VLSI Design
1995, Vol. 2, No. 4, pp. 375–388
Reprints available directly from the publisher
Photocopying permitted by license only

Designing Interconnection Networks for Multi-level Packaging

M.T. RAGHUNATH* and ABHIRAM RANADE*
Computer Science Division, University of California at Berkeley, Berkeley, CA 94720

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.

Key Words: *Parallel processing, interprocessor communication, network topology, packaging hierarchy*

1 INTRODUCTION

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 interconnection network critically depends 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 many studies of network design [9, 16, 18].

The choice of the interconnection network also depends upon the packaging technology available. In order to evaluate different network design alternatives, we need a model of packaging technology. A well explored packaging technology is VLSI, and it is possible to evaluate different networks layed out on a VLSI chip using Thompson's grid model [26]. The principal cost measure in this model is the layout area. It has been shown that for fixed layout area, most of the common networks such as hypercubes, multidimensional grids, butterflies, etc., (with channel widths adjusted to equalize area requirements) have the same bandwidth for delivering messages with random destinations. Dally [9] argues that lower dimensional networks have better latencies.

The VLSI model is not adequate for large ma-

*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.

chines, since large parallel machines typically employ several levels of packaging, e.g. racks, boards and VLSI chips. To compare different network designs, we need a model that takes packaging hierarchies into account. Developing such a model is difficult because 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 addition to the costs associated with these parameters, there may also be technology specific constraints, e.g. standard printed circuit boards may be limited to a pin-count of 500, standard IC packages may have pin-counts of up to 250 etc. Further, advances in technology can significantly change the cost characteristics of each level of the hierarchy, or even the number of levels.

We would like to identify interconnection networks that are suitable for multilevel packaging technologies. Although the precise characteristics of existing or emerging packaging technologies are hard to model, a common characteristic of most technologies is a progressive increase in costs and decrease in capacities as we go up the packaging hierarchy. For instance, it is clear that as we proceed up the hierarchy, wires usually get longer with larger inter-wire spacings. Metal wires on VLSI chips may be a few millimeters long and spaced about 1 micron apart, whereas wires connecting different boards may be as long as 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. Further the rate at which data can be transferred on a wire usually decreases as we go up the hierarchy. In other words, packaging hierarchies exhibit the property of *packaging locality,* i.e. the connections within any unit of packaging are cheaper, denser, and faster than connections at higher levels. This in turn means that design of the interconnection network is more constrained by higher level restrictions than those at lower levels. This suggests that interconnection networks should be designed level-by-level, highest level first.

In this paper we consider a generic packaging technology with two levels, in which the processors in the parallel computer are organized into a number of *modules*. A larger number of levels might be more accurate (considered in [22]), but we feel that significant insights can be obtained with just two levels. We define a cost model for networks built using our two level technology, and then go on to define and compare a number of different interconnection networks. We compare the performance of different networks using a random traffic model (section 3). First we fix the high level (inter-module) network. As will be seen, this can be done analytically (section 4). Next, we consider possible lower level (intra-module) networks (section 5), and these, we compare using simulation. Our simulations are based on a shared-memory model, but we believe that our observations also apply to other models requiring global communication.

Our results show that the networks used to connect modules must be substantially different from the ones considered customarily, e.g. multidimensional grids, hypercubes, butterflies, etc. We identify a new family of network topologies called n-hop networks which are important for the inter-module network. As it turns out, these topologies are also important in the case of three-level packaging hierarchies which we consider elsewhere [22]. As far as the overall network is considered, we find that a hybrid network (consisting of different networks at different levels, or having different channel widths at different levels) is likely to give better performance for fixed cost. Hybrid networks have so far received little attention in the literature.

Section 4.2 has a comparison of our work with other network design studies. Section 6 presents results of simulations for the case $N = 1024$ processors and $M = 32$ modules. Section 7 gives our conclusions.

2 A TWO-LEVEL MODEL OF PACKAGING

In our model, a parallel computer consists of M *packaging modules*. Each module holds 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. Further, a module itself will typically have a hierarchical structure, which we ignore for simplicity.

As mentioned earlier, it is difficult to model costs precisely, except in VLSI, where the layout area is a widely accepted metric. Informally, however, there is general agreement among hardware designers as to what the main costs are at higher packaging levels. In this paper we consider the cost to be a function

of the following metrics, which most hardware designers would believe are among the most significant ones[a]:

1. *Module Pin-count:* This is the number of wires leaving or entering each module.
2. *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 with a given number of wirers if the wires are organized into a small number of fat bundles; as opposed to a large number of skinny ones.
3. *Bisection Width:* This is the minimum number of wires that need to be cut in order to divide the network into two equal parts.

We may consider using a cost function that is a linear combination of the above metrics, where the relative weights of the metrics are determined according to the precise characteristics of the underlying technology. However, in many cases, the cost function may be strongly non-linear; at the extreme, there may even be hard constraints which cannot be overcome however much you are willing to pay, e.g. it may just be impossible, given a technology, to have a module pin-count of 2000. The constraints and non-linearities may make it difficult to obtain an accurate estimate of the dollar cost of a particular design. We defer further discussion of cost estimation to section 4.

In principle, these cost metrics could be used for the network at each level. In this paper we mainly focus on the cost of the inter-module network. For the intra-module network, we consider several candidates with widely varying costs. However, we feel that many of these candidates will be interesting since we expect the cost of the intra-module network to be dominated by the inter-module cost.

3 TRAFFIC MODEL

We evaluate networks using a traffic model in which each processor sends messages to randomly chosen destinations. For simplicity, we assume that the communication primitives are based on a shared-memory model, but we believe that our results apply qualitatively to message passing models as well, so long as message destinations are random. We assume that in each cycle, a processor either executes a local operation or issues a remote access to a random location. The average interval between remote accesses is a measure of the data locality in the computation; longer inter-access intervals imply greater locality. We also assume that accesses made by different processors are independent of each other. Several researchers have used similar models to evaluate network performance [1, 3, 9, 11, 15, 16, 18, 24].

In general, the distribution of accesses made by a processor is highly dependent on the application. While some applications, such as those based on dense matrix computations, may have extremely regular communication traffic, there is a large class of applications for which the communication traffic tends to be highly irregular and dynamically varying with time. Sparse matrix computations and graph algorithms are examples of such applications. 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 [6] or the $O(n)$ Greengard and Rokhlin algorithm [13] 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, the communication traffic involves every processor communicating equally often with every other processor [8]. In addition, many of the recently developed models of parallel programming also assume that each processor can communicate with all processors with equal ease [4, 8].

Access non-uniformities and *hot-spots* due to repeated access to synchronization variables were topics of active research in the recent past. Pfister and Norton [21] demonstrated that buffered networks were susceptible to *tree-saturation* in the presence of hot-spot traffic. Solutions to the hot-spot problem include hardware message-combining [12, 20, 23] and

[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 modern parallel computers 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 [25].

software techniques such as the ones proposed in [27] and [19]. We assume that hot-spots are prevented by using some software mechanism, and do not deal with them explicitly in this paper.

Additional details regarding our evaluation methodology and assumptions made for the purposes of simulation, are provided in section 6.

4 DESIGNING THE INTER-MODULE NETWORK

We first consider the design of the network at the top level of the hierarchy, namely the inter-module network. We would like to compare several networks in terms of both cost and performance, identify promising ones, and rule out those that are not competitive. We enumerated three cost metrics in section 2: bisection width, pin-count, and number of bundles, but we also observed that it was difficult to combine the three metrics to yield a dollar cost. Nevertheless, we can compare the costs of different networks by comparing the three metrics separately, i.e., we can think of cost as a three dimensional quantity. If we can show that all the three dimensions of network N_1 are smaller than the corresponding three dimensions of N_2, then we can conclude that N_1 has lower cost than N_2. If N_1 also has an equal or greater performance than N_2, we can clearly rule out N_2. In addition, let us say that we can argue that no network can have greater performance than N_1, unless it has a greater cost than N_1 in *at least* one of the three dimensions. Then we would consider N_1 to be a *promising* network. We would like to explore the three dimensional design space and identify such *promising* networks and study them.

There is a simple observation that enables us to reduce the size of the design space to two dimensions. Suppose network N_1 has a greater bisection width than N_2. Then, with reasonable confidence we can say that the performance of N_1 will be greater than the performance of N_2 under random traffic. Under random traffic, half the traffic from each processor must cross the bisection of the machine. The rate at which this traffic can be delivered is directly proportional to the bisection width. Therefore, in order to determine which network has a lower cost for a fixed performance, it is sufficient to compare networks with the same bisection width.

Table 1 shows a comparison of standard and non-standard topologies (*n-hop* topologies, described in section 5.2) keeping the bisection width constant[b]. The entries in the table are derived as follows: First the number of bundles per module is the same as the node degree of the graph topology. For example, the number of bundles per module in a complete graph topology is $M - 1$. We then compute the number of bundles (edges) that cross the bisection, which is $M^2/4$ for a complete graph. Since the bisection width is B, each bundle must correspond to $4B/M^2$ wires. The pin-count per module is the product of the number of bundles per module and the number of wires per bundle, which is $4B(M - 1)/M^2$ for a complete graph.

Unfortunately, Table 1 does not show a clear winner—the different topologies represent different tradeoffs between module pin-count and the number of bundles. 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 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 4.1. 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 for these networks correspond to the nearest power of 2 partitions.

As we go down the table we see that the pin-count increases while the number of bundles decreases. Only in some cases, we can rule out some topologies. For example, when we compare the butterfly and the 2-dimensional toroid, we see that both have the same number of bundles, but the toroid has a lower pin-count. Therefore, in this case, we can rule out the butterfly. But, asymptotically, the butterfly is better than the toroid, and for larger values of M, the butterfly will have a lower pin-count than the toroid. In the majority of the cases, however, we cannot demonstrate such a domination in terms of both cost metrics. We need to know the relative weights of the two cost metrics to determine which network has lower cost. But the weights are highly technology dependent. We are interested in obtaining qualitative results that are applicable over a wide range of technologies.

We believe that at the higher levels of the pack-

[b] All logarithms are to base 2.

TABLE I
Characteristics of Topologies Used to Connect M Packaging Modules

Inter-Module Topology	Bisection Width	Module Pin-count	Bundles Per Module	Total # Bundles
Complete graph	B	$(4B/M)(M-1)/M$	$M-1$	$M(M-1)/2$
2-hop Network	B	$(8B/M)(\sqrt{M}-1)/\sqrt{M}$	$2(\sqrt{M}-1)$	$M(\sqrt{M}-1)$
3-hop Network	B	$(12B/M)(\sqrt[3]{M}-1)/\sqrt[3]{M}$	$3(\sqrt[3]{M}-1)$	$3M(\sqrt[3]{M}-1)/2$
n-hop Network	B	$(4nB/M)(\sqrt[n]{M}-1)/\sqrt[n]{M}$	$n(\sqrt[n]{M}-1)$	$nM(\sqrt[n]{M}-1)/2$
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

aging hierarchy the number of pins per module has a greater effect on cost than the total number of bundles. Therefore, we consider the top rows of the table for further investigation. We first consider the complete graph since it achieves the minimum pin-count for a given bisection. However, as the table indicates, the complete graph also uses the maximum number of bundles. Some technologies may have constraints which prohibit such a large number of bundles. In such cases, we must use a topology that has fewer bundles, therefore, we also consider the 2-hop network.

4.1 Scalability

It is somewhat surprising that we chose the complete graph as a promising candidate; complete graphs are considered not to be scalable as an interprocessor network. The rationale for this belief, as we discussed earlier, is that they have very large number $O(M^2)$ of bundles. 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 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. We feel that the number of modules, M, will depend upon the technology, but not substantially on the number N of processors.

4.2 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 CM-1 uses denser networks to connect

TABLE II
Characteristics of Different Topologies Used to Interconnect 32 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

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 [7] and performance measurements of the CEDAR can be found in [2]. These measurements evaluate the performance of only a 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 [10]. He 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 [9] 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 multibutterflies [18] and other many other networks for random communication patterns. All of these studies primarily deal with *uniform networks,* where the topology is fixed before packaging is considered.

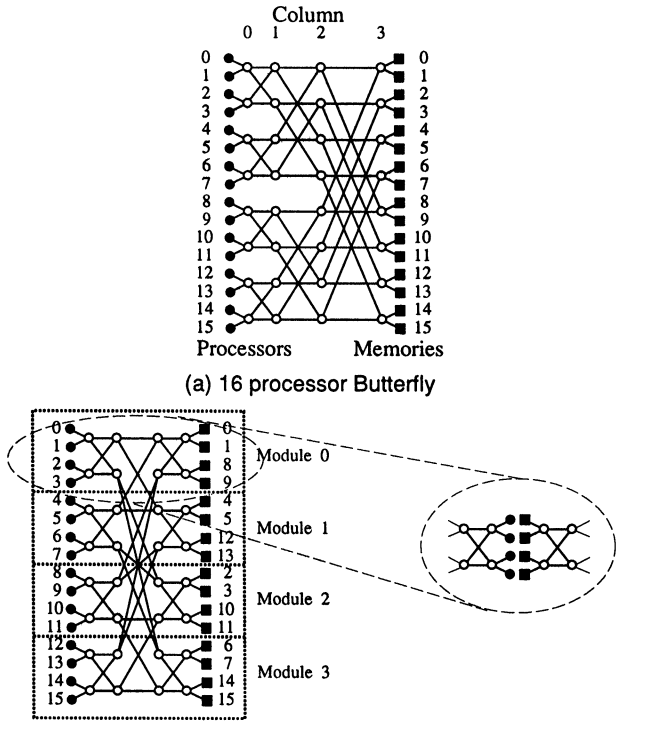
5 INTRA-MODULE NETWORKS

5.1 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. This occurs when M and N are powers of 2 and $M \leq \sqrt{N}$. Figure 1(a) shows a butterfly network connecting 16 processors to 16 memories and 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 other channels carry responses from A to B and B to A respectively.

Although we started with a *dance-hall* type net-



(a) 16 processor Butterfly

(b) Partitioning into 4 modules (c) Intra–Module connections

FIGURE 1 Butterfly Network and its partition

work, 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.

5.1.1 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 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.

5.1.2 Widening Channels

The previous network used a uniform topology that did not take advantage of the differences in packaging hierarchy. 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 intramodule 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.

5.1.3 Dilation

Another method for utilizing more local wires is dilation. We consider the 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 many messages need 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 other two 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.

5.1.4 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.

5.2 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, the resulting topology is a hypercube.

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

[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 $E = \{((u_1, u_2), (v_1, v_2))|((u_1 = v_1) \wedge (u_2, v_2) \in E_2) \vee ((u_2 = v_2) \wedge (u_1, v_1) \in E_1)\}$.

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 a n-hop network when the number of modules M is not of the form 2^{nx}, but the numbers in Table 1 do 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_4xK_8, where K_i denotes a complete graph on i nodes and x 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.

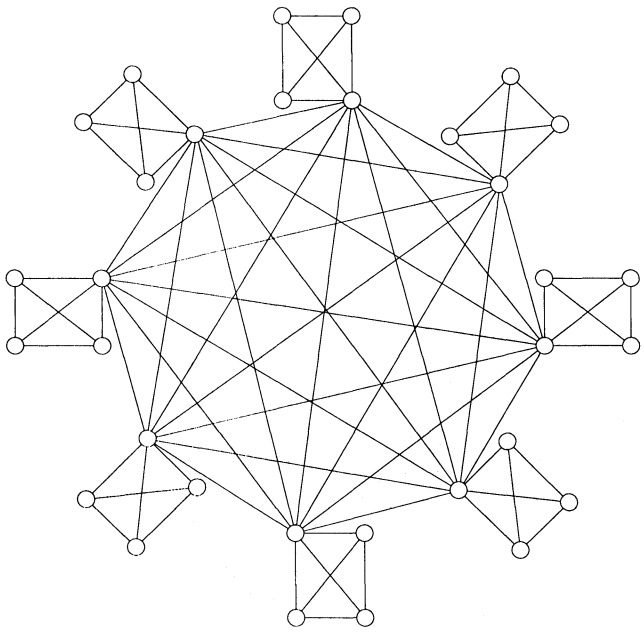


FIGURE 2 Inter-Module 2-hop Network (Only one of the 4 complete graphs on 8 nodes is shown)

As in the one-hop case, each edge in the figure corresponds to 2 pairs of channels, one in each direction. Of each pair of links, one carries 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).

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

The channels between columns 2 and 3 correspond to K_4. They connect between modules that differ in their least significant 2 bits. The channels between columns 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.

5.2.2 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 2-way dilation of the intra-module network.

5.3 Routing Algorithms

Our routing algorithms use cut-through routing [14], 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 mes-

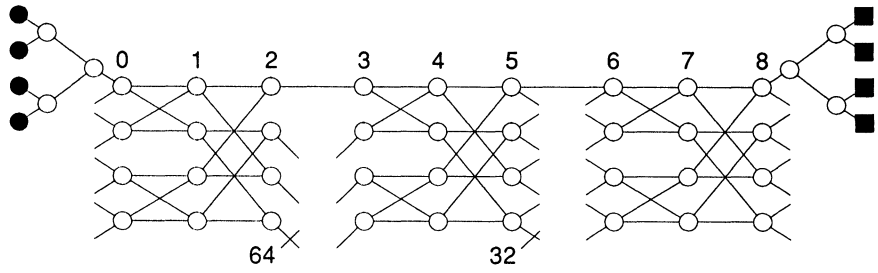


FIGURE 3 Intra-Module Network: Basic 2 hop network

sage 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 use packet-switched routing, i.e., they 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. This assumption is overly optimistic, but it is justified since we study the cross-bar networks only to establish an upper bound on performance.

6 SIMULATIONS

We simulated the networks described above under two kinds of random traffic models. In both models, we assumed that each access is sent over the network to the memory where the data resides, the access is performed and then returned to the processor. For simplicity we assumed that both the access requests and responses are 128 bits long, inclusive of the data, memory/network addressing information, and control information.

6.1 Open-Network Model

The first model we consider is the commonly used open-network model [17]. In this model, 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 $1/\lambda$.

Models similar to the open-network model have been used by many researchers to evaluate network performance, either using queueing theory (where messages are usually assumed to be Poisson streams) or simulation [1, 3, 9, 11, 15, 16, 18, 24].

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 load. The curves extend along the x-axis until the point at which the corresponding generation rate no longer 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.

We defer detailed discussion of the results to Section 6.3.

6.2 Multithreading Model

The preceding model is unrealistic in that it allows each processor to have an unbounded number of outstanding requests. Instead, in the multithreading model, each processor runs a fixed number of threads. Each thread is a sequence of instructions that can either be local operations or remote accesses. When a thread issues a remote access, the processor suspends the thread until the access completes. Each thread can have at most one outstanding access at any time. We assume a simple round-robin scheduling of threads, and also assume that the processor stalls when the previous access made by the next thread (in round-robin order) has not completed. We assume a single cycle thread-switch time, in the manner of [5].

We consider two inter-access interval distributions for the multithreading workload: 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

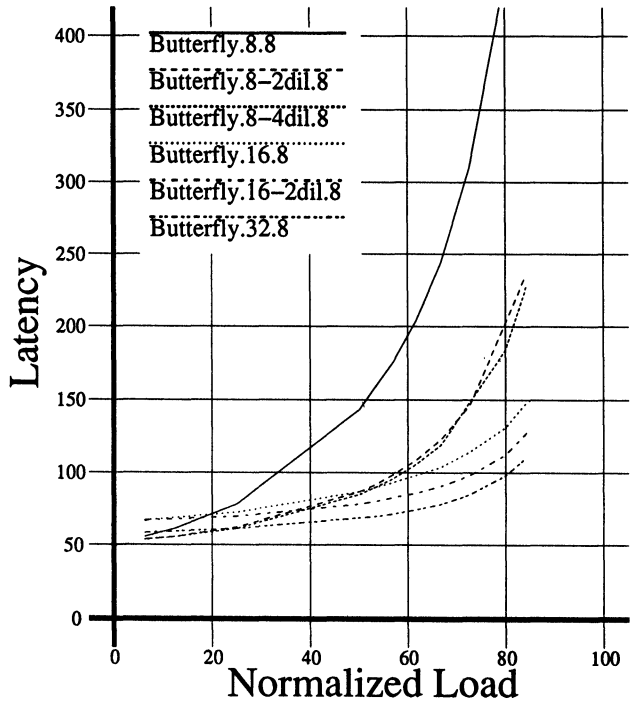


FIGURE 4 Open-Network Model; Butterfly Networks

interval is a constant. If the processor is executing a tight inner loop, the interval between remote accesses is likely to be a constant and 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.

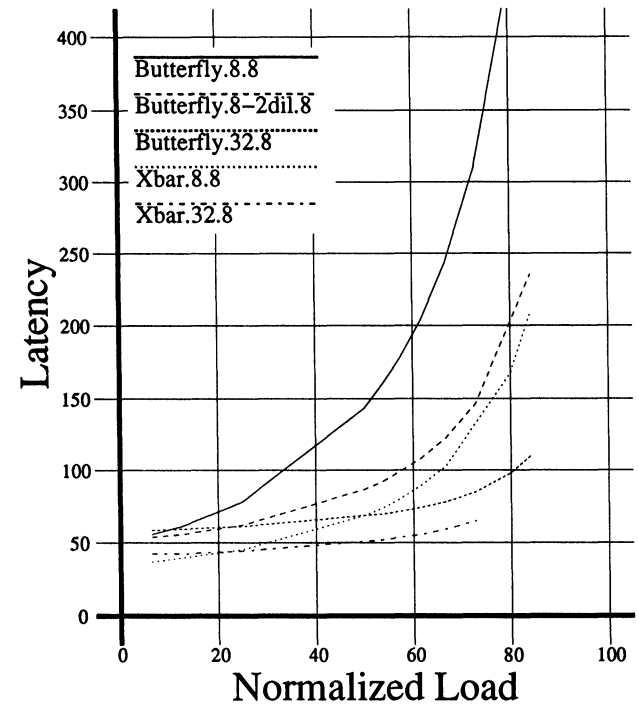


FIGURE 5 Open-Network Model; Cross-bar networks

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. 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 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 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 7 shows the processor utilizations for the different networks when the mean access interval is 16 cycles. The processor utilization is shown on the x-axis and the different networks are shown on the y-axis. For each network, there are five data points along each horizontal line, corresponding to multithreading factors of 1, 2, 4, 8, and 16. In order to make these data points easy to read, line segments connect points corresponding to the same multithreading factor. 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.

6.3 Discussion of Results

We see that in all the open-network graphs, as the applied load increases, the latency initially increases gradually, and as the network nears saturation, the latency increases rapidly.

The graphs for the multithreading model under both geometric and periodic access patterns are similar to each other, except that the utilizations for the

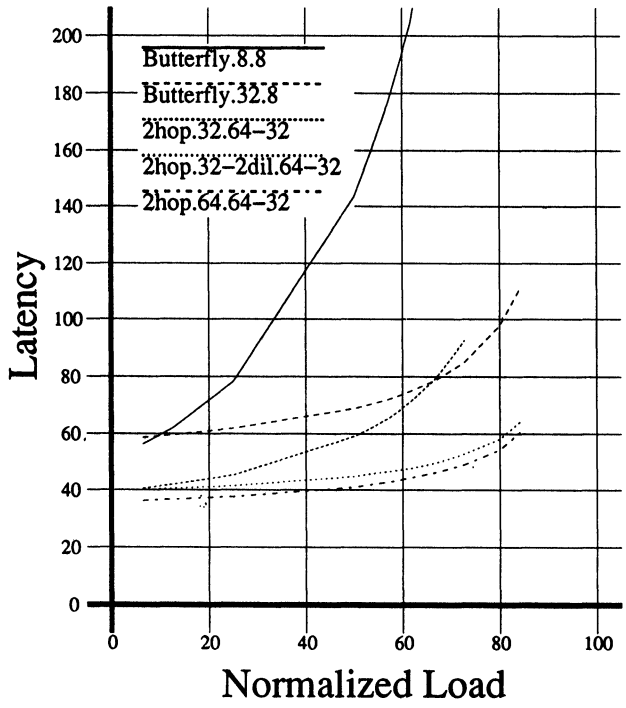


FIGURE 6 Open-Network Model; 2-hop networks

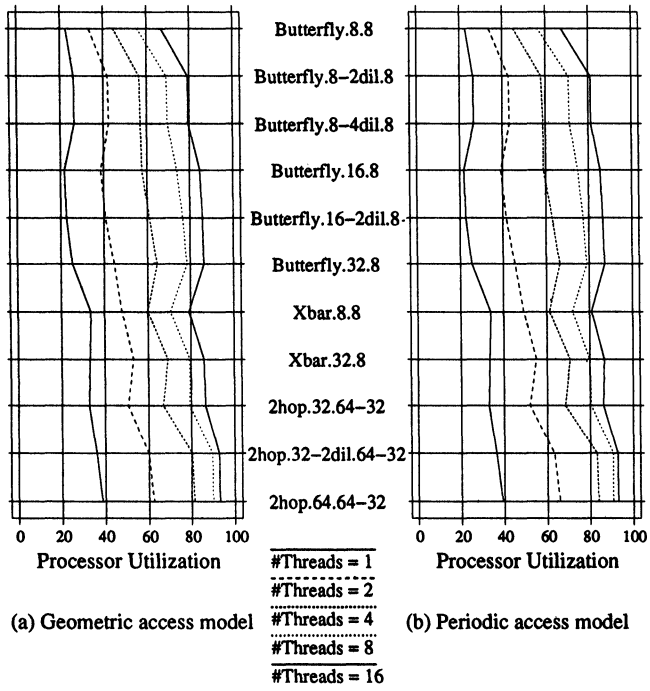


FIGURE 7 Processor utilization, mean access interval = 16 cycles

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.

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. Dilation does not increase the memory service rate because we do not dilate the channels to the memory units. Thirdly it increases message latency because of the non-uniformity in channel widths[d]. 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 *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.

[d]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.

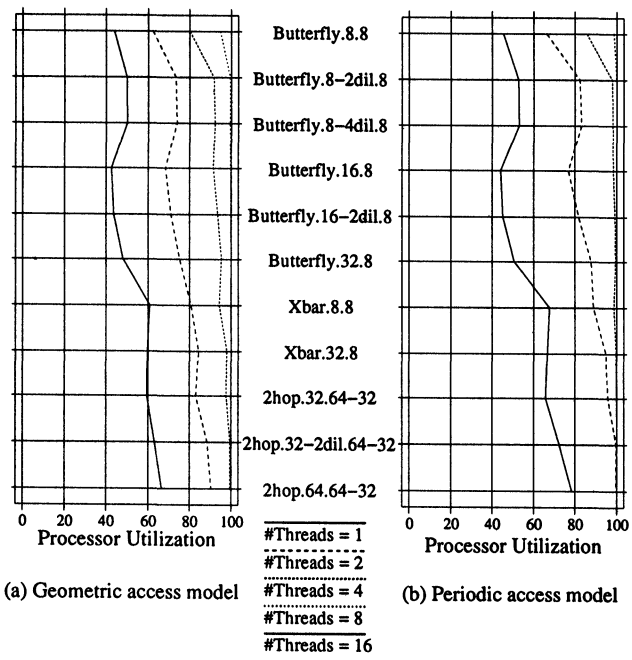


FIGURE 8 Processor utilization, mean access interval = 32 cycles

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 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. This difference does not show up under the multithreading workload (Figure 7) 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.

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. Under this comparison, we can clearly see that the complete graph networks out-perform the 2-hop networks because the complete graph is better than other inter-module topologies 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 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 8) show that at high levels of mul-

tithreading, 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 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 not grow faster than log N since the network latencies under cut-through routing grows 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.

7 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 adopt a hierarchical network design strategy in which we design different networks for each level of the hierarchy starting at the top level and proceeding down to the lower levels.

We have identified a family of networks which are promising candidates for interconnecting the top levels of the hierarchy since they exhibit a reasonable trade-off between pin-count and number of bundles. We call this family n-hop networks. A n-hop network topology is a n-fold product of complete graphs.

Our hierarchical design strategy yields hybrid network architectures which exploit packaging locality better than network architectures which start with a uniform topology that is later partitioned into a packaging hierarchy. Our results show that we can get significant improvements in performance by exploiting packaging locality. 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 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.

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] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. On communication latency in pram computations. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures,* pages 11–21, 1989.

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

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

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

[8] 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.

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

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

[11] N.J. Davis IV and H.J. Siegel. Performance studies of multiple-packet multistage cube networks and comparison to circuit switching. In *Proceedings of the International Conference on Parallel Processing,* pages 108–114, 1986.

[12] 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.

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

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

[15] Richard Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the IEEE Annual Symposium on The Foundations of Computer Science,* pages 221–230. IEEE Computer Society, 1988.

[16] 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.

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

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

[19] 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.

[20] G.F. Pfister et al. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In *Proceedings of the International Conference on Parallel Processing,* pages 764–771, 1985.

[21] 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.

[22] M.T. Raghunath. *Interconnection Network Design Based on Packaging Considerations.* PhD thesis, University of California, Berkeley, 1993. CSD-93-782.

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

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

[25] 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.

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

[27] 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.

Biographies

M.T. RAGHUNATH is a member of the Development Staff at the Highly Parallel Supercomputing Systems Laboratory of IBM in Kingston, NY. He received his B. Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Madras, India in 1987, and his M.S. and Ph.D. degrees in Computer Science from the University of California at Berkeley, in 1988 and 1993 respectively. His research interests include architectural and engineering issues associated with the development of parallel computers, interconnection network design, and software support for interprocessor communication.

ABHIRAM RANADE received the B.Tech degree in Electrical Engineering from the Indian Institute of Technology, Bombay, India in 1981 and the doctorate in Computer Science from Yale University, New Haven, CT in 1989.

Currently he is an Assistant Professor of Electrical Engineering and Computer Science at the University of California, Berkeley. His research interests include parallel architectures and algorithms, parallel programming techniques and data structures.

Dr. Ranade is a member of the ACM.