Abstract

Title of Dissertation: Message Routing and Problem Solving in Multiprocessor Networks

Lee Guan, Doctor of Philosophy, 1997

Dissertation directed by: Assistant Professor Ronald I. Greenberg
Department of Electrical Engineering

This thesis investigates three topics regarding message routing and problem solving in multiprocessor networks. The first topic is performance comparison of interconnection networks and routing schemes for massively parallel computers. This thesis provides empirical comparisons of the communication capabilities of several popular interconnection networks, using throughput and latency as performance metrics. Comparisons between the networks are performed with equal hardware cost, based alternately on bisection width, pin-out, and layout area. Routing schemes used for comparison include both wormhole routing and store-and-forward routing.

The second topic is performance modeling for wormhole routed interconnection networks. This thesis presents an extended performance model for

wormhole routed interconnection networks that includes the modeling of multi-server channels. The model is general and can be applied to indirect networks, such as tree-based networks with processors at the leaves, as well as direct networks, such as k-ary n-cubes. In particular, this thesis applies multi-server queueing models to obtain the first performance model for the wormhole routed butterfly fat-tree, a network with redundant communication links. This thesis also provides an analytical model for the 2D-mesh, a network that is popular in practice but has been overshadowed by the torus in wormhole routing performance models.

The last topic is a specific computational problem, labeling of connected components in an image, that is well suited to the particular network architecture of a scan line array processor. This thesis considers a new approach to finding connected components of an $N \times N$ image on a scan line array processor composed of N processing elements. This approach yields an algorithm guaranteed to complete in $O(N \lg N)$ time but that seems likely to approach $O(N)$ time for all or most images, whereas prior solutions usually require $\Omega(N \lg N)$ time or a more complicated architecture. Simulations on random images are consistent with the $O(N)$ time prediction.

Message Routing and Problem Solving in Multiprocessor Networks

by

Lee Guan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1997

Advisory Committee:

Assistant Professor Ronald I. Greenberg, Chairman/Advisor
Professor Joseph F. JáJá
Professor Evaggelos Geraniotis
Associate Professor Joel Saltz
Associate Professor William Gasarch

Dedication

To my wife Junying and my daughter Nancy

Acknowledgements

I would like to express my sincere appreciation to my advisor, Ronald I. Greenberg, for leading me into the field of parallel computing and providing excellent guidance throughout this work. Thanks also to Joseph F. JáJá, Evaggelos Geraniotis, Joel Saltz and William Gasarch for taking time to serve on my committee.

This thesis is partly supported by the NSF award CCR-9109550. Most parts of this thesis are based on joint works with Ronald I. Greenberg.

Thanks to Aris Christou and David F. Barbe for their support and guidance during the early years of my stay in College Park. Thanks also to many other faculty members at UMCP for providing excellent education, and to many friends and fellow students for their constant help and inspiration.

Finally, I would like to thank my entire family for their support and encouragement throughout this endeavor. Special thanks to my wife, Junying. Without her strong support this could not have been possible.

Table of Contents

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Interconnection Network Comparison 2

1.2 Performance Modeling 5

1.3 Problem Solving 7

1.4 Thesis Overview 7

I Interconnection Network Comparison 9

2 Interconnection Networks and Routing Protocols 10

2.1 Network Topologies 10

2.1.1 Bus and Ring Networks 11

2.1.2 Fully-connected Network and Crossbar 13

2.1.3 k-ary n-cubes 14

2.1.4 Fat-tree and Fat-pyramid 17

2.2 Routing Protocols 20

2.2.1 Packetization 21

2.2.2 Path Selection 22

2.2.3 Contention Resolutions 24

2.3 Synopsis 25

3 Equal-cost Comparisons of Interconnection Networks 26

3.1 Cost Measures 27

3.2 Channel Widths Under Equal-cost Constraints 28

3.2.1 Constant Bisection Width Constraint 29

3.2.2 Constant Pin-out Constraint 29

3.2.3 Constant Layout Area Constraint 31

3.3 Wormhole Routing Comparisons 33

3.4 Packet Routing Comparisons 38

3.5 Synopsis 41

II Wormhole Routing Performance Modeling 43

4 A Performance Model for Wormhole Routed Networks 44

4.1 Preliminaries 44

4.2 Performance Metrics 48

4.3 Average Latency 48

4.3.1 Service Time 51

4.3.2 Waiting Time 52

4.4 Throughput 59

4.5 Synopsis 59

5 Analysis of Butterfly Fat-Tree **61**

5.1 Message Arrival Rates 62

5.2 Waiting and Service Times 64

5.3 Average Latency 66

5.4 Throughput 66

5.5 Experimental Validation 67

5.6 Synopsis 68

6 Analysis of Mesh With Comparison to Torus **71**

6.1 Analysis of Meshes 72

6.1.1 Message Rates 74

6.1.2 Service Times 74

6.2 Comparison of Mesh and Torus Models to Simulations and to Each Other 77

6.3 Synopsis 79

III Problem Solving in Multiprocessor Networks **80**

7 Finding Connected Components on a Scan Line Array Processor **81**

7.1 Background 82

7.2 Greenberg's Algorithm 84

7.3 Simplified Algorithm and Simulations 91

7.4 Synopsis 95

8 Conclusions **97**

A Wire Density for Hypercube with Normal Layout **100**

B Performance Model for Wormhole Routed Torus **103**

Bibliography **108**

List of Tables

3.1 The bisection width with channel width 1 and the channel width to maintain constant bisection width across different networks. 30

3.2 The pin-out with channel width 1 and the channel width to maintain constant pin-out across different networks. 31

3.3 The wire density per row/column and the channel width under constant layout area constraints. 33

7.1 Mean number of FIND steps (T) and standard deviations (σ_T) at various image sizes (N). 95

List of Figures

2.1 A bus-oriented multiprocessor network. 11

2.2 A crossbar network. 13

2.3 A 4×4 mesh network. 15

2.4 A torus network. 16

2.5 A 16-processor hypercube. 17

2.6 A butterfly fat-tree. 18

2.7 A fat-pyramid. 20

2.8 Virtual channels in the ith dimension. 23

3.1 Throughput and latency comparison under constraint of equal bisection width. 36

3.2 Throughput and latency comparison under constraint of equal pin-out. 37

3.3 Throughput and latency comparison under constraint of equal interconnect area. 38

3.4 Comparison of packet routing latency under constraint of equal bisection width. 39

3.5 Comparison of packet routing latency under constraint of equal pin-out. 39

3.6 Comparison of packet routing latency under constraint of equal interconnect area. 40

4.1 A general routing model. A network consists of processing elements (PE) and routing elements (RE). A PE is attached to an RE through an injecting channel and an ejecting channel. . . . 49

4.2 Possible subsequent channels for incoming channel i 52

5.1 Butterfly Fat-Tree With 64 Processors 63

5.2 Comparisons for 64-processor BFT network for 16, 32 and 64-flit messages 68

5.3 Comparisons for 256-processor BFT network for 16, 32 and 64-flit messages 69

5.4 Comparisons for 1024-processor BFT network for 16, 32 and 64-flit messages 69

5.5 Comparison of maximum throughput for BFT networks with 64, 256, 1024 and 4096 processors for 32-flit messages 70

6.1 Comparison of model and simulation results for the mesh and torus with 64 nodes using message lengths of 20 and 32 flits. . 77

6.2 Comparison of model and simulation results for the mesh and torus with 256 nodes using message lengths of 32 and 64 flits. 78

7.1 The SLAP architecture. 83

7.2 The top-level procedure for the $O(N)$ component labeling algorithm. 85
7.3 The high-level structure of the left-connected component labeling algorithm. 86
7.4 Pseudocode for processor i in the union-find pass of the left-component labeling algorithm. 88
7.5 Pseudocode for APPLY called from UNION-FIND-PASS 89
7.6 Pseudocode for processor i in the labeling pass corresponding to line 3 of Algorithm LEFT-COMPONENTS. 90
7.7 The top-level procedure for the simplified component labeling algorithm. 92
7.8 Pseudocode for processor i in the adjusted union-find pass of the left-component labeling algorithm. 93
7.9 An example of input, output and intermediate results of the revised algorithm. 94

A.1 Normal layout for a hypercube. 101

Chapter 1

Introduction

Modern supercomputers use multiple processors to achieve increased computation power. The power of a single processor is limited by many factors. While advances in VLSI technologies continue, switching speed of logic gates may be getting closer to insurmountable limits. There is also a substantial software cost to using a single processor for applications that are inherently suited for parallel execution. An operating system may provide an emulated parallel environment by supporting multitasking through context switching among processes or through the use of threads, but these efforts take a share in the utilization of the CPU power. Therefore, the ultimate solution to the increasing demand for computational power relies on the use of multi-processor computers.

Many issues arise when designing multi-processor computers. First of all, an interconnection network is required to connect the collaborating processors. Proper routing protocols are then needed to efficiently transfer messages in the interconnection network. Furthermore, parallel operating systems are required to coordinate the activities of the processors, and algorithms suited for parallel executions of instructions are desired to best utilize the system resources. In addition to the technical issues, cost forms another important factor and usually plays a critical role in deciding among technological options. Despite the tremendous evolution in the field of supercomputing during the last two decades, many areas need improvements and many problems are yet to be solved.

This thesis explores issues in the choice of interconnection network and message routing scheme for general-purpose computers as well as algorithm design for a particular computational problem on a special-purpose parallel machine. More specifically, this thesis empirically compares the message routing capabilities of interconnection networks, develops analytical models for the performance of wormhole routed networks, and provides an improved algorithm for an important image processing problem utilizing a scan line array processor (SLAP).

1.1 Interconnection Network Comparison

There are various approaches to implementing the interconnections in a multiprocessor computer. Networks topologies from simple bus or ring oriented networks to complicated crossbars have been investigated. While specific real world applications may favor certain network architectures, it is the interconnection network for general purpose computers that we are primarily interested in. The first part of this thesis focuses on empirical comparisons between different interconnection networks.

Previous investigations on interconnection networks have been primarily based on two aspects, performance and cost. Commonly used performance metrics are average message latency and maximum throughput. As the grain size of concurrent computers continues to decrease, messages are sent more frequently, and fewer instructions are executed in response to each message. Communication latency becomes a critical factor for system performance. Throughput, the other important performance metric, measures the total number of messages the network can handle per unit time and is critical for applications that generate bursty message traffic. While the metrics for performance are relatively straightforward, it has been difficult to directly assess the cost of an interconnection network. Hardware parameters including bisection width [15, 33] and pin-out [1, 2] have been used instead to represent cost in practice.

Bisection width and pin-out are used for cost measures because they are both closely tied to the cost of VLSI implementations of interconnection graphs. Bisection width is of particular interest because its square provides a lower bound for the layout area of a graph [58]. More involved analysis of graph layout, however, can permit direct use of layout area as a more accurate measure for essentially two-dimensional VLSI technologies.

One trend in general-purpose parallel computation has been the development of *universal networks*. A universal network is one that is nearly as fast (with at most a polylogarithmic slowdown) for any computation when compared with any other network, thereby avoiding the cost of building application-specific networks. Previous research [31, 32, 44, 46] has shown that a fat-tree network is area-universal, i.e., that it is nearly as fast as any network of comparable layout area. Further, an augmentation of the fat-tree with hierarchical mesh connections, the fat-pyramid [24] network, has been shown to maintain area-universality under general conditions (without the usual simplifying assumption that unit time suffices to traverse any wire regardless of its length).

While asymptotic analysis has shown the fat-tree and fat-pyramid to be area-universal, it is desired to see how they compare with other networks in practice. This thesis performs such comparisons with the following goals:

- Compare the fat-tree and fat-pyramid with k-ary n-cubes using standard performance metrics.
- Compare area with bisection width and pin-out as measures of hardware cost.
- Investigate the effect of routing techniques, such as wormhole routing versus packet routing, on performance comparisons.

In this thesis, two networks that essentially correspond to the two extremes of the k-ary n-cube family, namely the 2D-mesh and the hypercube, are used for comparisons with the fat-tree and fat-pyramid. The comparisons are based on empirical simulations for random, uniformly distributed messages. In addition to providing much flexibility in manipulating network and routing parameters (routing strategy, packet/flit size, buffer size, etc), empirical simulation is particularly interesting since some empirical study on fat-trees [32] has already shown that it may be possible to achieve better performance in practice than is indicated by provable bounds.

1.2 Performance Modeling

Another important approach to network performance study is through analytical modeling using queueing theory. Compared with the empirical approach, analytical models are less flexible in terms of hardware configuration, message patterns, and routing protocols. Analytical models can, however, provide fast prediction of network performance under certain simplifying assumptions. Furthermore, analytical models are less prone to exhausting existing hardware resources (memory, CPU speed, etc). This is important since most simulations are time consuming and usually limited to networks up to a few thousand nodes in size. Analytical models are especially important for the study of the asymptotic behaviors of networks.

Interconnection network performance is generally determined by network topology, message traffic, and routing protocols. Network topology refers to the architectural aspects of a network, including node organization and number of available channels. Message traffic includes message arrival rates, message patterns, and distribution of channel service times. Routing protocols refer to the algorithmic decisions in transmitting messages, including packetization, path selection and contention resolution.

The key issues in performance modeling are the determination of waiting time and service time encountered at each stage on a message's path. Appropriate queueing principles are applied to resolve these issues. The utilization of the queueing models critically depends on the packetization technique. Store-and-forward routing (or packet routing) has been the subject of extensive modeling efforts due to is early origin, and results are available for many networks. These results, however, do not lead to a direct solution to the recently emerged *wormhole routing* technique, which is gaining rapid acceptance due to its ability to achieve much lower latency.

In contrast to store-and-forward routing, messages in wormhole routing are divided into small units called *flits* (short for flow control digits), the smallest unit of information that can be transmitted over a communication link. Only the first few flits of a message carry routing information; the rest of the flits follow the head flit through the network without being interrupted. When the head flit is blocked, the rest of the message is blocked in place. As a result, a message holds a communication channel for a indefinite length of time, thereby causing new problems in determining the service time and waiting time.

Several performance models have been presented for wormhole routing [15, 20, 33, 38, 39]. These models, however, are mainly concerned with a special class of networks, namely the k-ary n-cubes. A wide range of networks (especially the tree-based networks) has not been investigated. In addition, all existing models consider single-server channels; an important situation that calls for multiple-server channels (where redundant paths are present in the network) has not been addressed.

This thesis presents an extended performance model for wormhole routed networks. The model can be applied to indirect networks such as tree-based networks as well as direct networks such as k-ary n-cubes. Multi-server queueing for wormhole routing is analyzed and applied to the butterfly fat-tree for the first time, solving the problem caused by redundant links. The model is also applied to the important mesh network which has previously received incomplete attention. Model predictions are validated through comparison with empirical simulation for both the butterfly fat-tree and the mesh.

1.3 Problem Solving

Multiprocessor computing presents new challenges in best utilizing the system resources when solving real-world problems. A key issue is to properly integrate communication and local computation to solve the problem at hand. This thesis investigates such issue through a specific problem on a scan line array processor (SLAP) where the goal is to find a fast algorithm for labeling connected components. The problem has a variety of applications in image-processing. This thesis considers a new approach yielding an algorithm that is guaranteed to complete in $O(N \lg N)$ time and is likely to approach $O(N)$ time for all or most images. The best previous solutions require using a more complicated architecture or require $\Omega(N \lg N)$ time for most images. This thesis presents simulations of the new approach on random images that agree with the $O(N)$ time prediction.

1.4 Thesis Overview

This thesis contains three parts. The first part conducts performance comparison of interconnection networks using a variety of constraints on channel width as a reflection of cost. Chapter 2 introduces background knowledge about network topologies and routing protocols, which are the basis for this part as well as the entire thesis. Chapter 3 provides performance comparisons of the butterfly fat-tree, the fat-pyramid, the hypercube and the 2D-mesh networks under the equal-cost constraint and investigates the effect of packetization techniques on performance comparison.

The second part of this thesis is dedicated to performance modeling of wormhole routed interconnection networks. Chapter 4 presents an extended wormhole routing performance model that includes modeling for multi-server queues. Chapter 5 applies the wormhole routing model to the butterfly fat-tree network, using the multi-server model to deal with redundant channels. In Chapter 6 we apply the model to the 2D-mesh network and compare its performance with the torus network.

The last part of this thesis investigates the specific computational problem of connected component labeling on a SLAP in Chapter 7.

The final chapter of the thesis, Chapter 8, summarizes the conclusions reached from each part.

Most of the material in this thesis is based on material in the following references: [27, 28, 29, 30].

Part I

Interconnection Network Comparison

Chapter 2

Interconnection Networks and Routing Protocols

This chapter presents background material on interconnection networks and routing protocols. Section 2.1 discusses network topologies that fall into a wide spectrum from low-cost, low-bandwidth networks to high-cost, high bandwidth networks. Section 2.2 discuss important issues in routing protocols including packetization, path selection, and contention resolution techniques. The material presented in this section forms the foundation for the rest of this thesis.

2.1 Network Topologies

This section investigates the following leading candidates for multiprocessor systems:

- Bus and ring networks
- Fully-connected network and crossbar
- k-ary n-cubes

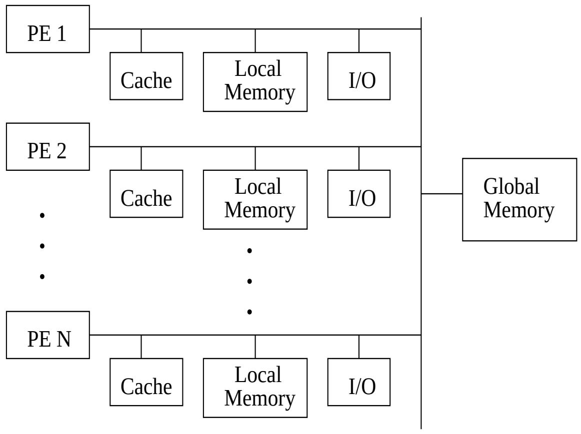

Figure 2.1: A bus-oriented multiprocessor network.

- Fat-tree and fat-pyramid

This is not an exhaustive list of possibilities. There are many other networks, such as the multistage Omega and shuffle exchange networks. The selected networks, however, are rather representative.

2.1.1 Bus and Ring Networks

The simplest way to construct a multiprocessor is to connect the processors on a shared bus that provides access to shared global memory for all processors. Figure 2.1 shows the block diagram of such a system. Each processor has access to a common bus to which a shared global memory for all processors is attached. In addition, each processor has its own local memory and cache. The local memory and cache enables the processors to reduce the use of the shared bus and thereby reduce the contention at the shared memory.

A fundamental bottleneck to bus-oriented multiprocessors is the interconnect technology for the common bus. The maximum clock speed of metalconductors are limited to a few hundred mega-hertz. As the number of processors tied to a bus increases, a processor has to share the bus with more other processors. In addition, most electrical buses also suffer from degradation due to the increased processor interfaces, which tends to reduce the bandwidth on the shared bus. Real-world commercial releases of bus-oriented multiprocessors have supported as many as 32-processors. Above 32 processors, the performance degradation due to bus contention becomes unacceptable.

A ring-network offers another low-cost solution to interconnecting processors. In a ring network, processors are connected by point-to-point connections in a cycle. By avoiding long buses, a ring network can achieve a higher clock speed. Furthermore, the simple interconnection structure makes it possible to use optical interconnects that yield giga-hertz clock speeds. To take the advantage of the ring interconnection, the ring network is operated like a pipeline with a short cycle-time and long-delay. The effective bandwidth can be utilized as long as the computations keep the pipeline filled. This implies each processor should overlap message transmissions with local computations. If the frequency of message transmission is high, performance degrades due to the large latency. This makes it less attractive for fine-grained computers.

As a note, if we break the ring at an arbitrary node, we get a network analogous to the structure of a scan line array processor (SLAP). We will discuss problem solving on a SLAP in Chapter 7 and illustrate a fast algorithm for solving connected components through proper overlapping of local computation and message passing.

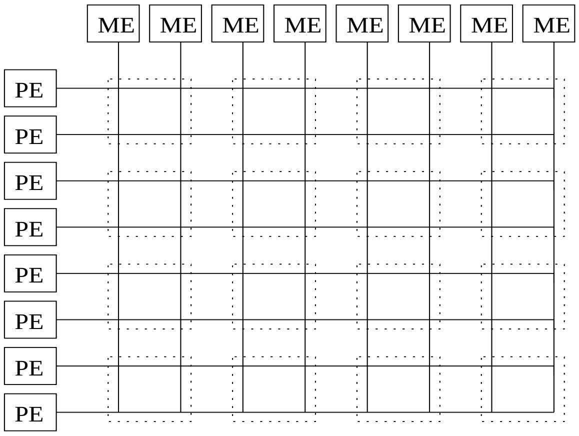

Figure 2.2: A crossbar network.

2.1.2 Fully-connected Network and Crossbar

From the latency point of view, the optimal interconnection network is a fully-connected network in which every processor has a direct connection to every other processor. No latency is incurred in switching in a fully-connected network. However, the degree (number of links incident on a node) of each node grows linearly with the size of the system, which has several implications: Firstly, we cannot build a very large network with bounded pin-out components. Secondly, a different processor is need for each different network size, causing difficulties for scaling. Furthermore, using techniques form [57] and [45], it can be shown that when implemented in a two-dimensional plane, the layout area will grow as $\Theta(N^4)$, where N is the number of processors in the system, therefore making the wiring distance grow as $\Theta(N^2)$. Since the latency is determined by the wiring distance, such a network is not very practical.

Figure 2.2 shows another low-latency network, the crossbar network. Any processor can directly access any memory with a unit delay only at the crosspoint. The communication network has no contentions, contention only occurs at memories when multiple processors accesses the same memory at the same time. Unfortunately, VLSI pin-out limitations will not allow a single crossbar of arbitrary size. Therefore we have to partition the functions among $(m \times m)$ smaller components (dashed lines), where m is the number of components in each dimension for a 2-dimensional layout. This would incur $O(m)$ switching latency and require $O(m^2)$ such switches. In practice switching components tend to take constant size due to both pin-out limitations and scalability considerations. This implies m would need to grow as $\Theta(N)$, making both latency and cost unacceptable.

We have just looked at the two extremes of interconnection networks in terms of the cost-bandwidth trade-off. The bus and ring represent the low-cost, low-bandwidth end of the spectrum, where performance degradation makes them unsuitable for large systems. The fully-connected network and crossbar represent the high-cost, high-bandwidth end of the spectrum, which are subject to high-cost as well as physical packaging limitations. It is the interconnection networks in the middle of the spectrum that generally strike the best balance. We will next look at representatives of such networks.

2.1.3 k-ary n-cubes

The k-ary n-cube networks are described using a *dimension* n and a *radix* k. Dimension and radix are related to the number of nodes N by the equation

$$N = k^n \; . \tag{2.1}$$

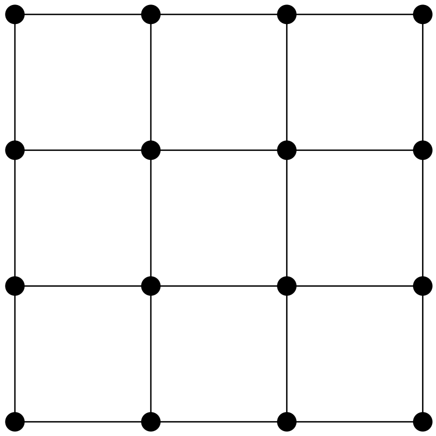

Figure 2.3: A 4×4 mesh network.

A node in the k-ary n-cube can be identified by an n-digit radix k address, $\langle a_0, \cdots, a_{n-1} \rangle$. The ith digit of the address, a_i, represent the node's position in the ith dimension ($0 \le a_i < k$). The usual definition of a k-ary n-cube is that there is a directed link from each node $\langle a_0, \cdots, a_i, \cdots, a_{n-1} \rangle$ to node $\langle a_0, \cdots, (a_i \pm 1) \bmod k, \cdots, a_{n-1} \rangle$ for each dimension i. The ring, torus, and hypercube are important members of the k-ary n-cube family, and the mesh is closely related to the torus. We have already discussed the ring as a low-cost, low-bandwidth network, we will focus on the mesh, torus and hypercube in the following discussions. There are other members of the k-ary n-cubes, but the selected networks are especially interesting for this thesis and are rather representative of the k-ary n-cubes under general conditions.

Figure 2.3 shows a 4×4 mesh network; each link in this illustration is bidirectional. Mesh interconnections have excellent characteristics for numerical problems that arise in scientific contexts. The combination of low-cost and high speed for near neighbor communications makes such connections quite attractive for implementation. The most serious disadvantage is that they

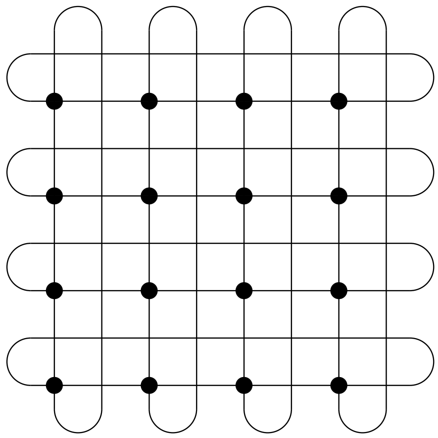

Figure 2.4: A torus network.

don't support global communication and synchronization directly. The overall speed of a parallel calculation on a mesh-based structure will depend on the proportion of global operations that have to be performed. If a mesh structure is supported by a second interconnection structure for global operations (a bus, for example [36]), the two structure together can provide a computer system that is well suited for a broad class of scientific applications.

A close relative of the mesh network is the torus. Figure 2.4 shows the 4×4 torus network (4-ary 2-cube). The torus differs from the mesh in that there are wrap-around links at the edges, making it symmetric. Also, unidirectional links in the torus suffice to provide a path from any node to any other. The torus has been used as an element to build networks of higher dimensions [12]. A new problem arises for the torus when wormhole routing is used; deadlock may occur due to the wrap-around links. Section 2.2 discusses techniques to avoid deadlocks.

The hypercube is the high-dimensionality extreme of the k-ary n-cube family. Figure 2.5 shows a 16-processor hypercube; interpreting each link

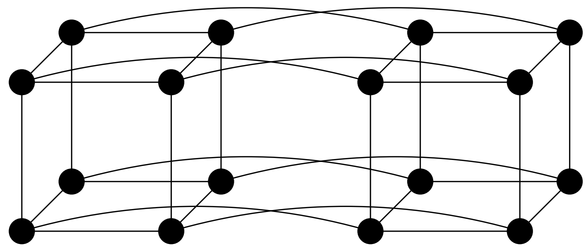

Figure 2.5: A 16-processor hypercube.

in this illustration as bidirectional is actually consistent with the earlier definition of k-ary n-cubes in terms of unidirectional links. The message distance is $\log_2 N$ since at most one link is traversed in each dimension. Unfortunately, to maintain this characteristic, the node degree grows as $\Theta(\log N)$. Node size soon runs into the pin-out limitation and a different size node is needed for each different system size. Additionally, when implemented in a two-dimensional plane, the interconnection requirement cause the system area to grow as $\Theta(N^2)$; therefore the message distance will actually grow as $\Theta(N)$.

2.1.4 Fat-tree and Fat-pyramid

The basic fat-tree network was introduced by Leiserson [46] based on Leighton's tree-of-meshes graph [43]. A set of N processors are located at the leaves of a complete binary tree. Each edge represents bidirectional links: one form parent to child and one from child to parent. Unlike a normal tree where the channel widths are constant in all levels, as the level increases from the leaves the channel capacity increases also. The idea of a fat-tree is to take advantage of a principle of locality in much the same way as does the telephone network: by using only slightly more hardware than that required to

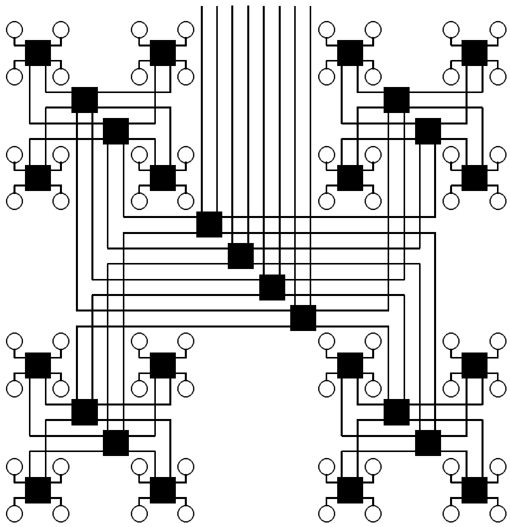

Figure 2.6: A butterfly fat-tree.

support *nonlocal* communications among a set of processors, much additional *local* communication among a larger set of processors can be supported.

Fat-trees can be built using switches with a constant degree as shown in Figure 2.6, where processors are represented by circles and switches are represented by squares. Since these switches are reminiscent of those in the "butterfly" network, such a fat-tree is referred as a butterfly fat-tree. A precise description of the butterfly fat-tree can be given as follow: Denote each node using a pair of indices (l, a), where l represents the level (distance from the leaves) of the node in the butterfly fat-tree and a represents the address of the node in that level. Each switch $S(l, a)$ has six ports: $parent_0$, $parent_1$, $child_0$, $child_1$, $child_2$ and $child_3$. At the lowest level ($l = 0$) are the N processors with addresses 0 to $N - 1$. These processors are connected to $N/4$ switches at level 1 such that the processor at $(0, a)$ is connected to the $child_{a \bmod 4}$ of $S(1, \lfloor a/4 \rfloor)$. At the l-th level ($l = 2$ to $\log_4 N$), there are $N/2^{l+1}$ switches.

The remaining connections are that $parent_0$ of $S(l,a)$ is connected to $child_i$ of $S(l+1, \lfloor \frac{a}{2^{l+1}} \rfloor \cdot 2^l + a \bmod 2^l)$, and $parent_1$ of $S(l,a)$ is connected to $child_i$ of $S(l+1, \lfloor \frac{a}{2^{l+1}} \rfloor \cdot 2^l + (a + 2^{l-1}) \bmod 2^l)$, where $i = \lfloor \frac{a \bmod (2^{l+1})}{2^l - 1} \rfloor$.

Leiserson [46] showed that fat trees can perform any communication performed by any other similarly sized (volume) network, and Greenberg and Leiserson [31] showed the simulation can be effectively performed on line. These analyses assume unit wire latency regardless of the length of the interconnections. Greenberg [24] presented an augmentation of the fat-tree network, the fat-pyramid, and shows that the fat-pyramid is universal without the assumption of unit wire latency.

The structure of the fat-pyramid is shown in Figure 2.7 from [24]. The figure shown is a simple version of the fat-pyramid; using the technique described in [24, Section II], processors can be packed a bit more densely. A precise description of the interconnection pattern for the switches in the fat-pyramid can be given as follows. (For simplicity, labels for the different ports of a switch are not specified here.) We begin with a collection of ordinary two-dimensional meshes at levels $0, 1, \ldots, \log_2 \sqrt{N/4}$ representing distance from the leaves in the underlying tree structure. At level h, there are 2^h copies of a $\sqrt{N/4}/2^h \times \sqrt{N/4}/2^h$ mesh. We then denote a switch by an ordered 4-tuple (h, c, x, y), where h is the level, c is the copy number of the mesh $(0 \leq c < 2^h)$ at this level that contains the switch, and x and y specify a mesh position in an ordinary Cartesian coordinate system $(0 \leq x, y < \sqrt{N/4}/2^h)$. Then for $0 \leq h < \log_2 \sqrt{N/4}$, switch (h, c, x, y) is connected to $(h+1, 2c, \lfloor x/2 \rfloor, \lfloor y/2 \rfloor)$ and $(h+1, 2c+1, \lfloor x/2 \rfloor, \lfloor y/2 \rfloor)$.

Greenberg [24] has shown that a fat-pyramid of area $\Theta(A)$ requires only

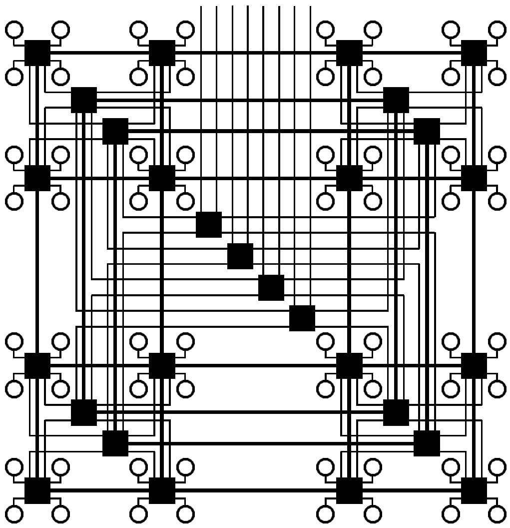

Figure 2.7: A fat-pyramid.

$O(\log A)$ slowdown to simulate any competing network of area A under very general conditions. The result holds regardless of the processor size (amount of memory) and number of processors in the competing networks as long as the limitation of total area is met. Furthermore, the result has been shown valid regardless of the relationship between wire length and wire delay.

2.2 Routing Protocols

This section discuss message *routing protocols*. Routing protocols generally refer to the techniques for packetization, path selection and contention resolution. We will discuss each of these issues as follows.

2.2.1 Packetization

Packetization is the process of packing messages into transmittable units. Messages from processors contains various sizes, the switching hardware, however, is usually designed to handle packets with smaller sizes (compared with the messages) to reduce latency. Various packetization techniques have been used in practice, among which the two most interesting techniques are store-and-forward (packet) routing and wormhole routing.

The traditional store-and-forward routing technique breaks messages into units called *packets*. Each packet contains the necessary routing information to find its destination. In a dynamic switching network, packets of a message may follow different paths such that the packets received at the destination may be out of sequence. The destination processor has to re-assemble the received packets according to the sequence number carried in each packet.

An alternative packetization technique, wormhole routing [18], has recently emerged and been adopted in a number of parallel systems [9, 12, 37]. Wormhole routing shows its advantage over packet routing for the following reasons. First of all wormhole routing achieves lower switching latency. This is because the latency in packet routing is determined by the product of the packet size and the distance of the message, while in wormhole routing the latency is the sum of the message size and the distance. Wormhole routing works in a pipelined fashion and therefore reduces the latency. In addition to achieving lower latency, wormhole routing requires only a small queue size at the switching elements to hold a few flits, while packet routing generally requires much bigger queue size to hold an entire packet. Wormhole routing therefore requires smaller switching devices, which reduces the size of the system. As the system size reduces, higher clock speed is made possible.

A potential problem for wormhole routing, however, is its blocking property. When the head of a worm is blocked due to contention, the whole message is blocked in place. This causes deadlock if a circular-wait situation occurs. Additional techniques are needed to avoid deadlocks when wormhole routing is used.

2.2.2 Path Selection

Resolving paths for messages depends on the topology of a network as well as the packetization technology. In store-and-forward routing, the major issues are the load-balancing of traffic on the interconnection network and the minimization of path-distances. In wormhole routing networks, deadlock may occur and deadlock avoidance must be taken into consideration in path selection.

Deadlock avoidance is especially important for k-ary n-cubes due to the existence of circular loops in this class of networks. In a wormhole routed hypercube, deadlock may be avoided solely by using a technique called *e-cube* routing [18]. In e-cube routing, worms route through dimensions in the order specified by scanning the address bits from MSB to LSB. For a mesh it also suffices to route first in one dimension and then in the other. This deadlock avoidance strategy also applies to the hierarchical mesh networks in the fat-pyramid network.

In other k-ary n-cube networks (including the torus) e-cube routing alone

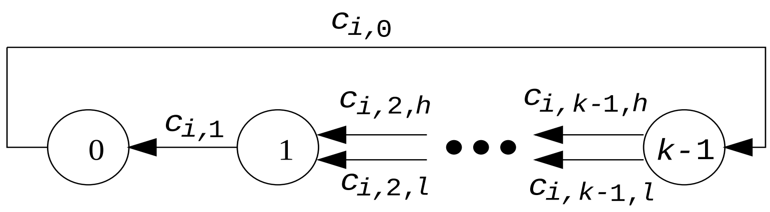

Figure 2.8: Virtual channels in the ith dimension.

is not sufficient to avoid deadlock. Dally [16, 17] has shown that a combination of e-cube routing and another technique called *virtual channel flow control* can avoid deadlock on such networks. In virtual channel flow control all but two channels in a unidimensional cycle are broken into pairs of virtual channels (Figure 2.8). A virtual channel pair is implemented by time-multiplexing onto the associated physical channel. When a worm enters the ring in the ith dimension either from a node in the dimension or form another dimension, the messages are routed along the high virtual channels. Upon traversing $c_{i,0}$, messages switch to low channels for continued routing in that dimension. By breaking the cyclic path in each dimension, virtual channel flow control eliminates the possibility of deadlocks.

Routing on the butterfly fat-tree is free of the deadlock concern. A message goes up the tree until it reaches a switch such that the destination is a child of that switch. When going upward, there are two redundant links to choose. The upward channel is selected following certain algorithms (Fixed-path, random-path or greedy-path selection). The downward path is unique. A message is routed to the ith child port from a switch at level l, where $i = \left\lfloor \frac{dest\text{-}addr \bmod 4^l}{4^l - 1} \right\rfloor$.

Messages in the fat-pyramid are routed over the same paths as in the fat-tree, except that we allow each message to take one shortcut via one or two of the new hierarchical mesh edges. More precisely, the routing path is formed by going up fat-tree edges towards a randomly selected switch at the root of the underlying 4-ary tree until a switch is reached that is adjacent horizontally, vertically, or diagonally in the mesh at that level to a switch from which the destination can be reached by going down fat-tree edges. When wormhole routing is used, the fat-pyramid is subject to deadlock concerns and deadlock avoidance methods for the mesh may be applied.

2.2.3 Contention Resolutions

Another issue in routing is contention resolution. When multiple incoming channels on a switch contend for the same outgoing channel, the switch has to use some algorithm to choose which one to serve. The algorithms used for contention resolution include:

- Fixed-order scan: At each time step a switch scans its incoming links in a fixed order and choose the first one that has a message.
- Random round-robin (RR): Similar to fixed order, except that a switches selects the first incoming link randomly and scans around from that link.
- First-come-first-serve (FCFS): Messages are served according to the order of arrivals.

FCFS guarantee fairness among incoming links and avoids extremely long waiting time for certain incoming channel. FCFS, however requires additional hardware to time-stamp messages arriving at each incoming channel. While fixed-order scan and random round-robin do not guarantee bounded waiting time, they have the advantage of simplicity in that they don't need to scan through all ports to determine the earliest one. In previous research [32] it was found that random random-robin outperforms fixed order and achieves close or slightly better performance than FCFS when proper path selection is followed.

2.3 Synopsis

We reviewed several representative interconnection network topologies over a wide spectrum in the cost-performance tradeoff. Low-cost low-performance and high-cost high-performance networks are generally not suited for massively parallel systems. For the rest of this thesis we will concentrate on the networks in the middle of the spectrum, namely the mesh, torus, and hypercube (which are essentially k-ary n-cubes), and the fat-tree and fat-pyramid. We also reviewed routing issues including packetization, path selection and contention resolution. Routing protocols discussed in this chapter will be used in the rest of this thesis.

Chapter 3

Equal-cost Comparisons of Interconnection Networks

This chapter compares the communication capabilities of two area-universal networks, the fat-tree and the fat-pyramid, to two representatives of the k-ary n-cube family, namely the mesh and the hypercube. The mesh represents the low-cost, low-bandwidth end of the k-ary n-cube family, while the hypercube represents the high-cost, high-bandwidth end. (In fact, we will see in Chapter 6 that the mesh is probably superior to the closely related torus, which is the actual k-ary 2-cube.) The comparisons are based on equal hardware cost using bisection width, pin-out, and area as cost measures. We compare networks with the same number of processors. To enforce equal hardware cost, channel width (the number of wires connecting adjacent nodes) is adjusted to achieve equal bandwidth, pin-out, or layout area. In addition to identifying the relative performance among different networks, the comparisons also investigate using area as a cost measure, versus bisection width or pin-out. Furthermore, the effect of packetization on interconnection network comparisons is studied using wormhole routing and packet routing.

The rest of this chapter is organized as follows. Section 3.1 describes hardware measures for cost, including bisection width, pin-out, and layout area. Section 3.2 describes the available channel widths under each cost constraint. Section 3.3 and Section 3.4 give the simulation results under wormhole routing and packet routing respectively. Section 3.5 summarizes this chapter.

3.1 Cost Measures

The cost of an interconnection network depends on the technologies for the switching devices and the interconnections at the various levels of packaging and is usually difficult to directly assess. In designing multiprocessor networks hardware parameters are used instead to estimate the cost. Dally [15] used bisection width (the minimum number of wires cut when the network is divided into two equal halves) as a cost measure to compare k-ary n-cubes. The channel widths of networks built on the same number of processors are adjusted to equalize bisection width. His comparison under the constant bisection width constraint favored low-dimensional networks (closer to the mesh than the hypercube). Abraham and Padmanabhan [1] used pin-out, the total number of wires emanating from the processors and switches in the network, as the cost measure for the same networks (with some difference in the routing model). Their comparison under the constant pin-out constraint found higher dimensionality to be more important than wider channel width.

Bisection width and pin-out both model cost through their relationship to the area required for the network. In the case of bisection width, Thompson [57, 58] showed that area is lower bounded by a constant times the square of bisection width. The focus on pin-out reflects the fact that the connectors and drivers for external connections of a chip are much larger than internal wires, and chips are generally fabricated within standard packages, based on the number of pins.

In this thesis, we also investigate using area directly as cost measure and compare the results with those using bisection width and pin-out. To obtain the area for an interconnection network, we assume all circuits are placed in a 2D-plane. The 2D-area model is a simplified model for the cost. Real world interconnections involve hierarchical packaging architectures. It is, however, at least possible to perform similar analyses if a uniform 3D packaging technology becomes available, e.g., see [15, 23, 46].

3.2 Channel Widths Under Equal-cost Constraints

Whenever we compare different networks, we use the same number of processors in each. To equalize hardware cost, the number of wires for each channel (channel width) is adjusted according to the appropriate cost constraints. For example, when we use bisection width as the cost measure, we enforce a constraint that all networks have the same number of wires crossing a bisection. Therefore, networks with more channels crossing a bisection (for example, the hypercube) have to reduce the number of wires in each channel to make the bisection width constant across different network. In this section we obtain the available channel widths for mesh, hypercube, fat-tree, and fat-pyramid under the constraints of bisection width, pin-out, and area. The results are used in Sections 3.3 and 3.4 to compare the performances of the networks under the condition of equal cost.

3.2.1 Constant Bisection Width Constraint

The bisection widths in terms of number of channels (B) for the mesh, hypercube, butterfly fat-tree, and fat-pyramid are $\sqrt{N}$, $N/2$, $\sqrt{N}$ and $\sqrt{N}\log_{16} 4N$, respectively. (The least obvious result is the last, for which we add the number of channels to bisect the underlying fat-tree as well as each of the 2^h meshes at level h: $\sqrt{N} + \sum_{h=0}^{\log_2 \sqrt{N/4}-1} 2^h(\sqrt{N/4}/2^h)$.) The corresponding channel widths (W) for a range of values of N are listed in Table 3.1. In each row, channel widths are given to equalize the bisection width across different networks. For convenience, the channel width for the mesh is fixed at 32, and the other channel widths are rounded to the nearest integer to approximately equalize bisection width. (Channel width of 32 for the mesh is a reasonable value, since this would correspond to up to 128 connections per processor.)

3.2.2 Constant Pin-out Constraint

The pin-out of a node is defined as the degree times the channel width W. For a k-ary n-cube, to maintain constant pin-out, a decrease in the dimensionality k mandates an increase in channel width W. The butterfly fat-tree and the fat-pyramid networks have different pin-outs for processor and switch nodes, but we can use the total pin-out, the sum of the pin-outs over all nodes.

N	*mesh*		*hypercube*		*BFT*		*fat-pyramid*	
	B	W	B	W	B	W	B	W
16	4	32	8	16	4	32	6	21
64	8	32	32	8	8	32	16	16
256	16	32	128	4	16	32	40	13
1024	32	32	512	2	32	32	96	11
4096	64	32	2048	1	64	32	224	9

Table 3.1: The bisection width with channel width 1 and the channel width to maintain constant bisection width across different networks.

The total pin-outs for N-processor networks in units of channel width are $4(N-\sqrt{N})$, $N\log_2 N$, $(N+3(N-\sqrt{N}))$ and $(N+5(N-\sqrt{N})-\sqrt{N}\log_2 N)$ for the mesh, hypercube, butterfly fat-tree and fat-pyramid networks respectively. The total pin-out for the fat-tree is obtained by adding the pin-out of the processors to 6 times the sum of the number of switches at all the levels: $N+6\sum_{h=0}^{\log_2\sqrt{N/4}} 2^h(\frac{N}{4}/2^{2h})$. For the fat-pyramid, we must add an additional 4 connections for each switch but subtract off the missing connections at mesh edges: $N+10\sum_{h=0}^{\log_2\sqrt{N/4}} 2^h(\frac{N}{4}/2^{2h}) - 4\sum_{h=0}^{\log_2\sqrt{N/4}} 2^h(\sqrt{N/4}/2^h)$. The number of pin-outs (PO) as described above are listed in Table 3.2 for a range of values of N. In each row, channel widths (W) are given to equalize the total pin-out across different networks.

N	*mesh*		*hypercube*		*BFT*		*fat-pyramid*	
	PO	*W*	*PO*	*W*	*PO*	*W*	*PO*	*W*
16	48	32	64	24	52	30	60	26
64	224	32	384	19	232	31	296	24
256	960	32	2048	15	976	31	1328	23
1024	3968	32	10240	12	4000	32	5664	22
4096	16128	32	49152	10	16192	32	23488	22

Table 3.2: The pin-out with channel width 1 and the channel width to maintain constant pin-out across different networks.

3.2.3 Constant Layout Area Constraint

Area is another constraint of interest. With the layout area constraint, different networks are laid out on the same area with the same number of processors. The VLSI layout area is evaluated based on the assumption that all processors and switches are placed on a 2-D substrate. The substrate has two layer of interconnects for x-direction and y-direction respectively. Such a model can serve as a good abstraction for a variety of VLSI packaging technologies, such as wafer-scale integration or printed circuit boards [5]. More accurate modeling may require considering a complicated hierarchy of interconnection that consists of many levels, but the model here captures much of the complexity of interconnection, with more accuracy at least than looking only at bisection width.

The layout area of a network includes the area for the processors and the area required due to interconnections. Since the area required for the processors is the same, we remove it from consideration and examine only the area necessary to achieve the interconnections. If processor area is very large relative to the wire pitch, it simply means we can scale up the channel widths of all the networks by a substantial factor.

For each of the networks, we can analyze the area by expressing the side length with N processors as

$$S(N) = \sqrt{N} \cdot d \cdot W \cdot P \tag{3.1}$$

where P is the wiring pitch (constant throughout our analyses) and d can be thought of as an average wire density per row or column (with channel width 1) when the processors are laid out in a $\sqrt{N}$ by $\sqrt{N}$ grid.

For the mesh $d = 1$. For the hypercube, we can use a layout as in [54], which is known to have optimal area. The maximum wire density per row is $d = \frac{2^{p+2}-(-1)^p-3}{6}$, where $p = \log_4 N$. This result for wire density is explained fully in Appendix A.

For the butterfly fat-tree and the fat-pyramid, we can obtain the side length from recurrence relations based on Figure 2.7. For the butterfly fat-tree, the recurrence is

$$S(N) = 2 \cdot S(N/4) + \sqrt{N} \cdot W \cdot P \tag{3.2}$$

with $S(4) = 2W \cdot P$. The solution is

$$S(N) = \sqrt{N} \log_4 N \cdot W \cdot P \tag{3.3}$$

so $d = \log_4 N$ for the butterfly fat-tree. Similarly, for the fat-pyramid $d = \frac{3}{2} \log_4 N$.

N	*mesh*		*hypercube*		*BFT*		*fat-pyramid*	
	d	W	d	W	d	W	d	W
16	1	32	2	16	2	16	3	10
64	1	32	5	6	3	10	4.5	7
256	1	32	10	3	4	8	6	5
1024	1	32	21	2	5	6	7.5	4
4096	1	32	42	1	6	5	9	4

Table 3.3: The wire density per row/column and the channel width under constant layout area constraints.

The wire densities as described above are listed in Table 3.3 for a range of values of N. In each row, channel widths (W) are given to equalize the area across different networks.

3.3 Wormhole Routing Comparisons

In this section we investigate the relative performances of the mesh, hypercube, fat-tree, and fat-pyramid under a wormhole routing strategy. Bisection width, pin-out, and area are used as constraints to equalize cost of different networks with the same number of processors. The comparisons are based on results from empirical simulations, assuming bidirectional links for the networks under study.

It has been customary to use network latency as the primary performance measure because of its tendency to limit performance in practice in today's fine-grained parallel systems. The maximum throughput is another important performance metric, and for certain applications such as sample sorting can be dominant [33]. The actual relationship of the execution time to the maximum throughput and average latency is determined by the nature of a parallel algorithm, and is beyond the scope of this thesis. We will investigate both maximum throughput and average low load-rate latency for the networks under study.

The maximum throughput of a network is the maximum number of messages that can come out of each node in the network per clock cycle. Throughput varies with load-rate, which is the frequency with which processors generate messages. Throughput and load-rate are sometimes measured in fractions of the capacity per node of a network, where the capacity is defined as the maximum number of messages that can be in a network at one time. It is appropriate and convenient to use capacity as measures of throughput when comparing the performance of the same network under different operating conditions, or when comparing networks with the same capacities. However, since we are dealing with networks with different capacities, the actual number of bits per cycle per node is used as the basis for measuring load-rate and throughput.

As load-rate increases from zero, throughput follows load-rate until the load-rate has reached a certain value, when throughput starts to saturate. Additional increase in load-rate does not further increase throughput. As network size grows, the total throughput goes up, but the throughput per node generally decreases. For the mesh, butterfly fat-tree and fat-pyramid, this is mainly caused by the bisection width limitation, i.e., the bisection width doesn't scale up at the same ratio as does the number of processors. The hypercube network doesn't have bisection width limitation. However, when subject to the constraints (bisection width, pin-out, or area) to maintain equal cost, the hypercube network has to further reduce the width of each channel as network size grows. Moreover, the blocking of available channels by messages in the network also contributes to the decrease of the per node throughout.

The average latency, however, stays rather constant at low load-rates. As load-rate increases, average latency increase slowly until the network saturates, when the average latency increases rapidly due to the queuing time. In practice, parallel networks should be designed to operate on the the flat portion of the latency curve.

Figure 3.1 through Figure 3.3 show the simulation results for latency under constant bisection width, constant pin-out and constant area constraints respectively. Maximum throughput can be read from the latency graphs by looking for the load-rate at which the network saturates. The results are obtained from network simulators for uniform random message patterns, using the channel widths calculated in Section 3.2. A worm length of 320 bits is used for all networks. Simulations are shown for several values of N up to $N = 4096$.

Under the constant bisection width constraint, the mesh network always has the best maximum throughput. The hypercube network has the least throughput and the largest latency (in the graphs with 64 processors and up). This agrees with Dally's findings that the bisection width constraint strongly favors low-dimensionality [15]. The butterfly fat-tree network tends to achieve the lowest average latency for large network size (≥ 256). The fat-pyramid also tends to achieve lower latency than the mesh when the network size is

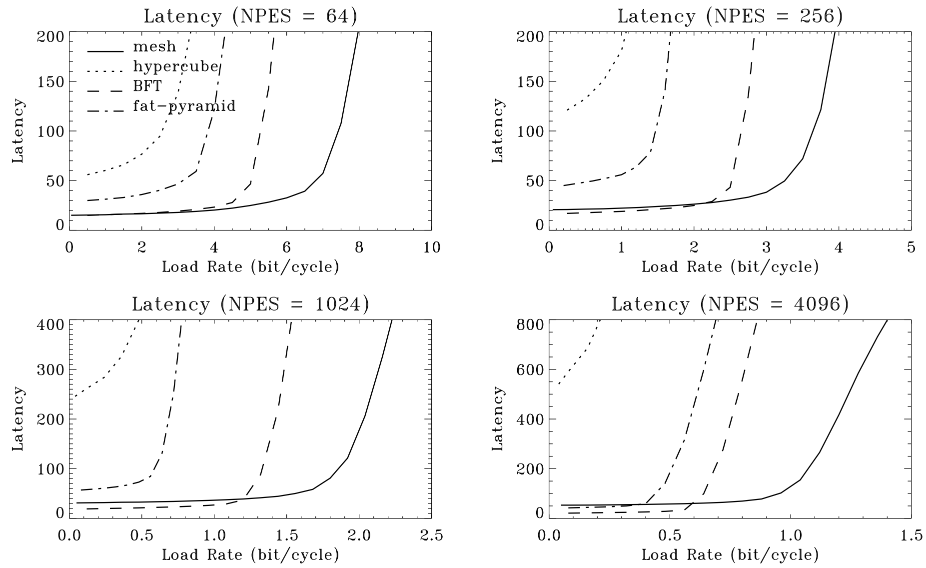

Figure 3.1: Throughput and latency comparison under constraint of equal bisection width.

as large as 4096. The fat-pyramid network has lower throughout and higher latency compared with the butterfly fat-tree because of its narrower channel width. Its performance is not far behind that of the fat-tree, but it remains poorer than the mesh for moderate network size. (Its potential advantage is in the scenario of non-unit wire delay, which is not simulated here.)

When the constant pin-out constraint is used, the hypercube network shows average latency close to that of the mesh and fat-pyramid. The hypercube network also has much higher throughput than other networks in agreement with Abraham and Padmanabhan's findings that constant pin-out favors high-dimensionality networks [1]. The butterfly fat-tree again shows the lowest latency for $N \geq 1024$. The fat-pyramid also achieves lower latency than mesh and hypercube when network size is as large as 4096. However both

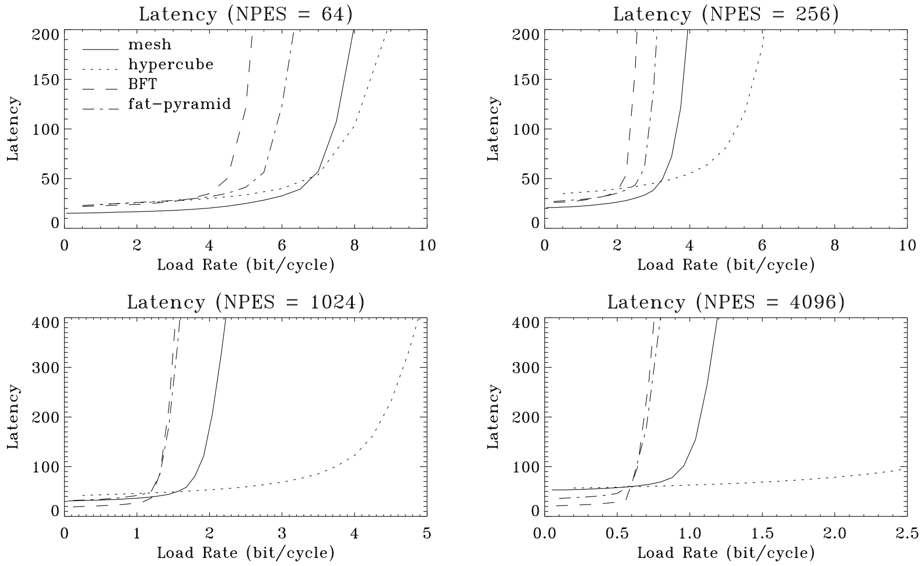

Figure 3.2: Throughput and latency comparison under constraint of equal pin-out.

the fat-tree and the fat-pyramid saturate at earlier load-rates than the mesh and the hypercube. (There is some disagreement with [1] in the relative low-load latencies of the mesh and the hypercube; the mesh shows lower low-load latency than the hypercube here due to the choice of a different routing model and other parameters.)

With the constant area constraint, the relative performance of the mesh and the hypercube is close to that under the constraint of constant bisection width. The butterfly fat-tree and the fat-pyramid networks show very close throughput, both small compared with those of the hypercube and mesh. The mesh network has the lowest average latency. The butterfly fat-tree and fat-pyramid have latencies between the mesh and the hypercube for network sizes of 256 and up. The poor showing of the fat-tree and fat-pyramid networks is

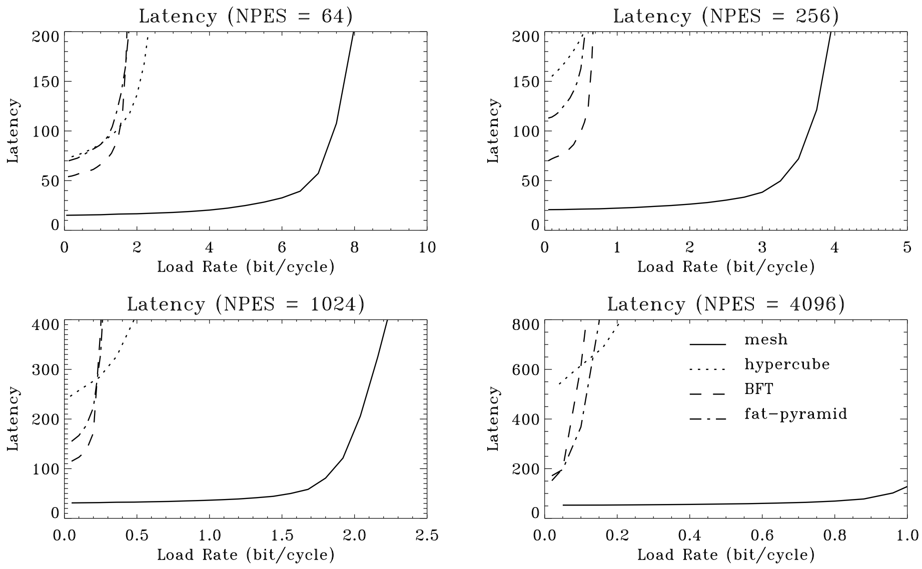

Figure 3.3: Throughput and latency comparison under constraint of equal interconnect area.

probably substantially attributable to the use of the simple, basic structure illustrated in Figure 2.7, which use area a little larger than linear in the number of processors. Further study would be desirable using better variants of these networks that achieve linear area as discussed in [24, Section II].

3.4 Packet Routing Comparisons

This section presents results under packet routing. We use the same networks and channel widths as used for the wormhole routing comparisons. Figures 3.4 through 3.6 show packet routing simulation results under constant bisection width, constant pin-out, and constant area constraints, respectively. Simulations are shown for several values of N up to $N = 4096$. Message lengths (packet size) of 320 bits are used throughout.

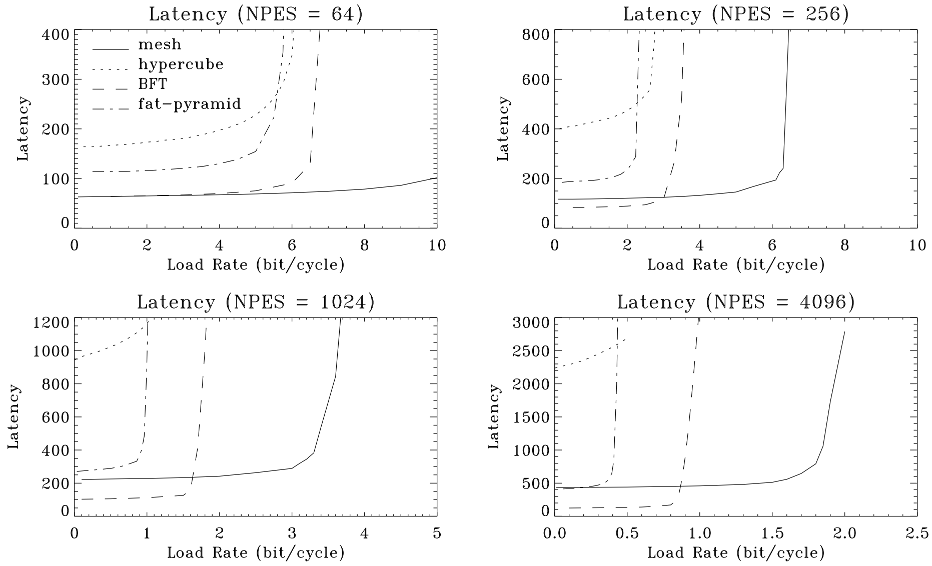

Figure 3.4: Comparison of packet routing latency under constraint of equal bisection width.

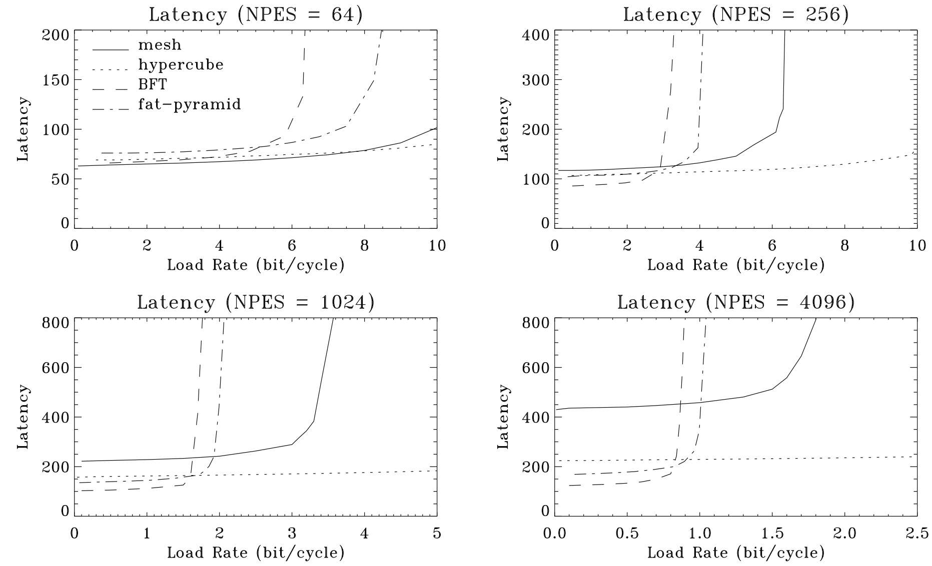

Figure 3.5: Comparison of packet routing latency under constraint of equal pin-out.

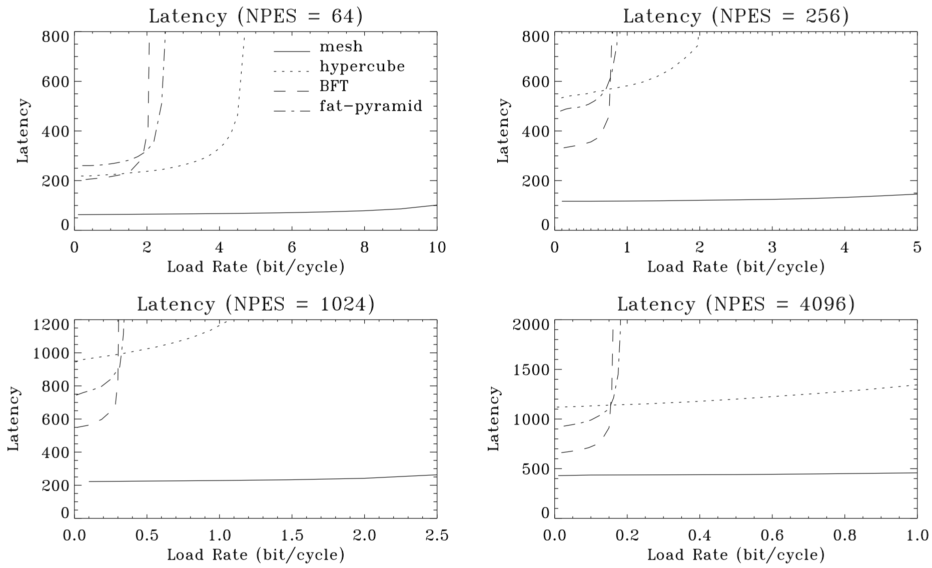

Figure 3.6: Comparison of packet routing latency under constraint of equal interconnect area.

It is interesting to see that for the most part, the packet routing graphs look qualitatively very similar to those obtained from wormhole routing with a reasonable range of worm lengths. Only in the case of constant pin-out did the choice of packet routing versus wormhole routing cause some change in the ranking of networks by low-load latency. However, packet-routing results tend to show much higher average latencies than wormhole routing, which was as expected. Additionally, packet routing results also show higher maximum throughput than wormhole routing results. Wormhole routed networks saturate at earlier load-rates because of their blocking properties.

3.5 Synopsis

This chapter investigated the performance of the area-universal networks, the fat-tree and the fat-pyramid, and two representatives of k-ary n-cubes, the mesh and the hypercube under different cost constraints and routing strategies. A summary of the results is given as follows.

Through the comparisons of the networks we found that under constant bisection width and constant pin-out constraints the butterfly fat-tree achieves lower latency for $N \geq 1024$. The results obtained through simulation of random message pattern are more optimistic than the theoretical worst-case upper bounds on the slowdown for a fat-tree to simulate another network [32, Section 3.1.1]. In addition, the butterfly fat-tree and fat-pyramid show close performance to each other under the unit wire delay model, but with a little poorer showing by the fat-pyramid. Since the purpose for introducing the fat-pyramid was to achieve favorable performance in comparison to other networks in the context of non-unit wire delay, additional empirical simulations should be performed with appropriate non-unit wire delay models.

Our results also show that the relative performances for the mesh and hypercube under the constant area constraint is close to that under constant-bisection width. Surprisingly, however, using the area constraint seems to be less favorable to the butterfly fat-tree and the fat-pyramid. Layout area should be a more accurate measure than bisection width, and further study is called for using the more area-efficient versions of the fat-tree and fat-pyramid in [24] rather than just the simplified versions of the network considered so far. In addition, the area model is still a simplification of real packaging constraints.

It would be desirable to more closely model this hierarchy of interconnection technologies.

When packet routing is used on the networks, our result show that the relative performance among the networks under study are close to that under wormhole routing, except that packeting causes much higher latency. Wormhole routed networks also tend to saturate at lower load-rates because of the blocking property of wormhole routing.

A very interesting aspect from the simulation results in that the mesh always performs very well in comparison to the other networks despite the use of the unit wire delay model. While the best low-load latency is obtained with the fat-tree under constant bisection and constant pin-out constraints (for large networks), it is surprising that the performance of the fat-tree is not generally better than what is shown by our simulations, particularly under the sort of area constraint that motivated study of the fat-tree.

As in most prior studies, the simulations here have been performed using uniform random message patterns. The performance of real parallel algorithms and the dependency of execution time on throughput and latency are topics of interest for future research.

Part II

Wormhole Routing Performance Modeling

Chapter 4

A Performance Model for Wormhole Routed Networks

In the second part of this thesis we take up the issue of performance modeling of wormhole routed networks. This chapter presents a general model for wormhole routed networks. In Chapters 5 and 6 we apply the model to the butterfly fat-tree and mesh networks, respectively, and show the comparisons between model predictions and simulation results.

4.1 Preliminaries

Many recent multicomputers (e.g., Paragon [37], iWarp [10, 52] and CM-5E [9, 47]) have adopted wormhole routing techniques [18] to reduce the communication latency for fine-grained parallel programs. In wormhole routing, messages are divided into flits. The flits of a message snake through the network one after another. Only the head flit of a message contains information about the destination of the message, so messages cannot be separated into independent parts. If the head flit is blocked due to contention for network link, the whole message is blocked in place (spanning over many network nodes). These properties makes modeling of wormhole routed networks very different than for packet routing.

Several performance models have been presented for wormhole routing [15, 20, 33, 38, 39]. Dally [15] was the first to present a wormhole routing performance model. Dally investigated the k-ary n-cubes using e-cube routing and virtual channel techniques to avoid deadlocks. Dally assumed that messages arrives at each node according to a Poisson process, and that message destinations are uniformly distributed and independent. In addition, messages are of fixed length and contain more flits than the diameter of the network. Also, once the head flit reaches the destination, the entire worm is accepted one flit at a clock step without any further blocking.

With these assumptions, Dally determines the message rate to each link in the network based on the probability that a random message is routed through a particular dimension and a particular node in that dimension. Message latencies are then resolved in the reverse order of the dimensions traversed, starting with the lowest dimension since the service time for exiting channels is deterministic, i.e., the length of the worm. The latency seen on the input to a dimension is obtained as the latency seen on the output of the dimension plus the incremental latency encountered at the dimension. The incremental latency at a dimension includes an entering latency to the dimension and a routing latency inside the dimension, both are determined as the product of the probability of contention and the expected latency due to contention. Dally used a simplifying assumption that a message waits upon the completion of at most one other message at each stage, so that the expected latency can be estimated as one half of the latency at the next stage. Since this approximation holds only for low-dimension networks or when the utilization approaches zero, his model becomes quite inaccurate for large binary hypercubes and high traffic rates.

Kim and Das [38] proposed an improved model for message latency based on the combination of a message transfer latency and a blocking latency. The model yielded much more accurate results for the hypercube but is extremely involved and presents no closed-form solution for arriving at performance measures. Draper and Ghosh [20] presented a simple yet accurate model by using well-known queueing principals that have been used for packet routed networks, namely the M/G/1 model to estimate the waiting time. They also present a novel technique to simplify the complexity caused by the use of virtual channels. Kim and Chien [39] also explored use of M/G/1 models as a basis for computing waiting time and expanded the analysis of wormhole routing to include bimodal traffics.

All the prior works, however, have focused on the k-ary n-cubes. The analyses on k-ary n-cubes do not lead directly to a suitable model for other networks, especially indirect networks such as the butterfly fat-tree and the fat-pyramid. Furthermore, messages often have a choice among multiple outgoing links from a node in networks like the butterfly fat-tree and the fat-pyramid, necessitating the use of multiple-server queuing models. Prior works have focused on the use of single server models; the behavior of wormhole routing waiting time in the presence of multiple servers has not been investigated.

This thesis presents a generalized performance model for wormhole routed networks. The model can be applied to indirect networks such as tree-based networks as well as direct networks such as k-ary n-cubes. The model emphasizes the role of service times in computing latency and throughput and presents a general relation between the service time for an incoming channel of a node and the service times on the outgoing channels, leading to an iterative method of computing results. In addition, this thesis applies multi-server queueing models to wormhole routing for the first time to solve the problem caused by redundant links. The models are applied to the butterfly fat-tree(BFT) and 2D-mesh and validated through comparison with empirical simulation.

The model presented in this chapter is based on the following assumptions:

- Arrivals at each source node are governed by a Poisson process, and destinations are uniformly random.
- Messages have fixed length and are longer than the diameter of the network.
- Contentions at incoming links to a node are resolved according to First-Come First-Served (FCFS) scheduling.
- Messages arriving at destinations are immediately consumed at the rate of one flit per time step, i.e., no blocking is encountered at destinations.

These assumptions has been commonly adopted for the analysis of wormhole routed networks. The rest of this chapter is organized as follows. Section 4.2 discusses the performance metrics that are used. Section 4.3 describes the calculation of average latency using service time and waiting time. Section 4.4 discusses calculation of maximum throughput. Section 4.5 summarizes the chapter.

4.2 Performance Metrics

The measures we seek to compute are average latency and throughput. Average latency represents the average time from the generation of a message at its source node until all flits are received at the destination node. Throughput is the maximum sustainable rate of message generation at source nodes; beyond this point, messages accumulate without bound at the sources. Latency at low load-rate and throughput are two important measures of performance, either of which may dominate for certain algorithms, e.g., see Hady's discussion of latency-sensitive and throughput-sensitive algorithms [33].

4.3 Average Latency

An interconnection network consists of processing elements (PE) and routing elements (RE). In direct networks(e.g., k-ary n cubes) a node consists of both a PE and an RE. In indirect networks (e.g., tree-based networks where processors are placed at leaves) processing elements and routing elements are separate nodes. Figure 4.1 shows a general routing model that can be used to represent both direct and indirect networks. A PE is always attached to an RE through an injecting channel and an ejecting channel. An RE, however, may or may not have a PE attached to it.

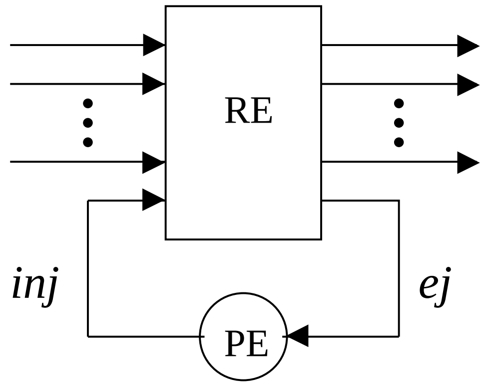

Figure 4.1: A general routing model. A network consists of processing elements (PE) and routing elements (RE). A PE is attached to an RE through an injecting channel and an ejecting channel.

When a message is generated at a processing node j, it encounters the following latencies: 1) A waiting time $W_{inj,j}$ for the injecting channel. This waiting time doesn't depend on the routing scheme (store-and-forward or wormhole), and can be determined as long as the behavior of message arrival rates and the service time for the injecting channel are known. Under the Poisson arrival assumption the waiting time can be resolved using the M/G/1 model. 2) A service time $x_{inj,j}$ at the injecting channel. This is the time from the moment that first flit of the message is accepted by the injecting channel to the moment that the last flit of the message has left the injecting channel. In wormhole routing, the flits of a message spread over many links on the message's path. When the head flit is blocked, the other flits of the worm are blocked in place. Under the long-worm assumption, the service time at the injecting channel includes the waiting times due to blocking at all subsequent channels. 3) An additional time to traverse the rest of the channels on the message's path. Under the assumptions that the length of the worm is longer than the diameter of the network and that there is no blocking at the destination, when the tail of the message has left the injecting channel, the head of the message must have arrived at the destination; the rest of the message will be received one flit per clock step. Therefore, it will take another $D-1$ clock steps for the entire message to be received at the destination, where D is the length of the path (including injecting and ejecting channels).

From the above analysis we can write the the latency L_j for the message injected at node j as

$$L_j = W_{inj,j} + x_{inj,j} + D - 1 \ . \tag{4.1}$$

Averaging over all processing nodes (and the probability distribution of message generation), the average latency $\overline{L}$ for the entire network is then

$$\begin{aligned} \overline{L} &= \frac{1}{N}\sum_j \overline{L}_j \\ &= \frac{1}{N}\sum_j \left(\overline{W}_{inj,j} + \overline{x}_{inj,j}\right) + \overline{D} - 1 \ , \end{aligned} \tag{4.2}$$

where N is the number of processing nodes in the network, $\overline{D}$ is the average message distance, and $\overline{W}_{inj,j}$ and $\overline{x}_{inj,j}$ are the mean waiting time and mean service time for the injecting channels at node j. Equation 4.2 shows that latency depends on the service times and waiting times of injecting channels. But the service time at an injecting channel depends, in turn, on the service time and waiting time for the next channel on a message's path.

4.3.1 Service Time

The service time at the injecting channel, $x_{inj,j}$, depends on the service time of the subsequent channels. More precisely, the service time of a channel is the sum of the waiting time and the service time encountered at the channel immediately following it. Service times are resolved in the reverse order of the channels traversed, from the last channel (ejecting channel) backwards to the injecting channel. Since no further blocking is encountered once the head flit of a message reaches its destination, the service time for the ejecting channel is deterministic, i.e., the length of the worm in flits. To resolve the service times we need to determine the waiting time for each channel.

The service time at the input to a arbitrary node in the network can be analyzed with the aid of Figure 4.2. At any RE, messages from an incoming channel i may be routed to outgoing channels denoted by $j = 0$, 1, etc. The service time for the incoming channel depends on the service times and waiting times at all possible outgoing channels. Denote the probability that a message from incoming channel i is routed to outgoing channel j by $R_{i|j}$; the service time for incoming channel i can then be expressed as

$$x_i^{\text{in}} = \sum_j (x_j + w_{i|j}) \cdot R_{i|j} \ , \tag{4.3}$$

where x_j is the service time for the outgoing channel j, and $w_{i|j}$ is the waiting time for outgoing channel j of messages from incoming channel i.

Equation 4.3 states that the service time at a channel depends on the service time of the subsequent channel and the waiting time for the subsequent channel. Therefore we need to determine the waiting time for the outgoing channels at each node.

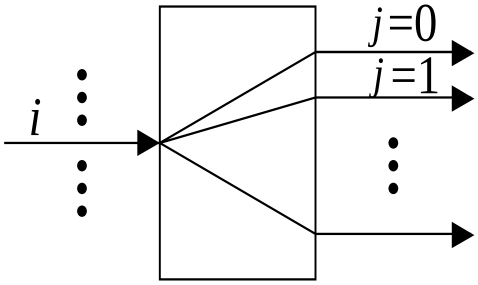

Figure 4.2: Possible subsequent channels for incoming channel i

4.3.2 Waiting Time

The mean waiting time $w_{i|j}$ is caused by contention for the outgoing channel j. When a message is blocked, it must wait for the message that is holding the outgoing channel to be fully serviced. (A worm in service can not be preempted since only the head flit contains routing information.) This motivates us to take advantage of well-known queuing models that have been employed to analyze store-and-forward routing.

When an outgoing channel is treated as single server, results from the M/G/1 model [40] can be used:

$$\overline{W}_{M/G/1} = \frac{\rho\overline{x}(1 + C_b^2)}{2(1 - \rho)} \ , \tag{4.4}$$

where $\rho = \lambda\overline{x}$ is the server utilization, λ is the rate of message arrivals destined for the outgoing channel, $\overline{x}$ is the mean service time, and $C_b^2 = \frac{\sigma_b^2}{\overline{x}^2}$, where σ_b^2 is the variance of the service time distribution.

The source of variance in service time is quite different from that in packet routing. In packet routing, the service time at a channel is determined only by the size of the message being transmitted. The variance in service time is actually determined by the variance in message lengths. Therefore if messages are of equal size then there is no variance in service time. In wormhole routing, the service time for a channel includes all the blocking latencies encountered from the channel to the destination. Since the blocking latencies for the downstream channels are indefinite, variance in service time arises even if the messages are of fixed length.

To reflect the large variance in channel service time, we used a generalized exponential distribution as proposed by [39]. A general distribution of service time can be described by a probability density function $f(t)$:

$$f(t) = \frac{c}{b} \cdot (\frac{t - s/f}{b})^{c-1} \cdot e^{-((t-s/f)/b)^c} \ , \tag{4.5}$$

where s and f are the length of the message and the flit width respectively, so that s/f is the length of the message in flits, and b and c are two parameters that governs the shape of the distribution. When c=1 and s=0, the above general distribution reduces to the well-known exponential distribution with density function $f(t) = (1/b)e^{-t/b}$.

From the general service time distribution we can determine the mean and variance to be

$$\overline{x} = s/f + b \cdot \Gamma(1 + \frac{1}{c}) \tag{4.6}$$

and

$$\sigma_b^2 = b^2 \cdot \left[\Gamma\left(1 + \frac{2}{c}\right) - \Gamma^2\left(1 + \frac{1}{c}\right)\right] \tag{4.7}$$

respectively, where the gamma function is defined by

$$\Gamma(a) = \int_0^{\infty} y^{a-1} \cdot e^{-y} dy \ . \tag{4.8}$$

Now we can determine C_b^2 using the expression for σ_b^2 as

$$C_b^2 = \frac{\sigma_b^2}{\overline{x}^2} = \frac{b^2 \cdot \left[\Gamma\left(1+\frac{2}{c}\right) - \Gamma^2\left(1+\frac{1}{c}\right)\right]}{\overline{x}^2} \ . \tag{4.9}$$

From Equation 4.6 we can get

$$b = \frac{\overline{x} - s/f}{\Gamma(1+\frac{1}{c})} \ . \tag{4.10}$$

By substituting the b here obtained in the C_b^2 expression, we get

$$C_b^2 = \frac{(\overline{x} - s/f)^2}{\overline{x}^2} \cdot \left[\frac{\Gamma\left(1+\frac{2}{c}\right)}{\Gamma^2\left(1+\frac{1}{c}\right)} - 1\right] \ . \tag{4.11}$$

Since the expression inside the square bracket is only a function of c, we can define

$$\gamma(c) \equiv \frac{\Gamma\left(1+\frac{2}{c}\right)}{\Gamma^2\left(1+\frac{1}{c}\right)} - 1 \ , \tag{4.12}$$

then for short hand we can write

$$C_b^2 = \gamma(c) \cdot \frac{(\overline{x} - s/f)^2}{\overline{x}^2} \ . \tag{4.13}$$

Kim and Chien [39] have investigated the distributions of service time for messages with varying sizes and found that a c value of in the range 0.5-0.75 best describes bimodal traffic. In general, A proper value of c and can be picked through some comparison of the model with experimental results.

An approximation to C_b^2 can be obtained by taking $c = 1$, as in the exponential distribution. With this approximation, the coefficient C_b^2 simplifies to

$$C_b^2 = \frac{(\overline{x} - s/f)^2}{\overline{x}^2} \; . \tag{4.14}$$

Comparisons between model and experimentation in Chapters 5 and 6 show that this simplification gives rather close agreement. Draper and Ghosh [20] also used this approximation for variance in service time in their analysis of k-ary n-cubes and have shown good approximations under general conditions.

Substituting for ρ and C_b^2 in the waiting time expression (Equation 4.4), we get the mean waiting time for single-server queues,

$$\overline{W}_{M/G/1} = \frac{\lambda \overline{x}^2}{2(1 - \lambda \overline{x})} \cdot \left[1 + \frac{(\overline{x} - s/f)^2}{\overline{x}^2} \right] \; . \tag{4.15}$$

In certain situations, multiple outgoing links from a switch must be treated as one multi-server channel. This is usually due to the existence of redundant paths to increase bandwidth. Multiple-server systems with general service time distributions (M/G/m queues) are more complicated than M/G/1 queues, but we make use of an approximation of Hokstad [35] giving the average waiting time as:

$$W_{M/G/m} = \overline{x}(1 + C_b^2)Q/2m(1 - \rho) \; , \tag{4.16}$$

where m is the number of servers, $\rho = \lambda \overline{x}/m$, $Q = \frac{\rho}{1-\rho} p_{m-1}$ is the probability that a message has to wait in the queue, and p_k, the probability of k messages in the system for $k \leq m$, is given by:

$$p_k = \begin{cases} \left[\frac{(m\rho)^m}{m!(1-\rho)} + \sum_{k=0}^{m-1}\frac{(m\rho)^k}{k!}\right]^{-1} & \text{for } k = 0 \\ p_0\frac{(m\rho)^k}{k!} & \text{for } k \leq m \end{cases} \tag{4.17}$$

It may be noted that the p_k values are consistent with the results for the M/M/m queue, and substituting for p_{m-1} in the expression for Q, gives Erlang's C formula [40],

$$Q = \frac{p_0(m\rho)^m}{m!(1-\rho)} , \tag{4.18}$$

with p_0 as above.

For $m = 2$, we obtain $p_0 = \frac{1-\rho}{1+\rho}$ and $Q = \frac{2\rho^2}{1+\rho}$. The average waiting time for the M/G/2 queue is then

$$\begin{aligned} \overline{W}_{M/G/2} &= \frac{\rho^2\overline{x}}{2(1-\rho^2)}(1+C_b^2) \\ &= \frac{\lambda^2\overline{x}^3}{2(4-\lambda^2\overline{x}^2)}(1+C_b^2) \end{aligned} \tag{4.19}$$

We again use Equation 4.14 to approximate C_b^2, yielding:

$$\overline{W}_{M/G/2} = \frac{\lambda^2\overline{x}^3}{2(4-\lambda^2\overline{x}^2)} \cdot \left[1 + \frac{(\overline{x} - s/f)^2}{\overline{x}^2}\right] . \tag{4.20}$$

Equations 4.15 and 4.20 are the basic models for waiting time under the assumptions of Poisson arrival and general service time distribution. These models, however, assumes independent arrivals at the inputs of a switch, all of which may block one another. Such an assumption is not quite accurate for wormhole routing. To understand the difference, consider a simplest case where a node has only one incoming channel and one outgoing channel. In packet routing, messages arriving at the incoming channel will wait for a time determined by the residual service time for the message that occupies the outgoing channel. This is so because a packet is usually much bigger than the number of bits that can be transmitted over a channel in one clock step, while a new message may arrive any time when the one in the server is being transmitted. With wormhole routing, however, there is no waiting time encountered in the above case. If a newly arrived worm sees another worm is being served by the outgoing channel, then the newly arrived worm must be following the tail of the preceding worm. Under the long-worm assumption, the head of the preceding worm must have already arrived at the destination, therefore the rest of the worm is accepted one flit at a time without further blocking. Therefore there is no blocking of the newly arrived worm by the one being served.

In general, when there are multiple incoming channels, a newly arrived worm may be blocked only if all the worms being served came from incoming channels other than the one the newly arrived worm is from. The waiting times for wormhole routing are therefore less than those predicted using Equations 4.15 and 4.20. Since a worm has to wait only if it is blocked by message from other incoming channels, we can obtain the waiting times for wormhole routing as a product of the waiting times obtained from Equations 4.15 and 4.20 and a blocking probability $P_{i|j}$:

$$w_{i|j} = P_{i|j} W_j \ , \tag{4.21}$$

where $P_{i|j}$ should reflect the probability that m messages deemed to be in service by the M/G/m model actually emanate from m distinct incoming links other than link i. A simple approximation is

$$P_{i|j} = \begin{cases} 0 & \text{if number of incoming channels} \leq m \\ 1 - m\frac{\lambda_i^{\text{in}}}{\lambda_j} R_{i|j} & \text{if number of incoming channels} > m \end{cases} . \qquad (4.22)$$

where λ_i^{in} is the total message rate on incoming channel i, λ_j is the total message rate on outgoing channel j, and the number of servers, m, is less than the number of incoming links. When $m = 1$, the expression is exact, i.e., $P_{i|j}$ is 1 minus the probability that an arbitrary message destined for output j is from input i. For larger m, we are approximately accounting for the probability that any of the servers holds a message from input link i; if all the arrival rates on incoming links are modest relative to the rate on outgoing channel j, the probabilities of multiple arrivals from the same input in the M/G/m model are small enough to safely ignore.

Note that for single-server case the above expression reduces to

$$P_{i|j} = 1 - \frac{\lambda_i^{\text{in}}}{\lambda_j} R_{i|j} \ , \qquad (4.23)$$

which agrees with the blocking probability presented in [20]. We may view $P_{i|j}$ as the probability that a message deemed by the M/G/1 model to block an incoming message from input channel i is actually a possible blocking message, i.e., one that comes from some other input channel.

By combining Equations 4.3, 4.21 and 4.22 we obtain the service time for messages incident to an incoming channel i:

$$x_i^{\text{in}} = \sum_j \left[x_j + (1 - m\frac{\lambda_i^{\text{in}}}{\lambda_j} R_{i|j}) W_j \right] R_{i|j} \ . \qquad (4.24)$$

Equation 4.24 is used together with Equations 4.15 and 4.20 to iteratively resolve the service times for all channels. Average latency is then determined from Equation 4.2.

4.4 Throughput

Throughput is another important metric of network performance. Through the above analysis, waiting times at each link on a route can be obtained, from which we can determine the service time at the source. To find the throughput, the source service time is set equal to the reciprocal of the source arrival rate [15]. At this operating point messages are being offered as fast as the network can deliver them; the network saturates and can accept no more traffic.

4.5 Synopsis

Wormhole routing differs from packet routing in several aspects. Firstly, the service times in wormhole routing depend on those in the subsequent channels and need to be resolved recursively backward from the destination. Secondly, the distribution of service is mainly determined be the blocking latencies on the subsequent channels, instead of the variations in message size (since we are assuming fixed-length messages). Furthermore, the waiting times obtained from M/G/m models has to be adjusted by a blocking probability since a worm encounters a waiting time at a node only when it is blocked by worms from other incoming channels to the node.

The model presented in this chapter has the following novel aspects: 1) We investigated the distribution in service time for wormhole routing and justified using the simple measure of variance in service time given by Equation 4.14. 2) We gave a generalized service time relation between the channels incident to a node and channels going out from the node (Equation 4.24). The relation is independent of any particular network and therefore may be applied to indirect networks as well as direct networks. 3) Most importantly, we discussed the computation of of waiting times for multi-server channels based on M/G/m queueing model and an extended analysis of the blocking probabilities in presence of multiple servers. The multi-server model has applications for networks that use redundant links to increase the bandwidth (e.g., the butterfly fat-trees). We will apply the model to the butterfly fat-tree and the mesh networks in the chapters that follow.

Chapter 5

Analysis of Butterfly Fat-Tree

We have presented a general performance model for wormhole routed networks in Chapter 4. In this chapter we apply the general model to the butterfly fat-tree. Fat-trees constitute an interesting class of networks due to their area-universality properties [6, 31, 32, 44, 46] and their influence on the design of actual parallel computers [7, 47, 49]. There are several ways that modeling the butterfly fat-tree differs from modeling k-ary n-cubes. First, the butterfly fat-tree is not node-symmetric, so it does not suffice to analyze the traffic situation at a single node. Still, the butterfly fat-tree has a very regular structure, and deadlock never results when messages are routed over shortest paths. For k-ary n-cubes, generally, e-cube routing and/or virtual channels are used to avoid deadlock [16, 18]. The deadlock avoidance schemes for k-ary n-cubes produce some complication in the sense that they create asymmetry among different links in the network, but they actually lead to a major simplification by fixing a specific path for any message with a given source and destination. More interestingly, in the butterfly fat-tree, messages often have a choice among two outgoing links from a node, necessitating the use of multiple-server queuing models. We will illustrate use of the techniques presented in Chapter 4 to solve the problem introduced by redundant links.

We start with an analysis of the message arrival rates in the butterfly fat-tree in Section 5.1. We then determine the service time and waiting time to each channel in Section 5.2. Latency and throughput are then resolved in Sections 5.3 and 5.4. In Section 5.5 we compare the results from modeling with results from empirical simulations. Section 5.6 summarizes this chapter.

5.1 Message Arrival Rates

We use the butterfly fat-tree with N processors as described in Chapter 2. To help illustrate the calculation of message rates, we consider a drawing of the butterfly fat-tree in Figure 5.1, where the squares represent switches and solid circles (leaves) represent processors. Each switch has four child ports and two parent ports. When a processor sends an message to another processor, the message goes up the tree to a switch at a level such that the the destination processor is in the subtree rooted from that switch. When a message goes down the tree, there is only one choice from the four child ports at each stage. For up-going messages, however, the two parent ports are completely redundant; each can route the message to its destination with the same distance. Therefore there is more than one shortest path between a pair of leaves in the butterfly fat-tree. When a worm needs to go up, it uses what we call greedy path selection that's a method of deciding priority among incoming links to select the up-going path, i.e., it selects an up-link randomly, if that link is blocked, it tries the other, and if both are blocked, it waits.

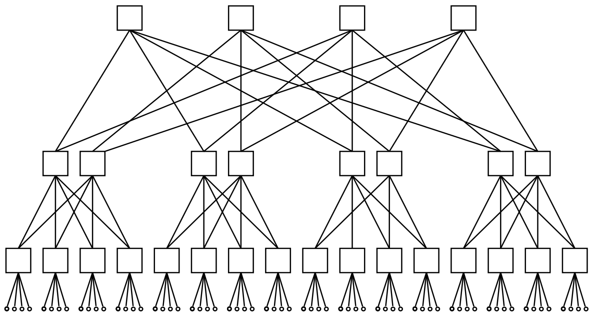

Figure 5.1: Butterfly Fat-Tree With 64 Processors

To obtain the message arrival rates to each link, we assume the mean departure rate of a node is equal to the mean arrival rate provided that the network is not saturated [20]. Note that in the butterfly fat-tree, links that are at the same level and run in the same direction (up or down) are symmetrical, hence there is no need to distinguish among them. We can label the links and their arrival rates by a pair of indices $\langle i, j \rangle$ where i is the starting level of the link and j is the ending level of the link in the network, $0 \leq i, j \leq p$ with $p = \log_4 N$.

Assume each processor injects messages into the network at a rate of λ_0. Under steady state conditions, we have $\lambda_{0,1} = \lambda_{1,0} = \lambda_0$ for links between the processors ($l = 0$) and the first level switches ($l = 1$).

Now consider links between switches between level l and $l+1$ ($1 \leq l < p$). Since there are $N = 4^p$ processors in the system, a message may have $4^p - 1$ destinations, of which $4^l - 1$ can be reached without going up from level l. Then, the probability that a message goes up from level l, denoted $P_l^{\uparrow}$, is

$$P_l^{\uparrow} = \frac{4^p - 4^l}{4^p - 1} \; . \tag{5.1}$$

The probability that a message goes down from level l, $P_l^{\downarrow}$, is then

$$\begin{aligned} P_l^{\downarrow} &= 1 - P_l^{\uparrow} \\ &= \frac{4^l - 1}{4^p - 1} \ . \end{aligned} \tag{5.2}$$

The total message rates going up from level l to level $l+1$ is $P_l^{\uparrow} 4^p \lambda_0$. There are $\frac{4^p}{2^l}$ links between level l and $l+1$. The message rate to each channel going from level l to level $l+1$ is $\lambda_{l,l+1} = P_l^{\uparrow} 4^p \lambda_0 / (4^p/2^l) = \lambda_0 P_l^{\uparrow} 2^l$. The message rate going downward from level $l+1$ to level l equals that going up from level l to level $l+1$ due to symmetry. In summary, we have

$$\lambda_{l,l+1} = \lambda_0 \frac{4^p - 4^l}{4^p - 1} 2^l \tag{5.3}$$

$$\lambda_{l+1,l} = \lambda_{l,l+1} \ . \tag{5.4}$$

5.2 Waiting and Service Times

Since a message is received by the destination processor one flit at a clock as soon as the head flit has reached the destination, the service time for links from a level 1 switch to a processor is deterministic, i.e, the length of a worm:

$$x_{1,0} = s/f \ . \tag{5.5}$$

The waiting time $\overline{W}_{1,0}$ is determined using Equation 4.15, i.e.,

$$\overline{W}_{1,0} = \overline{W}_{M/G/1}(\lambda_{1,0}, x_{1,0}) \ . \tag{5.6}$$

For any other down-going channels from level $l+1$ to level l ($1 \leq l < p$), there are 4 possible outgoing channels (the 4 children), each with the same probability (1/4). The service time $\overline{x}_{l+1,l}$ is determined using Equation 4.24:

$$\overline{x}_{l+1,l} = \overline{x}_{l,l-1} + \left(1 - \frac{1}{4}\frac{\lambda_{l+1,l}}{\lambda_{l,l-1}}\right)\overline{W}_{l,l-1} \ . \tag{5.7}$$

The waiting time $\overline{W}_{l+1,l}$ is determined using $\overline{x}_{l+1,l}$:

$$\overline{W}_{l+1,l} = \overline{W}_{M/G/1}(\lambda_{l+1,l}, \overline{x}_{l+1,l}) \ . \tag{5.8}$$

Now consider up-going channels, starting with channel $\langle p-1, p\rangle$. There are only 3 possible outgoing channels (siblings) after traversing channel $\langle p-1, p\rangle$, each with the same probability (1/3). Therefore

$$\begin{aligned}\overline{x}_{p-1,p} &= \overline{x}_{p,p-1} + \left(1 - \frac{\lambda_{p-1,p}}{\lambda_{p,p-1}}\frac{1}{3}\right)\overline{W}_{p,p-1} \\ &= \overline{x}_{p,p-1} + \frac{2}{3}\overline{W}_{p,p-1} \ . \end{aligned} \tag{5.9}$$

The waiting time $\overline{W}_{p-1,p}$ is determined using the two-server model (Equation 4.20), i.e.,

$$\overline{W}_{p-1,p} = \overline{W}_{M/G/2}(\lambda_{p-1,p}, \overline{x}_{p-1,p}) \ . \tag{5.10}$$

For any other up-going channels from level $l-1$ to level l ($1 \leq l < p-1$), a message may go upward from level l with probability $P_l^{\uparrow}$ or go downward with probability $P_l^{\downarrow}$. In the case that the message goes upward, there are two redundant up-going channels that are treated as one two-server channel. In the case that the message goes downward, there are three possible outgoing channels (siblings), each with the same probability (1/3). Therefore the service time is

$$\begin{aligned}\overline{x}_{l-1,l} &= \left[\overline{x}_{l,l+1} + \left(1 - \frac{\lambda_{l-1,l}}{\lambda_{l,l+1}}P_l^{\uparrow}\right)\overline{W}_{l,l+1}\right]P_l^{\uparrow} \\ &\quad + \left[\overline{x}_{l,l-1} + \left(1 - \frac{P_l^{\downarrow}}{3}\right)\overline{W}_{l,l-1}\right]P_l^{\downarrow} \ . \end{aligned} \tag{5.11}$$

The waiting time $\overline{W}_{l-1,l}$ is determined using the two-server model (Equation 4.20)

$$\overline{W}_{l-1,l} = \overline{W}_{M/G/2}(\lambda_{l-1,l}, \overline{x}_{l-1,l}) \ , \tag{5.12}$$

except for $l = 1$. Channel $\langle 0, 1 \rangle$ is from processor to first level switch with no redundant channel; therefore, the single server model should be applied, i.e.,

$$\overline{W}_{0,1} = \overline{W}_{M/G/1}(\lambda_{0,1}, \overline{x}_{0,1}) \ , \tag{5.13}$$

5.3 Average Latency

Now we can use Equation 4.2 to compute the average latency. For the butterfly fat-tree, $x_{inj,j} = x_{0,1}$ and $W_{inj,j} = W_{0,1}$. Since all processors are equivalent due to symmetry, averaging over injecting channels is unnecessary. Therefore the latency is determined as

$$\overline{L} = \overline{W}_{0,1} + \overline{x}_{0,1} + (\overline{D} - 1). \tag{5.14}$$

5.4 Throughput

Maximum throughput is computed by setting the source service time to the reciprocal of the source arrival rate, i.e.,

$$\overline{x}_{0,1} = \frac{1}{\lambda_0} \ . \tag{5.15}$$

Source service time $\overline{x}_{0,1}$ increases as arrival rate increases, while $\frac{1}{\lambda_0}$ is a monotonically decreasing function of λ_0. Graphically, if $\overline{x}_{0,1}$ and $\frac{1}{\lambda_0}$ are plotted against arrival rate, the maximum throughput is the arrival rate at the intersection of the two curves. In practice we let source arrival rate increase (starting at a small value) until the above equation is satisfied.

5.5 Experimental Validation

The performance model for the butterfly fat-tree was validated through comparisons with simulations. Fixed length messages are used for the simulation. Latencies from the model and simulation were compared for networks with up to 1024 processing nodes. Messages of 16, 32 and 64 flits in length are studied. Figures 5.2 through 5.4 show the result of the comparisons for average latencies. The experimental values were obtained from simulation over 100,000 clock steps at different load-rates. The data for the first 10,000 clocks are discarded to get rid of the start-up effect. At each load-rate, the mean and the standard deviation of the latency were computed using the data sampled. At low load-rates the network is stable; the latencies oscillates around the mean value with a small deviation. When the load-rate approaches saturation, the latency has a steeply increasing trend, and there is a large deviation in the latency. At each load-rate, we compute a percentage-error as the ratio of standard deviation and the mean. We show empirical data points at load-rates for which the error is less than 5 percent.

Figure 5.5 shows the comparison of throughputs for networks with up to 4096 processing nodes (for the 32-flit messages). The model produced accurate predictions on latency and throughput for all cases under study. Observe that there are only slight increases in latencies as network size grows since the average distance of messages depends on the height of the butterfly fat-tree.

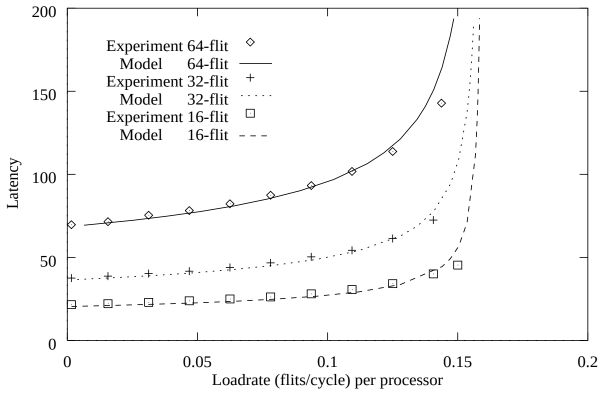

Figure 5.2: Comparisons for 64-processor BFT network for 16, 32 and 64-flit messages

The throughput per processor, however, decreases by a factor of $\approx 1/2$ as network size increases by a factor of 4 since the bisection bandwidth of the butterfly fat-tree increases at the rate of $\sqrt{N}$.

5.6 Synopsis

We analyzed the performance of the butterfly fat-tree under wormhole routing in this chapter using the general model presented in Chapter 4. We illustrated using of the multi-server queueing model to solve the waiting times for redundant channels. The results obtained from model were validated through comparison with simulation results; excellent agreement was obtained for different network sizes and worm lengths.

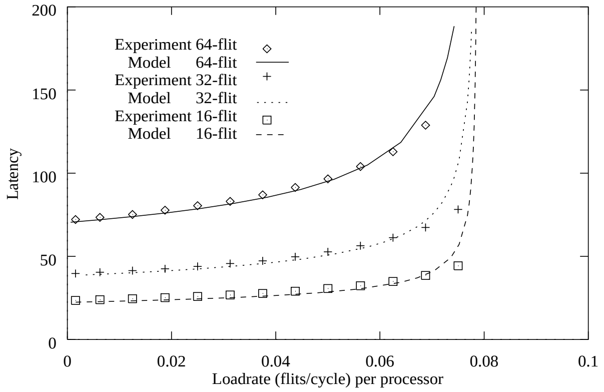

Figure 5.3: Comparisons for 256-processor BFT network for 16, 32 and 64-flit messages

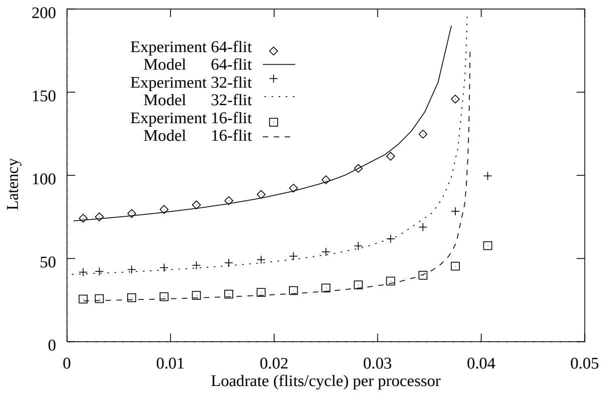

Figure 5.4: Comparisons for 1024-processor BFT network for 16, 32 and 64-flit messages

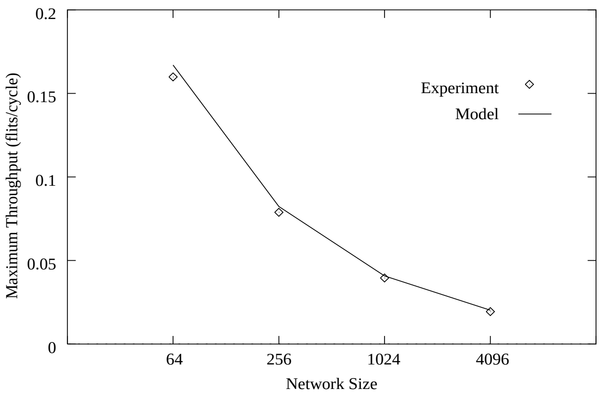

Figure 5.5: Comparison of maximum throughput for BFT networks with 64, 256, 1024 and 4096 processors for 32-flit messages

Chapter 6

Analysis of Mesh With Comparison to Torus

Mesh and torus networks have often been proposed as the interconnection pattern for parallel computers. For example, the Intel Paragon [9] uses a 2D-mesh arrangement of processors. A recent example of a torus-based machine, albeit in three dimensions, is the Cray T3E [12]. In addition, wormhole routing has increasingly been advocated as a method of reducing latency. Most analysis of wormhole routed networks [1, 15, 20, 38, 39], however, has focused on the torus and the broader class of k-ary n-cubes to which it belongs. This chapter presents a performance model for the wormhole routed mesh, and it compares the performance of the mesh and torus based on theoretical and empirical analyses.

Section 6.1 presents analysis of the mesh network using the general model in Chapter 4. Section 6.2 compares the model and simulations for the mesh network. In addition, the analytical and empirical results for the mesh are compared to corresponding results for the torus. Finally, Section 6.3 summarizes this chapter.

6.1 Analysis of Meshes

We now apply the model of Chapter 4 to the mesh network. In this network, nodes are connected by bidirectional links, without wrap-arounds at the edges of the mesh. Let the number of nodes be k^2, i.e., k is the side length in each dimension. Each node in the network represents a switch and a processor.

An important issue for wormhole routing is deadlock avoidance. In a mesh network (and in k-ary n-cubes), deadlock may be prevented by using e-cube routing, i.e., messages are routed in one dimension and then the other (in the 2D case). In the analysis presented here, we assume routing in the y dimension first.

We label each node in the network using a pair of indices $\langle j_0, j_1 \rangle$, with j_0 and j_1 ($0 \leq j_0, j_1 < k$) denoting the position of the node in the x and y-dimension, respectively. We can label each channel in the network using a node label together with a direction (N,S,E,W); for example, $\langle j_0, j_1, N \rangle$ denotes the channel going north from node $\langle j_0, j_1 \rangle$. Since each link is bidirectional, there are $2k(k-1)$ channels in each dimension.

We adopt the concept of channel equivalence class, defined by [20] as a set of channels that have identical statistical properties regarding message rate and service time. Because of the symmetry in a mesh network, an east-bound channel $\langle j_0, j_1, E \rangle$ has identical traffic behaviors as its corresponding west-bound channel $\langle k-1-j_0, j_1, W \rangle$, i.e., they belong to the same channel equivalence class. Similarly north-bound channel $\langle j_0, j_1, N \rangle$ and its corresponding south-bound channel $\langle j_0, k-1-j_1, S\rangle$ belong to the same class. Thus, we can reduce the number of equivalence classes to $k(k-1)$ in each dimension. We only have to determine the behavior of west and south-bound channels and then map the results to east and north-bound channels.

Furthermore, channels in the x-dimension with the same x-positions (i.e., $\langle j_0, j_1, E\rangle$, $j_1 = 0, 1, ..., k-1$) are equivalent, therefore the number of equivalence classes is reduced to $k-1$ for x-dimension channels and we can drop the y-index from the channel labels, i.e., we will use $\langle j_0, W\rangle$ to denote a west-bound channel from node $\langle j_0, j_1\rangle$ with an arbitrary j_1 $(0 \leq j_1 < k)$. Since our analysis is focused on west and south-bound channels, for convenience we will drop the direction labels for them, i.e. we will use $\langle j_0\rangle(0 < j_0 < k)$ to denote the $k-1$ channel equivalence classes in the x-dimension and $\langle j_0, j_1\rangle(0 \leq j_0 < k, 0 < j_1 < k)$ to denote the $k(k-1)$ channel equivalence classes in the y-dimension. We specify the direction labels explicitly when we refer to east and north-bound channels.

In this section, we first discuss message rates to each channel equivalence class. We then determine the service times for the $k-1$ channel equivalence classes in the x dimension, followed by the service times for the $k(k-1)$ channel equivalence classes in the y-dimension. Service times and waiting times at injection channels are then determined and average latency is determined using Equation 4.2.

6.1.1 Message Rates

Assume messages arrives to each injection channel according to a Poisson process with rate λ_{node}. Also assume the mean departure rate is equal to the mean arrival rate to a channel provided that the channel is stable (channel utilization factor < 1). The message rates for each channel is determined according to the number of source nodes that use the channel and the number of destination nodes that can be reached from the channel.

Extending the channel notation in a natural way to injection and ejection channels, we have:

$$\lambda_{\langle inj,j_0,j_1\rangle} = \lambda_{\langle j_0,j_1,ej\rangle} = \lambda_{node} \qquad (0 \le j_0 < k, \quad 0 \le j_1 < k)\ . \tag{6.1}$$

For the $k-1$ channel equivalence classes in the x-dimension, the message rates are

$$\lambda_{\langle j_0\rangle} = \frac{j_0(k-j_0)k}{k^2-1}\lambda_{node} \qquad (1 \le j_0 < k)\ . \tag{6.2}$$

There are $k(k-1)$ channel equivalence classes in the y-dimension; the message rates are

$$\lambda_{\langle j_0,j_1\rangle} = \frac{j_1(k-j_1)k}{k^2-1}\lambda_{node} \quad \begin{cases} 0 \le j_0 < k \\ 1 \le j_1 < k \end{cases} . \tag{6.3}$$

(Note that although the message rates for the channels in the y-dimension are independent of j_0, the service times are not, which is why there are different equivalence classes for different values of j_0).

6.1.2 Service Times

Service times are resolved in the order of ejection channels, x-dimension channels, y-dimension channels and injection channels. At the ejection channel, once the head flit of a message is received, the rest of the message will be received one flit at a time without any further blocking. Therefore the service time at the ejection channels is equal to the length (in number of flits) of the message:

$$x_{\langle j_0, j_1, ej \rangle} = s/f \ , \tag{6.4}$$

where s and f are the message and flit size, respectively.

The service times for x-dimension channels are determined as follows: Upon leaving channel $\langle j_0 \rangle$, a message exits to the ejection channel available at that point, with probability $R_{\langle j_0 \rangle | ej} = \frac{1}{j_0}$, or continues to the next channel in the same direction, with probability $R_{\langle j_0 \rangle | \langle j_0 - 1 \rangle} = \frac{j_0 - 1}{j_0}$. The service time $x_{\langle j_0 \rangle}$ is obtained using Equation 4.24 and Equation 6.2:

$$x_{\langle j_0 \rangle} = (s/f)\,\frac{1}{j_0} + \left[x_{\langle j_0 - 1 \rangle} + \frac{1}{k - j_0 + 1} W_{\langle j_0 - 1 \rangle} \right] \frac{j_0 - 1}{j_0} \ . \tag{6.5}$$

The service times for the y dimension channels are determined as follows: Upon leaving channel $\langle j_0, j_1 \rangle$, a message may exit to the ejection channel available at that point, with probability $R_{\langle j_0, j_1 \rangle | ej} = \frac{1}{j_1 k}$, continue to the next channel in the same direction, with probability $R_{\langle j_0, j_1 \rangle | \langle j_0, j_1 - 1 \rangle} = \frac{j_1 - 1}{j_1}$, switch to a west-bound channel, with probability $R_{\langle j_0, j_1 \rangle | \langle j_0 \rangle} = \frac{j_0}{j_1 k}$, or switch to an east-bound channel, with probability $R_{\langle j_0, j_1 \rangle | \langle j_0, E \rangle} = \frac{k - 1 - j_0}{j_1 k}$. The service time $x_{\langle j_0, j_1 \rangle}$ is obtained using Equation 4.24 and Equation 6.3:

$$\begin{aligned}
x_{\langle j_0, j_1 \rangle} \quad &= \quad (s/f)\,\frac{1}{j_1 k} \\
&+ \quad \left[x_{\langle j_0 \rangle} + \frac{k(k - j_0) - (k - j_1)}{k(k - j_0)} W_{\langle j_0 \rangle} \right] \frac{j_0}{j_1 k} \\
&+ \quad \left[x_{\langle k - 1 - j_0 \rangle} + \frac{k(j_0 + 1) - (k - j_1)}{k(j_0 + 1)} W_{\langle k - 1 - j_0 \rangle} \right] \frac{k - 1 - j_0}{j_1 k}
\end{aligned}$$

$$+ \quad \left[x_{\langle j_0, j_1-1 \rangle} + \frac{1}{k - j_1 + 1} W_{\langle j_0, j_1-1 \rangle} \right] \frac{j_1 - 1}{j_1} \; . \tag{6.6}$$

The service times at the injection channels are determined as follows: Upon leaving an injection channel at node $\langle j_0, j_1 \rangle$, a message may go north-bound, with probability $R_{inj|\langle j_0, j_1, N \rangle} = \frac{(k-1-j_1)k}{k^2-1}$, south-bound, with probability $R_{inj|\langle j_0, j_1 \rangle} = \frac{j_1 k}{k^2-1}$, east-bound, with probability $R_{inj|\langle j_0, E \rangle} = \frac{k-1-j_0}{k^2-1}$, or west-bound with probability $R_{inj|\langle j_0 \rangle} = \frac{j_0}{k^2-1}$ respectively. Therefore, the service time for an injection channel is obtained using Equation 4.24 and Equation 6.2 and 6.3:

$$\begin{aligned} x_{\langle inj, j_0, j_1 \rangle} &= \left[x_{\langle j_0 \rangle} + \frac{k(k-j_0)-1}{k(k-j_0)} W_{\langle j_0 \rangle} \right] \frac{j_0}{k^2-1} \\ &+ \left[x_{\langle k-1-j_0 \rangle} + \frac{k(1+j_0)-1}{k(1+j_0)} W_{\langle k-1-j_0 \rangle} \right] \frac{k-1-j_0}{k^2-1} \\ &+ \left[x_{\langle j_0, j_1 \rangle} + \frac{k-j_1-1}{k-j_1} W_{\langle j_0, j_1 \rangle} \right] \frac{j_1 k}{k^2-1} \\ &+ \left[x_{\langle j_0, k-1-j_1 \rangle} + \frac{j_1}{1+j_1} W_{\langle j_0, k-1-j_1 \rangle} \right] \frac{(k-1-j_1)k}{k^2-1} \; . \end{aligned} \tag{6.7}$$

The waiting times $W_{\langle inj, j_0, j_1 \rangle}$ are computed using $x_{\langle inj, j_0, j_1 \rangle}$ and Equation 4.15. The average latency for the entire network is then

$$\overline{L} = \frac{1}{k^2} \sum_{j_0=0}^{k-1} \sum_{j_1=0}^{k-1} \left(W_{\langle inj, j_0, j_1 \rangle} + x_{\langle inj, j_0, j_1 \rangle} \right) + \overline{D} - 1 \; . \tag{6.8}$$

For uniform random messages, $\overline{D} = \frac{2k}{3} + 2$.

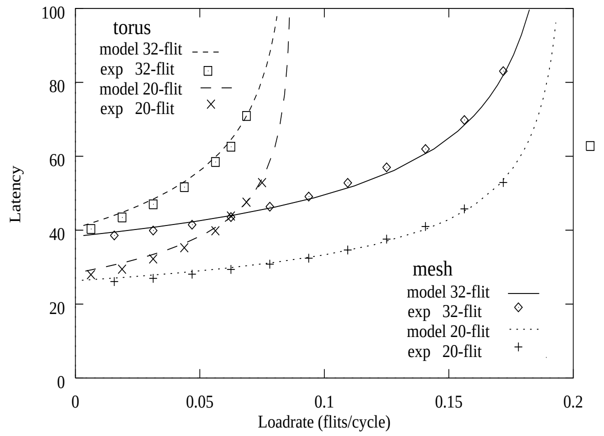

Figure 6.1: Comparison of model and simulation results for the mesh and torus with 64 nodes using message lengths of 20 and 32 flits.

6.2 Comparison of Mesh and Torus Models to Simulations and to Each Other

In this section we show the close correspondence of the analysis of Section 6.1 with simulations of the mesh, and we compare the mesh to the torus. Note that it is reasonable to compare the bidirectional mesh to the unidirectional torus, since these are networks of equal bisection width.

The lower right portions of Figures 6.1 and 6.2 show the comparison of latency results from the model and simulation for a mesh with 64 nodes and 256 nodes. The model agrees well with simulation results for all cases under study.

For modeling of the torus network we use the results of Draper and Ghosh [20] with only slight modification; we include the effect of injection and ejection channels (but without the relatively insignificant contribution of waiting time on ejection channels, for simplicity). Appendix B provides the details of the

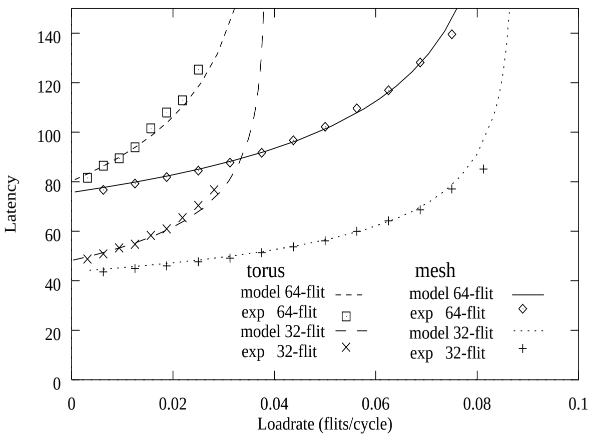

Figure 6.2: Comparison of model and simulation results for the mesh and torus with 256 nodes using message lengths of 32 and 64 flits.

model. (One important difference that may be noted between the mesh and torus networks is that e-cube routing does not suffice to prevent deadlock in the torus, but adding in the use of virtual channels [18] does.)

Figures 6.1 and 6.2 show both the close correspondence of the torus model with experimental results and a comparison of latencies between mesh and torus networks of 64 and 256 nodes. We see that the torus network tends to saturate at an earlier load-rate (about one half that of the mesh). Also, at low load-rates, the latency for the torus is slightly higher than that of the mesh under the same load condition and with the same message lengths. In addition to showing better performance than the torus at any load rate, the mesh is easier to lay out. The networks have the same bisection width, and although the torus can be laid out with area and maximum wire length only a constant factor larger than for the mesh [15, Fig. 5], it cannot quite achieve the *same* area and wire length bounds as the mesh.

One may further note that under very low load rate, a very simple model for latency may be applied, namely number of flits plus average message distance. The key difference between mesh and torus then is that the average message distance (without injecting and ejecting channels) is $2k/3$ for the mesh versus $k^2/(k+1)$ for the torus. The average distances may be computed by doubling the average distance in the x dimension, D_x, which may be found by considering a single row of the mesh or torus and utilizing the probability $k/(k^2-1)$ that a message from a node in a specified x-position goes to a node in some specific other x-position. We find

$$D_x = \frac{1}{k} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} |i-j| \frac{k}{k^2-1}$$

for the mesh, and

$$D_x = \frac{1}{k} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} j \frac{k}{k^2-1}$$

for the torus.

6.3 Synopsis

This chapter has derived an accurate model for predicting the performance of wormhole routed mesh networks. It has also presented empirical simulations and comparisons showing that the mesh network exhibits better performance than the torus.

Earlier comparisons among k-ary n-cubes [15] have favored low-dimensional networks under the important hardware cost measure of equal bisection width. The results of this chapter show that an even more dramatic comparison can be obtained by stepping out of the general k-ary n-cube framework, in which the two-dimensional network is the torus, to consider the two-dimensional mesh.

Part III

Problem Solving in Multiprocessor Networks

Chapter 7

Finding Connected Components on a Scan Line Array Processor

This chapter considers a variation on an algorithm of Greenberg [25] for finding connected components on a scan line array processor (SLAP). This algorithm is slightly simpler than algorithm in [25] and should have a better constant factor in its running time. The algorithm is guaranteed to run in $O(N \lg N)$ time and appears likely to complete in $O(N)$ time for all or most images. Simulations of this algorithm on random images agree with the $O(N)$ time prediction.

Sections 7.1 and 7.2, which borrow heavily from [25], provide background material and Greenberg's algorithm. Section 7.3 explains the simplified algorithm and describes simulation results. Conclusions are contained in 7.4.

7.1 Background

The SLAP (also referred to as the Princeton Engine or Sarnoff Engine) has been proposed by several authors as an efficient SIMD machine for low-level or intermediate-level image processing tasks [11, 21, 22, 41]. The basic structure of the SLAP for computations on an image of size up to $N \times N$ pixels is a linear array of N processors. The rows of the image are input to the SLAP one after another in the direction perpendicular to the array connections (as in Figure 7.1). Thus, each image row is input to the SLAP in constant time, one pixel per processor. For some low-level image processing tasks, such as median filtering with a small window size or convolution of an image with a small kernel (e.g., [34]), only a constant amount of memory per processor is required. But for intermediate-level tasks, it may be necessary to input the entire image to the SLAP before output is produced, so the SLAP is designed to have $\Theta(N)$ memory per processor. Thus, in $O(N)$ time, an entire image can be input to the SLAP, with each processor holding one column of the image. Communication can also be performed using the linear-array connections between the processors. On any given time step, one data item (i.e., $O(\lg N)$ bits) can be transferred on the link between each pair of adjacent processors.

A fundamental intermediate-level image processing task is labeling of the connected components in a binary image. That is, each pixel is 0 or 1, two pixels are connected if there is a path of adjacent (horizontally or vertically) 1-valued pixels from one to the other, and the problem is to label each pixel so that any two 1-valued pixels are assigned the same label if and only if they

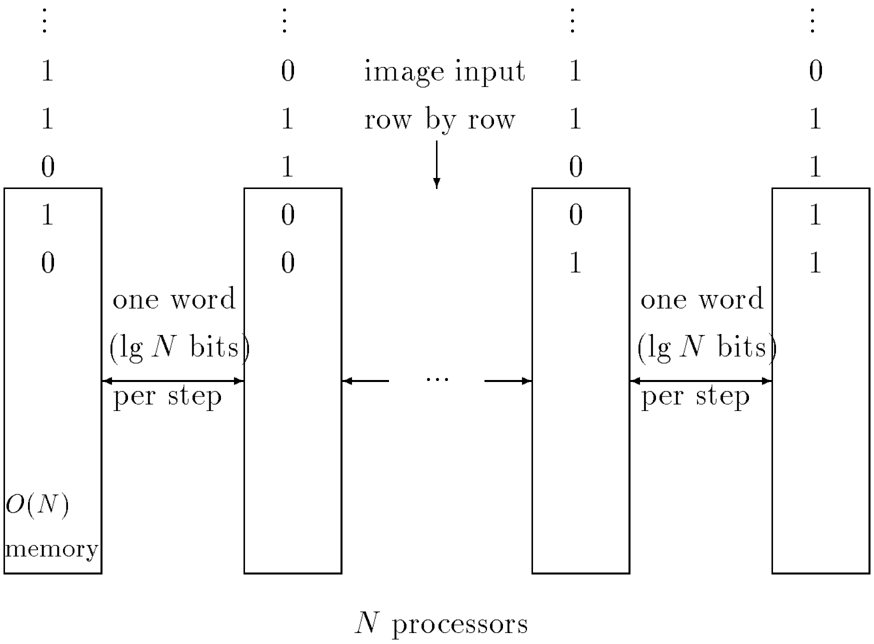

Figure 7.1: The SLAP architecture.

are connected.

Greenberg has presented a SLAP algorithm that labels connected components of an $N \times N$ image in $O(N \lg N / \lg \lg N)$ time as well as a variant likely to run in close to $O(N)$ time (which is a clear lower bound) on all or most images. Previous SLAP algorithms required $\Omega(N \lg N)$ time [4, 34], even with an unnatural "shuffled row-major" input ordering [4]. Various algorithms have been proposed to solve the problem in $O(N)$ time on a two-dimensional mesh of N^2 processors [14, 48, 51], but the drawbacks of such an approach are great. Even with $N = 128$, N^2 processors would greatly exceed the available resources on most existing parallel machines, whereas much larger images can easily be handled using a linear array configuration of processors. Other algorithms can yield even better than $O(N)$ time [13, 42, 50], but only with interconnection networks that are more complicated and, therefore, more costly. Related work has also been done by Schwartz, Sharir, and Siegel [55] and Dillencourt, Samet, and Tamminen [19]. They show that component labeling of a rectangular image can be performed in time linear in the number of pixels ($O(N^2)$ time for an $N \times N$ image) when the pixels of the image are read in scan line order. But they do not consider any type of multiprocessing solution.

7.2 Greenberg's Algorithm

It is assumed that the image has been input so that each processor holds one column of the image, as is natural on the SLAP. Also, the rows and columns are numbered from 0 to $N-1$, from top to bottom and left to right, respectively.

The top-level procedure involves computing what may be referred to as a *left-connected component labeling* and a *right-connected component labeling*. In a left-connected component labeling, two pixels in columns i and $i' > i$ receive the same label if they are in the same component of the subimage comprised of columns 0 through i'. In a right-connected component labeling, these pixels have the same label if they are in the same component of the subimage comprised of columns i through $N-1$. Once a left-connected component labeling and a right-connected component labeling (using different labels) have been found, it is easy to determine an overall connected component labeling as specified in Algorithm CC in Figure 7.2. Correctness follows from an observation also used in previous SLAP algorithms. That is, given a cut through the image and correct component labelings of the two subimages determined by the cut, the pixels bordering on the cut can be correctly labeled by performing

Algorithm CC

1 Find a left-connected component labeling of the image, placing the results for processor i in a local array *leftlabel*, such that *leftlabel*$[j]$ is the label for the pixel in column i and row j.

2 Do the same for right-connected component labeling, with results going into the array *rightlabel*.

3 Within each processor, in parallel, perform component labeling on the graph with nodes $\{\, \mathit{leftlabel}[j] \quad \forall j \,\} \cup \{\, \mathit{rightlabel}[j] \quad \forall j \,\}$ and edges $\{\, (\mathit{leftlabel}[j], \mathit{rightlabel}[j]) \quad \forall j \,\}$.

Figure 7.2: The top-level procedure for the $O(N)$ component labeling algorithm.

component labeling on just the pixels bordering on the cut. The approach of Algorithm CC can be viewed as fitting into the same framework by doubling each column and considering cuts running between the two copies of each column. We can maintain consistency of the various "border labelings" produced in the last step of Algorithm CC by imposing the rule that each component gets labeled with the least label seen on its pixels at this stage. A convenient initial labeling for each pixel is the position in column-major order (i.e., $iN+j$ for the pixel in column i and row j). With this initial labeling, the overall algorithm described in this section labels each component with the least initial label of its pixels.

The difficult part of performing component labeling is the implementation of steps 1 and 2 in Algorithm CC. The two steps are completely analogous, so the remaining discussion will consider only the left-component labeling of step 1.

Algorithm Left-Components

1 Perform union-find operations on the pixels within each processor (column) to place all pixels belonging to the same left-component into the same set.

2 For each pixel, find which set it belongs to.

3 Assign the appropriate left-component label to each set.

4 Assign the appropriate left-component label to each pixel.

Figure 7.3: The high-level structure of the left-connected component labeling algorithm.

The high-level structure of the left-component labeling procedure is specified in Algorithm Left-Components in Figure 7.3.

The most difficult part of the left-component labeling algorithm is step 1, which groups the pixels in each column into sets, with one set for each left-component intersecting the column. It uses the operations that define the familiar union-find problem [3]. The basic union-find operations may be expressed as follows:

1. Make-Set(x) creates a new set containing the single element x.

2. Find(x) finds the set that contains the element x.

3. Union(S, T) combines the sets S and T.

Tarjan [56] showed that any sequence of union-find operations can be performed with only slightly more than constant amortized time per operation, further details of these operations are described later. For now we will analyze the connected components algorithm under the assumption that each union-find operation can be performed in constant time.

To facilitate the operation of the LEFT-COMPONENTS procedure, we slightly augment the MAKE-SET and UNION operations to maintain two additional pieces of information for each set S, which we refer to as $adjnext[S]$ and $adjprev[S]$. The value of $adjnext[S]$ is some row index for which a 1-pixel of S is adjacent to a 1-pixel in the next column to the right; if no such index exists, $adjnext[S] = \text{NIL}$. Similarly, $adjprev[S]$ is a row index for which a 1-pixel of S is adjacent to a 1-pixel in the *previous* column if such an index exists. It is easy to see that only a constant amount of local computation and a constant number of nearest neighbor communications are required to maintain the *adjnext* and *adjprev* information during any union-find operation.

We can now use the procedure UNION-FIND-PASS of Figure 7.4 to express step 1 of Algorithm LEFT-COMPONENTS and the somewhat similar procedure LABEL-PASS in Figure 7.6 for step 3. Steps 2 and 4 of procedure LEFT-COMPONENTS just involve a sequence of N finds in each processor independently in parallel.

Greenberg proves the following lemma:

Lemma 1 *Algorithm* CC *computes the component labeling in $O(N)$ time under the assumption that unions and finds are constant time.* ■

It remains to detail the union-find operations, which have so far been treated as constant time. There is actually an $O(\lg N/\lg\lg N)$ time per operation implementation by Blum [8] that yields an $O(N \lg N/\lg\lg N)$ running time for Algorithm CC. Greenberg also argues, however, that we are likely to obtain time close to $O(N)$ in practice for CC using Tarjan's classic union-find

Algorithm UNION-FIND-PASS

for $j \leftarrow 0$ **to** $N-1$ **do** MAKE-SET(j) **endfor**

Set *outgoing*[i] to an empty queue.

for $j \leftarrow 1$ **to** $N-1$ **do**

    if $image[i, j-1] = image[i, j] = 1$

    then Call APPLY on the pair of rows $(j-1, j)$

    endif

endfor

if $i = 0$ **then** *incoming* $\leftarrow$ EOS **else** *incoming* $\leftarrow$ NIL **endif**

while *incoming* $\neq$ EOS **do**

    incoming $\leftarrow$ DEQUEUE(*outgoing*[$i-1$])

        $\langle$*returns* NIL *if empty queue*$\rangle$

    if *incoming* does not equal NIL or EOS

    then Call APPLY on *incoming*

    endif

endwhile

Enqueue EOS onto *outgoing*[i].

Figure 7.4: Pseudocode for processor i in the union-find pass of the left-component labeling algorithm. The dequeue on *outgoing*[$i-1$] represents a constant number of communications with the processor to the left; other variables are local. The two-dimensional array *image* contains the pixel values.

```
Algorithm APPLY(rowpair)
Set topset to FIND-SET of the top row in rowpair.
Set botset to FIND-SET of the bottom row in rowpair.
if topset ≠ botset then
    if adjnext is non-NIL for topset and botset
    then Enqueue the pair of rows (adjnext[topset], adjnext[botset])
    onto outgoing[i].
    endif
    UNION(topset,botset)
endif
```

Figure 7.5: Pseudocode for APPLY called from UNION-FIND-PASS

implementation [56] that achieves worst-case $O(\lg N)$ time per operation and a nearly constant average time per operation.

In Tarjan's implementation, each set is represented by a tree, with each element other than the root having a pointer to its parent. The MAKE-SET operation creates a representation of an element as a lone tree node. A FIND involves walks up a path in the tree and returning the root, which serves as the name of the set. For a UNION, we are given the roots of two trees (generally obtained by executing two finds), and we make one root the parent of the other by creating one new parent pointer. Tarjan showed that by incorporating the two heuristics of *path compression* and *weighted union*, any sequence of union-find operations can be executed in very nearly constant amortized time per operation [56]. Path compression is applied during finds; after we walk up a path to find the root for some node, we make all nodes that were encountered

Algorithm Label-Pass

```
for j ← 0 to N − 1 do
    S ← Find(j)
    if adjprev[S] = nil and label[S] is not set then
        label[S] ← iN + j
        Enqueue (label[S], adjnext[S]) onto outgoing[i].
    endif
endfor
if i = 0 then incoming ← eos else incoming ← nil endif
while incoming ≠ eos do
    incoming ← Dequeue(outgoing[i − 1])
        ⟨returns nil if empty queue⟩
    if incoming does not equal nil or eos then
        S ← Find(the row specified by incoming)
        label[S] ← the label specified by incoming
        Enqueue (label[S], adjnext[S]) onto outgoing[i].
    endif
endwhile
Enqueue eos onto outgoing[i].
```

Figure 7.6: Pseudocode for processor i in the labeling pass corresponding to line 3 of Algorithm Left-Components.

on the find path point directly to that root. Weighted union affects the choice as to which of the two given roots becomes the root of the combined tree; we point the root of the smaller set to the root of the larger set.

7.3 Simplified Algorithm and Simulations

This thesis investigates a slightly simpler variation of Greenberg's algorithm. Instead of performing left-component labeling and right-component labeling separately and merging the results, we can essentially complete our work by incorporating information from left-component analysis into the right-component analysis procedure. The basic approach involves beginning with a phase of passing information left to right across the image as in procedure UNION-FIND-PASS of Figure 7.4, which achieves the grouping of all pixels within each processor that belong to the same left-component. Then, with this degree of grouping already in place, we proceed to do a similar right-to-left pass. At this point, all the necessary grouping of pixels has been achieved within each column, and we just need to perform procedure LABEL-PASS of Figure 7.6. This approach should improve constant factors in the running time.

Figure 7.7 shows the top-level procedure SIMPLIFIED-CC for the simplified algorithm, and the procedure ADJUSTED-UNION-FIND-PASS called by it is shown in Figure 7.8. Since the right-to-left ADJUSTED-UNION-FIND-PASS uses the results of the left-to-right ADJUSTED-UNION-FIND-PASS as its starting point, the initialization activities contained in lines 1–7 of procedure UNION-FIND-PASS are moved out of procedure ADJUSTED-UNION-FIND-PASS and are

Algorithm SIMPLIFIED-CC

1 Each processor i performs MAKE-SET(j) (for $0 \leq j < N$), and UNION($j-1, j$) (for $1 \leq j < N$) if $image[i,j] = image[i,j-1] = 1$, to produce an initial set number for each pixel.

2 Call ADJUSTED-UNION-FIND-PASS.

3 Call a version of ADJUSTED-UNION-FIND-PASS that works right-to-left instead of left-to-right.

4 Call LABEL-PASS to assign labels to all pixels.

Figure 7.7: The top-level procedure for the simplified component labeling algorithm.

made as step 1 in procedure SIMPLIFIED-CC. After step 2, SIMPLIFIED-CC has achieved the grouping within each processor of all pixels belonging to the same left-component, and after step 3, the grouping within each processor of all pixels belonging to the same component has been achieved. The right-to-left version of ADJUSTED-UNION-FIND-PASS is obtained by simply changing "$i = 0$" to "$i = N-1$" and "$i-1$" to "$i+1$" and reversing the senses of "*adjnext*" and "*adjprev*".

A sequential simulation of the revised version of the algorithm was conducted through a C program. In this simulation, parallel time is counted in terms of the number of steps in the FIND operations; the remainder of the time can be proven to be $O(N)$ using the ideas in [25]. The input used for the simulation is a random image with a specified percentage of 1 pixels. For example if the percentage is 50 in a 10×10 image, then 50 distinct locations are selected at random and given pixel values of 1; the other pixel values are

```
Algorithm ADJUSTED-UNION-FIND-PASS
Set outgoing[i] to an empty queue.
for j ← 1 to N − 1 do
    r ← FIND(j)
    if r ≠ j and adjnext[r] ≠ NIL and adjnext[r] ≠ j and
    image[i + 1, j] = 1
    then Enqueue (adjnext[r], j) onto outgoing[i].
    endif
endfor
Include lines 8-15 of UNION-FIND-PASS
```

Figure 7.8: Pseudocode for processor i in the adjusted union-find pass of the simplified component labeling algorithm.

0.

Figure 7.9 gives an example of the input, output and intermediate results of the procedure on an 11×11 image.

Simulations were performed on random images for $32 \leq N \leq 512$ with 50% of pixels equal to 1. For each value of N, 25 runs were performed and the mean time (number of FIND steps as indicated above) and standard deviation were as shown in Table 7.1.

These data were analyzed using two curve fitting programs. First a Numerical Recipes [53] routine was applied directly to find the best fit to the mean time in the form $a + bN$ for constants a and b. The result was $a = -7.9081$ and $b = 10.5812$. (The standard deviations for the coefficients were $\sigma_a = 18.1350$ and $\sigma_b = 0.1399$, so that 0 is well within the likely range for a.) The chi-square

1	1	1	1	1	1	0	1	1	1	1
0	0	0	0	0	1	0	0	0	1	1
1	1	1	1	0	1	0	0	1	1	0
1	0	0	0	0	1	0	1	0	0	1
1	0	1	1	1	1	0	1	1	1	1
1	0	1	1	0	0	0	0	0	0	0
1	1	0	1	1	0	1	0	1	1	1
0	1	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	1	0	0
0	1	1	1	1	0	1	0	1	0	1
1	0	0	0	0	1	1	0	0	1	1

(a)

0	0	0	0	0	0	.	0	0	0	0
.	.	.	.	.	0	.	.	.	0	0
2	6	9	8	.	0	.	.	2	0	.
2	.	.	.	.	0	.	3	.	.	3
2	.	4	4	6	0	.	3	4	4	3
2	.	4	4	.	.	.	.	.	.	.
2	6	.	4	6	.	6	.	6	6	6
.	6	9	.	6	.	6	.	6	.	6
.	6	.	8	.	8	6	.	6	.	.
.	6	9	8	9	.	6	.	6	.	9
10	.	.	.	.	10	6	.	.	10	9

(b)

0	0	4	4	6	0	.	0	2	0	0
.	.	.	.	.	0	.	.	.	0	0
2	6	9	8	.	0	.	.	2	0	.
2	.	.	.	.	0	.	3	.	.	3
2	.	4	4	6	0	.	3	4	4	3
2	.	4	4	.	.	.	.	.	.	.
2	6	.	4	6	.	6	.	6	6	6
.	6	9	.	6	.	6	.	6	.	6
.	6	.	8	.	10	6	.	6	.	.
.	6	9	8	9	.	6	.	6	.	9
10	.	.	.	.	10	6	.	.	10	9

(c)

0	0	0	0	0	0	.	77	77	77	77
.	.	.	.	.	0	.	.	.	77	77
2	2	2	2	.	0	.	.	77	77	.
2	.	.	.	.	0	.	80	.	.	80
2	.	0	0	0	0	.	80	80	80	80
2	.	0	0	.	.	.	.	.	.	.
2	2	.	0	0	.	65	.	94	94	94
.	2	2	.	0	.	65	.	94	.	94
.	2	.	2	.	65	65	.	94	.	.
.	2	2	2	2	.	65	.	94	.	109
10	.	.	.	.	65	65	.	.	109	109

(d)

Figure 7.9: (a) Input image map. (b) The set numbers for all pixels after the left-to-right Adjusted-Union-Find-Pass. Note that only pixels belonging to the same left-component *and* lying in the same column are guaranteed to have the same set number at this point and that set numbers are reused in each column. (c) The set numbers for all pixels after the right-to-left Adjusted-Union-Find-Pass. Again, set numbers are reused in each column. (d) Output labels.

N	T	σ_T	N	T	σ_T
32	320.2	17.3	288	3062.1	92.6
64	681.7	44.1	320	3332.8	121.8
96	1033.7	56.0	352	3616.4	136.8
128	1430.1	64.5	384	4051.1	196.8
160	1718.9	80.8	416	4290.6	172.8
192	2131.9	120.6	448	4645.2	131.3
224	2332.9	65.3	480	4946.1	234.3
256	2806.1	110.6	512	5544.2	223.4

Table 7.1: Mean number of FIND steps (T) and standard deviations (σ_T) at various image sizes (N).

value was $\chi^2 = 6.6094$ and the Q value was 0.9487, where Q is the probability of getting the observed χ^2 value given the observed means and standard deviations. In this case, the Q value indicates a very good fit.

The second analysis was performed using code written at MIT that was supplied by Bradley Kuszmaul and attributed primarily to Eric Brewer. It is based on another Numerical Recipes routine, and it attempts to throw away terms that are insignificant. This program chose to eliminate the a term and produced the fit $10.54N$ with a high correlation of 0.997242.

7.4 Synopsis

This chapter has presented an improved algorithm for component labeling of an $N \times N$ image on an N-processor SLAP that is likely to achieve $O(N)$ running time in practice. A sequential simulation of this algorithm was conducted with provisions to measure parallel time. Results of the simulation on random images agree with the $O(N)$ time prediction.

Chapter 8

Conclusions

This thesis has investigated three issues in message routing and problem solving in multiprocessor networks: interconnection network comparisons, performance modeling of wormhole routed networks, and a fast algorithm for labeling connected components on a scan line array processor (SLAP). This chapter summarizes the major contributions and suggests further areas of investigation.

In the comparisons of interconnection networks we found that when bisection width and pin-out are used as constraints to equalize cost, the results obtained through simulation of uniform random messages are more optimistic than the theoretical upper bounds under which some slowdown could occur [32, Section 3.1.1]. In contrast to the bisection width and the pin-out constraints, use of the area constraint seems to be less favorable to the butterfly fat-tree and the fat-pyramid due to the simplified layouts used for them; further study is called for using the more area-efficient layouts in [24].

When packet routing is used on the networks under study, the *relative* performance among the networks is close to that under wormhole routing. Packetizing causes much higher latency across the board, while wormhole routed networks tend to saturate at lower load-rates because of the blocking property of wormhole routing.

We have also found that the butterfly fat-tree and fat-pyramid always show close performance to each other under the unit wire delay model, but with a little poorer showing by the fat-pyramid. Since the purpose for introducing the fat-pyramid was to achieve favorable performance in comparison to other networks in the context of non-unit wire delay, additional empirical simulations would be desirable with appropriate non-unit wire delay models.

In the performance modeling of wormhole routed networks, this thesis presented an extended model to compute the waiting times for multi-server channels under wormhole routing. We used multi-server queueing model to resolve the waiting times for redundant channels in the butterfly fat-tree, providing an analytical performance model for this network for the first time. The results obtained from the model were validated through comparisons with simulation results; excellent agreement were obtained for various network sizes, message lengths and load-rates. In addition, this thesis has derived an accurate model for predicting the performance of wormhole routed 2D-mesh networks. It has also presented empirical simulations and comparisons showing that the 2D-mesh network exhibits better performance than the torus.

With regard to finding connected components on a scan line array processor, this thesis studied an algorithm likely to run in close to $O(N)$ time for an $N \times N$ image with N processors. Simulations on random images show that the parallel time closely agrees with the $O(N)$ prediction. The most obvious open question is whether a connected components algorithm for the SLAP can be found that is provably $O(N)$ in the worst case.

Appendix A

Wire Density for Hypercube with Normal Layout

In this appendix we provide a detailed analysis of the wire density per row for the hypercube network with the two-dimensional normal layout proposed by Ranade and Johnsson [54]. Figure A.1 shows the normal layout of a hypercube with 64 processors. This layout has asymptotically optimal area for the hypercube network, and we determine here the exact wire density per row, defined as the maximum number of wires over all cut positions in a row or a column. Since the hypercube is symmetric both row-wise and column-wise, we will use the first row for the sake of clarity.

In the normal layout of the hypercube there are $\sqrt{N}$ nodes in a row, where N is the size of the network. We label the cut positions in the row as 1, 2, $\cdots$, $\sqrt{N}-1$ respectively. Among the interconnects in a row, there are links of distance 2^j, for each j with $0 \leq j < \log_4 N$. The number of distance 1 links at successive cut positions follows the repeating pattern 1,0,1,0,$\cdots$, and the number of distance 2 links follows the pattern 1,2,1,0,1,2,1,0,$\cdots$. Generally, the

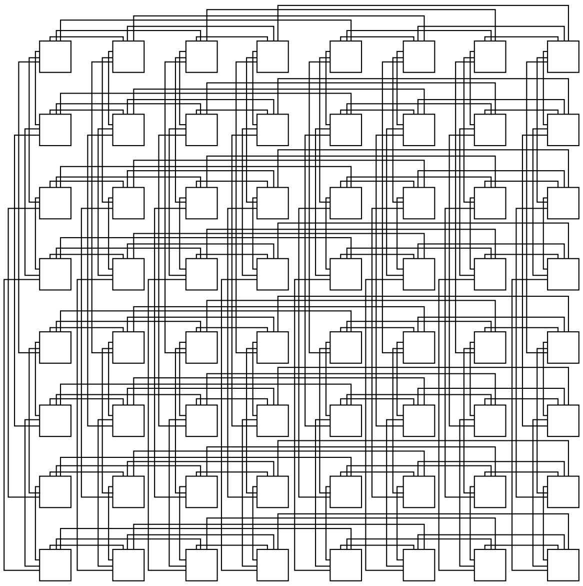

Figure A.1: Normal layout for a hypercube.

number of distance r links at each cut position follows the pattern 1,2,$\cdots$,$r-1$,r,$r-1$,$\cdots$,1,0,$\cdots$. By adding the number of links at each cut position and examining the totals, we can find the maximum number of wires over all cut positions. Denoting by d_p the maximum number of wires cut, where $p = \log_4 N$, we find the values 1, 2, 5, 10, 21, 42, $\cdots$ for d_1, d_2, $\cdots$. An interesting property of this series is that

$$\begin{aligned} d_p + d_{p-1} &= 2^p - 1, \\ d_{p-1} + d_{p-2} &= 2^{p-1} - 1, \\ \cdots \quad &\quad \cdots \\ d_2 + d_1 &= 2^2 - 1\ . \end{aligned} \tag{A.1}$$

By multiplying every other equations in the list by (-1) and add up both sides of the equation, we get

$$\begin{aligned} d_p + (-1)^p d_1 &= (2^p - 2^{p-1} + \cdots + (-1)^p 2^2) - \frac{1 + (-1)^p}{2} \\ &= \frac{2^{p+1} + 4(-1)^p}{3} - \frac{1 + (-1)^p}{2} \,. \end{aligned} \tag{A.2}$$

Noting that $d_1 = 1$, the above equation gives

$$d_p = \frac{2^{p+2} - (-1)^p - 3}{6} \,, \tag{A.3}$$

where $p = \log_4 N$.

Based on the ideas above, Greenberg has also derived a rigorous inductive proof that the maximum number of wires cut is as given by Equation A.3 and that the first cut position where the maximum is achieved is $\frac{2^p - (-1)^p}{3}$ [26].

Appendix B

Performance Model for Wormhole Routed Torus

This appendix provides the detailed performance model for the wormhole routed torus which has been used to obtain results for comparisons with the 2D-mesh in Chapter 6. The model is primarily based on that presented in [20] with modifications to include waiting times for injecting channels.

The total message rates are the same for each unidirectional link in the torus network, which is given as

$$\lambda_{net} = \lambda_{node}\frac{k-1}{2}\frac{k^n}{k^n-1}, \tag{B.1}$$

where n is the number of dimensions (n=2) and k is the width of each dimension. For two of the links in any given dimension, this is the message rate for single channels ($c_{i,0}$ and $c_{i,1}$ as illustrated in Figure 2.8); for other links this is the sum of the message rates on the high and low channels.

The total rate of messages entering a new dimension on any given link in that dimension is denoted by λ_{new}:

$$\lambda_{new} = \lambda_{node} \frac{k^{n-1}(k-1)}{k^n - 1}. \tag{B.2}$$

λ_{new} includes both arrivals form other dimensions, λ_{sw}, and new messages injected from the node. Note that for the 2D-torus this only applies to the lowest dimension (x-dimension). The message rate due to injections to the x-dimension at a node is $\lambda_{node}\frac{k-1}{k^2-1}$; therefore the message rate from the y-dimension to the x-dimension at each node is

$$\begin{aligned} \lambda_{sw} &= \lambda_{new} - \lambda_{node} \frac{k-1}{k^2-1} \\ &= \lambda_{node} \frac{(k-1)^2}{k^n - 1}. \end{aligned} \tag{B.3}$$

Denote the message rate on channel $c_{i,j}$ by λ_j. Then messages rates are determined as

$$\lambda_j = \begin{cases} \lambda_{net} & \text{for } 0 \le j \le 1 \\ \lambda_{node} \frac{k^{n-1}(k(k-1) - j(j-1))}{2(k^n - 1)} & \text{for } 2 \le j \le k-1 \end{cases} . \tag{B.4}$$

Note that λ_j is independent of dimension; channels in the x-dimension and the y-dimension has the same message rates if they are in analogous positions.

Due to the multiplexing on the physical channels, a flit requires more than one unit time to traverse a virtual channel. Neglecting the effects of other physical channels, the flit transfer time for virtual channel vc_1, which shares a physical channel with another virtual channel vc_2, is given by

$$1 + \lambda_{vc_2} \cdot msgl \ , \tag{B.5}$$

where the later term accounts for the rate at which flits of messages on vc_2 occupy the physical channel. Note, however, that the flit transfer time of a channel for a particular message is dictated by the maximum flit transfer time of all channels on the message's path. We therefore assume equal arrival rates on the virtual channels mapped to the same physical channel, i.e., $\lambda_{vc_1} = \lambda_{vc_2} = \lambda_{net}/2$, and approximate the flit transfer time of all channels in the network by

$$flit_{xfer} = 1 + \frac{\lambda_{net}}{2} msgl \ . \tag{B.6}$$

The raw service time (not including waiting times for subsequent channels) for a virtual channel is then given by

$$x_{base} = flit_{xfer} \cdot msgl \ . \tag{B.7}$$

Define a new function,

$$F_W(y,z) = \frac{yz^2\left(1 + \left(\frac{z - x_{base}}{z}\right)^2\right)}{2(1-yz)} \ . \tag{B.8}$$

Then the service times for the x-dimension channels are determined (using the techniques described in Chapter 5) as follows:

$$x_{j_0} = \begin{cases} x_{base} & j_0 = 0 \\ \frac{k+j_0-3}{k+j_0-1}\left(x_{j_0-1} + W_{j_0-1}\frac{\lambda_{new}}{\lambda_{j_0-1}}\right) + \frac{2}{k+j_0-1}x_0 & 1 \le j_0 \le k-1 \end{cases} \tag{B.9}$$

where the waiting times W_{j_0} are determined as

$$W_{j_0} = F_W(\lambda_{j_0}, x_{j_0}), \quad 0 \le j_0 \le k-1 \ . \tag{B.10}$$

Similarly the service times and waiting times for channels in the y dimensions are given as

$$x_{j_0,j_1} = \begin{cases} \frac{x_{base}}{k} + \frac{k-1}{k}\left(x_{j_0} + W_{j_0}\left(1 - \frac{\lambda_{sw}}{2\lambda_{j_0}}\right)\right) & \begin{cases} 0 \le j_0 \le k-1 \\ j_1 = 0 \end{cases} \\ \frac{k+j_1-3}{k+j_1-1}\left(x_{j_0,j_1-1} + \frac{\lambda_{new} W_{j_0,j_1-1}}{\lambda_{j_1-1}}\right) + \frac{2}{k+j_1-1} x_{j_0,0} & \begin{cases} 0 \le j_0 \le k-1 \\ 1 \le j_1 \le k-1 \end{cases} \end{cases}, \tag{B.11}$$

and

$$W_{j_0,j_1} = F_W(\lambda_{j_1}, x_{j_0,j_1}), \quad 0 \le j_0, j_1 \le k-1 \tag{B.12}$$

The service time and the waiting time for injecting channels are determined from the probabilities that a message is destined to the x-dimension and the y-dimension:

$$\begin{aligned} x^{inj}_{j_0,j_1} &= \left[x_{j_0} + \left(1 - \frac{\lambda_{node}}{\lambda_{j_0}}\frac{1}{k+1}\right) W_{j_0}\right] \frac{1}{k+1} \\ &+ \left[x_{j_0,j_1} + \left(1 - \frac{\lambda_{node}}{\lambda_{j_1}}\frac{k}{k+1}\right) W_{j_0,j_1}\right] \frac{k}{k+1}, \quad 0 \le j_0, j_1 \le k-1 \end{aligned} \tag{B.13}$$

with

$$W^{inj}_{j_0,j_1} = F_W(\lambda_{node}, x^{inj}_{j_0,j_1}), \quad 0 \le j_0, j_1 \le k-1 \;. \tag{B.14}$$

Finally the average message latency L is given by

$$\overline{L} = \frac{1}{k^2} \sum_{j_0=0}^{k-1} \sum_{j_1=0}^{k-1} \left(x^{inj}_{j_0,j_1} + W^{inj}_{j_0,j_1}\right) + (\overline{D} - 1) flit_{xfer} + 2 \;, \tag{B.15}$$

where $\overline{D}$ is the average message distance,

$$\overline{D} = \frac{k^2}{k+1} \ . \tag{B.16}$$

Bibliography

[1] S. Abraham and K. Padmanabhan. Performance of multicomputer networks under pin-out constraints. *Journal of Parallel and Distributed Computing*, pages 237–248, Dec. 1991.

[2] A. Agarwal. Limits on interconnection network performance. *IEEE Trans. Parallel and Distributed Systems*, 2(4):398–412, Oct. 1991.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[4] H. M. Alnuweiri and V. K. Prasanna. Optimal geometric algorithms for digitized images on fixed-size linear arrays and scan-line arrays. *Distributed Computing*, 5:55–65, 1991.

[5] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, 1990.

[6] P. Bay and G. Bilardi. Deterministic on-line routing on area-universal networks. *Journal of the ACM*, 42(3):614–640, May 1995.

[7] J. Beecroft, M. Homewood, and M. McLaren. Meiko CS-2 interconnect Elan-Elite design. *Parallel Computing*, 20:1627–1638, Nov. 1994.

[8] N. Blum. On the single-operation worst-case time complexity of the disjoint set union problem. *SIAM Journal on Computing*, 15(4):1021–1024, Nov. 1986.

[9] T. Bönniger, R. Esser, and D. Krekel. CM-5E, KSR2, Paragon XP/S: A comparative description of massively parallel computers. *Parallel Computing*, 21:199–232, 1995.

[10] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. iWarp: An integrated solution to high speed parallel computing. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 330–341. IEEE Computer Society Press, 1988.

[11] D. Chin et al. The Princeton engine: A real-time video system simulator. *IEEE Trans. Consumer Electronics*, 34(2):285–297, May 1988.

[12] http://www.cray.com/products/systems/crayt3e.

[13] R. Cypher, J. L. C. Sanz, and L. Snyder. Hypercube and shuffle-exchange algorithms for image component labeling. *Journal of Algorithms*, 10:140–150, 1989.

[14] R. Cypher, J. L. C. Sanz, and L. Snyder. Algorithms for image component labeling on SIMD mesh-connected computers. *IEEE Trans. Computers*, 39(2):276–181, Feb. 1990.

[15] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Computers*, 39(6):775–785, June 1990.

[16] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel and Distributed Systems*, 3(2):194–205, Mar. 1992.

[17] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986.

[18] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, C-36(5):547–553, May 1987.

[19] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253–280, Apr. 1992.

[20] J. T. Draper and J. Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *Journal of Parallel and Distributed Computing*, 23:202–214, 1994.

[21] A. L. Fisher. Scan line array processors for image computation. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 338–345, 1986.

[22] A. L. Fisher and P. T. Highnam. Real-time image processing on scan line array processors. In *Proceedings of the IEEE Computer Society Workshop on Computer Architectures for Pattern Analysis and Image Database Management*, pages 484–489, 1985.

[23] R. I. Greenberg. *Efficient Interconnection Schemes for VLSI and Parallel Computation.* PhD thesis, Department of Electrical Engineering

& Computer Science, Massachusetts Institute of Technology, Aug. 1989. MIT/LCS/TR-456.

[24] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Trans. Computers*, 43(12):1358–1364, Dec. 1994.

[25] R. I. Greenberg. Finding connected components on a scan-line array processor. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 195–202. Association for Computing Machinery, 1995.

[26] R. I. Greenberg. Personal communication, 1997.

[27] R. I. Greenberg and L. Guan. An empirical comparison of area-universal and other parallel computing networks. In *Proceedings of the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, pages 260–267, Sept. 1996.

[28] R. I. Greenberg and L. Guan. An empirical comparison of networks and routing strategies for parallel computation. In *Proceedings of the Eighth IASTED International Conference Parallel and Distributed Computing and Systems*, pages 265–269, Chicago, Oct. 1996.

[29] R. I. Greenberg and L. Guan. An improved analytical model for wormhole routed networks with application to butterfly fat-trees. In *Proceedings of the 1997 International Conference on Parallel Processing*, Aug. 1997. To appear.

[30] R. I. Greenberg and L. Guan. Modeling and comparison of wormhole routed mesh and torus networks. In *Proceedings of the Ninth IASTED International Conference Parallel and Distributed Computing and Systems*, Washington, D. C., Oct. 1997. To appear.

[31] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Randomness and Computation.* Volume 5 of *Advances in Computing Research*, pages 345–374. JAI Press, 1989.

[32] R. I. Greenberg and H.-C. Oh. Universal wormhole routing. *IEEE Trans. Parallel and Distributed Systems*, 8(3):254–262, Mar. 1997.

[33] F. T. Hady. *A Performance Study of Wormhole Routed Networks Through Analytical Modeling and Experimentation.* PhD thesis, University of Maryland Electrical Engineering Department, 1993.

[34] D. Helman and J. JáJá. Efficient image processing algorithms on the scan line array processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):47–56, Jan. 1995. Earlier version in Proceedings of the 1993 International Conference on Parallel Processing.

[35] P. Hokstad. Approximations for the M/G/m queue. *Operations Research*, 26(3):510–523, 1978.

[36] T. Hoshino. *PAX Computer: High-Speed Parallel Processing and Scientific Computing.* Addison-Wesley, 1989.

[37] Intel. Paragon XP/S product overview, 1991.

[38] J. Kim and C. R. Das. Hypercube communication delay with wormhole routing. *IEEE Trans. Computers*, 43(7):806–814, July 1994.

[39] J. H. Kim and A. A. Chien. Network performance under bimodal traffic loads. *Journal of Parallel and Distributed Computing*, 28:43–64, 1995.

[40] L. Kleinrock. *Queueing Systems*, volume I. John Wiley, 1975.

[41] S. Knight et al. The Sarnoff engine: A massively parallel computer for high definition system simulation. In *Proceedings of Application Specific Array Processors*, pages 342–357, 1992.

[42] V. K. P. Kumar and M. M. Eshaghian. Parallel geometric algorithms for digitized pictures on mesh of trees. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 270–273, 1986.

[43] F. T. Leighton. *Complexity Issues in VLSI.* MIT Press, Cambridge, Massachusetts, 1983.

[44] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[45] F. T. Leighton and A. L. Rosenberg. Three-dimensional circuit layouts. *SIAM Journal on Computing*, 15(3):793–813, Aug. 1986.

[46] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, C-34(10):892–901, Oct. 1985.

[47] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S.

Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the connection machine CM-5. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285. Association for Computing Machinery, 1992.

[48] S. Levialdi. On shrinking binary picture patterns. *Communications of the ACM*, 15(1):7–10, Jan. 1972.

[49] Meiko World Incorporated. Computing surface CS-2HA technical description, 1994.

[50] R. Miller and Q. Stout. Data movement techniques for the pyramid computer. *SIAM Journal on Computing*, 16(1):38–60, 1987.

[51] D. Nassimi and S. Sahni. Finding connected components and connected ones on a mesh-connected parallel computer. *SIAM Journal on Computing*, 9(4):744–757, 1980.

[52] C. Peterson, J. Sutton, and P. Wiley. iWarp: A 100-MOPS LIW microprocessor for multicomputers. *IEEE Micro*, pages 26–29, 81–87, June 1991.

[53] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[54] A. G. Ranade and S. L. Johnsson. The communication efficiency of meshes, boolean cubes and cube connected cycles for wafer scale inte-

gration. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 479–482, 1987.

[55] J. T. Schwartz, M. Sharir, and A. Siegel. An efficient algorithm for finding connected components in a binary image. Technical Report 154, Department of Computer Science, NYU, Feb. 1985. Revised July, 1985.

[56] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, Apr. 1975.

[57] C. D. Thompson. Area-time complexity for VLSI. In *Proceedings of the 11th ACM Symposium on Theory of Computing*, pages 81–88. ACM Press, 1979.

[58] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1980.