VLSI Design
1995, Vol. 2, No. 4, pp. 353–364
Reprints available directly from the publisher
Photocopying permitted by license only

Least Common Ancestor Networks[1]

ISAAC D. SCHERSON and CHI-KAI CHIEN
Department of Information and Computer Science, University of California, Irvine, Irvine, California 92717

Least Common Ancestor Networks (LCANs) are introduced and shown to be a class of networks that include fat-trees, baseline networks, SW-banyans and the router networks of the TRAC 1.1 and 2.0 and the CM-5. Some LCAN properties are stated and the circuit-switched permutation routing capabilities of an important subclass are analyzed. Simulation results for three permutation classes verify the accuracy of an iterative analysis for a randomized routing strategy. These results indicate that the routing strategy provides highly predictable router performance for all permutations. An off-line routing algorithm is also given, and it is shown how to realize certain classes of permutations by adapting Nassimi and Sahni's, and Raghavendra and Boppana's self-routing algorithms for Benes networks.

Key Words: *interconnection networks, permutation routing, SIMD, circuit-switching, randomized routing*

1 INTRODUCTION

Multistage interconnection networks (MINs) have been widely studied for effecting communication between processing elements (PEs) in SIMD and MIMD machines [4, 6, 8, 13, 25]. SIMD communication often takes the form of **permutation routing** where given a unique labelling of the PEs, all source-destination pairs (*s-d* pairs) form a bijection of the set of labels onto itself [2, pg. 333]. MIMD communication consists of a dynamic set of unrestricted *s-d* pairs. Many different MINs have been proposed for handling these communication patterns. Most of them can be derived using a common framework, namely Orthogonal Graphs, as shown in [23].

Routing in these MINs proceeds such that, while routing one *s-d* pair, all stages of the MIN are traversed once and only once. In other words, the MIN is **uni-directional** from a network input to a network output. Most research concerning MINs has examined uni-directional routing. In contrast, **bi-directional** networks have both inputs and outputs associated with the first stage. In addition, once a stage is traversed, it must be traversed again. Such a case may be found in SW-banyans and baseline networks as described in [19] and [27]. Fat-trees [15] and KYKLOS [20] also facilitate bi-directional routing utilizing tree topologies. This routing requires different switch functionality, an example of which is explicitly defined in [15]. Bi-directional routing is also used in the router networks of the TRAC 1.1 and 2.0 [19] and Thinking Machines Corporation's CM-5 [16].

Research on fat-trees has assumed MIMD traffic and examined off-line routing performance [15], randomized on-line performance [11] and deterministic on-line performance [3]. KYKLOS was studied in light of database applications. Simulation of SW-banyans assuming resource allocation strategies that localized MIMD communication was reported in [26]. Very little work, if any, has been invested in the study of bi-directional permutation routing.

In this paper, Least Common Ancestor Networks (LCANs) are defined as a model that helps represent all bi-directional networks and allows the derivation of the networks in [15, 16, 19, 27]. Defining characteristics and properties are given in Section 2.

Another contribution of this paper is the study of the circuit-switched permutation routing capabilities of LCANs. It is well-known from analysis and experimentation that the time needed to route permutations on most uni-directional networks in an on-line fashion (setting switches "on the fly" with local knowledge at each switch) varies greatly with the

[1]This research was supported in part by the Air Force Office of Scientific Research under grant number AFOSR-F49620-92-J-0126, the NASA under grant number NAG-5-1897 and the NSF under grants number MIP-9106949 and number MIP-9205737.

topology of the network and the permutation itself. Experiments have been reported on the MasPar MP-2 router network [1, 5]. On the other hand, LCANs with a randomized routing strategy seem to route all permutations in nearly the same amount of time, providing highly predictable router performance. These results are presented in Section 3. If a permutation is known *a priori,* the off-line routing algorithm given in Section 4 can be used to compute the switch settings necessary to realize the permutation. If only particular classes of "commonly-used" permutations are to be routed, on-line realization (*i.e.,* without any contention) can be accomplished. This is demonstrated in Section 4 with an extension of work by Nassimi and Sahni [21], and Raghavendra and Boppana [22].

LCANs are additionally promising because they are **partitionable,** whereas most unidirectional networks are not (the TRAC and CM-5 computers are partitionable; partitionability of banyan networks was studied in [9]). The network of a partitionable parallel machine can be split into independent sub-networks such that traffic in one sub-network does not affect traffic in the others. As a consequence, the corresponding PEs can be divided among multiple processes. As the number of available PEs in a machine grows, efficient machine utilization will require partitionability since some applications have limited inherent parallelism [7]. The nature of this partitionability lends itself to the construction of **scalable** computers.

2 CHARACTERIZATION

LCANs will be defined in a constructive manner by building on the basic switch definition and the switch interconnection topologies. Referring to Figure 1a, a basic LCAN switch is a crossbar element capable of connecting d inputs on port I_1 to d outputs on port O_1 and/or u out of d inputs on port I_1 to u outputs on port O_2. Conversely, u inputs on port I_2 can be routed to u out of d outputs on port O_1 (without loss of generality, assume $d \geq u$). Ports I_1 and O_1 can be viewed as one set of d bi-directional wires dubbed **downers.** Similarly, I_2 and O_2 can be merged into a set of u bi-directional wired dubbed **uppers** (Figure 1b).

Level i in an LCAN is a disconnected set of S_i switches. Hence, level i contributes dS_i downers and uS_i uppers (Figure 1c).

Finally, an LCAN is defined as interconnected levels. The "wiring" between levels will be referred to as the **level interconnect.** In Figure 1d, an l-level

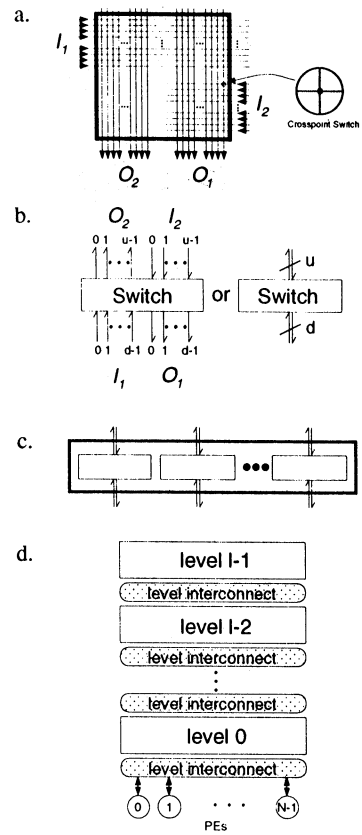

FIGURE 1 LCAN definition: a) switch showing crosspoints, b) switch, c) level of switches and d) LCAN structure.

LCAN is depicted. Note that there are uS_{l-1} uppers out of the top level which were neither drawn nor used. Discussion on the utility of these ports will be deferred until more detailed LCAN examples are presented later in Section 2.2.

An LCAN with N inputs/outputs is parameterized by the tuple (N, d, u, l, Π), where N is the total number of downer ports into the first level (PEs connect to these), the switch size is d by u, l is the total number of levels and Π is a "wiring list" that defines the level interconnects.

Only LCANs with identical switches at all levels are considered. Leiserson's fat-trees are an example of LCANs with potentially different switch sizes at all levels. The careful reader will notice that the parameterization could be extended to LCANs with switch sizes that vary by level.

2.1 LCAN Properties

If U_i denotes the set of all uppers of the switches in some level i, and D_{i+1} is the set of all downers of the switches in level $i + 1$, then Π_{i+1} describes a one-to-one mapping of D_{i+1} onto U_i. As with the switch sizes, only the case in which all levels are

connected by the same generic Π function is considered.

Recall that S_i is the number of switches in level i. Clearly S_0 equals N/d. Moreover, there are uN/d bi-directional connectors going up from level 0 to level 1. Hence, S_1 equals uN/d^2. In general,

$$S_{i+1} = \frac{u}{d} S_i, \text{ where } S_0 = N/d.$$

Recursively substituting,

$$S_i = \frac{N}{d} \left(\frac{u}{d}\right)^i. \quad (1)$$

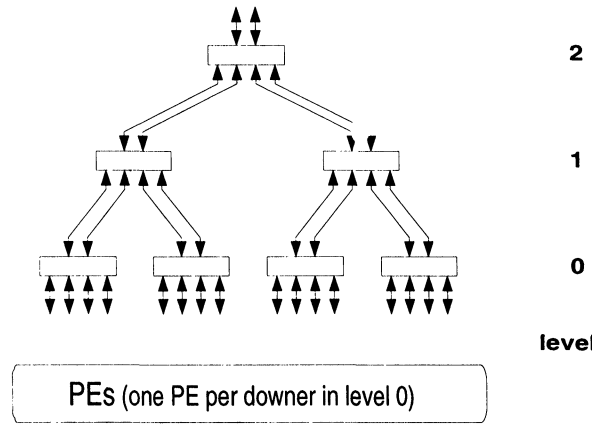

FIGURE 2 Example T-LCAN where $d = 4$ and $u = 2$.

An LCAN is said to be *connected* if for every pair of downer ports in level 0 there exists switch settings such that an electrical connection between the pair can be established.

Theorem 1 *For every connected LCAN(N, d, u, l, Π), l is less than or equal to l_{max}, where l_{max} is the smallest integer such that $N/d\ (u/d)^{l_{max}-1}$ is an integer and $N/d\ (u/d)^{(l_{max})-1}$ is not an integer.*

Proof: Proceed by contradiction. Assume $l > l_{max}$. Then there exist levels $(l_{max} - 1)$ and l_{max}. Using Equation 1, level $(l_{max} - 1)$ has $S_{l_{max}-1} = N/d\ (u/d)^{l_{max}-1}$ switches, and $uS_{l_{max}-1}$ upward connectors to level l_{max}. To fully use every one of these connectors, there need be exactly $u/d\ S_{l_{max}-1}$ switches in level l_{max}. However, $u/d\ S_{l_{max}-1} = N/d\ (u/d)^{l_{max}}$, which is not an integer by assumption. It follows that there must be unused downers in level l_{max}, or unused uppers in level $(l_{max} - 1)$, which is not possible. □ *Q.E.D.*

Note that by Theorem 1, for $d = u$ and some cases of $d < u$, l_{max} is unbounded. However, when $d > u$ there exists some maximum number of levels.

Two particularly interesting classes of LCANs arise from the definition of Π. On the one hand, if the switches are arranged as a (d/u)-ary tree where each edge of the tree represents u bi-directional connectors and d is a multiple of u, Π is dubbed **(d/u)-ary tree** and the LCAN is named **T-LCAN.** An example is shown in Figure 2. Note that d cannot equal u.

On the other hand, Π is dubbed **complete bipartite** if and only if the level interconnectivity is as follows:

Label the PEs consecutively using $log_d N$ base d digits and label the switches in level i consecutively using $(log_d N - 1)$ digits such that the $(log_d N - 1 - i)$ most significant digits are base d and the i least significant digits are base u.

A PE labelled $p_{log_d N-1} p_{log_d N-2} \cdots p_1 p_0$ is connected to switches in the first level labelled $p_{log_d N-1} p_{log_d N-2} \cdots p_1$ via the p_0th downer of the switch.

Consider switches in level i. Removing the rightmost base d digit, w_i, and appending a base u digit, k, to the right of the label, the d switches labelled

$$w_{log_d N-2} w_{log_d N-3} \cdots w_{i+1} w_i w_{i-1} \cdots w_0$$

are connected to the u switches in level $i + 1$ labelled

$$w_{log_d N-2} w_{log_d N-3} \cdots w_{i+1} w_i w_{i-1} \cdots w_0 k$$

via the kth upper of the switch in level i and the w_ith downer of the switch in level $i + 1$.

LCANs with the complete bipartite level interconnect are referred to as **CB-LCANs** (for examples, see Figures 3a and 3b) and are topologically equivalent to SW-banyans. For the special case where d and u equal 2, they are also equivalent to baseline and butterfly networks. In a CB-LCAN, a switch in some level $(i + 1)$ connects to d distinct switches in level i via each and every one of its downers. The CB-LCAN generalizes the definition for bi-directional baseline networks [27] (where $d = u = 2$), and leads to the following lemma:

Lemma 1 *In a CB-LCAN, u^i unique switches in level i are reachable from any downer port at level* 0.

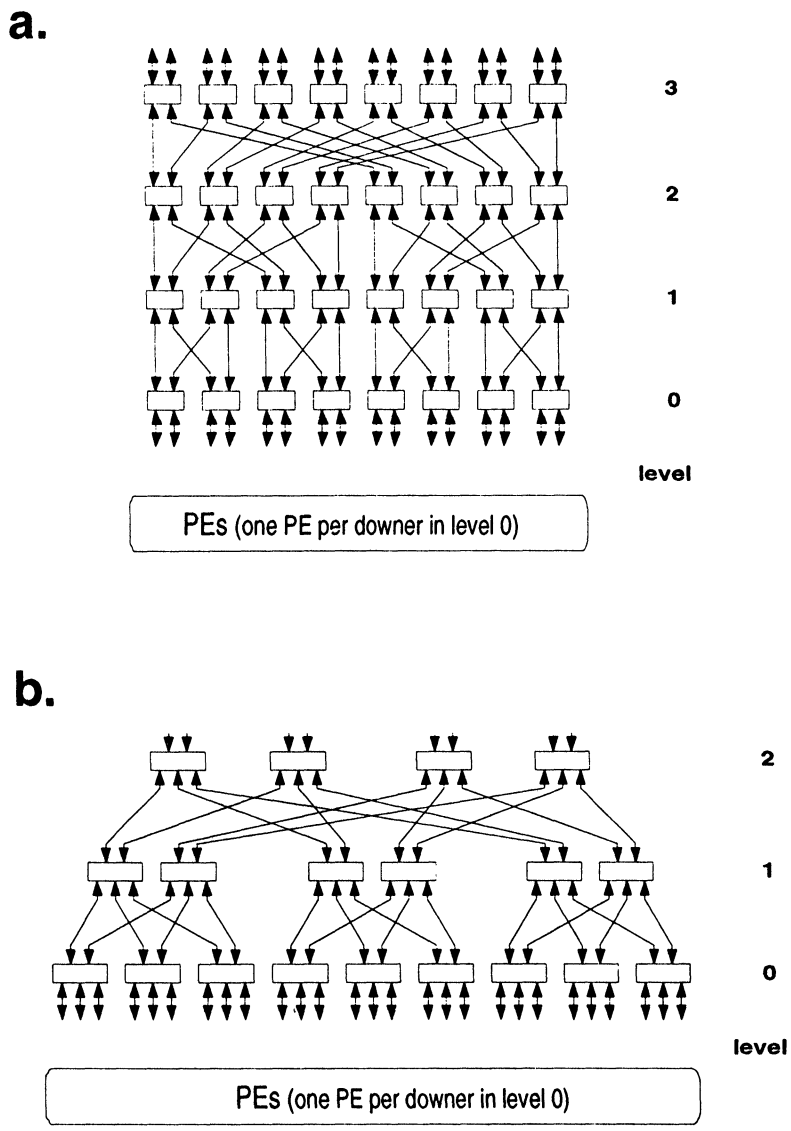

FIGURE 3 a) Example CB-LCAN where $d = u = 2$, b) example CB-LCAN where $d = 3$ and $u = 2$.

Proof: Observe that the u uppers of a switch in level 0 connect to u different switches in level 1 because a single base u digit in the switch label represents u distinct values. Likewise, the u level 1 switches connect to u^2 different switches in level 2 because the two base u digits represent u^2 distinct values. It follows that a processor can connect to u^i distinct switches in level i. □ *Q.E.D.*

At this point, the parameter l for the two classes under consideration can be defined as a function of N, d, and u.

Corollary 1 *In an LCAN (N, d, u, l, complete bipartite),* $l = \log_d N$ $(N = d^l)$.

Proof: By Lemma 1, the number of switches in level $l - 1$ that a processor can reach is u^{l-1}, the total number of switches in that level. From Equation 1, $S_{l-1} = N/d\ (u/d)^{l-1}$. Equating the two,

$$u^{l-1} = \frac{N}{d}\left(\frac{u}{d}\right)^{l-1}.$$

Reducing yields $N = d^l$, or $l = \log_d N$. □ *Q.E.D.*

LCAN (N, d, u, l, complete bipartite) is henceforth parameterized **CB-LCAN(N, d, u).** Figure 3a and 3b are examples a CB-LCAN(16, 2, 2) and a CB-LCAN(27, 3, 2), respectively.

Corollary 2 *In an LCAN(N, d, u, l, (d/u)-ary tree), $l = log_{d/u}(N/u)$ ($N = d^l/u^{l-1}$).*

Proof: By the definition of a T-LCAN, $S_{l-1} = 1$; by Equation 1, $S_{l-1} = N/d\ (u/d)^{l-1}$. Equating the two,

$$\frac{N}{d}\left(\frac{u}{d}\right)^{l-1} = 1.$$

Therefore, $N = d^l/u^{l-1}$, and it follows that $l = log_{d/u} N/u$. □ *Q.E.D.*

LCAN(Nd, u, l, (d/u)-ary tree) is henceforth parameterized **T-LCAN(N, d, u).** Figure 2 is a T-LCAN(16, 4, 2). Note that a T-LCAN where u equals 1 is a limiting case of a CB-LCAN; T-LCAN(N, d, 1) is equivalent to CB-LCAN(N, d, 1).

Note in Figures 2 and 3 that both T-LCANs and CB-LCANs are based on a tree structure. As defined in [12, 24], the **least common ancestor** of two nodes in a tree is the deepest node which counts both nodes among its descendants. Analogously, two PEs communicating using an LCAN need only utilize switches as high as one of their least common ancestor switches.

A **least common ancestor (LCA) switch** of two PEs is a switch in level i, such that: 1) it connects to both PEs through switches in levels 0 through $(i - 1)$, and 2) there is no common ancestor switch in a lower level. Recall from Lemma 1 that two PEs may have more than one LCA switch. The **LCA level** of two PEs is the level to which their LCA switch(es) belong. It can be computed in CB-LCANs simply by finding the most significant digit where two PE labels differ [19].

Lemma 2 *If two processors in a CB-LCAN have LCA level i then they have exactly u^i LCA switches.*

Proof: Assuming i is the LCA level of two PEs, A and B, it follows from Lemma 1 that A is connectable to u^i switches in level i, and B is connectable to u^i switches in level i. Due to the recursive nature of the interconnect, the two sets of switches are either disjoint or identical. If disjoint, then there are no switches in level i that can connect A and B, and thus the LCA level could not be i. Therefore, if identical, the number of LCA switches is u^i. □ *Q.E.D.*

Lemma 3 *Any two processors in a T-LCAN have one and only one LCA switch.*

Proof: In a k-ary tree, where $k = d/u$, given any two leaves (PEs) there is exactly one least common ancestor (LCA switch), the root of the smallest subtree containing both leaves. □ *Q.E.D.*

Let a **path** between two processors consist of an alternating sequence of wire-switch pairs such that a circuit-switched electrical connection is established, and let a **non-redundant switch path** be a path that does not pass through any switch more than once.

Theorem 2 *The number of non-redundant switch paths between any two processors in a CB-LCAN is the same as the number of their LCA switches.*

Proof: Each LCA switch is the root of a subtree of switches, the leaves are PEs. The subtrees share the same leaves due to the recursive nature of the CB-LCAN interconnect. In each subtree, there is exactly one non-redundant switch path between any two leaves. Utilizing switches in levels above an LCA switch does not add additional non-redundant switch paths because the path taken downwards must be the same path as that taken upwards and thus the LCA switch is passed through twice. □ *Q.E.D.*

2.2 Some Well-known LCANs

Topologically-speaking (*i.e.*, disregarding switch capability), some well-known networks introduced in the past can be classified as LCANs. These include, but are not limited to:

- A (f, s, $log_f N$) **SW-banyan** with fanout f and spread s is equivalent to a CB-LCAN(N, f, s). SW-banyans are MINs described in [19] together with bi-directional routing procedures which are given without an explicit switch definition.
- A **baseline** network with full communication capability is equivalent to CB-LCAN(N, 2, 2). In [27], bi-directional routing is called *full communication,* a network's ability to connect a terminal to any terminal on either side of the network. A routing strategy for full communication on a baseline network is described; with these bi-directional routing capabilities, the baseline network is an instance of an LCAN. A baseline network is also an instance of an SW-banyan.
- The **TRAC** network is a CB-LCAN(N, 3, 2). The TRAC 1.1 and 2.0 are reconfigurable, *i.e.*, partitionable, SIMD computers developed at the University of Texas at Austin [19]. They use an SW-banyan network and bi-directional routing; both are capable of packet and circuit-switching. In the TRAC 1.1, the PEs are positioned at the top of the network, and the memories are placed at the bottom. However, in the TRAC

2.0. PEs and memories are all positioned at the bottom.

- A **fat-tree** has the form of a binary tree, but the switch size varies by level; fat-trees are similar to T-LCANs. An exact description requires a vector of d's and u's, a vector element per level. Fat-trees reduce root contention by increasing the "capacity" (bandwidth) of the links between switches closer to the root switch in a binary tree [15]. However, they require switches that proportionally increase in size; for a large number of PEs this can be impractical. Fat-trees are instances of LCANs with switch sizes that vary by level.
- The **CM-5** router network is composed of a CB-LCAN(N, 4, 4) and many CB-LCAN(N, 4, 2)s.
  Thinking Machines Corporation's CM-5 is a MIMD computer that uses a router network called a "4-ary fat tree" [16]. The routing is bi-directional, and the switches have buffers. Both PEs and memories are at the bottom of the network. The first two levels comprise switches with $d = 4$ and $u = 2$; higher up, $d = u = 4$.

2.3 Routing

Lipovski and Malek describe a procedure for bi-directionally routing one s-d pair at a time in SW-banyans [19]. The same procedure is also used in CB-LCANs. And for the specific case of $d = u = 2$, it becomes the procedure for baseline networks given in [27]. For $u = 1$, it is the procedure for KYKLOS given in [20]. Any switch in the source-destination pair's LCA level that is reachable from the source is guaranteed to have a path to the destination (because of the recursive nature of the complete bipartite level interconnect). Note that by Lemma 1, more than one LCA switch may exist for a source-destination pair in a CB-LCAN. The LCA level is determined by finding the position of the most significant digit in which the source and destination labels differ (the labels are in base d). Thus, the basic idea is to route from the source up to the LCA level and then down to the destination.

The routing procedure provides some degree of freedom in that any of the LCA switches can be "chosen" based on random switch setting decisions made at each level while routing up. The path from the chosen LCA switch to the destination is then deterministically established downwards to the destination PE using one base d digit per level from the base d destination label; each digit governs which switch downer is used.

For example, in a CB-LCAN(27, 3, 2) (see Figure 4), the routing between PEs 4 (011) and 18 (200) is specified: the LCA level is 2, thus there are u^2 or four LCA switches. After choosing one by randomly setting level 0 and 1 switches, routing down to PE 18 is accomplished using the destination label, 200. The switch in level i routes to downer j, where j is the d^i-weighted digit of the label.

3 RANDOMIZED PERMUTATION ROUTING

A permutation consists of N s-d pairs that are routed using the aforementioned procedure. There are two types of permutation routing: **on-line** and **off-line.** In on-line routing, individual switching decisions are made "on-the-fly" at each switch, whereas with off-line routing, some central controller first determines the switch settings *a priori* and then sets the switches.

This section describes novel on-line permutation routing algorithms for CB-LCANs. They are first given, then one is chosen for experimental simulation purposes. A detailed probabilistic analysis leads to two interesting recurrences. The iterative solutions of these recurrences are shown to closely approximate simulation results for three permutation classes and a variety of network parameters. The simulations report network performance in terms of mean number of network cycles and the corresponding variance.

All routing algorithms in the remainder of this paper consider the case of traffic patterns that arise when performing synchronous permutations. Circuit-switching is assumed: given a permutation, s-d pairs send messages that compete to establish connections through the network, and route their data. If conflicts arise, decisions are made to drop some messages and successfully route the remaining ones. The process is repeated for the previously unsuccessful pairs until all pairs are routed. Routing each successful subset is said to require one network cycle.

Algorithms

An LCAN permutation routing algorithm can be broken into two phases: route upwards to an LCA switch and downwards to the final destination. Routing upwards, on the way to an LCA switch, d down-

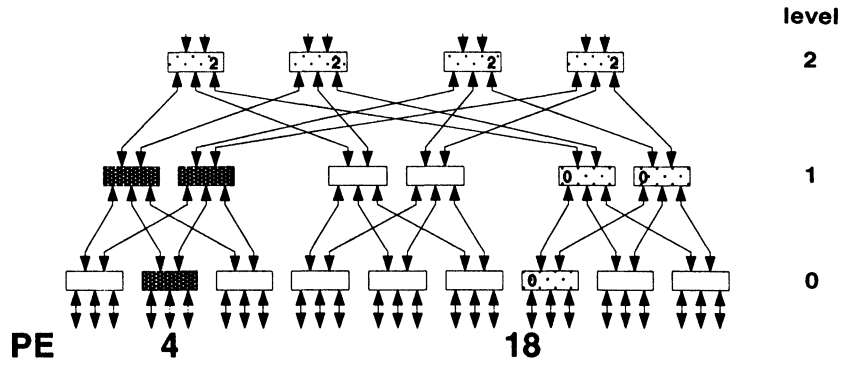

FIGURE 4 Example routing between PEs 4 and 18 in a CB-LCAN(27,3,2). The darkly-shaded switches are set randomly, then the chosen lightly-shaded switches are set deterministically. Note that no matter which LCA switch is chosen, routing down using the destination label 200 results in a connection to PE 18.

ers must be mapped to u uppers at each intermediate switch. These choices can be made randomly per switch each network cycle, randomly chosen per switch but fixed per permutation, and fixed per switch and network incarnation. Downwards, the path from the LCA switch to the final destination is unique, and contention between *s-d* pairs can be resolved in one of several ways. These include: randomly, priority to pairs with lower LCA levels, and priority to pairs with higher LCA levels.

Permutation classes

The three permutation classes considered are: root, bit-permute-complement (BPC) and random permutations. All *s-d* pairs of **root permutations** require routing through the switches in level $(l - 1)$ (the "root" level), where each switch is the root of a tree of switches. Bit-permute-complement (BPC) permutations are "commonly-used" in many applications [17, 21]. Random permutations give an average-case performance for all $N!$ permutations. In all cases, probabilistic analysis is given because of the randomized nature of the upwards routing.

Analysis of root permutations

Consider root permutations and focus on the number of *s-d* pairs that succeed in a network cycle as a function of the original number of attempting pairs. A sequence of network cycles can then be derived, leading to the predicted number of cycles necessary to route the entire permutation.

Assume that d equals u and that the routing strategy upwards is random. Thus all *s-d* pairs of a root permutation can route to the highest level without contention. The permutation of destination labels in the root level is dependent on the original permutation at the bottom level. However, by assuming independence between these two permutations, finding the throughput of the bi-directional routing reduces to the problem of finding the throughput for the unidirectional routing from the root level down the CB-LCAN to its leaves.

Leighton solves a recurrence for the probabilistic analysis of throughput assuming **random routing** in a butterfly network [14, pg. 615]. Destinations are assumed to be uniformly and independently distributed. The same reasoning can be generalized to unidirectional permutation routing in CB-LCANs as follows: consider a switch in level i. Let p_i be the probability that a *s-d* pair is routing down on one of its uppers. Then, p_{i-1} is the probability that one or more incoming packets from the uppers want to utilize a particular downer. Given a particular downer, δ:

$$p_{i-1} = 1 - \text{P[no pair wants } \delta]$$

$$\text{P[no pair wants } \delta] = (\text{P[pair doesn't want } \delta])^u$$

$$\text{P[pair doesn't want } \delta] = 1 - p_i \cdot \text{P[pair wants } \delta]$$

$$\text{P[pair wants } \delta] = 1/d.$$

Note that the subscript ordering decreases as the recurrence progresses since the permutation has now routed to the top of the network. The initial load is p_{l-1} and the final probability p_0 is the throughput. This leads to the following recurrence:

$$p_{i-1} = 1 - \left(1 - \frac{p_i}{d}\right)^u. \tag{2}$$

During the first network cycle, p_{l-1} is 1.0. Np_0 is the average number of successful *s-d* pairs in a given cycle. To adapt this recurrence to permutation routing, it is assumed that there is no conflict in the last level since no two PEs have the same destination in

a permutation. Hence, Equation 2 is iterated to compute p_1. The following recurrence uses p_1 from each network cycle to determine the number of pairs remaining to be routed:

$$x_i = x_{i-1} - Np_1, \text{ where } x_0 = N. \quad (3)$$

In turn, each x_i is used to compute the initial load, p_{l-1}, for the next network cycle. Equation 2 is a nonlinear recurrence which to the authors' knowledge has no closed-form solution, thus the two recurrences are solved iteratively. The iteration terminates when x_i becomes less than one. The computed number of cycles is $(i - 1 + x_i)$ because an extra pass is required with probability x_i.

To extend this analysis to LCANs with d not equal to u, it is assumed that the s-d pairs are uniformly distributed among the $N(u/d)^{l-1}$ downers of the highest level when computing the initial load, p_{l-1}. Furthermore, when $d > u$, the load is assumed to be 1.0 when the number of pairs remaining is greater than or equal to $N\,(u/d)^{l-1}$. Equations 2 and 3 are iterated with these additional assumptions.

Recall that this analysis is solely for root permutations. The problem of analyzing other permutation classes, namely BPC permutations, appears intractable, thus simulations were performed.

Simulation results

Consider the root, BPC and random permutation classes and assume a permutation routing algorithm that routs up randomly per switch each network cycle, and downwards giving priority to s-d pairs with lower LCA levels. 1000 randomly generated permutations per permutation class were simulated for each CB-LCAN tuple considered, and the mean and variance for each set was computed. The theoretical predictions from the above analysis for root permutations are graphed with the simulation results for comparison and validation in Figures 5, 6 and 7.

Figure 5 shows the performance tradeoff when N and d are fixed and u decreases. Depending on the cost of an LCAN switch, choices of d and u where $d > u$ might be made to attain a desired performance. For example, as u decreases from 64 to 16, the total number of wires and crosspoints decreases by approximately a factor of four, whereas the performance only decreases by a factor of two.

Figure 6 assumes a fixed N and shows the performance tradeoff using different size switches. As the switch size decreases, the number of levels increases, however, changing from $d = u = 64$ switches (2 levels) to $d = u = 2$ (12 levels) only causes a factor of two drop in performance. This may be important to keep in mind for building truly massively parallel systems with optical interconnects since currently feasible optical switches are very limited in size [10].

Figure 7 indicates that assuming a fixed switch size, the performance degrades linearly with respect to $f(d, u) \cdot \log_d N$, where the values of $f(d, u)$ are close to one in all the simulations. For example, while the number of processors increases from 1024 to 4096, the mean number of passes increases by less than one.

From all three figures, it can be concluded that the theoretical solutions for root permutations closely approximate the simulation results, hence validating the simulations. Moreover, they approximate the mean number of cycles for random and BPC permutations.

Although mean performance is important, from an applications standpoint *predictable* router performance is perhaps equally important [5]. The low variances reported in the simulations, ranging from 0.02 to 0.28 when $d = u$, suggest that the performance of CB-LCANs for *all* permutations is highly predictable. In contrast, simulation of BPC permutations on the MasPar MP-2 router network (assuming one PE per network input and 1K network inputs) generate a variance that is relatively high-4.24, with a mean of 3.51 cycles (simulating 1000 random BPC permutations) [1].

The performance of CB-LCANs is predictable because the random upwards routing per switch at each network cycle (which does not occur in uni-directional routing), tends to "equalize" all permutations. Consider uni-directional routing. There is some worst-case permutation, or class of permutations, that takes the most number of cycles to route. When routing bi-directionally in a CB-LCAN, the "worst-case" occurs only if the random choices made by the routing algorithm result in a "worst-case" permutation at the top of the network in *every* network cycle. Whether or not "worst-case" permutations exist for LCANs is an open question currently being pursued.

This randomization could explain why root permutations routed more quickly on average than random permutations. S-d pairs in root permutations route to the highest level, incurring more random choices than the typical random permutation, *i.e.*, more randomization. BPC permutations tend to route more quickly than random permutations because some BPC permutations are also root permutations, and some represent communication patterns that would occur if the network were partitioned. In a

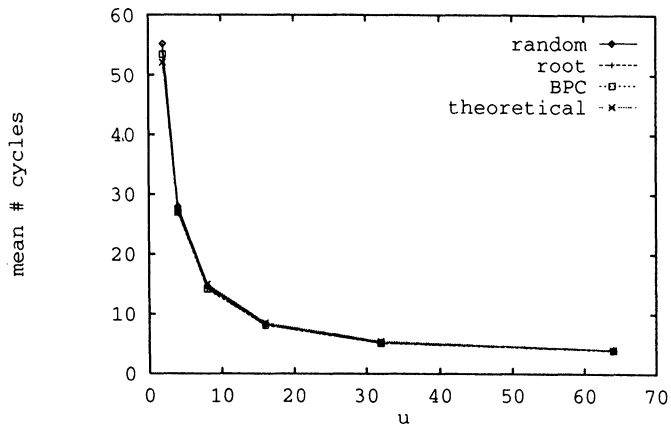

FIGURE 5 Fix $N = 4096$ and $d = 64$ and vary u.

partitioned network, a number of equal-sized smaller networks are effectively acting in parallel, thus less cycles are necessary (see Figure 7).

4 PERMUTATION REALIZATION

It is first shown how to realize permutations off-line in CB-LCANs. Then, an extension of prior work for on-line realization of "commonly-used" permutation classes is given.

4.1 Off-line Realization

Off-line routing algorithms typically yield routings that require fewer cycles than on-line algorithms, however there is the overhead cost of pre-computing the switch settings. Thus, an off-line algorithm is useful when the time gained due to the better routing offsets the pre-compute time. This can occur at compile-time when the permutation is known *a priori,* and at run-time if it is known that a permutation will be used often enough during the course of execution.

Lev, Pippenger and Valiant (LPV) developed an off-line routing algorithm to realize permutations in Benes-like networks, *i.e.,* recursively constructed networks with arbitrary switch size [18]. Their algorithm finds an intermediate permutation at the middle stage that the first half of the network can realize by self-routing, say $s - s'$, and the second half finishes the original permutation by self-routing $s' - d$.

The same algorithm can be applied to CB-LCANs since CB-LCANs have greater functionality than Benes-like networks. This functionality can be seen by "unfolding"an LCAN:

Let A be an LCAN (see Figure 8a); let A_1 comprise the PEs, the switches and the upward-going wires of the bi-directional links. Let A_2 comprise the downward links of the bi-directional connectors, and duplicates of the PEs and the switches (except those

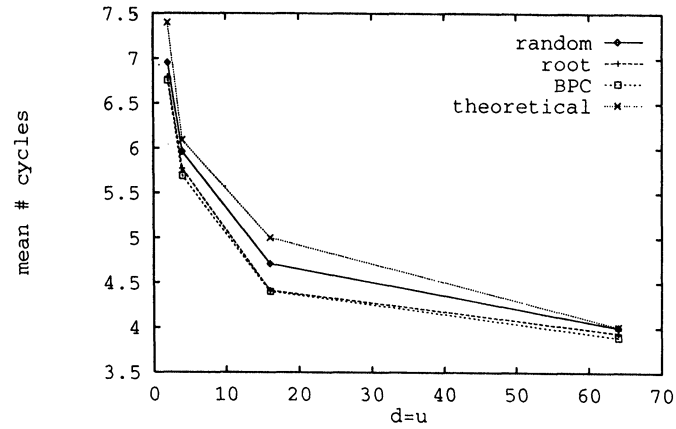

FIGURE 6 Fix $N = 4096$ and vary $d = u$.

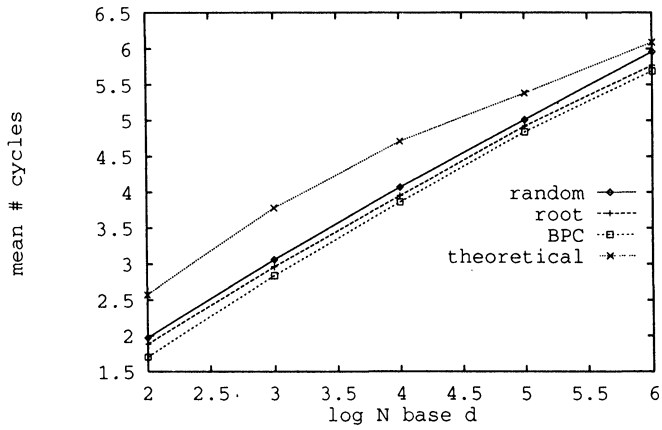

FIGURE 7 Fix $d = u = 4$ and vary N.

in level $l - 1$). Unfold A_2 in a mirror-like fashion (see Figure 8). Each switch in A_1 effectively has an extra link to its corresponding mirrored switch in A_2 (represented by the thick edges in Figure 8b). These are virtual links and are costless to traverse. The added functionality provided by the virtual links allows certain connections to be made without utilizing a switch in every level.

Thus in a CB-LCAN, the LPV method can be applied by forcing all *s-d* pairs to route to the highest level: self-route to ports in the highest level using the intermediate permutation s', then self-route to the ports in the lowest level using the original destinations d. Once the intermediate permutation is determined, the routing can be easily modified so that *s-d* pairs route only as high as their LCA levels, thus preserving partionability. Upon reaching their LCA level, *s-d* pairs simply route back down because any path taken further upwards must also be taken down (see proof of Theorem 2).

For CB-LCANs where $d = u$ and $d < u$, the application of the method is direct and all permutations are routed in one cycle. Where $d > u$ and d is a multiple of u, $\lceil (d/u)^{l-1} \rceil$ embeddings of the CB-LCAN are chosen in the corresponding Benes-like network comprising $d \times d$ switches, and $\lceil (d/u)^{l-1} \rceil$ cycles are required.

Theorem 3 *Any permutation may be routed off-line in a CB-LCAN(N, d, u) in* $\lceil (d/u)^{l-1} \rceil$ *network cycles* ($l = \log_d N$), *where* $d \leq u$ *or* d *is a multiple of* u.

Proof: As a CB-LCAN is unfolded, a Benes-like network is seen. Routing in the CB-LCAN (Benes-like network) requires identifying the intermediate permutation at the highest level (middle stage). There are three cases:

1. $d = u$—The unfolded CB-LCAN(N, d, d) is isomorphic to the Benes-like network with $d \times d$ switches. This isomorphism fixes the switch labelling in the first, middle and last levels (stages) of the unfolded CB-LCAN (Benes-like network). Thus, given a permutation, the intermediate permutation generated by the Lev *et al.* algorithm for the Benes-like network can be used as is for the CB-LCAN.
2. $d < u$—The CB-LCAN(N, d, u) embeds a CB-LCAN(N, d, d), therefore the same algorithm can be applied directly.

   In both cases, exactly one network cycle is required (if $d \geq u$, $\lceil (d/u)^{l-1} \rceil$ equals 1).

3. $d > u$—It is sufficient to find embeddings of the CB-LCAN(N, d, u) in the CB-LCAN(N, d, d) such that all of the switches in the top level of the CB-LCAN(N, d, d) are in one or more of the embeddings. The *s-d* pairs routed within an embedding do not conflict and together form the original permutation.

   Utilizing the LPV algorithm on the CB-LCAN(N, d, d), the intermediate permutation at level $l - 1$ of each of these embeddings defines the intermediate permutation to be routed in one network cycle. If d is a multiple of u, there are $(d/u)^{l-1}$ embeddings each with u^{l-1} switches in level $l - 1$, since there are N/d switches in level $l - 1$ of the CB-LCAN(N, d, d).

   Labelling the level $l - 1$ switches $0, 1, \ldots, (N/d) - 1$, each set, g_i, of consecutive u^{l-1} switches are grouped as follows: $g_0 = \{0, 1, \ldots, (u^{l-1} - 1)\}$, $g_1 = \{u^{l-1}, u^{l-1} + 1, \ldots, (2u^{u-1} - 1)\}$, etc. Each g_i contains the level $l - 1$ switches of one embedding because of the

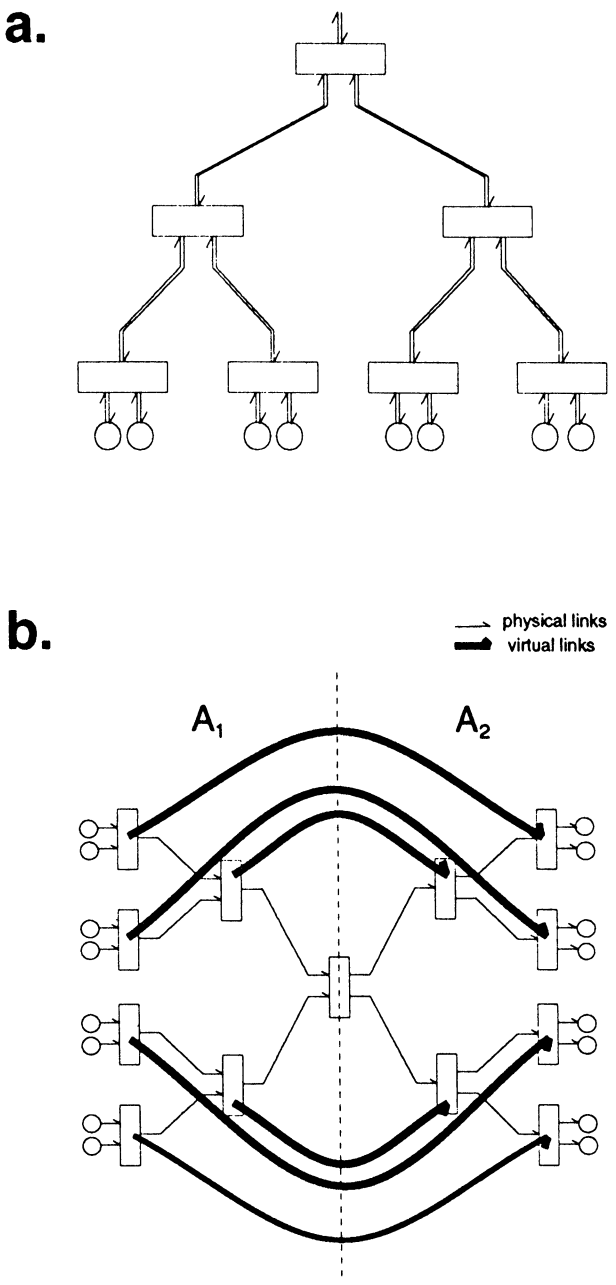

FIGURE 8 Unfolding an LCAN: a) A, b) A unfolded into A_1 and A_2 showing virtual links.

recursive definition of CB-LCANs. The *s-d* pairs routed through each g_i gives the intermediate permutation for an embedding.

□ *Q.E.D.*

Note that $\lceil (d/u)^{l-1} \rceil$ cycles is not the minimum number necessary for *any* permutation. For example, the identity permutation requires exactly one cycle in any LCAN.

4.2 On-line Realization

When it is known *a priori* that only certain classes of "commonly-used" permutations are to be routed, two on-line routing algorithms may be used. Nassimi and Sahni developed a self-routing algorithm that realizes a family of permutations in Benes networks [21]. This family, $F(n)$, includes BPC(n) and inverse $\Omega(n)$ permutations. In the algorithm, each switch simply gives priority to the lower-numbered input. Raghavendra and Boppana have an algorithm that realizes linear permutations in Benes networks by giving priority to the input with the smaller destination tag [22]. In a linear permutation, each bit of the destination tag is some linear combination of the source tag bits.

Since an unfolded CB-LCAN(N, 2, 2) without its virtual links is isomorphic to a Benes, both of these algorithms can be directly applied. Furthermore, *s-d* pairs may route back down upon reaching their LCA level since any path taken further upwards must also be taken down (see proof of Theorem 2).

As noted in [22], both algorithms can be extended to all networks with switch sizes that are a power of two. Each larger switch simulates an equivalent size network of 2 × 2 switches and switches appropriately. The same can be done for CB-LCANs.

References

[1] B.D. Alleyne, *Personal communication,* Princeton University, 1993.
[2] G.S. Almasi and A. Gottlieb, *Highly parallel computing,* Benjamin/Cummings, Redwood City, CA, 1989.
[3] P. Bay and G. Bilardi, *Deterministic on-line routing on area-universal networks,* 31st Annual Symp. on Foundations of Computer Science, 1990, pp. 297–306.
[4] V.E. Benes, *Mathematical theory of connecting networks and telephone traffic,* Academic, New York, 1965.
[5] T. Blank and R. Tuck, *Personal communications,* MasPar Computer Corporation, 1992.
[6] C. Clos, *A study of non-blocking switching networks,* Bell System Technical Journal, Vol. 32, 1953, pp. 406–424.
[7] R. Cyper, A. Ho, S. Konstantinidou and P. Messina, *Architectural requirements of parallel scientific applications with explicit communication,* Int'l, Symp. Comp. Arch., 1993, pp. 2–13.
[8] T. Feng, *A survey of interconnection networks,* Computer, Vol. 14, December 1981, pp. 12–27.
[9] L.R. Goke and G.J. Lipovski, *Banyan networks for partitioning multiprocessor systems,* Proc. 1st Annual Symp. on Comp. Arch., 1973, pp. 21–28.
[10] P.E. Green Jr., *Fiber optic networks,* Prentice-Hall, 1993.
[11] R. Greenberg and C.E. Leiserson, *Randomized routing on fat-trees,* Proc. of 26th Annual Symp. on Foundations of Computer Science, 1985, pp. 241–249.
[12] D. Harel and R.E. Tarjan, *Fast algorithms for finding nearest common ancestors,* SIAM J. Comput., Vol. 13, No. 2, May 1984, pp. 338–355.
[13] K. Hwang and F. Briggs, *Computer architecture and parallel processing,* McGraw-Hill, 1984.

[14] F.T. Leighton, *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes,* Morgan Kaufmann Publishers, 1992.

[15] C. Leiserson, *Fat trees: Universal networks for hardware-efficient supercomputing,* IEEE Trans. on Comput., Vol. C-34, No. 10, October 1985, pp. 892–901.

[16] C. Leiserson, *et al., The network architecture of the connection machine CM-5,* SPAA, June 1992, pp. 272–285.

[17] J. Lenfant, *Parallel permutations of data: A Benes network control algorithm for frequently used permutations,* IEEE Trans. on Comp., C-27, July 1978, pp. 637–647.

[18] G.F. Lev, N. Pippenger and L.G. Valiant, *A fast parallel algorithm for routing in permutation networks,* IEEE Trans. on Comput., Vol. C-30, No. 2, February 1981, pp. 93–100.

[19] G.J. Lipovski and M. Malek *Parallel computing,* Wiley & Sons, 1987.

[20] B.L. Menezes and R. Jenevein, *The KYKLOS multicomputer network: Interconnection strategies, properties, and applications,* IEEE Trans. on Comput., Vol. 40, No. 6, June 1991, pp. 693–705.

[21] D. Nassimi and S. Sahni, *A self-routing benes network and parallel permutation algorithms,* IEEE Trans. on Comput., Vol. C-30, May 1981, pp. 332–340.

[22] C.S. Raghavendra and R.V. Boppana, *On self-routing in benes and shuffle-exchange networks,* IEEE Trans. on Comput., Vol. 40, No. 9, September 1991, pp. 1057–1064.

[23] I.D. Scherson, *Orthogonal graphs for the construction of a class of interconnection networks,* IEEE Trans. on Parallel and Distr. Sys., Vol. 2, No. 1, January 1991, pp. 3–17.

[24] B. Schieber and U. Vishkin, *On finding lowest common ancestors: simplification and parallelization,* SIAM J. Comput., Vol. 17, No. 6, December 1988, pp. 1253–1262.

[25] H.J. Siegel, *Interconnection networks for large-scale parallel processing,* Lexington Books, 1985.

[26] W.L. Warren, *A performance evaluation of banyan interconnection networks,* Master's thesis, Colorado State University, 1978.

[27] C.W. Wu and T. Feng, *On a class of multistage interconnection networks,* IEEE Trans. on Comput., Vol. C-29, No. 8, August 1980, pp. 694–702.

Biographies

ISAAC D. SCHERSON is currently a Professor in the Dept. of Information and Computer Science at the University of California, Irvine. Dr. Scherson was a member of the Technical Program Committee for several conferences. He served as Workshops Chair for Frontiers' 92 and is the editor of the resulting book. He is also co-editor of an IEEE Tutorial on Interconnection Networks. In July 1992, he was appointed to the IEEE Computer Society Technical Committee on Computer Architecture. His research interests include massively parallel computer architectures, interconnection networks, associative memory and processing, computer graphics, and algorithms and their complexity.

CHI-KAI CHIEN is a graduate student in the Dept. of Information and Computer Science at the University of California, Irvine. He received his B.S. in EECS and M.S.E. in CS from the Johns Hopkins University in 1988 and 1989, respectively. His research interests include interconnection networks, routing algorithms, and the design of massively parallel computers.