INTERCONNECT TOPOLOGIES WITH POINT-TO-POINT RINGS

by

Ross E. Johnson and James R. Goodman

Computer Sciences Technical Report #1058

December 1991

Interconnect Topologies with Point-to-Point Rings

Ross E. Johnson
ross@cs.wisc.edu
Computer Sciences Department
University of Wisconsin - Madison
1210 W. Dayton Street
Madison, WI 53706
(608) 262-6617

and

James R. Goodman
University of Wisconsin - Madison

Abstract

The Scalable Coherent Interface (SCI) is a proposed IEEE standard (IEEE P1596) that provides low-latency and high-bandwidth communication for (shared-memory) multiprocessors. The basic mechanism for communication is a ring, consisting of a sequence of unidirectional, point-to-point links. Many classical topologies can be synthesized with multiple rings, but the rings add additional constraints that complicate the synthesis. This synthesis includes both the ring mapping and the routing function.

In this paper we discuss the synthesis constraints and compare a number of classical topologies that can be synthesized with rings. Ring mapping is not trivial because rings must be closed, the ring size must be kept small, and the mapping must lead to a good routing function. Routing is not trivial because no ring may be visited twice and deadlock between finite queues must be avoided. After synthesizing five topologies, we compare their performance based on latency and throughput.

KEYWORDS: Rings, Topology, Point-to-Point, Interconnects, SCI

This work was supported in part by NSF Grant CCR-892766 and a grant from Apple Computer, Inc.

1. Introduction

The Scalable Coherent Interface (SCI) is a proposed IEEE standard interface (IEEE P1596) for providing bus-like functions to a collection of computing nodes [IEEE91, JLGS90]. It is intended to scale to thousands of high-bandwidth nodes, up to 1 Gigabyte per second per node, while providing a low-latency, cache-coherent, shared-memory interface. Achieving these objectives precluded the use of a true bus. The basic mechanism for communication is a ring consisting of a sequence of unidirectional, point-to-point links. The links are truly unidirectional in that *all* signals flow in the same direction, *i.e.*, there are no separate handshaking or flow-control signals.

The interface provides cache coherence by maintaining a distributed directory through the use of linked lists. The coherence protocol is carefully designed to assure that only a single round-trip delay through the interconnect is necessary in order to achieve the most common operations. It has also been designed with reliability in mind, guaranteeing recoverability in all cases when a packet is lost in transit.

The basic interface consists of a single input link and a single output link. Packets passing through the node normally experience very low latency — a few nanoseconds. In the absence of contention, the progress of the packet around the ring is largely limited by speed-of-light transmission delays and the number of intermediate nodes. Routing information is provided in the first few bytes of each packet.

A simple SCI node contains a single interface and possibly a processor, memory, and connections to other communication media (I/O). The node can send and receive packets. As a receiver, it is capable of recognizing a packet addressed to it, and truncates the packet, turning it into an echo packet which allows the sender to recognize that the original packet has been received.

A switch is a simple node containing more than a single interface, with the capability for routing packets from the input of one interface to the output of another. The basic ring topology, then, can be enhanced to include any topology that can be constructed from a set of rings. This paper explores some of the candidate topologies for a large-scale, regular multiprocessor constructed from a set of SCI rings. While it is not immediately apparent that most of the

classical interconnection topologies can be synthesized from such rings, we demonstrate that such is possible, and explore several well-known topologies that seem promising. The exclusive use of rings presents new constraints that limit the options for some topologies or change the trade-offs. Three constraints are particularly significant.

(1) Favored switch setting. A switch can be constructed to have f input links and f output links, but each input link has a unique preferred output link. Because of the ring structure, routing to the corresponding output link will be significantly faster than the others. Thus a topology that minimizes the number of ring switches will provide better performance, other things being equal.

(2) Different size rings. Any architectures that can be conceived of as a cylinder, *i.e.*, a "dance-hall" architecture where the two sides are connected, can be constructed out of rings. However, some of the connections may require two iterations or more through the network to arrive back at the starting nodes, while others may not. Of course, an arbitrary topology can be constructed by connecting adjacent nodes in a two-element ring, simulating a bidirectional link with a ring, but we will demonstrate that much better topologies can be constructed from rings.

(3) Deadlock-free Routing. SCI inherently pipelines transfers through a link, and routinely permits a packet to emerge from a node before it has been received completely. However, deadlock avoidance for SCI is different than for wormhole routing [DaSe87] because SCI provides sufficient buffering in a node to store the entire message. This means that, for deadlock avoidance, much of the standard analysis of store-and-forward networks may be applied [Tane81]. However, routing in SCI topologies is less restricted than in simple store-and-forward networks because packets are only stored at nodes where packets change rings. This allows routing solutions that are better than are possible with either wormhole routing or simple store-and-forward networks.

A simple ring exhibits many of the properties of a bus. Like a bus, all processors can quickly witness an event. In earlier work, we considered a large-scale, shared-memory multiprocessor architecture that employs a snooping cache protocol over a grid of buses [GoWo88]. In a sense this is a continuation of that work, investigating the best topologies for

large-scale systems. But the focus has shifted from buses, which exhibit severe physical and electrical limitations, to rings which can provide much higher performance.

Traditional metrics of a topology include the average and worst-case distance between nodes, though in practical networks an additional consideration is the bandwidth limitations of the busiest links and other resources. While these metrics have been well studied for the common topologies, the construction through rings adds a new factor to the evaluation: the number of rings traversed. If topology A provides generally shorter paths between nodes than topology B, but requires the traversal of more rings, is it better or worse? For purposes of this paper, we consider the following metrics: (1) total distance (number of links traversed), (2) number of rings traversed, (3) amount of uniform traffic through the busiest link, and (4) amount of uniform traffic through busiest queue. We then demonstrate how these metrics can be combined to determine the best topology for a given set of nodes, each with a fixed number of links.

The remainder of the paper is organized as follows. Section 2 discusses the constraints on the synthesis of topologies and section 3 synthesizes five candidate topologies. Section 4 discusses the metrics and methods used for analyzing topology performance and section 5 presents the performance results. We conclude with a summary of this work in section 6. The appendix contains the calculations for some of the performance equations.

2. Topology Synthesis

In this section we show how various topologies and routing algorithms can be defined for point-to-point interconnects. Many of the traditional topologies can be synthesized with SCI rings and this topology synthesis requires the definition of ring mapping and routing.

2.1. Ring Mapping

We number the nodes of a topology from 0 to $N-1$, where N is the number of nodes. For every node we number the input and output links each from 0 to $f-1$, where f is the node fanout. Then, ring mapping is formally defined by the function that returns the node connected to the given node via the given link. For simulation we have implemented this function (and others) in C for each topology and the code is available from american.cs.wisc.edu

(128.105.2.60) via anonymous ftp; the file is ross/topology.c. For this paper, however, we keep the description more intuitive.

The link numbering from 0 to $f-1$ defines the rings by following the same numbered links around a sequence of nodes, shown in figure 1. For every node, each input and output link is numbered. In this figure, the numbers are redundant, but later we will add letters to these numbers to make them unique and to help with the discussion of deadlock avoidance. Also, note that the link numbering does not define uniquely numbered rings. For example, link 1 defines the ring that connects nodes 1 and 2, but link 1 also defines another ring that connects nodes 5 and 6. This non-uniqueness is a result of our C code that uses link numbering from 0 to $f-1$ for each node.

The topologies in figures 1 and 2 indicate the naive way to map rings onto interconnect topologies. This is done by converting every bidirectional link into a two node ring. Figure 1

Example Mesh Topology

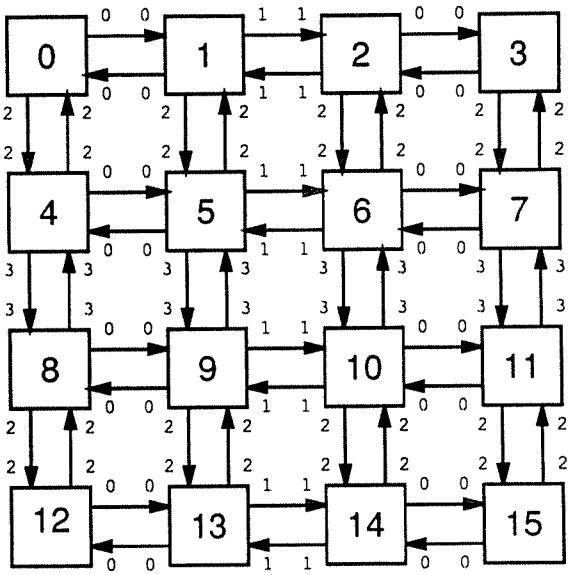


Fig. 1: In this 2D mesh topology, 16 nodes are connected by 24 rings, where each node is connected to at most 4 rings. Each 2-node ring simulates a bidirectional link. Rings are composed of links with the same link number (smaller font), although not every link with the same number is on the same ring.

shows a mesh that is mapped into rings such that each ring has two nodes and each node is on up to four rings. Each input and output link is numbered to indicate the ring to which each belongs.

Figure 2 shows a ring mapping for a completely connected topology with four nodes. Each ring has two nodes and each node is on four rings. However, the dotted lines show connections that are not needed since they form rings that have only one node. The dotted squares indicate nodes that are duplicated. Again, the links are numbered to indicate the ring to which each link belongs. For example, link 3 of node 2 goes to node 1 and the ring is completed by following link 3 from node 1 to node 2. Again, note that link numbering does not uniquely number the rings, that is, link 3 also forms a ring between nodes 0 and 3; in this topology there are 6 rings, but only 4 ring numbers. Of course, the completely connected topology is not a good topology for large systems because the hardware (specifically links) per node grows in proportion to the number of nodes.

Example Completely Connected Topology

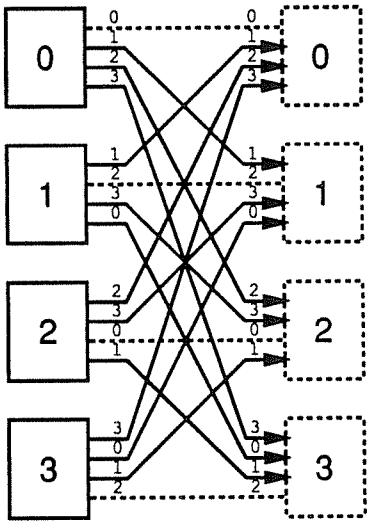


Fig. 2: In the completely connected topology, f nodes are connected by $(f+3)f/2$ rings, where $f=4$ for this figure. Dotted lines indicate unneeded links. Dotted squares indicate duplicated nodes. Excepting unneeded links, each node is connected to f rings of size 2. Rings are composed of links with the same link number (smaller font), although not every link with the same number is on the same ring.

While entirely general, this naive method of ring mapping (using only two-node rings) is inferior to other mapping strategies that are able to use all output links of a node to achieve a richer interconnection pattern. Simulating bidirectional links, a node with fanout f (f interfaces) can only reach an upper bound of $O((f-1)^k)$ nodes in k hops while other organizations may be able to reach $O(f^k)$ nodes. This is because the use of bidirectional links forces one of the links to connect to a previously-visited parent, visiting only $f-1$ new nodes at each stage. Other organizations may be able to visit f new nodes per stage. Note, for example, that no interesting topologies can be built the naive way unless at least some of the nodes have three or more links. Connections to more adjacent nodes means lower diameter graphs, which is important because the bandwidth required of the network is directly proportional to the average path length. A second argument against simulating bidirectional rings is that changing rings along a path is slower than continuing along the same ring for multiple hops. Therefore, we usually want rings to have more than two nodes.

However, we also want ring sizes to be small because echo packets must continue the full distance around every ring that is visited, even though send packets may only traverse a small number of nodes. (A new echo packet is generated on every ring visited by the send packet.) For send packets that travel a short distance on the ring, the echo packets waste a lot of bandwidth. This is because the size of echo packets is not small (about 1/5 the size of the average send packet, substantiated later). Also arguing for smaller rings, the returning echo packet (acknowledgement) allows release of the buffer resources on that ring, so smaller rings free up the resources faster. Therefore, very large rings are also bad.

2.2. Routing Functions

Given a mapping function, it remains to define the routing function. Routing is formally defined by the function that returns the next link for a packet, given (1) the node that originally generated the packet, (2) the current node through which the packet is traveling, (3) the node that is the target of the packet, and (4) the last link that was used to reach the current node. Note that the routing function defines the next link, not the next node; the mapping and routing functions together give the next node. This means that the two functions collectively provide more information than the traditional routing functions that do not deal with rings. Next, we discuss

the use of careful routing to avoid deadlock. After that, we discuss some other issues related to routing, including the definition of a general routing function that is based on breadth-first search.

2.2.1. Deadlock Avoidance

SCI is carefully defined so that forward progress is guaranteed on a single ring, but SCI does not prevent deadlock for multiple rings. Entire send packets are stored in queues at nodes where they switch rings, so circular wait and deadlock might occur if routing is not carefully thought out. This is the classical deadlock problem for store-and-forward networks [Tane81]. It should be stressed that there is nothing in SCI to prevent packets from being forwarded before they have arrived completely, as is done for wormhole routing [DaSe87]. However, SCI reserves sufficient space to store entire packets, so deadlock avoidance is easier to solve by viewing SCI as a store-and-forward network. Then, the standard solution for deadlock avoidance is to use some sort of *queue partitioning* [Tane81], where the storage is divided into classes and the assignment of space is restricted to provide a partial order on the use of the classes.

The problem with queue partitioning is that it is costly for SCI because space for an entire packet must be reserved in each partition. Alternately, we can restrict the routing algorithm so that queue partitioning is kept to a minimum, perhaps none. Restricting routing to help in deadlock avoidance was explored by Dally and Seitz for wormhole routing [DaSe87], but we explore the concept for store-and-forward networks in SCI. If the routing function requires no more than n partitions in any queue to avoid deadlock, then we say that the routing is *deadlock–n*. When queue partitioning is not required, the routing is *deadlock*–1, not to be confused with the deadlock-free routing of Dally and Seitz that still requires queue partitioning (virtual channels). If $n = O(1)$ for all sizes of a topology, we say that the routing is *deadlock–constant*.

Figure 3 shows a 12-node topology for which there is no deadlock–1 routing function. The problem is that many of the communication routes must use two queues that belong to the middle nodes. No matter how the routing is chosen, these routes form queue cycles that can lead

to deadlock. Note that the ring cycles have nothing to do with deadlock; the important resources are the queues in the nodes where packets switch rings.

Figure 4 shows a topology for which there exists a simple deadlock–1 routing. The letters relate to deadlock avoidance and will be explained shortly. The link numbers (following the letters) show where the rings are, as usual. The routing algorithm is as follows: first send packets on link 1 (if needed) and then on link 0 (if needed). For example, a packet sent from node 3 to 2 goes through nodes 3, 6, 0, 1, and 2 in that order. And, the packet visits only the queues in nodes 3, 0, and 2 because the packet changes rings at most once. In order to prove that the routing is deadlock–1, we name every input and output queue with a letter. Then, it is easy to show that there is no deadlock because packet are always transferred to queues with names

Deadlocked Topology

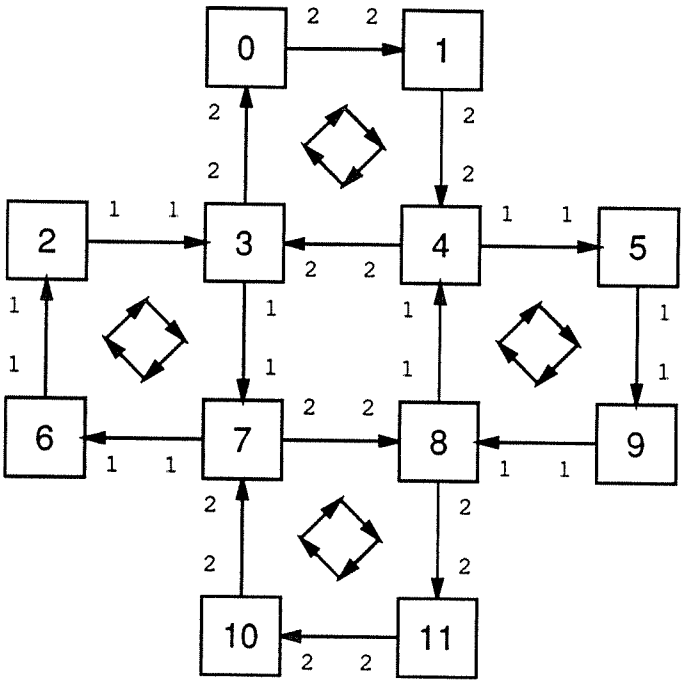


Fig. 3: This 12-node topology has no deadlock–1 routing. There are 12 nodes and 4 rings. Rings are composed of links with the same link number (smaller font), although not every link with the same number is on the same ring. That is, the 4 rings of the topology all go clockwise; the center 4 nodes are not on the same ring. Deadlock can occur because there are cycles between the store-and-forward queues, not to be confused with ring cycles.

that are strictly greater, alphabetically. That is, there can be no cycles in the graph and no deadlock. Note that this deadlock–1 routing generalizes to arbitrary dimensions and ring sizes. Also note that we were able to find a deadlock–1 routing in this k-ary n-cube, unlike Dally and Seitz, because queues are not visited at every node; queues are visited only when switching rings.

2.2.2. SCI Routing Limitation

Deadlock-1 routing is one *desirable* property of the routing function. (If a deadlock–1 routing can not be found, queue partitioning can be used to avoid deadlock.) However, a *necessary* property of the routing function is illustrated with the clock topology, shown in figure 5. This topology illustrates one routing limitation of SCI that is not related to rings. It

Example Topology with Deadlock-1 Routing

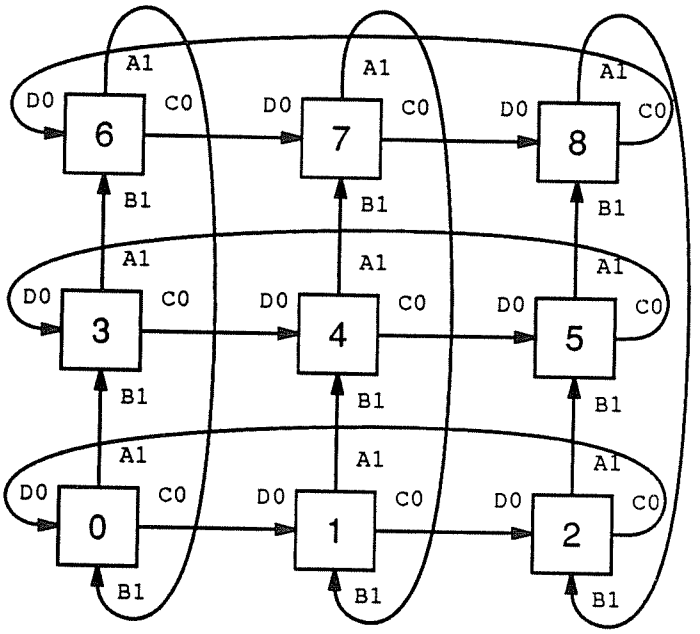


Fig. 4: In the 2D torus topology, r^2 nodes are connected by $2r$ rings, where $r=3$ for this figure. Each node is connected to at most 2 rings and each ring has r nodes. The letters show that the routing is deadlock–1, described in the text. Rings are composed of links with the same link number (following the letters). However, not every link with the same number is on the same ring.

would seem that node 2 could send a packet to node 7 by using the internal ring (entering at 3 and exiting at 6). However, this would generate two echo packets on the first ring, one echo from 3 to 2 and another echo from 7 to 6. (Recall that one echo packet is generated for every ring entry.) The problem is that echo packets contain the original source and final target nodes, but not any information about which node generated the echo. Since these echos are possibly identical and there is no ordering guaranteed by the ring, there is no way for nodes 2 and 6 to determine which echo to take off the ring. This shows a limitation of SCI: a packet can not visit the same ring twice (or more).

2.2.3. General Router

A general routing function for arbitrary topologies can be inferred from the mapping function by breadth-first search, provided that special care is taken to prevent packets from visiting the same ring twice. The general router with breadth-first search is cumbersome because the routing function usually has a more concise definition for a given topology. However, a

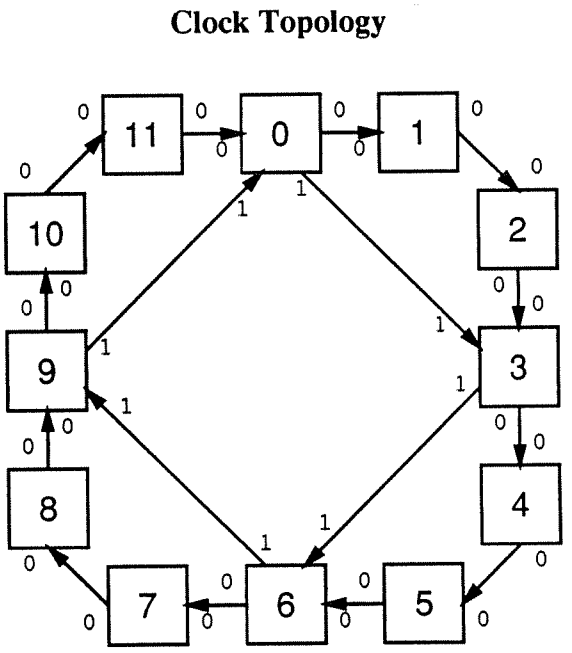


Fig. 5: In the clock topology, similar to a chordal ring [Feng81], 12 nodes are connected by two rings. This topology illustrates that the same packet can not visit the same ring twice.

general routing function is useful when creating a new topology; the mapping function can be developed using a general routing function before a more concise routing function is developed. Our general router, available via anonymous ftp, uses breadth-first search and prevents packets from visiting the same ring twice (for the topologies analyzed in this paper). Unfortunately, we do not know how to define our general router such that it is deadlock–1 (or even deadlock-constant), so deadlock avoidance must be guaranteed by queue partitioning.

3. Topology Candidates

Given the constraints placed on the mapping and the routing functions, we consider the synthesis of five topologies: a k-ary n-cube [GoWo88, Witt76], a single-stage shuffle-exchange network [Feng81, Ston71], and three versions of a multistage omega network [Feng81, GGKM83]. We call these five topologies the Multicube, Shuffle, Butterfly, Deadfly, and Livefly respectively. In this section we discuss the synthesis of each topology in turn, giving pictorial examples of the mapping functions and outlining the routing functions. Formal definitions of these functions in C are available from anonymous ftp at american.cs.wisc.edu in ross/topology.c. We also address the issue of deadlock avoidance for each topology.

3.1. Multicube Synthesis

A Multicube is an r-ary f-cube with r^f nodes, where f is the number of dimensions and r is the size of the rings. (If r varies with f, then the number of nodes is $\prod_{i=0}^{f-1} r_i$.) The dimension of a Multicube is determined by the fanout of the nodes. Two- and three-dimensional examples for $r=3$ are shown in figures 4 and 6 respectively. Each node is connected to one ring in each dimension and the link numbering (mapping function) corresponds to the dimension numbering, starting at zero. Deadlock-1 routing is simple: move packets through the dimensions in a fixed order. It is deadlock–1 because the queues can be partially ordered by the dimensions, as was illustrated in figure 4.

3.2. Shuffle Synthesis

The second ring-based topology, which we call the Shuffle, is the single-stage shuffle-exchange network of Stone [Ston71]. However, the nodes are numbered differently (because we

Example Multicube Topology

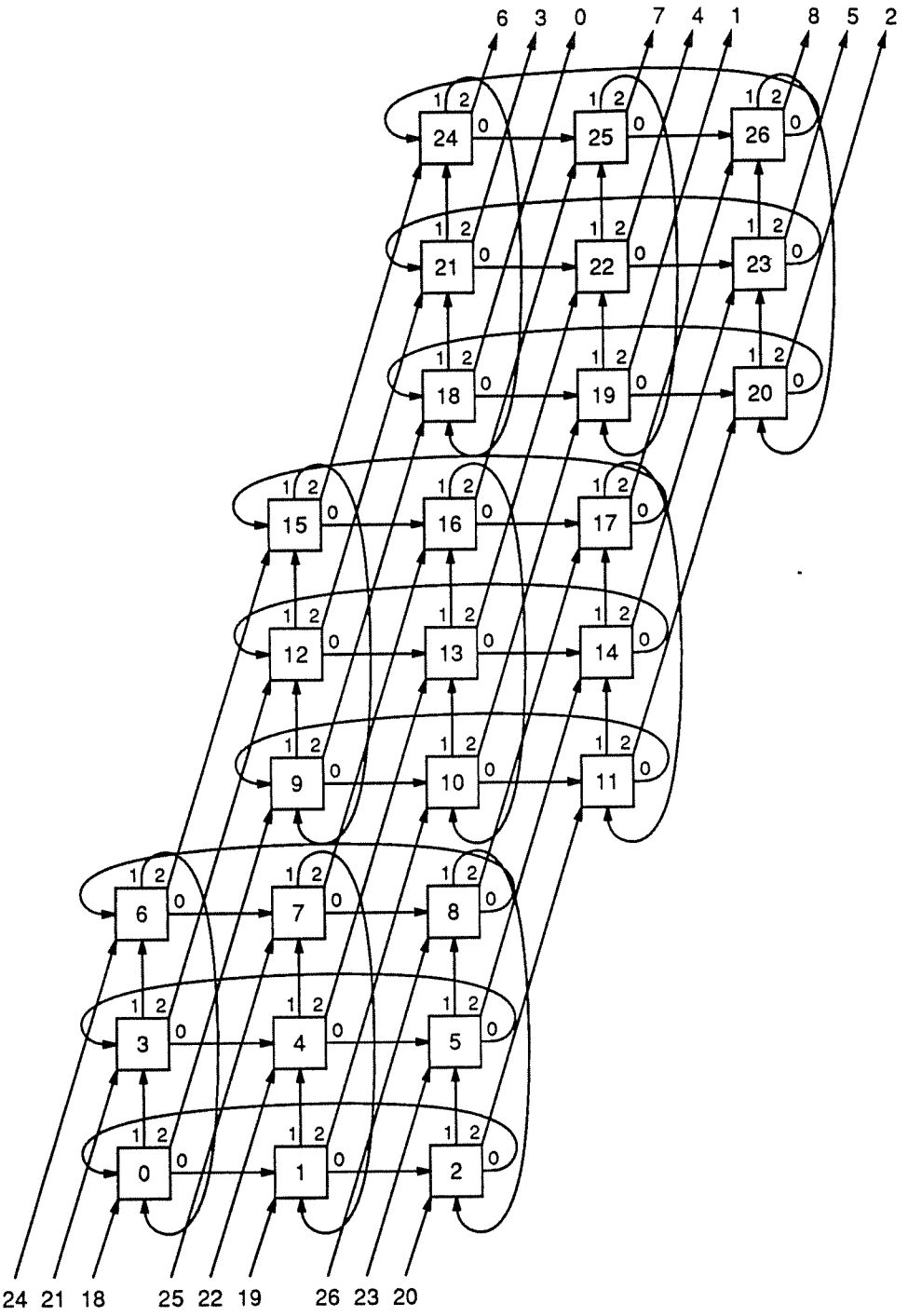


Fig. 6: The 3D torus with r^3 nodes is shown for $r = 3$. It is synthesized using $3r^2$ rings of size r. Each node is on 3 rings, shown by output links with the same link number (smaller font). However, not every output link with the same number is on the same ring. The numbering of input links is similar, but not shown.

constructed the topology before consulting the literature) and this complicates the routing function. The size of the Shuffle is f^r, where f is the fanout and $r+1$ is the maximum ring size. Two examples of link numbering (mapping function) are shown in figure 7. As of yet we have not been able to derive a concise routing function and, therefore, we use the general routing function described above. This means that we do not have a deadlock–1 routing for the Shuffle and so we must use queue partitioning for deadlock avoidance. It is likely that we would have found a concise routing function if we had used the node numbering as defined by Stone. However, we do not believe that any deadlock–1 routing exists for the Shuffle.

3.3. Butterfly Synthesis

Consider the next topology. When designing a multiprocessor with the multistage omega network, there are $O(\log(N))$ switching elements per processor. This violates our constraint of constant hardware per processor and we propose a number of ways to fix this. One way to fix it is to replace each processor with $O(\log(N))$ processors so that there are constant switches per processor; we call this the Butterfly. This solution creates two type of nodes, ones with zero processors and ones with clusters of processors. Note that clustering the processors does not exactly meet our hardware constraint because there still needs to be some multiplexing at the clusters that is not needed for other topologies. However, we will still analyze this topology in spite of the unfairness. Another way to satisfy the constant-hardware constraint is to embed the processors in the network, but we will discuss this option later.

The size of the Butterfly is $r \cdot f^r$, where f is the fanout and r is the size of the shortest rings (the number of interconnect stages). Synthesizing the ring-based Butterfly topology is not as simple as it might first appear. There are two significant problems to solve.

After numbering the nodes and drawing the connections, the first problem is to determine the mapping function. Note that the links must be numbered so that (a) no two input links of the same node are numbered the same, (b) no two output links of the same node are numbered the same, and (c) the size of all rings is minimized — our goal is to have ring sizes that are no larger than two times the number of stages in the interconnect. We have solved this problem. An example of a ring-based Butterfly is given in figure 8. Notice the numbering of the links and the rings (the mapping function). Some rings cycle once, straight through on one row. Other rings

Two Example Shuffle Topologies

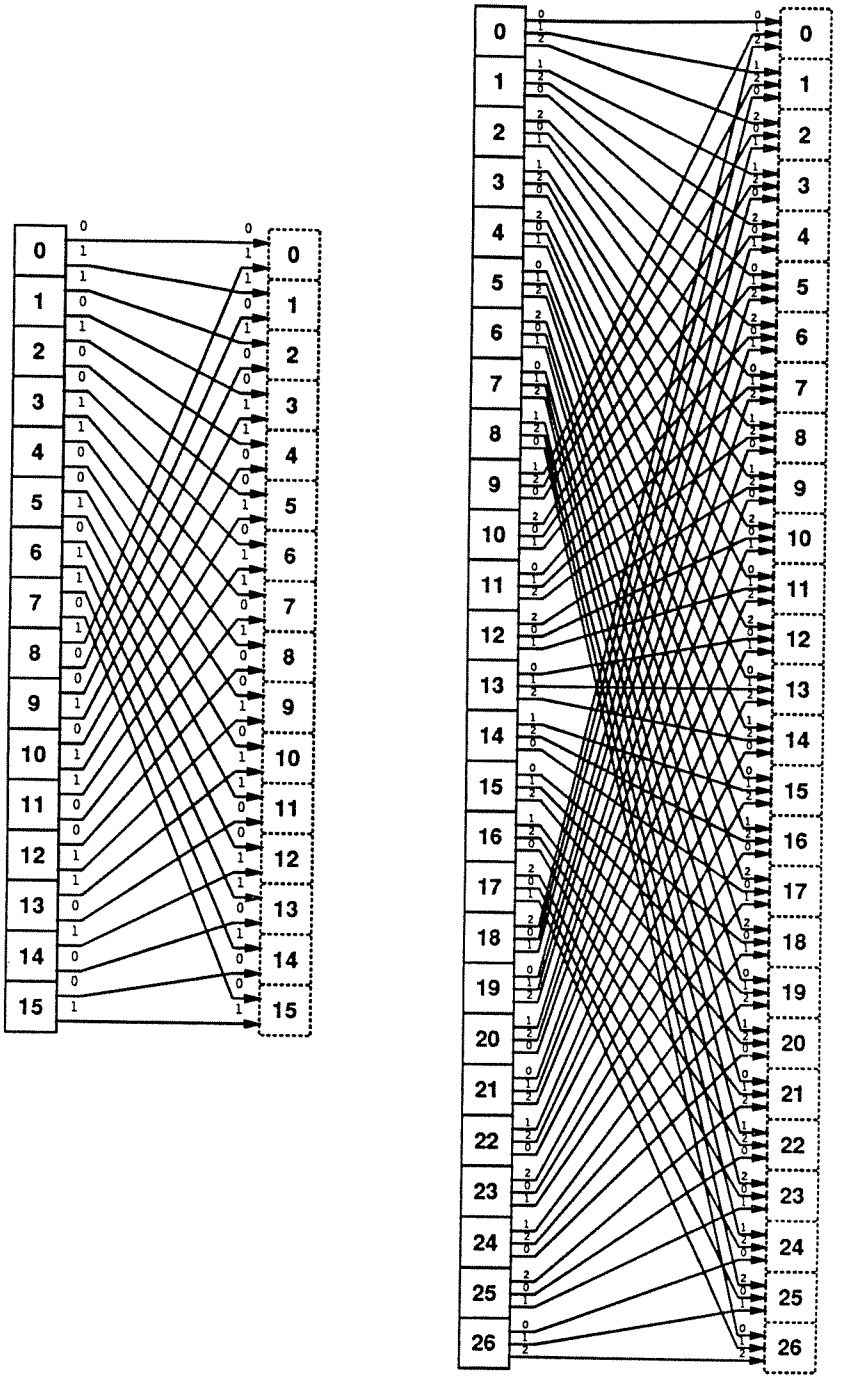


Fig. 7: Two examples of the single-stage shuffle topology are shown with f^r nodes. On the left is $f=2$ and $r=4$; on the right is $f=3$ and $r=3$. Each node is connected to f rings and the rings have maximum size of $r+1$. Rings are composed of links with the same link number (smaller font), although not every link with the same number is on the same ring.

jump rows and cycle twice before completing. The definition of the mapping function is relatively simple for binary fanout. But, the general case is fairly complex, relative to the Multicube, because it requires divide and mod operations to mask part of the node number. Incidentally, the Butterfly is isomorphic to multistage omega networks.

The second problem is to determine a deadlock–1 routing such that no packet visits the same ring twice. It is important that we have picked the correct mapping function, otherwise the routing might be impossible. Our routing for the Butterfly modifies a traditional routing, where the next node is found by changing part of the current-node number to match the target-node number (masked substitution). This routing is modified to determine the next link rather than the next node. The proof-sketch that this routing is deadlock–1 is that there exists a partial ordering of the queues (no cycles) such that packets visit queues in strictly increasing order. One partial ordering corresponds to the stages of the interconnect, except for the input queues of the first stage (which are visited last).

3.4. Deadfly Synthesis

The Butterfly makes the processor-switch ratio equal to 1 by replacing each processor by $O(\log(N))$ processors. Another way to make this ratio equal to 1 is to place a processor at every node in the topology, that is, at every switch. However, when we embed processors in the stages and use the same routing, we get the possibility of deadlock. This is because packets can start at an arbitrary stage and end at an arbitrary stage; the proof-sketch of deadlock avoidance no longer holds true. For this reason we call this topology the Deadfly and, like the Shuffle, the Deadfly requires queue partioning to avoid deadlock.

3.5. Livefly Synthesis

In order to find a deadlock–1 routing for the Deadfly, we modify the topology so that there are two short-ring links instead of one, as shown in figure 9. The link numbering illustrates the mapping function and we call this the Livefly. Deadlock-1 routing can then be accomplished in three phases. First, the packet is sent to the first column via one set of the short-ring links; call them the secondary links. Second, the packet is routed through the topology as normal, using the other short-ring links as needed; call them the primary links. Third, the packet is delivered to

Example Butterfly Topology

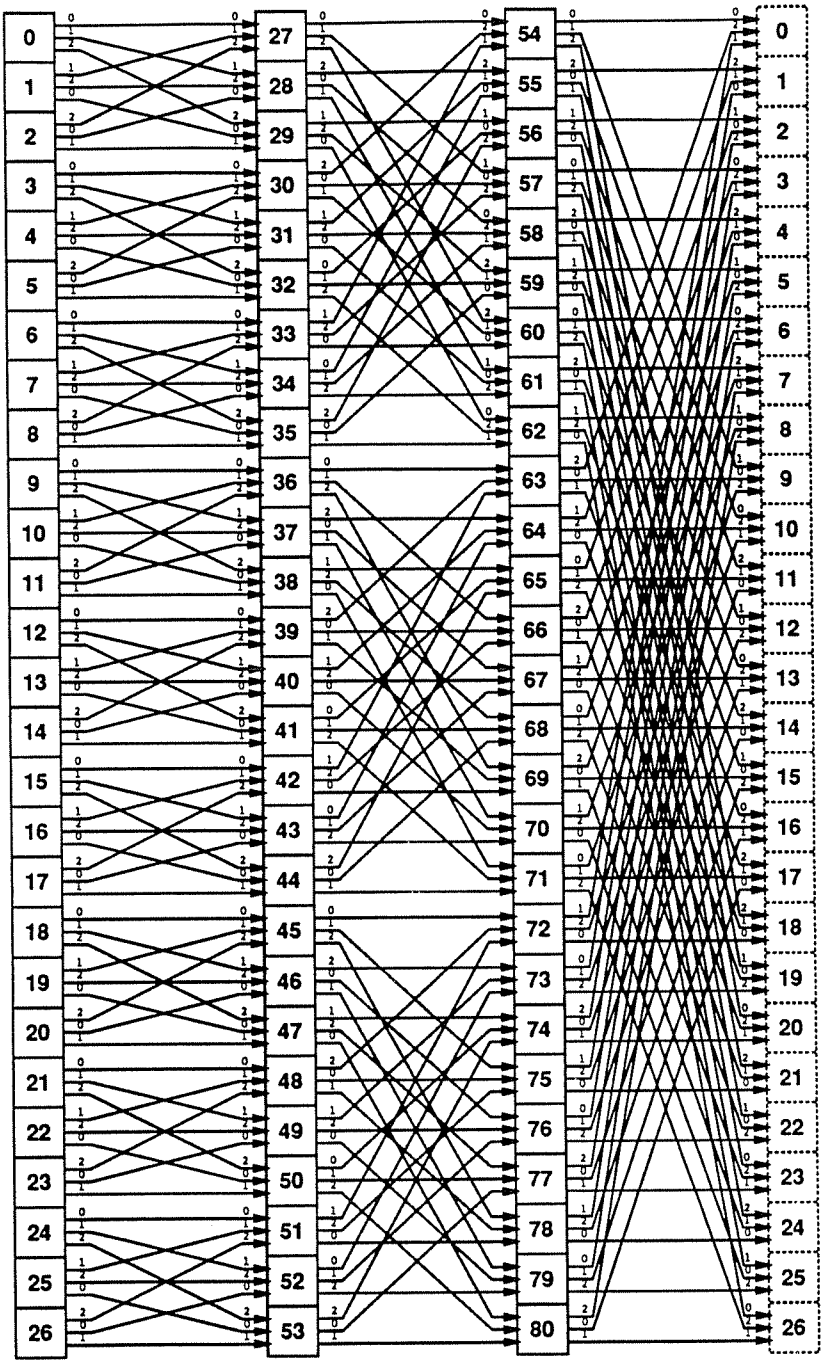


Fig. 8: This is the Butterfly with $r \cdot f^r$ nodes, where $r = f = 3$. Each node is connected to f rings and each of the $\frac{f+1}{2} f^r$ rings is connected to either r or $2r$ nodes. The short rings follow the straight links around once, but the other links make rings after two cycles. Rings are composed of links with the same link number (smaller font), although not every link with the same number is on the same ring.

the correct column via the secondary links. Any of these three phases is skipped when the phase would make no progress towards delivery of the packet.

The letters in the figure are an example of the proof-sketch that the routing for Livefly is deadlock–1. In particular, each input and output queue is given a name so that the given routing moves packets between queues of alphabetically increasing names. (Recall that queues are visited only when a packet changes rings). Notice that the input queues of the first column are labeled differently than the others and also notice that the input and output queues on the secondary links are labeled differently. These differences result from the three-phase routing. Finally, note that this topology has a performance disadvantage because one of the links is dedicated to solve the deadlock problem. Essentially, two short-ring links are used instead of

Example Livefly Topology

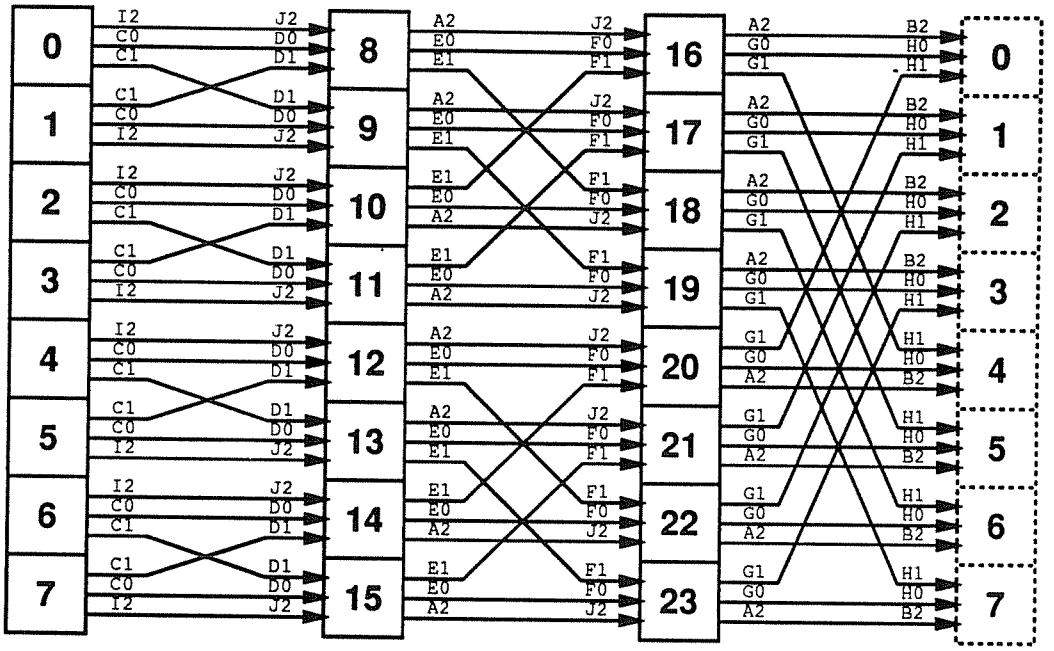


Fig. 9: Here we show the Livefly topology with 2 short-ring links per node and $r\cdot(f-1)^r$ nodes, where $f=3$ and $r=3$ for this picture. Each node is connected to f rings. There are a total of $\frac{f+2}{2}(f-1)^r$ rings of sizes r and $2r$. The letters demonstrate that the routing is deadlock–1, discussed in the text. Rings are composed of links with the same link number (following the letters), although not every link with the same number is on the same ring. The short rings follow the straight links around once, but the other links make rings after two cycles.

partitioning the one short-ring queue into two parts. Incidentally, a corollary to this proof-sketch shows that there exists a deadlock–2 routing for the Deadfly, but we chose not to explore deadlock–2 routings in this paper.

4. Metrics and Methods

In the previous sections we considered problems and solutions pertaining to the synthesis of topologies with point-to-point rings. Now that we have synthesized the five topologies, we consider their performance. First, we describe the metrics to be used. Then, we give analytical solutions for the tractable performance metrics. When a given metric is not analytically tractable, we rely on deterministic simulations. This simulation involves tracing every packet through the topology and counting the resources used. In many cases we also use simulations to verify the analytical solutions. For all comparisons we continue with the restriction of constant hardware per processor, noting exceptions when appropriate.

4.1. Metrics

It is important to know how a topology supports efficient communication under both light and heavy loads. For this reason we compare the given topologies under two scenarios. To consider light loads we compare worst-case latency when delivering one packet, assuming no congestion. For heavy loads we consider the worst-case bandwidth requirements to deliver packets between every pair of nodes in the topology.

For light loads we are mainly interested in the worst-case number of nodes visited by one packet. We call this the topology *distance*. The number of visited rings, called the *ring hops*, is also important for two reasons. First, moving packets between nodes on different rings often takes longer than moving packets between nodes on the same ring. Second, some topologies do not have deadlock-constant routings and need non-constant queue partitioning for deadlock avoidance; this requires queue resources that can hold at least ring-hops packets.

Given the distance and the ring hops we can combine these two metrics into one metric of latency under light load. Preliminary studies suggest that under a variety of light-loading conditions the time to pass through a node and change rings is approximately four times the time

to simply pass through a node. Assuming this, latency is the weighted sum of the distance plus three times the ring hops.

For heavy loads we consider the worst-case bandwidth requirements to deliver packets between every pair of nodes in the topology. We do not consider waits or retries after collisions. We merely determine the total bandwidth used per resource to complete all packet deliveries. This will show the topology bottlenecks and, subsequently, the capacity of each topology under uniform load. We determine the bandwidth used for each link and each queue to deliver the $O(N^2)$ packets. We call these traffic metrics *hot link* and *hot queue* respectively. The maximum rate at which processors can send and receive packets, as limited by the hot link for uniform traffic, is called the *throughput*. Throughput is inversely proportional to the hot-link traffic.

Figure 10 summarizes the metrics used. We use both analytical methods and simulation to determine the values of the various metrics. For the lightly loaded scenario we can use analytical methods because the math is tractable. For the heavily loaded scenario we also use analytical methods (verified by simulation), but only for symmetric topologies. We were unable to find analytical solutions for heavily-loaded asymmetric topologies, so we rely entirely on deterministic simulations for these.

Comparison Metrics	
A packet between any one pair of nodes:	
Distance	The maximum number of visited nodes
Ring Hops	The maximum number of visited rings
Latency	The weighted sum of the distance and ring hops
Packets between every pair of nodes:	
Hot-Link Traffic	The most bandwidth required of any one link
Hot-Queue Traffic	The maximum number of packets passing through any one queue
Throughput	Rate at which processors can send uniform traffic

Fig. 10: This is a listing of the comparison metrics.

4.2. Analysis

The latencies for the five topologies are derived in the appendix and summarized in figure 11. From the figure we can estimate the latency in nanoseconds by multiplying by the time to simply pass through a node, approximately 10ns for a prototype ECL implementation [Kris91].

The equations for the hot link and hot queue are derived in the appendix. The results are summarized in figure 12. In order to use the equations in the figure, we need to know e_s, the ratio between the sizes of send packets and echo packets. For the common cache-coherence transactions in SCI, half of the send packets will be 16–byte requests and the others will be 80–byte data packets. Since each packet is preceded by a 2–byte idle symbol [IEEE91], the average send packet consumes 50 bytes of bandwidth. Echo packets are 8 bytes plus the 2–byte idle, so the expected ratio is close to $e_s = 5$. This is also an upper bound for all packets related to cache-coherent data. A lower bound is 1.8, when all send packets are 16–byte request (or

Topology Latency			
Topology	Ring Hops *rings*	Distance *links*	Latency $(c-1)\cdot rings + links$
Multicube	f	$f \cdot r_m - f$	$f \cdot r_m + (c-2) f$
Butterfly	r_b	r_b	$c \cdot r_b$
Deadfly	$r_d + 1$	$2 r_d - 1$	$(c+1) r_d + (c-2)$
Livefly	$r_l + 2$	$3 r_l - 2$	$(c+2) r_l + (2c-4)$
Shuffle	r_s	r_s	$c \cdot r_s$

Fig. 11: Here we give the latency and its components, where f is the fanout and r is related to the ring size. Subscripted variables are used for r to indicate that r is not necessarily the same for different topologies with the same fanout and number of nodes. The hop penalty, c, is the ratio of latencies through a node, depending on whether or not the packet switches rings. See figure 10 for other definitions.

response) packets. To determine the bytes on the hot link we substitute 5 for e_s and multiply the formula by the average send packet size of 50.

Furthermore, we can determine the maximum issue rate, in bytes per second per processor, by taking the $N-1$ packets issued per processor, dividing by the hot-link traffic, and multiplying by the SCI throughput of 1 Gigabyte per second per link. Since each packet in the interconnect

Topology Hot Links and Hot Queues

Topology	Hot Link *send packets* $+ e_s \cdot$ *echo packets*	Hot Queue *send packets*
Multicube	$(1+e_s)\frac{r_m-1}{2}N$	$\frac{r_m-1}{r_m}f \cdot N$
Butterfly	$\frac{r_b \cdot N}{f}\left\{1+e_s\left[1+2(r_b-1)\frac{f-1}{f}\right]\right.$	$N\left[1+(r_b-1)\frac{f-1}{f}\right]$
Deadfly (long-ring links)	$\frac{r_d \cdot N}{f}\left\{1+e_s\left[1+2(r_d-1)\frac{f-1}{f}\right]\right.$	$N\left[1+r_d\frac{f-1}{f}-\frac{1}{r_d}\right]$
Deadfly (short-ring links)	$\frac{r_d \cdot N}{f-1}\left\{(r_d-1)\frac{f+1}{2r_d}-\frac{1}{f}+\frac{1}{N}+\right.$ $e_s\left[\frac{2r_d+f-5}{2}-\frac{2r_d-3}{f}-\frac{f-3}{2r_d}+\frac{r_d-1}{f^2}-\frac{1}{N}\right]$	
Livefly	simulation	simulation
Shuffle		

Fig. 12: Here we give the hot link and hot queue traffic, where N is the topology size, f is the fanout, and r is the ring size. Subscripted variables are used for the ring sizes to indicate that r is not necessarily the same for different topologies with the same f and N. The ratio between the sizes of send packets and echo packets is e_s. See figure 10 for other definitions. Except for the Butterfly, these equations have been verified by simulation.

is preceded by a 2–byte idle symbol, we also multiply by the ratio of processor bandwidth to interconnect bandwidth, namely 48 / 50. The result is maximum throughput in Gigabytes per second per processor, as limited by the hot link for uniform traffic.

5. Discussion

In this section we present the performance of the five selected topologies: Multicube, Shuffle, Deadfly, Livefly, and Butterfly. Recall that we have assumed constant hardware per processor, but have violated this constraint for two cases. First, the Shuffle and Deadfly require queue partitioning to guarantee forward progress. This partitioning requires additional information (a hop count) in each packet and more hardware complexity per node for the queue partitioning. Second, the Butterfly must multiplex $O(\log(N))$ processors at each processor cluster, adding significant complexity. In spite of these constraint violations, we compare these topologies in terms of latency and throughput and mention the violations as appropriate.

For all graphs, the keys order the topologies with respect to system sizes of 1024. Some graphs have missing data points for larger sizes for the shuffle. This is because the simulations required too much time to complete. Also, since the Livefly with fanout 2 is functionally equivalent to a single ring, the graphs do not show the Livefly for fanout 2.

5.1. Latency Graphs

We first compare the latency of the topologies with respect to latency under light load. One aspect of latency is ring hops, the maximum number of *rings* visited by any one packet, characterized by the graphs in figure 13. The ring hops for the Multicube is constant, making the shape of the curve significantly different than the others that have logarithmic shapes.

The other aspect of latency is the distance, the maximum number of *links* visited by any one packet, characterized by the graphs in figure 14. The difference between the Multicube and the others is most accented by the graph with fanout 2, where the distance grows as the square root for Multicube, rather than as approximately log (base two) for the others. This is because the Multicube's dimensionality (fanout) is held constant, rather than the radix (ring size), due to the constant-hardware assumption. Some claim that this is the wrong way to grow Multicubes. However, it is the only way to grow them, given the constant-hardware assumption.

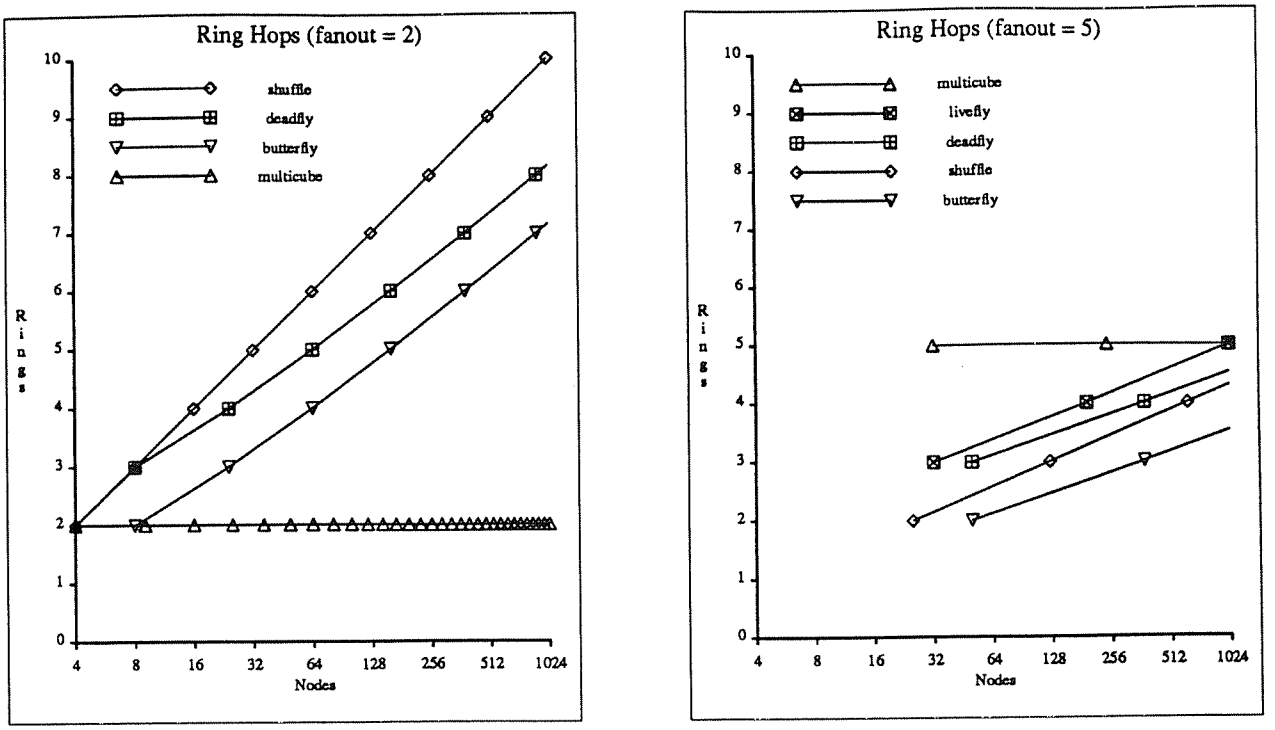


Fig. 13: These semi-log graphs show the maximum number of *rings* visited by any one packet as the fanout is varied from 2 to 8, some graphs not shown. Livefly, Deadfly, and Butterfly are always ordered as given for fanout 5. Multicube crosses the others at fanouts 3 and 4. Shuffle crosses Deadfly at fanout 3 (at about 80 nodes).

This root versus log phenomenon is also present in the curves for latency, shown in figure 15. Recall that the latency is a weighted sum of the ring hops and the distance. We also use the fact that a link hop is about 10ns [Kris91], not considering the wire-length delays. Clearly, Butterfly has the lowest latency for all configurations, but it must multiplex many processors per node. For low fanout and small systems, Multicube is competitive because it has fewer ring hops. The graphs also show that Multicube has more choices of system size for a given fanout.

As the topology size increases, so does the physical size of the multiprocessor. Some have considered the effects of three-space on the performance of topologies [Bian89, Dall90], but this is beyond our scope. Essentially, the physical layout of the topology becomes increasingly important for increasing system size, affecting the accuracy of our results on latency. On the other hand, three-space does not need to affect throughput because signals can be pipelined on the wire [ScGo91] and SCI exploits this pipelining.

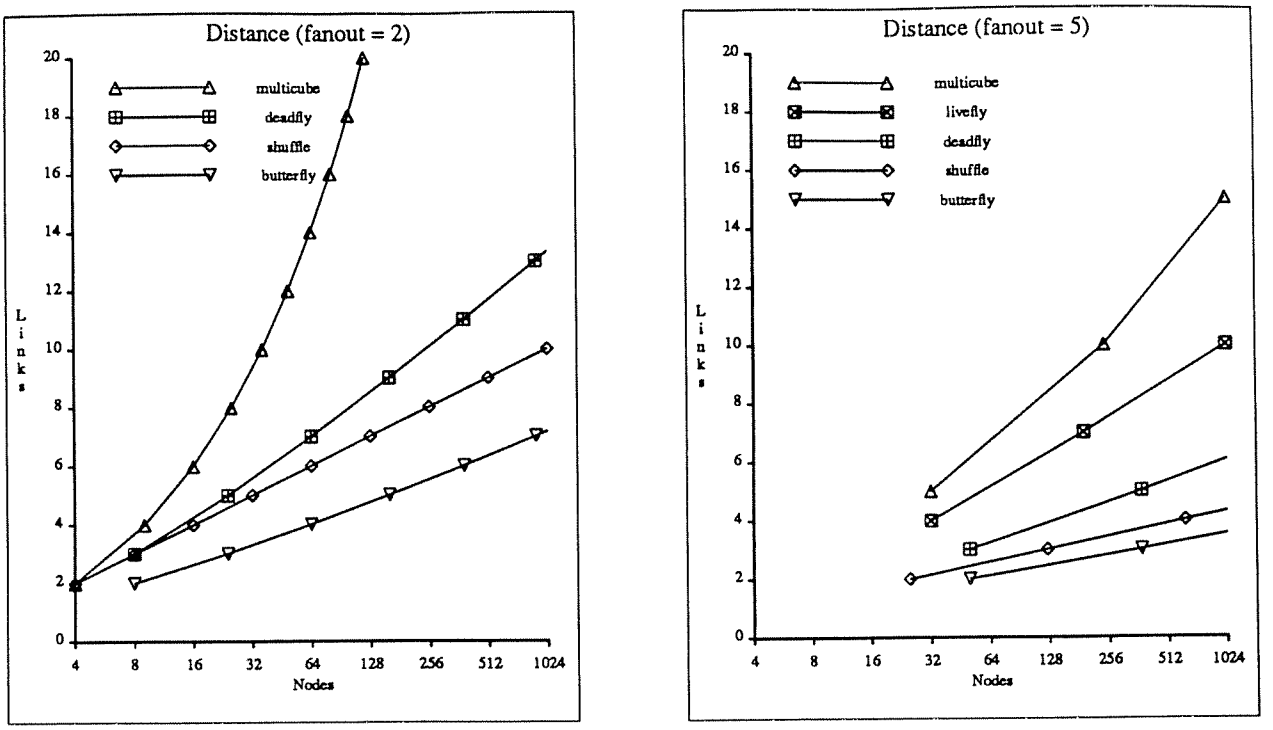
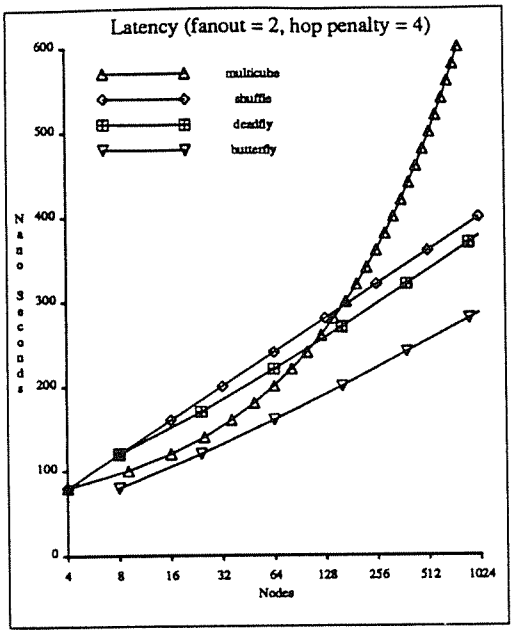


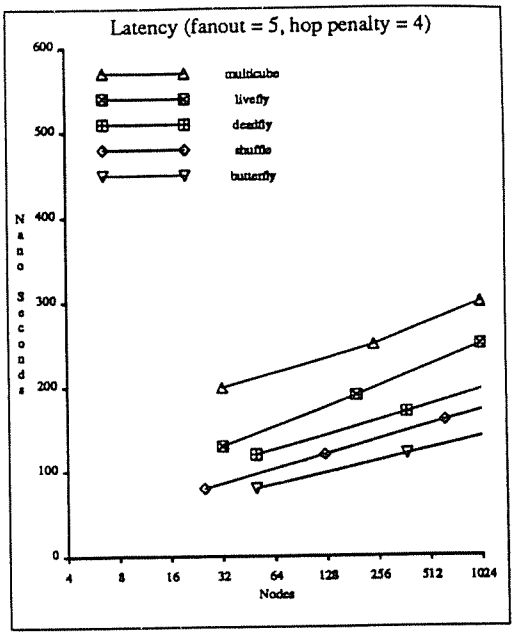
Fig. 14: These semi-log graphs show the maximum number of *links* visited by any one packet as the fanout is varied from 2 to 8, some graphs not shown. The topologies are always ordered as given for fanout 5, except that Multicube is lower than Livefly for fanout 3 and up to 128 nodes.

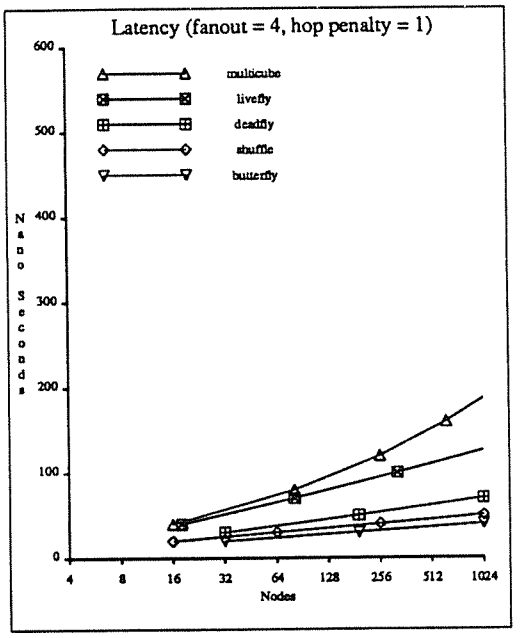
5.2. Throughput Graphs

One indicator of throughput is the hot link, which we measure by counting the maximum number of bytes through any one link when all pairs of nodes communicate at once with average sized packets. The hot link is shown in figure 16. Generally, Butterfly is best by this metric. It is interesting to note that the Shuffle is better than Multicube, even though it is not completely symmetric. The reason for this is that its asymmetry can be better characterized by its few cold links rather than its many near-average hot links.

The graphs of throughput, as limited by the hot link, are shown in figure 17. As predicted by the hot-link graphs, the Butterfly is best. However, these graphs point out that Multicube is best for small fanouts and very small system sizes. The bottom graphs show that smaller ratios with small fanouts make Multicube look better. This is because packets in Multicube visits fewer rings (ring hops), causing fewer echo packets to be generated.







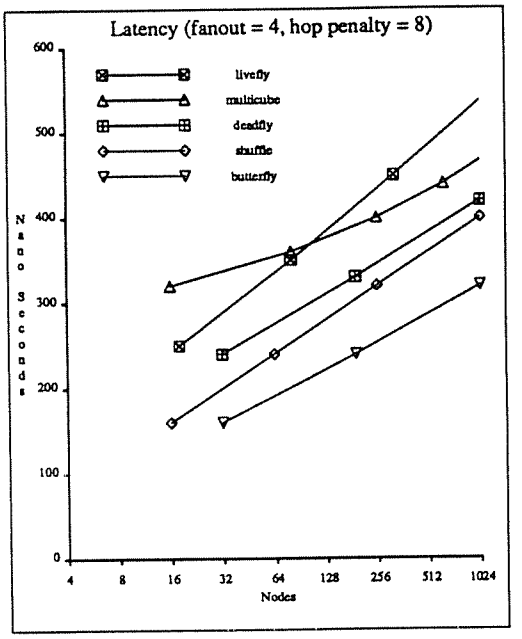


Fig. 15: These semi-log graphs show the worst-case latency of packet delivery as the fanout and hop penalty vary, some graphs not shown. The hop penalty is the the ratio of latencies through a node, depending on whether or not the packet switches rings (switching rings takes longer).

As the fanout varies from 2 to 8 with fixed hop penalty of 4 (on the top), the topologies usually remain ordered as shown for fanout 5. Fanout 2 is an exception, as shown, and Multicube is lower than Livefly for fanout 3. As the hop penalty varies from 1 to 16 with fixed fanout of 4 (on the bottom), the topologies are ordered as shown. The crossover point shown on the right begins at hop penalty 5 and stays fixed around 100 nodes as the hop penalty increases.

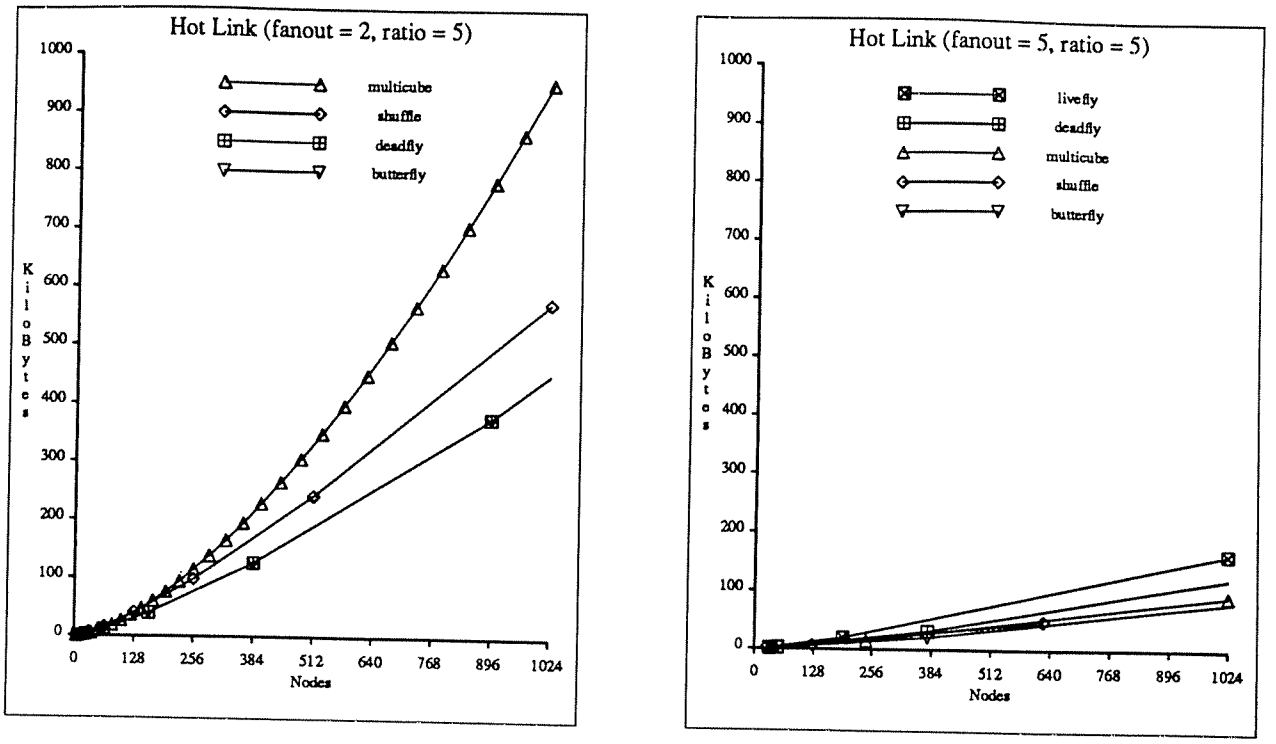
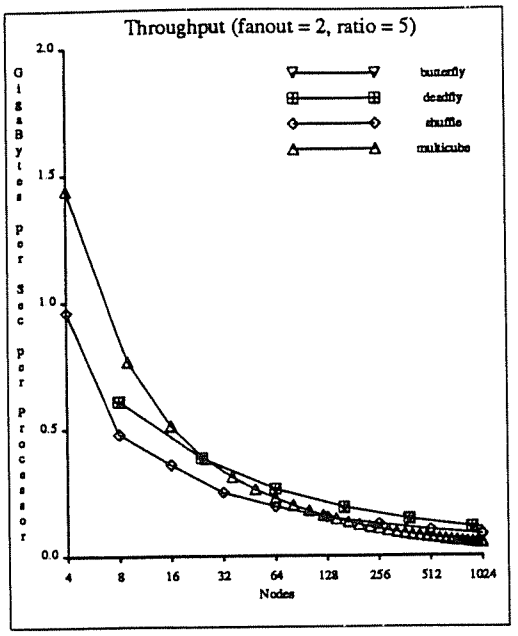
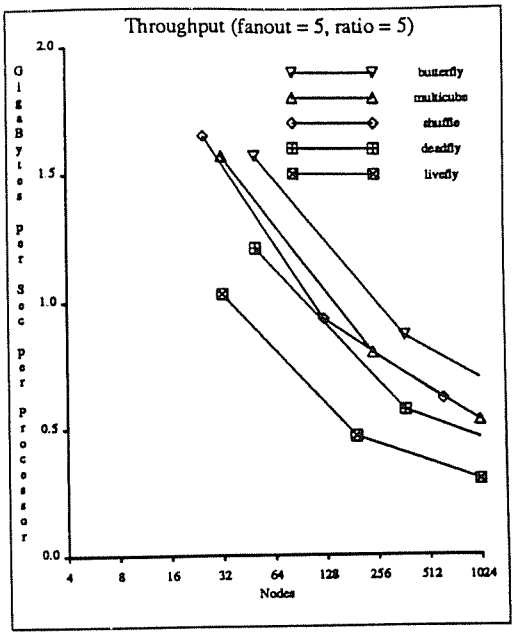


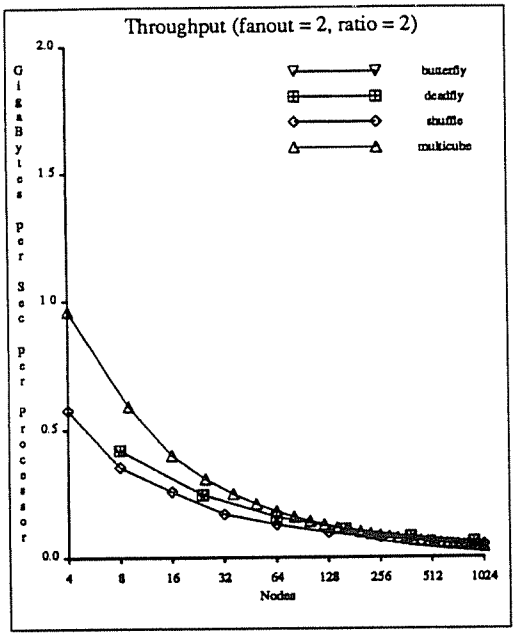
Fig. 16: These graphs show the maximum number of *bytes* passing through any one *link* when all pairs of nodes communicate once. The ratio parameter is the ratio of sizes between send packets and echo packets. Shuffle size is not fully represented because some of the simulations required too much time to complete.

Generally, for fanouts of 2 to 8, the topologies are ordered as shown for fanout 5. But for fanouts of 2 to 4, Multicube is higher than Deadfly. At fanout 2, Deadfly and Butterfly are equal. Shuffle is always slightly lower than Multicube and it is about equal with Butterfly for fanouts 6 to 8.

Figure 18 shows the hot-queue traffic, which is the number of packets passing through any one queue when all pairs of nodes communicate at once. The hotter the queue, the more likely that the finite queue will fill up. When the queue fills up, tree saturation [PfNo85] can occur, causing large losses in performance. In other words, the hot queue approximates the topology's tolerance to fluctuations in traffic flow (temporary hot spots). Since the Shuffle and Deadfly require queue partitioning to avoid deadlock, the graph lines for these topologies reflect a multiplier corresponding to the number of required partitions. This makes them look significantly worse, especially for low fanouts. For the hot queue overall, the Butterfly is best, except for fanouts 2 and 3, when the Multicube is best.







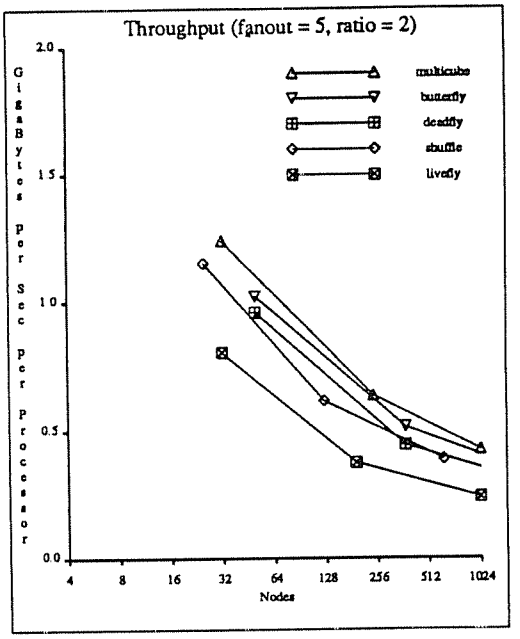


Fig. 17: These semi-log graphs show the maximum throughput as limited by the hot link. The ratio parameter is the ratio of sizes between send packets and echo packets. Shuffle size is not fully represented because some of the simulations required too much time to complete. Generally, for fanouts of 2 to 8 and a ratio of 5 (on the top), the topology ordering is as shown for fanout 5. However, Deadfly is higher than Multicube for fanouts 2 and 3 and larger system sizes; they are about equal for fanout 4.

Generally, for fanouts of 2 to 8 and a ratio of 2 (on the bottom), the topology ordering is as shown for fanout 5. However, Multicube is the lowest for fanouts of 2 and 3 for more than about 100 and 500 nodes respectively. Also, the Shuffle is higher than Deadfly for fanouts 6 to 8, but the Shuffle is the lowest for fanout 2 and up to 128 nodes.

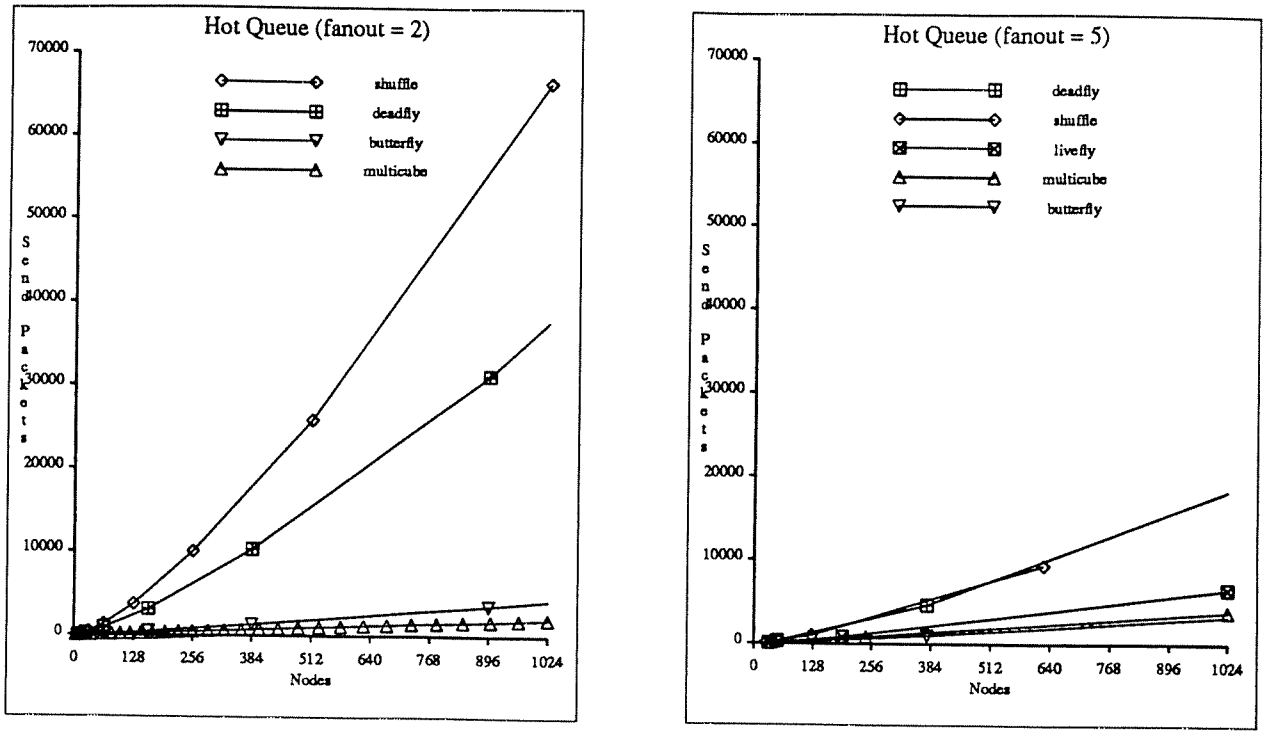


Fig. 18: These graphs show the maximum number of *packets* passing through any one *queue* when all pairs of nodes communicate once. Shuffle size is not fully represented because some of the simulations required too much time to complete.

Generally, for fanouts of 2 to 8, the topologies are ordered as shown by fanout 5. However, Multicube is the lowest for fanouts 2 and 3 and Livefly is lower than Multicube at fanout 8. Shuffle is the highest for fanouts 2 to 4 and slightly better than Deadfly for fanout 8.

6. Summary

The Scalable Coherent Interface [IEEE91, JLGS90] is a proposed IEEE standard (IEEE P1596) that provides low-latency and high-bandwidth communication for (shared-memory) multiprocessors. The basic mechanism for communication is a ring, consisting of a sequence of unidirectional, point-to-point links. Many classical topologies can be synthesized with multiple rings, but the rings add additional constraints that complicate the synthesis. This synthesis includes both the ring mapping and the routing function.

In this paper we discussed the synthesis constraints and compared five topologies that can be synthesized with rings. Ring mapping is not trivial because rings must be closed, the ring size must be kept small, and the mapping must lead to a good routing function. Routing is not trivial because no ring may be visited twice and deadlock between finite queues must be avoided. Also, the required routing functions are not trivial modifications of standard routing

functions. This is because the ring mapping and routing functions collectively provide more information than the standard topologies and associated routing functions that do not deal with rings. Concerning deadlock, we showed how some store-and-forward topologies can be synthesized without deadlock and without partitioning the queues, called deadlock–1 routing. This is important because queue partitioning complicates the implementation and makes the queues (of the same size) more vulnerable to tree saturation.

After the topologies were synthesized, they were compared with the constraint of constant hardware per processor. They were compared by four metrics: ring hops, distance, hot-link traffic, and hot-queue traffic. The first two were converted into a combined metric of latency and the second two were discussed in terms of throughput and temporary hot-spot tolerance. Where possible, analytical solutions were presented in the appendix; otherwise simulation data was given.

In summary of the graphs, we have compared Multicube, Shuffle, Butterfly, Deadfly, and Livefly. In general, the Butterfly performed better, by both metrics of latency and throughput. However, more hardware is require to implement the multiplexing of the processor clusters. Furthermore, it is not clear about how the layout in three-space will affect the wire lengths and, subsequently, the latency for large systems. As an alternative, Multicube is competitive for small system sizes and small fanouts, especially with large penalties for switching rings. It also has more choices of system size for a given fanout.

7. Acknowledgements

We thank Steve Scott for valuable comments on earlier drafts. This work was supported in part by NSF Grant CCR-892766 and a grant from Apple Computer, Inc.

8. References

[Bian89] Ronald Bianchini, "Ultracomputer Packaging and Prototypes," Ultracomputer #152, Courant Institute, Ultracomputer Research Laboratory, NYU, January 1989.

[DaSe87] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers* **36**, 5 (May 1987), 547-553.

[Dall90] William J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers* **39**, 6 (June 1990), 775-785.

[Feng81] Tse-yun Feng, "A Survey of Interconnection Networks," *IEEE Computer* **14**, 12 (December 1981), 12-27.

[GoWo88] James R. Goodman and Philip J. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor," *Proceedings of the Fifteenth Annual International Symposium on Computer Architecture*, May 1988, 422-431.

[GGKM83] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph and Marc Snir, "The NYU Ultracomputer—Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers* **32**, 2 (February 1983), 175-189.

[IEEE91] "SCI - Scalable Coherent Interface: Logical, Physical and Cache Coherence Specifications," in *The Scalable Coherent Interface*, David B. Gustavson and David V. James (eds.), IEEE Press, Montvale, New Jersey, January 1991. Draft for sponsor ballot review.

[JLGS90] David V. James, Anthony T. Laundrie, Stein Gjessing and Gurindar S. Sohi, "Scalable Coherent Interface," *IEEE Computer* **23**, 6 (June 1990), 74-77.

[Kris91] Ernst H. Kristiansen, SCI prototype under construction, personal communication, Dolphin Server Technologies, Oslo, Norway, November 1991.

[PfNo85] G. F. Pfister and V. A. Norton, "'Hot-Spot' Contention and Combining in Multistage Interconnection Networks," *ACM Transactions on Computer Systems* **34**, 10 (October 1985), 943-948.

[ScGo91] Steven L. Scott and James R. Goodman, "Performance of Pipelined K-ary N-cube Networks," Computer Sciences Technical Report #1010, University of Wisconsin-Madison, February 1991.

[Ston71] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers* **20**, 2 (February 1971), 153-161.

[Tane81] Andrew S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[Witt76] Larry D. Wittie, "Efficient Message Routing in Mega-Micro-Computer Networks," *Proceedings of the Third Annual International Symposium on Computer Architecture* **4**, 4 (January 1976), 136-140.

9. Appendix

Here we derive the equations for latency and throughput, including metrics of distance, ring hops, hot link, and hot queue.

9.1. Latency Analysis

Consider the Multicube with $N = r^f$ nodes, where r is the ring size and f is the fanout at each node. What is the distance of the topology? In the worst case a packet must visit all nodes in a ring (all links minus one) before going to the ring in the next dimension. Since it must do this in all dimensions, the distance is $(r-1)f$ and the ring hops is f. Given a ring change penalty of c, the latency is then $f \cdot r + (c-2)f$.

Consider the Butterfly with $N = r \cdot f^r$ nodes, where r is the size of the short rings and f is the fanout at each node. Recall that r processors are clustered at each node on first column. Notice that a packet must travel exactly once around the fly to get to the correct node, so the distance is r. The packet can change rings at each hop, so the ring hops is r. Given a ring change penalty of c, the latency is then $c \cdot r$.

Consider the Deadfly with $N = r \cdot f^r$ nodes, where r is the size of the short rings and f is the fanout at each node. Recall that processors are embedded in the interconnect. In the worst case a send packet must travel once around the fly to get on the correct row and then almost another time around to get to the target in the correct column. So, the distance is $2r-1$. The packet can change rings at each hop for the first time around the fly, but must stay on the same ring after being lined up. So, the ring hops is $r+1$. Given a ring change penalty of c, the latency is then $(c+1)r + (c-2)$.

Consider the Livefly with $N = r \cdot (f-1)^r$ nodes, where r is the size of the short rings and f is the fanout at each node. Recall that processors are embedded in the interconnect. In the worst case a send packet must travel almost once around the fly to get to the first column, once around the fly to get on the correct row, and then almost another time around to get to the target in the correct column. So, the distance is $3r-2$. The packet can change rings at each hop for the first time around the fly, but must stay on the same ring for the first and third phases. So, the ring hops is $r+2$. Given a ring change penalty of c, the latency is then $(c+2)r + (2c-4)$.

Consider the Shuffle with $N = f^r$ nodes, where $r+1$ is the maximum ring size and f is the fanout at each node. The worst-case distance is between nodes 0 and $N-1$, so the distance is $\log_f(N) = r$; this is also the ring hops.

The latencies for the five topologies are summarized in figure 11. Note that the ring sizes for these topologies are not exactly the same for identical values of N and f, illustrated by the different subscripts. In particular, r is $\log_f(N)$ for Shuffles, $N^{1/f}$ for Multicubes (which is at most $\log_f(N)$ when $r \le f$), approximately $\log_f(N) - \log_f(\log_f(N)) + \frac{\ln(\log_f(N))}{\log_f(N)}$ for the Butterfly and Deadfly, and approximately $\log_{f-1}(N) - \log_{f-1}(\log_{f-1}(N)) + \frac{\ln(\log_{f-1}(N))}{\log_{f-1}(N)}$ for the Livefly.

9.2. Throughput Analysis

Next we consider the scenario of packets being sent between every pair of nodes. What are the metrics of the hot link and hot queue as defined earlier? We derive the equations for the Multicube, Butterfly, and Deadfly. These equations have been verified by simulation. Since the equations for single-stage perfect-shuffle topologies do not seem tractable, we get results by simulation alone. Also, we use simulation for the Livefly because we have not yet derived those equations.

9.2.1. Multicube Analysis

Consider the Multicube with $N = r^f$ nodes, where r is the ring size and f is the fanout at each node. What is the hot-link traffic, the most bandwidth required of any one link? Since the Multicube is symmetric, we can add up the traffic over all links and divide by the total number of links. There are N^2 packets to deliver, counting the zero-bandwidth "packets" that have the same source and target. On average, each send packet traverses $\frac{r-1}{2}$ links in each of f dimensions, giving a total traffic of $N^2 \frac{r-1}{2} f$. Since there are $f \cdot N$ total links, the individual link traffic for send packets is $\frac{r-1}{2} N$. When echo packets are added, the entire rings are traversed, using r links in each dimension traversed. However, every dimension is not always used. The average links used per dimension is $r-1$ because one out of every r packets does not

need to be routed in the given dimension. This yields a total traffic of $N^2 \cdot (r-1)f$ and an individual link traffic of $(r-1)N$ when echo packets are the same size as send packets. The general formula for the hot-link traffic is then $(1+e_s)\frac{r-1}{2}N$, where e_s is the ratio of sizes between an echo packet and the average send packet.

To find the hot-queue metric we first determine the average number of rings visited by each packet. Since 1 out of every r packets is not routed in a given dimension and since there are f dimensions, the average number of rings visited by each packet is $\frac{r-1}{r}f$. Multiplying by the number of messages, N^2, and dividing by the number of nodes, N, gives the hot-queue metric of $\frac{r-1}{r}f \cdot N$.

9.2.2. Deadfly Analysis

Consider the Deadfly with $N = r \cdot f^r$ nodes, where r is the size of the short rings and f is the fanout at each node. What is the hot-link traffic? We will determine this by first finding the hot-link traffic with $e_s = 0$ (send-packet traffic) and then finding the hot-link traffic with $e_s = 1$ (total traffic). Note that packets are delivered in two rounds of the topology: the first round lines up the packet on the correct row (short ring) by hopping from ring to ring; the second finishes the delivery using only one ring. Of course, some packets find their target during the first round and do not require the second. To determine the hot-link traffic we first find the traffic during each round, assuming that all packets require two rounds. Second, we subtract the error due to the two-round assumption. In the derivation we also make use of the fact that the topology is symmetric. In particular, each node has 1 link that belongs to a ring of size r and $f-1$ links that belong to rings of size $2r$.

Consider the first round with the two-round assumption and find the send-packet traffic per long-ring link. Each node is the root of balanced tree for sending N packets, the targets of which are evenly distributed among the leaves. Each of these N^2 packets traverses r of the $f \cdot N$ links. So, each link supports transmission of $(r \cdot N)\frac{1}{f}$ packets in the first round. Note that none of the long-ring links are used in the second round because only the short-ring links are used after a

packet is lined up. Also, note that the error due to the two-round assumption only affects the short-ring links because packets are lined up after reaching their targets.

Next, find the hot-link traffic on long-ring links for $e_s = 1$. Consider the average number of long rings visited by one packet. The first traversed link has probability $\frac{f-1}{f}$ that it is a long-ring link. Each subsequent link has conditional probability of $\frac{1}{f}\frac{f-1}{f} + \frac{f-1}{f}\frac{f-2}{f}$ that it is a new long-ring link (depending on the previous link type) and this occurs $r-1$ times. Adding these together gives the average number of long rings touched by a packet. Multiplying this sum by the total number of packets, N^2, dividing by the total number of long-ring links, $(f-1)N$, and multiplying by the size of a long ring, $2r$, gives the total number of packets traversing any one long-ring link. So, the hot-link traffic for $e_s = 1$ is $(r \cdot N)\left[\frac{2}{f} + 2(r-1)\frac{f-1}{f^2}\right]$ and the hot-link metric for the long-ring links of the Deadfly is $(r \cdot N)\left[\frac{1}{f} + e_s\left[\frac{1}{f} + 2(r-1)\frac{f-1}{f^2}\right]\right]$.

Consider the hot-link traffic for short-ring links, first for $e_s = 0$. Under the two-round assumption, each short-ring link supports the same amount of traffic during the first round as do the long-ring links, namely $(r \cdot N)\frac{1}{f}$ send packets. In the second round the distances traveled by a packet are evenly distributed between 0 and $r-1$, so the total traffic in the second round is $N^2 \sum_{k=0}^{r-1} \frac{k}{r}$. Dividing this by the number, N, of short-ring links (because no long-ring links are used in the second round) gives $(r \cdot N)(r-1)\frac{1}{2r}$. Now, consider the error induced by the two-round assumption. Note that for every packet with a target that was reached in the first round, we added the distance of one full short ring. The number of erroneous targets is $\sum_{k=0}^{r-1} f^k$. Multiplying this by the error per target, $-r$, and the number of packets per target, N, and then dividing by the total number of short-ring links, N, gives the error term $-rN\left[\frac{1}{(f-1)r} - \frac{1}{(f-1)N}\right]$. So, the hot-link traffic for short-ring links and $e_s = 0$ is $(r \cdot N)\left[\frac{1}{f} + (r-1)\frac{1}{2r} - \frac{1}{(f-1)r} + \frac{1}{(f-1)N}\right]$. Simplifying gives $\frac{r \cdot N}{f-1}\left[(r-1)\frac{f+1}{2r} - \frac{1}{f} + \frac{1}{N}\right]$.

. Finally, find the hot-link traffic on short-ring links for $e_s = 1$. Consider the average number of short rings visited by one packet. The first traversed link has probability $\frac{1}{f}$ that it is a short-ring link. Each subsequent link has conditional probability of $\frac{1}{f}0 + \frac{f-1}{f}\frac{1}{f}$ that it is a new short-ring link (depending on the previous link type) and this occurs $r-1$ times. Adding these together gives the average number of short rings touched by a packet. Multiplying this sum by the total number of packets, N^2, dividing by the total number of short-ring links, N, and multiplying by the size of a short ring, r, gives the total number of packets traversing any one short-ring link. So, for the first round we have $(r \cdot N)\left[\frac{1}{f} + (r-1)\frac{f-1}{f^2}\right]$ packets per short-ring link. For the second round the probability is $\frac{f-1}{f}$ that a packet arrived on a long-ring link and starts a new short ring. So, the second-round traffic contribution per link is $(r \cdot N)\frac{f-1}{f}$. Now, consider the error term. For every target that was reached in the first round and if the previous link was a long-ring link, the two-round assumption adds one full short ring. This gives us $\sum_{k=1}^{r} \cdot f^k \frac{f-1}{f}$ erroneous targets plus one for the target that is the same as the source. To get the error term we multiply by the number of sources, N, divide by the number of short-ring links, N, multiply by the size of the short rings, r, and make it negative. After simplification the error term becomes $-(r \cdot N)\frac{1}{r}$. So, the hot-link traffic on the short-ring links for $e_s = 1$ is $(r \cdot N)\left[\frac{1}{f} + (r-1)\frac{f-1}{f^2} + \frac{f-1}{f} - \frac{1}{r}\right]$ or $(r \cdot N)(r-1)\left[\frac{f-1}{f^2} + \frac{1}{r}\right]$. Therefore, the general formula for short-ring hot-link traffic on the Deadfly is

$$\frac{r \cdot N}{f-1}\left[(r-1)\frac{f+1}{2r} - \frac{1}{f} + \frac{1}{N}\right] + e_s \frac{r \cdot N}{f-1}\left[\frac{2r+f-5}{2} - \frac{2r-3}{f} - \frac{f-3}{2r} + \frac{r-1}{f^2} - \frac{1}{N}\right].$$

To find the hot-queue metric we need to first determine the average number of rings visited by each packet. From above we know that each packet visits an average of $\frac{f-1}{f} + (r-1)\frac{(f-1)^2}{f^2}$ long-ring links and $\frac{1}{f} + (r-1)\frac{f-1}{f^2} + \frac{f-1}{f} - \frac{1}{r}$ short-ring links. If we

add these together, multiply by the total number of messages, N^2, and divide by the total number of nodes, N, then we get the hot-queue metric, $\left[1+r\,\frac{f-1}{f}-\frac{1}{r}\right]N$.

9.2.3. Butterfly Analysis

Now consider the hot link and hot queue of the Butterfly. Since exactly one full round is needed for the Butterfly, the short-ring links support the same number of send packets as do the long-ring links. However, the short-ring links must support fewer echo packets since the ring size is smaller. Therefore, the hot links are the long-ring links and this hot-link metric is already derived for the Deadfly.

Now consider the hot queue traffic. From above we know that each packet visits an average of $\frac{f-1}{f}+(r-1)\,\frac{(f-1)^2}{f^2}$ long-ring links. So, each packet visits an average of $\frac{f}{f-1}\left[\frac{f-1}{f}+(r-1)\,\frac{(f-1)^2}{f^2}\right]$ rings. Multiplying this by the number of packets, N^2, and dividing by the number of nodes, N, gives the hot queue metric, $\left[1+(r-1)\,\frac{f-1}{f}\right]N$.

Figure 15 summarizes the equations for the hot-link and hot-queue metrics. The hot-link metric for the Deadfly is the maximum of the metrics for the long-ring links and the short-ring links. Also, recall that the values of r are not exactly the same between topologies. In particular, r is $\log_f(N)$ for the Shuffle, $N^{1/f}$ for the Multicube (which is at most $\log_f(N)$ when $r \le f$), and approximately $\log_f(N)-\log_f(\log_f(N))+\frac{\ln(\log_f(N))}{\log_f(N)}$ for the Deadfly and Butterfly.

This ends the derivations of the equations.