Customizing Interconnection Networks to Suit Packaging Hierarchies

M. T. Raghunath* Abhiram Ranade*
Computer Science Division,
University of California, Berkeley, CA 94720

February 1, 1993

Abstract

A central problem in building large scale parallel machines is the design of the interconnection network. Interconnection network design is largely constrained by packaging technology. We start with a generic set of packaging restrictions and evaluate different network organizations under a random traffic model. Our results indicate that customizing the network topology to the packaging constraints is useful. Some of the general principles that arise out of this study are: 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) Providing the processors with the ability to tolerate latencies (by using *multithreading*) is very useful in improving performance.

*This work is supported by the Air Force Office of Scientific Research (AFSC/JSEP) under contract F49620-90-C-0029 and National Science Foundation Grant Number CCR-9005448. M. T. Raghunath is also supported by a fellowship from IBM Corp. This research was also supported in part by National Science Foundation under Infrastructure Grant Number CDA8722788.

1 Introduction

In this paper, we address the problem of designing interconnection networks for large scale parallel computers. Interconnection network design is a central issue in building large scale general purpose parallel computers. 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 issues in designing a network range from the very high level: what topology should we use, to the most mundane: what should be the widths of the different channels. But each of these questions can make or break a network design. Each question is also more complex than it appears superficially: should we use hybrid network topologies (e.g. mesh at board level, and butterfly between boards)? Is it useful to have variable channel widths in different parts of the machine? Is it useful to duplicate (also called *dilate*) channels?

The answers to these questions will critically depend upon the communication behavior of the applications running on the parallel computer. For instance, if the application consists of multiplying dense matrices, it is conceivable that a two-dimensional grid will be an appropriate interconnection network. However, our goal is to design networks to support general purpose parallel programming, so our design cannot be unduly influenced by the characteristics of a single program. For modeling general applications, it is customary to assume random communication patterns. In particular, we assume that every time a processor accesses shared data, the location accessed is randomly distributed over the entire address space. Similar models have been used by most studies of network design[6, 11, 13].

The choice of the interconnection network will also depend upon the packaging technology available. In order to answer the above questions on network design, we need a model of packaging technology. Researchers have developed cost models to deal with the lowest levels of packaging technology such as VLSI [15], but these are not applicable to large parallel machines. Large parallel machines will typically employ several levels of packaging, e.g. racks, boards and VLSI chips. Modeling the cost of the hierarchy is difficult mainly because each level of the hierarchy has its own costs and constraints, e.g. standard size printed circuit boards might be limited to a pin-count of about 500, standard IC packages may have pin-counts up to 250, etc. Further, the characteristics of each level of the packaging hierarchy, or even the number of levels in the hierarchy may change as technology evolves.

However, it is clear that in the foreseeable future, some kind of packaging hierarchy will continue to exist. In addition, we can be certain that as we go higher up the hierarchy (e.g. between racks) the wires will connect larger modules and are likely to be longer than the wires at the lower levels (e.g. within a chip). It is also reasonable to assume that the cost per wire will be higher at the higher levels of the packaging hierarchy. Further, the wiring density available at the higher levels will be substantially lower, partly because of the associated cost, and partly because of technological limitations.

Since the costs at the higher levels of packaging are higher and restrictions are more stringent, we calculate the cost of a network based on the amount of resources used at the top of the hierarchy, and ignore the costs involving the lower levels. We shall assume that the machine consists of M *packaging modules* that are interconnected by wires at the top level of the hierarchy. We will assume that each module will hold N/M processors (for a total of N), the corresponding fraction of total system memory, as well as some communication hardware. It is important to point out that we use the term *packaging module* in an abstract sense. What a *packaging module* physically corresponds to will depend on the scale of the machine; it could be a board, a rack or even a cabinet consisting of multiple racks. We will develop a model of network cost that takes packaging resources at the top level into account. We will evaluate different networks based on this cost model. The central questions we wish to consider are:

1. Is there a generic model for the highest level of the packaging hierarchy? The details will obviously depend upon the precise technology, but can we formulate a model that reflects general trends?

2. Given our generic model, what is the best network (highest performance for a given cost) for interconnecting modules?

3. How does the intra-module network influence the communication performance between modules?

4. Is it useful to have different topologies for the inter-module network and the intra-module network? Is it useful to have different channel-widths?

1.1 Overview and Summary of Results

In this paper, we propose a methodology for designing networks given N, the number of processors, and M the number of modules at the highest level. Implicit in our methodology is a model of packaging cost which is discussed in section 2. This model allows us to prune the search space of possible designs for the inter-module network. We then consider alternatives for the intra-module network (section 3) which we evaluate using simulations. We assume a shared memory model of interprocessor communication (section 4), but we believe our observations apply to other models requiring global communication. Section 2.2 has a comparison of our work with other network design studies. Section 5 presents results of simulations for the case $N = 1024$ processors and $M = 32$ modules. Section 6 gives our conclusions, which may be summarized as follows:

1. We find that the networks that achieve the best performance have a hybrid structure. While all the networks we studied are variants of a Butterfly network, the best performance was obtained for topologies that differed in small but important ways from a Butterfly.

2. Making networks denser locally improves global communication performance. We consider two ways of increasing local density: wider channels and channel replication.

3. We consider a new model multithreaded processor execution that allows processors to tolerate a fixed number of outstanding memory accesses (or alternatively communication operations). We believe this model is a more realistic approximation of processor execution than the *open network model* [12] which essentially corresponds to infinite multithreading. For the cases we simulated, we found that even small levels of multithreading are sufficient to give good performance.

2 Inter-Module Network

As we mentioned earlier, relatively precise models are available only at the lowest packaging levels such as VLSI. However, there is general agreement among hardware designers as to what the main costs are at higher packaging levels. In this paper we focus on the following measures, which most hardware designers would believe are among the most significant ones (in order of importance)[a]:

1. *Bisection Width:* This is the minimum number of wires that need to be cut in order to divide the network into two equal parts.

2. *Module Pin-count:* This is the number of wires leaving or entering each module.

3. *Number of Wire Bundles:* The wires connecting modules can be grouped into bundles, such that wires in each bundle connect the same pair of modules. The total number of bundles in a network is another indication of the difficulty of building the network: it is easier to assemble a network of a given bisection width if the wires are organized into a small number of fat bundles; as opposed to a large number of skinny ones.

Based on these cost measures, we would like to find a network that has the least cost for a given performance. Since it is easily seen that the bisection width of the network is proportional to its performance (measured in number of messages delivered per cycle) to a first approximation, we consider the problem of finding a network which has the minimum module pin-count and minimum number of bundles per module of all possible networks.

Table 1 shows a comparison of standard and non-standard topologies (*n-hop* topologies, described in section 3.2) keeping the bisection width constant. Unfortunately, we cannot pick a clear winner – typically different topologies represent different tradeoffs between module pin-count and the number of bundles. A way to resolve this problem is to possibly weight these two costs appropriately, and then consider the combined cost measure. Unfortunately, this is very technology dependent, and can only be done approximately.

Substituting absolute numbers for B and M can give us a better insight. Consider for example that we are designing a 1024 processor machine. A

[a]This is by no means an exhaustive list of possible cost measures. For instance, a network with long wires is harder to engineer electrically than a network with short wires. Many of the more modern parallel computers such as the CM-5 use asynchronous differential signaling with several bits being in transit on the same pair of wires at the same time. This overcomes some of the limitations imposed by long wires. Also see the paper by Scott and Goodman[14].

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	$(4nB/M)\frac{\sqrt[n]{M}-1}{\sqrt[n]{M}}$	$n(\sqrt[n]{M}-1)$	$nM(\sqrt[n]{M}-1)/2$
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 1: Characteristics of different topologies used to interconnect M packaging modules

Inter-Module Topology	Bisection Width	Module Pin-count	Bundles Per Module	Total #Bundles
Complete graph	8192	992	31	496
2-hop Network	8192	1664	10	160
3-hop Network	8192	2048	7	112
Butterfly	8192	5120	4	64
CCC	8192	7680	3	48
Hypercube	8192	2560	5	80
3-dim Toroid	8192	2560	6	96
2-dim Toroid	8192	3072	4	64
Ring	8192	8192	2	32

Table 2: Characteristics of different topologies used to interconnect 32 packaging modules

reasonable value of B for such a machine is 8192, corresponding to 8 bits of remote memory bandwidth per processor per cycle. We believe a reasonable value for M to be 32. Obviously, our conclusions will be different for dramatically different values of M. We discuss the issue of choosing M at more length in section 2.1. Table 2 gives the comparison. Note that for $M = 32$, networks such as toroids or the 2 or 3-hop networks will not have the same number of modules in each dimension because 32 is not a perfect square or a perfect cube. The numbers in the table correspond to the nearest power of 2 partitions. Also the numbers corresponding to the Butterfly and CCC are approximate.

We believe that at the higher levels of the packaging hierarchy the number of pins per module is a stronger constraint than the total number of bundles[b]. It is in fact possible to show that the complete graph topology achieves the minimum pin-count for a given bisection. However, as will be seen from the table, the complete graph also uses the maximum number of bundles; which may not be supportable in some technologies. In such cases, we must use a topology that has fewer bundles, such as the 2-hop network. Therefore, we will choose the top two rows of the table for further investigation.

[b] In fact, some hardware designers feel that the pin constraint may be even stronger than the bisection width constraint.

2.1 Scalability

Complete graphs are considered not to be scalable as an interprocessor network. A network is said to be scalable if it can be built for large values of N, the number of processors. Our concern in this paper, however, is M the number of modules. 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 consists of modules that contain 10 processors each. We feel that the number of modules at the highest level will depend upon the technology, but not substantially on the number of processors.

2.2 Comparison to Previous work

Most of the previous comparative studies of networks deal with *uniform networks*, i.e. a single network topology is used to connect together processors rather than customizing the topology to suit the different levels of the hierarchy. Further, almost none of the previous work has considered the possibility of having communication channels of different widths at different levels of the hierarchy. As our results demonstrate, customizing network topologies and channel widths to packaging technology significantly improves performance.

Several of the existing and proposed parallel machines are based on hybrid networks that attempt to match the network to the packaging technology. For instance the Connection Machine CM-1 uses denser networks to connect the 16 bit-serial processors on a chip, while the entire machine can be considered a hypercube over these chips. The CEDAR system likewise organizes the machine into Omega network connected clusters, with each cluster consisting of several processors connected by a cross-bar network. Estimates of the communication performance of the CM family can be found in [5] and performance measurements of the CEDAR can be found in [2]. These measurements evaluate the performance of only single design point. In this paper we provide a comprehensive evaluation of a variety of design choices, albeit in an abstract setting.

The work closest to ours is that by Dandamudi [9]. He has evaluated the performance of a number of hierarchical networks using the number of bundles as a cost measure. His motivation and approach are however very different from ours. His work is driven by the goal of trading off system performance to reduce the cost of the highest level connections ("Inter-Module" connections as per our terminology). For this he considers different interconnection networks for the highest level and evaluates their cost performance trade-offs. On the other hand, as will become clear soon, the bulk of our study is concerned with how to design a lower level network ("intra-module" network) to efficiently exploit the maximum bandwidth provided by the inter-module interconnect.

Many researchers have examined the networks corresponding to the lower rows of table 1. Dally[7] has analyzed the class of multi-dimensional toroids using bisection width as the cost measure. Abraham and Padmanabhan [1] have analyzed the same class of networks (k-ary n-cubes) based on a constant pin-count restriction and Agarwal [3] has analyzed the same class of networks under three different restrictions: constant bisection, constant channel width and constant pin-count. There are also a variety of studies analyzing the performance of networks like meshes and hypercubes[8], multibutterflies[13] for random communication patterns. All of these studies primarily deal with *uniform networks*, where the topology is fixed before packaging is considered.

3 Intra-Module Networks

3.1 Inter-Module Topology: Complete graph

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. When M and N are powers of 2 and $M \leq \sqrt{N}$ the butterfly network has a partitioning where the interconnection between modules is a complete graph. Figure 1(a) shows a butterfly network connecting 16 processors to 16 memories. Figure 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. In this case, the implementation is simplified because each processor could use a single memory that is partitioned into private memory and shared memory. While the private memory would be accessed directly by the processor, the shared memory would be accessible to all the processors via the network. We represent this in figure 1(c) by drawing the processors and their associated memories adjacent to each other.

3.1.1 Basic butterfly network

The first network we consider is a butterfly connecting 1024 processors to 1024 memories; partitioned into 32 modules in the manner of figure 1(b).

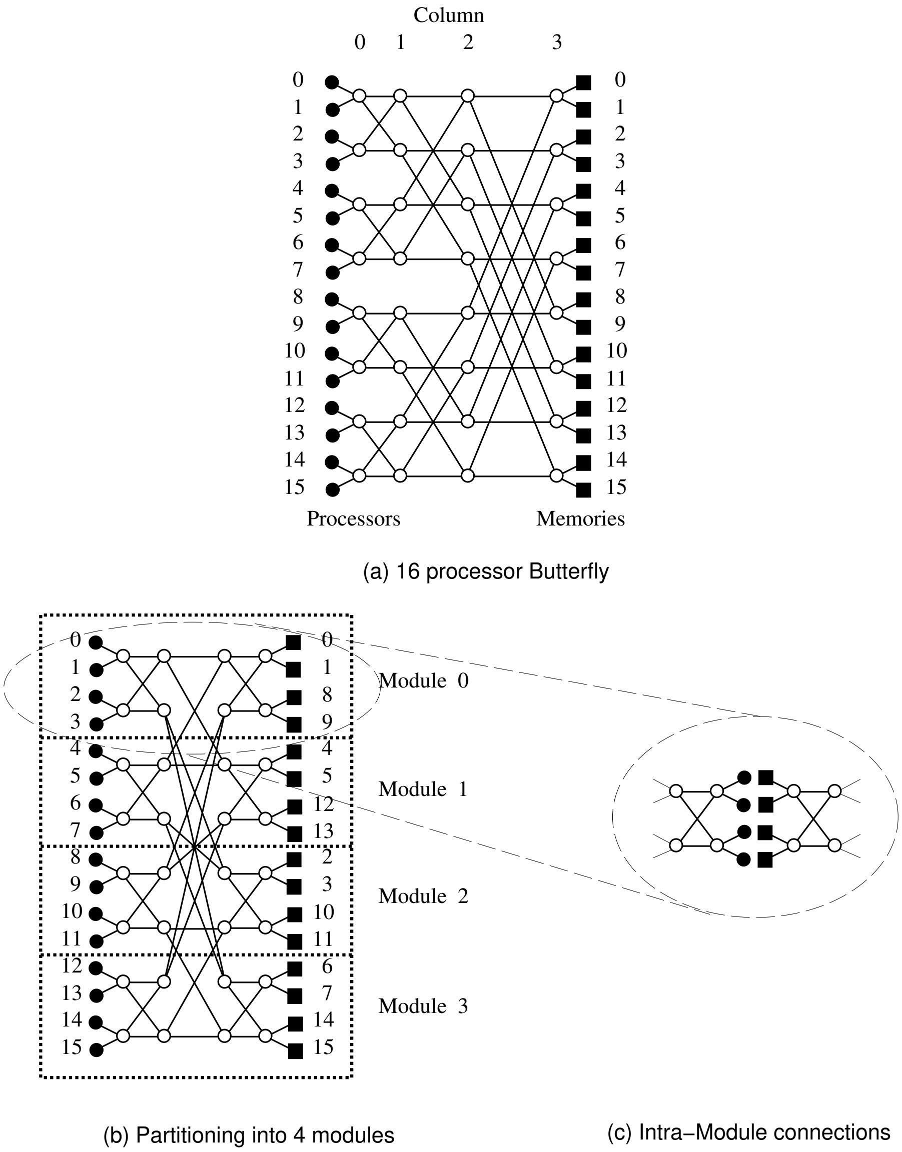
A bisection width of $B = 8192$ implies that pin-count per module is 992 and that each bundle has 32 wires. Since each bundle has 4 channels, each channel will be 8 bits wide. Since the inter-module channels are 8 bits wide, we get a uniform network if we make the intra-module channels also 8 bits wide. We will 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.

3.1.2 Widening channels

We next consider networks *Butterfly.16.8* and *Butterfly.32.8* both of which have the same topology as *Butterfly.8.8*, but they have 16 and 32 bit wide intra-module channels respectively.

3.1.3 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



(a) 16 processor Butterfly

(b) Partitioning into 4 modules

(c) Intra−Module connections

Figure 1: Butterfly Network and its partition

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 uses the same number of pins 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 logic to prevent multiple accesses from being made simultaneously.

3.1.4 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.

3.2 Inter-Module Topology: 2-Hop Network

A 2-hop network is a product graph of two complete graphs[c]. The nodes in the graph can be thought of labeled using a pair of numbers (i, j) where each number ranges from $0..\sqrt{M}$. There are complete graphs between all nodes of the form $(*, j)\forall j$, and also between all nodes of the form $(i, *)\forall i$. We call this a 2-hop network because every pair of nodes is guaranteed to be connected by a shortest path of length at most 2. In general a n-hop network is a product of n complete graphs, each of size $\sqrt[n]{M}$. A complete graph interconnect is a 1-hop network in this sense. When the number of hops is $\log M$[d], the resulting topology is a hypercube.

Interestingly, it turns out that the starting point for generating a 2-hop inter-module network is also a butterfly. In fact, any butterfly network of size

[c]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 $G_1 * G_2 = (V, E)$ where $V = V_1 \times V_2$ and $E = \{((u_1, u_2), (v_1, v_2)) | ((u_1 = v_1) \wedge (u_2, v_2) \in E_2) \vee ((u_2 = v_2) \wedge (u_1, v_1) \in E_1)\}$

[d]All logarithms are to base 2

$N = 2^{(n+1)x}$ 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 will have $(n+1)x$ 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 will 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 a n-hop network when the number of modules M is not of the form 2^{nx}, but the numbers in table 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 differ slightly from that shown in the table.

In our present case, we need to build a 2-hop network on 32 Modules. Since 32 is not a perfect square, we use the topology $K_4 * K_8$, where K_x denotes a complete graph on x nodes and $*$ denotes graph product. Figure 2 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 each node of the K_4s.

As in the one-hop case, each edge in the figure will correspond to 2 pairs of channels, one in each direction. Of each pair of links, one will carry processor to memory traffic, and the other memory to processor traffic. In order to achieve a bisection width of 8192, 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).

3.2.1 Basic 2-hop network

The intra-module network of the basic 2-hop network is shown in figure 3. The processors are connected to the nodes on the left. 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 2 level tree as shown on the top node. The memories are also connected in a similar fashion. Processors

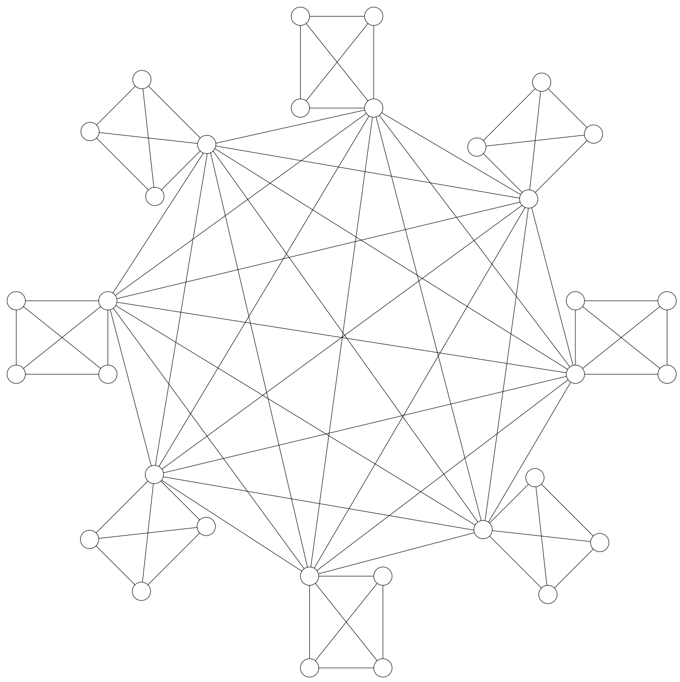


Figure 2: Inter-Module 2-hop Network (Only one of the 4 complete graphs on 8 nodes is shown)

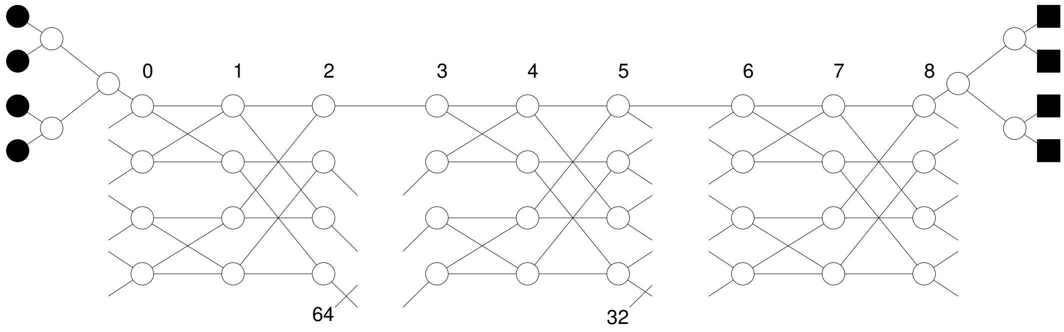


Figure 3: Intra-Module Network: Basic 2 hop network

and memories are paired as in the earlier network topologies, but this is not illustrated pictorially.

The channels between the column 2 and 3 correspond to K_4. They connect between modules that differ 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 *2hop.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.

3.2.2 Widening channels and dilation

Similar to the networks defined for the complete graph inter-module topology, we define the network *2hop.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 *2hop.32-2dil.64-32* with a degree 2 dilation of the intra-module network.

3.3 Routing Algorithms

Our routing algorithms use cut-through routing [10], with adequate buffer space in each routing node. Buffer space is allocated on a per-message basis. We 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. When a routing node's outgoing channels are wider than its incoming channels, the node cannot send messages in a pipelined fashion. We assume that these nodes wait for the entire message to come in before sending out any flits. For brevity, we omit a detailed description of the routing nodes.

For the cross-bar networks discussed earlier, we treat the entire cross-bar as a single routing node that imposes a single cycle pipeline delay. These assumptions are overly optimistic, but we believe that they are justified since we study the cross-bar networks only to establish an upper bound on performance.

4 Traffic Model

We base our evaluation of network performance on a shared-memory model. The memory accesses made by a processor are assumed to be distributed randomly over the entire shared address space. Each access is sent over the network to the memory where the data resides, the access is performed and returned to the processor. For simplicity, we assume that the both the access requests and the responses are 128 bits long, inclusive of the data, memory/network addressing information, and control information. We consider two models:

Open-network model[12]: In each cycle of execution, each processor generates a message with a fixed probability of λ, independent of other processors and previous cycles of execution. The interval between accesses is geometrically distributed with mean $1/\lambda$.

Multithreading work-load model: The preceding model is unrealistic in that it allows each processor to have an unbounded number of outstanding requests. Instead, in the multithreading work-load model, each processor runs a fixed number of threads, each of which can have at most one outstanding request at any time. When a thread issues an access the processor switches to the next thread in round-robin order. If the previous access of the thread switched into the processor has not completed, the processor stalls until the network returns the access. We assume a single cycle thread-switch time, in the manner of [4].

We consider two inter-access interval distributions for the multithreading work-load: geometric and periodic. In the geometric model the inter-access times are geometrically distributed similar to the open-network model. In the periodic model, the inter-access interval is a constant. If the processor is executing a tight inner loop, the interval between remote accesses is likely to be a constant. The periodic model attempts to capture this effect. In reality the distribution of accesses is likely to be somewhere in between. Note that in both the periodic and geometric models, the accesses are directed to random addresses and therefore will follow random paths in the network.

5 Simulation Results

5.1 Open-network

Figure 4 shows the results for the butterfly 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. The curves extend along the x-axis to the point until which the corresponding generation rate yielded a stable latency.

Figure 5 shows the results for the cross-bar networks. The graphs corresponding to *Butterfly.8.8*, *Butterfly.8-2dil.8* and *Butterfly.32.8* are included for comparison.

Figure 6 shows the results for the networks with a 2-hop network between modules. The graphs corresponding to *Butterfly.8.8*, and *Butterfly.32.8* are included to establish the relative performances.

5.2 Multithreading work-load

The performance metric under this work-load model is the average processor utilization, as a function of number of threads, and 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. This corresponds to a mean access rate of one every 16 cycles, since each access is 128 bits long. We considered 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, trying to run the network at 100% load does not pose a problem. The processors 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 50% load. In this case, since the network is lightly loaded, the processors should be able to achieve almost 100% utilization if they can successfully hide the network latency using multithreading.

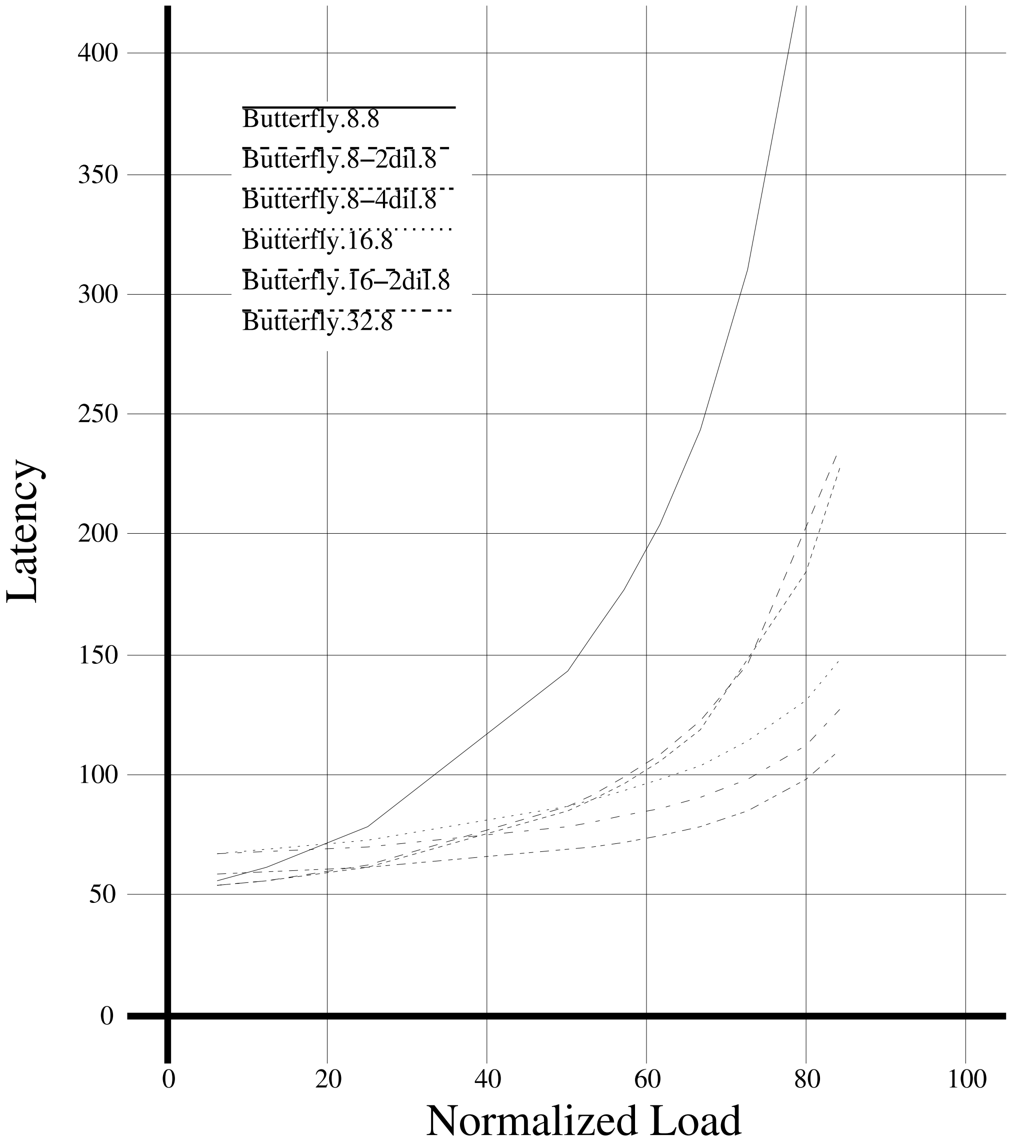


Figure 4: Open-Network Model; Butterfly Networks

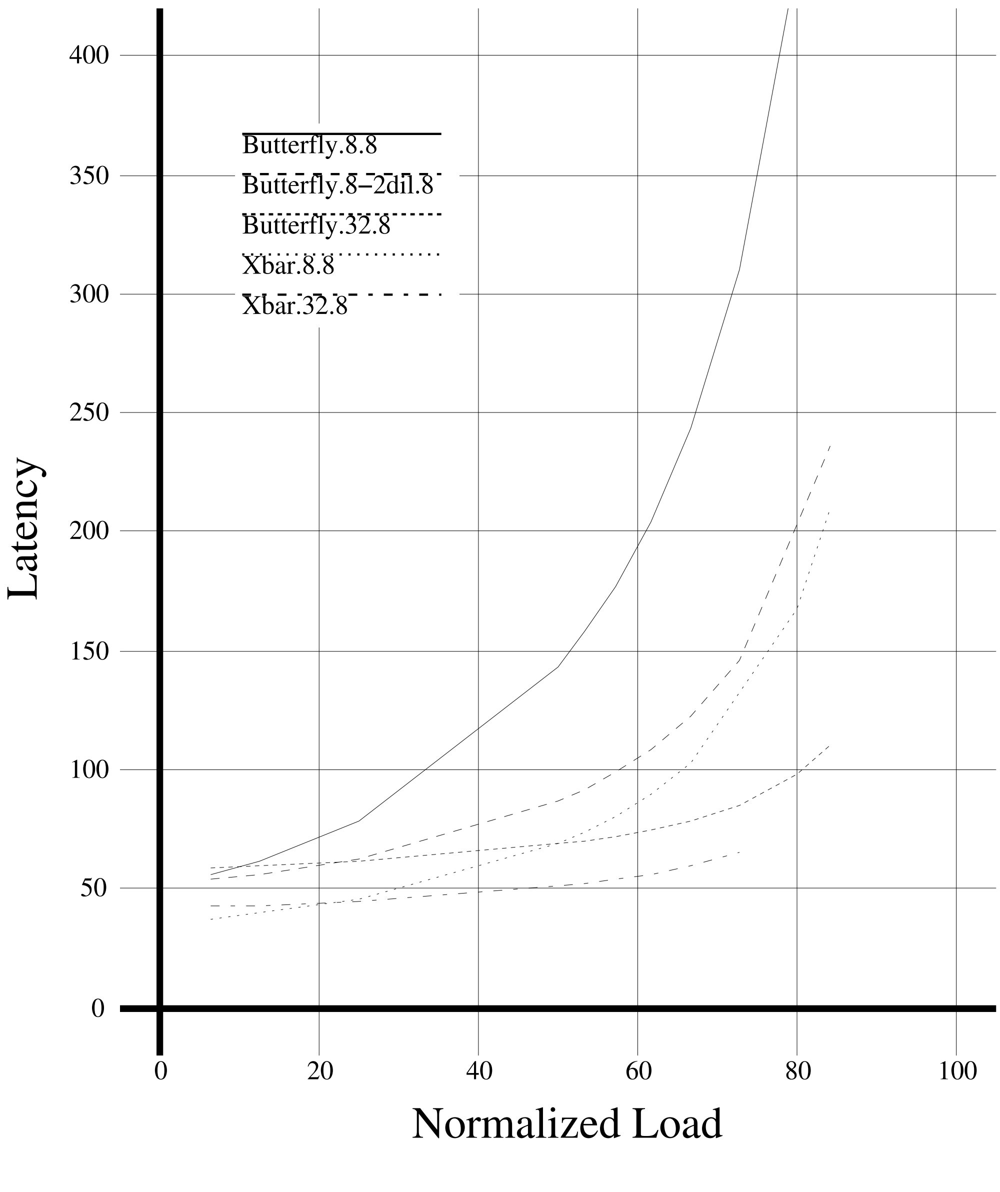


Figure 5: Open-Network Model; Cross-bar networks

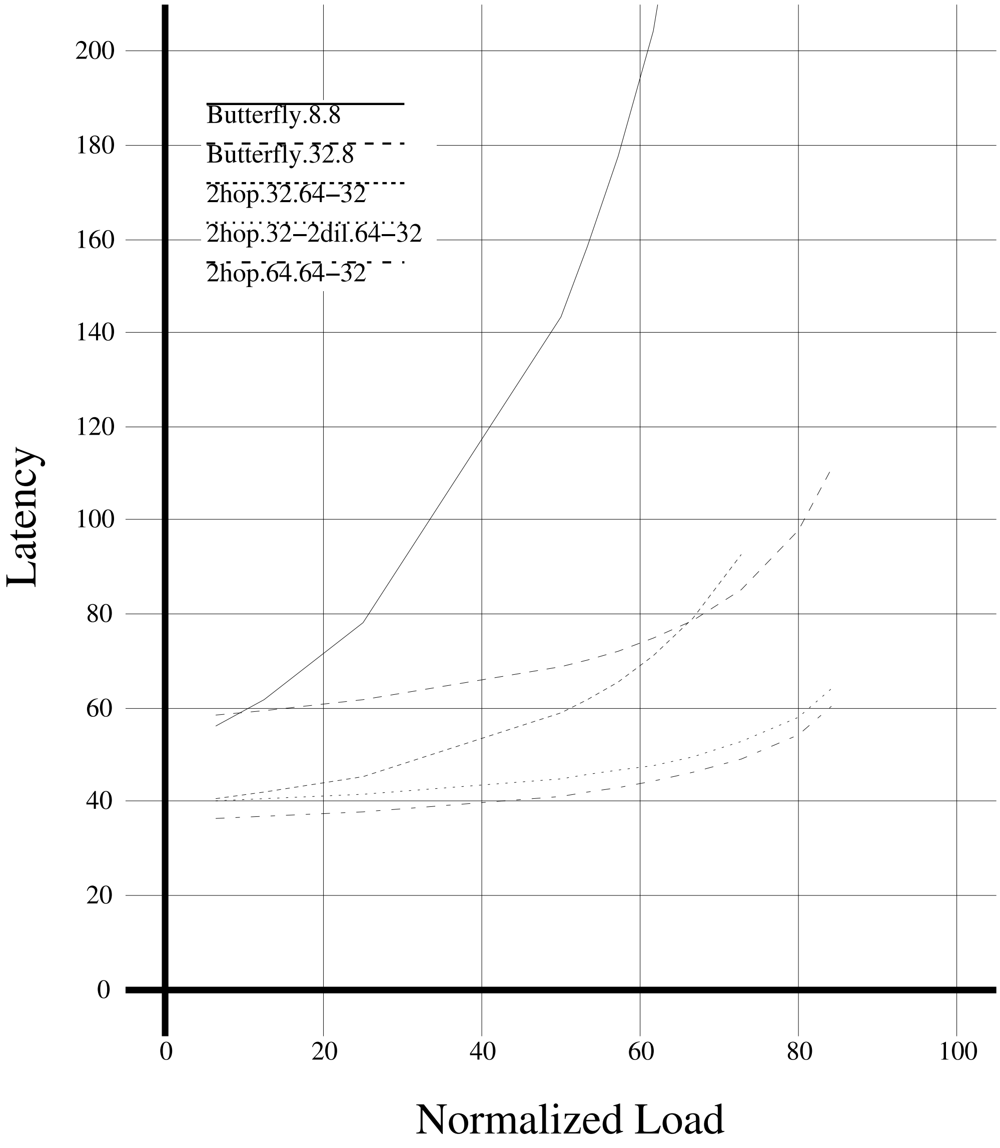


Figure 6: Open-Network Model; 2-hop networks

Figure 7 shows the processor utilizations for the different networks when the mean access interval is 16 cycles. The graphs shows four lines corresponding to multithreading factors of 1, 2, 4, 8 and 16. Figure 7(a) shows processor utilizations under the geometric model and figure 7(b) corresponds to the periodic access model.

Figure 8 shows similar curves for a mean access interval is 32 cycles.

5.3 Discussion of results

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 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 of the absence of variance in the inter-access times.

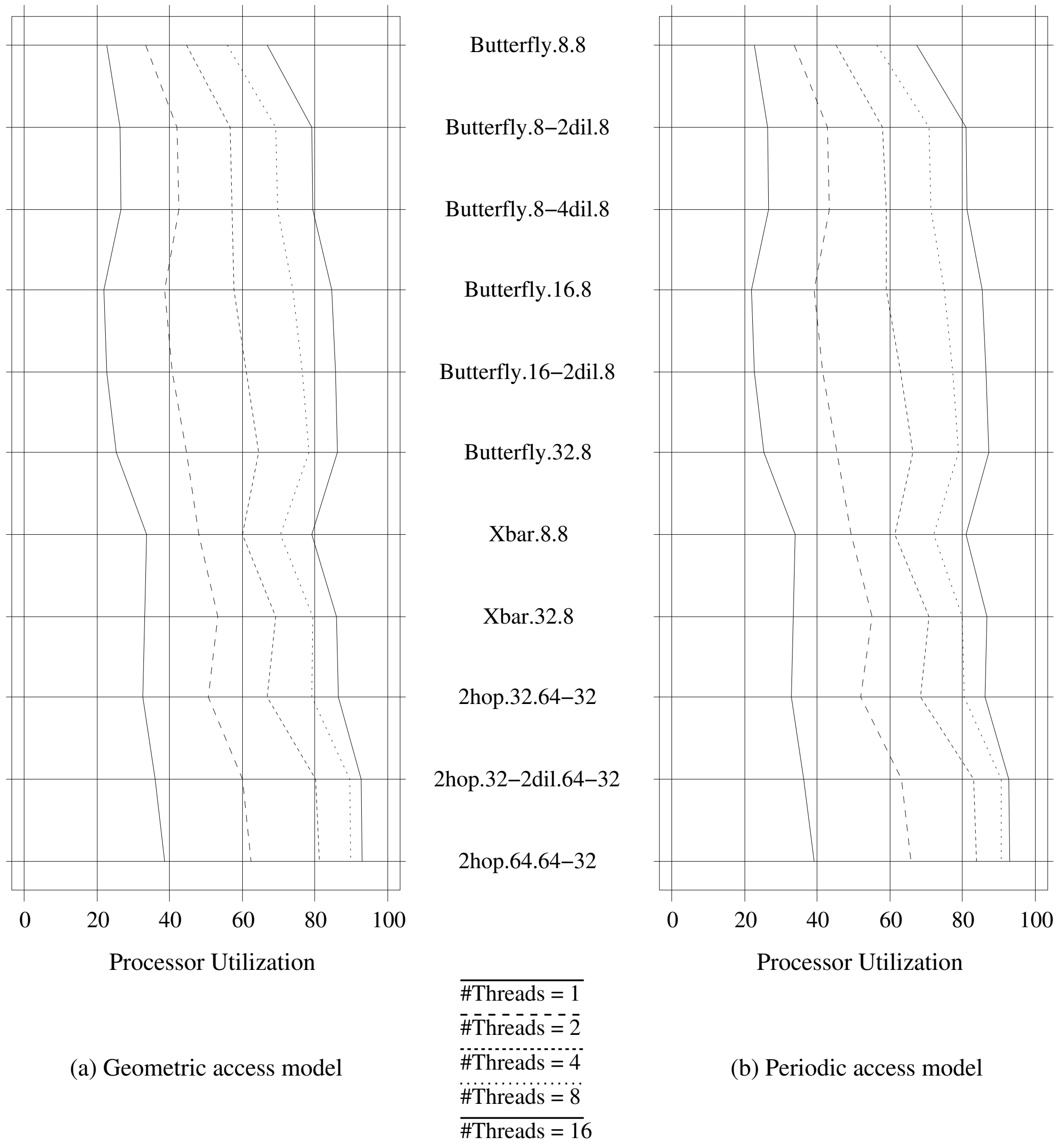
Dilation versus wider channels

The data corresponding to *Butterfly.8.8*, *Butterfly.8-2dil.8* and *Butterfly.8-4dil.8* in figures 4 and 7 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. Thirdly it increases message latency because of the non-uniformity in channel widths[e]. 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

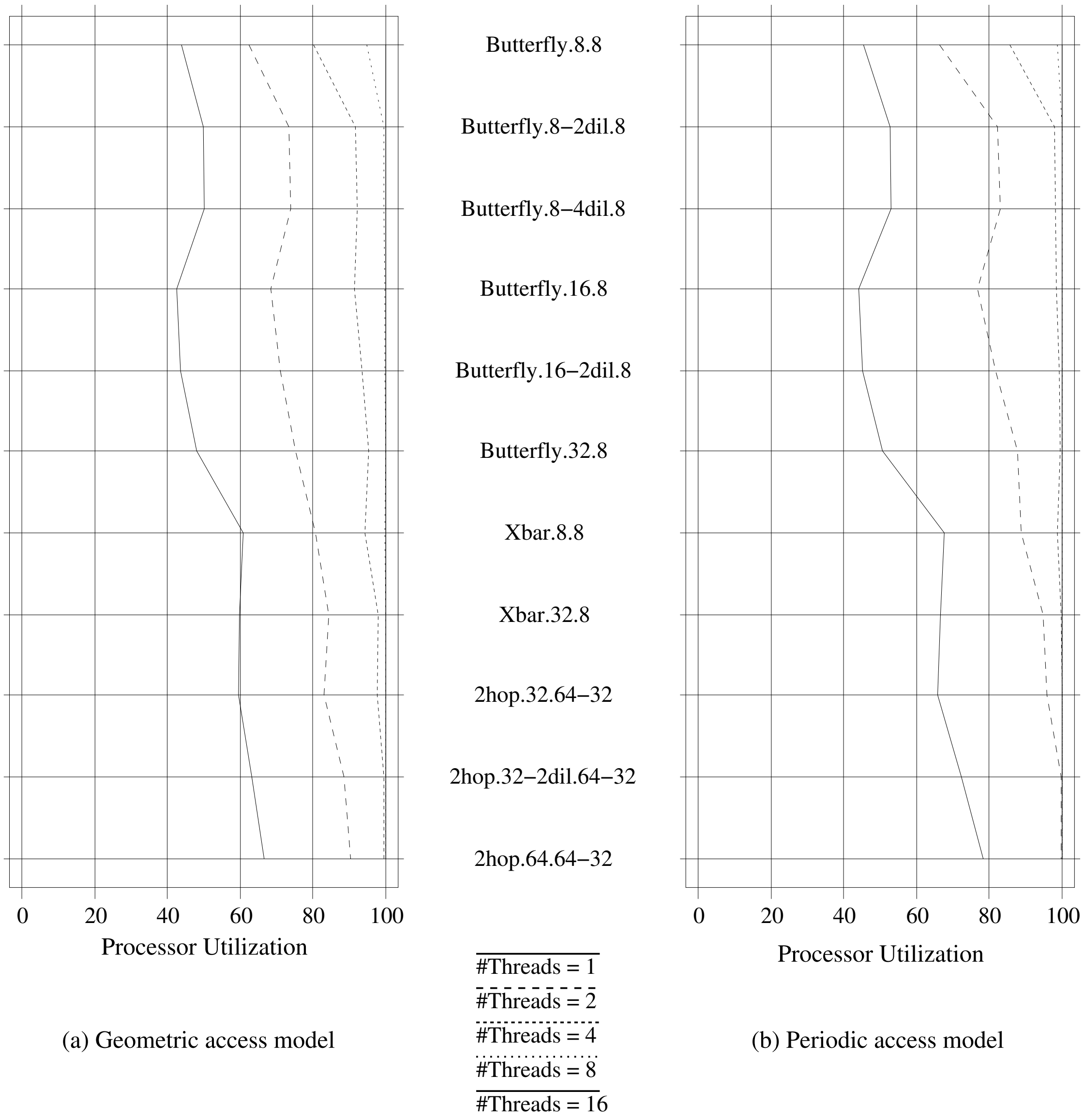
[e]When a message transits through a node whose output channel is wider than its input channel, the message suffers a larger delay.



(a) Geometric access model

(b) Periodic access model

Figure 7: Processor utilization, mean access interval = 16 cycles



(a) Geometric access model

(b) Periodic access model

Figure 8: Processor utilization, mean access interval = 32 cycles

worse than *Butterfly.8.8*. This is because of the third effect mentioned above. 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.

Cross-bar

It is useful to examine the results for the intra-module cross-bar network, although it may not be feasible to build it, because the cross-bar completely eliminates the possible contention for links in the intra-module network. 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 5 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. When we look at figure 7 we do not see any appreciable difference between the two networks indicating that the effect of this difference in latency is small. These results indicate that the 2-way dilated network is successful in eliminating almost all the contention for the intra-module channels.

The open-network graphs for *Xbar.32.8* and *Butterfly.32.8* are also parallel, but the difference between them is greater. This difference in latency shows up as increased processor utilization for the cross-bar when the number of threads is small. However, when the number of threads increases, this constant difference in latency does not have any impact on processor utilization.

2-hop Networks

The results for the 2-hop networks indicate similar relative behavior as the *Butterfly.*.** networks; dilation improves the performance over the base network, and widening channels is even better. However, the difference between dilation and wider channels is not as pronounced since the network initially has 32-bit wide memory channels. Increasing the channel width from 32 to 64 leads to a smaller marginal increase in performance.

Since the 2-hop and complete graph networks use different pin-counts and number of bundles, it is not really fair to compare them. However, since both networks are based on the butterfly topology, the results indicate that, for a given bandwidth, it is better to build a smaller butterfly with wider channels and load each input of the butterfly with more processors. When there are fewer channels that are wider, it is easier to balance the load on them and achieve better utilization.

On the other hand, if we compare the two networks based solely on pin-count rather than bisection width, we should consider a complete graph network with double the pin-count, since a 2-hop network uses approximately twice the number of pins(table 1). In this case, the complete graph networks would have twice the bisection bandwidth. Therefore, it is reasonable to compare the performance of the 2-hop networks at 100% load with that of the complete graph networks at 50% load. We can clearly see that the complete graph networks out-perform the 2-hop networks because the complete graph optimal when pin-count is the only limitation.

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, the processor utilizations improve, but the marginal increase per added thread becomes smaller. This is 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 8) 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 successfully hide the network latency. For the networks we evaluated we can see that about 4 threads can achieve this. 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 only grow as $\log N$ since the network latencies will grow as $\log N$.

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.

6 Conclusions

We considered interconnection network design from a packaging point of view and developed a cost model for the highest level of the packaging hierarchy. We compared the performance of different networks of the same cost using simulation. Our results show that there is significant performance improvement possible by designing networks in a hierarchical fashion, taking advantage of the higher wiring densities available at the lower levels of the packaging hierarchy.

We observed that increasing the connections local to a module by using either dilation or wider channels improved performance. It was better to increase the channel widths rather than use dilation, especially if our processors were capable of a high level of multithreading. With 8 threads per processor (geometric distribution of access intervals), we observed that a increasing the intra-module channel width from 8 to 32 improved the processor utilization of the butterfly network from around 55% to almost 80%. Also the performance of the network with 32 bit wide intra-module channels came very close to that of the best possible cross-bar network.

All our results indicate that the performance improvement achieved by multithreading the processors is quite significant. Processor utilizations increased by about a factor of 2 with 4 threads and almost 3 with 8 threads when compared to the utilization with a single thread. Designing processors capable of running multiple threads and rapidly switching between threads is therefore useful for large scale parallel computers.

We also observed that if we want to build butterfly networks with a fixed bisection width, it is better to build smaller butterflies with wider channels. Each input of the butterfly would service multiple processors, since the smaller butterfly will have fewer inputs. Since the smaller butterfly has wider channels, it is easier to balance the load on these channels and get an better effective throughput.

References

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

[2] S. G. Abraham and E. S. Davidson. A Communication Model for Optimizing Hierarchical Multiprocessor systems. In *Proceedings of the International Conference on Parallel Processing*, pages 467–474, 1986.

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

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

[5] *Connection Machine Model CM-2 Technical Summary*, 1987. Technical Report No: TR HA87-4.

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

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

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

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

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

[11] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.

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

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

[14] Steven L. Scott and James R. Goodman. The Impact of Pipelined Channels on k-ary n-cube Networks. To appear in IEEE Transactions on Parallel and Distributed Systems.

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