MBN $\mathcal{M}$, then assume that the degree of the root processor is at most $d_x - 1$; this is indeed the case for the Tree MBN of Section 3.6 (page 44). Under these assumptions, we will show that $D_1(n,m) \leq d_m$, the degree of a $2^m \times 2^{m-1}$ instance of $\mathcal{M}$. In other words, $D_1(n,m)$ is independent of x. Subsequently, we will eliminate the assumption on the degree of the root processor of $\mathcal{M}_m$.

In Region 1, each processor is connected to one bus, and therefore has a degree 1. For Region 2, if $n-m \leq m$, then the region is run on an $\mathcal{M}_{n-m}$ that has degree $d_{n-m} \leq d_m$. If $n-m > m$, then the degree due to Region 2 is $D_1(n-m,n)$ which by the induction hypothesis is at most d_m. In particular, the degree of the root processor r (say) of the tree in Region 2 is at most $d_m - 1$.

In Region 3 we have $2^m - 1$ processors p_i (for $1 \leq i < 2^m$) from the root of the trees in Region 2. Each processor p_i $(1 \leq i < 2^m)$ has degree 1 and processor r has degree at most $d_m - 1$. In the first step of Region 3, processors are paired and each pair reduced to one partial result holding processor. These processors proceed further in Region 3, while the remaining processors are not used any further. Let the processor pairs for

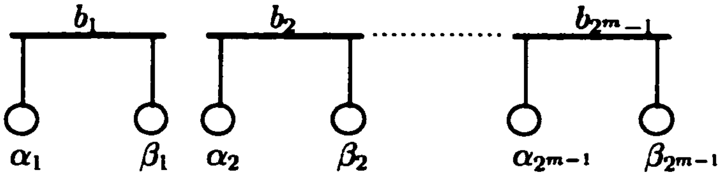


Figure 5.10: Connection of processors and buses in Region 3

the first step be (α_i, β_i), where $1 \leq i \leq 2^{m-1}$ and $\alpha_i, \beta_i \in \{p_1, p_2, \cdots, p_{2^m-1}, r\}$. Of these, let α_i hold the partial results and proceed further and let all α_i have degree 1; this implies the processor $r = \beta_i$ for some i and it does not proceed beyond the first step of Region 3. Let processor α_i (whose degree is 1) be connected to some bus b_i. Since each subtree of Region 2 uses a different bus, we can ensure that each α_i is connected to a distinct bus b_i. The first step of Region 3 connects processors α_i and β_i to bus b_i, allowing processor β_i to send its value to α_i (Figure 5.10). This increases the degree of β_i by 1 and the loading of b_i by 1, as α_i is already connected to b_i. Thus, at the end of this first step each processor β_i has degree of 2 or $d_m \geq 2$ (from Lemma 4.1, page 66). These processors proceed no further in Region 3. The processors α_i that proceed in Region 3 each have a degree of 1, and each is connected to a different bus. Observe every bus of a $2^n \times 2^{n-1}$ binary-tree MBN can be assigned to a distinct processor pair that is connected to it (namely those of the first step). Therefore, by permuting processors and buses of $\mathcal{M}_{m-1}$ appropriately (see Section 2.2, page 13) the remainder of Region 3 can proceed with processors α_i and MBN $\mathcal{M}_{m-1}$ as if these processors had no connections to begin with. After scheduling Region 3, processors α_i now have a degree of at most $d_{m-1} \leq d_m$. Thus the degree of $\mathcal{M}_1'$ is (at most) d_m.

The above derivation assumed that the degree of the root processor of $\mathcal{M}_x$ had a degree of at most $d_x - 1$. As observed earlier, this is indeed the case for many binary-tree MBNs. If this is not the case then assume the degree of $\mathcal{M}_x$ to be $d_x + 1$.

This would allow the degree of the root processor to be incremented in the first step of Region 3 without increasing the MBN degree. Notice that this is possible as the root of the Region 2 subtree does not proceed beyond the first step of Region 3. In summary, the degree of $\mathcal{M}'_1$ is at most $d_m + 1$.

Loading: The loading $L_1(n, m)$ of $\mathcal{M}'_1$ is upper bounded by the sum of the loading due to the three regions. (Unlike processors, that are different for different subtrees of Regions 1 and 2, the same set of buses are used for all regions). Let ℓ_x denote the loading of $\mathcal{M}_x$.

From the discussion of the degree, the loadings due to Regions 1 and 3 are 2^{n-m} and $1 + \ell_{m-1}$, respectively. If $n - m \leq m$, the loading of Region 2 is $\ell_{n-m} \leq \ell_m$; otherwise Region 2 has loading $L_1(n-m, m)$. Thus we have the following recurrence.

$$L_1(n,m) = \begin{cases} 2^{n-m} + 1 + \ell_{m-1} + \ell_{n-m}, & \text{if } n - m \leq m \\ 2^{n-m} + 1 + \ell_{m-1} + L_1(n-m, m), & \text{if } n - m > m \end{cases}$$

which has the solution $L_1(n,m) = 2^{n-m}\left(\frac{1-2^{-i_1 m}}{1-2^{-m}}\right) + i_1(\ell_{m-1} + 1) + \ell_n - i_1 m$, where $i_1 = \lceil \frac{n}{m} \rceil - 1$.

Theorem 5.11 *For any $0 < m < n$, given a $2^m \times 2^{m-1}$ binary-tree MBN $\mathcal{M}_m$ of degree d_m, loading ℓ_m, and running time t_m to run Bin(m), there exists a $2^n \times (2^m - 1)$ MBN $\mathcal{M}'_1$, with degree at most d_m+1, loading of $2^{n-m}\left(\frac{1-2^{-i_1 m}}{1-2^{-m}}\right)+i_1(\ell_{m-1}+1)+\ell_{n-i_1 m}$ and running time of $2^{n-m}\left(\frac{1-2^{-i_1 m}}{1-2^{-m}}\right) + i_1 t_{m-1} + t_{n-i_1 m}$, where $i_1 = \lceil \frac{n}{m} \rceil - 1$.* ■

Remark: If the MBN $\mathcal{M}$ is the Tree MBN of Section 3.6, then the degree of $\mathcal{M}'_1$ is only 3.

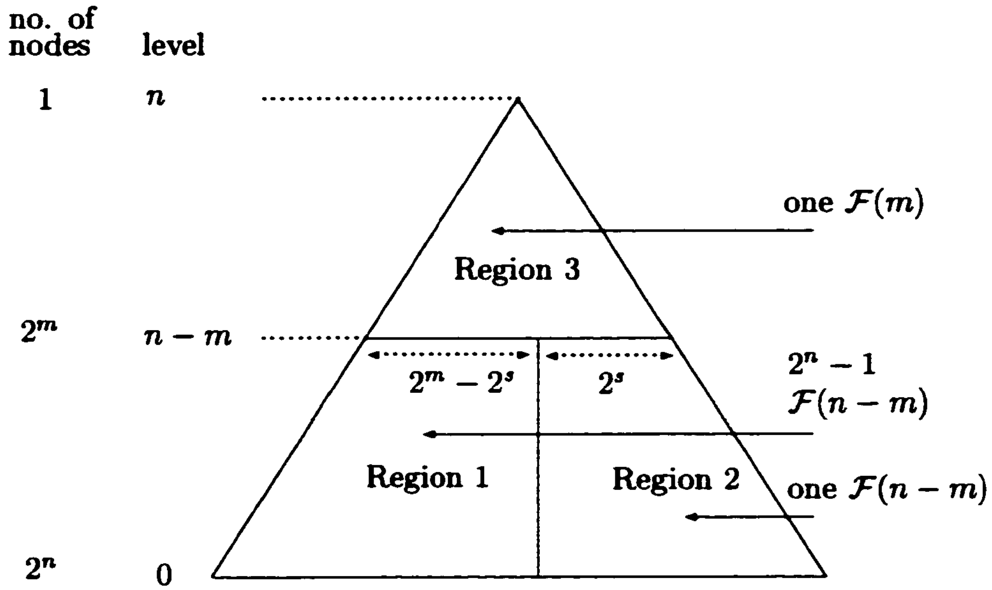


Figure 5.11: Regions of $\mathcal{F}(n)$ for k bus faults

5.3.2 Recursive Scheduling with 2^s Bus Faults

Here we describe the construction of $\mathcal{M}'_k$, a $2^n \times (2^m - k)$ MBN for running $\mathcal{F}(n)$ where $2^{n-1} \geq k = 2^s > 1$. The approach is the same as that for $\mathcal{M}'_1$ (Section 5.3.1). Indeed we use $\mathcal{M}'_1$ to construct $\mathcal{M}'_k$. Divide $\mathcal{F}(n)$ into three regions as shown in the Figure 5.11.

Region 1 now consists of $2^m - 2^s$ subtrees, while Region 2 has the remaining 2^s subtrees. Region 3 is the same as in the $k = 1$ case. Schedule Region 1 as before, with one bus for each subtree. Schedule Region 3 as before with $2^{m-1} < 2^m - k$ buses (as $k \leq 2^{m-1}$). The difference here is in the way Region 2 is scheduled. Region 2 consists of 2^s subtrees, each an $\mathcal{F}(n-m)$; also $s < m$. Divide the available $2^m - 2^s$ buses equally among the subtrees so that each subtree uses $2^{m-s} - 1$ buses. Thus each subtree runs on a $2^{n-m} \times (2^{m-s} - 1)$ MBN, which is an instance of $k = 1$ with n replaced by $n - m$ and m replaced by $m - s$. The running time for each of these subtrees of Region 2 is

$$T_1(n-m, m-s) = 2^{n-2m+s}\left(\frac{1-2^{-i_k(m-s)}}{1-2^{-(m-s)}}\right) + i_k t_{m-s-1} + t_{n-m-i_k(m-s)},$$

where $i_k = \left\lceil \frac{n-m}{m-s} \right\rceil - 1$.

The overall running time $T_k(n,m)$ for $\mathcal{M}_k'$ is $(2^{n-m}-1) + (t_{m-1}+1) + T_1(n-m, m-s)$. By a similar argument the degree and loading of $\mathcal{M}_k'$ are at most $d_m + 1$ and $(2^{n-m}) + (\ell_{m-1}) + L_1(n-m, m-s)$.

Theorem 5.12 *For any $0 < k = 2^s < m < n$, given a $2^m \times 2^{m-1}$ binary-tree MBN $\mathcal{M}_m$ of degree d_m, loading ℓ_m, and running time t_m to run Bin(m), there exists a $2^n \times (2^m - k)$ MBN $\mathcal{M}_k'$, with degree at most $d_m + 1$, loading of $2^{n-m} + \ell_{m-1} + 2^{n-2m+s}\left(\frac{1-2^{-i_k(m-s)}}{1-2^{-(m-s)}}\right) + i_k(\ell_{m-s-1}+1) + \ell_{n-m-i_k(m-s)}$ and running time of $2^{n-m} - 1 + t_{m-1} + 1 + 2^{n-2m+s}\left(\frac{1-2^{-i_k(m-s)}}{1-2^{-(m-s)}}\right) + i_k t_{m-s-1} + t_{n-m-i_k(m-s)}$, where $i_k = \lceil \frac{n-m}{m-s} \rceil - 1$.*

■

5.3.3 Putting it All Together

Given a $2^n \times (2^m - k)$ MBN $\mathcal{M}_k'$ we construct the fault-tolerant $2^n \times 2^m$ MBN $\mathcal{S}_k$ by first adding k dummy buses, then applying replication, and finally superimposing the given $2^n \times 2^m$ MBN $\mathcal{M}$ on it. Clearly, the k designated buses for replication are the added dummy buses they are not used in $\mathcal{M}_k'$. Therefore, $\mathcal{S}_k$ runs in the same time as $\mathcal{M}_k$ when at most k bus-faults are present. The degree of $\mathcal{S}_k$ is at most $d_1 + (k+1)d_2$, where d_1 and d_2 are the degrees of $\mathcal{M}$ and $\mathcal{M}_k'$. Its loading is at most $\ell_1 + (k+2)\ell_2$, where ℓ_1 and ℓ_2 are the loadings of $\mathcal{M}$ and $\mathcal{M}_k'$. Thus we have the following result.

Theorem 5.13 *Let $\mathcal{M}$ be a $2^n \times 2^{n-1}$ binary-tree MBN and let $0 < k = 2^s < m < n$ and $i_k = \lceil \frac{n-m}{m-s} \rceil - 1$. Then recursive scheduling constructs a $2^n \times 2^m$ binary-tree MBN $\mathcal{S}_k$ with the following properties.*

(*i*) *If no bus is faulty, then $\mathcal{S}_k$ can emulate $\mathcal{M}$ with no overhead.*

(*ii*) *If $\mathcal{M}$ takes t_n steps to run $Bin(n)$ then, with at most there are k faulty buses, $\mathcal{S}_k$ runs $Bin(n)$ in $2^{n-m} - 1 + t_{m-1} + 1 + 2^{n-2m+s}\left(\frac{1-2^{-i_k(m-s)}}{1-2^{-(m-s)}}\right) + i_k t_{m-s-1}$ $+ t_{n-m-i_k(m-s)}$ steps.*

(*iii*) *If the degree of $\mathcal{M}$ is d_n, then the degree of $\mathcal{S}_k$ is at most* $\min(2^m, (k+2)d_n)$.

(*iv*) *If the loading of $\mathcal{M}$ is $\ell_{n,m}$, then the loading of $\mathcal{S}_k$ is at most* $[2^{n-m} + \ell_{m-1} + 2^{n-2m+s}\left(\frac{1-2^{-i_k(m-s)}}{1-2^{-(m-s)}}\right) + i_k(\ell_{m-s-1} + 1) + \ell_{n-m-i_k(m-s)}](k+1) + 2^{n-m} + \ell_{m+1} - 2.$

■

Since bus faults and processor faults are treated independently of each other, we can use the results derived in Section 5.2.5 to augment the MBNs that are tolerant to bus faults. Therefore, MBN $\mathcal{S}_k$ can be made tolerant to processor faults as well.

5.4 Comparison of Results

In this section, we compare the two methods. As explained earlier, expect the running time of recursive scheduling to be no more than that of replication in all cases. In addition, we expect the loading of replication to be lower than recursive scheduling in all cases. This is because recursive scheduling uses buses more efficiently (and often), incurring more connections in the process. Table 5.1 shows the running time, loading and the degree of the two methods when the Tree MBN of Section 3.6 is used as the input MBN for the two methods. The running times of both recursive scheduling and replication are the same for cases $m = n - 1$ (regardless of the value of k). This is because here the failure of one bus has the same impact as the failure of half the buses [2]. Therefore, replication is optimal and recursive scheduling cannot improve on it. When the number of faults is large, (for example $k = 2^{m-1}$ in the table, then), both methods again have the same running time. This is because replication assumes that half the buses are faulty, regardless of the actual number of faults. The case where $m = \frac{n}{2}, k = 1$ shows the advantage of recursive scheduling; the running time is about half that of replication. In this case, recursive scheduling makes the maximum use of all the available $(2^{\frac{n}{2}} - 1)$ buses, while replication only uses $2^{\frac{n}{2}-1}$ of the available buses. When the number of faulty buses approaches the total number of buses, then both the methods give the same running time. This situation is not unusual because both the methods have very few buses available and the inefficiency of replication becomes insignificant. As expected, loading of replication is superior to that of recursive scheduling in all cases. The degree of recursive scheduling is marginally larger in all cases due to the fact that we superimpose the original MBN on the fault tolerant MBN to obtain S_k.

5.5 Concluding Remarks

We have proposed two methods for converting any binary-tree MBN to one that is resilient to arbitrary bus faults. One of the methods presented can be used with both processor and bus faults. It also works with any type of MBN while the other method works only with binary-tree MBNs. The fault tolerant MBNs we have designed do not run optimally. However they have much better degree and loading than that proposed by Ali and Vaidyanathan [2]. The problem of designing low-degree, fault-tolerant MBNs that run binary-tree algorithms in optimal number of steps is open.

Replication specifies the additional connections needed in an MBN to map faulty elements to less important elements. An algorithm to perform the required reallo-

Table 5.1: Summary of results

	Case	Recursive scheduling		Replication
Time	$m = n - 1,\ k = 1$	$n + 1$	$=$	$n + 1$
	$m = n - 1,\ k = 2^{n-2}$	$n + 1$	$=$	$n + 1$
	$m = \frac{n}{2},\ k = 1$	$2^{\frac{n}{2}} + n - 1$	$<$	$\frac{n}{2} + 2^{\frac{n}{2}+1} + 1$
	$m = \frac{n}{2},\ k = 2^{\frac{n}{2}-1}$	$\frac{n}{2} + 2^{\frac{n}{2}+1} - 1$	$\approx$	$\frac{n}{2} + 2^{\frac{n}{2}+1} + 1$
Loading	$m = n - 1,\ k = 1$	17	$>$	6
	$m = n - 1,\ k = 2^{n-2}$	$7.2^{n-2} + 10$	$>$	$3.2^{n-2} + 3$
	$m = \frac{n}{2},\ k = 1$	$2.2^{\frac{n}{2}} + 15$	$>$	$2.2^{\frac{n}{2}} + 6$
	$m = \frac{n}{2},\ k = 2^{\frac{n}{2}-1}$	$(2^{\frac{n}{2}-1} + 1)(2^{\frac{n}{2}+1} + \frac{n}{2} + 3)$	$>$	$(2^{\frac{n}{2}-1} + 1)(2^{\frac{n}{2}} + 3)$
Degree	$m = n - 1,\ k = 1$	9	$>$	6
	$m = n - 1,\ k = 2^{n-2}$	$(2^{n-2} + 2)3$	$>$	$(2^{n-2} + 1)3$
	$m = \frac{n}{2},\ k = 1$	9	$>$	6
	$m = \frac{n}{2},\ k = 2^{\frac{n}{2}-1}$	$(2^{\frac{n}{2}} + 2)3$	$>$	$(2^{\frac{n}{2}} + 1)3$

number of processors = 2^n, number of buses = 2^m, number of faulty buses, $k = 2^s$.

We denote $T(n, n-1)$, $\ell_{n,n-1}$ and $d_{n,n-1}$ by $T(n)$, ℓ_n and d_n respectively.

cation of identities is important as well. Though our approach could accommodate handling of bus-faults on the fly, it would incur larger overheads for processor faults, where entire contexts will have to be relocated. Another possible drawback of this work is that it does not address link faults that render the connection from a processor to a bus (rather than an entire bus or the processor) unusable.

Chapter 6

VLSI Layout Lower Bound

This chapter deals with VLSI layouts for optimal-time MBNs. In a related topic, VLSI layouts for the balanced tree point-to-point topology have been thoroughly studied [80]. The balanced tree represents a structure where all edges of a balanced binary tree could be used simultaneously. In contrast, an optimal-time binary-tree algorithm represents a situation in which one level of edges is used at a time. This implies that any layout for a balanced tree would also suffice for a binary-tree MBN. The converse is not true, however. This is because the MBN could reuse the communication resources (and VLSI real state) over different steps, in a manner not possible on a balanced tree. The question we ask here is "is it possible for a binary-tree MBN to be laid out in a smaller area than a balanced tree?" For two of the three cases that we consider, the answer is easily provable to be "no." For the third case, we conjecture that the answer is again "no" and we outline the basis of this conjecture in this chapter.

An $X \times Y$ layout of a structure accommodates the structure in two layers within an $X \times Y$ rectangle. Clearly, the area of an $X \times Y$ layout is XY. In a *perimeter layout*, all processors are placed on the perimeter of the enclosing rectangle. On the other hand, a *dense layout* has no restriction on where processors may be placed. As the name indicates, a dense layout is usually more compact than a perimeter layout.

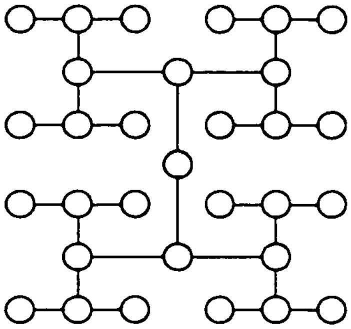


Figure 6.1: H-Tree layout of a 31-processor binary tree

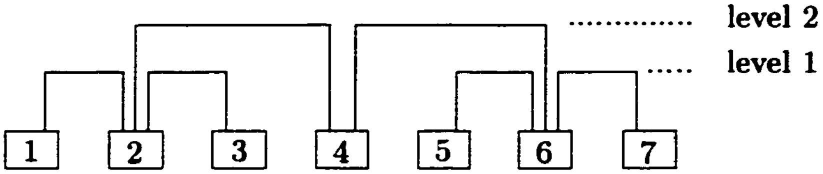


Figure 6.2: 7-node binary tree layout

A perimeter layout, on the other hand, places processors more conveniently for use within a larger context such as meshes enhanced with MBNs (see Chapter 4), or for connecting to pins of a chip. The *aspect ratio* of an $X \times Y$ layout is $\frac{\max(X,Y)}{\min(X,Y)}$

An N-leaf ($\Theta(N)$-node) balanced tree has an optimal $\Theta(N)$ area, constant aspect ratio layout [80] (see Figure 6.1). Therefore an N-processor binary-tree MBN also has such an optimal layout. On the other hand, if a constant aspect ratio, perimeter layout is required, then the perimeter must have $\Omega(N)$ length, as a result of which the area is $\Omega(N^2)$. The well known perimeter layout of a tree [80] can easily be bent around the perimeter of a $\Theta(N \times N)$ square to construct such a layout. Again, this is optimal for a binary-tree MBN.

It can be shown [80] that a high-aspect ratio layout for a balanced tree (with all the processors on one side of the layout) requires $\Omega(N \log N)$ area. Does the same bound

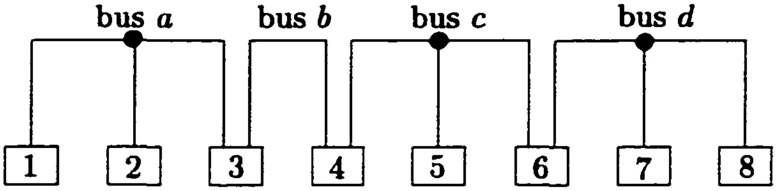


Figure 6.3: 8-processor MBN layout

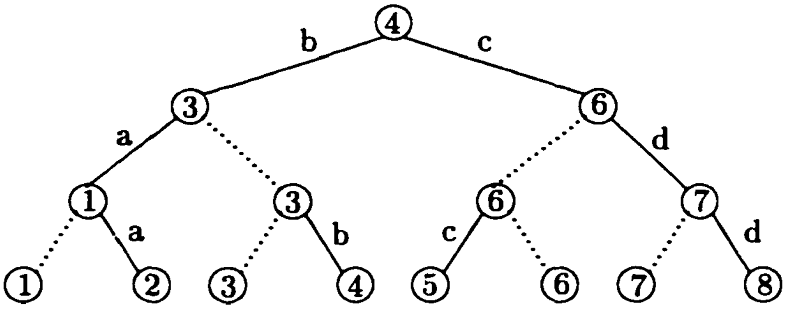


Figure 6.4: 8 processor MBN running *Bin*(3)

apply for binary-tree MBNs as well? The answer is not simple, as a binary-tree MBN uses only one level of edges at a time, and therefore could reuse buses over several steps. For example, a 7-node binary tree (that has two degree-3 nodes) requires at least two levels of wires, as shown in Figure 6.2. On the other hand an optimal-time 8-processor binary-tree MBN can accommodate its buses in a single level (Figure 6.3). Figure 6.4 shows how this MBN runs the 8-input binary-tree algorithm, *Bin*(3). In the remainder of this chapter we describe several steps towards developing a lower bound on the perimeter layout area for optimal-time binary-tree MBNs. It forms the basis of our conjecture that binary-tree MBNs do not have a lower layout area than balanced binary trees. Our argument is arranged as a series of lemmas and one conjecture. If this conjecture can be proved to be true, then this work will establish that any perimeter layout of an optimal-time, binary-tree MBN with N processors requires $\Omega(N \log N)$ area.

In the next section we discuss some preliminary ideas. In Section 6.2, we describe our steps towards the lower bound derivation.

6.1 Preliminaries

In this section we state some assumptions and establish conventions used in subsequent discussion.

6.1.1 VLSI Model

We adopt the most widely used mathematical model for VLSI algorithms [78, 79]. In this model, a VLSI layout consists of horizontal and vertical wires of unit width. Horizontal and vertical wires are laid out on separate layers, and wires on the same layer are separated by unit distance. Whenever a horizontal wire is to be connected to a vertical wire, a contact hole or *via* is cut at the intersection of the two wires and a contact made through this hole. Processors are assumed to occupy unit area. The assumption usually requires a processor to be of constant degree; our lower bound argument does not rely on this assumption, however. Note that this is a "word model" that assumes unit area for processors and width of wires, regardless of the word size used. Since the number of layers in actual fabrications is limited to a few, the size of a VLSI layout is primarily measured by the area of the largest layer (enclosing rectangle). Practical considerations of VLSI fabrication, such as cost and yield dictate that the area be kept as small as possible.

As explained earlier, we will only consider a high aspect ratio, perimeter layout for an optimal-time, binary-tree MBN. Such an MBN for $Bin(n)$, has 2^n processors,

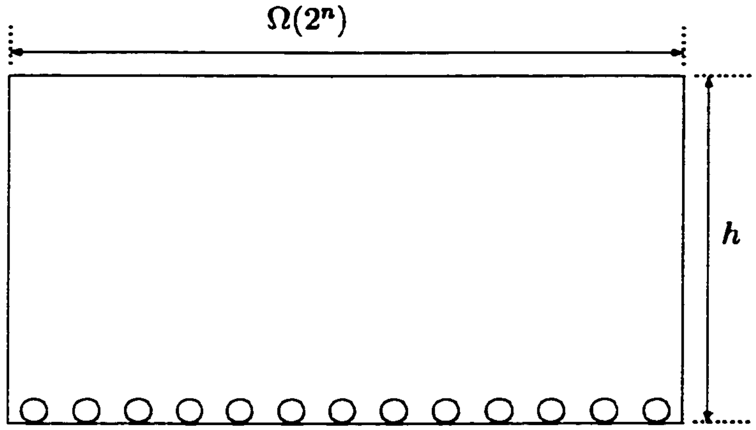


Figure 6.5: Processors are shown as circles

each of which occupies unit area. Therefore, one of the two dimensions of the layout is $\Omega(2^n)$ units long (see Figure 6.5). Without loss of generality, we assume that all 2^n processors are placed on one side of the layout. We will focus on finding a lower bound on the other dimension h (height) of the layout. In deriving this lower bound we assume that vertical wires have no width and, concentrate entirely on the horizontal wire segments. Initially, each processor holds an input. However, no assumption is made about which processor holds the final result of $Bin(n)$.

6.1.2 Definitions and Figure Conventions

Let the *processor axis* of a perimeter layout be the line (edge) of the layout on which processors are placed (in Figure 6.5, the bottom horizontal side of the layout is the processor axis). Assume that, in general the layout orients the processor axis as the lower horizontal lines of the layout. Our approach to the area lower bound first identifies the minimum communication requirements for an optimal-time binary-tree MBN. This communication requirement is represented as horizontal *links*. A link between processors p_1 and p_2 denotes a communication between these processors.

The link is represented as a horizontal line, whose projection on to the processor axis is a line connecting processors p_1 and p_2. The link is not to be confused with a wire or a bus. It is simply a channel (not necessarily placed in a layout) dedicated for communications between processors p_1 and p_2. Our goal at this point is only to identify the existence of such links.

In general, we will view the links from a processor's perspective, and our interest will be restricted to questions such as "does the link cover other links?" (A link λ_1 is said to *cover a link* λ_2 if an infinite vertical line through any point in λ_2 intersects λ_1. A link is said to *cover a processor* iff vertical lines drawn immediately to the left and right of the processor intersect the link.) We now introduce some notation that will help in explaining ideas about the MBN's communication requirements.

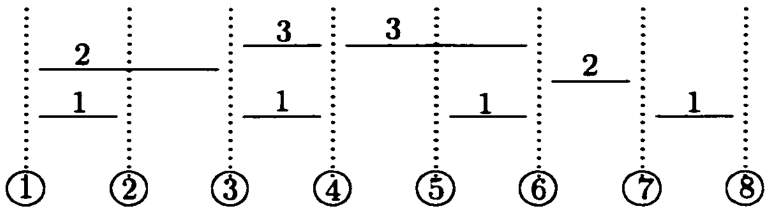


Figure 6.6: Links between processors

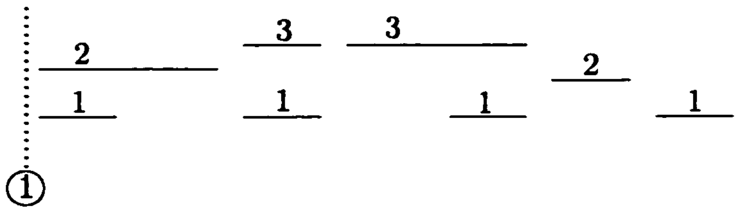


Figure 6.7: View from processor 1

Consider the links shown in Figure 6.6. These links represent the communication requirements shown in figure Figure 6.4. Links labeled 1 are at the lowest level of the tree. Notice that these links are between processor pairs $(1,2)$, $(3,4)$, $(5,6)$ and $(7,8)$, that are involved in a communication in the first step. In step 2, processor pairs $(1,3)$ and $(6,7)$ communicate; these communications correspond to the links labeled 2 in Figure 6.6. The two links labeled 3 between processor pairs $(3,4)$ and $(4,6)$ represent the corresponding non-trivial edges to the roots of Figure 6.4.

In general, each link is labeled with the step at which it is used. For now we will use these link labels only to show the correspondence with Figure 6.6. Figure 6.7 shows the view of these links from processor 1. This view only captures the existence of links and the fact that some links cover others. The length of a link is not an important consideration, except that each link is at least one unit long and a covering link is at least as long as the covered link. In most cases we will only be interested in portions of a subset of the links (as viewed from a processor). For example we may choose to consider the three subsets shown in Figures 6.8, 6.9 and 6.10 as the view from processor 1. Additionally, we may restrict the view to only portions of some links.

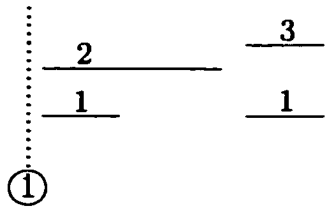


Figure 6.8: Subset view I

Since the links with labels 2 and 3 in Figure 6.8 do not cover any link other than the link labeled 1 directly below it, we can shorten these two links shown (without the labels) in Figure 6.11. Since only the relative position of links is important for our consideration, Figure 6.11 also represents the views in Figure 6.9 and 6.10. Figure 6.11 is also representative of the view from processors 2 and 3 of Figure 6.6

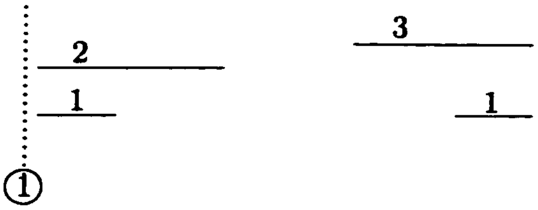


Figure 6.9: Subset view II

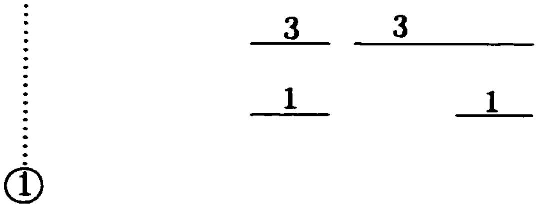


Figure 6.10: Subset view III

but not processors 4, 5, 6, 7 and 8. The view from these processors contain the links shown in Figure 6.12.

Since we will not make a distinction based on the side of the processor that contains links, the views in Figures 6.13 are considered identical. We use a two sided arrow to indicate this as shown in Figure 6.13. Indeed, this represents the view from any of the processors of Figure 6.6.

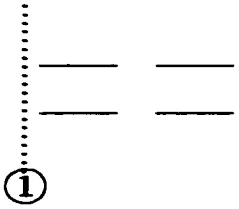


Figure 6.11: Subset view from processor 1

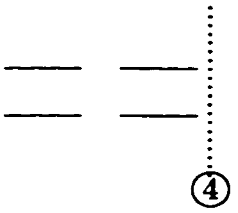


Figure 6.12: Subset view from processor 4

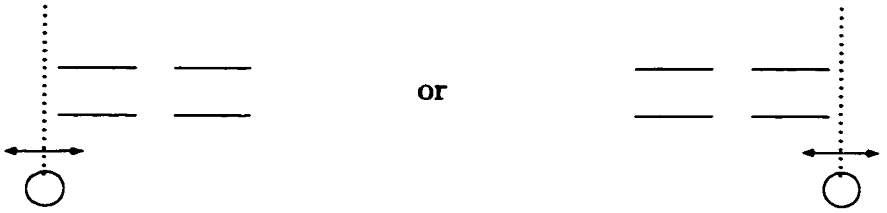
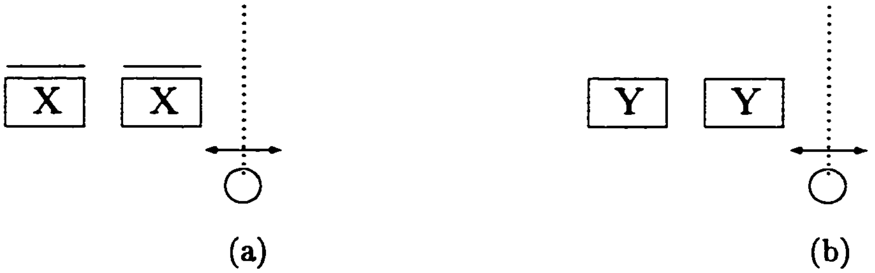


Figure 6.13: Equivalent views

A set of links in the view of a processor will be symbolically represented by a letter enclosed in a box (for instance $\boxed{X}$ or $\boxed{Y}$). For example, if $\boxed{X}$ denotes a single link, then Figure 6.13 can be redrawn as shown in Figure 6.14(a). In general $\boxed{X}$ could



(a) (b)

Figure 6.14: Symbolic representation of a subset view

be any set of links. The notation $\overline{\boxed{X}}$ denotes the links of X along with another link that covers all links of X. If $\boxed{Y}$ denotes $\overline{\boxed{X}}$, then Figure 6.14(a) can be redrawn as Figure 6.14(b). Note that a subset view or the view from a processor <u>containing</u> a set

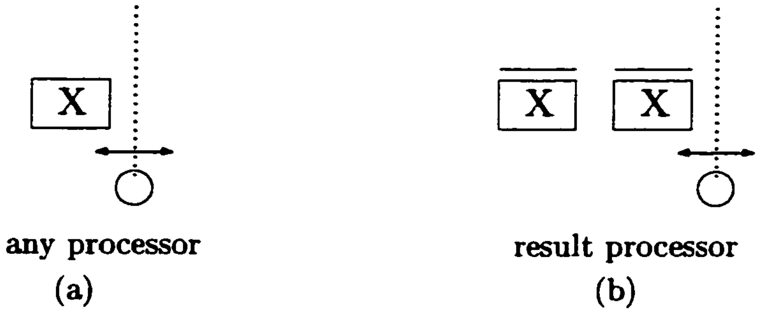


Figure 6.15: Communication Structure for $\mathcal{F}(3)$

of links denotes a subset of the true view from the processor. Since our lower bound argument counts the length of links, such a conservative "subset view" is acceptable.

6.2 Towards the Lower Bound

In this section we derive the results necessary for establishing the lower bound on the height of a perimeter layout of an optimal-time, binary-tree MBN. Our approach first determines a set of links that the view from the final result processor must contain. This "minimum" communication pattern is used with the concept of "collapsing" (that captures the notion of bus reuse) to derive a lower bound on the wire length represented by links. This finally bounds the height of the layout.

6.2.1 Minimum Communication Structure

We start by establishing the minimum communication requirement for any MBN running $Bin(3)$ optimally in 3 steps. We then use this result to derive the communication requirement for $Bin(n)$.

Lemma 6.1 *Let $\mathcal{M}$ be an 8-processor MBN in which each processor contains the subset view shown in Figure 6.15(a) for some set* $\boxed{X}$ *of links. (Assume this view to* *be unrelated to running Bin(3) on $\mathcal{M}$.) If $\mathcal{M}$ can run Bin(3) optimally, then the final result processor contains the subset view shown in Figure 6.15(b).*

Proof: Since $\mathcal{M}$ executes $Bin(3)$ optimally in 3 steps, it must provide a path of at most 3 hops[1] from each processor to the final result processor p_0 (say). Also observe that regardless of where processor p_0 is placed on the processor axis, there must be at least four other processors on one of its sides. In summary, the final result processor p_0 has at least four processors on one side of it, with each of these processor connected to p_0 by a path of at most 3 hops. Without loss of generality, let p_1, p_2, p_3 and p_4 be these four processors to the right of processor p_0, with p_1 nearest to p_0 and p_3 furthest.

Since $\mathcal{M}$ executes $Bin(3)$ optimally, each communication in this execution must be a 1-hop path. Therefore, the subset view from p_0 must contain links to processors p_1, p_2, p_3, p_4 such that each of the processors can be reached from p_0 by traversing at most 3 links. We now consider some cases.

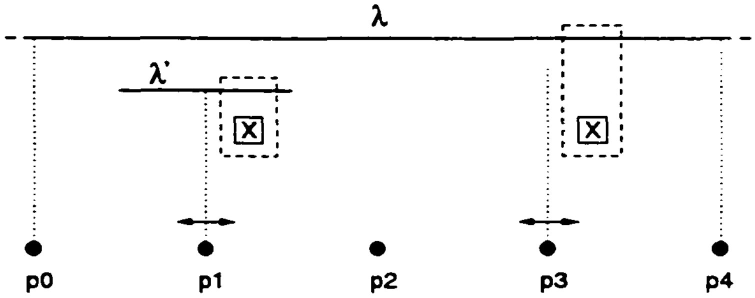


Figure 6.16: Subcase 1(a)

Case1: Suppose there is a link λ (of length 4) between p_0 and p_4; this incudes the case where λ covers p_0 and/or p_4. We now consider some subcases.

[1] A *k-hop path* between processors p and p' is a sequence $\langle p = p_0, b_1, p_1, b_2, p_2, \cdots, p_{k-1}, b_k, p_k \rangle$, where for $1 \leq i \leq k$ processors p_i and p_{i-1} are connected to bus b_i.

Subcase 1(a): Suppose there is a link $\lambda' \neq \lambda$ that covers any of processors p_1, p_2 or p_3 (as shown for p_1 in Figure 6.16). Then the $\boxed{X}$ of the processor (p_1, p_2 or p_3) is covered by λ' while the $\boxed{X}$ of a different one of p_1, p_2 or p_3 is covered by λ. A subset of this situation is the view of Figure 6.15(b) (indicated by the dashed boxes of Figure 6.16).

Subcase 1(b): Suppose λ is the only link covering processors p_1, p_2 and p_3. Then for p_2 to have a path to p_0, there must be links λ', λ'' on both sides of either p_1 (Figure 6.17) or p_3 (Figure 6.18). As shown in these figures, the view from p_0 contains the subset view of Figure 6.15(b).

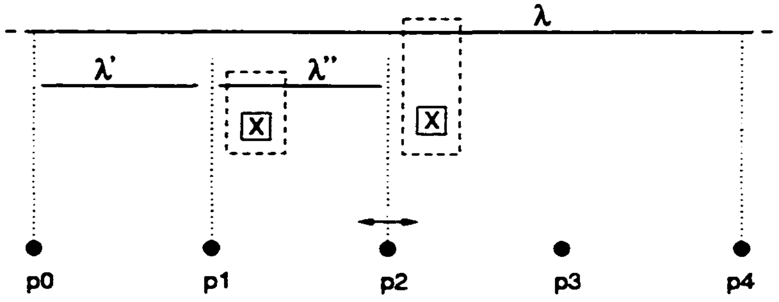


Figure 6.17: Subcase 1(b): p_2-p_1-p_0 link

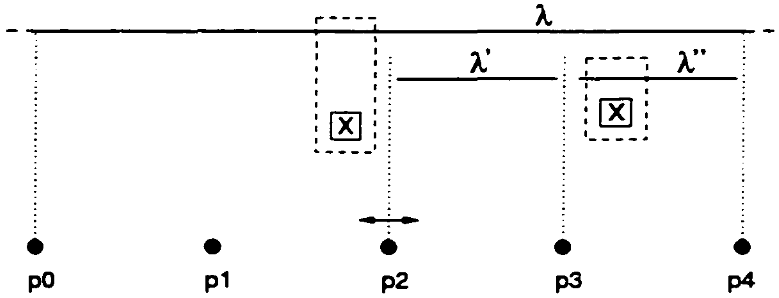


Figure 6.18: Subcase 1(b): p_2-p_3-p_0 link

Case 2: Suppose there is a link λ of length 3. If there is a link $\lambda' \neq \lambda$ that covers any of the processors, then the proof follows as in Figure 6.16. Assume therefore, that there is no link other than λ that covers any of the processors. Without loss of generality, let λ be between p_0 and p_3 (the case where λ is between p_1 and p_4 is analogous). Clearly there must be a link λ' from p_3 to p_4 (Figure 6.19). Processor p_2 connects to p_0 using a link λ'', λ''' via processor p_1 (Figure 6.20) or link λ'' via processor p_3 (Figure 6.21). These figures explain why the lemma hold for these cases.

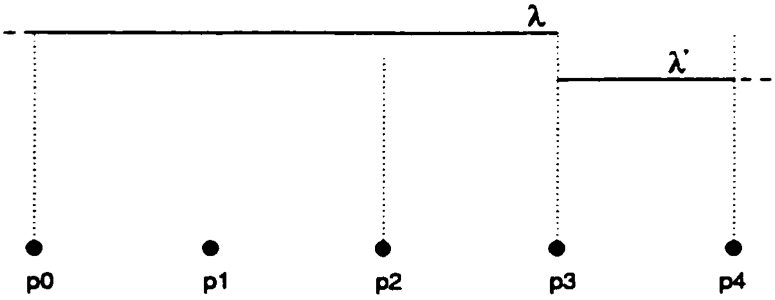


Figure 6.19: Case 2

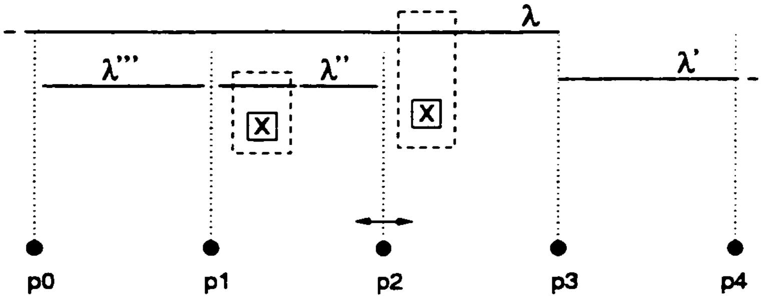


Figure 6.20: Case 2: p_2-p_1-p_0 link

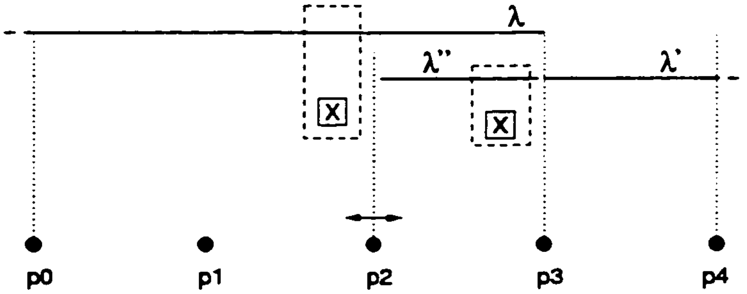


Figure 6.21: Case 2: p_2-p_3-p_0 link

Case 3: Suppose there is a link λ of length 2. Once again assume that there is no link other than λ that covers any of the processors; otherwise it represents the situation in Figure 6.16. We now consider some subcases.

Subcase 3(a): Suppose λ is from p_0 to p_2 (or analogously from p_2 to p_4). For p_4 to get to p_0 there must be links λ', λ'' from p_4 to p_3 and p_3 to p_2. This situation is handled as shown in Figure 6.22.

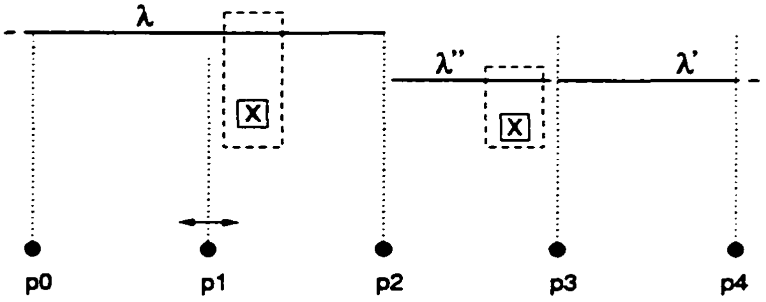


Figure 6.22: Subcase 3(a)

Subcase 3(b): Suppose λ is in the middle between p_1 and p_3. Then the path from p_2 to p_0 must include edge λ' from p_1 to p_0 and λ'' from p_2 to either p_1 (Figure 6.23) or p_3 (Figure 6.24). In addition, there is a link λ''' from p_4 to p_3. These figures show how these cases are handled.

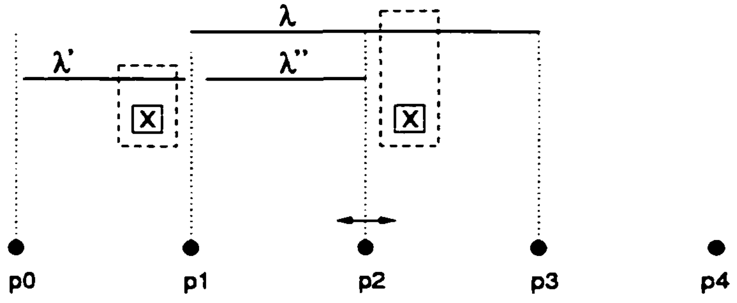


Figure 6.23: Subcase 3(b): p_2-p_1-p_0 link

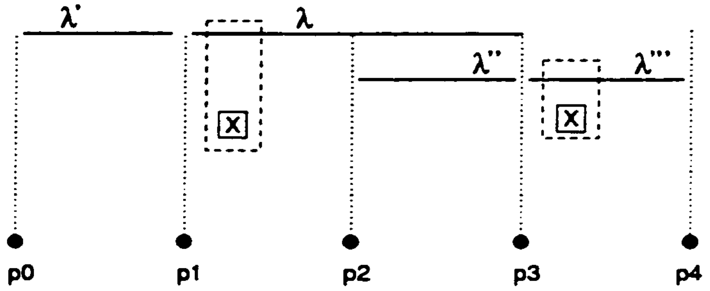


Figure 6.24: Subcase 3(b): p_2-p_3-p_0 link

Since the paths from p_1, p_2, p_3, p_4 to p_0 can have at most 3 hops, there must be at least one link of length 2 or more, so all cases are covered. ■

We now use Lemma 6.1 to identify a minimum set of communication links for an MBN running $Bin(n)$ optimally.

Lemma 6.2 *If an MBN $\mathcal{M}$ runs $Bin(n)$ optimally, then the final result processor contains the subset view of Figure 6.26.*

Proof: Without loss of generality, let $\frac{n}{3}$ be an integer. We proceed by induction on $h = \frac{n}{3} \geq 1$.

If $h = 1$, then we have $n = 3$. From Lemma 6.1 with $\boxed{X}$ being empty, we have the desired result. Assume the assertion of the lemma to hold for $h \geq 1$ and consider an MBN that runs $Bin(3(h+1))$ optimally. The tree $\mathcal{F}(3(h+1))$ can be decomposed into 8 $\mathcal{F}(3h)$s as shown in Figure 6.25. Let the processors at level $3h$ (roots of the $\mathcal{F}(3h)$) each contain subset view $\boxed{X}$. By induction hypothesis $\boxed{X}$ is as shown in Figure 6.26. Then by Lemma 6.1 the roots of $\mathcal{F}(3(h+1))$ contains the subset view of Figure 6.26. Expanding each $\boxed{X}$ as in Figure 6.26 completes the proof. ■

6.2.2 Labeling Links

Subset Figure 6.26 shows the links that the view from the result processor must contain for any MBN running $Bin(n)$ optimally. Clearly, the links are drawn in levels

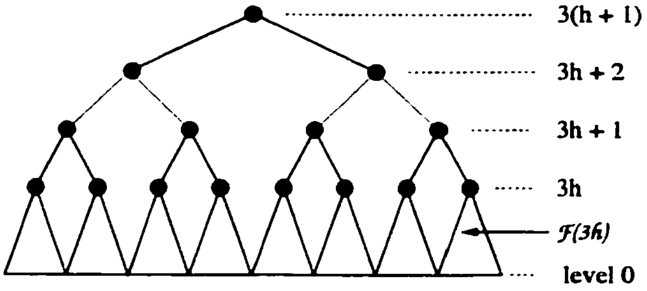


Figure 6.25: $\mathcal{F}(n)$

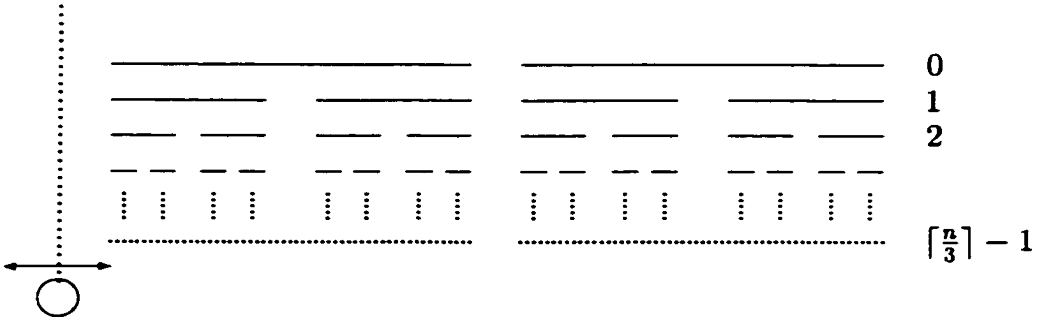


Figure 6.26: View from final result processor

$0, 1, \cdots, \lceil \frac{n}{3} \rceil - 1$ corresponding to 3-level chunks of $\mathcal{F}(n)$. If we label each link by the (unique) step at which it is used, then no two levels of links have common labels, and within a level, there are at most 3 distinct labels (as each level of Figure 6.26 represents a set of $Bin(3)$s that run in 3 steps).

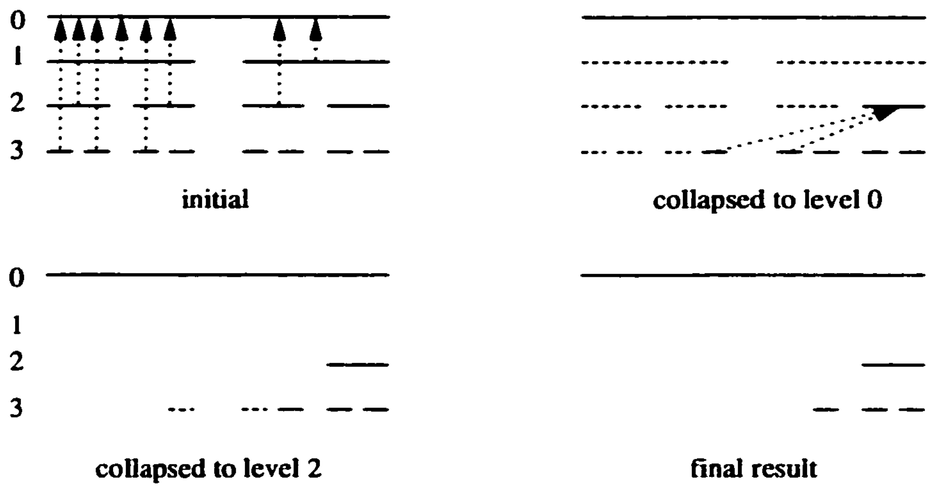
6.2.3 Collapsing Links

To translate the minimum communication requirement of Figure 6.26 into the minimum requirement of perimeter layout, the possibility of links labeled differently using the same physical wire (bus) must be accommodated, as this has the potential to reduce the area. To capture this idea of bus reuse, we introduce the concept of *collapsing* that allow links at different levels (with different labels) to merge. As observed earlier, each level of links represents a set of sub-problem $Bin(3)$s and has at most 3 different time labels. For $0 \leq \ell < \lceil \frac{n}{3} \rceil$ the length of a link at level ℓ is at least $2^{n-\ell-1}$.

Collapsing causes a link to have multiple labels, that indicates the times at which it is (re)used. That is, a link now has a set of labels (rather than a single label). For $i = 1, 2$ let λ_i be a level-ℓ_i link with label set L_i. If $\ell_1 > \ell_2$, then link λ_1 can be *collapsed* into link λ_2 iff λ_2 covers λ_1 and $L_1 \cap L_2$ is empty. After the collapse, the link λ_2 is removed from the communication requirement structure and the label set of λ_2 is changed to $L_1 \cup L_2$. This collapsing captures the idea that link (bus) λ_2 can be used for all its original communications as well as those represented by link (bus) λ_1. Since their labels are disjoint, the link will not be used simultaneously for two communications. As λ_2 covers λ_1, link λ_2 also reaches all processors reached by λ_1. Since the aim is to derive a lower bound on the area using the total length of collapsed links, we will attempt a set of collapses that minimizes this total link length in the communication structure. Indeed, because of the lower bound setting, we will assume that three links from each level $\ell_1 > \ell_2$ can be collapsed into each level-ℓ_2 link, regardless of whether or not the level-ℓ_2 link covers the level-ℓ_1 links.

Define a *maximal collapse* of the communication requirement of Figure 6.26 (or a substructure of this structure) as the result of the following procedure.

```
for level $\ell \longleftarrow 0$ to $\lfloor \frac{n}{3} \rfloor - 1$ do
   for each remaining level-$\ell$ link $\lambda$
      $(i)$ collapse two of the remaining level-$\ell + 1$ links into $\lambda$
      $(ii)$ from each of levels $\ell + 2, \ell + 3, \cdots, \lfloor \frac{n}{3} \rfloor - 1$ collapse three of
        the remaining links from that level into $\lambda$
   end
end
```



initial

collapsed to level 0

collapsed to level 2

final result

Figure 6.27: At each level, collapsed links are shown dotted

Figure 6.27 shows an example of a maximal collapse for a 4 level structure. Note the above procedure allows a link λ_1 to be collapsed into another link λ_2 even if λ_2 does not cover λ_1; however, each link λ_1 is collapsed into at most one other link λ_2.

We now derive a formula for the number of links left at each level after following the above maximal collapse procedure. Without loss of generality assume $\frac{n}{3}$ to be an integer. Before any collapse, level-ℓ (where $0 \leq \ell < \frac{n}{3}$) of the communication structure has $2^{\ell+1}$ links. Let $\eta(\ell)$ denote the number of links left at level-ℓ after a maximal collapse.

Clearly, level-0 links cannot be collapsed, so $\eta(0) = 2$. The 4 level-1 links are all collapsed into the level-0 links (two in each), so $\eta(1) = 0$. For the remaining levels $\ell > 1$, ther are $\eta(\ell - 1)$ collapses into level-$(\ell - 1)$ links and 3 in each of the remaining level-$(\ell - 2)$, level-$(\ell - 3), \cdots, level - 0$ links (assuming level-ℓ has sufficient links for the collapse).

Thus, we have the following relationships. For $\ell \geq 2$

$$\eta(\ell) = 2^{\ell+1} - 2\eta(\ell - 1) - 3\sum_{j=0}^{\ell-2}\eta(j)$$

Therefore,

$$\eta(\ell - 1) = 2^{\ell} - 2\eta(\ell - 2) - 3\sum_{j=0}^{\ell-3}\eta(j).$$

Substituting the second equation from the first we have, $\eta(\ell)+\eta(\ell-1)+\eta(\ell-2) = 2^{\ell}$. That is, the total number of links in three consecutive levels $\ell, \ell - 1$ and $\ell - 2$ is 2^{ℓ}.

Lemma 6.3 *Assuming sufficient higher level links remain for a collapse, the total length of wires after a maximal collapse of the communication structure of Figure 6.26 is* $\Omega(n2^n)$.

Proof: Without loss of generality, let $\frac{n}{3} = h$ be an integer. For $0 \leq k < h$, our earlier observations give $\eta(3k) + \eta(3k + 1) + \eta(3k + 2) = 2^{3k+2}$. Since the shortest wire of level $3k, 3k + 1$ and $3k + 2$ has length $\Omega(2^{n-3k})$ the total length of wires in levels $3k, 3k + 1$ and $3k + 2$ is $L(k) = \Omega(2^{n-3k}2^{3k+2}) = \Omega(2^n)$. Thus the total wire length is $\Omega\sum_{k=0}^{h} 2^n = \Omega(h2^n) = \Omega(n2^n)$. ■

The maximal collapse procedure collapses into lower level (longer) wire before it gets to shorter wires. This is not the only method possible. For example, if shorter wires from some level $\ell > 1$ were collapsed into both level-0 and level-1 wires, then some level-1 wires can no longer be collapsed into level-0 wires.

Figure 6.28 shows another collapsing method. Notice that only one level-2 link can be collapsed to each level-1 link. This is because each level-2 link has two level-3 links collapsed into it. As a result, any set of two level-2 links must have 4 level-3 links collapsed into them, guaranteeing at least one duplicate label (as each level has 3 labels). Thus collapsing two level-2 links into a level-1 link would be equivalent to collapsing 4 level-3 links into level-1 link; this is not permitted. Assuming unit length for the level-3 links, the collapse of Figure 6.28 leaves links whose total length is at

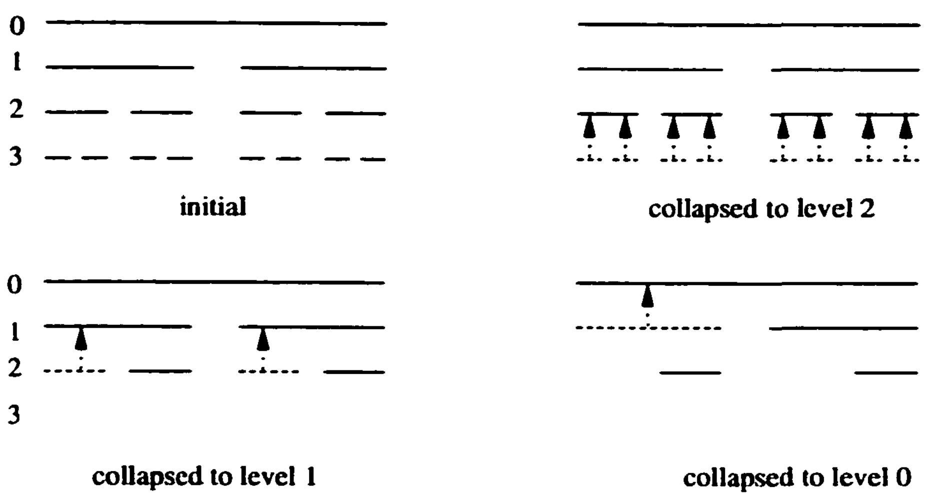


Figure 6.28: A different collapse

least 16. By the same token, the maximal collapse of Figure 6.27 has a length of at least 13.

Clearly many other approaches are possible. The question is "which one leads to the best possible collapse with the shortest total wire length?" Computer simulations seem to indicate that the maximal collapse produces the smallest length of links. Therefore, we have the following conjecture.

Conjecture 1 *No collapsing procedure reduces the total wire length more than the maximal collapse.*

■

We now state the main result of this chapter.

Theorem 6.4 *If Conjecture 1 is true, then the height of a perimeter layout of any 2^n-processor optimal-time binary-tree MBN is $\Omega(n)$.*

Proof: Without loss of generality, assume that the processors are placed a unit distance apart. (They certainly cannot be placed closer, and if they are spread further apart, then their wire length is proportionately larger.) Thus the "width" of the layout can be assumed to be $\Theta(2^n)$. From Lemma 6.3 and Conjecture 1 the layout height is $\Omega\left(\frac{n2^n}{2^n}\right) = \Omega(n)$. ■

Chapter 7

Summary and Future Work

In this research we have investigated various issues on running binary-tree algorithms on MBNs. We have identified relationships among important MBN parameters and established some non-trivial lower bounds. Most of the results are general and apply to all (or a very large class of) binary-tree MBNs. We have developed some novel techniques that may find use in solving problems in other related areas. Most of the results also extend to k-ary trees for $k > 2$.

In Chapter 3 we investigated the relationships among loading, degree and running time of binary-tree MBNs. We developed an accounting scheme to count the number of connections on a bus. We established a series of lower bounds on the loading of optimal-time, degree-2 MBNs for running 2^n-input binary-tree algorithms. The tightest of these bounds established the loading to be $\Omega(\frac{n}{\log n})$. We also identified two important mappings called direct and indirect and established that indirect mapping is essential to achieving constant loading. This result is somewhat surprising, because indirect mapping increases the number of communications. We also showed that if the degree is increased to 3, then optimal-time, constant loading binary-tree MBN exists. We constructed the degree-3, loading-3, Tree MBN with the best possible degree-loading product.

In Chapter 3 we also investigated the possibility of making trade-offs between the running time and loading. We showed that by increasing the running time by a constant factor, loading can be reduced by a non-constant factor. Specifically, we established that if the additional time (beyond the optimal) used by the MBN is t, and if the largest problem size that can be solved in optimal time on a loading-L, degree-2, binary-tree MBN is $2^{\tau(L)}$, then $t \geq \left\lfloor \frac{n}{\tau(L)+1} \right\rfloor$. We presented an example of a degree-2, loading-4, $(2n-3)$-step binary-tree MBN that matches this bound (to within a constant factor) when L is constant.

In Chapter 4 we used MBNs to enhance 2-dimensional meshes. We showed that this method of connecting processors together by multiple buses has significant advantages over the conventional single bus approach for connecting processors together. It allows all existing algorithms on enhanced meshes to be automatically translated into a more implementable platform with a realistic loading. As an MBNs can employ a single bus, our architecture captures all features of most existing enhanced mesh architectures. We derived the running time, loading, degree, number of buses, VLSI area and the aspect ratio of meshes enhanced with the Tree MBN, and showed that our results are better than the best previous results. We also studied buses with segment switches, and showed that segment switches help to reduce loading.

In Chapter 5 we introduced two methods of imparting fault tolerance to MBNs. We accomplished this by adding connections in a controlled manner to MBNs that are not tolerant to faults. The first method, called replication, is a general method that can be used with any MBN (not only binary-tree MBNs) and for both processor and bus faults. An important feature of replication is that it allows a designated set of buses/processors to be treated as faulty, regardless of which buses/processors are actually faulty. This allows the network designer to designate a set of less important buses/processors to be faulty. The second method, called recursive scheduling, is specific to bus faults in binary-tree MBNs. It uses the features of binary-tree MBNs to achieve better speeds compared to replication. The methods for bus faults are independent of that for processor faults. Therefore, tolerance to processor faults can be imparted to an MBN that is already tolerant to bus faults and vice versa.

In Chapter 6 we investigated the VLSI area requirement for a perimeter layout of optimal-time, binary-tree MBNs. The corresponding problem for balanced binary tree topology is well studied. Unlike in a complete binary tree, however, a binary-tree algorithm uses only one level of the tree at a step. Therefore, binary-tree MBNs could reuse the same buses at different steps of the algorithm. We developed a technique to identify the minimum communication requirements for perimeter layouts of optimal-time, binary-tree MBNs and then to "collapse" links to mimic bus reuse. We conjectured that a particular collapsing scheme minimizes the total wire length. (Several computer simulations seemed to indicate that this conjecture is true.) Assuming this conjecture to be true, we established an $\Omega(N \log N)$ lower bound on the VLSI area of a perimeter layout for optimal-time MBNs for N-input binary-tree algorithms.

Future Work: We believe that the $\Omega(\frac{n}{\log n})$ lower bound on the loading established in Chapter 3 is not tight. This is based on the existence of an optimal-time, degree-2, loading-$\Theta(n)$ binary-tree MBN [85] and the fact that degree-2 MBNs tend to introduce a large number of direct nodes for binary-tree algorithms. This is because the ability of a processor to get rid of a partial result while receiving <u>two</u> new partial results is crucial for small loading, and this is not possible on a degree-2 MBN. Future work in this area can focus on bridging the gap between the $\Omega(\frac{n}{\log n})$ lower bound and the

$O(n)$ upper bound. A possible approach for this could be to combine the methods used for establishing the $\Omega(n^{\frac{2}{3}})$ and $\Omega\left(\frac{n}{\log n}\right)$ lower bounds.

The fault tolerance results of Chapter 5 can handle k faults within a given binary-tree MBN. Extension of these methods to the enhanced mesh architecture of Chapter 4 is sensitive to the number of faults in an MBN building block, rather than the entire network. That is, if there are k faults distributed in the entire enhanced mesh, then the best way to address this problem is not known. Currently, the only fail-safe way to handle k faults in the entire enhance mesh is to assume that each of the MBNs can tolerate k faults. This approach could be wasteful for large k.

In Chapter 6, we conjectured that the method used for collapsing the links in the communication structure is optimal. Establishing that this indeed the best is still open. Also we only investigated the area requirements of optimal-time, $N \times \frac{N}{2}$ MBNs. The area requirements for $N \times M$ (for $M < \frac{N}{2}$) binary-tree MBNs and sub-optimal time MBNs are still open problems.

Bibliography

[1] A. Aggarwal, "Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing and Sorting," *IEEE Trans. Computers*, **45**, 1996, pp. 529–539.

[2] A. Ali and R. Vaidyanathan, "Exact Bounds on Running ASCEND/DESCEND and FAN-IN Algorithms on Synchronous Multiple Bus Networks," *IEEE Trans. Parallel & Distributed Systems*, **7**, 1996, pp. 783–790.

[3] B. E. Aupperle and J. F. Meyer, "Fault-Tolerant BIBD Networks," *Proc. International Symposium on Fault Tolerant Computing,* 1988, pp. 306–311.

[4] A. Bar-Noy and D. Peleg, "Square Meshes are not always Optimal," *IEEE Trans. Computers*, **40**, 1991, pp. 196–203.

[5] P. Berthomè, Th. Duboux, T. Hagerup, I. Newman, and A. Schuster, "Self-Simulation for the Passive Optical Star Model," *Proc. 3rd European Symposium on Algorithms*, vol. 979 1995, pp. 369–380.

[6] C. Berge, *Hypergraphs*, North Holland Mathematical Library, vol. 45, 1989.

[7] D. Bhagavathi, V. Bokka, H. Gurla, S. Olariu, and J. L. Schwing, "Time-Optimal Visibility-related Algorithms on Meshes with Multiple Broadcasting," *IEEE Trans. Parallel & Distributed Systems*, **6**, 1995, pp. 687–702.

[8] D. Bhagavathi, P. J. Looges, S. Olariu, J. L. Schwing, and J. Zhang, "A Fast Selection Algorithm for Meshes with Multiple Broadcasting," *IEEE Trans. Parallel & Distributed Systems*, **5**, 1994, pp. 772–777.

[9] L. N. Bhuyan and A. K. Nanda, "Multistage Bus Networks (MBN): An Interconnection Network for Cache Coherent Multiprocessors," *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, 1991, pp. 780–787.

[10] S. H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus," *IEEE Trans. Computers*, **33**, 1984, pp. 133–139.

[11] V. Bokka, H. Gurla, S. Olariu, J. L. Schwing, and L. Wilson, "Time-Optimal Domain-Specific Querying on Enhanced Meshes," *IEEE Trans. Parallel & Distributed Systems*, **8**, 1997, pp. 13–23.

[12] D. Bulka and J. B. Dugan, "Design and Analysis of Multibus Systems Using Projective Geometry," *Proc. International Symposium on Fault Tolerant Computing,* 1992, pp. 122-129.

[13] D. A. Carlson, "Solving Linear Recurrence Systems on Mesh Connected Computers with Multiple Global Buses," *J. Parallel & Distributed Computing*, **8**, 1990, pp. 89–95.

[14] C.-H. Chen and F.-F. Lin, "An Easy to Use Approach for Practical Bus-Based System Design," *IEEE Trans. Computers*, **48**, 1999, pp. 780–793.

[15] T. Chen, T. Kang and R. Yao, "The Connectivity in Hypergraphs and the Design of Fault-Tolerant Multiple Bus Systems," *Proc. International Symposium on Fault Tolerant Computing,* 1988, pp. 374-379.

[16] W. T. Chen and J. P. Sheu, "Performance Analysis of Multiple Bus Interconnection Networks with Hierarchical Requesting Models," *IEEE Trans. Computers*, **40**, 1991, pp. 834–842.

[17] Y. Chen, W. Chen, G. Chen, and J. Sheu, "Designing Efficient Parallel Algorithms on Mesh Connected Computers with Multiple Broadcasting," *IEEE Trans. Parallel & Distributed Systems*, **1**, 1990, pp. 241–246.

[18] I. Chlamtac and S. Kutten, "Tree-based Broadcasting in Multihop Radio Networks," *IEEE Trans. Computers*, **36**, 1987, pp. 1209–1223.

[19] K.-L. Chung, "Prefix Computations on a Generalized Mesh-Connected Computer with Multiple Buses," *IEEE Trans. Parallel & Distributed Systems*, **6**, 1995, pp. 196–199.

[20] D. Coudert, A. Ferreira and X. Munoz, "Multiprocessor Architectures Using Multi-OPS Lightwave Networks and Distributed Control," *International Parallel Processing Symposium*, **12**, 1998, pp. 151–155.

[21] C. Das and L. Bhuyan, "Bandwidth Availability of Multiple-Bus Multiprocessors," *IEEE Trans. Computers*, **34**, 1985, pp. 918–926.

[22] R. Decher and L. Kleinrock, "Broadcast Communication and Distributed Algorithms," *IEEE Trans. Computers*, **35**, 1986, pp. 210–219.

[23] H. P. Dharmasena and R. Vaidyanathan, "An-Optimal Multiple Bus Networks for Fan-in Algorithms," *Proc. International Conference on Parallel Processing,* 1997, pp. 100–103.

[24] H. P. Dharmasena and R. Vaidyanathan, "Lower Bound on the Loading of Degree-2 Multiple Bus Networks for Binary-Tree Algorithms," *Proc. International Parallel Processing Symposium,* 1999, pp. 21–25.

[25] O. M. Dighe, R. Vaidyanathan and S. Q. Zheng, "The TBN: A Versatile Building Block for VLSI Parallel Architectures," *Proc. 6th ISCA International Conference on Computer Applications in Design, Simulation and Analysis,* 1993, pp. 52–55.

[26] O. M. Dighe, R. Vaidyanathan and S. Q. Zheng, "Bus-Based Tree Structure for efficient Parallel Computation," *Proc. International Conference on Parallel Processing,* 1993, pp. 158–161.

[27] O. M. Dighe, R. Vaidyanathan and S. Q. Zheng, "The Bus-Connected Ringed Tree: A Versatile Interconnection Network," *J. Parallel & Distributed Computing*, **33**, 1996, pp. 189–196.

[28] M. Dubois, "Throughput Analysis of Cahche-Based Multiprocessors with Multiple Buses," *IEEE Trans. Computers*, **37**, 1988, pp. 58–70.

[29] M. Feldman, R. Vaidyanathan and A. El-Amawy, "High Speed, High Capacity Bused Interconnects Using Optical Slab Waveguides," *Proc. 1999 Workshop on Optics in Computer Science,* Springer Verlag Lecture Notes in Computer Science, vol. 1586, pp. 924–937.

[30] S. Fujita and M. Yamashitar, "Fast Gossiping on Mesh-Bus Computers," *IEEE Trans. Computers*, **45**, 1996, pp. 1326–1330.

[31] A. Ghafoor, A. L. Goel, J. K. Chan and S. Sheikh, "Reliability Analysis of a Fault-Tolerant Multi-Bus Multiprocessor Systems," *Proc.3rd IEEE Symposium on Parallel and Disributed Processing,* 1991, pp. 436–443.

[32] R. Giorgi and C. Antonio, "PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors," *IEEE Trans. Parallel & Distributed Systems*, **7**, 1999, pp. 742–761.

[33] Z. Guo and R. G. Melhem, "Embeddding Binary X-Trees and Pyramids in Processor Arrays with Spanning Buses," *IEEE Trans. Parallel & Distributed Systems*, **5**, 1994, pp. 664–672.

[34] J. D. Hadley and B. L. Hutchings, "Design Methodologies for partially Reconfigured Systems," *Proc. Workshop on FPGAs for Custom Computing Machines,* 1995, pp. 78–84.

[35] T. Hayashi, K. Nakano and S. Olariu, "Randomized Initialization Protocols for Packet Radio Networks," *Proc. International Parallel Processing Symposium,* 1999, pp. 544–548.

[36] M. A. Holliday and M. K. Vernon, "Exact Performance Estimates for Multiprocessor Memory and Bus Interferences," *IEEE Trans. Computers*, **36**, 1987, pp. 76–85.

[37] K. Hwang, P. S. Tseng and D. Kim, "An Orthogonal Multiprocessor for Parallel Scientific Computations," *IEEE Trans. Computers*, **36**, 1989, pp. 47–60.

[38] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Co., 1992.

[39] H. Jiang and K. C. Smith, "PPMB: A Partial-Multiple-Bus Multiprocessor Architecture with Improved Cost Effectiveness," *IEEE Trans. Computers*, **41**, 1992, pp. 361–366.

[40] S. T. Kamath and R. Vaidyanathan, "Running Weak Hypercube Algorithms on Multiple Bus Networks," *Proc. ISCA International Conference on Parallel and Distributed Systems,* 1997, pp. 217–222.

[41] M. A. S. Khalid and J. Rose, "Hardwired-Clusters Partial-Crossbar: A Hierarchical Routing Architecture for Multi-FPGA Syatems," *Proc. 6th Reconfigurable Architecture Workshop,* 1999, pp. 597–605.

[42] J. Kilian, S. Kipnis and C. E. Leiserson, "The Organization of Permutation Architectures with Bussed Interconnections," *IEEE Trans. Computers*, **39**, 1990, pp. 1346–1358.

[43] J. Kim, "Segmented Multiple Bus Systems," Ph.D Thesis, Dept. of Electrical & Computer Eng., Louisiana State University, 1997.

[44] J. H. Kim and P. K. Rhee, "The Rule-Based Approach to Reconfiguration of 2-D Processor Arrays," *IEEE Trans. Computers*, **42**, 1993, pp. 1403–1408.

[45] H.-K. Ku and J. P. Hayes, "Connective Fault tolerance in Multiple-Bus Systems," *IEEE Trans. Parallel & Distributed Systems*, **8**, 1997, pp. 574–586.

[46] P. Kulasinghe and A. El-Amawy, "On the Complexity of Bussed Interconnections," *IEEE Trans. Computers*, **44**, 1995, pp. 1248–1251.

[47] P. Kulasinghe and A. El-Amawy, "Optimal Realizations of Sets of Interconnection Functions on Synchronous Multiple Bus Systems," *IEEE Trans. Computers*, **45**, 1996, pp. 964–969.

[48] P. Kulasinghe, "Combinatorial Analysis and Design of Optimal Multiple Bus Systems for Parallel Algorithms," Ph.D Thesis, Dept. of Electrical & Computer Eng., Louisiana State University, 1995.

[49] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[50] Y. Li, J. Wu and S. Q. Zheng, "An Interconnection Network Based on the Dual of a Hypercube," *Proc. ISCA International Conference on Parallel and Distributed Systems,* 1997, pp. 263–268.

[51] R. Lin, S. Olariu, J. L. Schwing and B. F. Wang, "The Mesh with Hybrid Buses: An Efficient Parallel Architecture for Digital Geometry," *IEEE Trans. Parallel & Distributed Systems*, **10**, 1999, pp. 266–280.

[52] J. M. Marberg and E. Gafni, "Sorting and Selection in Multi-Channel Broadcast Networks," *Proc. International Conference on Parallel Processing,* 1985, pp. 846–850.

[53] S. M. Mahmud, "Performance Analysis of Multilevel Bus Networks for Hierarchical Multiprocessors," *IEEE Trans. Computers*, **7**, 1994, pp. 789–799.

[54] M. D. Mickunas, "Using Projective Geometry to Design Bus-Connection Networks," *Proc. ACM/IEEE Workshop on Interconnection Networks for Parallel and Distributed Processing,* 1980, pp. 47–55.

[55] T. N. Mudge and A. B. Al-Sadoun, "A Semi-Markov Model for the Performance of Multiple-Bus Systems ," *IEEE Trans. Computers*, **34**, 1985, pp. 934–942.

[56] T. N. Mudge, J. P. Hayes and D. C. Winsor, "Multiple Bus Architectures," *IEEE Computer,* 1987, pp. 42–48.

[57] S. Nadella, "Fault Tolerant Multiple Bus Networks for Fan-in Algorithms," Masters Thesis, Dept. of Electrical & Computer Eng., Louisiana State University, 1993.

[58] H. Nagano, A. Matsura and A. Nagoya, "An Efficient Implementation Method of Fractal Image Compression on Dynamically Reconfigurable Architecture," *Proc. 6th Reconfigurable Architecture Workshop,* 1999, pp. 670–678.

[59] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures," *Parallel Processing Letters*, **5**, 1995, pp. 111–124.

[60] K. Nakano, S. Olariu, and J. L. Schwing, "Broadcast-Efficient Sorting in the Presence of Few Channels," *Proc. International Conference on Parallel Processing*, 1997, pp. 12–15.

[61] K. Nakano, S. Olariu and J. L. Schwing, "Broadcast-Efficient Algorithms on the Coarse-Grain Broadcast Communication Model with Few Channels," *Proc. International Parallel Processing Symposium,* 1998, pp. 1–6.

[62] D. Nassimi, "Parallel Algorithms for Classes ($\pm 2^b$) DESCEND and ASCEND Computations on a SIMD Hypercube," *IEEE Trans. Parallel & Distributed Systems*, **4**, 1993, pp. 1372–1381.

[63] A. Padmanabhan, "Design of Multibus Networks for ASCEND/DESCEND and FAN-IN Algorithms," Masters Thesis, Dept. of Electrical & Computer Eng., Louisiana State University, 1992.

[64] Y. Pan, S. Q. Zheng, K.- L., and Hong Shen, "Semigroup and Prefix Computations on Improved Generalized Mesh-Connected Computers with Multiple Buses," To appear in *Proc. International Symposium on Parallel & distributed Processing,* 2000.

[65] G. Panchapakesan and A. Sengupta, "On a Light Wave Network Topology Using Kautz Diagraphs," *IEEE Trans. Computers*, **48**, 1999, pp. 1131–1137.

[66] R. C. Pearce, J. A. Field and W. D. Little, "Asynchronous Arbiter Module," *IEEE Trans. Computers*, **24**, 1975, pp. 931–932.

[67] D. K. Pradhan, "Fault-Tolerant Multiprocessor Link and Bus Network Architectures," *IEEE Trans. Computers*, **34**, 1985, pp. 33–45.

[68] D. K. Pradhan, Z. Hanquan and M. L. Schlumberger, "Fault-Tolerant Multibus Architectures for Multiprocessors," *Proc. Symp. on Fault-Tolerant Computing*, 1984, pp. 400–408.

[69] V. K. Prasanna Kumar and C. S. Raghavendra, "Array Processor with Multiple Broadcasting," *J. Parallel & Distributed Computing*, **4**, 1987, pp. 173–190.

[70] C. Qiao and R. G. Melhem, "Time-Division Optical Communications in Multiprocessor Arrays," *IEEE Trans. Computers*, **42**, 1993, pp. 577–590.

[71] C. S. Raghavendra, "HMESH: A VLSI Architecture for Parallel Processing," *Proc. Conference on Algorithms and Hardware for Parallel Processing,* Springer Verlag Lecture Notes in Computer Science, vol. 237, 1986, pp. 76–83.

[72] S. Rajasekaran, "Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing and Sorting," *IEEE Trans. Computers*, **45**, 1996, pp. 529–539.

[73] M. R. Samantham and D. K. Pradhan, "The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI," *IEEE Trans. Computers*, **38**, 1989, pp. 567–581; Corrections in *IEEE Trans. Computers*, **40**, 1991, pp. 122.

[74] S. M. Scalera, J. J. Murray and S. Lease, "A Mathematical Benefit Analysis of Context Switching Reconfigurable Computing," *Proc. Reconfigurable Architecture Workshop,* 1998, pp. 73–78.

[75] M. J. Serrano and B. Parhami, "Optimal Architectures and Algorithms for Mesh-Connected Parallel Computers with Separable Row/Column Buses," *IEEE Trans. Parallel & Distributed Systems*, **4**, 1993, pp. 1073–1080.

[76] Q. F. Stout, "Mesh-Connected Computer with Broadcasting," *IEEE Trans. Computers*, **32**, 1983, pp. 826–829.

[77] R. K. Thiruchelvan, J. L. Trahan and R. Vaidyanathan, "On the Power of Segmenting and Fusing Buses," *J. Parallel & Distributed Computing*, **34**, 1996, pp. 82–94.

[78] C. D. Thompson, "Area-time Complexity for VLSI," *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, **5**, 1979, pp. 81–88.

[79] C. D. Thompson, "A Complexity Theory for VLSI," Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, 1980.

[80] J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Potomac, MD, 1984.

[81] A. Varma, "Combinatorial Design of Bus-based Interconnection Structures," *Research Report RC 12550,* IBM Research Division, September, 1986.

[82] R. Vaidyanathan, C. R. P. Hartmann and P. K. Varshney, "Running ASCEND, DESCEND and PIPELINE Algorithms in Parallel Using Small Processors ," *Information Processing Letters*, **46**, 1993, pp. 31–36.

[83] R. Vaidyanathan, "Design of Multiple Bus Interconnection Networks for Fan-in Computations," *Proc. 29th Annual Allerton Conf. on Communication, Control & Computing,* 1991, pp. 1093–1102.

[84] R. Vaidyanathan and S. Nadella, "Fault-Tolerant Multiple Bus Networks for Fan-In Algorithms," *Proc. International Parallel Processing Symposium*, 1996, pp. 674–681.

[85] R. Vaidyanathan and A. Padmanabhan, "Bus-Based Networks for Fan-in and Uniform Hypercube Algorithms," *Parallel Computing*, **21**, 1995, pp. 1807–1821.

[86] R. Vaidyanathan and J. L. Trahan, "Optimal Simulation of Multidimensional Reconfigurable Meshes by Two Dimensional Reconfigurable Meshes," *Information Processing Letters*, **47**, 1993, pp. 267–273.

[87] T. A. Varvarigou, V. P. Roychowdhury and T. Kailath, "Reconfiguring Processor Array Using Multiple Track Models: The 3-Track-1-Spare-Approach," *IEEE Trans. Computers*, **42**, 1993, pp. 1281–1293.

[88] J. Villasenor and W. H. Mangione-Smith, "Configurable Computing," *Scientific American,* vol. 276 no. 6 1997, pp. 66–71.

[89] J. F. Wakerly, *Digital System Design, Principles & Practice*, Prentice Hall Inc., Upper Saddle River, NJ, 1994.

[90] D. C. Winsor and T. N. Mudge, "Analysis of Bus Hierarchies for Multiprocessors," *Proc. International Symposium on Computer Architecture,* 1988, pp. 100–107.

[91] M. J. Wirthlin and B. L. Hutchings, "DISC: The Dynamic Instruction Set Computer," *Proc. of the SPIE - Field Programmable Gate Arrays (FPGAs) for Fast Board and Reconfigurable Computing,* vol. 2607 1995, pp. 92–103.

[92] M. Wojko and H. ElGindy, "Configuration Sequencing with Self-Configurable Binary Multipliers," *Proc. 6th Reconfigurable Architecture Worrkshop,* 1999, pp. 643–651.

[93] "Xilink Inc. XC400E and XC4000X Series Field Programmable Gate Arrays," product specification, 1997.

[94] Q. Yang and L. N. Bhuyan, "Analysis of Packet-Switched Multiple-Bus Multiprocessor Systems," *IEEE Trans. Computers*, **40**, 1991, pp. 352–357.

Vita

Hettihewage Prasanna Dharmasena was born in Kurunegala, Sri Lanka. He received the bachelor of science degree in electronics and telecommunication engineering from the University of Moratuwa, in 1983. From 1983 to 1985 he worked as an assistant lecturer at the same University. He received the degree of master of science in electrical and computer engineering from Louisiana State University in 1987. He is currently employed as a computer analyst at Louisiana State University. He will receive the degree of Doctor of Philosophy in May, 2000.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: H. P. Dharmasena

Major Field: Electrical Engineering

Title of Dissertation: Multiple Bus Networks for Binary - Tree Algorithms

Approved:

R. Vaidyanathan
Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

March 23, 2000