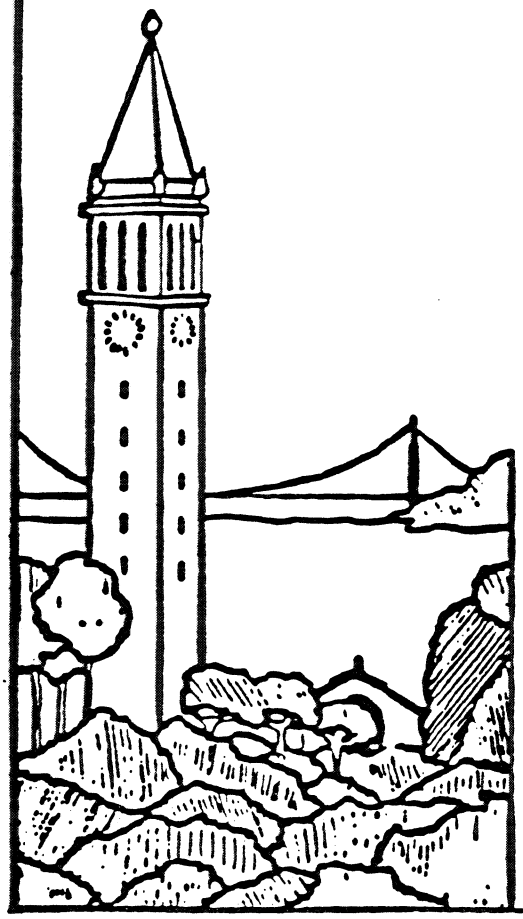
Interconnection Network Design Based on Packaging Considerations

Mandayam Thondanur Raghunath



Report No. UCB/CSD 93/782
December 1993

Computer Science Division (EECS)

University of California

Berkeley, California 94720

Interconnection Network Design Based on Packaging Considerations

by

Mandayam Thondanur Raghunath

B. Tech. (Indian Institute of Technology, Madras) 1987
M. S. (University of California at Berkeley) 1988

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Abhiram G. Ranade, Chair
Professor David E. Culler
Professor J. George Shanthikumar

1993

Interconnection Network Design
Based on Packaging Considerations

by

Mandayam Thondanur Raghunath

Abstract

Interconnection Network Design Based on Packaging Considerations

by

Mandayam Thondanur Raghunath
Doctor of Philosophy in Computer Science
University of California at Berkeley
Professor Abhiram G. Ranade, Chair

An important problem in building large scale parallel computers is the design of the interconnection network. The network affects the performance, cost, scalability, and availability of the parallel computer. In this dissertation, we examine in detail the problem of interconnection network design for a large scale parallel machine. Our objective is to design networks that achieve a high performance for a low cost.

In general, the cost of a network is a complex function of a wide variety of parameters and is difficult to compute. We first develop a packaging model to characterize network costs. Our model is based on the fact that most large scale machines have to be packaged in a hierarchical fashion. We argue for a hierarchical design strategy, where the design of the network proceeds in levels corresponding to the levels of the packaging hierarchy. We evaluate several network designs using analysis and detailed simulations of random traffic. We identify families of networks (product of complete graphs, high-degree de Bruijn networks) that we believe to be useful for multi-level packaging technologies. Our results also indicate that making the networks denser at the lower levels of the packaging hierarchy has a significant positive impact on the overall performance, even when the higher levels use a sparser interconnect.

We also characterize network designs in terms of scalability, dividing network families into three broad categories of scalability based on the flexibility with which the networks can be scaled. The three categories illustrate the trade-offs between hardware costs and scaling flexibility. We provide quantitative evaluations of these trade-offs.

Many of the interesting networks, namely those that provide high performance for a low cost, also have the disadvantage of being vulnerable to faults because they have a single path between any pair of processors. We devise a scheme to tolerate faults in such networks. The scheme is simple to implement and allows the network to tolerate a large number of faults with a slight degradation in performance.

Professor Abhiram G. Ranade
Dissertation Committee Chair

Acknowledgements

It is a pleasure to acknowledge the years of guidance, support, and encouragement that I received from my thesis advisor, Abhiram Ranade. I am extremely grateful to Abhiram for providing my research with a sense of direction and helping me develop a better understanding of the research issues involved. I am also grateful to him for offering me sufficient challenge and carefully evaluating my work.

I would also like to express my gratitude to David Culler and George Shantikumar for reading through this lengthy dissertation. Their detailed comments helped improve the content and presentation of this thesis.

I am also grateful to all the instructors of the various courses I have attended at U. C. Berkeley and at I. I. T. Madras, for enabling me to acquire sufficient knowledge to carry out independent research. U. C. Berkeley, gave me the opportunity to take a wide range of classes, not just in Computer Science, but also in Sanskrit, Music, etc.

I would also like to thank Howard Ho, and many others at the IBM Almaden Research Center, for comments and suggestions relating to my research.

Many of my fellow graduate students at Berkeley have helped me academically, and also in making my stay at Berkeley enjoyable and fun. I would especially like to thank Bob Boothe, Jeff Rothman, and Brian O'Krafka for many technical discussions relating to my thesis. Besides helping me develop and evaluate research ideas, and commenting on drafts of papers, Bob Boothe and Jeff Rothman helped relieve the occasional monotony of work and moments of depression or cynicism with interesting discussions on a wide variety of topics such as vegetarianism, US foreign policy, sports, etc. I also wish to thank the numerous friends I developed while at Berkeley; my senior students who helped me settle down and get to know Berkeley, my roommates and others who braved my culinary attempts, and many others who joined me in various activities; games, movies, concerts, etc.

I would also like to thank Kathryn Crabtree and the other members of the administrative staff of the CS division for making the rather difficult bureaucracy of the university administration appear friendly and cordial.

Finally, I also wish to thank my parents for bringing me up with love and affection, for giving me a strong sense of determination and self-esteem, and for providing me with the best opportunity in education.

This research was supported in part by the Air Force Office of Scientific Re-

search (AFOSR/JSEP) under contract F49620-90-C-0029, National Science Foundation Grant Number CCR-9005448, and a fellowship from IBM Corp. Computing resources were provided by NSF Infrastructure Grant number CDA-8722788. Their support is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid technological advances and increasing demands for computer power have ushered in large scale parallelism. Several machines consisting of large numbers of processors have been built, either commercially or as research vehicles [GGK+83, Hil85, PBG+85, BBN86, H+86, Int86, LT88, A+90, R+90, Nic90, ND90, L+91, TMC91, Ken92, FIR93, DKN93, KS93]. Large scale parallelism has the potential for providing quantum jumps in computing power; complex computational problems that were once thought to be intractable, now seem feasible. However, there are a number of open research issues that need to be solved before the power of large scale parallelism can be conveniently applied to solve general problems.

One central issue is the design of the interconnection network that handles the communication between processors. Since parallel programs typically exchange large amounts of data between processors during the course of their execution, substantial design effort is usually directed towards the design of the interconnection network. The network takes up a significant fraction of the total cost; is often the hardest part of the system to engineer and for many applications determines the final performance.

The design of the interconnection network is also important from the point of view of making parallel machines easy to program. Parallel machines are generally regarded as being difficult to program and this difficulty stems from a variety of factors, one of which is the time spent in communication. When communication is slow, the programmer has to expend a great deal of effort in carefully partitioning the data among processors so as to reduce the amount of data movement. By designing a fast communication network that reduces the time spent in communication, we can simplify this aspect of parallel programming.

Fast interconnection networks also permit the programmer to use higher level programming models which provide programming simplicity at the cost of increased communication.

Since the network is a key aspect of a parallel machine, the problem of designing efficient interconnection networks has received much attention from researchers. However there is no apparent consensus: each of the different parallel machines currently on the market has a completely different interconnection mechanism. For instance, while the CM - 5 uses a fat-tree topology, the Intel Paragon uses a two dimensional mesh and the KSR - 1 uses a ring-of-rings interconnect. Further, the machines recently announced by both Thinking Machines and Intel have an entirely different network organization compared to their previous models. In both cases, they have abandoned the hypercube network topology in favor of the above mentioned topologies.

The lack of consensus clearly indicates that designing efficient networks is still an open problem. Some of the reasons for this lack of consensus are as follows:

1. *Modeling network cost:* In general, modeling the cost of a network is a difficult problem. Existing models of network cost (discussed in section 1.5) are often inadequate to model the complex costs associated with multi-level packaging hierarchies that most large scale network designs are forced to contend with. In the absence of a good way to model network costs, it is difficult to evaluate the cost-performance trade-offs of different network designs. While one design may promise greater performance than another (on the basis of simulation studies, for example), it is difficult to determine their relative costs until they are actually built. As a result, it is difficult to arrive at a consensus on whether one network is better than another.

2. *Traffic patterns:* We do not yet have a clear understanding of the characteristics of the communication traffic seen in parallel machines. Some applications (e.g. matrix multiplication) have communication traffic patterns that are extremely regular, while others (e.g. graph algorithms) have patterns that display no such regularity and may also vary with time. There is some disagreement on whether network designs should be optimized for regular traffic patterns. While some may argue that regularity in communication traffic is indeed the common case, and should be speeded up, others argue in favor of presenting the programmer with a uniform view of the network where any processor can send data to any other processor, with equal ease.

3. *Scalability:* Parallel machine manufacturers usually want to be able to supply machines of different sizes, because customers have different computing requirements. It is common for manufacturers to target a wide range of systems, starting from low-cost entry-level systems, all the way to a high-end system consisting of thousands of processors. It is also desirable to provide customers with the ability to incrementally upgrade their machines. Therefore, networks must be designed for *scalability*. However, there is no consensus on a formal definition of scalability, and there is no definitive method of characterizing network designs in terms of scalability. While some manufacturers may abandon a design because they consider it to be unscalable, others may regard it as being scalable.

4. *Fault-Tolerance:* Large scale networks typically consist of thousands of components and it is useful to build networks that are resilient to faults in at least a small fraction of their components. The extent of fault-tolerance required, depends on the reliability of the individual components, the time required to service faulty components, and the desired reliability of the entire machine. In many cases, neither of these three aspects is easy to quantify. There are also several different approaches to achieving fault-tolerance. One approach to providing fault-tolerance is to augment the network with extra components that are brought into use when faults occur. In a scheme of this type, the stand-by components are used to replace the faulty-ones, effectively retaining the structure and performance of a fault-free network. Another approach is to re-route messages around faulty components instead of replacing them. While the first approach usually implies an increase in cost without degradation in performance, the second approach might be cheaper but suffers performance degradation when faults occur. It is not clear as to which of these two approaches is better because the trade-offs between fault-tolerance, cost, and performance are not well understood.

In this dissertation we attempt to clarify some of the above issues and seek to design efficient networks for large scale parallel machines. By "large scale" we mean a machine size of around 1000 uniprocessors, with each processor as powerful as a state-of-the-art work-station.

The set of questions that come up while designing networks range from the very high-level to the very mundane, but many of these questions can make or break a network design. One of the important high-level questions we are faced with is: "What topology

should we use?". For example, we may have a choice between a two dimensional mesh and a butterfly. We may consider the butterfly to be better because the worst case distance between the processors is shorter in the butterfly. However, the mesh has multiple paths between any pair of processors and may have better fault-tolerance properties. It is not clear which of the two topologies should be chosen. It is also conceivable that there may exist other topologies that have the advantages of both the mesh and the butterfly, and therefore, are better than both of them.

Even after we resolve the high level questions, we are still left with a number of lower level details. For example, let us consider the case where we have decided to build a network with processors on one side connected to memory units on the other using a log-depth multi-stage switching network with some specific communication bandwidth. Even now, there are a number of unresolved questions. Should we use a simple butterfly as shown in figure 1.1(a) or a dilated butterfly as shown in figure 1.1(b)? Since both networks are required to have the same communication bandwidth, the widths of channels in the dilated butterfly should be half the widths of the channels of the simple butterfly. Both networks have nominally the same bandwidth and very similar costs since both networks have the same number of wires going between any pair of nodes. How do we discriminate between the two networks? Another possibility is shown in figure 1.1(c). In this network we have a butterfly with half the number of rows, and we have connected two processors to each input of the butterfly using a tree of height 1. In order for this network to have the same bandwidth as the two previous networks, the channel widths in this network have to be double that of the simple butterfly. How does this network compare with the ones shown in figure 1.1(a) and (b)? How do these three compare with the network shown in figure 1.1(d), a butterfly of higher radix, and fewer stages of switching? We may also consider yet another possibility shown in figure 1.1(e). Here we use fewer memory units than processors. To achieve the same bandwidth as the other networks, we have to increase the channel widths to the memory. How does this network compare with the others?

The answers to many of these questions are more complex than they appear superficially because networks must be packaged in a hierarchical fashion, and we may be faced with a similar set of questions at each level of the packaging hierarchy. We are also faced with the question of whether it is advantageous to use different network architectures at different levels of the packaging hierarchy. In other words, are hybrid network architectures better than uniform ones? For example, is there any benefit to using different channel

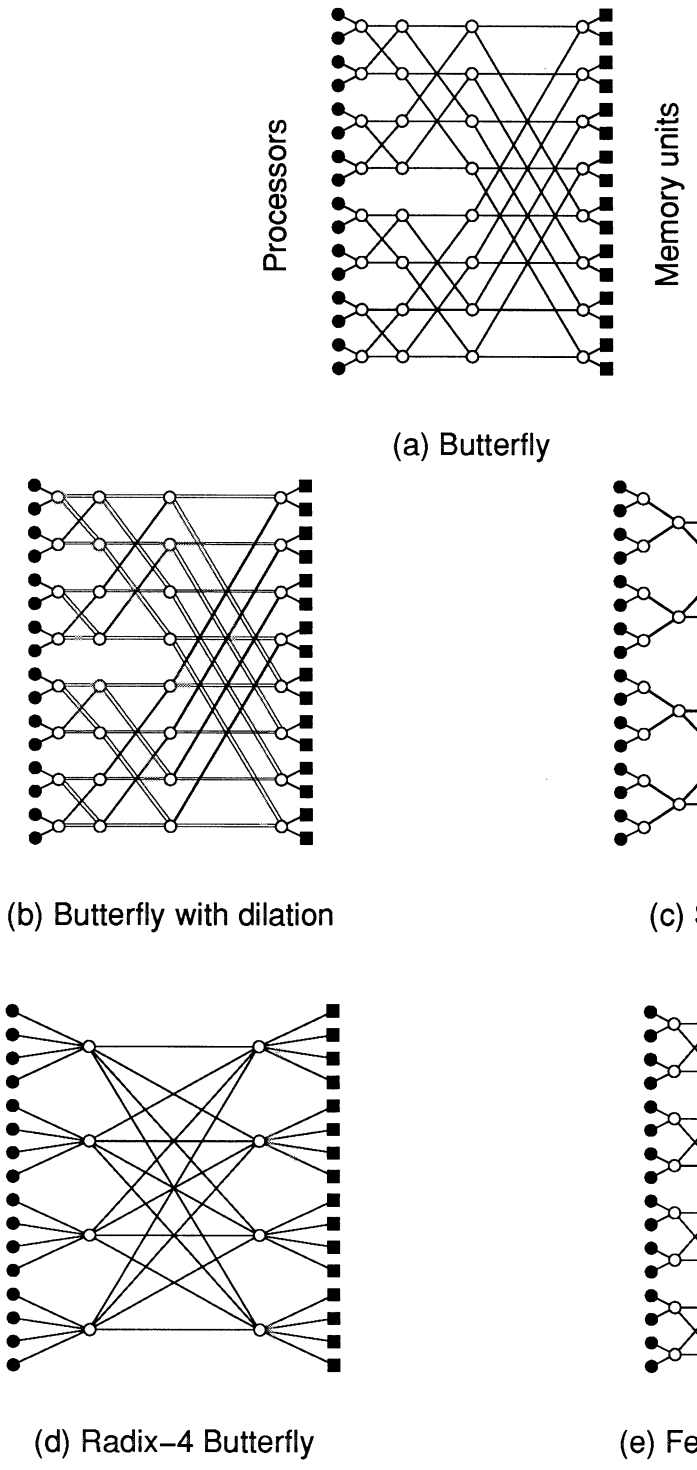


Figure 1.1: Butterfly Choices

widths at different levels of the hierarchy?

The objective of this dissertation is to develop a systematic method for dealing with questions of the type raised above. We believe that many of these decisions are currently made in a somewhat ad-hoc manner. Instead, we would like to formalize the decision process. We would like develop a systematic way of exploring the search space of design choices to enable us to pick out the promising ones while discarding the uninteresting choices. In order to motivate the exploration of the search space, we shall pursue the goal of designing the *best* network. We define the *best* network to be one that achieves the maximum performance for a fixed cost, or equivalently one that achieves a fixed performance at minimum cost. It is also important for the *best* network to possess good scalability and fault-tolerance properties.

Although trying to design the *best* network is a good way to drive the design process, we shall see that it is not always possible to obtain a design that can be proved to be the best. In some cases, we shall be able to prove that a particular design is optimal, but in many other cases we shall only be able to demonstrate that one set of designs is better than all other designs, thereby implying that the best network belongs to this set. Determining which element of the set is the best, will require knowing more details of the packaging technology than we are willing to consider as part of this thesis.

In this chapter we consider the above issues in greater detail (Sections 1.1, 1.2, 1.3 and 1.4), survey some of the related work (Section 1.5), list the main contributions of this dissertation(Section 1.6), and finally conclude with an overview of the remaining chapters.

Part of the work described in this thesis has already been presented in [RR90] and [RR93].

1.1 Modeling Network Cost

Modeling the cost of a large scale network is a difficult problem. The cost of such a network depends on the amounts of various packaging resources used to build the network. Since large scale networks are typically packaged in a hierarchical fashion, we have to take into account a large number of different packaging resources at each level of the packaging hierarchy in order to accurately model network costs. For example, under current packaging technology, a typical hierarchy would consist of chips, boards, and racks. The network is first partitioned so that it fits into some number of racks with wires going between them.

The contents of a rack are further partitioned into smaller units of the packaging hierarchy. To compute the cost of the network, we have to take into account many resources, such as, the number of racks, the number of wires that go between racks, number of wires connecting to a board, the pin-counts of the chips, etc.

Existing cost models deal primarily with the lowest levels of the hierarchy, and a better model is required to characterize multi-level packaging hierarchies. We develop a model of packaging technology that uses a number of parameters to explicitly characterize the constraints and costs associated with multi-level packaging hierarchies. The parameters of our model can be used to specify limits on pin-counts, bounds on layout areas, cabling restrictions, etc.

Although the model is general enough to characterize a wide range of current and future packaging technologies, we choose not to base our network designs on a specific packaging technology, because packaging technology continues to evolve with time; with new packaging methods being discovered, costs of existing technologies being driven down by economies of scale, etc. Instead we define three generic models, that capture fundamental technology trends that are likely to be valid even in the future, and study various network designs based on these generic models.

The purpose of defining generic models, is to obtain results that are likely to apply to a wide range of packaging technologies instead of being specific to just one technology. In our generic models, we leave some of the parameters unspecified; by appropriately fixing these parameters, the generic model can be used to characterize a variety of real technologies. Under each generic model, we broadly classify networks into *interesting* and *uninteresting* ones. A network is considered *interesting* if it is better than all other networks for *some* instantiation of the unspecified parameters. A network is *uninteresting* if there are other networks that are better, for *all* instantiations of the unspecified parameters. We provide detailed evaluations of the *interesting* networks.

1.2 Traffic Patterns

As mentioned earlier, applications can be broadly classified into two categories: those that have regular communication patterns and those that do not. When the communication patterns of an application are regular, it may be possible to transform this regularity into *network locality*, where the messages predominantly travel between proces-

sors that are proximal in the network topology. Assuming that all the communication traffic in the network has network locality can usually simplify the task of designing efficient networks. However, there are a number of reasons why assuming network locality may not always be justified.

First, a large number of applications have communication patterns that are either irregular or keep changing with time, and for such applications achieving network locality is either impossible or overwhelmingly difficult. Second, for many applications, algorithmic optimizations that reduce the running time, generally destroy any regularity in communication patterns that was present before the optimizations were performed. Third, when parallel machines are used to solve problems that are much larger than the number of processors (as is usually the case), it is difficult to achieve network locality. Fourth, exploiting network locality usually requires either the programmer or the compiler to know the precise manner in which the processors are interconnected, and this may not always be desired. Fifth, many high-level programming constructs attempt to hide the details of the interconnection network and usually rely on the ability to communicate data from any processor to any other processor. Finally, the presence of network faults generally implies that the physical interconnect of a machine changes with time, making it difficult to write programs that exploit network locality.

For all of these reasons, we shall assume that the communication traffic pattern is not characterized by network locality. It is common practice to model such traffic using randomly distributed communication traffic. In this dissertation, we make a similar assumption in our evaluations of network performance. We evaluate networks using simulation, and for the purposes of simulation, we assume a shared-memory model, but we believe that our results are equally applicable to other models of inter-processor communication where the messages are evenly distributed over the entire network.

1.3 Scalability

Scalability is a property that is often used to characterize interconnection networks, but is not very well understood [Hil90, NA91]. There is considerable disagreement about which networks are considered scalable and which ones are not. One of the reasons that prompted some manufacturers to move from the hypercube topology to a two or three dimensional mesh for interconnecting processors, was the opinion that hypercubes were not

scalable because the degree of each node was not constant. Constant node degree is one of the desirable properties from the point of view of scalability, because we can easily build networks of various sizes using the same nodes. On the other hand, it is not clear that meshes are scalable either, because their bisection bandwidth does not grow linearly in the number of processors. The bisection bandwidth of a machine is a rough estimator of its performance and we usually want network performance to scale linearly in the size of the machine. In the absence of a formal definition of scalability, it is difficult to decide whether a particular network design is scalable or unscalable.

In this dissertation we provide a formal definition of scalability, in terms of how the performance and the cost of a network vary as we increase the number of processors. Briefly, we consider a network to be scalable if its performance meets a certain lower bound, and its cost is lower than a certain upper bound, over a specified range of machine sizes.

We also classify networks into three broad scalability categories depending on the fraction of the machine that remains unaltered as the size of the machine is changed. The larger this fraction, the greater the flexibility in scaling the machine. We analyze some of the network designs studied in this dissertation in terms of scalability, and demonstrate how to extend these designs as per each of the three broad scalability categories. We shall see that greater scaling flexibility is usually accompanied by lower performance or greater cost. We provide a detailed analysis and quantification of the trade-offs between scalability, cost and performance in Chapter 7.

1.4 Fault-Tolerance

Designing networks with the ability to tolerate faulty components is becoming increasingly important as the networks increase in size and complexity. There is an extensive amount of work in the area of fault-tolerant network design. A large fraction of this work is concerned with augmenting the network by adding extra components that are basically idle until failures occur. We believe that this approach is wasteful of resources, primarily because failures are usually rare (but not non-existent). Hardware components generally tend to be fairly reliable after the initial developmental errors are fixed and the common case is for networks to be either fault-free or have a small number of faults.

We believe that a better approach to fault-tolerance is to devise mechanisms to use the existing network to route messages around faulty components. This approach results

in a reduction in performance when faults occur, since messages share fewer pathways in the network. However, if the number of faults is small, and the fault-tolerance scheme is efficient, the performance impact can be maintained at a minimal level. We present one such fault-tolerance scheme in this dissertation.

Most of the networks considered in this thesis have a single path from any input to any output. When a path is affected by a fault, we send messages using two passes over the network. In the first pass, the message travels from the source processor to a relay processor and in the second pass the relay processor sends the message to its appropriate destination. We select the relay processors in a manner that minimizes the performance impact of the faults.

1.5 Related Work

A major portion of this dissertation is concerned with modeling the cost of networks and designing efficient networks under our models of network cost. Accordingly, we classify the existing related research into two broad areas: cost modeling and network design. Related work on fault-tolerance is presented in Chapter 8, along with our proposed fault-tolerance scheme.

1.5.1 Cost Models

The two cost models that have been formally defined and theoretically explored in detail are the VLSI grid model and the Pin-requirements model. In the VLSI grid model, the layout area of a network is used as the metric of the network cost. This model was developed by Thompson [Tho80] primarily for circuit layouts in VLSI. In this model, layouts are assumed to conform to a regular grid consisting of unit squares. Each unit square contains either a node (defined as a transistor or a small cluster of transistors), or a single wire, or a wire crossing. Since the model deals with VLSI layouts, a natural cost metric under this model is the number of unit squares (= area) required by a layout of the network. As opposed to the grid model, the Pin-requirements model focuses on I/O limitations rather than on area. In the Pin-requirements model [Cyp90], the network is assumed to be partitioned into some number of chips. In this model, the maximum pin-count per chip is used as the cost metric.

We define three generic models (Chapter 2) that explicitly characterize packaging hierarchies to different levels of detail using cost functions that are more comprehensive. The first of our generic models is similar to the Pin-requirements model, but is applied at the top level of the packaging hierarchy. The other two generic models are more detailed, and account for other packaging resources. Since our models are more representative of the costs in the presence of packaging hierarchies, we believe that results have a greater applicability than the results under the two previous models.

1.5.2 Designing Networks

In this thesis, we design networks that achieve high performance for a low cost as specified by our cost model. There have been similar design studies under the two cost models mentioned above. In many of the existing design studies, performance is measured under the assumption of randomly distributed communication traffic. A similar assumption is made in this thesis.

Lower bounds

One way of tackling the problem of designing efficient networks is to prove theoretical lower bounds on the cost or the ratio of cost to performance. Then, if we are able to demonstrate that a network is close to the lower bound, we can claim that this network is one of the *better* networks.

Some of the initial work under the VLSI model and the Pin-requirements model consisted of proving such lower bounds. For example, under the VLSI grid model, Thompson [Tho80] showed that the area of a network is lower bounded by the square of its bisection width[1]. Other researchers have proved tighter lower bounds on the area of various network topologies [Lei82, BL84, Lei84, Ull84]. Since the cost of a network and its area are synonymous in this model, these lower bounds are essentially lower bounds on network cost. In addition, Thompson also proved lower bounds of the form AT^2, where A is the area of the chip, and T is the time taken to solve a specific problem (e.g. sorting). These bounds represent a trade-off between the cost of the chip and its performance (represented by $1/T$). Similarly, in the Pin-requirements model, Cypher [Cyp90] proved lower bounds on Q, the

[1]The bisection width of a network is the minimum number of wires that need to be cut in order to divide the network into two equal parts.

maximum number of pins per chip (= cost) for a number of commonly used network topologies. He also proved lower bounds on the product $Q * C * T$, where C is the number of chips and T is the time taken to solve a specific problem.

Universal Networks

Another approach to designing efficient interconnection networks is to demonstrate the existence of *universal* networks. A universal network is one that can emulate any other network of the same cost with minimal difference in performance. Based on the VLSI grid model, Leiserson demonstrated that a class of topologies called fat-trees [Lei85] are universal, in that they can simulate (off-line) any other network with the same area with only a polylogarithmic slowdown. Greenberg [Gre90] later extended Leiserson's work and proved the universality of a similar class of networks called fat-pyramids. The fat-pyramid network has the additional property that any network occupying the same area, can be mapped onto the fat-pyramid in a manner which ensures that the path lengths in the fat-pyramid are not much greater than the path lengths of the original network. This property makes the fat-pyramid applicable under many different assumptions with respect to wire delays. While universal networks are interesting from a theoretical point of view, we approach the network design problem from an engineering viewpoint where even constant factors of difference in performance are important, and polylogarithmic slowdowns are unacceptable. We are also interested in designing networks under cost models that are more detailed than the VLSI grid model.

Network comparisons

A third approach to designing efficient networks is to explicitly define several interconnections and compare them in terms of cost and performance. In this thesis we primarily take this approach, but we shall also prove that some of our networks satisfy theoretical bounds on performance and cost.

A large number of different interconnection topologies have been proposed for parallel machines [Fen81, AG89], but only a few classes of topologies have received substantial attention with respect to comparative evaluations of performance. One such class of topologies is the family of k-ary n-cubes. Rings, multi-dimensional meshes and hypercubes belong to this class of networks. William Dally [Dal86] analyzed the cost-performance trade-offs

of this class of networks under Thompson's VLSI grid model, where the cost of a network is its layout area. Different network topologies can be made to occupy the same area by suitably varying the channel widths so that all the networks have the same bisection width. Dally's analysis shows that when bisection width is held constant, all networks have the same bandwidth for delivering messages with random destinations. However, he argues that low dimensional networks are better since they have lower latencies [Dal87].

Abraham and Padmanabhan [AP90] analyzed the k-ary n-cube family of networks under the Pin-requirements model, where the cost metric is the pin-count per node. They compared various networks of this family under the assumption that the pin-count per node was held constant. In contrast to Dally's results, their results indicate that high dimensional networks are better because they can provide higher bandwidths for random communication.

Agarwal [Aga91] conducted a similar analysis of the same class of networks under three different constraints: constant bisection width, constant channel width and constant pin-count. The first two constraints correspond to the VLSI model and the Pin-requirements model respectively, but Agarwal's analysis is based on a more comprehensive model of node and wire delays. His results indicate that the relative ranking of networks is highly sensitive to which cost model is used.

Another class of networks that has been considered by many researchers are multi-stage switching networks such as butterflies, omega networks, delta networks, etc. We can show that most of these networks are isomorphic to one another and are therefore equivalent [WF80]. One interesting variant of the butterfly network that was proposed recently is the multibutterfly network [Upf89, LM89, LLM90]. Konstantinidou and Upfal [KU91] compared the performance of the butterfly network to the multibutterfly network. Their results indicate that the performance of the multibutterfly network is comparable to the butterfly. However, the multibutterfly has far superior fault-tolerance properties in comparison to the butterfly network.

Dandamudi [Dan88] compares networks under a cost model that is different from the two cost models mentioned above. He uses the number of channels in the network as the cost metric. He compares the performance of several *hierarchical* networks, where different topologies are used at different levels of the packaging hierarchy. His work is similar to ours to the extent that the packaging hierarchy is explicitly considered in the design process. However, his cost model is different from ours; under his model, each channel costs the

same, irrespective of which level in the hierarchy it belongs to. We believe that in most current packaging hierarchies the cost per channel is usually lower at the lower levels of the packaging hierarchy. Our cost models reflect this variation in costs.

1.6 Contributions

The main contributions of this thesis are:

1. *A model for computing network cost:* We develop a detailed model to characterize packaging costs and provide a method to compute the cost of a network in the presence of a complex hierarchy with a large number of cost factors.

2. *A systematic method for exploring the design space:* We develop a systematic method for exploring the search space of network designs. We argue for a top-down hierarchical network design strategy and use this design strategy to identify several network organizations that offer high performance for relatively low cost. Some of these network organizations (e.g. products of complete graphs, high degree de Bruijn networks) have not received much attention in the literature, even though they are better than many of the popular networks in terms of cost and performance.

3. *General principles for network design:* From extensive simulation studies of networks we arrive at the following general principles for network design: 1) Making the networks denser at the lower levels of the packaging hierarchy has a significant positive impact on global communication performance, 2) It is better to organize a fixed amount of communication bandwidth as a smaller number of high bandwidth channels, 3) For shared memory based communication primitives it is better to make the number of memory modules smaller than the number of processors and 4) Providing the processors with the ability to tolerate latencies (by using *multithreading*) is very useful in improving performance.

4. *A definition and categorization of scalability:* We provide a formal definition of scalability and give a method to compare two network designs in terms of scalability. We categorize scalability based on the level of flexibility in scaling networks. As examples of these categories, we describe extensions to some of the networks discussed in this dissertation. By comparing networks from these categories in terms of performance,

we demonstrate the trade-offs between the level of scaling flexibility and network performance.

5. *A simple fault-tolerance mechanism:* We present a simple and efficient scheme to tolerate faults in single path networks and evaluate this scheme in the presence of a large number of faults.

1.7 Overview of the dissertation

In Chapter 2 we develop a model of packaging technology. The model characterizes multi-level packaging hierarchies and provides a method to compute the cost of a network. In Chapter 3 we describe the strategy that we adopt in our network design process. We first define three generic models of packaging that deal with the costs of packaging at progressively increasing levels of detail. Chapter 3 also describes the methodology we use to evaluate network performance. In Chapters 4, 5, and 6 we consider networks designed under each of the three generic models. For each model, we compare several networks in terms of cost and performance. We shall see that the interconnect at the top level of the hierarchy has the greatest impact on network performance. For each model, we identify top-level networks that offer high performance at low cost, and then consider the lower levels of the hierarchy. The lower level interconnects also affect network performance, but not to the same extent as the top level network. The lower level interconnects affect the utilization of the top level network. We explore various alternatives for the lower level networks that include varying channel widths, channel dilation, etc., and draw quantitative comparisons between them. In Chapter 7, we provide a formal definition of scalability, and classify networks into three scalability categories depending on the flexibility that the networks provide, in going from one size to the other. In Chapter 8 we design a fault-tolerance scheme for the networks presented in this thesis. Finally in Chapter 9 we summarize the main results of this thesis and conclude.

Chapter 2

Packaging Model

Neither the VLSI model nor the Pin-requirements model, described in the previous chapter, is adequate for large machines, since both models deal only with the lowest levels of the packaging hierarchy, where as large parallel machines typically employ several levels of packaging [Nic92, McM92]. There are inherent technological constraints that limit VLSI die sizes, board areas, number of boards per card-cage, number of card-cages per cabinet, etc. As a result, if a system does not fit on a chip, it will first have to be partitioned into multiple chips; if all the chips do not fit on a board, we need multiple boards, and so on. Multi-level hierarchies become inevitable for packaging large parallel systems consisting of thousands of processors.

In the presence of a multi-level packaging hierarchy, it becomes harder to compare the cost of different designs. Cost is a function of a large number of parameters, such as the number of wires at each level of the hierarchy, the size of the printed circuit boards, the number of boards, number of connectors, etc. In order to get accurate cost estimates, we need a better model of packaging technology that explicitly deals with the costs associated with the various parameters. In addition to the costs, the model should also characterize technology-specific constraints such as limits on pin-counts and areas of chips, boards, etc.

In this chapter we define a model of packaging technology that characterizes multi-level packaging hierarchies (Section 2.1). Our model has a number of parameters to explicitly deal with the various constraints and costs associated with each level of the packaging hierarchy. By appropriately specifying these parameters we can model a wide range of packaging technologies (Section 2.2). Nevertheless, our model is an approximation and cannot model all the details of packaging technologies. We discuss the approximations made by

the model and its limitations in Section 2.3.

2.1 Model

We model packaging technology as consisting of l levels (numbered from 0 to $l-1$). We call the ith level of the hierarchy as a level-i module. Each level-i module contains some number of level-$(i-1)$ modules and these, in turn contain level-$(i-2)$ modules and so on. Level-0 modules are processing elements, memory units and routing nodes. Figure 2.1 illustrates the hierarchy of modules. *Pins* are used to communicate across module boundaries. Pins are also classified into levels; a level-i pin is a connection that crosses a level-i module boundary. A connection between any two pins is called a *wire*. Wires do not cross module boundaries. A wire within a level-$(i+1)$ module, but outside a level-i module, is called a level-i wire. A level-i wire is used to make a connection between two level-i pins, or between a level-i pin and level-$(i+1)$ pin (figure 2.2). If we wish to make a connection between two level-i modules in different level-$(i+1)$ modules, but the same level-$(i+2)$ module, the connection has to cross level-$(i+1)$ module boundaries. This connection would be considered as 3 wires; two level-i wires and one level-$(i+1)$ wire. A *bundle* is a group of wires, all of which connect the same pair of modules. A bundle of level-i wires is called a level-i bundle.

Our packaging model is defined using two sets of parameters: constraint parameters and cost parameters. Packaging constraints are specified using a set of vectors, $(s_{max}, p_{max}, b_{max}, w_{max}, t)$ of length l, which are defined as follows:

- Module sizes (s_{max}): $s_{max}[i]$ is the maximum number of level-i modules that can be contained in a level-$(i+1)$ module. $s_{max}[l-1]$ is the maximum number of modules at the top of the hierarchy.

- Pin counts (p_{max}): $p_{max}[i]$ is the maximum level-i pin-count per level-i module.

- Bundle counts (b_{max}): $b_{max}[i]$ is the maximum number of level-i bundles per level-i module.

- Bisection widths (w_{max}): $w_{max}[i]$ is the maximum bisection width of the network connecting the level-i modules within a level-$(i+1)$ module. The bisection width of

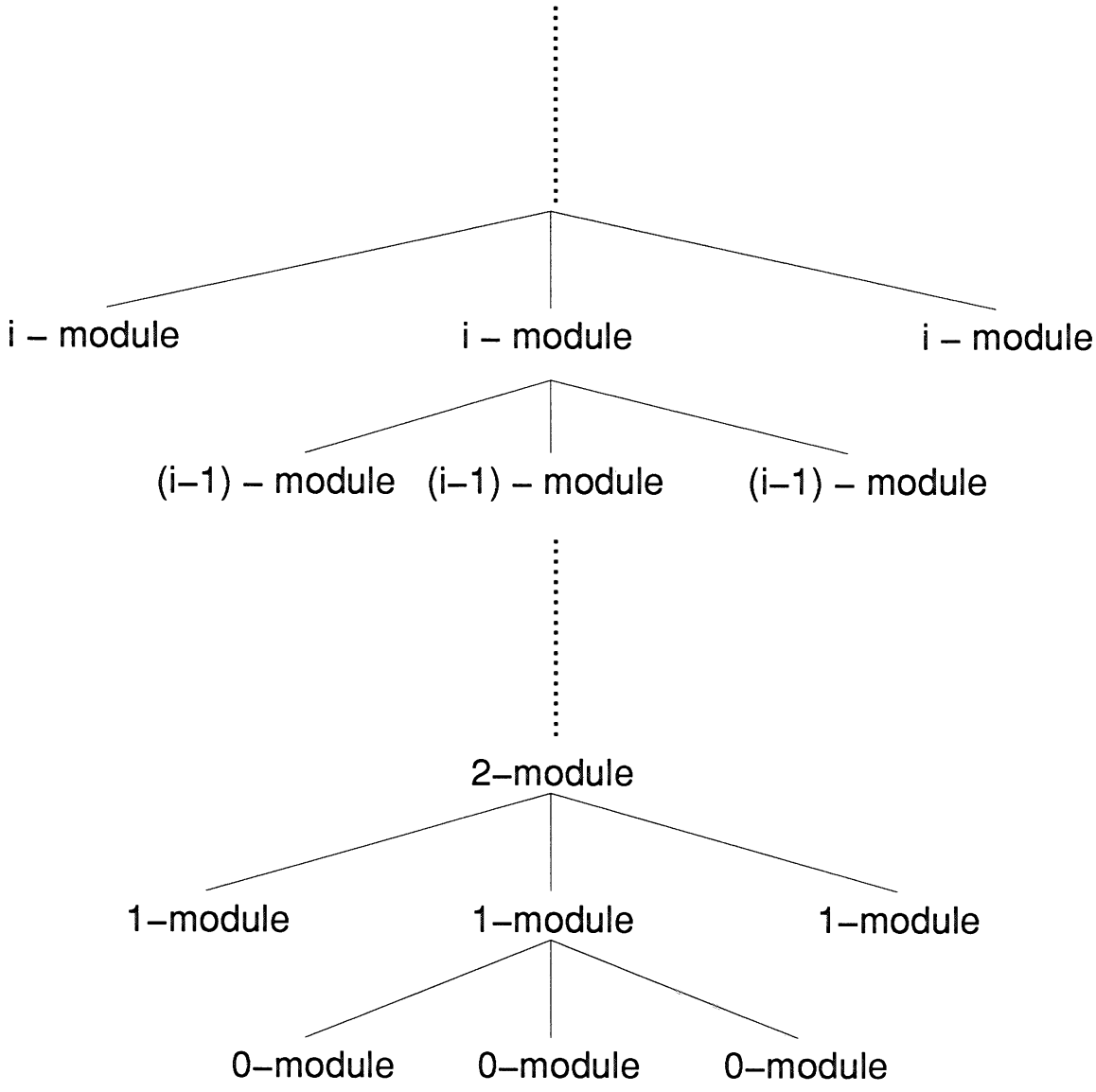


Figure 2.1: Packaging Hierarchy

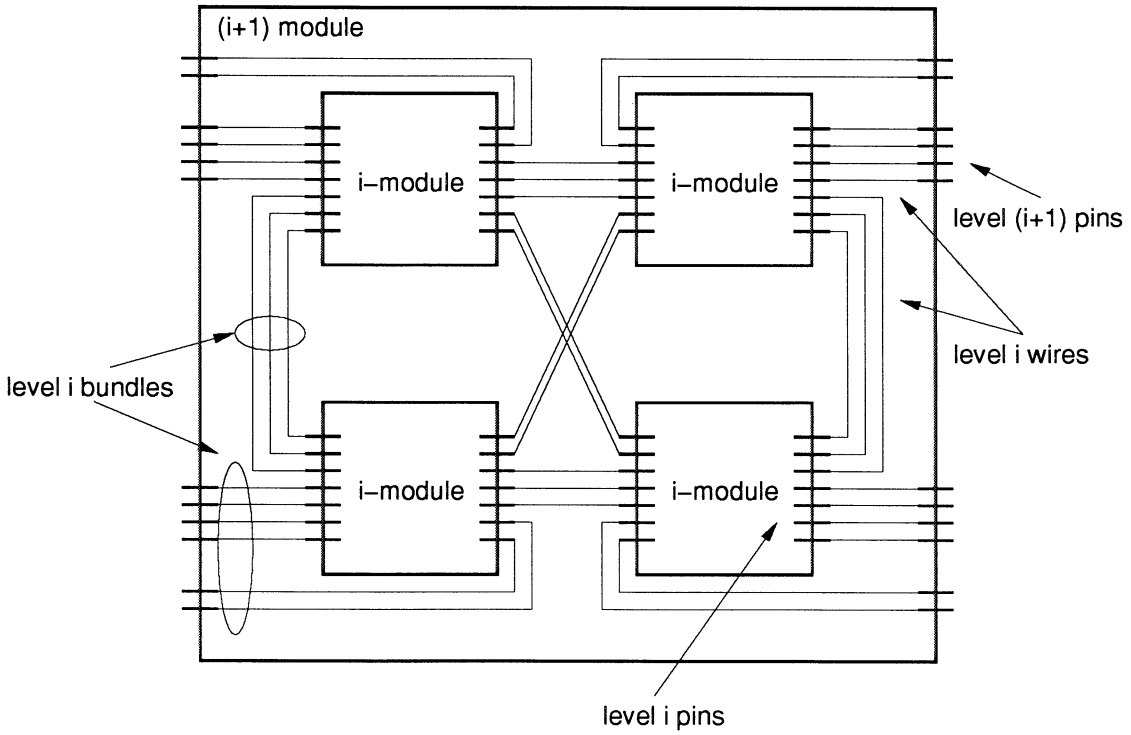


Figure 2.2: Classification of pins, wires and bundles

a network is the minimum number of wires that are cut when the network is divided into two equal parts.

- Clock rates (t): $t[i]$ is the rate at which bits are transmitted over level-i wires.

A valid network design under our model is one that satisfies all the constraint parameters. Such a design is described using the set of vectors $(s_{tot}, p_{tot}, b_{tot}, w_{tot})$. $s_{tot}[l-1]$, where $s_{tot}[l-1] \leq s_{max}[l-1]$, is the number of level-$(l-1)$ modules (top level modules) used by the design. Each level-$(l-1)$ module could contain different numbers of level-$(l-2)$ modules, provided each level-$(l-1)$ module contains no more than $s_{max}[l-2]$ modules. $s_{tot}[l-2]$ is the total number of level-$(l-2)$ modules used by the design ($s_{tot}[l-2] \leq s_{tot}[l-1] * s_{max}[l-2]$). In general $s_{tot}[i]$ is the total number of level-i modules used by the network[1]. $p_{tot}[i]$, $b_{tot}[i]$ and $w_{tot}[i]$ are similarly defined to be the total level-i pin-count, the total number of level-i bundles and the total level-i bisection respectively. We treat the clock rates as being fixed by the packaging technology, so they are not part of the description of the network.

The cost parameters of the model are the vectors (c_s, c_p, c_b, c_w) and are the relative weights of the corresponding quantities s_{tot}, p_{tot}, b_{tot} and w_{tot} respectively. The cost of a network is computed using the function:

$$Cost = \sum_{i=0}^{l-1}(c_s[i] * s_{tot}[i] + c_p[i] * p_{tot}[i] + c_b[i] * b_{tot}[i] + c_w[i] * w_{tot}[i])$$

2.2 Choosing the model parameters

Both the VLSI grid model and the Pin-requirements model are special cases of our model. The VLSI grid model is based on a single level hierarchy and uses layout area as the cost function. To convert our model to the VLSI grid model, we choose $l = 1$, $c_w[0] = 1$, and set all other cost parameters to zero. Since the bisection width of a network is proportional to the square root of its area, if one network is more expensive than another under the grid model, it will also more expensive under our model. However, the cost ratios will not be the same since our cost function is linear in the bisection width, while the cost function of the grid model is quadratic in the bisection width. To convert our model to the

[1]Note that a network design need not use all the l levels of the packaging hierarchy. Small networks may use fewer than l levels. The values of $s_{tot}[i]$ for unused levels is taken to be 0.

Pin-requirements model, we choose $l = 1$, $c_p[0] = 1$, and set all other cost parameters to zero.

Our packaging model is general enough to characterize a wide range of packaging technologies. By appropriately chosing the parameters, we can model most packaging technologies to reasonable accuracy. For example, consider a conventional packaging technology consisting of printed circuit boards in a card-cage, with the boards connected over a back plane. Each processor or routing node is a separate VLSI chip, and several VLSI chips are placed on a board and connected to each other using copper wires etched on the board. A natural representation of this technology in the model will consist of a two level hierarchy ($l = 2$) where the boards are considered to be level-1 modules and the chips are level-0 modules. The values of $s_{max}[0]$ and $s_{max}[1]$ will depend on the number of chips per board and the number of slots in the back plane respectively. The values of $p_{max}[0]$ and $p_{max}[1]$ correspond to pins per chip and pins per board respectively. In this technology there is no real limitation on the number of bundles, so $b_{max}[i] = \infty$. However, since the number of pins is a natural limit on the number of bundles, it is equivalent to say $b_{max}[i] = p_{max}[i]$. In this technology, we usually have a tight restriction on the number of wires in the backplane, and this is modeled using an appropriate value for $w_{max}[1]$. The value of $w_{max}[0]$ would depend on the area of the printed circuit board and number of layers of wiring. Choosing $t_{max}[0]$ and $t_{max}[1]$ is straight-forward. However, choosing the cost parameters is more difficult since our cost function is linear, while realistic cost functions will usually be non-linear. For example, the incremental cost of a wire on a printed circuit board will be small unless it requires increasing the number of layers, at which point the cost will increase by a step function. The cost parameters have to be chosen in a manner that provides a good approximation to the actual cost function.

If we have multiple card-cages within a rack, we may consider connecting boards in different card-cages using cables, while the boards within a card-cage are connected using the back-plane. To model this technology, we would require an additional level of the hierarchy, where the card-cages are considered to be level-2 modules. Since the connections between different card-cages (level-2 modules) are actually made between boards (level-1 modules), there are no physical level-2 pins. However, each board has two types of pins: those that connect to the back plane and those that connect to cables. We define the level-2 pin-count as the total number of pins connecting to cables, over all the boards in a card-cage.

We can also model technologies that use direct die mounting [Mey89, Ath90], multichip modules [Bak90] or wafer-scale integration, by using more levels in the abstract hierarchy. In such technologies, the number of pins per chip can be significantly higher than technologies which place single dies on a chip carrier. This can be modeled using lower per-pin costs and less stringent pin constraints. There may also be stricter limitations on bisection width, since there may be constraints on layout area. The number of bundles is not an important cost metric at this level.

When the machine is too large to fit within a rack, and we need to partition it among multiple racks, the bundling criteria becomes important. Racks are typically interconnected using cables and each cable corresponds to a bundle in our model. Each bundle contributes to the cost of the network in the form of connectors at either end. The number of bundles is also important from the point of view of the complexity involved in wiring the network.

Free-space optical interconnection technologies are being proposed for future parallel machines [TG92, Nef92]. In such technologies, laser beams are used to make connections, instead of pins and wires. Some of these technology proposals also include diffractive or reflective holograms to steer the laser beams. Since laser beams can cross each other in free space without interference, it appears that restrictions on bisection width and bundling may be irrelevant under such technologies. However, the laser sources and receivers may be expensive. In order to fit our model to such a technology, we could model each light beam as a wire and use high pin costs to model the costs of the laser sources and receivers.

In our model of the packaging hierarchy, the processors, routing nodes, and memory units form the leaf nodes (or level-0) modules. It is straightforward to extend our hierarchy further, so that the leaf nodes are smaller units, perhaps individual gates. However, since the primary focus of this dissertation is the interconnection network between processors, this extension is not required.

2.3 Limitations

Our model places no restrictions on the interconnection between modules at a level of the hierarchy, except for pin counts, bundling and bisection width constraints. Pins on any two modules can be connected by a wire, provided the constraints are satisfied. This flexibility may not be permitted in some packaging technologies. For instance the packaging

technology proposed for the NuMesh project at M.I.T. uses modules which are connected by abutting each other in a *lego block* fashion [War92]. Each pin of a module can connect only to a physically adjacent module. The modules are assembled in a three dimensional lattice structure and the geometrical positioning of the pins on a module determines the kinds of structures that can be built using these modules. It is not possible to use the parameters of our model to restrict network designs to such lattice structures. However, we can overcome this problem to a certain extent by using additional levels of hierarchy in the model. We can recursively partition the lattice structure into sub-units and treat each sub-unit as a separate level of the hierarchy.

Another limitation of our model is that, at a given level of the hierarchy, all wires are assumed to cost the same. In reality, the wires may have different costs depending on how long they are. For example, wires that go between adjacent boards in a back-plane may be cheaper than wires that traverse the entire length of the back-plane, because longer wires need more powerful drivers. We can use a similar technique of using additional levels of hierarchy to overcome the problem of modeling the difference in costs. We can get a better cost approximation by sub-dividing the card-cage into two or more levels and use different cost parameters for the different levels.

Our model treats the clock rate (rate of bit transmission over wires) as being independent of network topology, but, clock rates may depend on wire length. In general, longer wires have larger capacitances and it takes longer to charge and discharge them, resulting in higher signal propagation delays and lower clock rates. However, in order to overcome the problem of propagation delays on long wires, many current systems treat wires as transmission lines and send bits along them in a pipelined fashion. When pipelined data transmission is used, the assumption made by our model is justified since the clock rates become independent of wire length.

There are many reasons to believe that pipelined data transmission will be the norm in future large scale systems. Technological advances have decreased processor cycle times by a much larger factor than the physical size of parallel computers, resulting in a substantial mismatch between processor speeds and wire delays. This makes it essential to use pipelined data transmission to achieve good performance. Some of the currently available commercial parallel machines and research machines, such as the CM5 [TMC91, L$^+$92], the MIT Alewife [A$^+$91], and the Monsoon [Joe90] use pipelined data transmission. The IEEE Standard, Scalable Coherent Interface (SCI) [IEE], that is currently under develop-

ment, also includes pipelined data transmission. If machines use optical fiber interconnects instead of wires, pipelining is easier and usually the default method of data transmission.

Our model permits different clock rates at different levels of the hierarchy, but for the remaining part of the thesis we assume that the clock rates are the same over all the levels of the hierarchy. The clock rates at the lower levels of the hierarchy are usually higher. We model this using wider channels at the lower levels while keeping the clock rate constant over the entire hierarchy. Both higher clock rates and wider channels essentially have the same effect; namely increasing channel bandwidth. Since we find that network evaluation is simplified under the assumption of constant clock rate, and does not make a difference as far as our results are concerned, henceforth we will assume constant clock rates.

Chapter 3

Methodology

The main goal we pursue in this dissertation is to design the *best* possible network for a large scale parallel machine. We define the *best* network to be the one that achieves the maximum performance for a fixed cost or equivalently one that achieves a fixed performance for a minimum cost. In the preceding chapter, we defined a packaging model using which we can compute network cost. It is possible to use the model to characterize many packaging technologies and design the best network for that technology. However, packaging technologies change with time; a technology that is popular today may be superseded by a different one that becomes feasible in the future. When the technology changes, we would have to repeat the entire design process.

However, instead of basing our designs on a specific technology, we would like to obtain qualitative results with respect network design for packaging hierarchies in general. Therefore, instead of choosing the model parameters specific to a packaging technology, we wish to choose them in a manner that reflects the generic characteristics of packaging hierarchies. We would like choose parameters in a manner that represents general technological trends and omits details particular to a specific technology. The objective of choosing parameters in this manner, is to make our results more general and applicable over a wide range of packaging technologies, both current and future.

In order to choose the parameters in this fashion, we model the fundamental properties that are true of all packaging hierarchies. One such property exhibited by all current packaging hierarchies is *packaging locality*, i.e., there is a progressive increase in costs and decrease in density as we go up the levels of the hierarchy. It is clear that as we proceed up the hierarchy, wire length and pitch (spacing between wires) increase. For

instance, metal wires on VLSI chips may be a few millimeters long and spaced a few microns apart, whereas wires connecting different boards may be as long as a 1 meter, and spacing between wires on a cable connector may be as much as 1 mm. This results in a higher cost per wire at the higher levels of the hierarchy, and the higher cost translates into a reduction in number of wires available at the higher levels. *Packaging locality* is also seen when we consider memory hierarchies [ACF90], where the bandwidth at lower levels (e.g. registers) is higher than the bandwidth at the upper levels (e.g. DRAM). We believe that *packaging locality* is a fundamental property that will be applicable to all future packaging technologies as well.

A consequence of *packaging locality* is that the costs at the higher levels of the packaging hierarchy dominate the total cost of the network. Accordingly, we define three generic models that emphasize the costs at the higher levels of the hierarchy. These models are briefly described in Section 3.1. The three models progressively characterize costs at increasing levels of detail. Each model is the topic of a subsequent chapter, where we evaluate different network designs under that model.

Section 3.2 outlines our network design strategy. We design networks in a hierarchical fashion starting at the top, proceeding down the levels of the hierarchy. In Section 3.3 we describe the work-load model and the types of messages we expect the network to handle. We approximate this work-load using synthetic work-loads and evaluate the networks by simulating these work-loads. Our synthetic work-loads assume that the messages in the network are sent to randomly chosen destinations. Section 3.3.1, describes our network evaluation methodology. Sections 3.3.2 and 3.3.3 describe the synthetic work-loads in detail. Finally in Section 3.4, we provide details regarding the routing algorithm, messages queueing mechanism, message lengths, etc.

3.1 Generic Models

We start with a very simple model and gradually move on to more complicated models. Since we expect the costs at the higher levels of the packaging hierarchy to dominate the cost of the network, for our first generic model, *model 1*, we shall limit ourselves to the costs at the top level of the hierarchy. Note that the actual hierarchy may consist of several levels of packaging, but since *model 1* only deals with the costs at the top, it is sufficient to consider a two level hierarchy: level 0 and level 1. Everything below the top level of the

hierarchy is modeled as being internal to a level-1 module, and the costs internal to a level-1 module are ignored. The parameters describing the interconnection between the level-1 modules are used to compute cost. In our model described in Section 2.1, we used four parameters to characterize the cost at each level: the number of modules, the pin-count, the number of bundles and the bisection width. To keep *model 1* simple, we model the cost of the network using only two of these four parameters: the number of level-1 modules, and the level-1 pin-count. Although under *model 1*, the level-1 bisection width does not contribute to the cost of the network, we shall see that when two networks have different bisection widths, it is possible to show that the one with the lower bisection width will have lower performance. Therefore, we shall see that for the networks we compare under *model 1* (Chapter 4), the level-1 bisection width is held constant. However, *model 1*, ignores any restrictions on the number of level-1 bundles. The costs at level-0 are also ignored.

In order to fully define *model 1*, we also need to define the cost function. Since we have decided to compute the cost of the network using the number of level-1 modules, and the level-1 pin-count, we need to specify the relative weights of these two parameters to define the cost function.

However, these relative weights are likely to be specific to a particular technology, and the weights are likely to change significantly when the underlying technology changes. Therefore, any results obtained under a particular choice of weights are likely to be applicable only to the specific technology that the weights correspond to. Since we are interested in obtaining results that are applicable under a wide range of packaging technologies, we seek a technology independent way of computing the network cost.

The approach we take in this thesis is to retain the cost of a network as a multi-dimensional quantity. Cost, then becomes a vector rather than a simple number. Under *model 1*, the two components of the cost vector are the number of modules and the pin-count. When cost is a multi-dimensional quantity, it becomes more difficult to compare the costs of networks. If we have networks with different costs, we can only specify a partial order on the costs of the networks, because the cost of one network is greater than the cost of another only if all the components of the cost vector of the first network are greater than the corresponding components of the second. When we only have a partial order, the set of cost comparisons we can make is limited. However, we shall see that even with just a partial order available to us, we can still draw qualitative conclusions regarding the design of networks.

The second generic model, *model 2* is more detailed than *model 1*; but it is also based on a two level hierarchy. In going from *model 1* to *model 2* we add one more dimension to the cost vector; namely the number of level-1 bundles. Therefore, the cost vector has three dimensions: the number of level-1 modules, the level-1 pin-count, and the number of level-1 bundles. As in *model 1*, we shall see that the bisection width of the network is held constant, even though it does not form part of the cost vector. Therefore, *model 2* takes into account all the parameters of the top level of the packaging hierarchy. Similar to *model 1*, we ignore the costs at level-0. We shall see that when we use a more complex cost vector, the search space of interesting network designs widens. For example, the each of the dimensions of the cost vector of one network might have been lower the corresponding dimensions of another under the simpler cost vector of *model 1*, but when we make the cost vector more complex, we may not have a similar domination in terms of the dimensions of the new cost vector. We shall see that comparing the costs of networks under *model 2* is more difficult than under *model 1*, for this reason. Detailed descriptions of network evaluations under *model 2* are presented in Chapter 5.

In *model 3*, we increase the level of detail in the modeling of costs by extending the number of levels in the hierarchy. In *model 3*, we consider the costs at the top two levels of the hierarchy. Therefore, it is sufficient to think of the hierarchy as consisting of 3 levels. The cost vector under *model 3* is more complex than the cost vectors of the other two models. In *model 3* the cost vector has five components: the number of level-2 modules, the number of level-1 modules, the level-2 pin-count, the number of level-2 bundles and the level-1 pin-count. The top-level bisection width (level-2 bisection width), is held constant as in the other two models. Since we have a large number of dimensions in the cost vector, we shall fix some of the dimensions and explore the trade-offs between the others. Specifically we shall fix the number of top level modules, the number of level-1 modules and the level-2 pin-count and explore the trade-offs between the level-1 pin-count and the number of bundles. Details of the networks that exhibit these trade-offs are presented in Chapter 6.

3.2 Design Strategy

We wish to design the best network under each of the three models described above. We pose this design problem as a search problem, where we compare the performance

different networks of fixed cost. A network is defined by specifying the interconnections at each level of the hierarchy. There are several choices at each level and we are confronted with the cross-product of the design choices. Evaluating each of the designs in this cross-product can be time consuming because the set of possible designs can be very large. Therefore, we seek a better strategy of exploring this search space.

We adopt a hierarchical design strategy where we start at the top and go down the levels of the hierarchy sequentially. For instance, in the case of our first generic model, *model 1*, we design the level-1 network first and then consider the level-0 network. When we design the top level network, we ignore the costs at level-0, and design it so as to maximize the performance for a fixed cost. Then we proceed to level-0 and consider the design of the network at that level. Let us assume that the level-0 network costs a fraction x of the cost of the top level network, for a total cost of $(1 + x)$. It is conceivable that we might have been able to design a better network for the same cost of $(1 + x)$, had we considered the cross-product of possible designs at both levels. As we shall see later, the design of the level-1 network has a greater impact on overall performance than the level-0 network. Therefore, a network that is better than the one our strategy yields, is expected to have a higher cost allocated to the top level network. However, we expect that the fraction x is likely to be small to begin with, since we believe that for most technologies, the costs at the higher levels tend to dominate the total network cost. Besides, we also expect that the amount of cost reallocation possible is also likely to be small. Therefore, we believe that our strategy is justified.

While designing the top level network we do not have a design for the lower level network, so we can only estimate the performance of the top level network, instead of measuring it. We estimate the performance of the network using its *peak throughput*. The *peak throughput* of a network is the maximum amount of data that it can deliver in steady state, in unit time when the communication traffic is randomly distributed. We compare several alternatives for the top level network and choose those that have the highest peak throughput for the least cost.

We then proceed to the next lower level of the hierarchy and consider design alternatives at that level. We prune the number of design choices using a strategy similar to the one applied to the top level network, namely choosing networks that offer high estimated performance for low cost. Since the top-level network fixes the maximum performance that can be achieved, when we design the lower level networks, our objective is to maximize the

utilization of the top-level network, and we attempt to achieve a level of performance that is as close as possible to the peak. After we have designed a network for each level, we evaluate the entire network using detailed simulation.

The level-by-level design strategy simplifies the network design process, and is justified from the point of view of packaging costs, but we may potentially run the risk of eliminating good networks from consideration apriori. For instance, we eliminate a top-level network that has a lower peak performance. Consider a situation where we have a choice between two top-level networks A and B. Let the peak performance of B be 0.8 that of A. Further let there exist lower level networks for B that achieve 0.8 of its peak performance. Unless we can design lower level networks for A that achieve greater than 0.64 of its peak, we would have skipped over a better network design since we would have eliminated B because it had a lower peak performance. However, we shall see that for most of the networks we evaluate, we are able to design lower level networks that achieve close to the peak performance of the top-level network reducing the risk of eliminating better networks without evaluating them.

3.3 Work-load Model

All of our network evaluations are based on randomly distributed inter-processor communication traffic. While some applications, such as those based on dense matrix computations, have extremely regular communication traffic, there is a large class of problems for which the communication traffic tends to be highly irregular and varying with time. Sparse matrix computations and graph algorithms belong to this latter class. Further, many programs that initially had regular communication patterns tend to lose this property when algorithmic optimizations are performed. For instance, the $O(n^2)$ version of the N-body program can be written so that processor i only has to communicate with processors $(i+1)$ and $(i-1)$. However, when a better algorithm such as the $O(n \log n)$ Barnes and Hut algorithm [BH86] or the $O(n)$ Greengard and Rokhlin algorithm [GR87] is used, the communication traffic ceases to be as regular as in the $O(n^2)$ case. Further, for many programs, the regularity in communication patterns is lost when the size of the problem is much larger than the number of processors available. For instance, an FFT computation where the number of processors is equal to the number of elements in the FFT, has a communication pattern that maps well onto a hypercube, but if the number of elements in

the FFT is substantially greater than the number of processors, an efficient implementation generally requires communication traffic that is no longer hypercube-like. For all of the above reasons, it is essential to design networks that are capable of supporting global communication traffic efficiently. In this thesis we model global communication using randomly distributed communication requests (the communication requests issued by each processor are assumed to be distributed uniformly across the entire machine).

For simplicity, we assume that the communication primitives are based on a shared-memory model, but we believe that our results do not rely heavily on this assumption. The logical view of the machine is shown in figure 3.1. The machine consists of N_p processors connected to N_m memory units using an interconnection network. The processors communicate with each other by reading or writing locations in the shared address space. The shared address space is distributed among the memory units. When a processor needs to access a location in the shared address space, it sends a message to the corresponding memory unit. The message travels from the processor the memory unit, performs the access and returns to the processor. Each access corresponds to a short message that typically deals with a small amount of data, such as a single word or a cache line. In addition to the memory units shown in the figure, the processors may have private local memory and caches which are not shown.

The logical view of the machine encompasses a variety of different architectures. It includes machines like the IBM RP3 where all the memories are equidistant from all the processors, and machines where the latency associated with different accesses are different because the path lengths to the corresponding memory units are different. It also includes bus-based shared-memory machines with a single memory unit sitting on a bus ($N_m = 1$) and machines which consist of processor memory pairs. In the latter case, $N_p = N_m$ and each processor may have a single memory system which is partitioned into private and shared memory. For most of the machine architectures evaluated in this dissertation N_p will be equal to N_m, but we also consider a few cases where $N_m < N_p$.

We assume that each time a processor wishes to make an access to shared-memory, it directs the access to a randomly chosen memory address. Access non-uniformities and *hot-spots* [PN85] due to repeated accesses to synchronization variables were topics of active research in the recent past. Message combining was proposed as a means of dealing with hot-spots and was implemented in a few research machines [GGK+83, PBG+85]. See [LKK86] for an evaluation of message combining techniques. However, combining-networks tend to

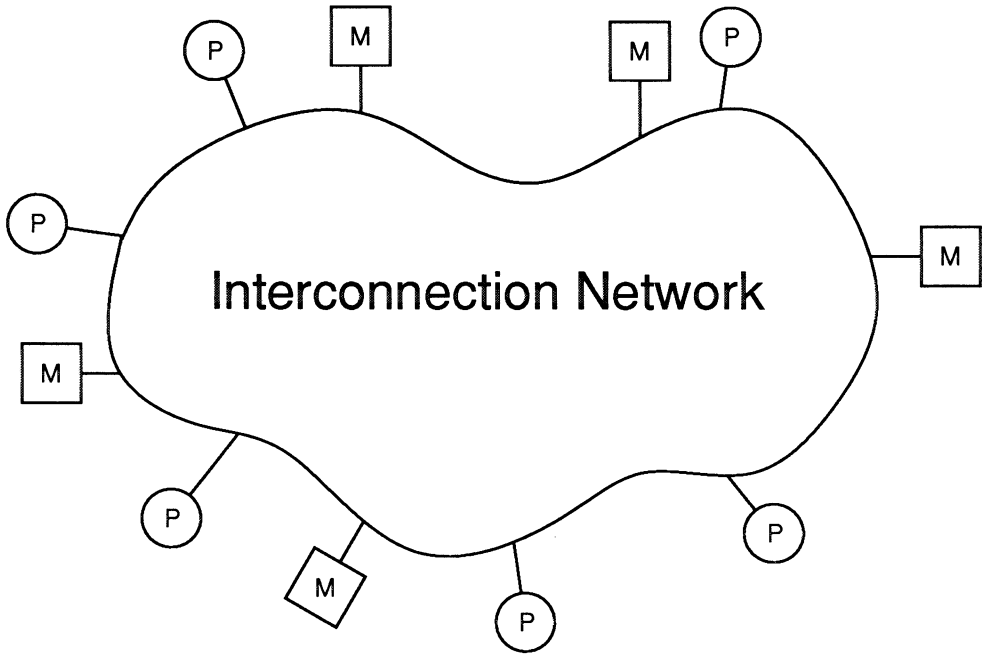


Figure 3.1: Logical machine model

be expensive to build and are sometimes not justified since software techniques such as the ones proposed in [YTL86] and [MCS91] can be used to prevent hot-spots. Nevertheless, combining-networks may be justified for certain specialized functions such as barrier synchronization [L+92]. If the hot-spot accesses are handled either using software techniques or by the use of special purpose networks, the remaining accesses will tend to be uniformly distributed. Therefore, in the remaining part of this thesis we will not deal with hot-spots explicitly.

3.3.1 Evaluation Methodology

We use simulation of synthetic work-loads to evaluate network performance. We prefer simulation to other options such as prototyping and analytical modeling for a variety of reasons. Prototyping is both expensive and time consuming. It is also unfeasible when a large number of networks need to be evaluated. Solving analytical models can be intractable unless simplifying assumptions are made, and it may not always be possible to make such assumptions without sacrificing the accuracy of the results. Under these conditions, the only available option is simulation. Simulation models can be much closer to the actual

network since the only real limitation of model complexity is simulation time.

We use synthetic work-loads to drive the simulation instead of real parallel programs or execution traces. We evaluate systems consisting of a thousand processors, and since only a small number of such systems have been built, it is difficult to get a good collection of real programs. Besides, many parallel programs that have been written, tend to be proprietary and therefore unavailable. It might be possible to get traces of these proprietary programs, but it is not clear whether a trace obtained on one machine can be used to evaluate another with a different network architecture and timing. Even if we were able to configure the trace to take differences in network architecture and timing into account, simulation of the trace will only tell us how the architecture performs on that particular program. By using a synthetic work-load and varying its parameters, we can exercise the network under a wide range of conditions leading to a better evaluation of network performance.

3.3.2 Open-Network Model

The first work-load model we use is the well-known *Open-Network* [Lee89] model and has been used by many researchers. Under this model, each processor makes an access in each cycle with probability λ and performs a local operation with probability $1-\lambda$. Each access is directed to a random address that is chosen uniformly over the address space. The activity of any two different processors are independent of one another. Processors continue execution without waiting for their previous accesses to complete; hence the name *open-network* model.

The interval between successive accesses made by a processor is a geometrically distributed random variable with a mean of $1/\lambda$. The accesses form a discrete time version of a *poisson process* that is commonly used in queueing theory. The message generation process is memory-less, i.e., the probability of generating a message in a given cycle is independent of the history of the system.

We simulate each network under the open-network work-load for different values of λ and measure the average latency at steady state. Results are reported as graphs of latency versus the offered load λ. We compute the average latency at 1000 cycle intervals and when the last 10 observations (corresponding to 10,000 cycles) lie within 2% of each other, we conclude that the network has reached steady state and record the latency. If

the offered load λ is greater than the bandwidth that the network can provide, the latency does not stabilize and therefore there is no data point for the corresponding value of λ. The interesting metrics under this work-load model are the maximum sustainable value of λ and latency.

3.3.3 Multithreading Model

In the open-network model, network latency has no influence on message generation. It is unreasonable to expect that a processor can have an unbounded number of outstanding messages waiting in the network. Therefore, it is desirable to incorporate feedback in the message generation process to make the work-load model more realistic.

Incorporating this feedback using an execution model based on a simple uniprocessor can lead to poor performance. In the case of a simple uniprocessor, the processor has to wait for each memory access to complete before it can continue executing. This is inappropriate for large scale parallel machines because each access can typically take a large number of cycles to complete. Our measurements indicate that it is reasonable to expect access latencies of approximately 100 processor cycles for a 1000 processor machine. If a processor has to wait for each access, the processors will spend most of their time idling. For example if the memory access takes 100 cycles and the processors make a shared access every 20 cycles, processor utilization can never be more than 16%. Further the network will be significantly under-utilized because each processor can have only one access in the network at any given time.

It is essential for the processors used in large scale parallel machines to have a mechanism to tolerate long access latencies. We shall first assume that remote accesses are split-phase operations. Each remote access consists of two instructions: an *initiate* instruction and a *use* instruction. When a processor needs to make an access, it executes the *initiate* instruction which creates a message and sends it over the network. Later, the processor executes a *use* instruction, which causes the processor to stall for the corresponding access to complete. We assume that both *read* and *write* accesses are acknowledged. In response to a *use* instruction, the processor waits for the corresponding acknowledgement to be received. By breaking up a remote memory access into *initiate* and *use* instructions, we may be able to overlap the memory access latency with useful computation. Typically a compiler would first split a remote memory access into *initiate* and *use* instructions and

then attempt to move the *initiate* instructions backward so that the memory access is initiated many cycles before the result is required. It may also be possible to pipeline multiple memory accesses by executing successive *initiate* instructions before the corresponding *use* instructions.

In addition to split-phase accesses, we assume that each processor has the capability to run multiple threads of execution and the ability to switch between execution contexts rapidly [Kow85, A+90]. If the processor has multiple register sets, (one per thread) and we use a simple round-robin thread scheduling mechanism, it is possible to context-switch in a single processor cycle [Boo93]. We assume that the processor has a *context-switch* instruction which causes the processor to suspend the current thread and start executing the next thread (in round-robin order). The task of the compiler is to insert *context-switch instructions* before any *use* instruction that can be expected to stall the processor. By using such a multithreading mechanism, the processors can tolerate much longer latencies than possible by merely using split-phase instructions. It is also possible for each processor to have multiple outstanding accesses in the network, thereby running the network at reasonable load.

The multithreading work-load model is defined based on the above view of processor execution where each processor runs multiple threads that perform split-phase accesses. Each processor is assumed to be running t threads using a round-robin thread scheduling policy. When a thread starts executing, it first executes a *use* instruction which stalls the processor till its previous accesses to completes. In the next cycle, the thread initiates a remote access. Then the thread executes a sequence of *local* instructions followed by a *context-switch* instruction. The *context-switch* instruction suspends the thread until the processor has stepped through the other threads. Each thread repeatedly executes the following sequence of instructions: Initiate, local, local, local, ..., Context-switch, Use. The accesses are tagged so that each *use* instruction waits for the access issued by the corresponding *initiate* instruction.

We can control the interval between accesses by varying the number of local instructions executed between the *initiate* and *context-switch* instructions. We use two inter-access interval distributions: geometric and periodic. In the geometric model, in each cycle following the *initiate* instruction, the processor either executes a *context-switch* instruction with probability p or a *local* instruction with probability $1 - p$. In the periodic model the number of local instructions is constant. If the processor is executing a tight inner loop, the

interval between accesses is likely to be constant. The periodic model attempts to capture this effect. In reality the distribution of access intervals is likely to be somewhere in between the two models.

In both the periodic and the geometric models, the accesses generated by a processor were assumed to be distributed randomly across the entire memory address space, as in the previous open-network work-load model. Note that even though the access intervals are constant in the periodic model, the accesses typically take different paths in the network depending on their destinations, and contention for message paths leads to queueing of messages at routing nodes.

We simulate the networks under both the periodic and geometric models for different numbers of threads per processor and for different values of the mean interval between accesses. The performance metric of interest, is the average processor utilization rather than the latency or the bandwidth of the network. As in the previous case, we measure average processor utilization at 1000 cycle intervals and when 10 consecutive observations (corresponding to 10,000 cycles) fall within 2% of each other, we conclude that the network has reached steady state and record the value. We note that we will always have a steady state value for the processor utilization since the network can never become unstable. Because of the feedback in the message generation process, an increase in network latency results in a decrease in processor utilization which results in a decrease in the network load. Eventually the processor utilization stabilizes, at which point, we record its value.

3.4 Routing Algorithm

We assume that all networks use virtual cut-through routing [KK79] to forward messages. Cut-through routing achieves high bandwidth that is a characteristic of packet switched routing while simultaneously providing low latency similar to circuit switched routing. In cut-through routing, each message is divided into multiple flits that are sent in a pipelined fashion, one behind the other. The head flit of a message contains the routing information needed by a node to decide where the message is to be sent. When a routing node receives the head flit of a message, it immediately attempts to forward the head flit without waiting for the entire message to come in. If a message cannot be forwarded because the output channel requested by the message is busy, or there are other messages queued in front of it, the node buffers the flits of the message.

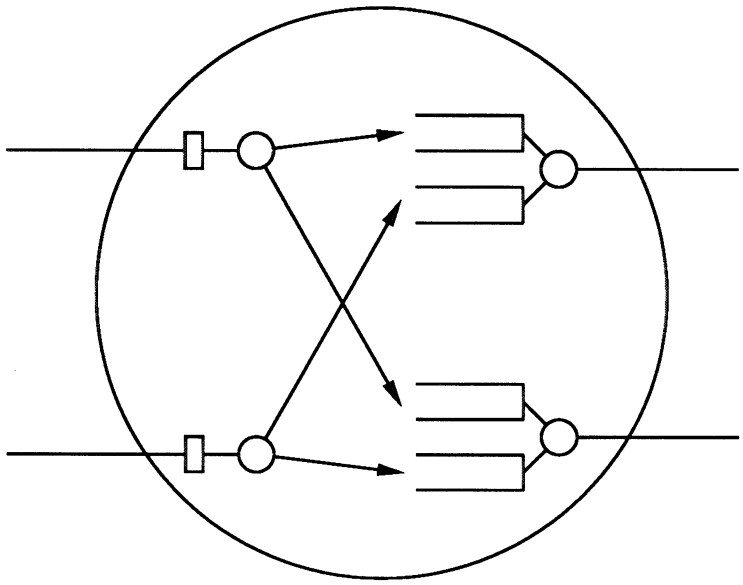


Figure 3.2: Buffers in a routing node

Virtual cut-through routing is different from worm-hole routing in that when the head flit of a message is blocked at a node, that node accepts the subsequent flits of the message and buffers the flits. In worm-hole routing a blocked message occupies buffer space on various nodes along its path. The primary difference between worm-hole routing and cut-through routing is the amount of buffer space available at the routing nodes. In cut-through routing each node is required to have sufficient buffer space to buffer entire messages and when a message is accepted for routing, buffer space is allocated for the entire message.

The allocation of buffer space does not pose a significant problem since our networks need to handle only short messages. Since we assume a shared memory model, messages sent over the network are typically accesses to shared memory — single or double word fetches or stores. All messages are round-trip messages, meaning that all write accesses are acknowledged. For the sake of simplicity, we assume that both access requests and replies are 128 bits long. This message length includes data, memory/network addressing information, and control information.

The queuing mechanism within the nodes is as follows: Each node that has I inputs and O outputs, has O sets of I message queues. The I queues of each set are connected to an output link using a I to 1 multiplexer. Any message arriving on input link i which needs to go out along output link o, enters the ith queue of the oth set of queues.

Figure 3.2 shows the buffers in a 2 input, 2 output routing node. Each output link services its queues in a simple round-robin fashion. A single flit buffer is assumed to be present at each input port to simplify the problem of allocating queue space to the messages. If the message whose head flit is in the input buffer cannot be accepted because of unavailability of queue space, it is forced to wait there until space becomes available. If this happens, the corresponding link remains idle until queue space is available and the message is accepted. In all the simulations, each message queue was assumed to be capable of holding 8 messages. Therefore, a 2 input, 2 output node has 4 queues, each of size 8 messages for a total of 512 bytes of buffer memory.

We also assume that all the wires in the network can transfer one bit per clock cycle and that each routing node imposes a single cycle pipeline delay on any message that does not suffer link or queue space contention. Recall that we model differences in clock rates using increased wiring densities at the lower levels of the packaging hierarchy.

In many networks evaluated in this dissertation, the channel widths at different parts of the network are different. However, all input channels (respectively output channels) of a routing node have the same channel width. If the output channels are wider than the input channels, the routing node cannot forward the flits as they come in. We assume that these nodes use store-and-forward routing, i.e., they wait for the entire message to come in before sending out anything. For example, consider a routing node that has 16 bit wide input channels and 32 bit wide output channels. If the first flit of a message comes in at cycle t, the node waits until cycle $t + 7$ to get the entire 128 bit message. Output transmission begins at cycle $t + 8$ and the last flit of the message leaves the node at cycle $t + 11$. Waiting for the entire message is a bit conservative, because it would be possible to start transmitting the message in cycle $t + 5$ and the last incoming flit would be able to leave the node in cycle $t + 8$. Hence our assumptions are pessimistic for networks that have non-uniform channel widths.

Chapter 4

Model 1: Pin-counts

In this chapter, we consider the first of our generic models, as defined in Section 3.1. Recall that this model consisted of a two-level packaging hierarchy where the machine is partitioned into some number of level-1 modules at the top-level. In this chapter, we shall simply refer to the level-1 modules as *packaging modules* or just *modules*. There are two levels of wiring: connections between modules (inter-module) and connections within a module (intra-module). Cost is a two dimensional vector, with the number of modules and the pin-count being the two components.

To avoid degenerate designs where we put all the processors in one module, and essentially have a pin-count of zero, we shall assume that there is a limit on the maximum number of processors that we can put in a module.

4.1 Design Strategy

We would like to design the *best* network under this model. Recall that we defined the *best* network to be one that achieves the maximum performance for a fixed cost, or equivalently one that achieves a fixed performance at a minimum cost. The strategy we adopt in this chapter is to fix the cost and compare the performance of networks.

Since the cost is a two dimensional quantity, we fix the cost by fixing both the dimensions. We fix the number of modules at M and the pin-count per module at P. Also let N be the total number of processors in the machine and s be the maximum number of processors per module.

We have some flexibility in choosing the value of M. The minimum value for M is

N/s, since each module cannot contain more than s processors. We are also likely to have some maximum limit on M; if we have a large value of M, we may have to organize these M modules themselves in a hierarchical fashion, but we assumed the modules formed the top level of the hierarchy. If we choose $M = N/s$, we minimize one of the dimensions of the cost vector, and we shall choose M thus. We note that other choices are equally valid, but they represent different points in the cost space. By choosing $M = N/s$, we are restricting our attention to a smaller portion of the cost space. We believe that this portion of the cost space is interesting because this portion of the cost space corresponds to minimizing one of the dimensions of the cost vector, and also because it corresponds to partitioning the machine into symmetric modules, with each module consisting of an equal fraction of the machine. A machine in which all modules are identical, is easier to build than a machine which uses different types of modules.

We also have some flexibility in choosing a value for P. Technology may dictate an upper bound on the value for P, but any choice of P that is less than the upper bound is a valid one. As we shall see later, the performance of any network will be directly proportional to P, so we shall defer the selection of a value for P until later.

As mentioned in Section 3.2, we shall adopt a hierarchical design strategy; first we design the top-level network (the inter-module network) and then consider the lower level network (the intra-module network). For the top level design we shall estimate the performance of the network using its peak throughput for randomly distributed communication traffic. Recall that the peak throughput of a network is defined as the maximum amount of data that a network can deliver in steady state in unit time. The design objective for the top-level network is finding a network of M modules and P pins per module, that has the maximum peak throughput.

After we select the network for the top level we shall consider several alternatives for the lower level network and simulate the entire (top level + lower level network) to determine the fraction of the peak throughput that is achieved by the network.

4.2 Inter-Module network

For randomly distributed communication traffic, the bisection width of a network is a good estimator of its peak throughput. Under random traffic, half of the messages sent by each processor cross the bisection of the network. The amount of time required to deliver

Inter-Module Topology	Pins per Module	Bisection Width
Complete graph	P	$(M^2/4)\frac{P}{M-1} = O(PM)$
Butterfly	P	$(M/\log M)\frac{P}{4} = O(PM/\log M)$
CCC	P	$(M/2\log M)\frac{P}{3} = O(PM/\log M)$
Hypercube	P	$(M/2)\frac{P}{\log M} = O(PM/\log M)$
d-dim Toroid	P	$(2M^{(d-1)/d})\frac{P}{2d} = O(PM/\sqrt[d]{M})$
3-dim Toroid	P	$(2M^{2/3})\frac{P}{6} = O(PM^{2/3})$
2-dim Toroid	P	$(2\sqrt{M})\frac{P}{4} = O(P\sqrt{M})$
Ring	P	$P = O(P)$

Table 4.1: Characteristics of topologies used to connect M packaging modules

these messages is inversely proportional to the bisection width of the network[1]. Therefore, we can argue that the peak throughput of a network is proportional to its bisection width.

Table 4.1 shows a comparison of a few standard topologies for M modules, with P pins per module[2]. We can see that among the different topologies, the complete graph has the maximum bisection width for a constant pin-count. Therefore, we expect that the complete graph has the maximum peak throughput.

In fact, we can formally prove the following claim:

Claim 4.1 *When we have M modules with P pins each, and each module has an equal number of processors, the inter-module network that achieves the maximum peak throughput is a complete graph in which each edge of the complete graph corresponds to $\frac{P}{2(M-1)}$ wires. This network provides a peak throughput of $\frac{M^2P}{2(M-1)}$ bits per clock cycle.*

For the proof, we pose the problem of finding the network with highest peak throughput as a graph problem and show that the solution to the problem has the topology of a complete graph.

We model the inter-module network as a digraph $G = (V, E)$, with edge weights. We call G the *network graph*. Each node in the graph corresponds to a module, and each

[1]Note that our model of packaging technology assumes that the clock rate is independent of network topology(Section 2.3).

[2]All logarithms are to base 2.

edge corresponds to a connection between modules. The capacity of an edge e, denoted by $w(e)$ is the number of wires that correspond to that edge. Since we have M modules,

$$|V| = M$$

and since each module has a pin-count of P, for every module x

$$\sum_{y \in V} w(x,y) + w(y,x) = P$$

The communication traffic in the network is modeled as a complete digraph $G' = (V', E')$, with edge weights. We call G' the *communication graph*. Each node in G' corresponds to a processor, and each edge corresponds to the communication between a pair of processors. The weight of an edge $e' \in E'$, denoted by $p(e')$ is the probability that there is a message between the corresponding pair of processors. We use the communication graph to model one communication step, in which each processor sends exactly one message. In general, for each processor x

$$\sum_{y \in V'} p(x,y) = 1$$

Let the mappings $\phi : V' \rightarrow V$ and $\psi : E' \rightarrow E$ define an embedding of G' in G. The mapping ϕ is a many-to-one function, and specifies a partitioning of the processors into modules. A symmetric partitioning implies that exactly $\frac{N}{M}$ nodes of G' are mapped to each node of G. ψ is a one-to-many mapping since each edge in G' gets mapped onto some path in G, and ψ specifies the (multi)set of edges which form this path. Let ψ^{-1} be the inverse mapping of ψ; $\psi^{-1}(e)$ specifies the set of edges in G' that are mapped onto the edge e in G.

The total traffic through an edge e of G in one communication step, is denoted by $t(e)$ where

$$t(e) = m \sum_{e' \in \psi^{-1}(e)} p(e')$$

where m is the message length in bits. Since the edge e corresponds to $w(e)$ wires, the time taken to send the data across e is $\frac{t(e)}{w(e)}$ clock cycles. The time T taken to finish a single communication step is

$$T \geq \max_{e \in E} \frac{t(e)}{w(e)} \geq \frac{\sum_{e \in E} t(e)}{\sum_{e \in E} w(e)}$$

$$= \frac{\sum_{e \in E} (m \sum_{e' \in \psi^{-1}(e)} p(e'))}{(MP/2)}$$

because $\sum_{e \in E} w(e)$ is the total number of wires in the inter-module network.

$$= \frac{\sum_{e \in E} (m/N) |\psi^{-1}(e)|}{(MP/2)}$$

because $p(e') = \frac{1}{N}$ for random traffic. Note that $\sum_{e \in E} |\psi^{-1}(e)| = \sum_{e' \in E'} |\psi(e')|$. Define the average path length to be $h = \sum_{e' \in E'} \frac{|\psi(e')|}{|E'|} = \frac{1}{N^2} \sum_{e' \in E'} |\psi(e')|$. Thus

$$T \geq \frac{(m/N) N^2 h}{MP/2} = \frac{2Nmh}{MP}$$

In each communication step, all the N processors send one message of m bits. Therefore for any graph the peak throughput is given by

$$\frac{Nm}{T} \leq \frac{MP}{2h}$$

The smallest value for h is $\frac{M-1}{M}$ because the maximum number of edges in $e' \in E'$ for which $|\psi(e')| = 0$ is $M(\frac{N}{M})^2$. This is because there are M modules with N/M processors each, and for any pair of processors that map onto the same module, $|\psi(e')| = 0$. This implies that $h \geq \frac{1}{N^2}(N^2 - \frac{N^2}{M}) = \frac{M-1}{M}$. Therefore, for any graph, the peak throughput cannot exceed

$$\frac{M^2 P}{2(M-1)}$$

In steady state the complete graph network with $w(e) = \frac{P}{2(M-1)}$ achieves this upper bound. In the complete graph all edges of E' that are not local to a module, are mapped onto only one edge in E, giving $h = \frac{M-1}{M}$. Further, $t(e)$ is the same for all edges, and making $w(e)$ the same for all edges ensures that $\max_{e \in E} \frac{t(e)}{w(e)} = \frac{\sum_{e \in E} t(e)}{\sum_{e \in E} w(e)}$. Therefore we have proved that the best inter-module network to connect M modules with P pins each, is a complete graph. □

Since the complete graph provides the best performance for a fixed cost, we choose the complete graph topology to interconnect the modules. We consider several intra-module network organizations for N=1024 processors. We also consider different module sizes, by choosing different values for s, the number of processors per module. We consider s = 16, 32 and 64, corresponding to M = 64, 32 and 16 respectively. For the sake of fairness, when we make comparisons between networks with different M, we hold the peak throughput constant. Note that holding the peak throughput constant, is equivalent to holding the bisection width constant, since the bisection width is equal to half of the peak throughput. For concreteness, we fix the bisection width at 8192, corresponding to a peak throughput of 16384 bits per cycle, or 16 bits per processor per cycle.

4.2.1 Scalability

It is somewhat surprising that we chose the complete graph to interconnect modules; complete graphs are considered not to be scalable as an interprocessor network. The rationale for this belief, is that they have a very large number $O(M^2)$ edges, with O(M) degree per node. We note however, that we wish the network design to be scalable for large values of N rather than M. The number of modules is not directly related to the number of processors: as we build larger machines, we will have to build them using a taller hierarchy, involving larger modules at the top level. If we are building a network for a 100,000 processors, it is unlikely that the top level of the packaging hierarchy will consist of modules that contain 10 processors each. The number of modules M at any level will depend upon the technology, but not substantially on the number N of processors.

We consider the issue of scalability at length in Chapter 7.

4.3 Intra-module networks

The only standard N processor network known to us that has a partitioning into M modules such that the interconnection between the modules is a complete graph is a Butterfly network. This requires M and N to be powers of 2 and $M = \sqrt{N}$. Figure 4.1(a) shows a butterfly network connecting 16 processors to 16 memories and figure 4.1(b) shows how this network can be partitioned into 4 modules, such that each module has 4 processors and 4 memories. Also note that the nodes in columns 2 and 3 have been reordered to make the connections between these columns local to the module. The numbers on the processors and memories are shown primarily to illustrate the reordering and do not carry any special significance.

Since we need to route access requests from the processors to the memories and the corresponding replies back from the memories to the processors, we need two separate butterfly networks. Therefore, each link in the figure should be interpreted as a pair of directed links. Note that the bundle of wires going between any two modules, A and B, actually consists of 4 different channels, two in each direction. One channel carries access requests from the processors in module A to memories in module B, another carries requests from B to A. Similarly two channels carry responses from A to B and B to A respectively.

Although we started with a *dance-hall* type network, it is useful to associate each

memory with a processor from a practical point of view. This simplifies building the machine since each processor can use a a single memory that is partitioned into private memory and shared memory. While the private memory is accessed directly by the processor, the shared memory is accessible to all the processors via the network. We represent this in figure 4.1(c) by drawing the processors and their associated memories adjacent to each other.

Since the partitioning shown above is applicable when $M = \sqrt{N}$, we first consider the case $M = 32$. Later we will examine the cases $M = 64$, and $M = 16$.

4.3.1 M=32 Modules

Basic butterfly network

We consider a butterfly connecting 1024 processors to 1024 memories; partitioned into 32 modules in the manner of figure 4.1(b).

A bisection width of $B = 8192$ implies that pin-count per module is equal to $4B(M-1)/M^2 = 992$, for a total of $32 * 992 = 31744$ level-1 pins. Each pair of modules is connected by a bundle of 32 wires. Since each bundle has 4 channels, each channel is 8 bits wide. Since the inter-module channels are 8 bits wide, we get a uniform network if we make the intra-module channels also 8 bits wide. We call this network Butterfly.8.8. The first suffix stands for the intra-module channel width and the second suffix for the inter-module channel width.

The interconnections within a module are are shown in figure 4.2

Widening channels

The previous network used a uniform topology that did not take advantage of packaging locality. In general, at the lower levels of the packaging hierarchy we can have more wires and also transfer bits across these wires at greater speeds. Both these characteristics permit us to increase the capacity of the networks at the lower levels of the hierarchy. Instead of separately modeling increased wire density and clock rates, we consider networks that keep the clock rate constant and just use higher wire densities. Accordingly, we consider networks with wider intra-module channels: Butterfly.16.8 and Butterfly.32.8. Both networks have the same topology as Butterfly.8.8, but they have 16 and 32 bit wide intra-module channels respectively.

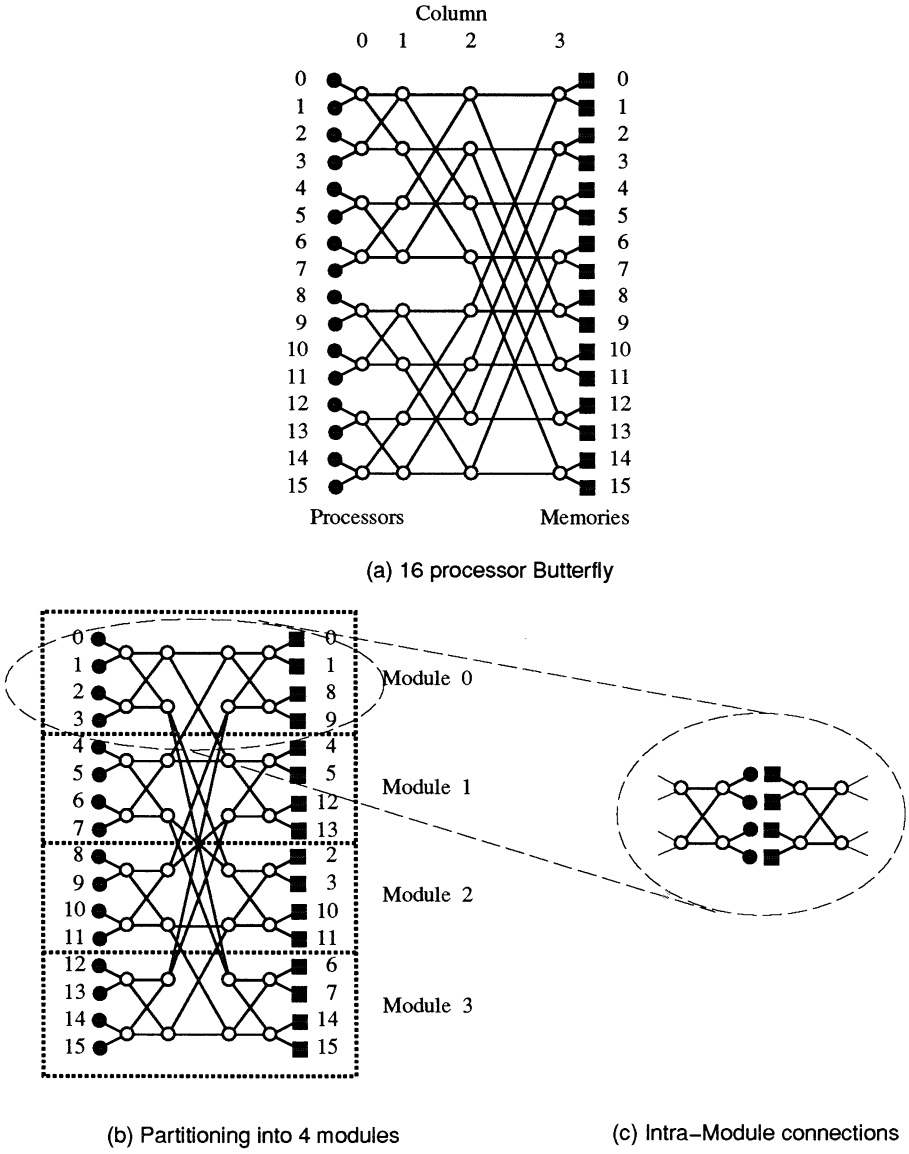


Figure 4.1: Butterfly Network and its partition

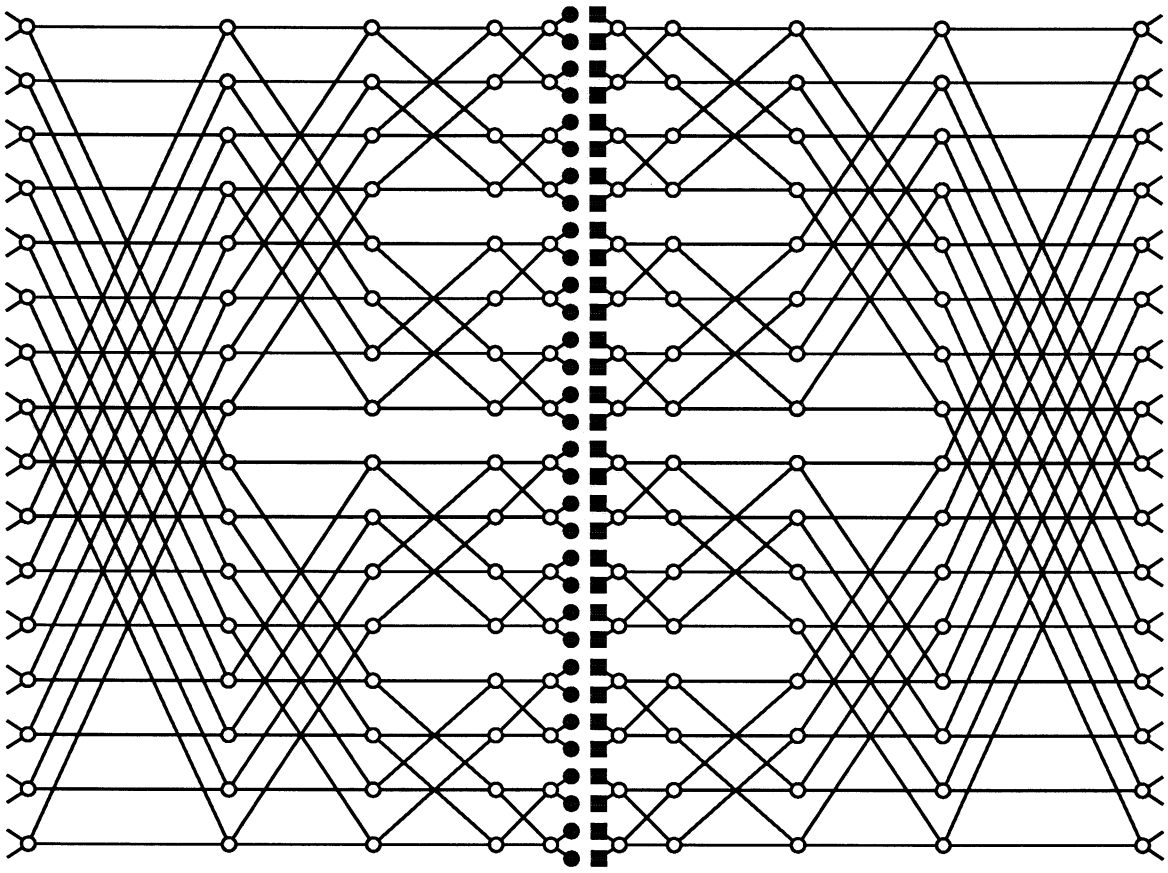


Figure 4.2: Intra-Module connections, Butterfly Network

It is important to point out that the costs of the these networks are different because they use different amounts of resources within a module. However, all of them have the same inter-module network, and if we assume that network cost is dominated by the cost of the top-level network, the difference in the costs of these networks will be small.

Dilation

Another method for utilizing more local wires is dilation. We consider networks Butterfly.8-2dil.8, Butterfly.8-4dil.8, and Butterfly.16-2dil.8. The first network has 2 8-bit wide channels between nodes, the second has 4 8-bit wide channels and the third has 2 16-bit wide channels. If there is more than one message that needs to go out in the same direction, they can be simultaneously sent along the dilated channels. The first network requires the same number of wires per routing node as Butterfly.16.8 while the second and the third use the same as Butterfly.32.8.

In the case of network dilation, we do not dilate the channels connecting into the memories or the processors. If we dilate the channels connecting to the memories, we may have to use multi-ported memories or add some complicated circuitry to prevent multiple accesses from being made simultaneously.

Multibutterfly

A 2-way dilated butterfly can be thought of as a pair of identical butterflies that are merged together. In the case of a multibutterfly [Upf89, LLM90] (or more precisely a *twin-butterfly*), the nodes within each column of one of the two butterflies are permuted in a special fashion before the two butterflies are merged. To generate a multibutterfly with I inputs, we take a collection of random permutations $\Pi = (\pi_0, \pi_1, \ldots, \pi_{\log I - 1})$, where $\pi_x : [0, \frac{I}{2^{x+1}} - 1] \rightarrow [0, \frac{I}{2^{x+1}} - 1]$, and merge the node in row $\frac{jI}{2^{x+1}} + i$ and column x in the first butterfly with node in row $\frac{jI}{2^{x+1}} + \pi_x(i)$ and column x of the second butterfly, for all $0 \leq i \leq \frac{I}{2^{x+1}} - 1$, all $0 \leq j \leq 2^{x+1} - 1$, and all $0 \leq x \leq (\log I - 1)$.

A multibutterfly requires the same number of wires per routing node as a 2-dilated network, but the connections between the adjacent stages are more complex. Using random permutations in the wiring of these networks helps to avoid problems with non-uniformities in load distribution.

Note that the multibutterfly network described above is used only within the

modules. Other network evaluation studies [LLM90, KU91] have considered connecting all the processors using a multibutterfly network. We do not consider such a network since we believe that the multibutterfly cannot be partitioned as well as a butterfly. Partitioning the multibutterfly into modules in a manner similar to a butterfly, necessitates a much larger pin-count per module. We suspect that any other type of partitioning of the multibutterfly also has the same problem.

Fewer memory units

When we increase the channel widths within the packaging modules, we increase the channel widths into the memory units, effectively increasing the service rate for memory access requests made over the network. In such networks, the effective memory bandwidth is greater than the network bandwidth. We observe that it might be possible to use fewer memory units when we have wider intra-module channels without appreciable degradation in performance. Reducing the number of memory units can potentially reduce the cost of the system, although it does not affect cost as per our coarse packaging model. A system with fewer memory units than processors corresponds to building a network using small shared memory multiprocessors as building blocks, instead of single processors.

We accordingly consider Butterfly.32.8.M-512 and Butterfly.32.8.M-256. Both networks have 32 bit wide intra-module channels and 8 bit wide inter-module channels, but the former has only 512 memory units and the latter has 256 memory units. Both networks have 1024 processors, implying that the former uses building blocks which are 2-processor shared-memory multiprocessors, whereas the latter uses 4-processor multiprocessors. Figure 4.3 shows the intra-module network for Butterfly.32.8.M-256. Since there are fewer than 32 memory units per module, there are fewer stages of switching connecting the memory units to the inter-module channels.

Dilating the inter-module network

We consider the network Butterfly.32.4-2dil which has 32 bit wide intra-module channels similar to Butterfly.32.8, but the channels of the inter-module network are 2-way dilated. When we dilate the inter-module network, we reduce the width of each channel by a factor of 2 to keep the pin-count per module the same. Therefore the inter-module channels in this network are 4 bits wide.

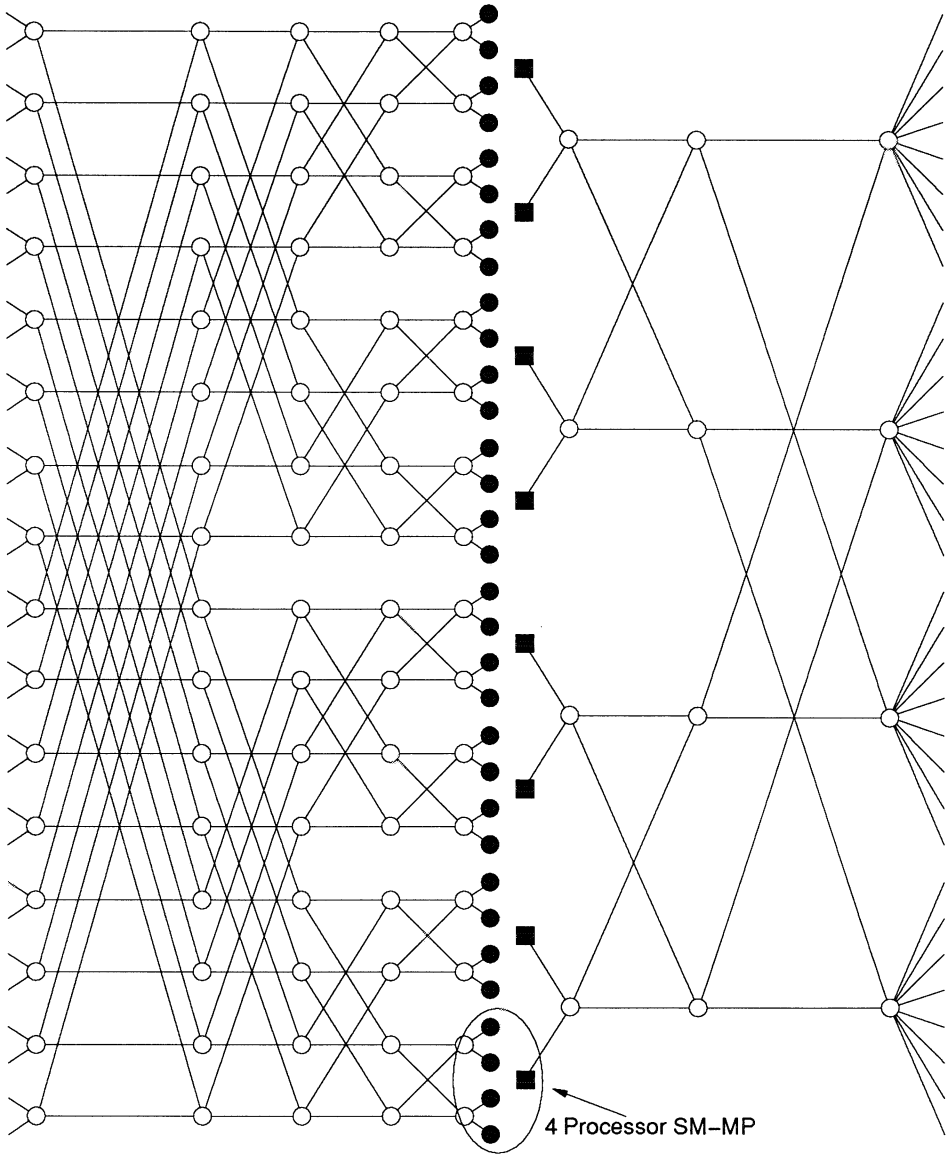


Figure 4.3: Intra-module network: Butterfly.32.8.M-512

Sharing inter-module channels

We note that in all the networks described above, there are two networks: a request network that routes access requests from the processors to the memories, and a response network that routes replies from the memories to the processors. We can merge these two networks so that there is only one pair of channels going between any two modules. We can think of this network as being complementary to the previous network that dilated the inter-module channels.

In order to share the inter-module channels, we need an additional set of routing nodes that reduce the 64 channels of the Butterfly topology to 32 channels. We put these 2x1 merger nodes just before the out-going channels we get the network organization shown in figure 4.4. We call this network Shared-Butterfly. The inter-module channels are 16 bits wide, therefore we only consider topologies Shared-Butterfly.16.16, Shared-Butterfly.16-2dil.16 and Shared-Butterfly.32.16.

In this network the sharing of channels between access requests and responses makes this network susceptible to message deadlock. It is possible to have situations where message buffers get filled up and a circular wait situation gets created. In order to eliminate the possibility of deadlock, we assume that there are very large buffers at the memory units and at the processors. These large buffers guarantee that messages will be drained out of the network either at the processors or the memories. This eliminates the possibility of deadlock. The MIT Alewife machine uses a similar network architecture in which channels are shared between requests and replies. The technique to avoid deadlocks is also similar; when the buffers fill up, messages are emptied from the network into local memory, effectively simulating infinite buffers [A+91].

Cross-bar

To establish an upper bound on the best performance achievable, we also consider the networks Xbar.8.8, and Xbar.32.8. Both networks connect the processors and memories to the inter-module channels using 32 by 32 cross-bars. In the first case both the input and output channels of the cross-bar are 8 bits wide. In the latter case the channels that connect to the processors and memories are 32 bits wide. The inter-module channels remain 8 bits wide as before.

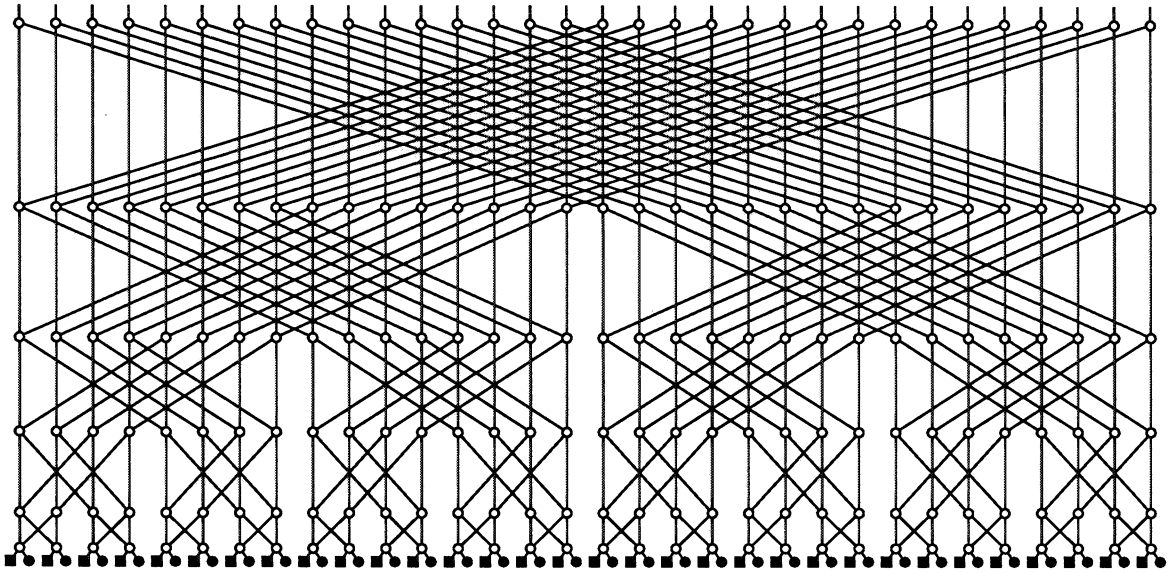


Figure 4.4: Intra-Module Network: Shared-Butterfly

4.3.2 M=64 Modules

We now consider the case where the packaging modules are smaller. We assume that each packaging module is capable of holding 16 processors, with a total of 64 modules for a 1024 processor system. The interconnection between the modules is a complete graph.

To make the comparisons fair, we keep the bisection width the same as before. Since there are 64 packaging modules, a bisection width of $B = 8192$ implies that pin-count per module is equal to $4B(M-1)/M^2 = 504$, for a total of $64 * 504 = 32256$ pins. Each pair of modules is connected by a bundle of 8 wires. Each bundle has 4 channels and each channel will be 2 bits wide.

The first intra-module network we consider is Butterfly.8.2 and is shown in figure 4.5. We build a butterfly on 16 inputs but with a degree 8 fan-out in the last column instead of degree 2. The degree 8 fan-out is required to generate enough channels to connect to all the modules. To balance the bandwidth at the last column of switching nodes, we have to provide 8-bit wide intra-module channels.

We also consider the network Butterfly.32.2, which has 32 bit wide intra-module channels.

To establish an upper bound, we also consider the network Xbar.32.2 in which a cross-bar is used to connect 16 processors to 64 inter-module channels.

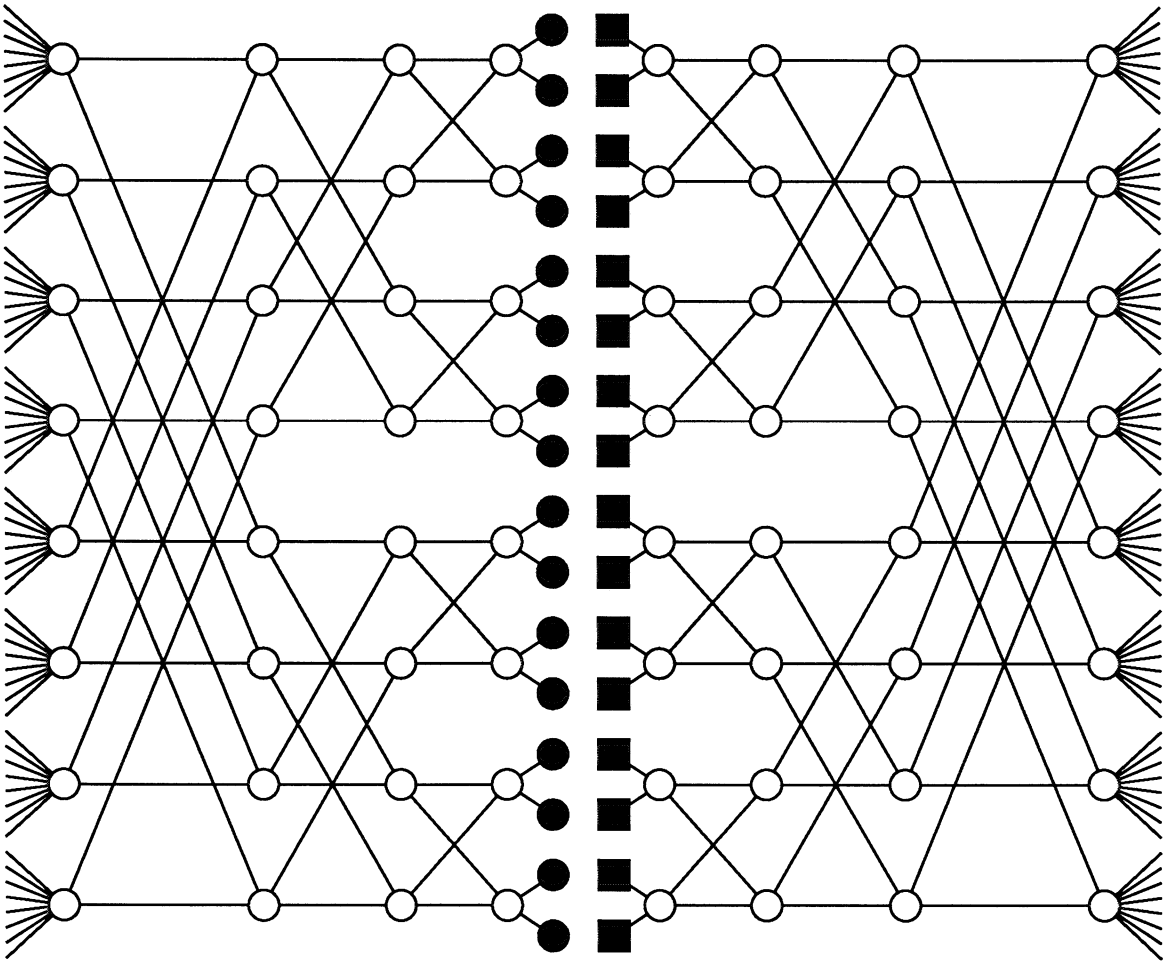


Figure 4.5: Intra-Module Network: Butterfly.8.2

4.3.3 M=16 Modules

Next, we consider the case where the packaging modules are larger. We assume that each packaging module is capable of holding 64 processors, requiring 16 modules for a 1024 processor system. The interconnection between the modules is a complete graph as before.

We derive this network by partitioning a butterfly with 256 inputs in a manner similar to the earlier case Section 4.3.1. Since we have to build a network on 1024 processors, we connect 4 processors to each input of the 256 input butterfly, using a 2 level tree, as shown in figure 4.6.

As before, we hold the bisection width constant at 8192. Since there are 16 packaging modules, a bisection width of $B = 8192$ implies that pin-count per module is equal to $4B(M-1)/M^2 = 1920$, for a total of $16 * 1920 = 30720$ pins. Each pair of modules is connected by a bundle of 128 wires. Each bundle has 4 channels and each channel is 32 bits wide.

Since the inter-module channels are 32 bits wide, we get a uniform network if we make the intra-module channels also 32 bits wide. This is the first network we evaluate and as per our naming conventions, we call this network Butterfly.32.32. We also consider the network Butterfly.128.32, with 4 times wider intra-module channels.

To establish an upper bound, we also consider the network Xbar.32.32 in which a cross-bar is used to connect 64 processors to 16 inter-module channels.

4.4 Simulation Results

4.4.1 Open-network model

Figures 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, and 4.13 show the results for the different networks. The latencies are round-trip latencies, i.e., the time taken for the access request to go from the processor to the memory, make the access and return to the processor. The x-axis is the normalized load expressed as a percentage of the peak load.

We see that in all the open-network graphs, as the applied load increase, the latency initially increases gradually and as the network nears saturation, the latency increases rapidly. The curves extend along the x-axis until the point at which the corresponding generation rate no longer yielded a stable latency. We defer detailed discussion of these

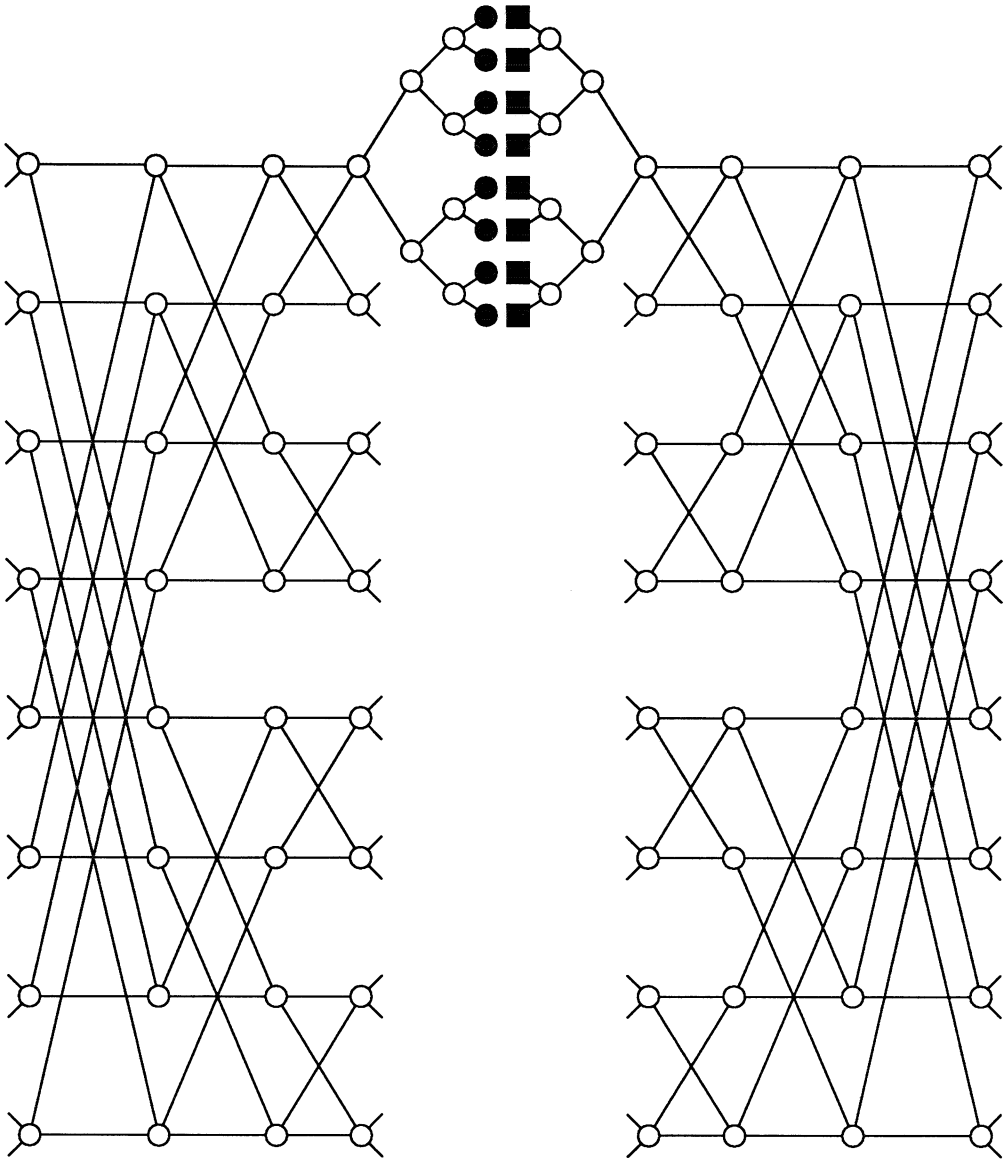


Figure 4.6: Connecting 4 processors to each network input

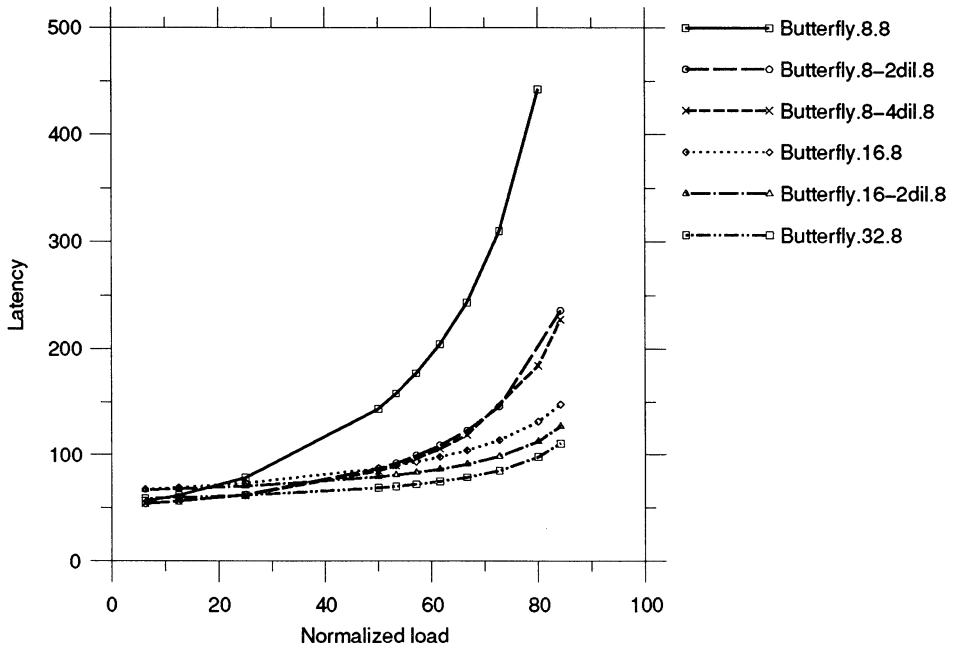


Figure 4.7: Open-Network Model; Butterfly Networks

results to section 4.4.3.

4.4.2 Multithreading work-load model

The performance metric under this work-load model is the average processor utilization, as a function of number of threads and of the rate at which threads make memory accesses. Based on the bisection width of the networks we can see that the peak bandwidth that each network can support is 8 bits per processor per cycle in each direction. This corresponds to a mean access rate of one every 16 cycles, since each access is 128 bits long. We consider mean access intervals of both 16 and 32 cycles.

A mean access interval of 16 cycles can potentially saturate the bisection and corresponds to running the network at 100% load. Since message insertion is linked with the delivery of responses, injecting messages with the mean access interval = 16 does not pose a problem. The processors themselves merely slow down until the injection rate matches the response rate. As the number of threads per processor increases, the processors become increasingly latency tolerant and are able to get more out of the network.

A mean access interval of 32 cycles corresponds to running the network at about 50% load. In this case, since the network is lightly loaded, the processors should be able

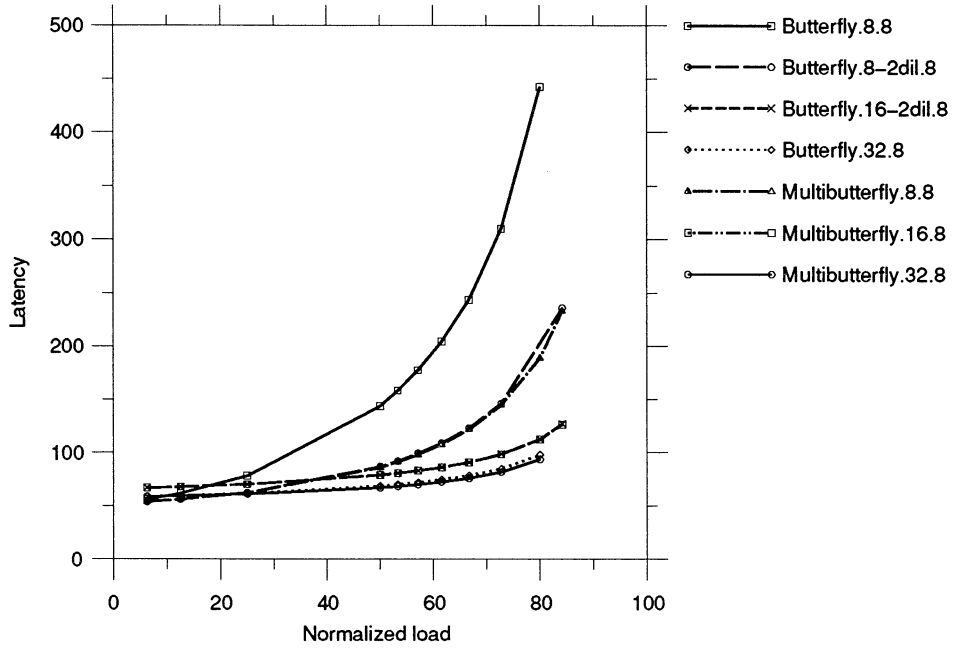


Figure 4.8: Open-Network Model; Multibutterfly Networks

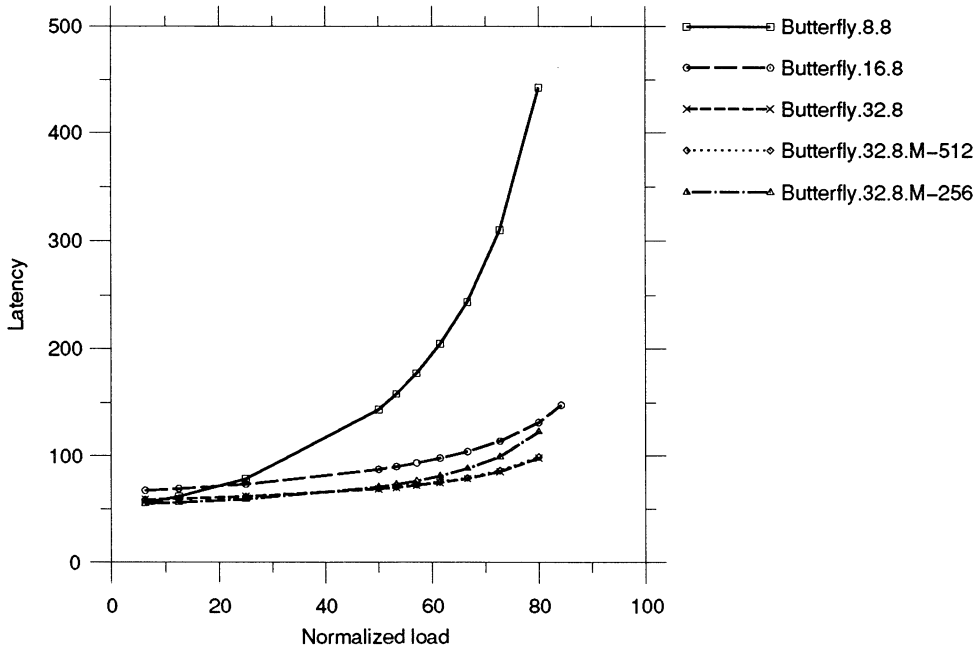


Figure 4.9: Open-Network Model; Fewer Memories

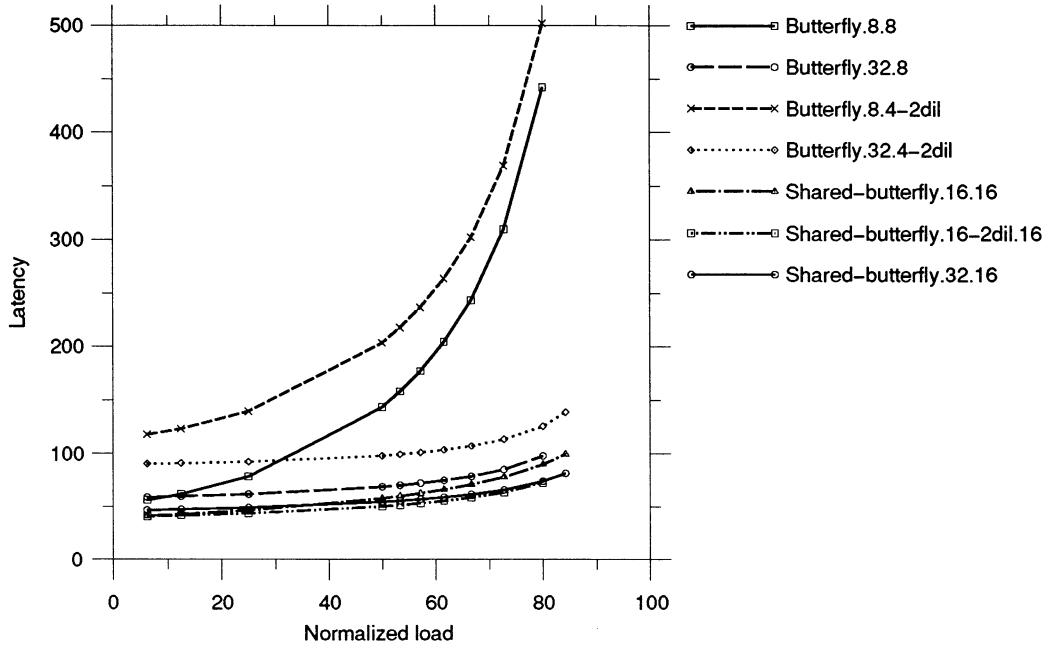


Figure 4.10: Open-Network Model; Sharing/Dilating Inter-module Channels

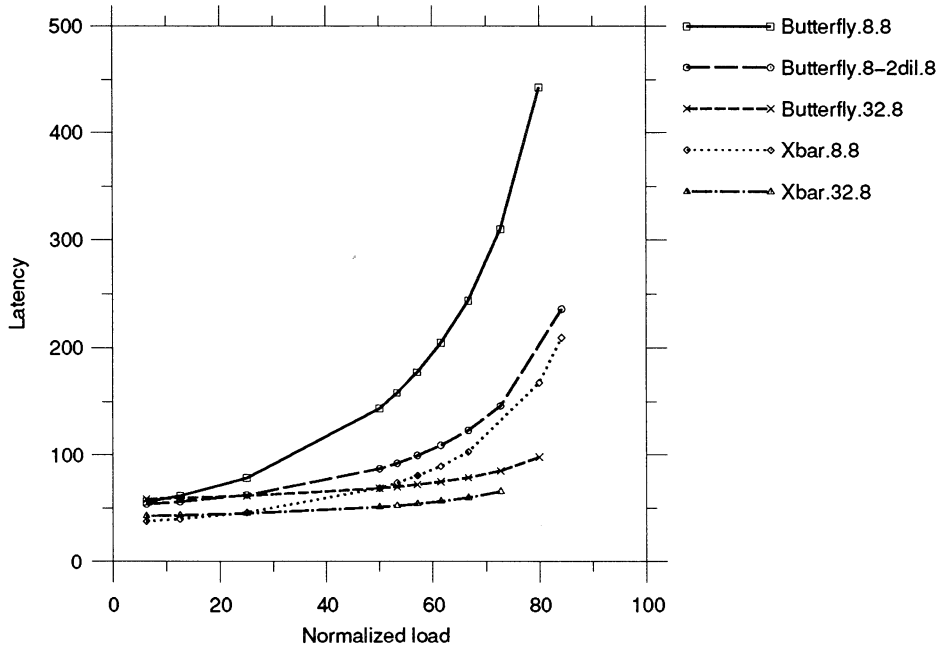


Figure 4.11: Open-Network Model; Cross-bar Networks

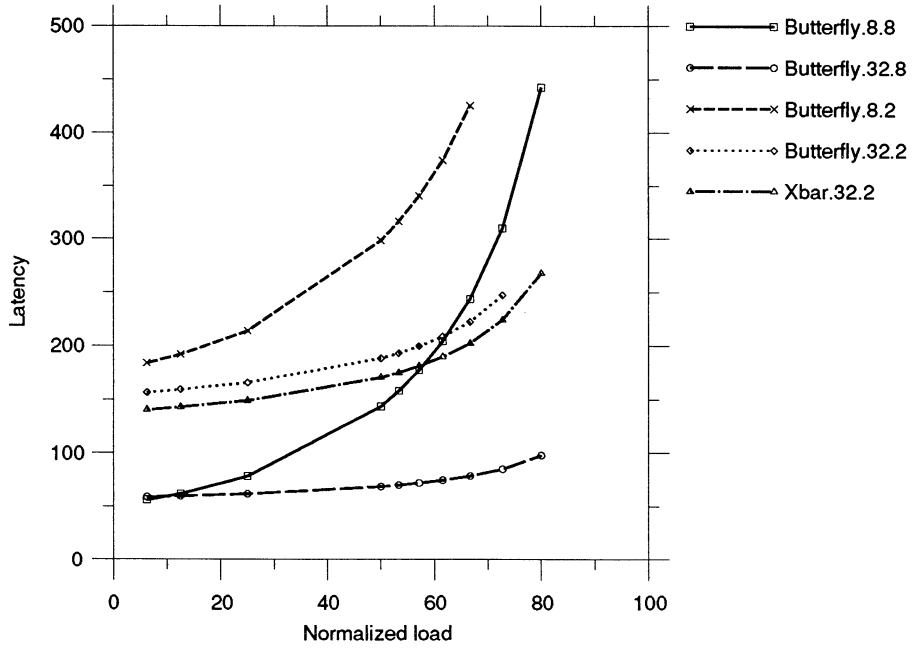


Figure 4.12: Open-Network Model; 64 Module Networks

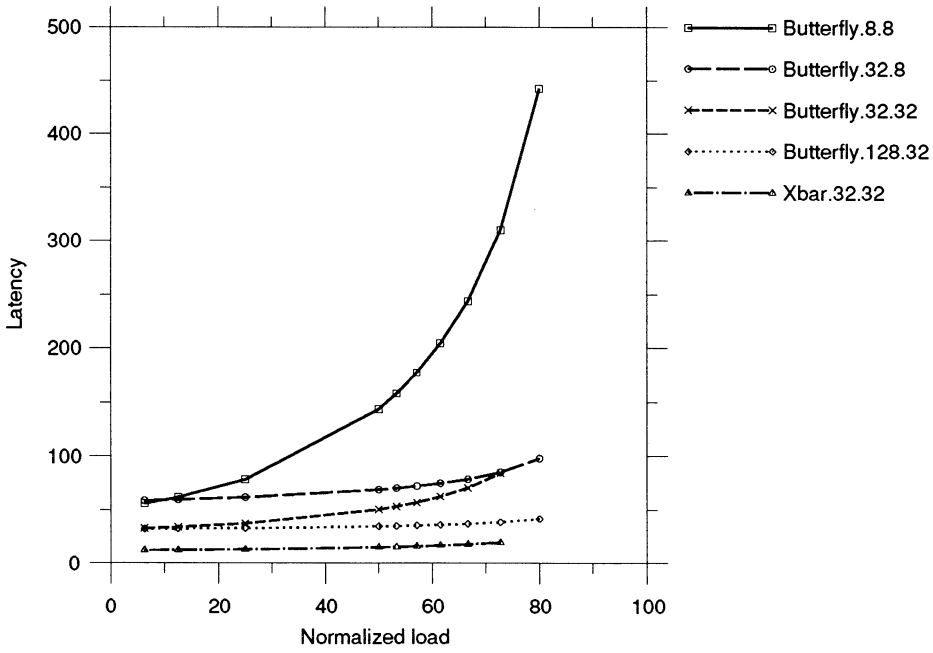


Figure 4.13: Open-Network Model; 16 Module Networks

to achieve almost 100% utilization if they can successfully hide the network latency using multithreading.

Figure 4.14 and 4.15 show the processor utilizations for the different networks when the mean access interval is 16 cycles. The graphs shows five lines corresponding to multithreading factors of 1, 2, 4, 8 and 16.

Figures 4.16 and 4.17 show similar curves for a mean access interval is 32 cycles.

The graphs for both geometric and periodic access patterns are similar to each other, except that the utilizations for the periodic model are higher. This is to be expected because, under the geometric model, there is a greater variance in inter-access times, and if many short inter-access intervals occur together, the instantaneous load increases and the processor utilization gets reduced.

4.4.3 Discussion of results

Dilation versus wider channels

The data corresponding to Butterfly.8.8, Butterfly.8-2dil.8 and Butterfly.8-4dil.8 in figures 4.7, 4.14 and 4.15 indicate that dilation of channels helps in improving performance. This improvement results from the reduced contention for the intra-module channels. Increasing the dilation from 2 to 4 does not seem to have much of an effect, indicating that a dilation factor of 2 is sufficient to eliminate most of the intra-module contention.

As opposed to dilation, widening the channels within a module affects the network performance in three different ways. First, it reduces the contention for intra-module channels. Secondly, it increases the effective memory service rate. Dilation does not increase the effective memory service rate because we do not dilate the channels to the memory units(Section "Dilation" in 4.3.1). Thirdly it increases message latency because of the non-uniformity in channel widths[3]. It is clear that the first effect has a positive influence on network performance. The second effect also helps improve performance because conflicts between accesses at the memories have less of an effect. The third effect hurts network performance because it increases the latency.

When the load is small (in the open-network case) or there are few threads (multithreading), the performance of the networks with wider channels is slightly worse than

[3]When a message transits through a node whose output channel is wider than its input channel, the message suffers a larger delay because it must be entirely buffered before it can be forwarded(Section 3.4).

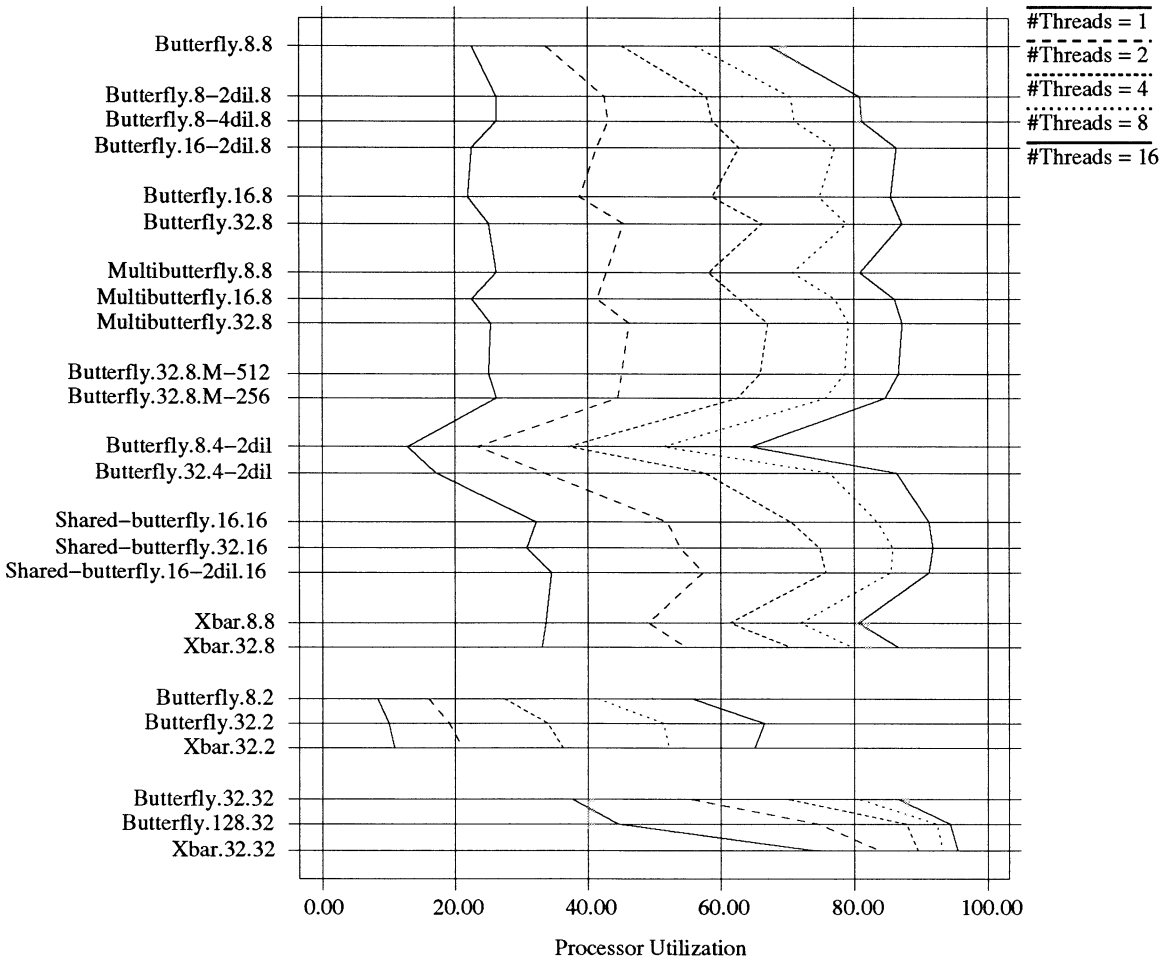


Figure 4.14: Periodic Access Model, access interval = 16 cycles

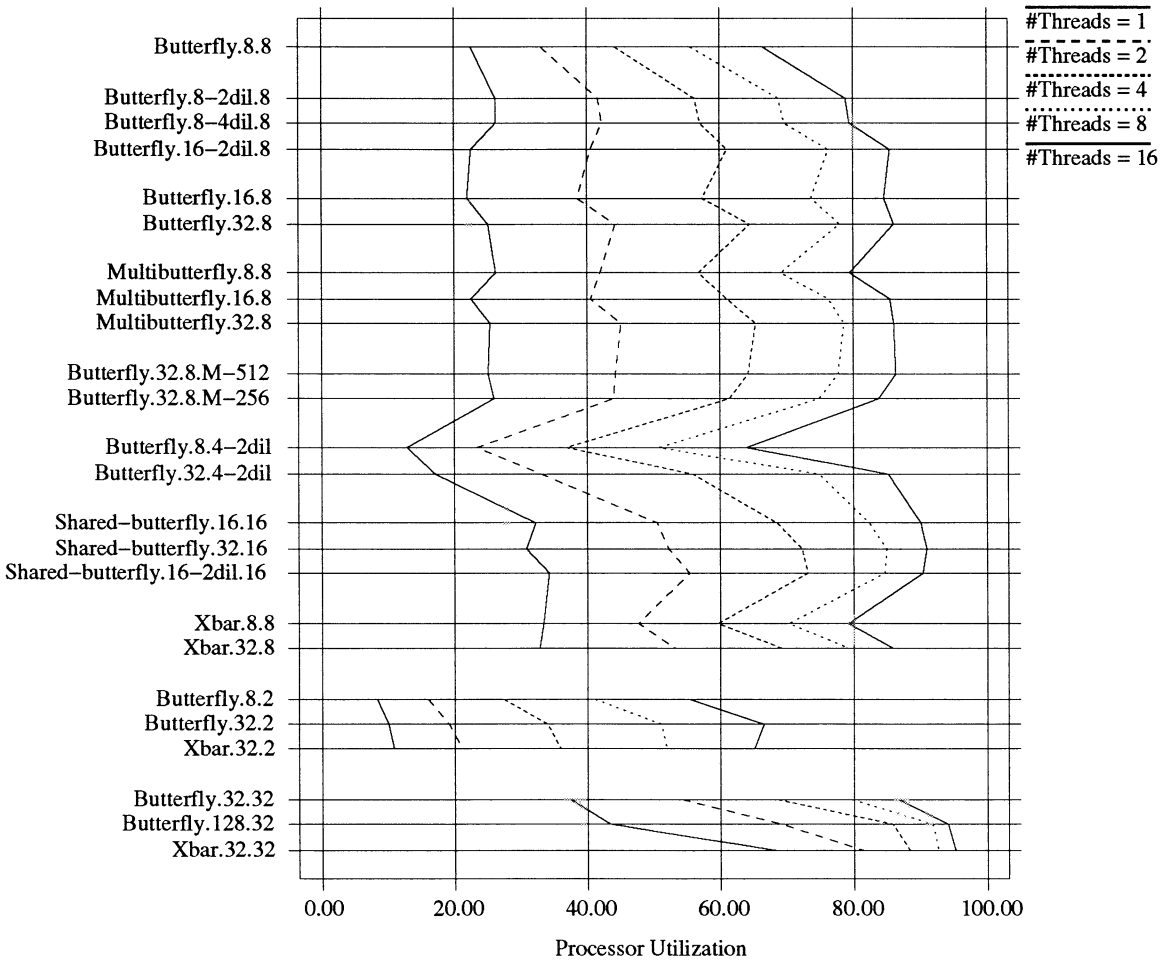


Figure 4.15: Geometric Access Model, mean access interval = 16 cycles

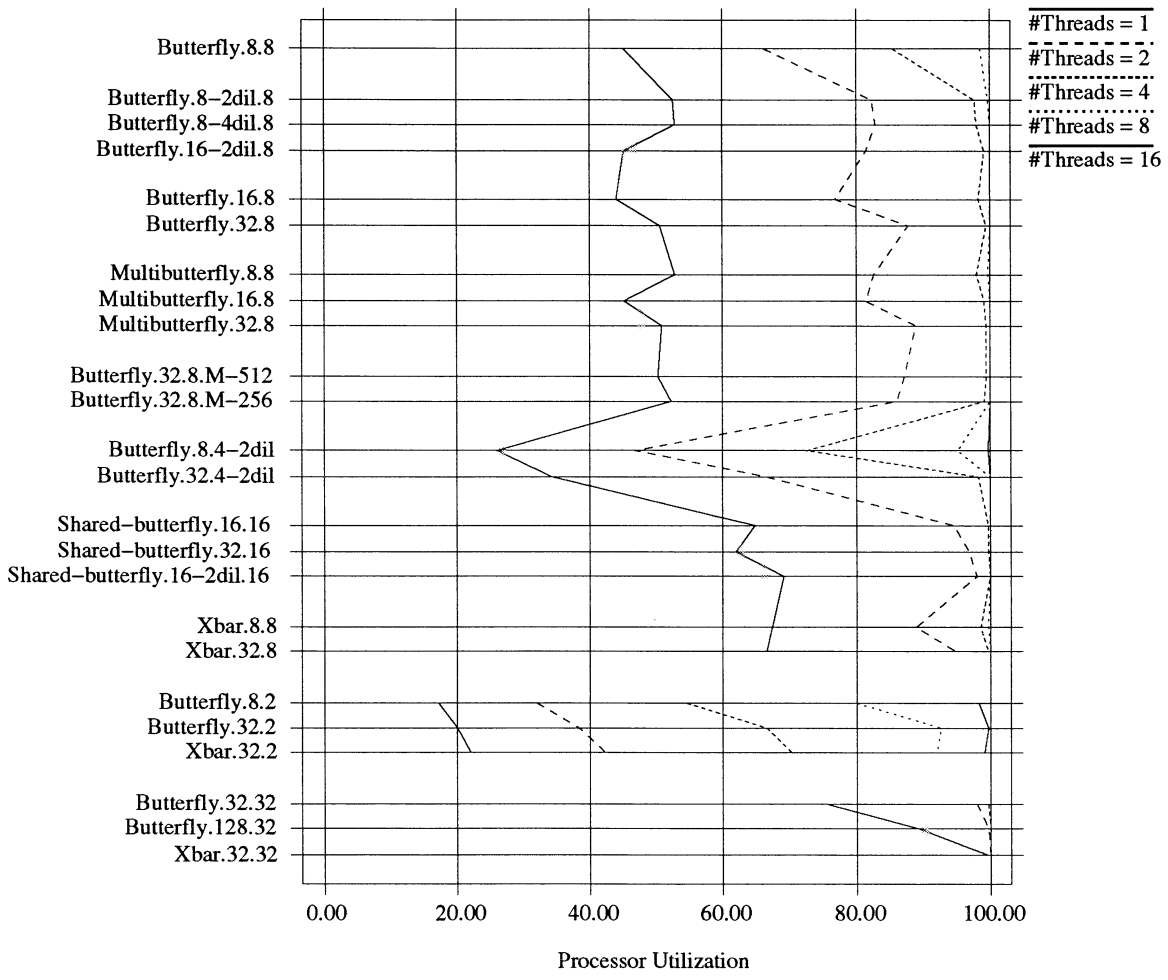


Figure 4.16: Periodic Access Model, access interval = 32 cycles

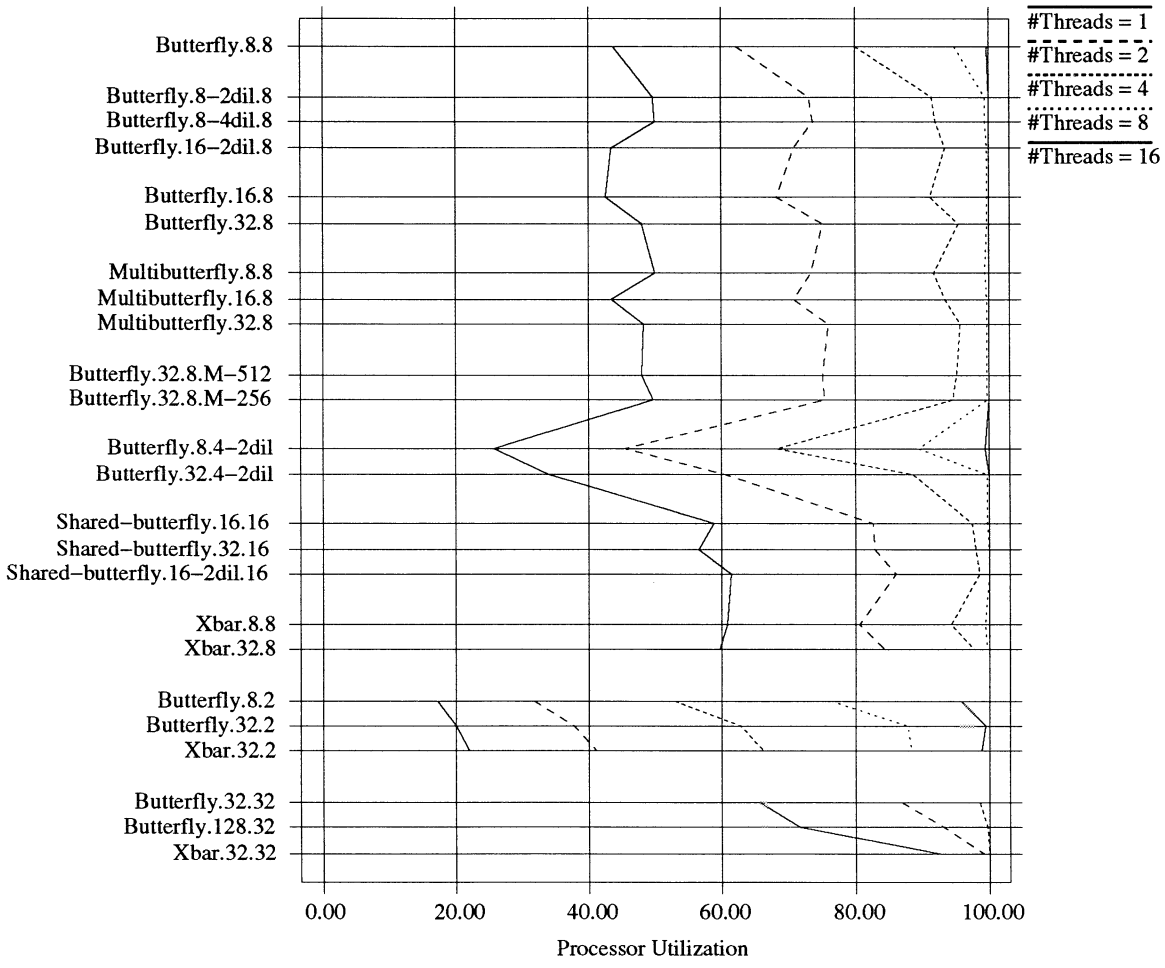


Figure 4.17: Geometric Access Model, mean access interval = 32 cycles

Butterfly.8.8. This is because of the third effect mentioned above. But, when the load on the network is increased, the effects of reduced contention and the higher memory service rate become more predominant. In the long run we can expect all memories to receive approximately the same number of requests because the access distribution is random, but in the short run the access distribution can be uneven. A higher memory service rate helps to reduce the effect of this non-uniformity.

Combining dilation and widening of channels does not seem to help much either, versus simply widening the channels as seen by comparing Butterfly.16-2dil.8 and Butterfly.32.8.

We can also try to compare the effects of dilation versus widening channels analytically using queueing theory. Dilation corresponds to using multiple servers with the same service rate, while widening channels corresponds to increasing the service rate of a single server. If we model the traffic between a pair of routing nodes using a simplistic queueing model consisting of poisson arrival streams and exponential service times, we can show that both dilation and wider channels have comparable performance when the load factor is high, but wider channels are better at lower loads. It is difficult to extend the analysis from a single pair of nodes to the entire network. Further, we need to take into account a variety of factors such as constant service times, bounded queues, packet-switched routing in some nodes due to non-uniform channel widths, contention at the memory units, stalling of processors when all threads have outstanding accesses, etc.

Multibutterfly versus Dilation

The multibutterfly networks require the same number of wires as the 2-dilated networks. The difference between them is the randomized wiring between stages in the multibutterfly. Figure 4.8 and the graphs corresponding to the multithreading workload indicate that the multibutterfly has approximately the same performance as a 2-dilated butterfly. Specifically, Multibutterfly.8.8 has approximately the same performance as Butterfly.8-2dil.8, and Multibutterfly.16.8 has the same performance as Butterfly.16-2dil.8. Simulations of large multibutterflies have shown that they perform better than dilated butterflies [LLM90, KU91], but the multibutterflies in our networks are too small to display any noticeable difference in performance. Since we use multibutterflies only for the intra-module network, they only have 32 inputs and five stages. For networks of this size, there is hardly

any benefit from using randomized connections between stages.

Fewer memories

Butterfly.32.8.M-256 has the same amount of memory bandwidth as Butterfly.8.8, since the channel width to memory is 4 times as wide and there are 4 times as fewer memory units. Similarly Butterfly.32.8.M-512 has the same memory bandwidth as Butterfly.16.8. Comparing these pairs of networks illustrates that using fewer memory units with wider channels, results in a significant improvement in performance. The reason for this improvement in performance is the increased memory service rate at each memory unit. Even though a reduction in the number of memory units increases the load on each unit, temporary non-uniformities in the load distribution have less of an effect due to the higher service rate.

Instead of comparing networks based on equal memory bandwidth, we can also compare them based on identical memory channel width. We see that the performance of Butterfly.32.8 is indistinguishable from that of Butterfly.32.8.M-512 and Butterfly.32.8.M-256. The effect of the lower memory bandwidth is not felt since network bandwidth is the main limiting factor. Figure 4.9 indicates a slight increase in latency as the load increases because of increased contention for the fewer memory units, but this increase in latency is small and does not appear to affect the processor utilizations under the multithreading workload model.

Dilating versus sharing inter-module channels

The graphs indicate that dilating the inter-module channels causes a significant drop in performance, while sharing the inter-module channels improves performance over the base network. The inter-module channels are the bottlenecks as far as network throughput is concerned. A network that achieves a higher utilization of the inter-module channels will, therefore, achieve higher throughput. When we configure the available inter-module bandwidth as a pair of narrower channels (Butterfly.32.4-2dil), it is more difficult to achieve high channel utilization because we need twice as many independent messages to keep all the wires busy. In contrast, when we share the channels we need only half as many messages. This explains the difference in performance. However, sharing the inter-module channels between the forward and the return networks makes the network susceptible to deadlock

and we need to devise some mechanism to avoid deadlock (We assumed that large buffers were available at the processors and the memory units).

Cross-bar

The intra-module cross-bar network completely eliminates the possible contention for links within a module. It also has a lower minimum latency because we assume that it imposes only a single cycle pipeline delay as opposed to one cycle per stage in a multi-stage network.

In figure 4.11 we see that the graph for Xbar.8.8 is approximately parallel to that of Butterfly.8-2dil.8 indicating that both networks have similar performance with respect to contention. The graph for the cross-bar network is lower because of its lower minimum latency. This difference does not show up under the multithreading workload (figures 4.14 and 4.15) indicating that the effect of this difference in latency is small. The comparison between Xbar.32.8 and Butterfly.32.8 is similar. This indicates that providing extra local bandwidth either through dilation or wider channels is successful in eliminating almost all the contention for the intra-module channels.

Multithreading

It can be seen that all networks achieve significant improvement in processor utilization as the number of threads is increased. This demonstrates that multithreading is successful in masking the access latencies. Increasing the number of threads per processor corresponds to operating the network farther down the x-axis on the load-latency graphs of the open-network model. Though this increases the latency experienced by the messages, the processor is able to better overlap the latency with useful computation. It is important to note that even when the number of threads is small(2-4), there is a significant improvement in processor utilization in relation to the utilization corresponding to a single thread. As we increase the number of threads further, the processor utilizations improve, but the marginal increase per added thread diminishes because increasing the number of threads also increases the loading of the network which increases latency.

Performance at low network loads

The graphs corresponding to an inter access time of 32 cycles (figure 4.16 and 4.17) show that at high levels of multithreading, it is possible to get processor utilizations of almost 100%. Since the load never gets so high as to make the latencies increase dramatically, multithreading can completely hide the network latency. For most of the networks we evaluated we can see that about 4 threads can achieve this (The networks corresponding to $M = 64$ needed about 16 threads to achieve 100% utilization since their latencies were high). Naturally, as we scale the machine to more processors we expect that we will need more threads, but we expect that the number of threads will not grow faster than $\log N$ since the network latencies under cut-through routing will grow slower than $\log N$.

We also see that at low levels of multithreading, the effects due to non-uniform channel widths are more pronounced. Since the network load is small, the increase in latency due to non-uniform channel widths dominates the effects due to reduced contention and increased memory service rates.

4.4.4 Choosing module size

We can also compare the results of the cases $M = 16$ and $M = 64$ with $M = 32$, but in doing so we should keep in mind that the costs of these networks are not the same, because they use different numbers of modules and different pin-counts per module. Since all the networks have the same peak throughput, the difference in cost vectors illustrates a trade-off between the number of modules and the pin-count per module. Reducing the number of modules by a factor of 2 requires increasing the pin-count per module by the same factor of 2, in order to maintain the same peak throughput.

Our results in comparing the performance across different module sizes are similar to the results of Cypher [Cyp90] in that the performance of the network is proportional to the product of the number of modules and the pin-count per module. This is to be expected since the cost model used in this chapter is similar to Cypher's pin-requirements model. However, the network design problem we consider here is substantially different from the one considered by Cypher.

Although the costs are different for different module sizes, since the peak throughput is the same, we can compare the fraction of peak throughput that each of these networks achieves. We see that using larger modules leads to better performance than using smaller

modules. When we increase the module size by a factor of 2, we have a complete graph on half as many modules. This means that number of channels leaving a module drops by a factor of 2. Also the pin-count per module increases by a factor of 2, making the inter-module channels 4 times as wide. We already observed that we were able to improve performance when we organized the same number of inter-module wires as fewer wider channels. The same effect is observed in a more pronounced manner when we vary the module size. This explains the difference in performance between $M = 64$, $M = 32$, and $M = 16$.

4.5 Summary

We started with a simple model of the packaging hierarchy that consisted of only two levels and used a simple network cost model that had only two parameters; the number of top-level modules and the top-level pin-count. We fixed the network cost by fixing both the parameters and compared different networks in terms of their performance. We showed that for a fixed pin-count, the best network at the top level of wiring was a complete graph interconnect where each module was connected to every other module. At the lower level the packaging hierarchy, we examined several alternatives. We did not exhaustively examine all the possible choices at the lower level. However, we demonstrated that simple butterfly networks with wider channels at the lower level were able to match the performance of the best possible network (cross-bar), indicating that exploiting *packaging locality* can lead to better networks. Our results, therefore, argue in favor of designing networks on the basis of the availability of packaging resources, rather than designing networks first and packaging them later.

Chapter 5

Model 2: Pin-counts and bundles

In this chapter, we consider a model that is slightly more complex than model 1. In the previous chapter, we saw that the complete graph topology had the maximum performance for a fixed pin-count per module. However, the complete graph also had the maximum number of bundles per module of all the topologies considered in the previous chapter. In many packaging technologies, a complete graph interconnect may be considered expensive because it uses many bundles. For example, if the technology used in the interconnection of modules is cables with connectors at each end, we may have fixed per-cable and per-connector costs, independent of the total number of wires in the cables and connectors. Under such a packaging technology a network in which the wires leaving a module are organized into a smaller number of thicker bundles, will be cheaper than one in which the same number of wires are organized as a larger number of thinner bundles. In order to model this difference in costs, we include the number of bundles per module in the cost vector of Model 2.

Recall that model 2 is based on the same two level hierarchy as model 1 (inter-module and intra-module connections), but the cost vector has three dimensions: the number of modules, pin-count and number of bundles per module. As in model 1, we ignore the costs of the intra-module network.

5.1 Inter-module Networks

Under model 1, we compared the performance of different inter-module networks after fixing the cost vector. Similarly, under model 2, we first compare networks for a fixed

cost vector (Section 5.1.1). However, this approach does not illustrate the trade-offs between the different components of the cost vector. To examine these trade-offs, we compare the cost vectors of different networks of the same estimated performance (peak throughput). For this comparison we shall hold the number of modules constant and compare networks in terms of pin-count and number of bundles. We shall see that reducing the number of bundles per module requires each module to have a larger pin-count (Section 5.1.2).

5.1.1 Fixing all three parameters

We showed in Chapter 4 that for a fixed pin-count per module, the peak throughput of a network is inversely proportional to h, the average number of hops made by a message over the inter-module network. This result is valid even under model 2, but under model 2 we also fix the number of bundles per module. Therefore, to maximize the performance, we have to minimize the number of hops subject to the bundling restriction. Clearly, if the number of bundles per module is greater than or equal to $M-1$, we can minimize h by building a complete graph interconnect between the modules, as we did in Chapter 4. The problem we consider in this section is: How do we minimize h, given a tighter restriction on the number of bundles?

A related problem that has received much attention in graph theory is the (Δ, D) graph problem [BDQ86]. The (Δ, D) graph problem consists of finding the largest graph in which the node degree is Δ and the diameter is D[1]. In the graph theoretic sense, the number of bundles per node is the same as its degree, and the diameter of a graph is usually an upper bound on h. While the (Δ, D) problem is concerned with finding the graph with the largest number of vertices for a fixed D, we are interested in the complementary problem, namely finding the smallest h for a fixed number of modules. For most of the interesting graphs (graphs that have small D for fixed Δ and a given number of nodes), h is usually quite close to D. Therefore, a solution to the (Δ, D) problem and a solution to the problem we are interested in, are closely related.

However, it is well known that the (Δ, D) problem is difficult [BDQ86]. For many values of Δ and D, the (Δ, D) problem is still open. We can show by means of a simple counting argument that the number of nodes in a graph with degree Δ and diameter D, cannot exceed $1 + \Delta + \Delta(\Delta-1) + \Delta(\Delta-1)^2 + \cdots + \Delta(\Delta-1)^{D-1} = \frac{\Delta(\Delta-1)^D - 2}{\Delta - 2}$. This

[1]The diameter of a graph is the maximum distance over all pairs of vertices, where distance between two vertices is defined as the shortest path between them.

bound is due to E. F. Moore (circa 1958). For some small values of Δ and D there exist graphs that achieve the Moore bound, but for other values, the largest known graphs are much smaller than the Moore bound. In some of these cases, researchers have been able to prove that the largest known graphs are optimal, but for a large number of other cases, we do not know if larger graphs exist.

The apparent complexity of the (Δ, D) problem suggests that the problem we are interested in is also complex. Therefore, instead of trying to find a network that achieves the *maximum* performance, we shall try to find a network that is close to the optimal.

One possibility is to search for a network whose performance is off by a small constant factor from the optimal. Consider a network on M nodes with diameter D and degree Δ. Let the optimal degree be Δ_{min}, when the diameter is D, and the optimal diameter be D_{min} when the degree is Δ. In other words, (M, D, Δ_{min}) and (M, D_{min}, Δ) satisfy the Moore equation. Further, let us assume that $\Delta = c\Delta_{min}$ where c is a small constant. For a fixed pin-count and number of bundles, we have already argued that, to a first approximation, the performance is inversely proportional to the diameter. Therefore, the network with parameters (M, D_{min}, Δ) is optimal, because any other network on M nodes and degree Δ has a greater diameter. Further, the performance of the optimal network is greater than the performance of the network with parameters (M, D, Δ) by a factor of D/D_{min}, the inverse ratio of the diameters.

Since (M, D_{min}, Δ) and (M, D, Δ_{min}) satisfy the Moore equation, $M \approx \Delta^{D_{min}}$ and $M \approx \Delta_{min}^{D}$. Therefore, $D/D_{min} \approx \log \Delta / \log \Delta_{min}$. But we assumed that $\Delta = c\Delta_{min}$. Therefore, $\frac{D}{D_{min}} \approx 1 + \frac{\log c}{\log \Delta_{min}}$. This implies that so long as Δ is within a small constant factor of Δ_{min}, the performance of the network will be very close to the optimal network with the same pin-count and number of bundles (Network with parameters (M, D_{min}, Δ)).

One family of networks for which $\Delta = c\Delta_{min}$ is the family of products of complete graphs. A product graph is defined as follows: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The product of G_1 and G_2, denoted by $G_1 \times G_2 = (V, E)$ where $V = V_1 \times V_2$ and $E = \{((u_1, u_2), (v_1, v_2)) | ((u_1 = v_1) \wedge (u_2, v_2) \in E_2) \vee ((u_2 = v_2) \wedge (u_1, v_1) \in E_1)\}$. We call a product of two complete graphs a 2-hop network. In general an n-hop network is a product $G_1 \times G_2 \times \cdots \times G_n$, where each G_i is a complete graph. For a fixed total number of nodes, the number of bundles needed by an n-hop network is minimized when each of the G_is is of the same size, i.e., a complete graph on $\sqrt[n]{M}$ nodes. We use the term n-hop network to refer to a product graph of complete graphs because the diameter of the graph

is n. We can think of the nodes in an n-hop network as being labeled using a vector of length n; $(m_0, m_1, \ldots, m_{n-1})$. Any pair of nodes whose labels differ in exactly one of the m_is are connected by an edge. Therefore, we can go from source node $(s_0, s_1, \ldots, s_{n-1})$ to destination node $(d_0, d_1, \ldots, d_{n-1})$ via the node: $(s_0, s_1, \ldots, s_{n-1})$, $(d_0, s_1, \ldots, s_{n-1})$, $(d_0, d_1, \ldots, s_{n-1})$, ..., $(d_0, d_1, \ldots, d_{n-1})$. When the number of hops is equal to 1, we have a complete graph, and when the number of hops is $\log M$, the resulting topology is a hypercube.

In an n-hop network, the degree of each node is $n(\sqrt[n]{M} - 1)$, which can be shown to be approximately n times the degree of the optimal network as specified by the Moore equation. Therefore, the performance of the n-hop intermodule network is close to the optimal for small values of n. In fact, we can prove that when $n = 2$, an n-hop network is optimal, based on the value of h rather than the network diameter.

Formally we have the following claim:

Claim 5.1 *When we have M modules with P pins each, the pins being organized as no more than $2(\sqrt{M} - 1)$ bundles, and each module contains an equal number of processors, the inter-module network that achieves the maximum peak throughput is a 2-hop network in which each edge corresponds to $P/2(\sqrt{M} - 1)$ wires. The peak throughput of this network is $\frac{MP}{4(1-1/\sqrt{M})}$*

For a fixed M and P, we have already proved that the peak throughput is bounded by $MP/2h$. The average number of hops is $h = \sum_{i=0}^{\infty} ip_i$, and in the case of a 2-hop network, $p_0 = 1/M$, $p_1 = 2(\sqrt{M} - 1)/M$, and $p_2 = 1 - p_0 - p_1$. Therefore, for a 2-hop network $h = 2(1 - 1/\sqrt{M})$. Further, the load on each edge of a 2-hop network is the same implying that making all the edges have an equal number of wires implies that the network achieves the upper bound on the peak throughput: $\frac{MP}{2h} = \frac{MP}{4(1-1/\sqrt{M})}$.

We shall now show that it is not possible to build a network with the same number of modules, pins and bundles such that it has a larger peak throughput. Assume that such a network is possible. Clearly this network should have a smaller average number of hops. We cannot reduce the average number of hops by increasing p_0, since p_0 is fixed at $1 - 1/M$. We cannot increase p_1 above $2(\sqrt{M} - 1)/M$, because this would require increasing the node degree beyond $2(\sqrt{M} - 1)$. Therefore, we cannot reduce h below that of the 2-hop network, implying that we cannot build a network with greater peak throughput than the 2-hop network for the above set of parameters. □

Other networks that have node degree to within a constant factor of the optimal are de Bruijn graph and Kautz graphs [BP89]. A de Bruijn network with diameter n, on M nodes has node degree $2\sqrt[n]{M}$, which is closer to the optimal than an n-hop network for $n \geq 3$ and large M. Kautz graphs are even closer to the optimal than de Bruijn graphs. However, the difference in performance between the n-hop network and the other graphs is unlikely to be significant, since all networks are already extremely close to the optimal. In this chapter, we primarily consider the 2-hop and the 3-hop networks for purposes of evaluation. Some of the advantages of n-hop networks over de Bruijn and Kautz networks are symmetry and regularity. Besides, for the sizes of networks we are concerned with in this chapter, the differences in pin-counts and number of bundles, between the n-hop networks and de Bruijn and Kautz graphs are minimal.

5.1.2 Comparing cost for fixed performance

We note that the peak throughput of a 2-hop network on M modules and P pins per module is $\frac{MP}{4}(1 - 1/\sqrt{M})$, while that of a complete graph with the same number of modules and pins, is $\frac{MP}{2}(1 - 1/M)$. This means that the 2-hop network has approximately half the performance of a complete graph when we keep the pins per module constant. If we wish to design a 2-hop network with the same performance, we have to approximately double the pin-count. Therefore, we see that there is a trade-off between the pin-count and the number of bundles. We have obtained a reduction in the number of bundles from $M - 1$ to $2(\sqrt{M} - 1)$ by doubling the pin-count per module. This trade-off is illustrated by table 5.1. Since the bisection width is a good approximator of network performance, the table holds the bisection width constant across the different topologies.

We can clearly see from the table that as the number of bundles decreases, the pin-counts increase. Intuitively we can see the reason behind this trade-off. As we reduce the number of bundles we also increase the number of hops that messages need to make. As the number of hops increases, we need more wires in order to achieve the same network capacity resulting in an increase in pin-counts.

In this chapter we consider two different values for M, the number of modules; M=32 and M=64, corresponding to a limit on the number of processors per module at 32 and 16 respectively. For detailed analysis, we choose the complete graph, 2-hop and 3-hop intermodule networks. We choose these networks because we believe that at the higher

Inter-Module Topology	Bisection Width	Module Pin-count	Bundles Per Module	Total #Bundles
Complete graph	B	$(4B/M)\frac{M-1}{M}$	$M-1$	$M(M-1)/2$
2-hop Network	B	$(8B/M)\frac{\sqrt{M}-1}{\sqrt{M}}$	$2(\sqrt{M}-1)$	$M(\sqrt{M}-1)$
3-hop Network	B	$(12B/M)\frac{\sqrt[3]{M}-1}{\sqrt[3]{M}}$	$3(\sqrt[3]{M}-1)$	$3M(\sqrt[3]{M}-1)/2$
n-hop Network	B	$(4nB/M)\frac{\sqrt[n]{M}-1}{\sqrt[n]{M}}$	$n(\sqrt[n]{M}-1)$	$nM(\sqrt[n]{M}-1)/2$
2-hop de Bruijn	B	$8B/M$	$2\sqrt{M}$	$M\sqrt{M}$
3-hop de Bruijn	B	$12B/M$	$2\sqrt[3]{M}$	$M\sqrt[3]{M}$
n-hop de Bruijn	B	$4nB/M$	$2\sqrt[n]{M}$	$M\sqrt[n]{M}$
Butterfly	B	$4B\log M/M$	4	$2M$
CCC	B	$6B\log M/M$	3	$3M/2$
Hypercube	B	$2B\log M/M$	$\log M$	$M\log M/2$
d-dim Toroid	B	$dB\sqrt[d]{M}/M$	$2d$	Md
3-dim Toroid	B	$3B/M^{2/3}$	6	$3M$
2-dim Toroid	B	$2B/\sqrt{M}$	4	$2M$
Ring	B	B	2	M

Table 5.1: Characteristics of topologies used to connect M packaging modules

levels of the packaging hierarchy, the number of pins per module is a more important factor than the number of bundles. We have already evaluated the networks for a complete graph inter-module network in the previous chapter, and here we consider a 2-hop network for $M = 32$, and 2-hop and 3-hop networks for $M = 64$.

Interestingly, it turns out that any n-hop inter-module network can be generated by appropriately partitioning a butterfly. A butterfly network of size $N = 2^{(n+1)x}$ can be partitioned into 2^{nx} modules so that the interconnection between the modules is a n-hop network. A $2^{(n+1)x}$ butterfly has $(n+1)x$ columns. We separate these columns into $(n+1)$ groups of x columns each and shuffle the rows such that the connections within each group form butterflies of size 2^x. Next we partition the network into 2^{nx} modules such that each module has 2^x rows and $(n+1)x$ columns. The edges that connect between the modules form a n-hop network. Note that this construction only applies when the number of processors is of the form $2^{(n+1)x}$, and the number of modules is 2^{nx}. This construction can be easily extended to the case where the number of processors is not of the form $2^{(n+1)x}$, by building a network of size $2^{(n+1)x}$, partitioning it into 2^{nx} modules and connecting multiple processors to each network input.

We can also build an n-hop network when the number of modules M is not of the form 2^{nx}, but the numbers in table 5.1 will not apply directly. The table assumes that the n-hop network topology is a n-fold product of complete graphs of size $\sqrt[n]{M}$. If the nth root of M is not an integer, the sizes of the complete graphs that form the product will not be equal. The number of bundles and the pin-count will be slightly larger than that shown in the table.

Since 32 is not a perfect square, we use the topology $K_4 \times K_8$, where K_i denotes a complete graph on i nodes and x denotes graph product. Figure 5.1 shows this graph. For the sake of simplicity, the figure only shows one of the 4 K_8s. The actual topology, will have 3 more K_8s, one on each node of the K_4s. In this topology, there are 10 edges per node, for a total of 160 bundles (in comparison to 31 bundles per node or a total of 496 bundles in a complete graph on 32 modules).

When M=64, we use the 2-hop network $K_8 \times K_8$, and the 3-hop network $K_4 \times K_4 \times K_4$. Figure 5.2 shows the 2-hop network. The figure shows only one of 8 complete graphs. The number of bundles per module is 14, or a total of 448 (in comparison to 63 bundles per module, or a total of 2016 in a complete graph). Figure 5.3 shows the 3-hop network. For the sake of simplicity, the figure does not show all the edges. This network

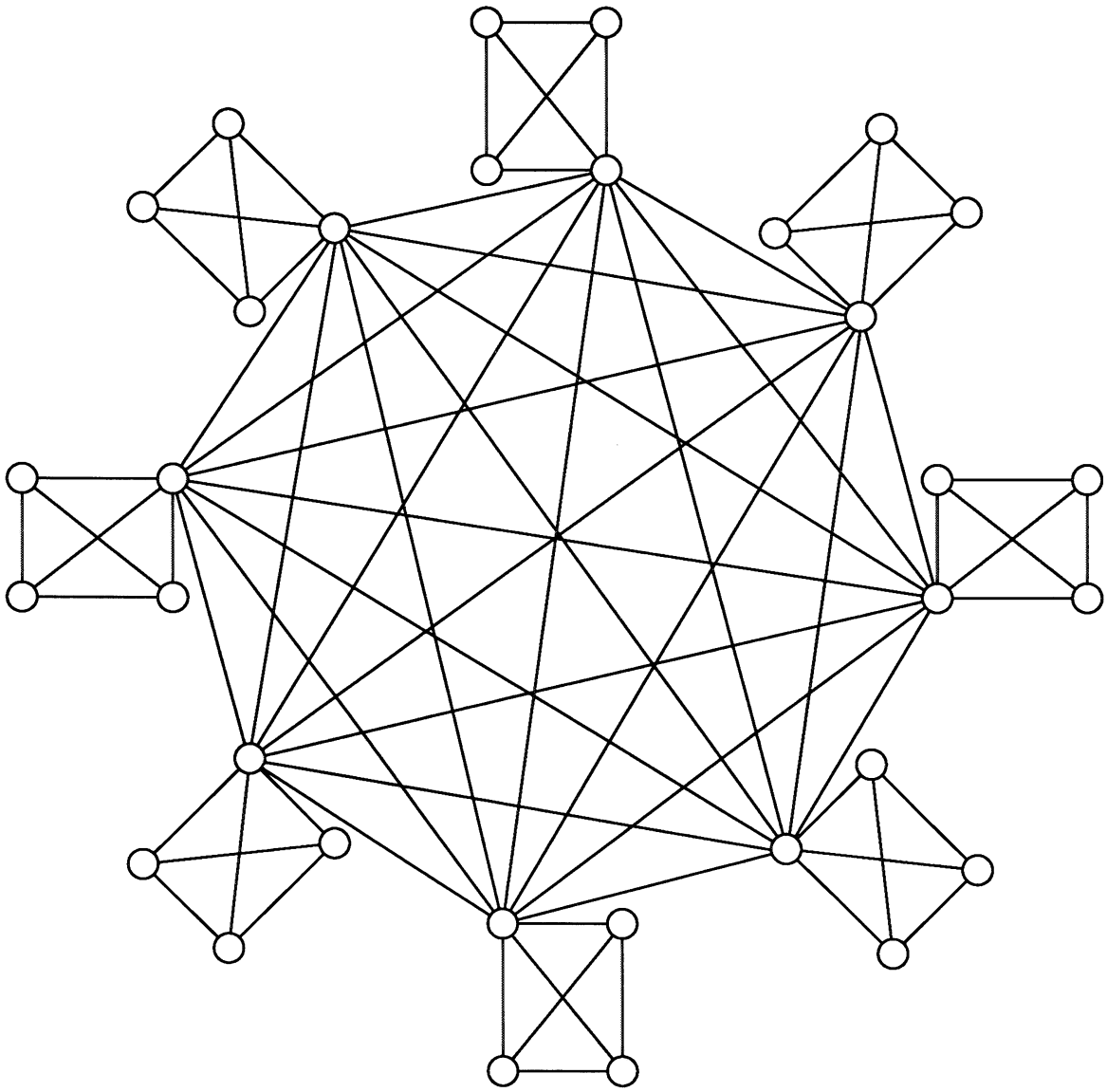


Figure 5.1: $K_4 \times K_8$ 2-hop Network (Only one of the 4 complete graphs on 8 nodes is shown)

Inter-Module Topology	Module Pin-count	#Bundles per Module	Total #Bundles
M=32			
Complete graph	992	31	496
$K_4 \times K_8$	1664	10	160
M=64			
Complete graph	504	63	2016
$K_8 \times K_8$	896	14	448
$K_4 \times K_4 \times K_4$	1152	9	288

Table 5.2: Pin-counts and number of bundles required

requires 9 bundles per module, or a total of 288 bundles. Table 5.2 shows the pin-counts and number of bundles required by the inter-module topologies considered in this chapter. All networks have a bisection width of 8192 as before. The complete-graph topologies are presented for comparison.

5.2 Intra-module Networks

5.2.1 M=32 Modules (2-hop inter-module network)

As in the previous chapter, we consider two separate networks, one for access requests and one for the responses, and each edge in the inter-module topology corresponds to 4 different channels, two in each direction. As before we fix the bisection width B at 8192. In order to achieve this bisection width, the edges of the K_4 will have to be 256 bits wide (4 64-bit wide channels), and the edges of the K_8 will be 128 bits wide (4 32 bit wide channels). This inter-module network requires 1664 pins per module as opposed to 992 for a complete graph.

Basic 2-hop network

The intra-module network of the basic 2-hop network is shown in figure 5.4. The processors are connected to the nodes on the left. The inter-module network is a $K_4 \times K_8$ and is generated by partitioning a 256-input butterfly as described in section 5.1. A 256-input butterfly leaves us with 8 inputs per module. Since there are 32 processors per module and only 8 inputs, we have to connect 4 processors to each input. This is done by building a

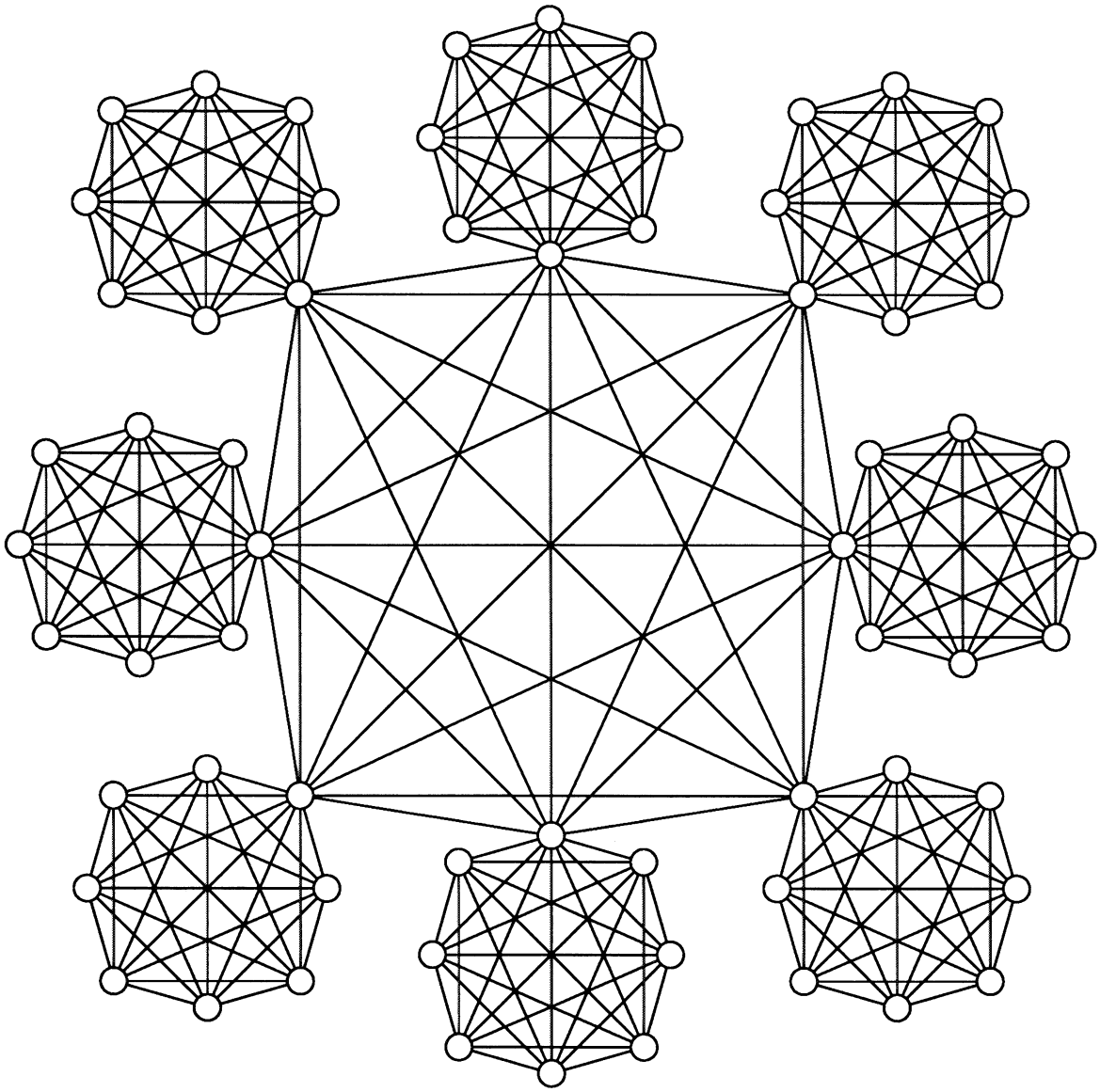


Figure 5.2: $K_8 \times K_8$ 2-hop Network (Only one of the 8 complete graphs on 8 nodes is shown)

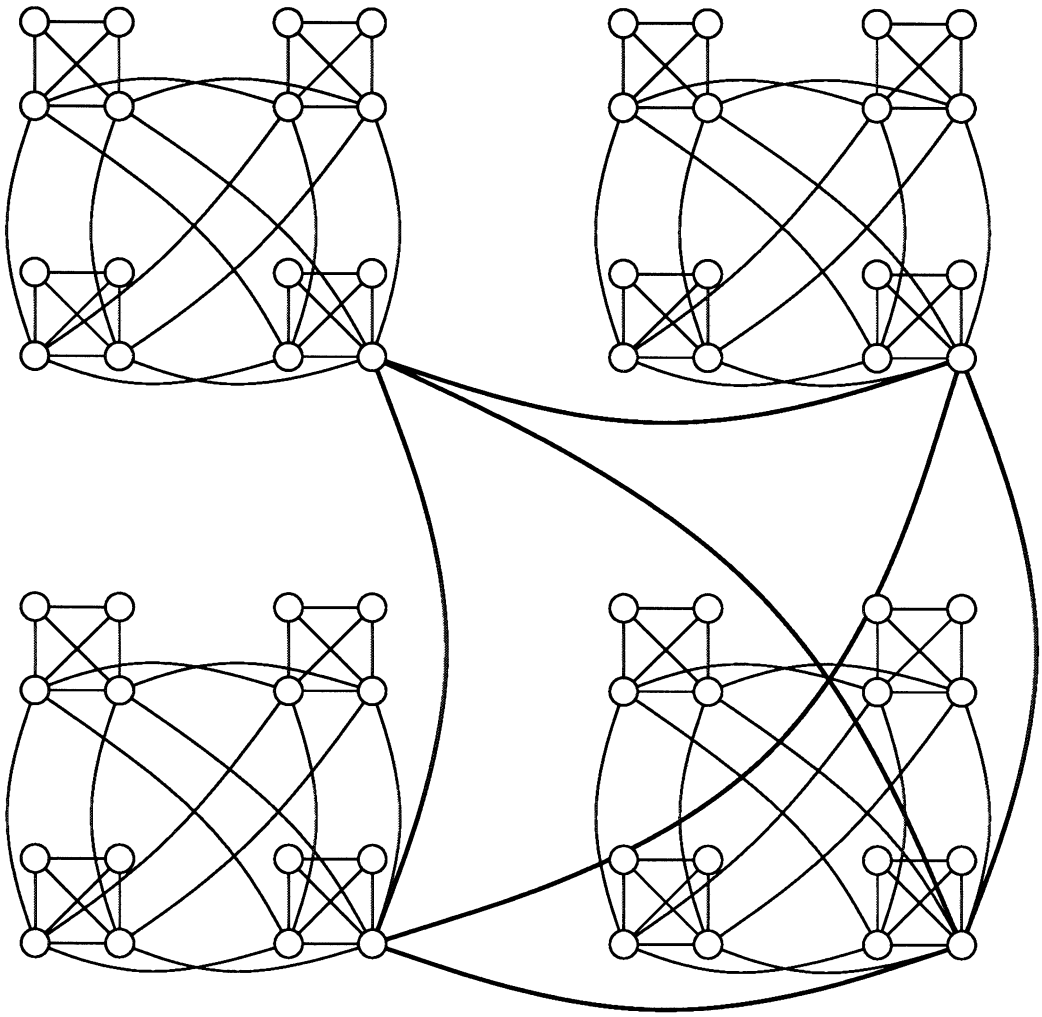


Figure 5.3: $K_4 \times K_4 \times K_4$ 3-hop Network (Only 2 of 4 complete graphs, and 1 of 16 complete graphs are shown)

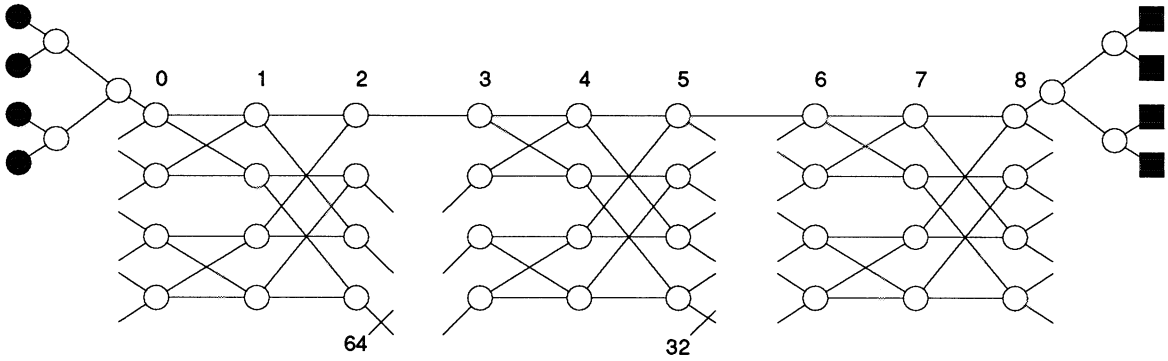


Figure 5.4: Intra-Module Network: 2-hop network, M=32 Modules

2 level tree as shown on the top node. The memories are also connected in a similar fashion. Processors and memories are paired as in the network topologies of the earlier chapter, but this is not illustrated pictorially.

The channels between the column 2 and 3 correspond to K_4. They connect between modules that differ in their least significant 2 bits. The channels between column 5 and 6 correspond to K_8 and connect between modules that differ in their most significant 3 bits. All channels are 32 bits wide except those between columns 2 and 3, which are 64 bits wide.

As per our naming convention we call this network *Butterfly(2-hop).32.64-32* since the intra-module channels are 32 bits wide and there are two kinds of inter-module channels, which are 64 and 32 bits wide.

Widening channels and dilation

Similar to the networks defined for the complete graph inter-module topology, we define the network *Butterfly(2-hop).64.64-32*. This network is the same as the previous network except that all the intra-module channels are 64 bits wide, including those that connect to the processors and memories.

We also define the network *Butterfly(2-hop).32-2dil.64-32* with a degree 2 dilation of the intra-module network.

Fewer memory units

In the basic 2-hop network we used a 2 level tree to connect 4 memories to each network input. Since the required memory throughput is only 8 bits per cycle, we can

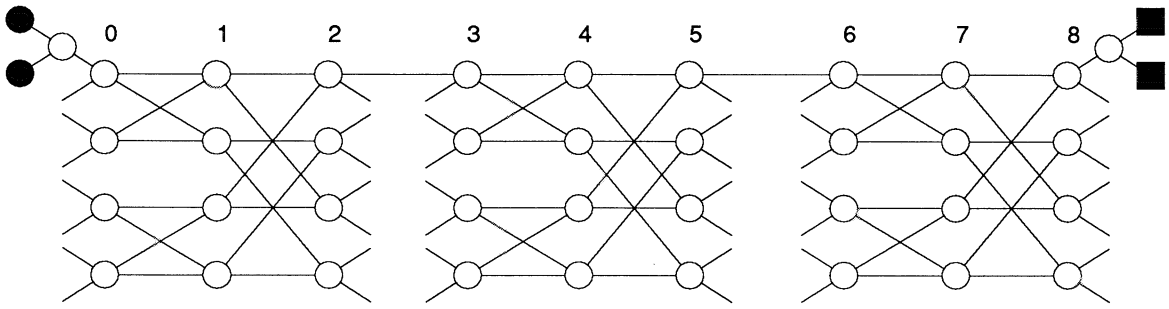


Figure 5.5: Intra-Module Network: 2-hop network, M=64 Modules

replace the 2 level tree by a single memory unit when the channels are 32 bits wide. We call this network *Butterfly(2-hop).32.64-32.M-256*. We also evaluate *Butterfly(2-hop).64.64-32.M-256* where the intra-module channels are 64 bits wide.

5.2.2 M=64 Modules (2-hop inter-module network)

To achieve a bisection width of 8192, the bundles of the 2-hop network have to be 64 wires wide. Since each bundle corresponds to 4 channels the inter-module channels will be 16 bits wide.

Basic 2-hop network

We derive the basic 2-hop network by partitioning a butterfly as described earlier. Since we need to construct a $K_8 \times K_8$ inter-module network, we start with a butterfly network with $8 * 8 * 8 = 512$ inputs. We connect two processors to each input of the butterfly network to get a 1024 processor machine. Since the inter-module channels are 16 bits wide, we get a uniform network if we make the intra-module channels also 16 bits wide. We call this network *Butterfly(2-hop).16.16* and is shown in figure 5.5.

Widening channels

We consider using wider intra-module channels of 32 bits instead of 16 and we call this network *Butterfly(2-hop).32.16*.

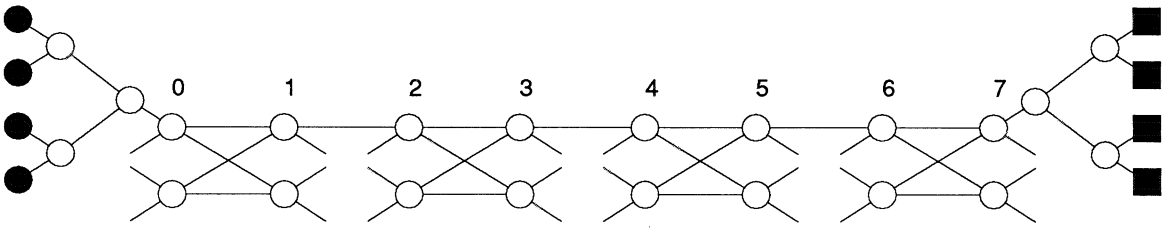


Figure 5.6: Intra-Module Network: 3-hop network, M=64 Modules

5.2.3 M=64 Modules (3-hop inter-module network)

To achieve a bisection width of 8192, the bundles of the 3-hop network have to be 128 wires wide. Since each bundle corresponds to 4 channels the inter-module channels will be 32 bits wide.

Basic 3-hop network

Since we need a 3-hop network on 64 modules, we start with a butterfly network on $4*4*4*4 = 256$ inputs and partition it to get a $K_4 \times K_4 \times K_4$. We connect 4 processors to each network input. This network is shown in figure 5.6. The inter-module channels are 32 bits wide. We get the network *Butterfly(3-hop).32.32* when the intra-module channels are also 32 bits wide.

Widening channels

We consider using wider intra-module channels of 64 bits instead of 32 and we call this network *Butterfly(3-hop).64.16*.

5.3 Simulation Results

The results corresponding to the open-network work-load model are presented in figures 5.7, 5.8 and 5.9. Figure 5.7 compares the different networks which use a 2-hop inter-module network, with M=32. The data corresponding to *Butterfly.8.8* and *Butterfly.32.8*, from the previous chapter are included for comparison. In the latter two networks the inter-module network is a complete graph. Although all the networks have the same bisection width, the latter two networks use fewer pins per module, and more bundles. Therefore, their costs are not directly comparable. Figures 5.8 and 5.9 compare networks

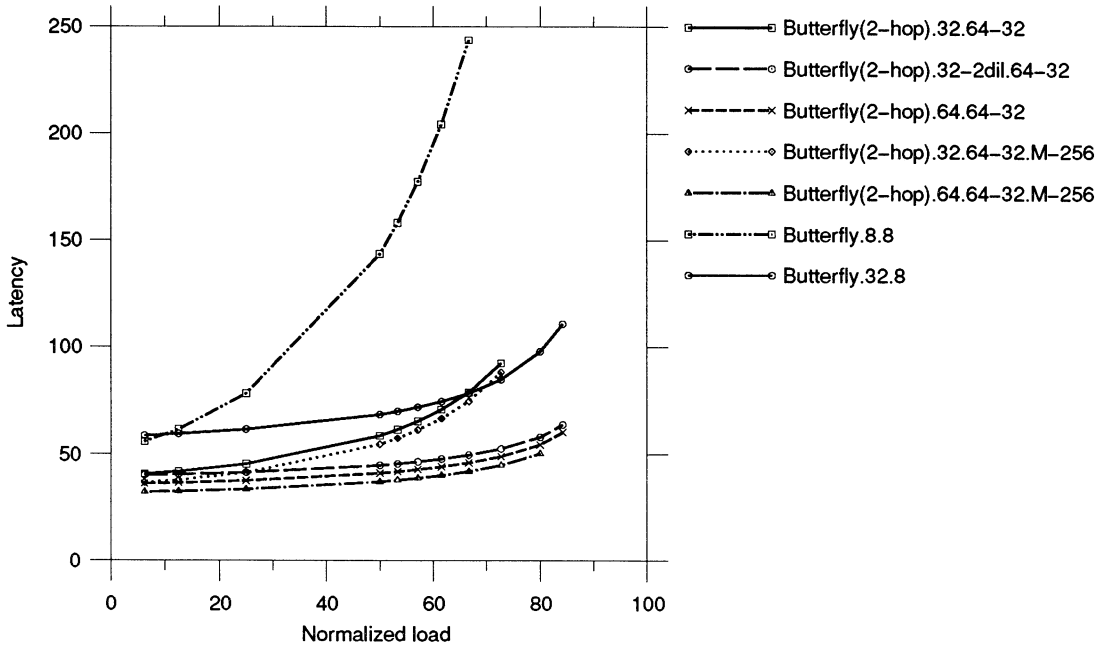


Figure 5.7: Open-Network Model; M=32 Modules, 2-hop intermodule network

where M=64. The data corresponding to *Butterfly.8.2* and *Butterfly.32.2* (inter-module network = complete graph) are included for comparison.

Figures 5.10 and 5.11 show the results for the multithreading work-load when the mean access interval is 16 cycles, and figures 5.12 and 5.13 correspond to a mean access interval of 32 cycles.

5.3.1 Discussion of results

The results indicate the same general trends as seen in the previous chapter. We can improve performance by making the intra-module networks richer either using dilation or wider channels. We do not see much difference between dilation and widening channels since the channels are already quite wide to begin with. Increasing the channel widths further does not make much of an impact with respect memory service rate. Dilation and widening channels have similar effects with respect to reducing contention. As before, we see that reducing the number of memory units does not degrade performance. Actually it improves the performance by a small amount since we reduce the number of switching stages that the messages have to pass through, by eliminating the fan-out trees at the memory end.

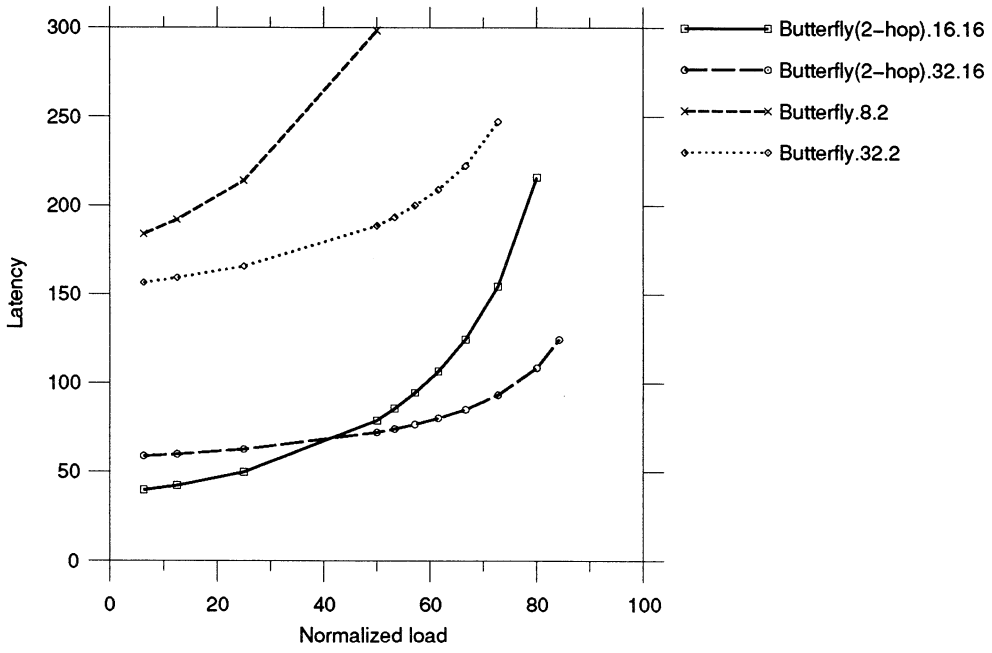


Figure 5.8: Open-Network Model; M=64 Modules, 2-hop intermodule network

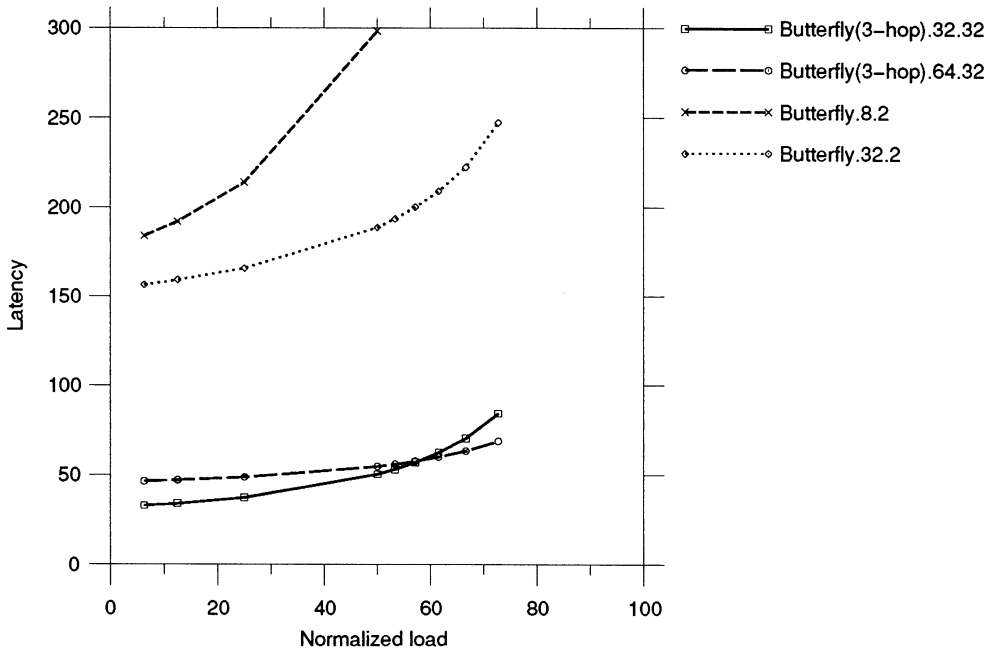


Figure 5.9: Open-Network Model; M=64 Modules, 3-hop intermodule network

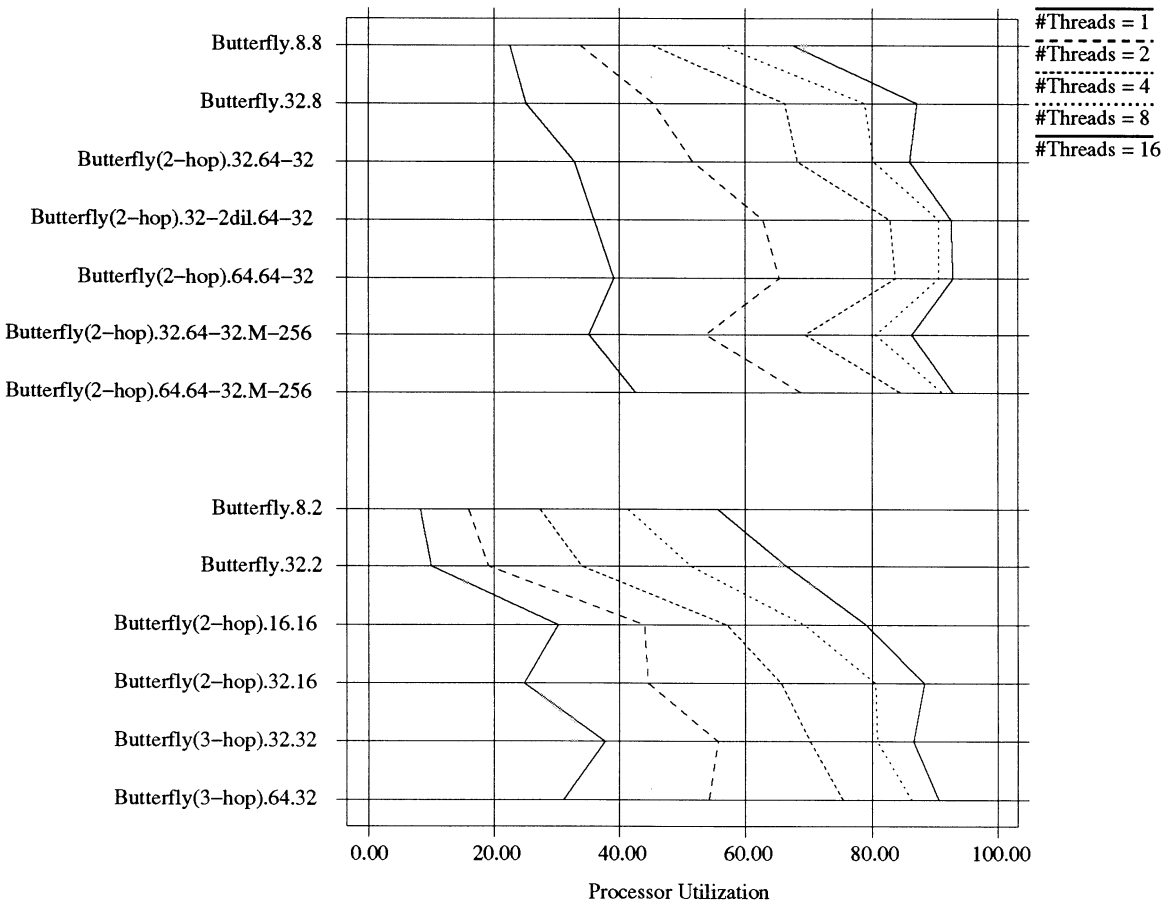


Figure 5.10: Periodic Access Model, access interval = 16 cycles

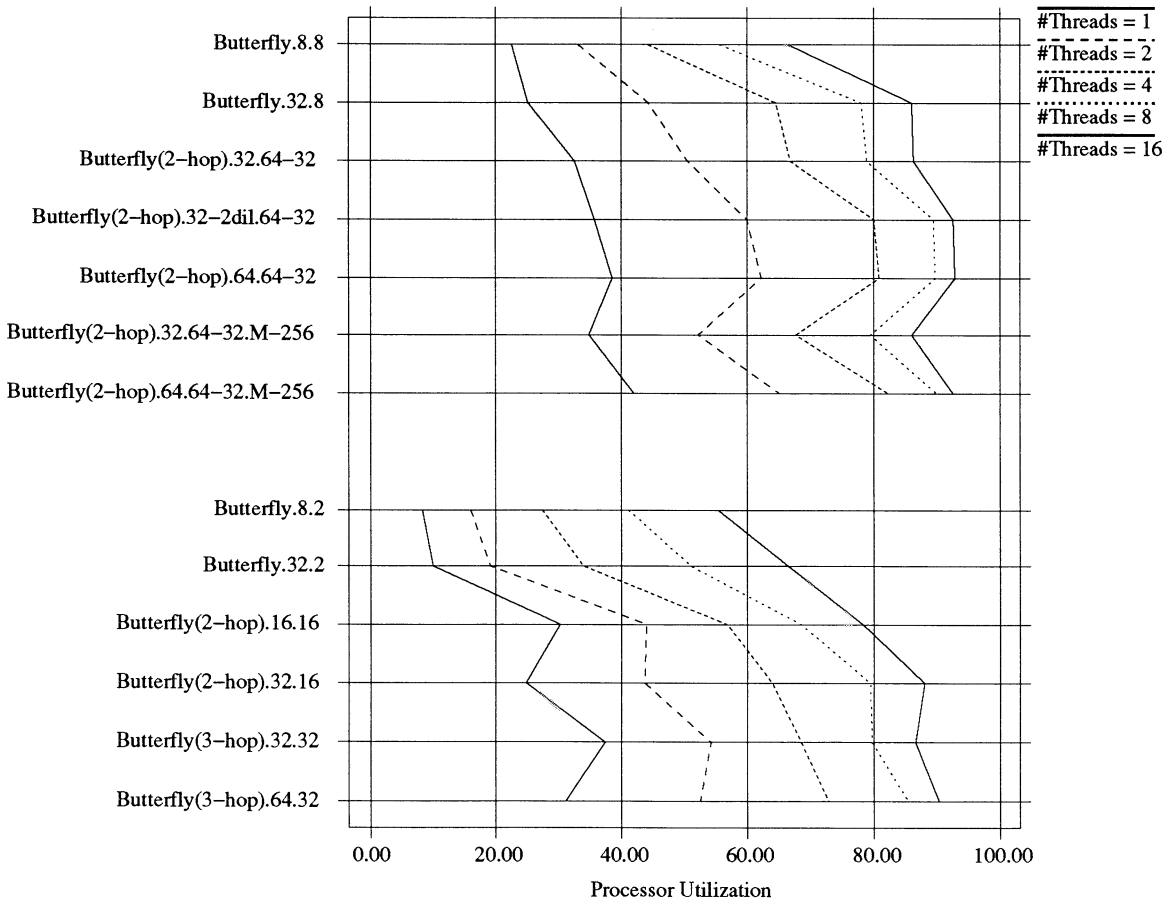


Figure 5.11: Geometric Access Model, access interval = 16 cycles

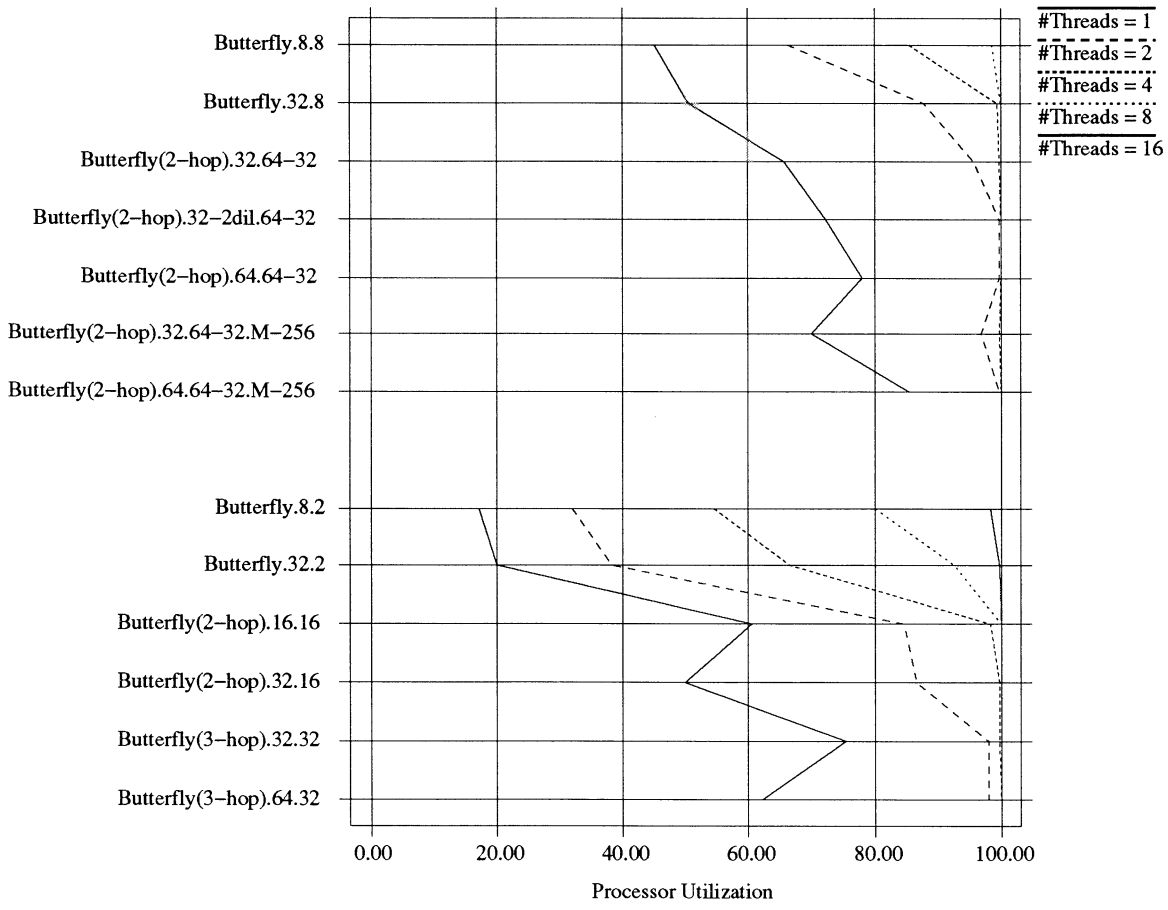


Figure 5.12: Periodic Access Model, access interval = 32 cycles

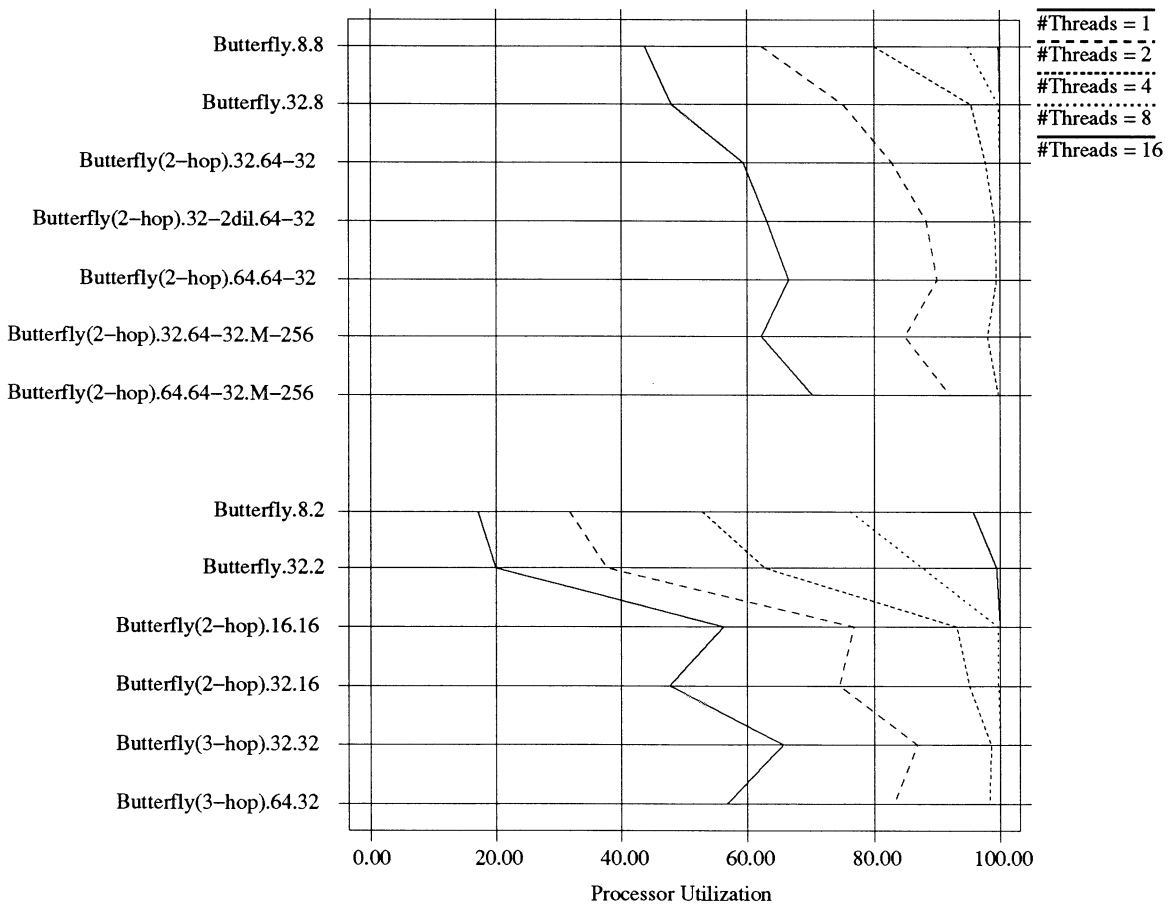


Figure 5.13: Geometric Access Model, access interval = 32 cycles

While comparing 1-hop, 2-hop, and 3-hop networks, we must keep in mind that their cost vectors are different, although they have the same peak throughput. However, if we ignore the differences in the cost vector, and just compare networks with the same peak throughput, we see that the 1-hop networks are worse than 2-hop and 3-hop networks. The reason for this difference in performance is the wider channels in the latter networks. When the channels are wider, it is easier to achieve higher channel utilizations.

5.4 Summary

In this chapter, we considered a cost vector that was slightly more complex than that of the previous chapter. We included the number of bundles in the list of cost parameters. When we expanded our list of cost parameters to include the number of bundles, we saw that our set of interesting top-level was no longer limited to a complete graph network, as in the previous model, where we ignored the number of bundles.

In this chapter, we first compared the performance of networks of fixed cost, where we fixed the cost by fixing all the three cost parameters; namely the number of modules, the pin-count per module and the number of bundles per module. We saw that it was difficult to find the network with maximum performance for any arbitrary set of parameters. Instead, we were able to find networks that had performance close to the maximum possible performance. We showed that the n-hop networks and de Bruijn networks were close to optimal.

When we examined the network design problem from the point of view of fixing performance and comparing costs, we discovered a trade-off between pin-count and number of bundles. In going from a complete graph to a 2-hop network we reduced the number of bundles per module from $O(M)$ to $O(\sqrt{M})$, but doubled the pin-count. In general, reducing the number of bundles implied that the pin-count had to increase to maintain the same level of performance. We also showed that the trade-offs offered by n-hop and de Bruijn networks were close to optimal, in the sense it was not possible to reduce the pin-count and number of bundles by more than small constant factors and still achieve the same performance.

A question that we might ask is, "Which of the n-hop or de Bruijn networks should we use?". In order answer this question, we need to compare the costs of the different n-hop and de Bruijn networks. To compare their costs, we need to know the relative weights of the cost dimensions, but we chose not to specify these weights because they were technology

dependent. Therefore, the answer to the above question is technology dependent, and we can answer it by first determining the relative weights of the cost dimensions based on the technology of interest. Then, we would use these weights to compute the actual monetary costs of the different networks and select the network that has the lowest cost.

Similar to the previous chapter, we evaluated several alternative network organizations for the lower level of the hierarchy, and found that the general results were similar.

The results in this chapter also reinforce a result we obtained in the previous chapter. In the previous chapter, we did not include the number of bundles in the cost vector, and under the previous model we saw that the complete graph was the best inter-module network. Comparing networks evaluated under model 2 on the basis of equal pin-counts illustrates this result. If we permit the complete graph network to use the same number of pins as a 2-hop network, the bisection width of the complete graph becomes twice that of the 2-hop network. Although we did not explicitly simulate a complete graph with twice the bisection width, we can get a good approximation of the performance of such a network by looking at the performance of the complete graph network under 50% load, because doubling the bisection width also doubles the peak throughput. When we compare the performance of the complete graph with a mean access of 32 cycles with the performance of the 2-hop network with a mean access interval of 16 cycles, we see that the complete graph network achieves significantly greater processor utilizations. In fact, most of the networks in which the inter-module network is a complete graph, achieve almost 100% processor utilization. Therefore, we can clearly see that comparing networks based on equal pin-counts favors the complete graph.

Chapter 6

Model 3: Three level hierarchy

This model is motivated by the fact that the cost of the different wires that go between modules may not all be identical. Wires that connect modules which are physically close to each other will be short and therefore, likely to be less expensive than wires that connect long distances. For instance, in a packaging hierarchy where modules correspond to card-cages, wires that go between card-cages within a cabinet are likely to be cheaper than wires that go between cabinets. In this model, we use an additional level of packaging to characterize this difference in costs. In general, we can increase the number of levels in the packaging model, to characterize the technology to a greater level of detail. In this chapter, we assume that the machine is packaged using a 3 level hierarchy. At the top level the machine consists of level-2 modules which we call *clusters*. Each level-2 module consists of level-1 modules which we simply call *modules* as before. The connections between modules in the same cluster correspond to the short connections and the connections that go between modules in different clusters correspond to the long distance connections.

As before we have intra-module and inter-module connections, but now, the inter-module connections are divided into two categories: intra-cluster connections and inter-cluster connections. Each module has two kinds of pin-count: intra-cluster pin-count and inter-cluster pin-count. The intra-cluster pins are used to make connections between modules within the same cluster and the inter-cluster pins are used to make connections between modules in distinct clusters.

The cost vector under this model is more complicated than the ones used in the previous chapters. In this model, the cost vector has five dimensions: the number of clusters, number of modules per cluster, the level-2 pin-count, number of level-2 bundles, and the

level-1 pin-count. The level 2 pin-count is the number of long-distance wires connecting to each cluster or the total number of inter-cluster pins used by the modules within a cluster. As before, we shall assume that there is a limit on the maximum number of processors that we can put in a module. We shall also assume that a similar limit exists on the number of modules in a cluster.

Since the cost vector has many more dimensions than the earlier models, an approach we took in the earlier chapters, namely one of fixing the cost by fixing all the parameters, is overly restrictive. Instead, in this chapter, we shall fix the performance and explore the trade-offs between the dimensions of the cost vector. Similar to the previous models, we shall fix some of the dimensions of the cost vector and examine the trade-offs between the other dimensions. In this chapter, we fix the number of clusters, the number of modules per cluster and the level-2 pin-count. In fixing the first two parameters, we minimize both the number of clusters and total number of modules based on the limits on the maximum number of modules per cluster and maximum number of processors per module respectively. Since the previous models have already analyzed the relationship between the top-level pin-count and performance, in this chapter we also fix the top-level (or level-2) pin-count. We are now left with two dimensions of the cost vector: the number of level-2 bundles and the level-1 pin-count. We explore the trade-off between these two parameters by considering different values for the level-1 pin-count. We shall see that we if we have more intra-cluster pins, we can reduce the number of bundles, while maintaining the same level of performance. As before, we use fix the performance of the network by fixing its bisection width.

6.1 Network choices

Let the total number of modules be M, and the number of modules per cluster be C, for a total of M/C clusters. We have already seen that the best top-level network when the number of top-level pins is fixed, is a complete graph. Since there are (M/C) clusters, fixing the bisection width at B, implies that each edge in the complete graph topology must correspond to $\frac{4B}{(M/C)^2}$ wires. Since each cluster is connected to $(M/C)-1$ other clusters, the number of level-2 pins required per cluster is $\frac{4B(M/C-1)}{(M/C)^2} = \frac{4B(M-C)C}{M^2}$. In this chapter we consider the case M=32, and C=4 and as before, we fix B=8192. For this particular case, each pair of clusters is connected using 512 wires, and the number of level-2 pins required

per cluster is 3584.

However, we still consider the connections to be between modules, since clusters are conceptual units of the hierarchy that mainly help us distinguish between short and long-distance connections. We observe that we can have many different inter-module topologies that lead to a complete graph between clusters.

For example, if we use a complete graph between modules, the interconnection between the clusters is also a complete graph. Each inter-cluster edge corresponds to $C^2 = 16$ inter-module bundles, since each module within a cluster has a connection to each module in the other cluster. Since the inter-cluster edges correspond to $\frac{4B}{(M/C)^2} = 512$ wires, each inter-module bundle consists of $\frac{4B}{M^2} = 32$ wires. There are a total of $\frac{M(M-1)}{2} = 496$ inter-module bundles, of which $(M/C) * \frac{C(C-1)}{2} = 48$ are short-distance bundles and $\frac{1}{2}(\frac{M}{C})(\frac{M}{C} - 1) * C^2 = 448$ are long-distance bundles. Each module requires $\frac{4B}{M^2}(C-1) = 96$ intra-cluster pins and $\frac{4B}{M^2}(M-C) = 896$ inter-cluster pins. Figure 6.1 shows this network.

Another inter-module topology that leads to a complete graph between clusters is the 2-hop network $K_C \times K_{M/C}$. In this network, messages make 2 hops, one hop over the intra-cluster connections and the other hop over the inter-cluster connections. Between any pair of modules, there are $C = 4$ inter-module bundles, because each module is only connected to its corresponding module in the other cluster. Therefore, the long-distance bundles consist of $\frac{1}{C}\frac{4B}{(M/C)^2} = 128$ wires. This network has a total of $\frac{1}{2}(\frac{M}{C}) * (\frac{M}{C} - 1) * C = 112$ long-distance bundles. The number of short-distance bundles is the same as before: $(M/C) * \frac{C(C-1)}{2} = 48$, but each bundle is wider, in order to support the larger bandwidth that is required by the 2-hop network. Each of these short-distance bundles is 256 wires wide and the number of intra-cluster pins per module is be 768. This topology trades-off a larger number of intra-cluster pins for a reduction in the number of bundles. Figure 6.2 shows this network.

The complete graph and the 2-hop network topologies discussed above are the fundamentally the same as the inter-module topologies that we used in Chapters 4 and 5. The only difference between them is the way the pins and bundles are accounted for. For the complete graph inter-module network, all the networks with $M = 32$ considered in Chapter 4, are valid candidates under this packaging model. We only consider the networks *Butterfly.8.8* and *Butterfly.32.8* for the purposes of comparison in this chapter. Similarly all the networks for M=32, considered in Chapter 5, are valid candidates, and we consider the networks *Butterfly(2-hop).32.64-32* and *Butterfly(2-hop).64.64-32* in this chapter.

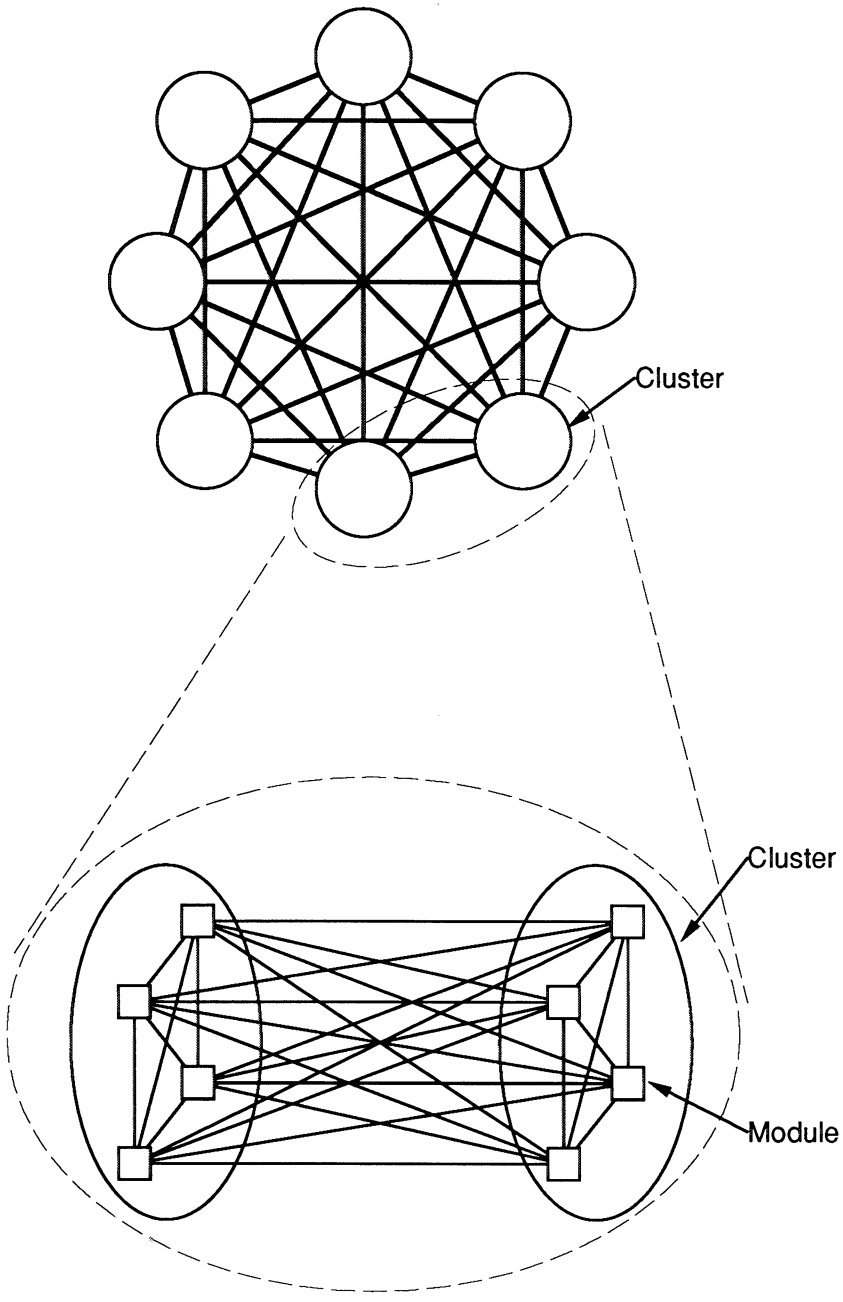


Figure 6.1: Complete graph between modules

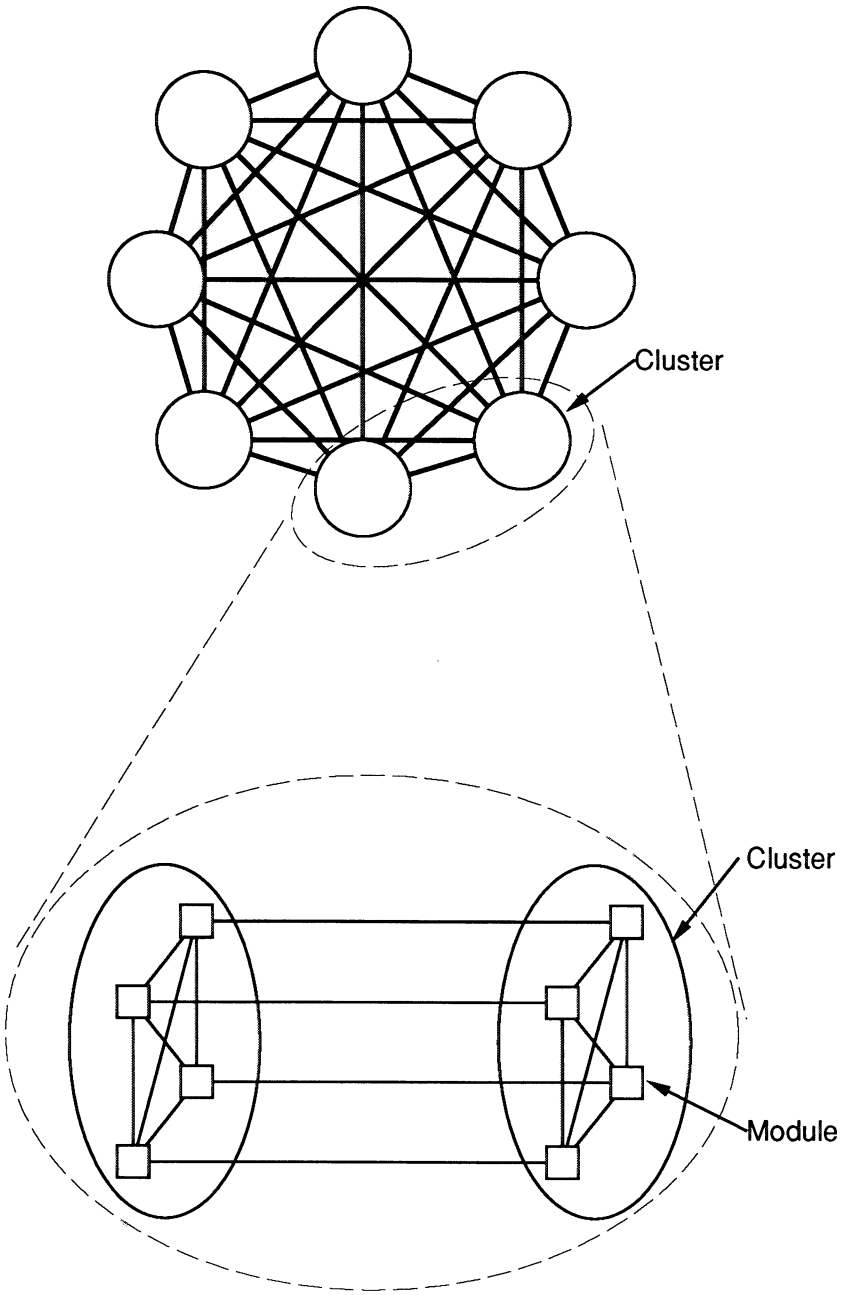


Figure 6.2: 2-hop network $K_4 \times K_8$ between modules

The 2-hop network described above requires 112 long-distance bundles and 768 intra-cluster pins per module. If we have an even larger number of intra-cluster pins, we can reduce this number of bundles further. In going from the complete graph to the 2-hop network we reduced the number of long-distance bundles by a factor of C. The number of bundles between any pair of clusters was C^2 in the case of the complete graph, and C in the case of the 2-hop network. We can build a network which we call a *Clustered network*, in which we reduce the number of inter-cluster bundles by another factor of C, down to exactly one bundle between any pair of clusters. Figure 6.3 shows this network. The total number of inter-cluster bundles required by this network is $\frac{1}{2}(\frac{M}{C}) * (\frac{M}{C} - 1) = 28$ and the number of intra-cluster bundles remains $(M/C) * \frac{C(C-1)}{2} = 48$. In this network, all the wires that go between any pair of clusters are connected between some pair of modules, one in each cluster. Messages in this network make a maximum of 3 hops to get from one module to another. First the message goes from the source module to the appropriate module within its cluster. The second hop is made between clusters, taking the message to the correct destination cluster. The third hop then moves the message within the destination cluster to the correct destination module. Of the 3 hops, two hops are within clusters and one hop is between clusters. Therefore, the short-distance bundles have to be twice as wide as the bundles in the 2-hop network, namely 512 wires each. Each module requires 1536 intra-cluster pins. As in the two earlier cases, the total number of inter-cluster pins per cluster is 3584, but since there are 7 inter-cluster channels and 4 modules per cluster, three of the modules have 2 channels each, and one module only has a single channel. Since each channel is 512 bits wide, three of the 4 modules require 1024 inter-cluster pins while one module requires 512 inter-cluster pins.

We can use a strategy similar to the one used in Chapter 5 to reduce the number of intra-cluster bundles, by using a multi-hop network that requires more intra-cluster pins. But for the case that we consider in this chapter, $C = 4$ modules per cluster, it does not make much sense to make the intra-cluster network sparser since the complete graph interconnect is only on 4 modules. For larger cluster sizes, we could consider using a 2-hop or a 3-hop network between the modules within a cluster.

We can generate a clustered network by suitably partitioning a butterfly network as shown in figure 6.4. The figure shows the connections within a single cluster. Based on the bisection width and the total number of pins used for inter-cluster, and intra-cluster connections, the inter-cluster channels are 128 bits wide and the intra-cluster channels are

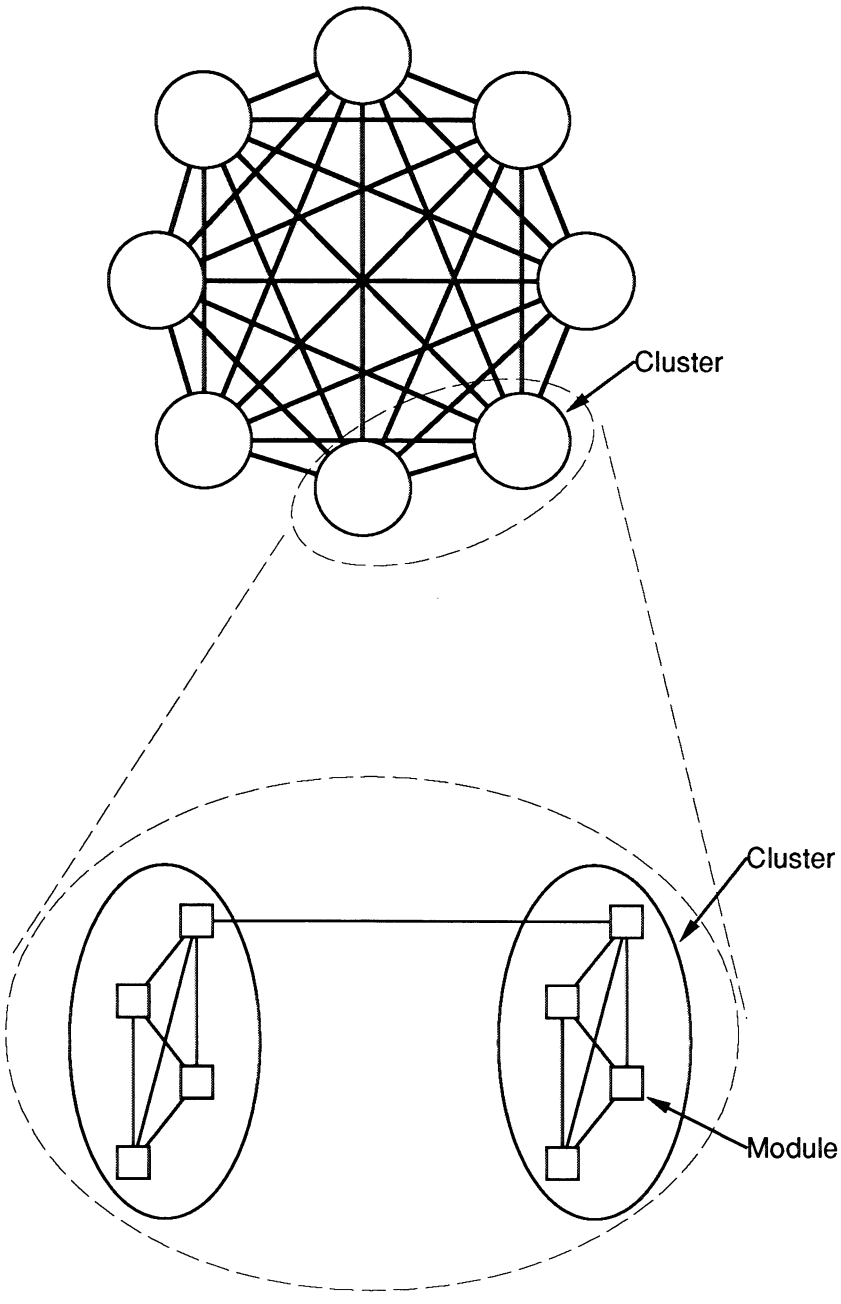


Figure 6.3: Clustered Network

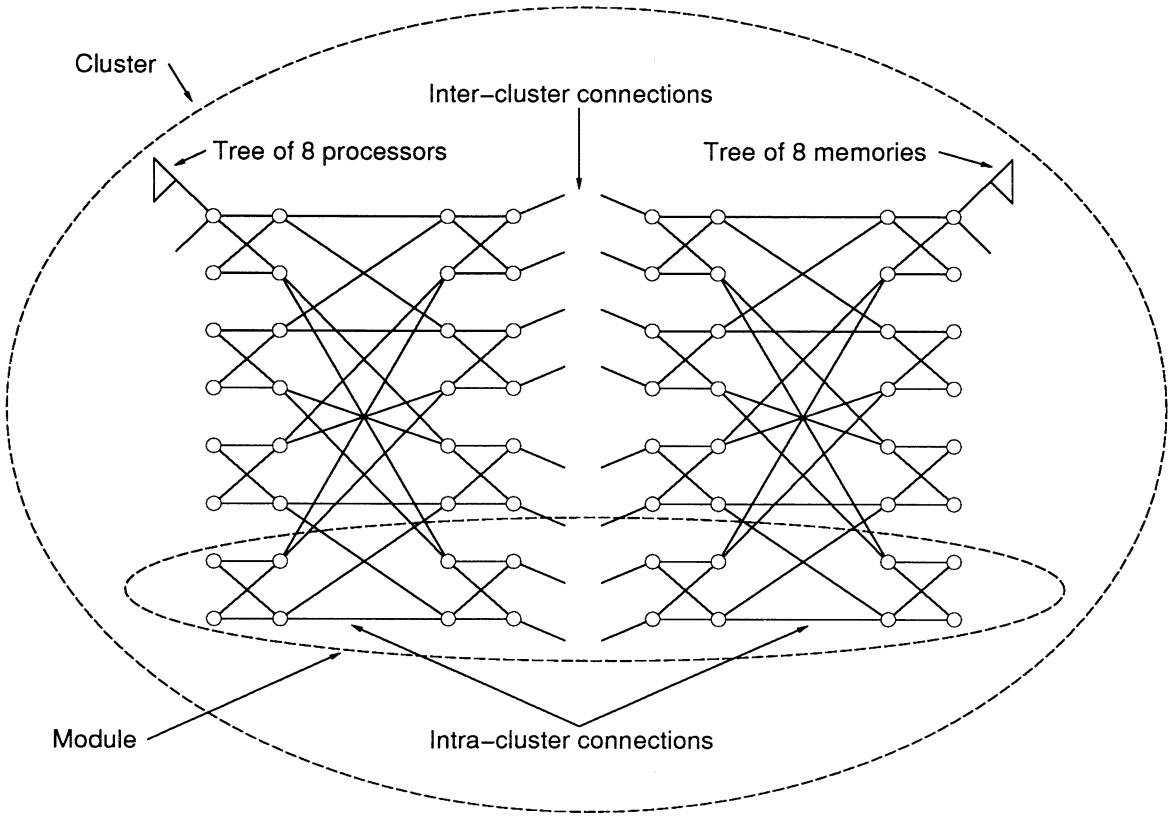


Figure 6.4: Partitioning a butterfly to yield a Clustered network

Network	Inter-cluster Pins/module	Inter-cluster Bundles	Intra-cluster Pins/module	Intra-cluster Bundles
Butterfly.8.8	896	448	96	48
Butterfly.32.8	896	448	96	48
Butterfly(2-hop).32.64-32	896	112	768	48
Butterfly(2-hop).64.64-32	896	112	768	48
Butterfly(Clus).64.64.128	512, 1024	28	1536	48
Butterfly(Clus).128.128.128	512, 1024	28	3072	48

Table 6.1: Pin and bundle requirements

64 bits wide. To balance the bandwidths, intra-module channels of 64 bits are sufficient. As per our convention, we call this network, with the above channel width specifications, *Butterfly(Clus).64.64.128*. If we have more intra-cluster pins than the number required by *Butterfly(Clus).64.64.128*, we can make the intra-cluster connections wider. We also consider the network *Butterfly(Clus).128.128.128* in which all the channels are 128 bits wide. This network uses twice the number of intra-cluster pins per module as *Butterfly(Clus).64.64.128*. Table 6.1 summarizes the characteristics of the different networks.

6.2 Simulation Results

Figure 6.5 shows the results of the open-network workload simulations. Figures 6.6, 6.7, 6.8 and 6.9 show the results for the multithreading workload model.

6.2.1 Discussion of results

Again, the results this model follow the same general trends as seen in the previous chapters, i.e., network performance can be improved by improving the local communication bandwidth (even when the top level bandwidth is the same), and it is better to organize a fixed amount of communication bandwidth as fewer wider channels. This is evident from the improvement in performance resulting from using denser networks at the intra-cluster and intra-module levels. All the networks shown in the graphs have the same number long-distance pins per cluster, but they use different numbers of bundles and intra-cluster pins. The 2-hop and Clustered networks have fewer bundles that are wider, and they achieve better network performance than the networks based on a complete graph between modules.

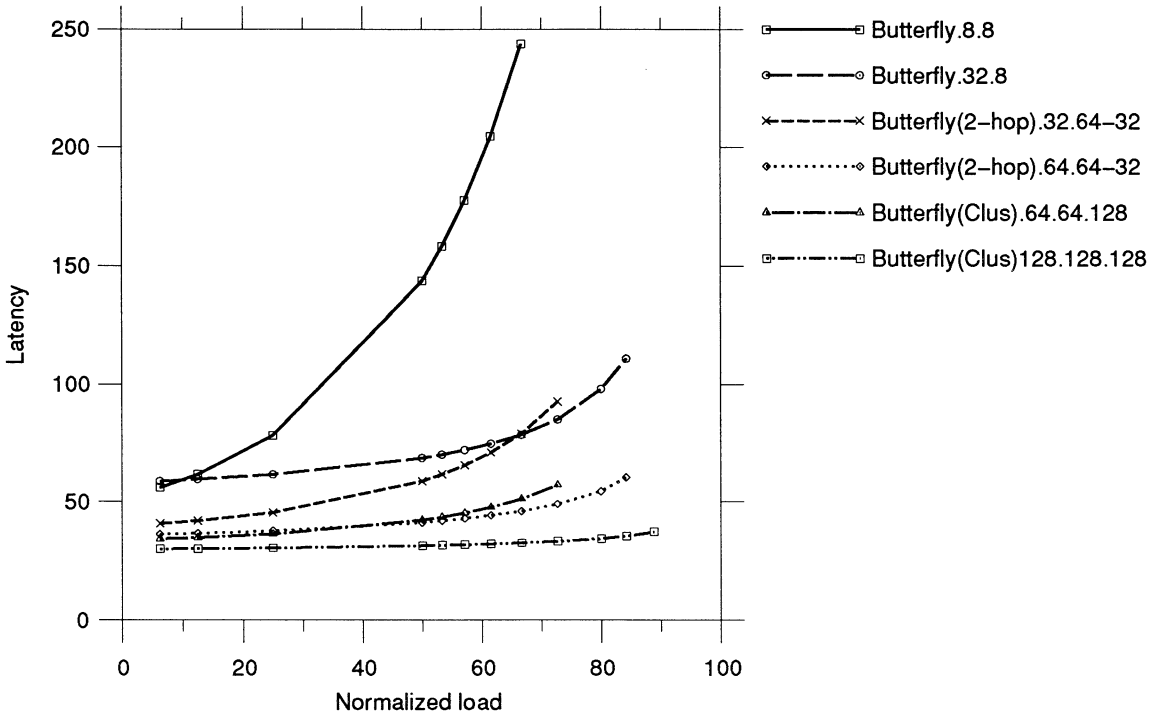


Figure 6.5: Open-Network Model; M=32 modules, C=4 modules per cluster

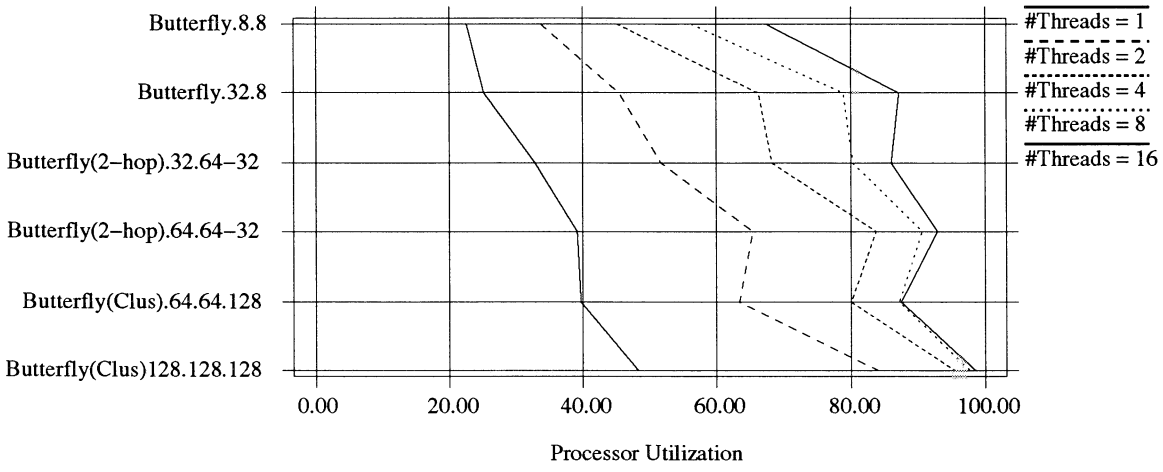


Figure 6.6: Periodic Access Model, access interval = 16 cycles

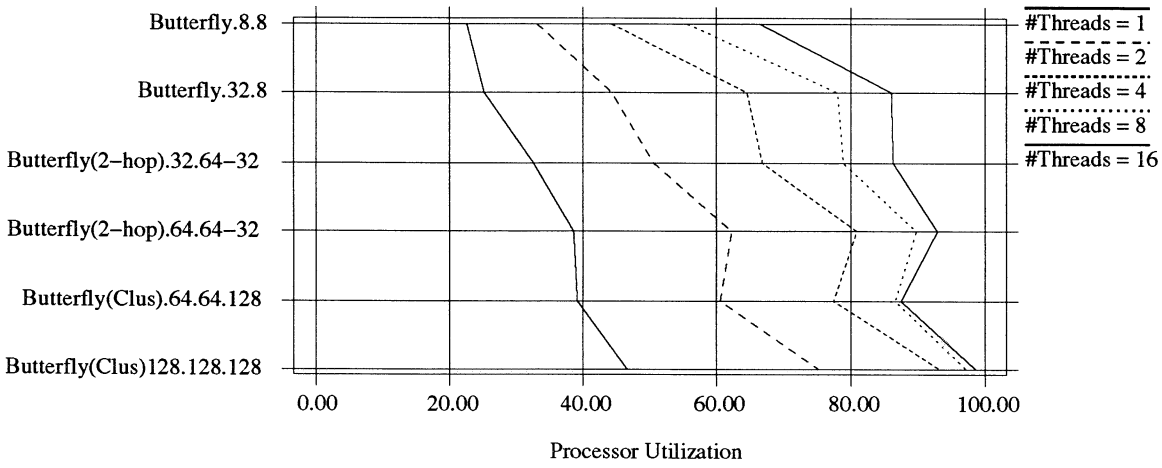


Figure 6.7: Geometric Access Model, access interval = 16 cycles

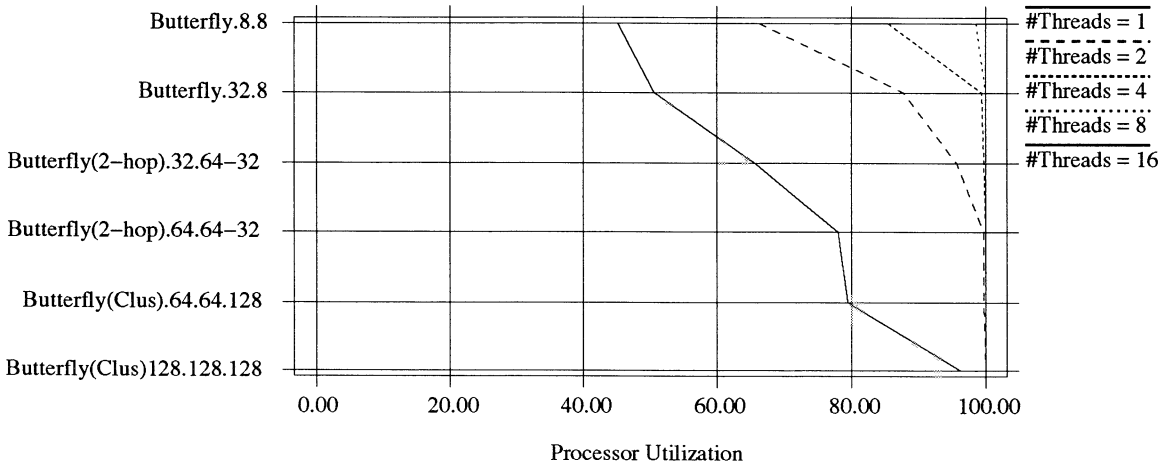


Figure 6.8: Periodic Access Model, access interval = 32 cycles

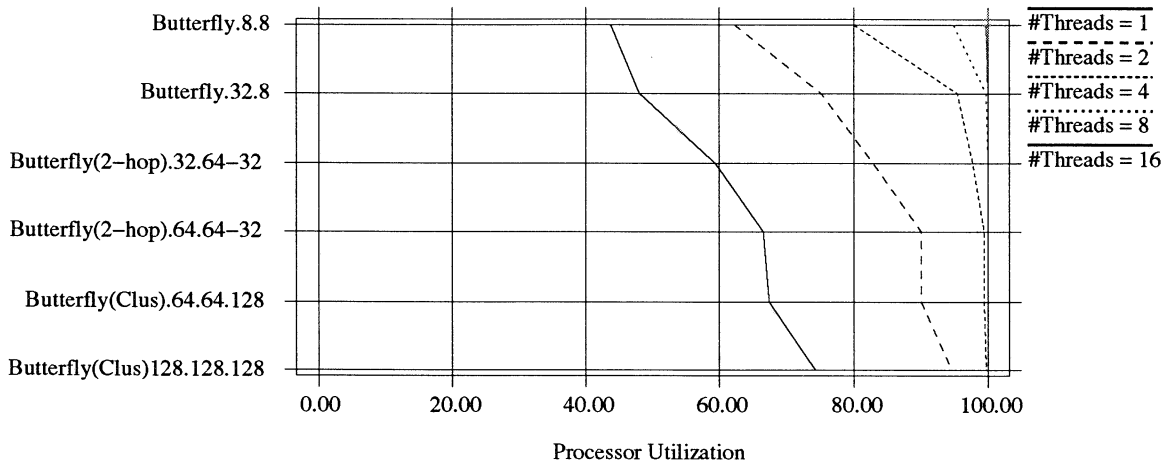


Figure 6.9: Geometric Access Model, access interval = 32 cycles

Another interesting factor seen from the open-network results, is that *Butterfly(2-hop).64.64.32* and *Butterfly(Clus).128.128.128* have almost flat latency curves even up to the point where the load is as high as 75 to 80% of the peak. This indicates that the high bandwidth available at the lower levels of the packaging hierarchy eliminates almost all the network contention for the channels at the lower levels of the hierarchy.

6.3 Summary

In this chapter, we considered a packaging model that was more complex than the ones considered in the two previous chapters. The model was based on a three level hierarchy and was motivated by the need to model short and long-distance connections. Under this model, we compared the cost of several networks with equal peak performance. Since the cost vector under this model was more complex than the two previous models, we only examined the trade-off between two of the dimensions of the cost vector, namely the short-distance pin-count and the number of long-distance bundles. We saw that we could reduce the number of long-distance bundles by using a larger number of short-distance wires. We illustrated this using three different interconnection networks that represented three separate positions in this trade-off. When we looked at the performance on a finer scale, we saw that as we reduce the number of long-distance bundles, we got better performance because reducing the number of long-distance connections increased the widths of the long-distance channels, which, in turn, improved their utilization.

Chapter 7

Scalability

Computer manufacturers usually want to build machines over a wide range in price and performance to cater to the needs of different customers with varying budgets and computing requirements. Such a design goal was proposed as early as the 1960s for the IBM System/360 [SBN82]. The IBM System/360 achieved a performance range of fifty to one by employing a wide choice of components, engineering techniques and technologies. The designers of the 360 family invested considerable effort in the design of separate machines for each point on the price-performance curve.

A similar design goal of providing a wide range in price and performance becomes even more important in the case of parallel machines, since it seems natural to buy more processors when a more powerful machine is needed. Further, in the case of parallel machines achieving this design goal seems easier (than in the case of System/360), since performance can be increased by increasing the number of processors. Therefore, parallel machine manufacturers wish to design machines that can be built for a wide range of sizes to enable customers to purchase a machine appropriate to their needs. Besides, customers also frequently discover that their computing needs outgrow the capacity of the machine they currently have [Bel92], and often desire to upgrade their machine by increasing the number of processors. Therefore, manufacturers often want the ability to incrementally upgrade parallel machines to larger sizes.

Since designing parallel machines usually requires substantial investment in time, effort, and money, it is desirable to reduce the complexity associated with the design process. While designing machines over a range of sizes, manufacturers prefer that many aspects of the design carry over from one machine size to another. It is often preferable to build

the entire range of machines from some small set of building blocks. The effort invested in designing the building blocks, then is amortized over the whole family of machines. In contrast, designing each machine of the family separately, in a manner similar to the System/360 would imply substantially higher design costs and time.

However, it is conceivable that when each member of the family is designed individually, each member may provide better performance because we may be able to perform many design optimizations that would not be possible in the building block approach. Since the building blocks have to be usable all the machines of the family, we may be faced with more stringent constraints in the design of the building blocks themselves. It is also conceivable that the additional constraints on the building blocks may lead to an increased usage of packaging resources such as pins, wires etc. In other words, cutting design costs may result in an increase in hardware (or packaging) costs or a reduction in performance. In this chapter we explore this trade-off in greater detail.

The ability to build parallel machines over a wide range of sizes is usually referred to as *scalability*. However *scalability* is currently an intuitive notion that has no formal definition [Hil90]. First, in Section 7.1 we formalize the general intuition regarding *scalability* and provide a definition of the term. Our definition permits us to distinguish machine families as being scalable or unscalable. We then classify scalable machine families into a number of broad categories based on the relative design costs associated with them(Section 7.2). This classification is based on the size of the building blocks that are common across the members of the family. We illustrate this classification by extending a few of our earlier 1024 processor designs to families corresponding to each of the scalability categories (Section 7.3). A comparison across the families illustrates the trade-off mentioned earlier(Section 7.4).

7.1 What is scalability?

The main reason for building a family of machines over a range of sizes is to provide a range in cost and performance. Larger machines should provide correspondingly higher performance for higher cost. Therefore we define scalability as follows:

Definition 7.1 *A network family is (P, C)-scalable in the range $[N_0, N_1]$, if for each machine of size $N_0 \leq N \leq N_1$, the performance of the machine exceeds $P(N)$ and the hardware*

cost is less than $C(N)$.

Note that the definition is qualified by the functions $P(N)$, $C(N)$, and the constants N_0 and N_1. The functions $P(N)$ and $C(N)$ are the desired performance and cost respectively. Ideally, network performance should increase linearly in network size[1]. Therefore, we define $P(N) = constant{*}N$. We would also like the cost to be linear in the size of the network, however a linear increase in performance usually implies a super-linear increase in cost.

The reason for the super-linear increase in cost is as follows. As the network size increases, the average distance traveled by a message usually increases. In order to maintain a linear increase in peak throughput, as the network size increases, the total number of wires in the network should increase in proportion to the product of the network size and the average distance traveled by messages. This implies that, in order to provide a linear increase in peak throughput, the number of wires in the network must increase in a super-linear fashion. Therefore, unless $C(N)$ is a super-linear function, we will end up classifying all networks as unscalable. Typically we might choose $C(N) = constant{*}N \log N$.

Given $P(N)$ and $C(N)$, each network family will have a (possibly null) range of scalability — $[N_0, N_1]$. Figure 7.1 shows the ranges for a few common network topologies[2]. The networks have different ranges of scalability depending on where their performance and cost curves cross the bounding functions $P(N)$ and $C(N)$. Rings have a constant bisection bandwidth and a cost that is linear in network size. 2D meshes also have linear cost, but their bisection bandwidth is proportional to $\sqrt{N}$. For both these networks, the range of scalability is limited by performance. On the other hand, both the bisection bandwidth and cost of the complete graphs are proportional to N^2, and the range of scalability is limited by cost.

If a network family has a range of scalability $[N_0, \infty]$, we call it *asymptotically scalable*. Most practical networks are not asymptotically scalable; instead networks have a bounded range of scalability. Although asymptotic scalability is interesting from a theoretical point of view, practical considerations limit the range of machine sizes that are interesting. It is unnecessary to consider building machines that cost so much, or consume

[1] In this dissertation, we measure performance in terms of the bandwidth of the network under randomly distributed communication traffic and we estimate the network performance using its peak throughput. We saw that for such traffic the bisection bandwidth is a good estimator of network performance

[2] assuming a fixed channel width

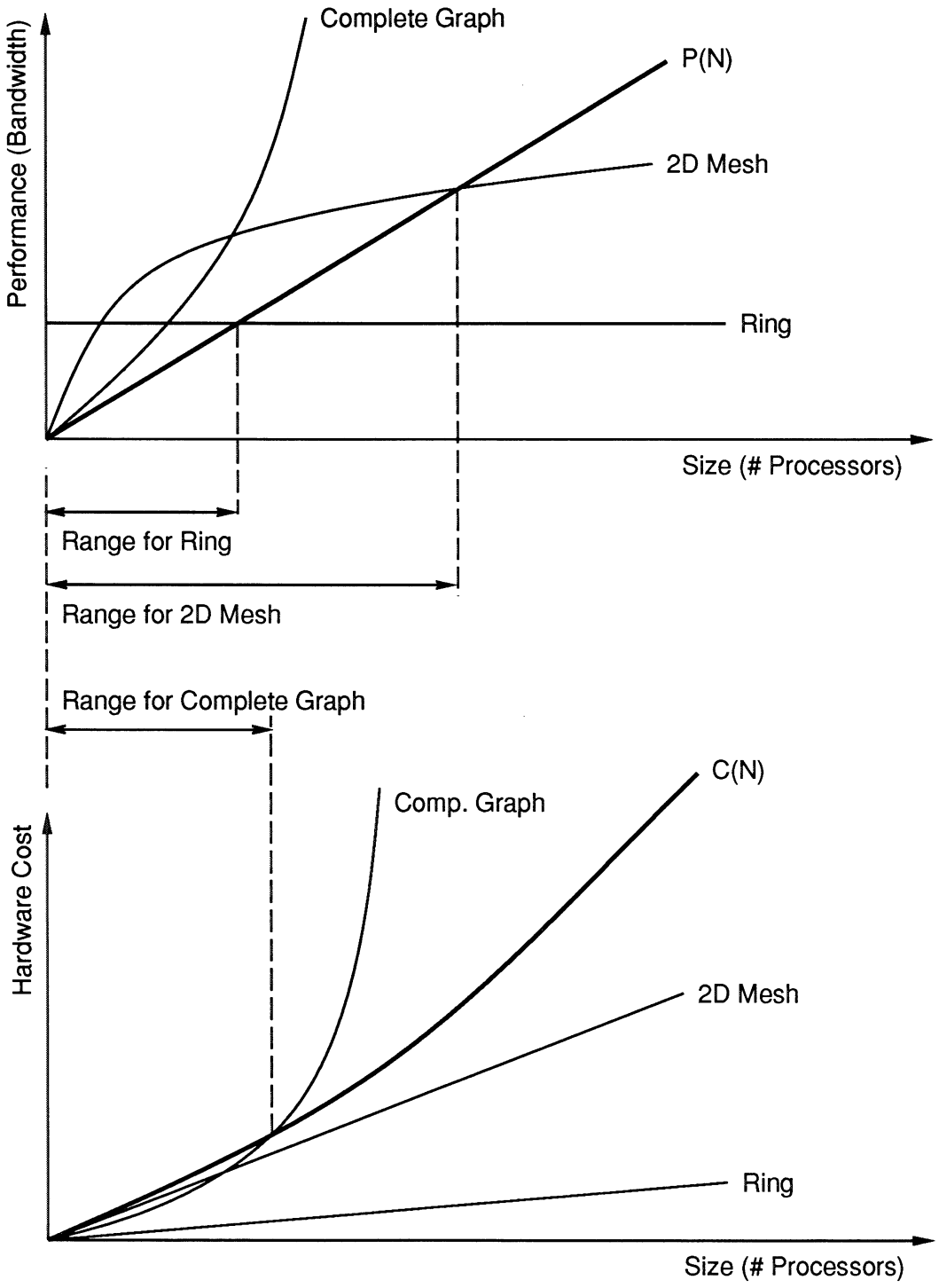


Figure 7.1: Ranges of Scalability

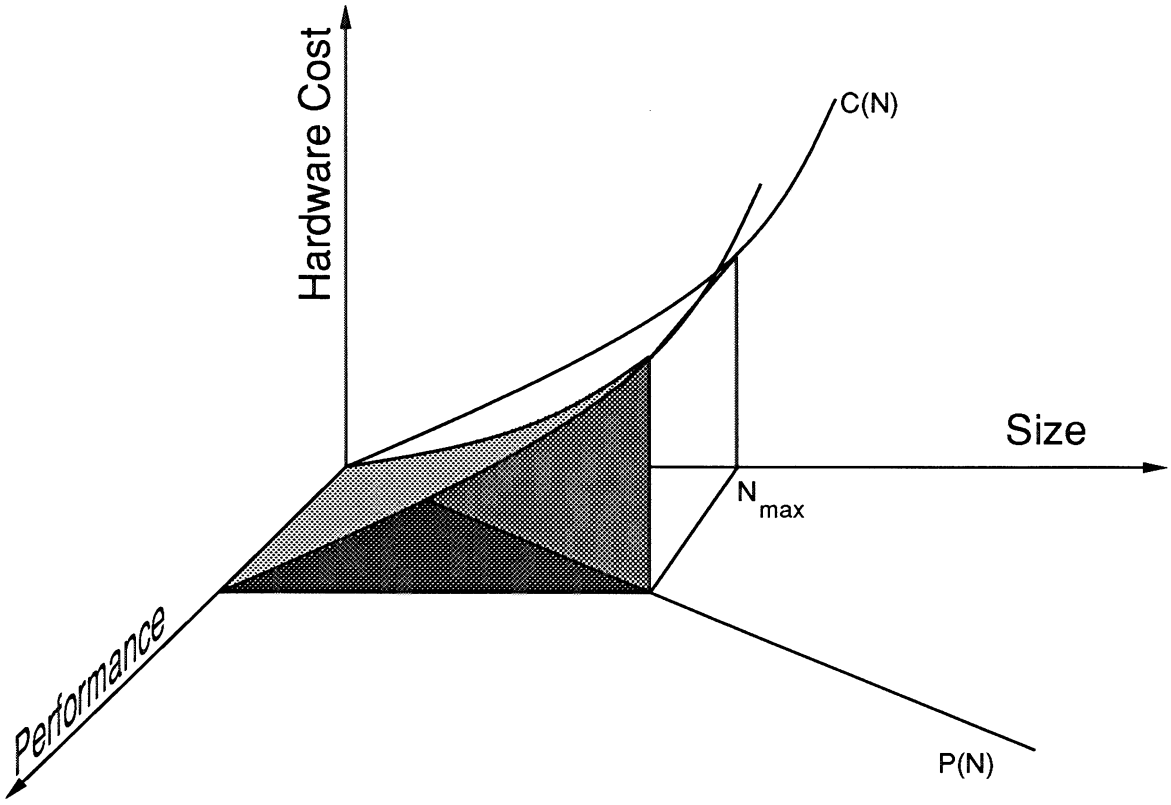


Figure 7.2: Scalability space

so much power, or occupy so much space, that nobody would be able to afford them. For parallel computers built using state-of-the-art microprocessors, we believe that the upper limit is in the neighborhood of 1000–2000 processors[3]. Accordingly, we are mainly interested in networks with a range of scalability $[N_0, N_{max}]$, where N_{max} is defined based on technological and economic considerations. N_0 can be taken to be 0 without loss of generality.

To summarize, we define a network family to be *scalable* if all the members of the family fall within the shaded region shown in figure 7.2.

[3]The largest available configurations of many of the current parallel machines that use reasonably powerful 32-bit processors, such as the CM5, KSR, Intel Delta, etc, are of the order of a 1000 processors [CKP+92]. The reason for this limit is primarily cost; the average cost these machines is in the range of $10,000 to $30,000 per processor [BBDS92], and not many customers are willing to spend more than $30-50 Million on a parallel machine.

7.2 Design costs vs. Hardware costs

Our definition of scalability does not specifically exclude families in which each member of the family has a completely different architecture design. For example, it is possible to define a family of machines which use busses for up to, say, 32 processors, 2D mesh from 32 to 128 processors, and maybe a butterfly for sizes larger than 128 processors. This artificial family illustrates a deficiency in our definition, namely that it ignores design costs.

While it is relatively easy to compare the performance of two network families, and it is possible to use models developed in this thesis to characterize hardware (or packaging) costs, modeling design costs is an extremely complex task. While it is possible to associate physical resources such as pins and area with hardware costs, design costs tend to be intangible and hard to quantify. Instead of trying to develop a model to characterize design costs we will characterize design costs in terms of the fraction of the machine that remains common across the members of the family. An artificial family such as the one described above will have a high design cost since there is very little in common between all the members of the family.

Recall that we model packaging technology as consisting of a hierarchy of l levels of packaging modules, where each level-i module consists of some number of level-$(i-1)$ modules which in turn consist of level-$(i-2)$ modules and so on. The packaging hierarchy gives us a natural way of classifying network families according to design costs. We consider a network family which uses the same level-i modules in all members of the family to have lower design costs than one in which only level-$(i-1)$ modules are common to all members, but the level-i modules are different. In the latter family the building blocks are smaller, and the design of all the levels above level-$(i-1)$ have to be done separately for each member.

When all the members of a network family use the same level-i modules, we say that the family is *scalable by level-i reconfiguration* since changing the machine size requires reconfiguring the interconnections between level-i modules. If a customer wants to upgrade a machine, the machine is dismantled down to its level-i modules, additional level-i modules (which include the additional processors) are brought in and all the level-i modules are then interconnected to obtain the new machine.

Scalability by level-0 reconfiguration means that all machines in a family use the same level-0 modules, which we earlier defined to be processors, memory units or routing

nodes. We can consider networks in which even the level-0 modules change from one member to another. Such families have nothing in common across their members, except perhaps, some high level design concept. We call such families *scalable by concept*. Each member of the family uses distinct building blocks that may all be designed based on the same concept. Upgrading such machines to larger sizes essentially means the customer replaces a smaller machine by a larger one. This form of upgrade is similar to upgrading work-stations or other uniprocessor machines.

At the other extreme are families in which the larger networks actually contain the smaller networks as a sub-portion. We say that such families are *scalable by inclusion*. Note that scalability by inclusion is not the same as scalability by level-$(l-1)$ reconfiguration though the top level of the hierarchy is level-$(l-1)$. In scalability by level-$(l-1)$ reconfiguration, the connections between the top level modules can be changed from one family to another, but this is not permitted by scalability by inclusion. Upgrading machines that are scalable by inclusion is extremely simple. The customer purchases additional processors and network hardware, and simply connects them to the existing machine.

Clearly a machine that is scalable by inclusion is also trivially scalable by level-$(l-1)$ reconfiguration, which in turn is trivially scalable by level-$(l-2)$ reconfiguration and so on. Therefore these categories classify interconnection networks into mutually inclusive categories as shown in figure 7.3. As we go towards the center of the figure, design costs decrease owing to the additional constraints on network design. However, as we pointed out earlier, these constraints may increase packaging costs or reduce performance. We shall examine these trade-offs in the following sections.

7.3 Scaling of previous network designs

In Chapters 4, 5 and 6, we considered network designs for 1024 processor machines under different models of packaging costs. The designs that we evaluated were specific to a machine size of 1024. We now consider extending the designs to a family of networks of different sizes and analyze the variation in packaging costs as the network designs conform to the various categories of scalability described above. Here, we consider networks under packaging model 1, but a similar analysis can be performed for the other packaging models. We consider a range of scalability of 1–1024 processors, and a peak communication bandwidth of 8 bits per processor, per cycle, under randomly distributed communication.

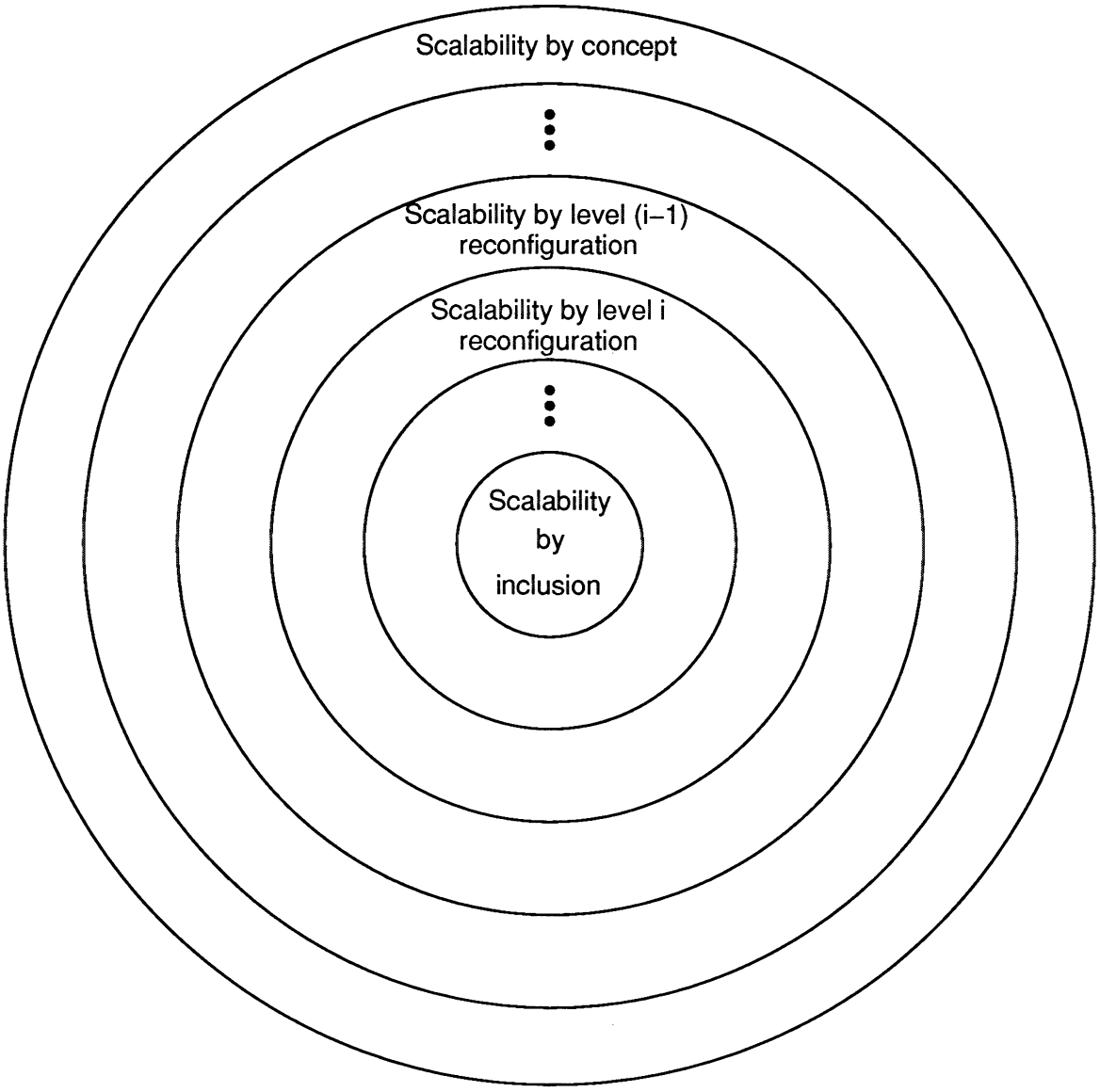


Figure 7.3: Classifying networks in terms of scalability

The packaging technology used under model 1 consisted of a two level hierarchy where processors were placed within *modules*, and were interconnected using two levels of wiring, namely inter-module connections and intra-module connections. The cost function under this model had two components, the number of modules and the pin-count per module. We also had a restriction on the maximum number of processors we could place within a module. In Chapter 4, we considered modules which could hold 16, 32 or 64 processors, but in this chapter we restrict ourselves to 32 processors per module. In this section we show how we can build families of networks for the following network sizes: N=32, 64, 128, 256, 512 and 1024. We shall illustrate three separate families, corresponding to the three categories described above.

7.3.1 Scaling by concept

Under scaling by concept, we permit each member of a network family to be designed independent of the others. This approach permits us to design each member of the family so that it achieves the maximum performance for a given cost. Recall that under model 1, we showed that maximizing the performance for a fixed cost required the inter-module network to be a complete graph. We also saw that we could partition a butterfly network on N inputs into $\sqrt{N}$ modules, to get a complete graph between modules.

We could take the same approach to designing each member of the family of networks, but in this approach, since each module has $\sqrt{N}$ processors, networks smaller than 1024 processors use fewer processors per module than permitted. This results in a larger number of modules than necessary and increases the cost. So we fix the number of modules $M = N/32$ and build a complete graph over these modules.

A natural way of building a complete graph on M modules, is to partition a butterfly network on M^2 inputs. However, the resulting network will only have M inputs per module, but N/M processors per module. Therefore, we connect N/M^2 processors to each input using a binary tree. Memory units are connected in a similar manner. Since the per processor network bandwidth remains constant, all the networks will have approximately the same number of pins per module. However, these pins will be partitioned among fewer channels in the smaller machines, resulting in wider inter-module channels in the smaller machines. Figure 7.4 shows the internals of a module for a network with $N = 256$.

In this family of networks, the channel widths and the amount of routing hardware

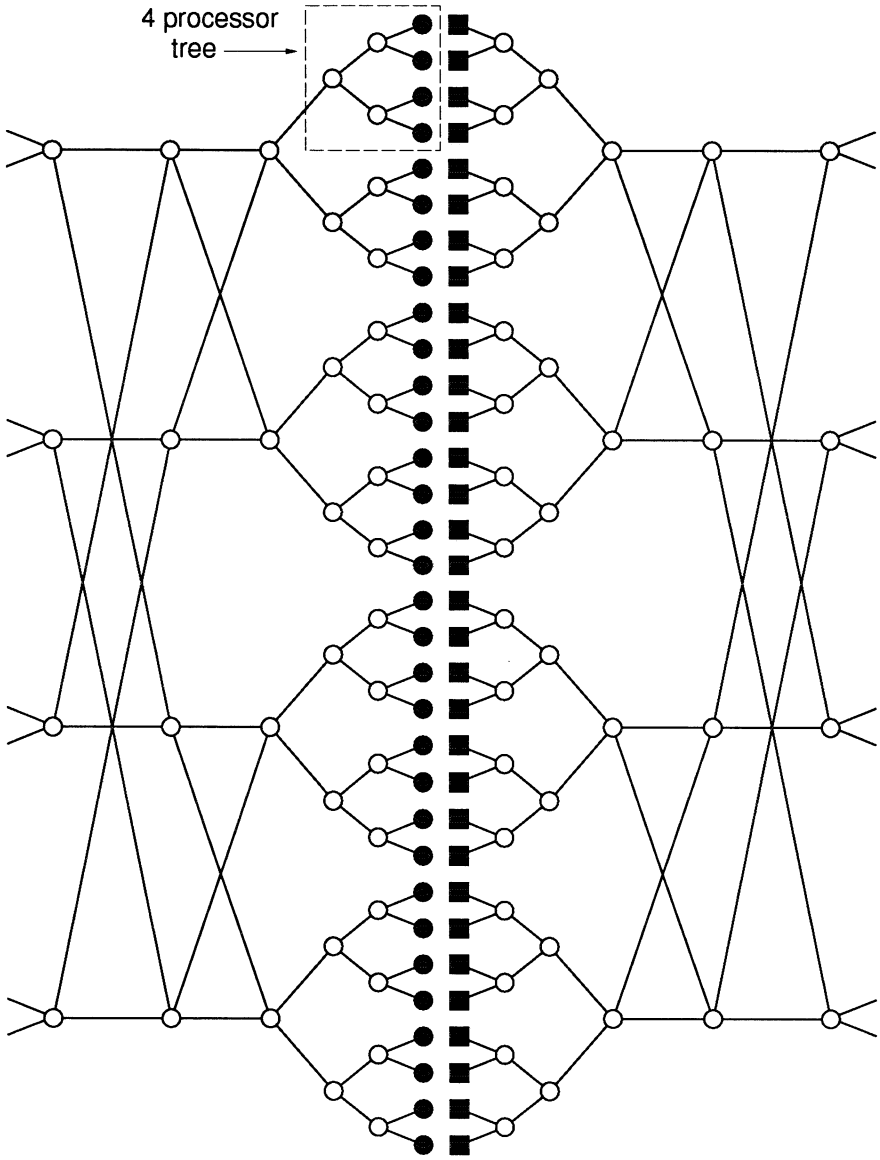


Figure 7.4: Scaling by concept: 32 processor module for a 256 processor network

per module change as we scale the machine. Hence this mode of scaling is considered to be *scaling by concept*. We can also extend this construction to machine sizes larger than 1024 until the point where we have sufficient pins per module to build a complete graph between modules. Machine sizes beyond that are not feasible in this family.

7.3.2 Scaling by level-1 reconfiguration

For scaling by level-1 reconfiguration, we fix the network within a level-1 module, and use the same module to build machines of different sizes. Fixing the contents of a module also fixes the total number of wires leaving the module, and the way the wires are partitioned into channels. We can divide the total number of wires leaving a module into 32 channels corresponding to a machine size of 1024 processors. Machine sizes smaller than 1024 processors, require wider channels between the modules. Since we cannot change the widths of the channels, as we did in the previous family, we use channel dilation to achieve the required number of wires between any pair of modules. If there are N processors, we need $M = N/32$ modules. The connection between any pair of modules consists of $32/M$ channels. This gives us the required peak bandwidth needed to meet the performance specification. In this network, when we double the number of processors, we have to re-organize connections between modules by reducing the dilation of the inter-module channels by a factor of 2 to accommodate the new modules that are to be included.

Each network in this family requires approximately the same number of pins as the previous family. Since packaging cost was assumed to be equal to the pin-count under model 1, this network family has approximately the same packaging cost as the previous one. However, this family requires more routing hardware per module. The additional hardware contributes to higher packaging costs, but this difference is ignored by our packaging model.

7.3.3 Scaling by inclusion

In the two previous families, we used a butterfly network to interconnect the processors. This network consisted of $\log N$ switching stages and connected the processors at one end to the memories at the other end. Since processors and memories are logically paired with each other, we can think of this network as folding a butterfly over itself so that each input is paired with an output. If we have to double the number of processors, we have to double the number of rows of the butterfly and increase the number of columns

by 1 and then fold the network over itself. Since the additional column of switching nodes falls between the processors and memories, this network does not lend itself to scaling by inclusion.

One way of achieving scaling by inclusion, is to double the number of routing stages to $2 \log N$ to generate a Benes network which is shown in figure 7.5. The network is shown with processors on the left and memories on the right, but it is actually folded over itself so that the processors and memories are placed adjacent to each other. A Benes network is equivalent to a fat-tree network in which the total bandwidth at each level remains constant. Scaling the network in this design requires adding additional modules consisting of only routing nodes. An example of scaling the network from 16 to 32 processors is shown in figure 7.6.

Two of the currently available commercial machines, the Meiko CS2 and the CM5[4] use the folded Benes network. Another network seen in commercial machines that exhibits the property of scaling by inclusion is the Hypercube. In both networks, the average number of hops made a message grows logarithmically in N, the machine size, and in both networks the total number of wires grows as $N \log N$. We already noted that we need a super-linear increase in the number of wires to achieve a linear increase in throughput. The folded benes achieves this super-linear increase in wires by adding routing nodes, where as the hypercube increases the number of wires per node. For scaling by inclusion, we can not actually increase the number of wires per node, so each node in the hypercube should already have enough wires for the largest machine we plan to build. The hypercube, then, wastes these wires in the smaller configurations since they remain unused.

In the family of folded benes networks, the average number of hops that messages have to make, to get from their sources to destinations, is twice the number of hops in the butterfly network. To meet the same peak bandwidth specification, networks in this family require approximately twice the number of routing nodes and twice the number of wires as the butterfly network. We also have restrictions in partitioning this network because we do not want to modify the existing network when we scale the network to a larger size. We can put 32 processors per module as before, and include five columns switching stages within each module for each of the two butterfly networks. For larger machines we need more routing stages and these have to be placed in additional modules which are connected

[4]In the case of the CM5 the bandwidth does not remain constant, but reduces by factor of 4 in the first two levels

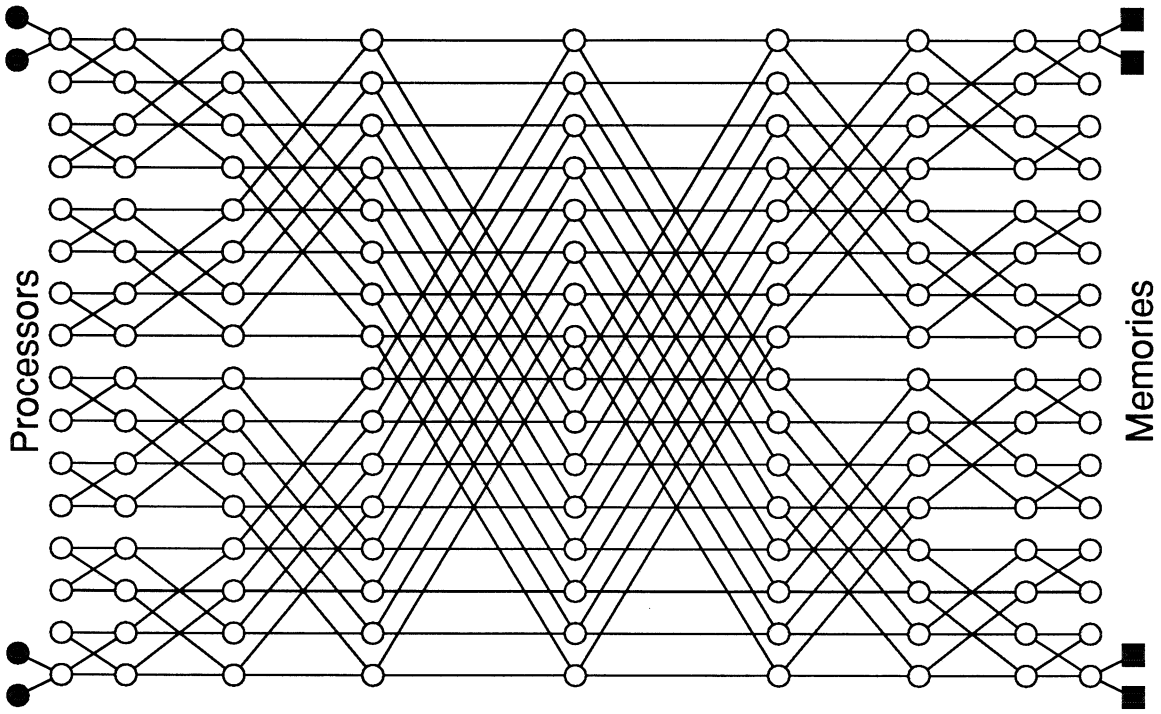


Figure 7.5: Benes network

to the modules that contain processors.

Since the partitioning of the network into modules is different from that of the two previous families, it is difficult to compare the packaging costs of this family with that of the two previous families. However, since we can see that this family uses approximately twice as many wires and routing nodes as the two previous families, we can expect the packaging costs of this family to be around twice that of the previous families.

7.4 Performance Comparison

In this section, we compare the performance of the three families of networks described above. For fairness, we hold the packaging costs of the three families the same while comparing the performance. The differences in performance illustrate the trade-offs between network performance and design cost.

The family of networks that are scalable by concept and the family of networks that are scalable by level-1 reconfiguration have the same packaging cost as per our packaging model. Therefore a direct performance comparison between the two families is justified.

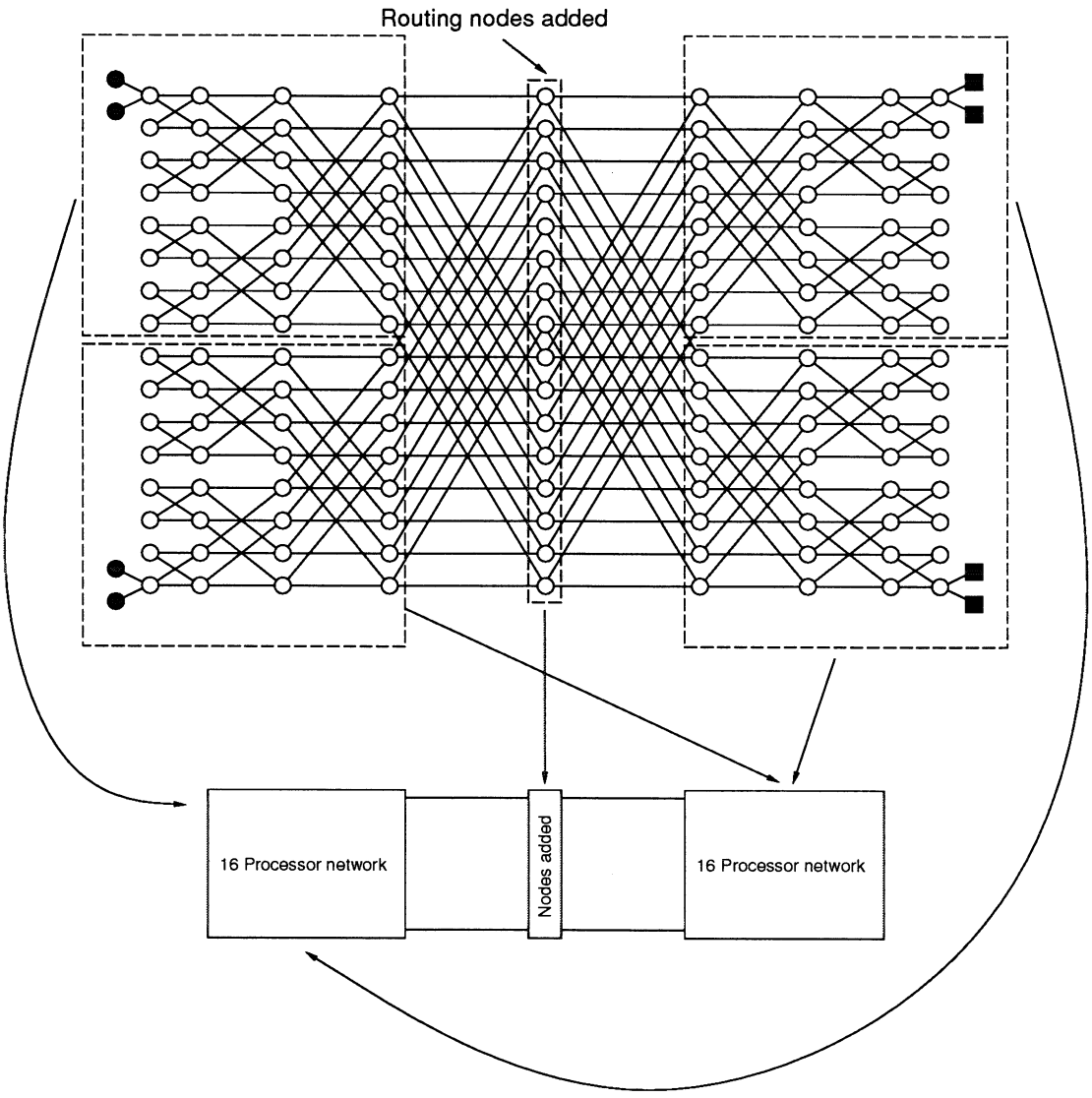


Figure 7.6: Scaling a benes network

However, the packaging cost of the third family (fat-trees or benes networks) is approximately twice that of the other two families. To normalize packaging costs to the same level as the other two families, we have to reduce the bisection width of this network by a factor of two. This will lead to a drop in the performance by approximately a factor of two, making the latter family substantially worse in comparison to the other two families. This suggests that we lose approximately a factor of two in performance for the lower design cost and higher scaling flexibility provided by scaling by inclusion.

Figures 7.7 and 7.8 compares the performance of the networks that are scalable by concept to the networks that are scalable by level-1 reconfiguration. All the networks have the same peak performance (on a per processor basis), but they differ in the fraction of peak performance that is achieved under simulation. When the number of threads is small, we see that the networks that are scalable by concept provide better performance. The main reason for this difference in performance is the difference in the organization of the inter-module wires. In the first family, the wires that go between modules are organized as wide channels, while in the latter the same number of wires is organized as many narrow channels. At low levels of multithreading, the number of messages in the network is insufficient to utilize all the wires in the latter family. As the number of threads increases, the difference in performance reduces, since increasing the number threads results in a larger number of independent messages in the network. In other words, the higher design cost of scaling by concept does not buy much in terms of performance when the number of threads is high, but does so when the number of threads is low.

7.5 Summary

Scalability of parallel machines has long been an intuitive notion without a formal definition. In this chapter we first provided a formal definition of the term which gave us a means for classifying network families as being scalable or unscalable. Our definition of scalability was based on the manner in which the performance and cost increased as we increased the size of the machine. We argued for a linear increase in performance and that a linear increase in performance usually implied a super-linear increase in hardware cost.

We considered two aspects of the cost of a machine, namely hardware (or packaging) costs and design costs. Although design costs were difficult to quantify, we were able to classify scalable networks into three broad categories based on the relative design

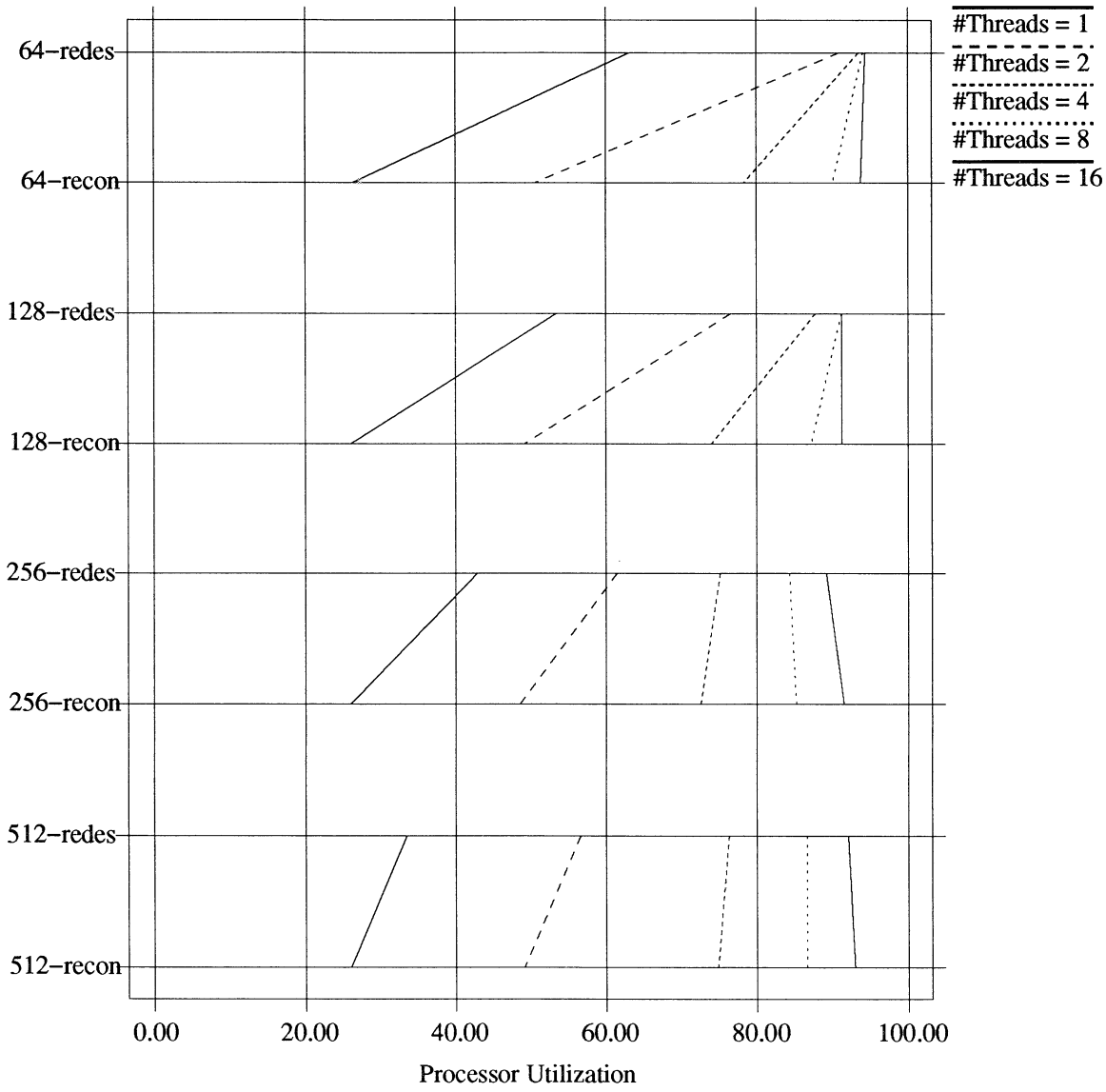


Figure 7.7: Periodic Access Model, access interval = 16 cycles

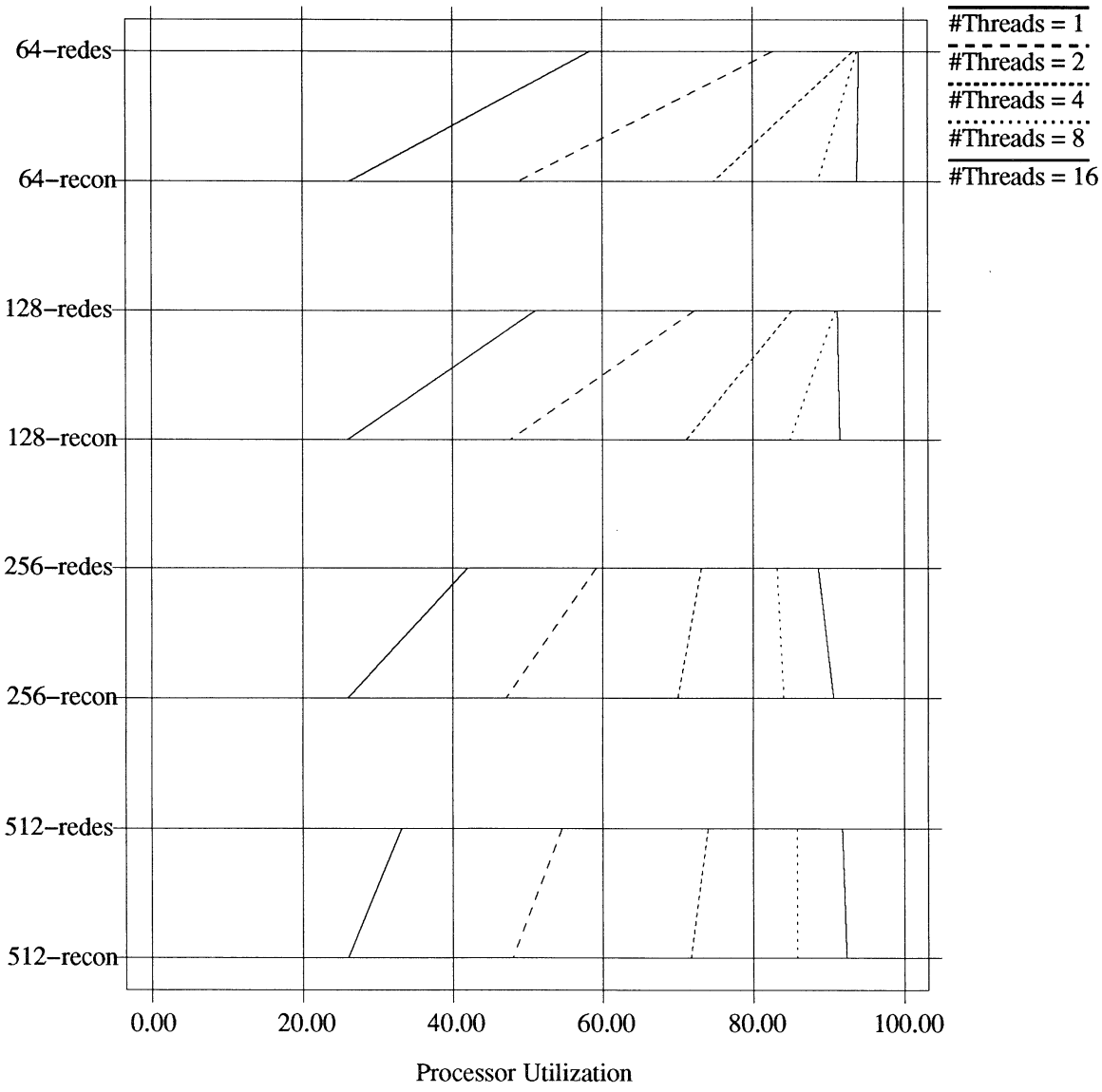


Figure 7.8: Geometric Access Model, mean access interval = 16 cycles

costs associated with them. At one extreme there were families of networks which had high design costs, but low hardware costs for a given performance, and at the other extreme were networks that had low design costs but high hardware costs. We demonstrated extensions to some of our previous designs according to each of the three categories. We compared these three families on the basis of equal hardware costs to illustrate the trade-off between performance and design costs.

Chapter 8

Fault-Tolerance

Most of the networks that we evaluated in this thesis, and found to provide the highest performance for a fixed cost, also had the property that there was a single path from any input port to any output port. While the single path property simplifies the design of the routing nodes, it also poses difficulties in the presence of faults. Even a single faulty routing node or faulty channel in the network can affect many different paths.

In this chapter we present a simple and practical fault-tolerance scheme to overcome the problems caused by network faults. Our approach to dealing with faults is to send messages using two passes over the network when the direct path is faulty. Our approach is an extension to the work by Varma and Raghavendra [VR89] and Leighton et al. [LMS92]. Our fault-tolerance scheme has the following properties:

1. It requires no modification to the underlying network. We do not add additional links or extra stages to the butterfly. The design of the routing hardware remains the same as a network that does not have any fault-tolerance properties.

2. When new faults are discovered, the machine is halted and it enters a recovery phase. During the recovery phase, we compute two pass routes for messages whose paths are affected due to faults. At the end of the recovery phase, each processor will have a routing table that specifies these routes. Since we wish to maintain complete communication capability between all the *alive* processors, we may need to disable some processors to ensure this property. Our experimental results show that the number of processors lost is minimal.

3. The performance of the network suffers some degradation. The extent to which this

happens is quantified in Section 8.4.

In Section 8.1 we give a brief description of our fault-tolerance scheme and in Section 8.2 we describe the operations that are carried out during the fault-recovery phase. Section 8.3 discusses related work. Finally Section 8.4 presents a simulation based evaluation of our fault-tolerance scheme.

8.1 Approach

We model network faults as faults in the routing nodes. No messages may be routed through faulty nodes. We assume that faults are placed randomly and that the locations of the faulty nodes are globally known.

We denote the fact that the direct path from processing element (PE) p to q is fault-free by $p \rightarrow q$. $p \not\rightarrow q$ implies that at least one node along the path is faulty. If $p \not\rightarrow q$, then we attempt to find an alternate path for messages from p to q. This path consists of two passes through the network. The message first goes from p to some other PE, say r. The message is then sent from r to q. We call r the relay for messages from p to q. The selection of relays is done off-line during the recovery phase and stored in tables at each PE. Each PE maintains a table of size N which tells if the direct path to the corresponding destination is faulty, and if so, the address of the chosen relay. Since the selection of relays is done during recovery, all messages from p to q pass through the same relay r.

The routing nodes in the network route the messages based on routing tags that specify the destination of a message. In order to use the two pass scheme for sending messages, each message carries two addresses; the address of the relay and that of the final destination. During the first pass, the address of the relay forms the routing tag. When the message reaches a PE, it checks the destination address of the message. If the destination address is different from its address, it replaces the routing tag with the destination address and sends the message along.

8.2 Fault Recovery

During the recovery phase, we compute the routing tables for all the PEs. The computation of these tables is done in two parts. First we compute the set of *alive* PEs

subject to the constraint that all of them can communicate with each other. Then, we select the relays in a manner that maximizes the throughput of the network.

We describe the computation of the routing tables for the network *Butterfly.32.8* that was discussed in Chapter 4 under model 1, but the computation readily extends to other networks as well.

8.2.1 Computing the *alive* set

We first compute the N by N direct connectivity matrix D. Matrix D is a boolean matrix with entry $D(p,q) = 1$ if $p \rightarrow q$, otherwise $D(p,q) = 0$. Each row of the matrix can be easily computed using a breadth first search from the corresponding source to figure out which destinations are reachable. Note that $D(p,q)$ need not be equal to $D(q,p)$ in general. This is because the path from p to q is different from the path from q to p and it is possible that only one of the two paths is faulty.

Next we compute the boolean matrix $T = D^2 \vee D$. $T(p,q) = 1$ if $D(p,q) = 1$ or there is a two pass route from p to q. Since we require complete communication capability for all the PEs in the *alive* set, we will disable PEs until the sub-matrix of T corresponding to the *alive* set is all 1s. Disabling a PE corresponds to deleting the row and column associated with that PE from the matrix T. Obviously, we would like to disable the minimum number of PEs, but finding the minimum set is a difficult problem. T represents a directed graph on N vertices as an adjacency matrix. In order to compute a maximum *alive* set, we need to find the maximum clique in the directed graph represented by T. In general, finding the maximum clique in a graph is an NP-complete problem, but we find that when the number of faults is small, the following greedy algorithm yields large cliques and is acceptable in practice.

We first transform this directed graph into an undirected graph by replacing each pair of back-to-back edges by a single undirected edge. Unpaired edges get deleted. We sort the vertices of this undirected graph by vertex degree and eliminate vertices in increasing order of vertex degree until we are left with a complete graph. The first vertices to get eliminated are the isolated vertices. The isolated vertices correspond to the processing elements that are connected to faulty switches in the first or last columns. When the number of faults is small, we find that we are usually left with a complete graph after the isolated vertices are eliminated. When the elimination of isolated vertices does not result

in a complete graph, the number of additional vertices that need to be eliminated is usually very small. It may be possible to retain these vertices in the *alive* set if we allow messages to make more than 2 passes through the network, but increasing the number of passes degrades network throughput. Our results indicate that the increase in the size of the *alive* set does not offset the throughput degradation nor the added complexity.

8.2.2 Relay selection

After computing the *alive* set we select relays for all PE pairs that cannot communicate directly. Let the set $R_{p,q} = \{r \mid p \rightarrow r \wedge r \rightarrow q\}$. Any element of $R_{p,q}$ can serve as a relay for the messages from p to q. However, the selection of relays affects the throughput of the network. For example, we do not want our relay selection algorithm to send a large number of messages through a single channel or a single PE. Our objective is to select relays so that network throughput is maximized under random communication.

Recall that the best network architectures had wider channels at the lower levels of the packaging hierarchy and narrower channels at the higher levels of the hierarchy. Therefore, the channels at the upper levels of the hierarchy are critical as far as network throughput is concerned and maximizing throughput requires balancing the load on these critical channels. For the network *Butterfly.32.8* the critical channels are those that go between the modules. The load on a critical channel x, L_x, is the expected number of messages that pass through the channel when each PE sends a message to a randomly chosen destination. Let $\hat{L} = \max_x(L_x)$. $\hat{L}$ is the worst case load and we use the function $1/\hat{L}$ as an estimator of network throughput.

In a fault-free network, since there are N PEs, the probability that any given pair of PEs will communicate is $1/N$. Each direct path between any two PEs (in *Butterfly.32.8*) will contain exactly one critical channel and the number of message paths that pass through any critical channel will be equal to N. The load on each critical channel will be 1. In the presence of faults, some messages require 2 passes through the network. These messages will contribute to the load on two critical channels, one for each pass. Also the number of alive PEs (denoted by N_A) will typically be less than N, therefore the individual communication probabilities will be $1/N_A$. Given an assignment of relays we first determine the number of message paths that pass through each critical channel x. The number of paths divided by N_A gives L_x.

Finding an optimum assignment of relays that minimizes the function $\hat{L}$ is difficult and expensive in terms of computation time[1]. Instead of an algorithm that finds an optimum assignment, we use an algorithm that sequentially selects relays in a greedy manner. We first compute the loads for the direct fault-free paths as follows. Let $C(p,q)$ denote the critical channel on the path from p to q. For all pairs (p,q) such that $p \rightarrow q$, we increment the load $L_{C(p,q)}$ by $1/N_A$. Next we pick pairs of PEs (p,q) such that $p \not\rightarrow q$ one at a time and assign relays for them. For each $r \in R_{p,q}$, we evaluate $\max(L_{C(p,r)}, L_{C(r,q)})$ based on the current loads on the critical channels. We choose that r for which this function is the smallest. If many PEs have the same minimum value, we choose one out them at random. We assign this PE as the relay for pair (p,q) and increment the load on the corresponding two critical channels by $1/N_A$. We repeat this until all relays have been selected.

Our relay selection algorithm has a time complexity of $O(N^3)$ and a space complexity of $O(N^2)$. Despite the high asymptotic time complexity, computing the tables for a machine of size 1024, and 200 faults, requires only about 10 Megabytes of memory and takes less than 15 minutes on a 16 Mhz RISC work-station.

8.3 Related work

A large amount of work has been done on the topic of fault-tolerance of butterfly networks. A significant fraction of this work consists of techniques that augment the network by adding extra links or additional stages. See [IAS87] for a survey.

Varma and Raghavendra [VR89] have studied a problem similar to the one discussed in this chapter. They provide a characterization of fault patterns in a butterfly network based on how many passes are required to guarantee full connectivity between PEs. They base their analysis on the absence of faults in the first and last columns of routing nodes and show that under certain conditions 3 passes are sufficient to establish full connectivity. If a set of weaker conditions is satisfied, they show that a maximum of $\log N - 2$ passes is sufficient. Their main concern is connectivity; balancing the load on the channels and maximizing network throughput is not considered. We limit ourselves to 2 passes through the network since increasing the number of passes can cause significant degradation in throughput. In practice, network throughput is probably as important as connectivity. If all messages require $\log N$ passes, the network throughput will degrade by

[1] We suspect that this problem is also NP-complete.

at least a factor of $\log N$, and if all the processors are active, it is likely they will run at $1/\log N$ of their normal speed.

In a recent paper [LMS92] Leighton et al. analyze the fault-tolerance properties of butterflies from a theoretical viewpoint. They consider a network that has a PE at each node of the butterfly. They prove that even when a large number of nodes are faulty, the network can emulate a fault-free butterfly with $O(1)$ slowdown. Their routing algorithm is similar to ours in the sense that messages essentially make two passes over the butterfly. They select relays at random and show that the congestion in the network is $O(\log N)$ with high probability. They then use Ranade's scheduling algorithm [Ran88] to route messages in $O(\log N)$ steps. Their results are not directly comparable to ours since our network architecture is different, but the underlying ideas are similar.

In [LLM90], Leighton et al. evaluate a network called the multibutterfly [Upf89] which has excellent fault-tolerance properties. A multibutterfly consists of a pair of superimposed butterfly networks, where the nodes of one of the networks are randomly permuted within each column. The random wiring between the stages helps in ensuring a high degree of fault tolerance, but it also destroys the partitionability of the network. The pin-requirements of a partitioned multibutterfly are significantly greater than a partitioned butterfly.

8.4 Performance results

The fault-tolerance scheme presented earlier can be applied to any of the networks presented earlier in this dissertation. In this chapter, we evaluate this scheme for the network *Butterfly.32.8*. This network has 1024 processors, divided among 32 modules, with 32 processors and memory units (processing elements) in each module. The intra-module channels are 32 bits wide and the inter-module channels are 8 bits wide. The network has a total of 5120 routing nodes, organized as 10 columns of 512 nodes each. Faults are placed on the routing nodes at random.

We first measured the cost of limiting the message paths to a maximum of two passes. If we permitted messages to make more than two passes we might potentially be able to include more PEs in the *alive* set. Figure 8.1 shows the fraction of PEs that remain alive in the presence of faults. The line labeled `Alive` is the actual fraction of PEs that remained alive under our scheme and the line labeled `Max alive` is the fraction of PEs that

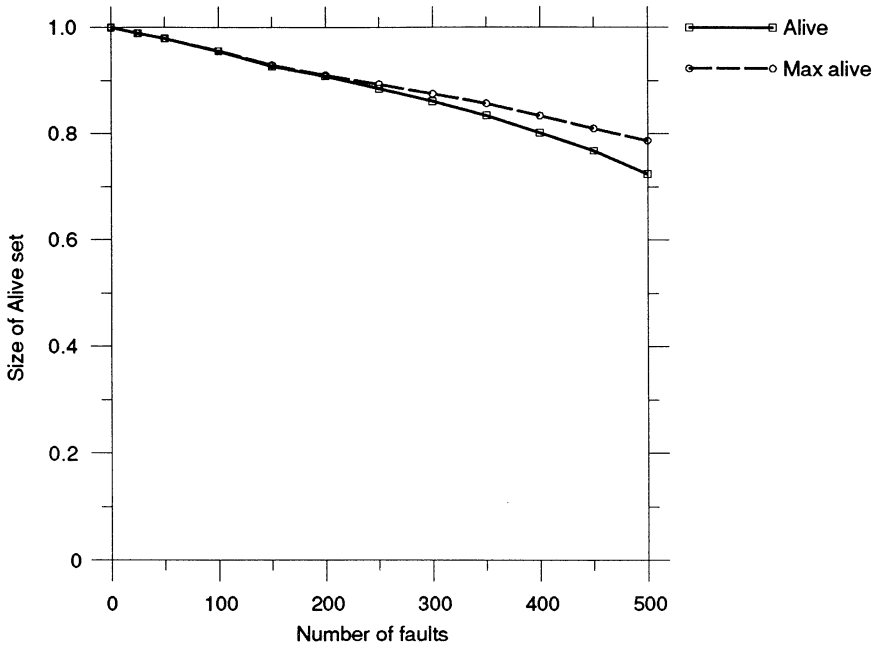


Figure 8.1: Cost of limiting paths to two passes

would remain alive if we had permitted an unlimited number of passes through the network. We can clearly see that when the number of faults is small (less than 200), two passes are sufficient. Almost all the PEs that are lost are those that are connected to faulty routing nodes in the first or the last column. Even when there are 500 faults, the discrepancy between the two lines is only 6% indicating that, in practice, two passes are sufficient.

In figure 8.2 the line labeled `Greedy` shows the value of the worst case load ($\hat{L}$) that was obtained by the relay selection algorithm described in the previous section. The line labeled `Random` shows the value of $\hat{L}$ for an algorithm that assigns relays at random from among the set of valid relays. The line labeled `Average` is the average load over all the critical channels. We can see that our algorithm performs much better than `Random`, and that the peak load is within 30% of the average when the number of faults is less than 200.

We first simulated the network under a saturating load where each *alive* PE sent a message to a random destination whenever the network was capable of accepting a message. We measured the peak throughput as the average number of messages delivered per unit time in steady state. Messages were routed using virtual cut-through routing (see section 3.4 for a description of the routing nodes, buffering, etc). Figure 8.3 shows the observed peak

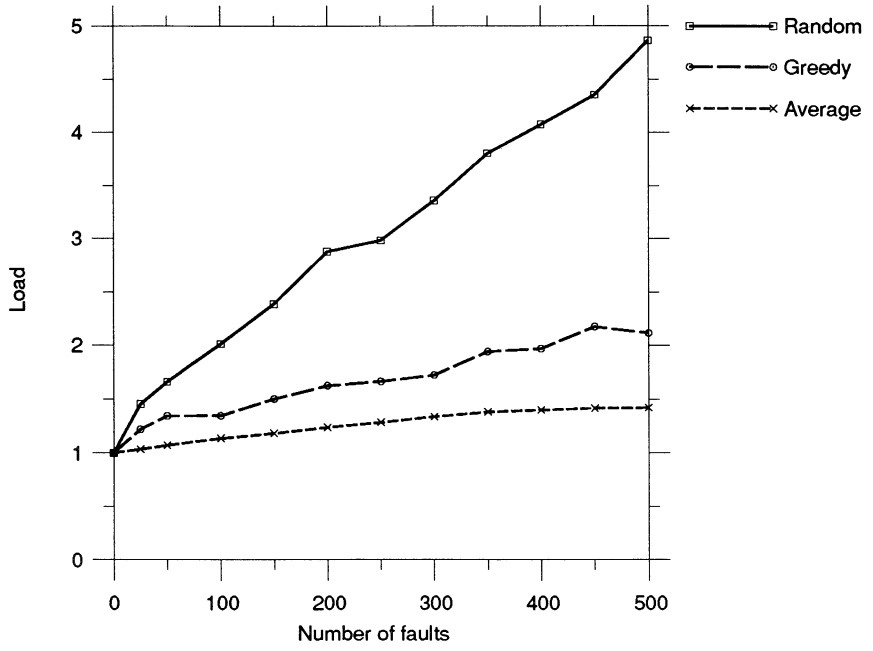


Figure 8.2: Performance of the relay selection algorithm

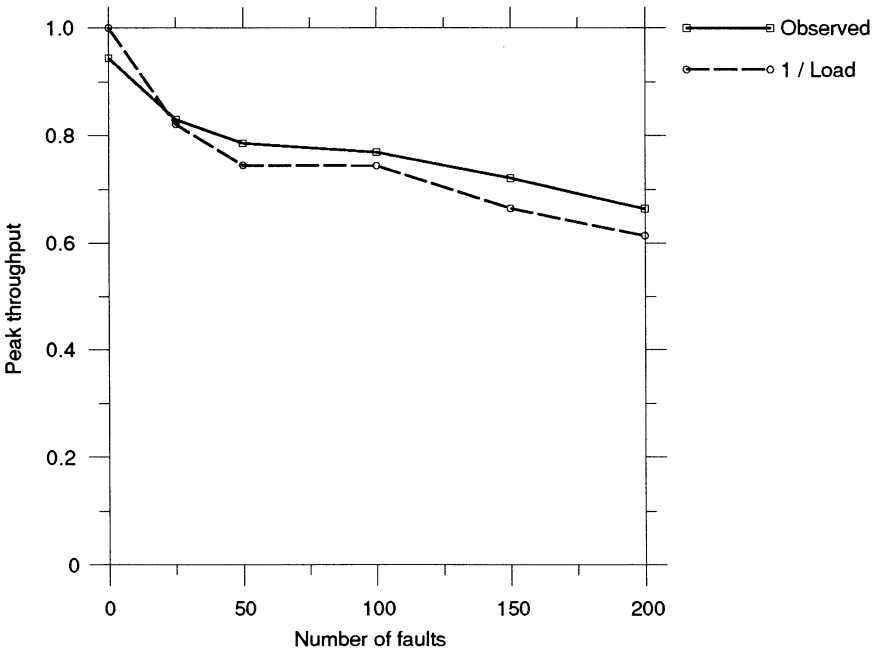


Figure 8.3: Peak throughput under saturation load

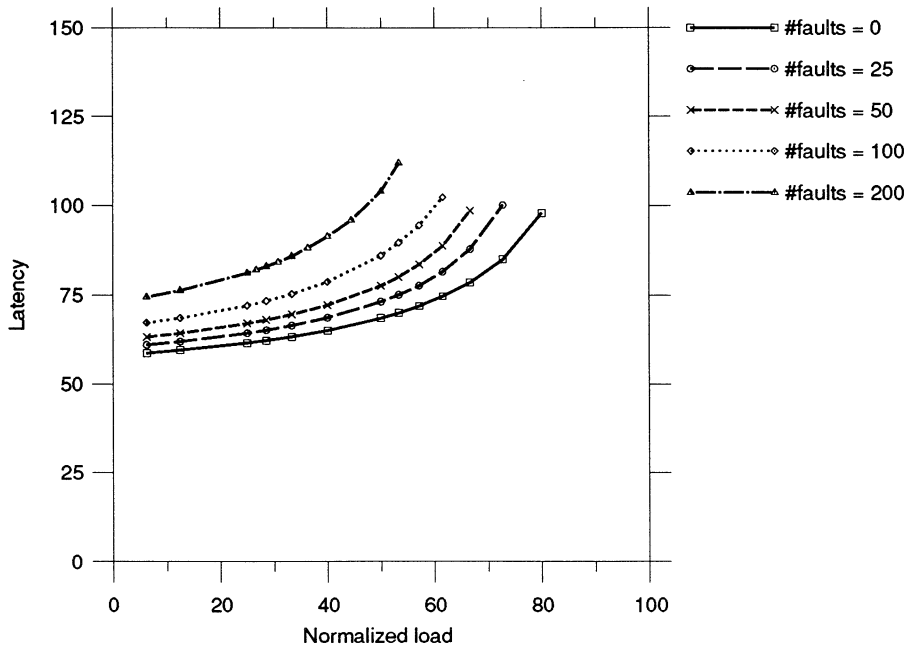


Figure 8.4: Latency versus message rate

throughput (on a per processor basis as a fraction of the capacity of a fault-free network) versus the number of faults. It also shows the throughput estimated as $1/\hat{L}$. We can see that $1/\hat{L}$ is a reasonable estimator of peak throughput and that the throughput degrades by about 20% with 100 faults and about 30% with 200 faults.

We also simulated the network under the open-network workload model. Figure 8.4 shows the results of the simulation. The message rate is shown as a fraction of the capacity of a fault-free network. We can see that as the number of faults increases, the average latency increases because more messages need two passes. As the number of faults increases we also see that the knee of the curve shifts to the left indicating that the peak throughput decreases. When there are 100 faults, there is approximately a 20% increase in latency and a 20% decrease in throughput.

Chapter 9

Conclusions

In this dissertation , we addressed the problem of designing efficient interconnection networks for large scale parallel machines. The main objective was to design networks which provided high performance for low cost.

We developed a model of packaging technology to characterize the network costs in the context of a multi-level packaging hierarchy. In order to model the general characteristics of a wide range of packaging technologies we defined a few generic packaging models that relied heavily on the property of packaging locality. The generic models based the computation of network cost on the usage of resources the higher levels of the hierarchy while ignoring the lower levels.

For each of the generic models, we evaluated different networks using detailed simulation. The simplest of our generic models used the number of top-level modules and the pin-count per module as metrics of network cost. Under this model, we found that the best interconnection network at the top level was a complete graph. A complete graph at the top level corresponded to a butterfly network between the processors, and we found that using wider channels at the lower levels of the packaging hierarchy helped in improving the utilization of the networks at the top level of the hierarchy.

In the second generic model, the computation of network cost was more complex, and included a third parameter, namely the number of top-level bundles. Under this model, we were unable to prove the optimality of any particular topology, but we were able to demonstrate that some classes of topologies were close to optimal. We evaluated in detail one such class of topologies called n-hop networks. This class of topologies has not received much attention in literature, even though they provide a good trade-off between pin-counts

and number of bundles.

The third generic model dealt with costs at the top two levels of the hierarchy and illustrated an interesting trade-off between the number of bundles at the top level and the pin-counts at the next lower level. We found that by using a denser network at the lower level, we could reduce the number of bundles at the top level.

The results from the three generic models illustrate that although the top level network determines the performance of the network to a first approximation, we can get significant improvements in performance by making the lower level networks richer. We also found that, as a general rule, it was better to use fewer wider channels than many narrow channels. Wider channels were able to achieve higher utilization than narrow channels because more independent messages are needed to keep all the narrow channels busy. The results from the three models also illustrate that for shared memory based communication primitives, we can get better performance by using fewer memory units with wider ports to the network. This is due to a reason similar to the benefits provided by wider channels; it is easier to keep the memory units busy when there are fewer memory units with wider channels. This result suggests that it might be better to build networks out of small shared-memory multiprocessing units rather than single processor-memory pairs. Many of our network evaluations were based on multithreaded processors, and our results indicate that multithreading provides the processors with the ability to tolerate network latencies; latency tolerance translates to higher network performance.

We did not exhaustively compare all the network architectures at the lowest level of the packaging hierarchy, but we were able to demonstrate that some simple architectures achieved performance close to the upper bound. There may be other network architectures which have similar performance; for example it may be possible to build a 2D mesh at the lowest level, with sufficiently wide channels, so that it achieves performance close to the upper bound. When we consider the lowest level network in isolation, it is conceivable that many different networks achieve the same performance for different costs, and it may seem desirable to pick the one with the lowest cost. However, we believe that the cost of the lowest level network is only a small part of the overall network cost, and therefore, were ignored by our packaging models. If these costs did indeed matter, we could extend our model to include these costs, and analyze the cost-performance trade-offs at the lowest level[1].

[1] However, due to pin-count and bisection width considerations, similar to those considered in Chapter 4,

We formally defined scalability and divided networks into three broad categories based on the difficulty involved in scaling them. At the one end were networks in which the smaller networks were sub-portions of the larger networks. These networks can be easily scaled at the customer's site by simply adding on the additional parts required by a larger machine. At the other extreme were networks in which the larger networks had components that were different from the smaller ones. Scaling such networks essentially meant replacing the smaller network by a larger one. The other category of scalability represented networks that fell in between these two extremes. We evaluated the performance differences between the three categories of scalability and found that achieving greater scaling flexibility implied either an increase in cost or a reduction in performance. We provided a quantitative evaluation of the implications of designing networks with different levels of scaling flexibility.

We also presented a mechanism to tolerate faults in butterfly networks. The idea behind the fault-tolerance scheme was simple; any message that could not be sent directly was sent in two passes over the butterfly network. The difficult aspect of this fault-tolerance scheme, was the selection of paths for messages that were sent in two passes. These paths had to be chosen in a manner that minimized the performance degradation due to load imbalances in the network. In general, finding optimum paths for these messages was a difficult problem. We presented a few heuristics that selected paths in a manner that yielded a good balance in the load distribution in the network. We demonstrated that the network was able to provide good performance even in the presence of a large number of faults.

---

we expect that the 2D mesh is unlikely to be a contender for the network at the lowest level.

Bibliography

[A+90] R. Alverson et al. The TERA Computer System. In *1990 International Conference on Supercomputing*, pages 1–6, 1990.

[A+91] Anant Agarwal et al. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. Technical Report MIT/LCS Memo TM-454, Massachusetts Institute of Technology, 1991.

[ACF90] Bowen Alpern, Larry Carter, and Ephraim Feig. Uniform Memory Hierarchies. In *Proceedings of the IEEE Annual Symposium on The Foundations of Computer Science*, pages 600–608, 1990.

[AG89] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, 1989. Chapter 8.

[Aga91] Anant Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[AP90] S. Abraham and K. Padmanabhan. Constraint based evaluation of multicomputer networks. In *Proceedings of the International Conference on Parallel Processing*, 1990.

[Ath90] William C. Athas. Physically Compact, High-Performance Multicomputers. In *6th International MIT VLSI Conference*, pages 302–313, 1990.

[Bak90] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Publishing Company, 1990.

[BBDS92] David H. Bailey, Eric Barszcz, Leonardo Dagum, and Horst D. Simon. NAS Parallel Benchmark Results. Technical Report RNR-92-002, NASA Ames Research Center, Moffett Field, CA 94035, December 1992.

[BBN86] BBN Laboratories Inc. *Butterfly-TM Parallel Processor Overview*, 1986.

[BDQ86] J.-C. Bermond, C. Delorme, and J.-J. Quisquater. Strategies for Interconnection Networks: Some Methods from Graph Theory. *Journal of Parallel and Distributed Computing*, 3:433–449, 1986.

[Bel92] Gordon Bell. Ultracomputers: A Teraflop before its time. *Communications of the ACM*, 35(8):26–47, August 1992.

[BH86] Joshua E. Barnes and Piet Hut. A Hierarchical $O(n \log n)$ Force Calulation Algorithm. *Nature*, 324(4):446–449, 1986.

[BL84] Sandeep N. Bhatt and Frank Thomson Leighton. A Framework for Solving VLSI Graph Layout Problems. *Journal fo Computer and System Sciences*, 28(2):300–343, April 1984.

[Boo93] Robert Francis Boothe. *Evaluation of Multithreading and Caching in Large Shared Memory Parallel Computers.* PhD thesis, University of California, Berkeley, 1993. Computer Science Division, UCB//CSD–93–766.

[BP89] J.-C. Bermond and C. Peyrat. de Bruijn and Kautz Networks: A Competitor for the Hypercube? In F. André and J. P. Verjus, editors, *Hypercube and Distributed Computers*, pages 279–293. Elsevier Science Publishers B. V. (North Holland), 1989.

[CKP+92] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramoniam, and Thorsten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. Technical Report UCB/CSD 92/713, University of California, Berkeley, 1992.

[Cyp90] R. Cypher. Theoretical aspects of VLSI pin limitations. In *6th International MIT VLSI Conference*, pages 314–327, 1990.

[Dal86] William Dally. *A VLSI Architecture for concurrent data structures.* PhD thesis, California Institute of Technology, 1986. Tech Report: 5209:TR:86.

[Dal87] W. J. Dally. Wire-Efficient VLSI Multiprocessor Communication Networks. In *Advanced Research in VLSI*, pages 391–415, 1987.

[Dan88] Sivarama Dandamudi. *Hierarchical Interconnection Networks for Multicomputer Systems.* PhD thesis, University of Saskatchewan, Canada, 1988. Department of Computational Science 88-18.

[DKN93] William J. Dally, John S. Keen, and Michael D. Noakes. The J-Machine Architecture and Evaluation. In *IEEE Computer Society International Computer Conference, COMPCON*, pages 183–188, 1993.

[Fen81] T. Feng. A survey of interconnection networks. *IEEE Computer*, 14(12):12–27, Dec 1981.

[FIR93] Steve J. Frank, Henry Burkhardt III, and James Rothnie. The KSR1: Bridging the Gap Between Shared Memory and MPPSs. In *IEEE Computer Society International Computer Conference, COMPCON*, pages 285–294, 1993.

[GGK+83] A. Gottlieb, R. Grishman, C. Kruskal, K. McAuliffe, L. Rudolph, and M. Snir. The NYU Ultracomputer - designing an MIMD shared memory parallel computer. *IEEE Transactions on Computers*, C-32:175–189, February 1983.

[GR87] Leslie Greengard and Vladimir Rokhlin. A Fast Algorithm for Particle Simulation. *Journal of Computational Physics*, 73(325), 1987.

[Gre90] Ronald I. Greenberg. The Fat-Pyramid: A Robust Network for Parallel Computation. In *6th International MIT VLSI Conference*, pages 195–213, 1990.

[H+86] J. P. Hayes et al. Architecture of a hypercube supercomputer. In *Proceedings of the International Conference on Parallel Processing*, pages 653–660, 1986.

[Hil85] W. Daniel Hillis. *The Connection Machine.* The MIT Press, 1985.

[Hil90] Mark D. Hill. What is Scalability? *Computer Architecture News*, 18(4):18–21, December 1990.

[IAS87] George B. Adams III, Dharma P. Agarwal, and Howard Jay Siegel. A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks. *IEEE Computer*, 20(6):14–27, June 1987.

[IEE] IEEE. IEEE Std 1596-1992 (Scalable Coherent Interface). Currently under development.

[Int86] Intel Corp. *Intel iPSC System Overview*, January 1986.

[Joe90] Christopher Frank Joerg. Design and implementation of a packet switched routing chip. Master's thesis, MIT, December 1990.

[Ken92] Kendall Square Research Corporation. *Kendall Square Research Technical Summary*, 1992.

[KK79] P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, 1979.

[Kow85] Janusz S. Kowalik, editor. *Parallel MIMD computation : the HEP supercomputer and its applications.* The MIT Press, 1985.

[KS93] R. E. Kessler and J. L. Schwarzmeier. Cray T3D: A New Dimension for Cray Research. In *IEEE Computer Society International Computer Conference, COMPCON*, pages 176–182, 1993.

[KU91] Smaragda Konstantinidou and Eli Upfal. Experimental Comparison of Multistage Interconnection Networks. Technical report, IBM Research Division, Yorktown Heights, New York, 1991. Report No. RJ 8451 (76459).

[L+91] Daniel Lenoski et al. Overview and Status of the Stanford DASH Multiprocessor. In *International Symposium on Shared-Memory Multiprocessing*, pages 102–108, 1991.

[L+92] Charles E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. In *Proc. of the 1992 ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, 1992.

[Lee89] Gyungho Lee. A performance bound of multistage combining networks. *IEEE Transactions on Computers*, C-38(10):1387–1395, October 1989.

[Lei82] Charles E. Leiserson. *Area-Efficient VLSI Computation.* PhD thesis, Carnegie-Mellon University, 1982. Also available as the ACM Doctoral Dissertation Award 1982, MIT Press.

[Lei84] F. T. Leighton. New Lower Bound Techniques for VLSI. *Mathematical Systems Theory*, 17(1):47–70, April 1984.

[Lei85] Charles E. Leiserson. Fat trees: Universal Networks for Hardware-Efficient Supercomputing. In *Proceedings of the International Conference on Parallel Processing*, pages 393–402, 1985.

[LKK86] Gyungho Lee, Clyde P. Kruskal, and David J. Kuck. The effectiveness of combining in shared memory parallel computers in the presence of 'hot spots'. In *Proceedings of the International Conference on Parallel Processing*, pages 35–41, 1986.

[LLM90] Tom Leighton, Derek Lisinski, and Bruce Maggs. Empirical evaluation of randomly-wired multistage networks. In *International Conference on Computer Design*, 1990.

[LM89] Tom Leighton and Bruce Maggs. Expanders might be practical: fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 384–389, 1989.

[LMS92] T. Leighton, B. Maggs, and R. Sitaraman. On the Fault Tolerance of Some Popular Bounded-Degree Networks. Technical Report CS–TR–385–92, Department of Computer Science, Princeton University, Princeton, NJ, September 1992. Short version: Proceedings of the IEEE Annual Symposium on the Foundations of Computer Science, 1992, pp 542–552.

[LT88] Tom Lovett and Shreekant Thakkar. The Symmetry Multiprocessor System. In *Proceedings of the International Conference on Parallel Processing*, pages 303–310, 1988.

[McM92] R. J. McMillen. The Influence of Packaging Issues on the Design of the Massively Parallel NCR 3700 Computer. In *Workshop Report: Packaging, Interconnects and Optoelectornics for the Design of Parallel Computers Workshop*, 1992.

[MCS91] John M Mellor-Crummey and Michael L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, Feb 1991.

[Mey89] Ernest Meyer. Fine-pitch technology shrinks system size, boosts performance. *Computer Design*, pages 71–76, November 1989.

[NA91] Daniel Nussbaum and Anant Agarwal. Scalability of Parallel Machines. *Communications of the ACM*, 34(3):56–61, March 1991.

[ND90] Michael Noakes and William J. Dally. System Design of the J-Machine. In *6th International MIT VLSI Conference*, pages 179–194, 1990.

[Nef92] John A. Neff. Holographic Optical Interconnections in the 3-D Computer. In *Workshop Report: Packaging, Interconnects and Optoelectornics for the Design of Parallel Computers Workshop*, pages 61–65, 1992.

[Nic90] John R. Nickolls. The Design of MasPar MP-1: A Cost Effective Massively Parallel Computer. In *IEEE Computer Society International Computer Conference, COMPCON*, 1990.

[Nic92] John R. Nickolls. Interconnection Architecture and Packaging in Massively Parallel Computers. In *Workshop Report: Packaging, Interconnects and Optoelectornics for the Design of Parallel Computers Workshop*, pages 4–8, 1992.

[PBG+85] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe E. A. Melton, V. A. Norton, and J. Weiss. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In *Proceedings of the International Conference on Parallel Processing*, pages 764–771, 1985.

[PN85] G. F. Pfister and V. A. Norton. Hot-spot contention and combining in multistage interconnection networks. In *Proceedings of the International Conference on Parallel Processing*, pages 790–797, 1985.

[R+90] R. D. Rettberg et al. The Monarch Parallel Processor Hardware Design. *IEEE Computer*, pages 18–30, April 1990.

[Ran88] Abhiram G. Ranade. *Fluent Parallel Computation.* PhD thesis, Yale University, 1988. Department of Computer Science TR-663.

[RR90] M. T. Raghunath and Abhiram Ranade. A Simulation-Based Comparison of Interconnection Networks. In *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*, pages 98–103. IEEE, December 1990.

[RR93] M. T. Raghunath and Abhiram Ranade. Designing Interconnection Networks For Multi-level Packaging. In *Supercomputing '93, Portland, Oregon*, November 1993. To appear.

[SBN82] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples.* McGraw Hill International Book Company, 1982. Chapter 41.

[TG92] D. Z. Tsang and T. J. Goblick. Free Space Optical Interconnections for Parallel Computers. In *Workshop Report: Packaging, Interconnects and Optoelectornics for the Design of Parallel Computers Workshop*, pages 56–60, 1992.

[Tho80] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Carnegie-Mellon University, 1980.

[TMC91] TMC. CM5 Reference Manual. Technical report, Thinking Machines Inc, 1991.

[Ull84] Jeffrey D Ullman. *Computational aspects of VLSI.* Computer Science Press, 1984.

[Upf89] Eli Upfal. An O(log N) deterministic packet routing scheme. In *Proceedings of the ACM Annual Symposium on Theory of Computing*, pages 241–250, 1989.

[VR89] Anujan Varma and C. S. Raghavendra. Fault-Tolerant Routing in Multistage Interconnection Networks. *IEEE Transactions on Computers*, 38(3):385–393, March 1989.

[War92] Steve Ward. Toward Legoflops: A Scalable 3D Interconnect. In *Workshop Report: Packaging, Interconnects and Optoelectornics for the Design of Parallel Computers Workshop*, pages 9–14, 1992.

[WF80] C. Wu and T. Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-29:694–702, August 1980.

[YTL86] P. Yew, N. Tzeng, and D. H. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. In *Proceedings of the International Conference on Parallel Processing*, pages 51–58, 1986.