Performance Analysis of k-ary n-cube Interconnection Networks[1,2]

William J. Dally
Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

last revised March 22, 1988

To appear in IEEE Transactions on Computers

Abstract

VLSI communication networks are wire limited. The cost of a network is not a function of the number of switches required, but rather a function of the wiring density required to construct the network. This paper analyzes communication networks of varying dimension under the assumption of constant wire bisection. Expressions for the latency, average case throughput, and hot-spot throughput of k-ary n-cube networks with constant bisection are derived that agree closely with experimental measurements. It is shown that low-dimensional networks (e.g., tori) have lower latency and higher hot-spot throughput than high-dimensional networks (e.g., binary n-cubes) with the same bisection width.

Keywords

Communication networks, interconnection networks, concurrent computing, message-passing multiprocessors, parallel processing, VLSI.

1 Introduction

The critical component of a concurrent computer is its communication network. Many algorithms are communication rather than processing limited. Fine-grain concurrent programs execute as few as 10 instructions in response to a message [7]. To efficiently execute such programs the communication network must have a latency no greater than about 10 instruction

[1] The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-80-C-0622 and N00014-85-K-0124 and in part by a National Science Foundation Presidential Young Investigator Award with matching funds from General Electric Corporation.

[2] A preliminary version of this paper appeared in the proceedings of the 1987 Stanford Conference on Advanced Research in VLSI [10].

times, and a throughput sufficient to permit a large fraction of the nodes to transmit simultaneously. Low-latency communication is also critical to support code sharing and garbage collection across nodes.

As the grain size of concurrent computers continues to decrease, communication latency becomes a more important factor. The diameter of the machine grows, messages are sent more frequently, and fewer instructions are executed in response to each message. Low latency is more difficult to achieve in a fine-grain machine because the available wiring space grows more slowly than the expected traffic. Since the machine must be constructed in three dimensions, the bisection area grows only as $N^{\frac{2}{3}}$ while traffic grows at least as fast as N, the number of nodes.

VLSI systems are wire limited. The cost of these systems is predominantly that of connecting devices, and the performance is limited by the delay of these interconnections. Thus, to achieve the required performance, the network must make efficient use of the available wire. The topology of the network must map into the three physical dimensions so that messages are not required to *double back* on themselves, and in a way that allows messages to use all of the available bandwidth along their path.

This paper considers the problem of constructing *wire-efficient* communication networks, networks that give the optimum performance for a given wire density. We compare networks holding wire bisection, the number of wires crossing a cut that evenly divides the machine, constant. Thus we compare low dimensional networks with wide communication channels against high dimensional networks with narrow channels. We investigate the class of k-ary n-cube interconnection networks and show that low-dimensional networks out perform high-dimensional networks with the same bisection width.

The remainder of this paper describes the design of wire-efficient communication networks. Section 2 describes the assumptions on which this paper is based. The family of k-ary n-cube networks is described in Section 2.1. We restrict our attention to k-ary n-cubes because it is the dimension of the network that is important, not the details of its topology. Section 2.2 introduces *wormhole routing* [20], a low-latency routing technique. Network cost is determined primarily by wire density which we will measure in terms of bisection width. Section 2.3 introduces the idea of *bisection width*, and discusses delay models for network channels. A performance model of these networks is derived in Section 3. Expressions are given for network latency as a function of traffic that agree closely with experimental results. Under the assumption of constant wire density, it is shown that low-dimensional networks achieve lower latency and better hot-spot throughput than do high-dimensional networks.

2 Preliminaries

2.1 k-ary n-cubes

Many different network topologies have been proposed for use in concurrent computers: trees [6] [15] [21], Benes networks[4], Batcher sorting networks [2], shuffle exchange networks [23], *Omega* networks [14], *indirect* binary n-cube or *flip* networks [3] [22], and direct binary n-cubes

Figure 1: A Binary 6-Cube Embedded in the Plane

[19], [17], [24]. The binary n-cube is a special case of the family of k-ary n-cubes, cubes with n dimensions and k nodes in each dimension.

Most concurrent computers have been built using networks that are either k-ary n-cubes or are isomorphic to k-ary n-cubes: rings, meshes, tori, direct and indirect binary n-cubes, and Omega networks. Thus, in this paper we restrict our attention to k-ary n-cube networks. We refer to n as the *dimension* of the cube and k as the *radix*. Dimension, radix, and number of nodes are related by the equation

$$N = k^n, \quad \left(k = \sqrt[n]{N}, \quad n = \log_k N\right). \tag{1}$$

It is the dimension of the network that is important, not the details of its topology.

A node in a k-ary n-cube can be identified by an n-digit radix k address, $a_0, \ldots, a_{n-1}$. The i^{th} digit of the address, a_i, represents the nodes position in the i^{th} dimension. Each node can forward messages to its upper neighbor in each dimension, i, with address, $a_0, \ldots, a_i + 1 (\bmod k), \ldots, a_{n-1}$.

In this paper we assume that our k-ary n-cube are unidirectional for simplicity. We will see that our results do not change appreciably for bidirectional networks. For an actual machine, however, there are many compelling reasons to make our networks bidirectional. Most importantly, bidirectional networks allow us to exploit locality of communication. If an object, A, sends a message to an object, B, there is a high probability of B sending a message back to A. In a bidirectional network, a round trip from A to B can be made short by placing A and B close together. In a unidirectional network, a round trip will always involve completely circling the machine in at least one dimension.

Figures 1-3 show three k-ary n-cube networks in order of decreasing dimension. Figure 1 shows a binary 6-cube (64 nodes). A 3-ary 4-cube (81 nodes) is shown in Figure 2. An 8-ary 2-cube (64 nodes), or torus, is shown in Figure 3. Each line in Figure 1 represents two communication channels, one in each direction, while each line in Figures 2 and 3 represents a single communication channel.

Figure 2: A Ternary 4-Cube Embedded in the Plane

Figure 3: An 8-ary 2-Cube (Torus)

2.2 Wormhole Routing

In this paper we consider networks that use *wormhole*[20] rather than *store-and-forward* [25] routing. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few flow control digits (flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or channel can accept or refuse.

As soon as a node examines the header flit(s) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message becomes spread out across the channels between the source and destination. It is possible for the first flit of a message to arrive at the destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

A method similar to wormhole routing, called *virtual cut-through*, is described in [13]. Virtual cut-through differs from wormhole routing in that it buffers messages when they block, removing them from the network. With wormhole routing, blocked messages remain in the network.

Figure 4 illustrates the advantage of wormhole routing. There are two components of latency, distance and message aspect ratio. The distance, D, is the number of *hops* required to get from the source to the destination. The message aspect ratio (message length, L, normalized to the channel width, W) is the number of channel cycles required to transmit the message across one channel. The top half of the figure shows store-and-forward routing. The message is is entirely transmitted from node N_0 to node N_1, then from N_1 to N_2 and so on. With store-and-forward routing, latency is the product of D, and $\frac{L}{W}$.

$$T_{SF} = T_c \left(D \times \frac{L}{W} \right). \tag{2}$$

The bottom half of Figure 4 shows wormhole routing. As soon as a flit arrives at a node, it is forwarded to the next node. With wormhole routing latency is reduced to the sum of D and $\frac{L}{W}$.

$$T_{WH} = T_c \left(D + \frac{L}{W} \right). \tag{3}$$

In both of these equations, T_c is the channel cycle time, the amount of time required to perform a transaction on a channel.

Figure 4: Latency of store-and-forward routing (top) vs. wormhole routing (bottom).

2.3 VLSI Complexity

VLSI computing systems [16] are wire-limited; the complexity of what can be constructed is limited by wire density, the speed at which a machine can run is limited by wire delay, and the majority of power consumed by a machine is used to drive wires. Thus, machines must be organized both logically and physically to keep wires short by exploiting locality wherever possible. The VLSI architect must organize a computing system so that its form (physical organization) fits its function (logical organization).

Networks have traditionally been analyzed under the assumption of constant channel bandwidth. Under this assumption each channel is one bit wide ($W = 1$) and has unit delay ($T_c = 1$). The constant bandwidth assumption favors networks with high dimensionality (e.g., binary n-cubes) over low-dimensional networks (e.g., tori). This assumption, however, is not consistent with the properties of VLSI technology. Networks with many dimensions require more and longer wires than do low-dimensional networks. Thus, high-dimensional networks cost more and run more slowly than low-dimensional networks. A realistic comparison of network topology must take both wire density and wire length into account.

To account for wire density, we will use bisection width [26] as a measure of network cost. The bisection width of a network is the minimum number of wires cut when the network is divided into two equal halves. Rather than comparing networks with constant channel width, W, we will compare networks with constant bisection width. Thus, we will compare low-dimensional networks with large W with high-dimensional networks with small W.

The delay of a wire depends on its length, l. For short wires, the delay, t_s, is limited by charging the capacitance of the wire and varies logarithmically with wire length.

$$t_s = \tau_{\text{inv}} e \log_e Kl, \tag{4}$$

where τ_{inv} is the inverter delay, and K is a constant depending on capacitance ratios.

For long wires, delay, t_l, is limited by the speed of light.

$$t_l = \frac{l\sqrt{\epsilon_r}}{c} \tag{5}$$

In this paper we will consider three delay models: constant delay, T_c independent of length, logarithmic delay, $T_c \propto \log l$, and linear delay, $T_c \propto l$. Our main result, that latency is minimized by low-dimensional networks, is supported by all three models.

3 Performance Analysis

In this section we compare the performance of unidirectional k-ary n-cube interconnection networks using the following assumptions:

Figure 5: A Folded Torus System

- Networks must be embedded into the plane. If a three-dimensional packaging technology becomes available, the comparison changes only slightly.
- Nodes are placed systematically by embedding $\frac{n}{2}$ logical dimensions in each of the two physical dimensions. We assume that both n and k are even integers. The long end-around connections shown in Figure 3 can be avoided by folding the network as shown in Figure 5.
- For networks with the same number of nodes, *wire density is held constant.* Each network is constructed with the same bisection width, B, the total number of wires crossing the midpoint of the network. To keep the bisection width constant, we vary the width, W, of the communication channels. We normalize to the bisection width of a bit-serial ($W = 1$) binary n-cube.
- The networks use *wormhole* routing.
- Channel delay, T_c, is a function of wire length, l. We begin by considering channel delay to be constant. Later, the comparison is performed for both logarithmic and linear wire delays; $T_c \propto \log l$ and $T_c \propto l$.

When k is even, the channels crossing the midpoint of the network are all in the highest dimension. For each of the $\sqrt{N}$ rows of the network, there are $k^{\left(\frac{n}{2}-1\right)}$ of these channels in each direction for a total of $2\sqrt{N}k^{\left(\frac{n}{2}-1\right)}$ channels. Thus, the bisection width, B, of a k-ary n-cube with W-bit wide communication channels is

$$B(k,n) = 2W\sqrt{N}k^{\left(\frac{n}{2}-1\right)} = \frac{2WN}{k}. \tag{6}$$

Figure 6: Wire Density vs. Position for One Row of a Binary 20-Cube

For a binary n-cube, $k = 2$, the bisection width is $B(2, n) = WN$. We set B equal to N to normalize to a binary n-cube with unit width channels, $W = 1$. The channel width, $W(k, n)$, of a k-ary n-cube with the same bisection width, B, follows from (6):

$$\begin{aligned} \tfrac{2W(k,n)N}{k} &= N, \\ W(k,n) &= \tfrac{k}{2}. \end{aligned} \tag{7}$$

The peak wire density is greater than the bisection width in networks with $n > 2$ because the lower dimensions contribute to wire density. The maximum density, however, is bounded by

$$\begin{aligned} D_{\max} &= 2W\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\left(\frac{k^{\frac{n}{2}}-1}{k-1}\right) \\ &= k\sqrt{N}\left(\frac{\sqrt{N}-1}{k-1}\right) < \left(\frac{k}{k-1}\right)B. \end{aligned} \tag{8}$$

A plot of wire density as a function of position for one row of a binary 20-cube is shown in Figure 6. The density is very low at the edges of the cube and quite dense near the center. The peak density for the row is 1364 at position 341. Compare this density with the bisection width of the row, which is 1024. In contrast, a two-dimensional torus has a wire density of 1024 independent of position. One advantage of high-radix networks is that they have a very uniform wire density. They make full use of available area.

Figure 7: Pin Density vs. Dimension for 256, 16K, and 1M Nodes

Each processing node connects to $2n$ channels (n input and n output) each of which is $\frac{k}{2}$ bits wide. Thus, the number of pins per processing node is

$$N_p = nk. \tag{9}$$

A plot of pin density as a function of dimension for $N = 256$, 16K and 1M nodes[3] is shown in Figure 7. Low-dimensional networks have the disadvantage of requiring many pins per processing node. A two-dimensional network with 1M nodes (not shown) requires 2048 pins and is clearly unrealizable. However, the number of pins decreases very rapidly as the dimension, n, increases. Even for 1M nodes, a dimension 4 node has only 128 pins. All of the configurations that give low latency also give a reasonable pin count.

3.1 Latency

Latency, T_l, is the sum of the latency due to the network and the latency due to the processing node,

$$T_l = T_{\text{net}} + T_{\text{node}}. \tag{10}$$

In this paper we are concerned only with T_{net}. Techniques to reduce T_{node} are described in [7] and [11].

If we select two processing nodes, P_i, P_j, at random, the average number of channels that must be traversed to send a message from P_i to P_j is given by

[3] $1K = 1024$ and, $1M = 1K \times 1K = 1048576$.

$$D = \left(\frac{k-1}{2}\right) n. \tag{11}$$

The average latency of a k-ary n-cube is calculated by substituting (7) and (11), into (3)

$$T_{\text{net}} = T_c \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k} \right). \tag{12}$$

Figure 8 shows the average network latency, T_{net}, as a function of dimension, n, for k-ary n-cubes with 2^8 (256), 2^{14} (16K), and 2^{20} (1M) nodes[4]. The left most data point in this figure corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary n-cube ($k = 2$). This figure assumes constant wire delay, T_c, and a message length, L, of 150 bits. This choice of message length was based on the analysis of a number of fine-grain concurrent programs [7]. Although constant wire delay is unrealistic, this figure illustrates that even ignoring the dependence of wire delay on wire length, low-dimensional networks achieve lower latency than high-dimensional networks.

The latency of the tori on the left side of Figure 8 is limited almost entirely by distance. The latency of the binary n-cubes on the right side of the graph is limited almost entirely by aspect ratio. With bit serial channels, these cubes take 150 cycles to transmit their messages across a single channel.

In an application that exploits locality of communication, the distance between communicating objects is reduced. In such a situation, the latency of the low-dimensional networks, dominated by distance, (the left side of Figure 8) is reduced. High-dimensional networks, on the other hand, cannot take advantage of locality. Their latency, because it is dominated by message length, will remain high.

In applications that send short messages, the component of latency due to message length is reduced resulting in lower latency for high-dimensional networks (the right side of Figure 8).

For the three cases shown in Figure 8, minimum latencies are achieved for $n = 2$, 4, and 5 respectively. In general the lowest latency is achieved when the component of latency due to distance, D, and the component due to message length, $\frac{L}{W}$, are approximately equal $D \approx \frac{L}{W}$. The following assertion makes this statement more precise.

Assertion: Minimum latency, T_{net}, occurs at a dimension, $n \leq n_x$, where n_x is the dimension for which $D = \frac{L}{W}$.

Proof: Differentiating (12) with respect to n gives

$$\frac{\partial T_{\text{net}}}{\partial n} = \frac{k - 1 - k \log k}{2} + \frac{2L \log^2 k}{k \log N}. \tag{13}$$

[4] For the sake of comparison we allow radix to take on non-integer values. For some of the dimensions considered, there is no integer radix, k, that gives the correct number of nodes. In fact, this limitation can be overcome by constructing a *mixed-radix cube* [5].

Figure 8: Latency vs. Dimension for 256, 16K, and 1M Nodes, Constant Delay

For $n = n_x$, substituting $D = \frac{L}{W}$ into (7) and (11) gives

$$4L = nk(k-1) = \frac{k(k-1)\log N}{\log k}. \tag{14}$$

Substituting into the derivitive (13) gives

$$\left.\frac{\partial T_{\text{net}}}{\partial n}\right|_{n=n_x} = \frac{k-1-k\log k}{2} + \frac{(k-1)\log k}{2} = \frac{k-1-\log k}{2}. \tag{15}$$

For all $k \geq 2$, $\left.\frac{\partial T_{\text{net}}}{\partial n}\right|_{n=n_x} \geq 0$. The derivitive is monotonically increasing for $n \leq n_x$. Thus, the minimum latency $\left(\frac{\partial T_{\text{net}}}{\partial n} = 0\right)$ occurs for $n < n_x$.

$\square$

Empirically, for all networks with $N < 2^{20}$ and integral valued k and n the minimum latency occurs when k and n are chosen so that $|D - \frac{L}{W}|$ is minimized.

The longest wire in the system becomes a bottleneck that determines the rate at which each channel operates, T_c. The length of this wire is given by

$$l = k^{\frac{n}{2}-1}. \tag{16}$$

If the wires are sufficiently short, delay depends logarithmically on wire length. If the channels are longer, they become limited by the speed of light, and delay depends linearly on channel length. Substituting (16) into (4) and (5) gives

$$T_c \propto \begin{cases} 1 + \log_e l = 1 + \left(\frac{n}{2} - 1\right) \log_e k & \text{logarithmic delay} \\ \\ l = k^{\frac{n}{2}-1} & \text{linear delay.} \end{cases} \tag{17}$$

We substitute (17) into (12) to get the network latency for these two cases:

$$T_l \propto \begin{cases} \left(1 + \left(\frac{n}{2} - 1\right) \log_e k\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{logarithmic delay} \\ \\ \left(k^{\frac{n}{2}-1}\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{linear delay.} \end{cases} \tag{18}$$

Figure 9 shows the average network latency as a function of dimension for k-ary n-cubes with 2^8 (256), 2^{14} (16K), and 2^{20} (1M) nodes, assuming logarithmic wire delay and a message length, L, of 150. Figure 10 shows the same data assuming linear wire delays. In both figures, the left most data point corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary n-cube ($k = 2$).

In the linear delay case, Figure 10, a torus ($n = 2$) always gives the lowest latency. This is because a torus offers the highest bandwidth channels and the most direct physical route between two processing nodes. Under the linear delay assumption, latency is determined solely by bandwidth and by the physical distance traversed. There is no advantage in having long channels.

Under the logarithmic delay assumption, Figure 9, a torus has the lowest latency for small networks ($N = 256$). For the larger networks, the lowest latency is achieved with slightly higher dimensions. With $N = 16K$, the lowest latency occurs when n is three[5]. With $N = 1M$, the lowest latency is achieved when n is 5. It is interesting that assuming constant wire delay does not change this result much. Recall that under the (unrealistic) constant wire delay assumption, Figure 8, the minimum latencies are achieved with dimensions of 2, 4, and 5 respectively.

The results shown in Figures 8 through 10 were derived by comparing networks under the assumption of constant wire cost to a binary n-cube with $W = 1$. For small networks it is possible to construct binary n-cubes with wider channels, and for large networks (e.g., $1M$ nodes) it may not be possible to construct a binary n-cube at all. The available wiring area grows as $N^{\frac{2}{3}}$ while the bisection width of a binary n-cube grows as N. In the case of small networks, the comparison against binary n-cubes with wide channels can be performed by expressing message length in terms of the binary n-cube's channel width, in effect decreasing the message length for purposes of comparison. The net result is the same: lower-dimensional networks give lower latency. Even if we perform the 256 node comparison against a binary n-cube with $W = 16$, the torus gives the lowest latency under the logarithmic delay model,

[5]In an actual machine the dimension n would be restricted to be an even integer.

Figure 9: Latency vs. Dimension for 256, 16K, and 1M Nodes, Logarithmic Delay

Figure 10: Latency vs. Dimension for 256, 16K, and 1M Nodes, Linear Delay

and a dimension 3 network gives minimum latency under the constant delay model. For large networks, the available wire is less than assumed, so the effective message length should be increased, making low-dimensional networks look even more favorable.

In this comparison we have assumed that only a single bit of information is in transit on each wire of the network at a given time. Under this assumption, the delay between nodes, T_c, is equal to the period of each node, T_p. In a network with long wires, however, it is possible to have several bits in transit at once. In this case, the channel delay, T_c, is a function of wire length, while the channel period, $T_p < T_c$, remains constant. Similarly, in a network with very short wires we may allow a bit to ripple through several channels before sending the next bit. In this case, $T_p > T_c$. Separating the coefficients, T_c and T_p, (3) becomes

$$T_{\text{net}} = \left(T_c D + T_p \frac{L}{W} \right). \tag{19}$$

The net effect of allowing $T_c \neq T_p$ is the same as changing the length, L, by a factor of $\frac{T_p}{T_c}$ and does not change our results significantly.

When wire cost is considered, low-dimensional networks (e.g., tori) offer lower latency than high-dimensional networks (e.g., binary n-cubes). Tori outperform binary n-cubes because they better match form to function. The logical and physical graphs of the torus are identical; Thus, messages always travel the minimum distance from source to destination. In a binary n-cube, on the other hand, the fit between form and function is not as good. A message in a binary n-cube embedded into the plane may have to traverse considerably more than the minimum distance between its source and destination.

3.2 Throughput

Throughput, another important metric of network performance, is defined as the total number of messages the network can handle per unit time. One method of estimating throughput is to calculate the capacity of a network, the total number of messages that can be in the network at once. Typically the maximum throughput of a network is some fraction of its capacity. The network capacity per node is the total bandwidth out of each node divided by the average number of channels traversed by each message. For k-ary n-cubes, the bandwidth out of each node is nW, and the average number of channels traversed is given by (11), so the network capacity per node, Γ, is given by

$$\Gamma \propto \frac{nW}{D} \propto \frac{n\left(\frac{k}{2}\right)}{\left(\frac{k-1}{2}\right)n} \approx 1. \tag{20}$$

The network capacity is independent of dimension. For a constant wire density, there is a constant network capacity.

Throughput will be less than capacity because contention causes some channels to block. This contention also increases network latency. To simplify the analysis of this contention, we make the following assumptions:

Figure 11: Contention Model for A Network

- Messages are routed using e-cube routing (in order of decreasing dimension) [8]. That is a message at node $a_0, \ldots, a_{n-1}$ destined for node $b_0, \ldots, b_{n-1}$ is first routed in dimension $n-1$ until it reaches node $a_0, \ldots, a_{n-2}, b_{n-1}$. The message is then routed in dimension $n-2$ until it reaches node $a_0, \ldots, a_{n-3}, b_{n-2}, b_{n-1}$, and so on. As shown in Figure 11, this assumption allows us to consider the contention in each dimension separately.
- The traffic from each node is generated by a Poisson process with arrival rate $\lambda \frac{\text{bits}}{\text{cycle}}$.
- Message destinations are uniformly distributed and independent.

The arrival rate of $\lambda \frac{\text{bits}}{\text{cycle}}$ corresponds to $\lambda_E = \frac{\lambda}{L} \frac{\text{messages}}{\text{cycle}}$. At the destination, each flit is serviced as soon as it arrives, so the service time at the sink is $T_0 = \frac{L}{W} = \frac{2L}{k}$. Starting with T_0 we will calculate the service time seen entering each preceding dimension.

Figure 12: Contention Model for A Single Dimension

For convenience, we will define the following quantities as illustrated in Figure 12:

$$
\begin{array}{rcll}
\gamma & = & \dfrac{1}{k} & \text{probability that a message } \textit{skips} \text{ (does not route in) a dimension,} \\
\lambda_S & = & \gamma\lambda_E & \text{rate of traffic that skips the previous dimension, } i+1, \ (\tfrac{\text{messages}}{\text{cycle}}), \\
\lambda_R & = & (1-\gamma)\lambda_E & \text{rate of traffic that routes in the previous dimension, } i+1, \ (\tfrac{\text{messages}}{\text{cycle}}), \\
\lambda_{SS} & = & \gamma^2\lambda_E & \text{rate of traffic that skips both the previous dimension, } i+1, \text{ and the current dimension, } i, \ (\tfrac{\text{messages}}{\text{cycle}}), \\
\lambda_{SR} & = & \gamma(1-\gamma)\lambda_E & \text{rate of traffic that skips the previous dimension, } i+1, \text{ and routes in the current dimension, } i, \ (\tfrac{\text{messages}}{\text{cycle}}), \\
\lambda_{RS} & = & \gamma(1-\gamma)\lambda_E & \text{rate of traffic that routes in the previous dimension, } i+1, \text{ and skips the current dimension, } i, \ (\tfrac{\text{messages}}{\text{cycle}}), \\
\lambda_{RR} & = & (1-\gamma)^2\lambda_E & \text{rate of traffic that routes in both the previous dimension, } i+1, \text{ and the current dimension, } i, \ (\tfrac{\text{messages}}{\text{cycle}}).
\end{array}
\tag{21}
$$

Consider a single dimension, i, of the network as shown in Figure 12. All messages incur a

Figure 13: Contention Model for Routing Latency

latency due to contention on entering the dimension. Those messages that are routed incur an additional latency, T_{Ri}, due to contention during routing. The rate λ_E message stream entering the dimension is composed of two components: a rate λ_S stream that skipped the previous ($i + 1^{\text{st}}$) dimension, and a rate λ_R stream that was routed in the previous dimension. These two streams are in turn split into components that will skip the i^{th} dimension (λ_{SS} and λ_{RS}) and components that will be routed in the i^{th} dimension (λ_{SR} and λ_{RR}). The entering latency seen by one component (say λ_{SS}) is given by multiplying the probability of a collision (in this case $\lambda_{RS}T_i$) by the expected latency due to a collision, (in this case $\frac{T_i}{2}$). The components that require routing must also add the latency due to contention during routing, T_{Ri}. Adding up the four components with appropriate weights gives the following equation for T_{i+1}.

$$T_{i+1} = T_i + (1-\gamma)T_{Ri} + \gamma(1-\gamma)^3\lambda_E(T_i + T_{Ri})^2 + \gamma^3(1-\gamma)\lambda_E T_i^2. \tag{22}$$

The first term of (22) is the latency seen entering the next dimension. The second term accounts for the routing latency, T_{Ri}, incurred by messages routing in this dimension (λ_{SR} and λ_{RR}). The entering latency due to contention when the two routing streams merge is given by the third term. The final term gives the entering latency for the messages that skip the dimension.

For large k, *gamma* is small and the latency is approximated by $T_{i+1} \approx T_i + T_{Ri}$. For $k = 2$ (binary n-cubes), $T_{Ri} = 0$; thus, $T_{i+1} = T_i + \frac{\lambda_E T_i}{8}$.

To calculate the routing latency, T_{Ri}, we use the model shown in Figure 13. Messages enter the dimension with rate λ_R, route through a number of stages, denoted by boxes, and exit the dimension. The latency due to contention in the stages sums to T_{Ri}. Given that a message is to be routed in a dimension, the expected number of channels traversed by the message is $\frac{k}{2}$, one entering channel and $\sigma = \frac{k-2}{2}$ continuing channels. Thus, the *average* message rate on channels continuing in the dimension is $\lambda_C = \sigma\lambda_R$.

Using the average message rate to calculate latency is an approximation. The symmetry of the network assures that the traffic on physical channels is uniform. However, using virtual channels and e-cube routing [8] results in logical channels that form a spiral. Traffic on the j^{th} channel of this spiral is given by $\lambda_{Cj} = \left(j - \frac{i^2+i}{2k}\right)\lambda_R$. Using the uniform physical message rate results in a slightly pessimistic estimate of latency since contention for the physical channel occurs on flit boundaries while contention for the logical channel occurs on message boundaries.

To compute, T_{Ri}, we work backwards from the output. The service time in the last continuing channel in dimension i is $T_{i(\sigma-1)} = T_i$. Once we know the service time for the j^{th} channel, T_{ij}, the additional service time due to contention at the $j-1^{\text{st}}$ channel is given by multiplying the probability of a collision, $\lambda_R T_{i0}$, by the expected waiting time for a collision, $\frac{T_{i0}}{2}$. Repeating this calculation σ times gives us T_{i0}.

$$\begin{aligned}
T_{i(j-1)} &= T_{ij} + \frac{\lambda_R T_{i0}^2}{2}, \\
T_{i0} &= T_i + \frac{\sigma\lambda_R T_{i0}^2}{2} = T_i + \frac{\lambda_C T_{i0}^2}{2}, \\
&= \frac{1-\sqrt{1-2\lambda_C T_i}}{\lambda_C}.
\end{aligned} \tag{23}$$

Equation (23) is valid only when $\lambda_C < \frac{T_i}{2}$. If the message rate is higher than this limit, there is no steady-state solution and latency becomes infinite. There are two solutions to (23). Here we consider only the smaller of the two latencies. The larger solution corresponds to a state that is not encountered during normal operation of a network.

To calculate T_{Ri} we also need to consider the possibility of a collision on the entering channel.

$$T_{Ri} = T_{i0}\left(1 + \frac{\lambda_C T_{i0}}{2}\right) - T_i. \tag{24}$$

If sufficient queueing is added to each network node, the service times do not increase, only the latency and equations (24) and (22) become.

$$T_{Ri} = \left(\frac{T_i}{1-\frac{\lambda_C T_0}{2}}\right)\left(1 + \frac{\lambda_C T_0}{2}\right) - T_i, \tag{25}$$

$$T_{i+1} = T_i + (1-\gamma)T_{Ri} + \left(\gamma(1-\gamma)^3 + \gamma^3(1-\gamma)\right)\lambda_E T_0. \tag{26}$$

To be effective, the total queueing between the source and destination should be greater than the expected increase in latency due to blocking. Two flits of queueing per stage is sufficient when $\lambda < 0.3$ and $L < 200$. Longer messages result in a longer service time, T_0, and require additonal queueing. The analysis here is pessimistic in that it assumes no queueing.

Figure 14: Latency vs. Traffic (λ) for 1K node networks: 32-ary 2-cube, 4-ary 5-cube, and binary 10-cube, L=200bits. Solid line is predicted latency, points are measurements taken from a simulator.

Using equation (22), we can determine (1) the maximum throughput of the network and (2) how network latency increases with traffic.

Figures 14 and 15 show how latency increases as a function of applied traffic for 1K node and 4K node k-ary n-cubes. The vertical axis shows latency in cycles. The horizontal axis is traffic per node, λ, in bits/cycle. The figures compare measurements from a network simulator (points) to the latency predicted by (24) (lines). The simulation agrees with the prediction within a few percent until the network approaches saturation.

For 1K networks, a 32-ary 2-cube always gives the lowest latency. For 4K networks, a 16-ary 3-cube gives the lowest latency when $\lambda < 0.2$. Because latency increases more slowly for 2-dimensional networks, a 64-ary 2-cube gives the lowest latency when $\lambda > 0.2$.

At the left side of each graph ($\lambda = 0$), latency is given by (12). As traffic is applied to the network latency increases slowly due to contention in the network until saturation is reached. Saturation occurs when λ is between 0.3 and 0.5 depending on the network topology. Networks should be designed to operate on the flat portion of the curve ($\lambda < 0.25$).

When the network saturates, throughput levels off as shown in Figures 16 and 17. These figures

Figure 15: Latency vs. Traffic (λ) for 4K node networks: 64-ary 2-cube, 16-ary 3-cube, 8-ary 4-cube, 4-ary 6-cube, and binary 12-cube, L=200bits. Solid line is predicted latency, points are measurements taken from a simulator.

Figure 16: Actual Traffic vs. Attempted Traffic for 1K node networks: 32-ary 2-cube, 4-ary 5-cube, and binary 10-cube, L=200bits.

Figure 17: Actual Traffic vs. Attempted Traffic for 4K node networks: 64-ary 2-cube, 16-ary 3-cube, 8-ary 4-cube, 4-ary 6-cube, and binary 12-cube, L=200bits.

Parameter	1K Nodes			4K Nodes				
Dimension	2	5	10	2	3	4	6	12
radix	32	4	2	64	16	8	4	2
Max Throughput	0.36	0.41	0.43	0.35	0.31	0.31	0.36	0.41
Latency $\lambda = 0.1$	46.1	128.	233.	70.7	55.2	79.9	135.	241.
Latency $\lambda = 0.2$	50.5	161.	269.	73.1	70.3	112.	181.	288.
Latency $\lambda = 0.3$	59.3	221.	317.	78.6	135.	245.	287.	357.

Table 1: Maximum Throughput as a Fraction of Capacity and Blocking Latency in Cycles

show how much traffic is delivered (vertical axis) when the nodes attempt to inject a given amount of traffic (horizontal axis). The curve is linear (actual = attempted) until saturation is reached. From this point on, actual traffic is constant. This plateau occurs because (1) the network is source queued, and (2) messages that encounter contention are blocked rather than aborted. In networks where contention is resolved by dropping messages, throughput usually decreases beyond saturation.

To find the maximum throughput of the network, the source service time, T_0, is set equal to the reciprocal of the message rate, λ_E, and equations (22), (23), and (24) are solved for λ_E. At this operating point the network can accept no more traffic. Messages are being offered as fast as the network can deliver them. The maximum throughput as a fraction of capacity for k-ary n-cubes with 1K and 4K nodes is tabulated in Table 1. Also shown is the total latency for L = 200bit messages at several message rates. The table shows that the additional latency due to blocking is significantly reduced as dimension is decreased.

In networks of constant bisection width, the latency of low-dimensional networks increases more slowly with applied traffic than the latency of high-dimensional networks. At $\lambda = 0.2$, the 32-ary 2-cube has $\approx \frac{1}{5}$ the latency of the binary 10-cube. At this point, the additional latency due to contention in the 32-ary 2-cube is $7T_c$ compared to $64T_c$ in the binary 10-cube.

Low-dimensional networks handle contention better because they use fewer channels of higher bandwidth and thus get better queueing performance. The shorter service times, $\frac{L}{W}$, of these networks results in both a lower probability of collision, and a lower expected waiting time in the event of a collision. Thus the blocking latency at each node is reduced quadratically as k is increased. Low-dimensional networks require more hops, $D = \frac{n(k-1)}{2}$, and have a higher rate on the continuing channels, λ_C. However, messages travel on the continuing channels more frequently than on the entering channels, thus most contention is with the lower rate channels. Having fewer channels of higher bandwidth also improves hot-spot throughput as described below.

3.3 Hot Spot Throughput

In many situations traffic is not uniform, but rather is concentrated into *hot spots*. A *hot spot* is a pair of nodes that accounts for a disproportionately large portion of the total network traffic. As described by Pfister [18] for a shared-memory computer, hot-spot traffic can degrade performance of the entire network by causing congestion.

The *hot-spot throughput* of a network is the maximum rate at which messages can be sent from one specific node, P_i, to another specific node, P_j. For a k-ary n-cube with deterministic routing, the hot-spot throughput, Θ_{HS}, is just the bandwidth of a single channel, W. Thus, under the assumption of constant wire cost we have

$$\Theta_{\mathrm{HS}} = W = k - 1. \tag{27}$$

Low-dimensional networks have greater channel bandwidth and thus have greater hot-spot throughput than do high-dimensional networks. Low-dimensional networks operate better under non-uniform loads because they do more resource sharing. In an interconnection network the resources are wires. In a high-dimensional network, wires are assigned to particular dimensions and cannot be shared between dimensions. For example, in a binary n-cube it is possible for a wire to be saturated while a physically adjacent wire assigned to a different dimension remains idle. In a torus all physically adjacent wires are combined into a single channel that is shared by all messages that must traverse the physical distance spanned by the channel.

4 Conclusion

Under the assumption of constant wire bisection, low-dimensional networks with wide channels provide lower latency, less contention, and higher hot-spot throughput than high-dimensional networks with narrow channels. Minimum network latency is achieved when the network radix, k, and dimension, n, are chosen to make the components of latency due to distance, D, and aspect ratio, $\frac{L}{W}$ approximately equal. The minimum latency occurs at a very low dimension, 2 for up to 1024 nodes.

Low dimensional networks reduce contention because having a few high-bandwidth channels results in more resource sharing and thus better queueing performance than having many low-bandwidth channels. While network capacity and worst-case blocking latency are independent of dimension, low-dimensional networks have a higher maximum throughput and lower average blocking latency than do high-dimensional networks. Improved resource sharing also gives low-dimensional networks higher hot-spot throughput than high-dimensional networks.

The results of this paper have all been made under the assumption of constant channel delay, independent of channel length. The main result, that low-dimensional networks give minimum latency, however, does not change appreciably when logarithmic or linear delay models are considered. In choosing a delay model one must consider how the delay of a switching node compares to the delay of a wire. Current VLSI routing chips [9] have delays of tens of nanoseconds, enough time to drive several meters of wire. For such systems a constant delay model is adequate. As chips get faster and systems get larger, however, a linear delay model will more accurately reflect system performance.

Fat-tree networks have been shown to be universal in the sense that they can *efficiently* simulate any other network of the same volume [15]. However, the analysis of these networks has not considered latency. k-ary n-cubes with appropriately chosen radix and dimension are also universal in this sense. A detailed proof is beyond the scope of this paper. Intuitively, one cannot do any better than to fill each of the three physical dimensions with wires and place switches at every point of intersection. Any point-to-point network can be embedded into such a 3-D mesh with no more than a constant increase in wiring length.

This paper has considered only *direct* networks [19]. The results do not apply to *indirect* networks. The depth and the switch degree of an indirect network are analogous to the dimension and radix of a direct network. However, the bisection width of an indirect network is independent of switch degree. Because indirect networks do not exploit locality it is not possible to trade off diameter for bandwidth. When wire density is the limiting resource, a high-bandwidth direct network is preferrable to an indirect network.

The low-dimensional k-ary n-cube provides a very general communication media for digital systems. These networks have been developed primarily for message-passing concurrent computers. They could also be used in place of a bus or indirect network in a shared-memory concurrent computer, in place of a bus to connect the components of a sequential computer, or to connect subsystems of a special purpose digital system. With VLSI communication chips the cost of implementing a network node is comparable to the cost of interfacing to a shared bus, and the performance of the network is considerably greater than the performance of a bus.

The networks described here have been demonstrated in the laboratory and incorporated into commercial multiprocessors. The Torus Routing Chip (TRC) is a VLSI chip designed to implement low-dimensional k-ary n-cube interconnection networks [9]. The TRC performs wormhole routing in arbitrary k-ary n-cube interconnection networks. A single TRC provides 8-bit data channels in two dimensions and can be cascaded to add more dimensions or wider data channels. A TRC network can deliver a 150-bit message in a 1024 node 32-ary 2-cube with an average latency of $7.5\mu s$, an order of magnitude better performance than would be achieved by a binary n-cube with bit-serial channels. A new routing chip, the Network Design Frame (NDF), improves this latency to $\approx 1\mu$s [12]. The Ametek 2010 uses a 16-ary 2-cube (without end around connections) for its interconnection network [1].

Now that the latency of communication networks has been reduced to a few microseconds the latency of the processing nodes, T_{node}, dominates the overall latency. To efficiently make use of a low-latency communication network we need a processing node that interprets messages with very little overhead. The design of such a *message-driven processor* is currently underway [7] [11].

The real challenge in concurrent computing is software. The development of concurrent software is strongly influenced by available concurrent hardware. We hope that by providing machines with higher performance internode communication we will encourage concurrency to be exploited at a finer grain size in both system and application software.

Acknowledgments

I thank Chuck Seitz of Caltech for his many helpful suggestions during the early stages of this research.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency in part under contract number N00014-80-C-0622 and in part under contract number N00014-85-K0124.

References

[1] Ametek Corporation, Ametek 2010 product announcement, 1987.

[2] Batcher, K.E., "Sorting Networks and Their Applications," *Proceedings AFIPS FJCC,* Vol. 32, 1968, pp. 307-314.

[3] Batcher, K.E., "The Flip Network in STARAN," *Proceedings, 1976 International Conference on Parallel Processing,* pp. 65-71.

[4] Benes, V.E., *Mathematical Theory of Connecting Networks and Telephone Traffic,* Academic, New York, 1965.

[5] Bhuyan, L.N. and Agrawal D.P., "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Transactions on Computers,* Vol. C-33, No. 4, April 1984, pp. 323-333.

[6] Browning, Sally, *The Tree Machine: A Highly Concurrent Computing Environment,* Dept. of Computer Science, California Institute of Technology, Technical Report 3760, 1985.

[7] Dally, William J., *A VLSI Architecture for Concurrent Data Structures,* Kluwer, Hingham, MA, 1987.

[8] Dally, William J. and Seitz, Charles L., *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, IEEE Transactions on Computers,* Vol. C-36, No. 5, May 1987, pp. 547-553.

[9] Dally, William J. and Seitz, Charles L., "The Torus Routing Chip," *J. Distributed Systems*, Vol. 1, No. 3, 1986, pp. 187-196.

[10] Dally, William J. "Wire Efficient VLSI Multiprocessor Communication Networks," *Proceedings Stanford Conference on Advanced Research in VLSI,* Paul Losleben, Ed., MIT Press, Cambridge, MA, March 1987, pp. 391-415.

[11] Dally, William J., "Architecture of a Message-Driven Processor," *Proceedings of the* 14^{th} *ACM/IEEE Symposium on Computer Architecture,* June 1987, pp. 189-196..

[12] Dally, William J., and Song, Paul., "Design of a Self-Timed VLSI Multicomputer Communication Controller," To appear in, *Proc. IEEE International Conference on Computer Design*, 1987.

[13] Kermani, Parviz and Kleinrock, Leonard, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks,* Vol 3., 1979, pp. 267-286.

[14] Lawrie, Duncan H., "Alignment and Access of Data in an Array Processor," *IEEE Transactions on Computers,* Vol. C-24, No. 12, December 1975, pp. 1145-1155.

[15] Leiserson, Charles, L., "Fat Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers,* Vol. C-34, No. 10, October 1985, pp. 892-901.

[16] Mead, Carver A. and Conway, Lynn A., *Introduction to VLSI Systems,* Addison-Wesley, Reading, Mass., 1980.

[17] Pease, M.C., III, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Transactions on Computers,* Vol. C-26, No. 5, May 1977, pp. 458-473.

[18] Pfister, G.F. and Norton, V.A., "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers,* Vol. C-34, No. 10, October 1985, pp. 943-948.

[19] Seitz, Charles L., "Concurrent VLSI Architectures," *IEEE Transactions on Computers,* Vol. C-33, No. 12, December 1984, pp. 1247-1265.

[20] Seitz, Charles L., et al., *The Hypercube Communications Chip,* Dept. of Computer Science, California Institute of Technology, Display File 5182:DF:85, March 1985.

[21] Sequin, Carlo, H., "Single Chip Computers, The New VLSI Building Block," *Caltech Conference on VLSI,* C. L. Seitz Ed. January 1979, pp. 435-452.

[22] Siegel, Howard Jay, "Interconnection Networks for SIMD Machines," *IEEE Computer,* Vol. 12, No. 6, June 1979, pp. 57-65.

[23] Stone, H.S., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers,* Vol. C-20, No. 2, February 1971, pp. 153-161.

[24] Sullivan, H. and Bashkow, T.R., "A Large Scale Homogeneous Machine," *Proc. 4th Annual Symposium on Computer Architecture,* 1977, pp. 105-124.

[25] Tanenbaum, A. S., *Computer Networks,* Prentice Hall, Englewood Cliffs, N.J., 1981.

[26] Thompson, C.D., *A Complexity Theory of VLSI,* Department of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-80-140, August 1980.