Lowest Common Ancestor Interconnection Networks

Brian D. Alleyne[1], Chi-Kai Chien[2] and Isaac D. Scherson[2]

[1]Department of Electrical Engineering
Princeton University
Princeton, NJ 08544

[2]Department of Information and Computer Science
University of California
Irvine, CA 92717

Technical Report #92–19

February 14, 1992

Lowest Common Ancestor Interconnection Networks *

Brian D. Alleyne[1], Chi-Kai Chien[2] and Isaac D. Scherson[2]

[1]Department of Electrical Engineering
Princeton University
Princeton, New Jersey 08544

[2]Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717

Abstract

Lowest Common Ancestor (LCA) networks are built using switches capable of connecting $u + d$ inputs/outputs in a permutation pattern. For n source nodes and l stages of switches, $\frac{n}{d}$ switches are used in stage $l-1$, $\frac{n}{d} \cdot \frac{u}{d}$ in stage $l-2$, and in general, $\frac{n \cdot u^{l-i-1}}{d^{l-i}}$ switches in stage i. The resulting hierarchical structure possesses interesting connectivity and permutational properties. A full characterization of LCA networks is presented together with a permutation routing algorithm for a family of LCA networks. The algorithm uses the network itself to collect and disseminate information about the permutation. A schedule of $O(dp\ log_{d/u}\ n)$ passes is obtained with a switch set-up cost factor of $O(log_{d/u}\ n)$ (p is the minimum number of passes that an algorithm with global knowledge schedules).

Keywords: SIMD, interconnection network, tree, permutation routing

*This research was supported in part by the Air Force Office of Scientific Research under grant number AFOSR-90-0144 and NSF under grant number MIP9106949.

1 Introduction

There have been many multistage interconnection networks (MIN's) proposed [1], [4], [6], [14]. These networks can be used to effect communication between processing elements (PE's) in SIMD machines, as well as MIMD machines. SIMD communication takes the form of *permutation routing* where given a unique labelling of the PE's, source-destination pairs are obtained from a one-to-one mapping of the set of labels onto itself. Past research suggests that determining switch settings to route permutations in MIN's capable of routing any permutation takes at least $O(n\ log\ n)$ sequential time [11], [13], [15]. To reduce this set-up cost, it is important to analyze networks that could potentially take many passes to route any permutation; in each pass, a subset of the permutation's connections (source-destination pairs) are set up and data is transmitted.

In this paper, a full characterization of *Lowest Common Ancestor (LCA)* interconnection networks is presented together with a permutation routing algorithm for a family of LCA networks. LCA networks are built using switches capable of connecting $u+d$ inputs/outputs in a permutation pattern. For n source nodes and l stages of switches, $\frac{n}{d}$ switches are used in stage $l-1$, $\frac{n}{d} \cdot \frac{u}{d}$ in stage $l-2$, and in general, $\frac{n \cdot u^{l-i-1}}{d^{l-i}}$ switches in stage i. The resulting hierarchical structure possesses interesting connectivity and permutational properties. The permutation routing algorithm uses the network itself to collect and disseminate information about the permutation. A schedule of $O(dp\ log_{d/u}\ n)$ passes is obtained with a switch set-up cost factor of $O(log_{d/u}\ n)$ (p is the minimum number of passes that an algorithm with global knowledge schedules).

LCA networks solve some implementation problems for SIMD computers. For instance, in the MasPar MP-1, a massively parallel computer, there are many PE's on each chip and there are many chips per board and boards are connected through a backplane [2]. Connections going through the backplane are expensive in terms of connection time and the number of edge-connector pins. Using traditional MIN's, the routing of all permutations requires utilizing one switch in every stage; thus, all connections must go through the backplane. However, with LCA networks, every connection need not utilize switches in every stage; LCA networks exploit the inherent hierarchy in the hardware structure: permutations requiring only on-chip communication can be performed the quickest, then permutations which require on-board communication and lastly, permutations which require communication across the backplane.

Previous work uses trees as a model of communication, either by employing a tree of processors, *e.g.* X-tree, the DAC, and the P-tree [5], [8], [9], [10], or a tree of switches as an interconnection network between processors, *e.g.* Hypertree and KYKLOS [7], [12]. LCA networks fall into the latter category. Trees are attractive to use as interconnection networks because they are highly scalable and require a number of switches that is linear with respect to the number of PE's. However, trees are unappealing because of the high degree of contention near the root of the tree. Hypertree and X-tree are variations upon complete binary trees, both networks add links between nodes on the same level to reduce the average interprocessor distance. KYKLOS comprises replications of k-ary trees; the isomorphic replications provide alternate paths, again reducing the average interprocessor distance. In [5], [7] and [12], X-tree, Hypertree and KYKLOS were analyzed in a MIMD context, *i.e.* they have strategies for PE-to-PE routing, not permutation routing.

Section 2 formally defines and characterizes LCA networks, shows when they are fully connected and discusses how they are related to MIN's with identical switches. Section 3 explains the permutation routing algorithm, and section 4 concludes and discusses future research directions.

2 Characterization of LCA's

In an LCA network, each stage of switches corresponds to a level in the network. Switches can communicate with both their children and parent(s), *i.e.* information can be sent up and down the network (see Figure 3), using their bi-directional *uppers* and *downers*, connectors which facilitate communication up and down the network, respectively.

In a tree, the lowest common ancestor of two nodes is the node at greatest depth which counts both nodes among its descendants. Two PE's communicating in an LCA network need only utilize switches as high as their lowest common ancestor (a switch).

2.1 Switch Description

Each switch has d bi-directional links which connect to switches in next lower level (labeled $1, 2, \cdots, d$) called *downers*, and u bi-directional links which connect to switches in the next higher level $(1, 2, \cdots, u)$ called *uppers* (see Figure 1).

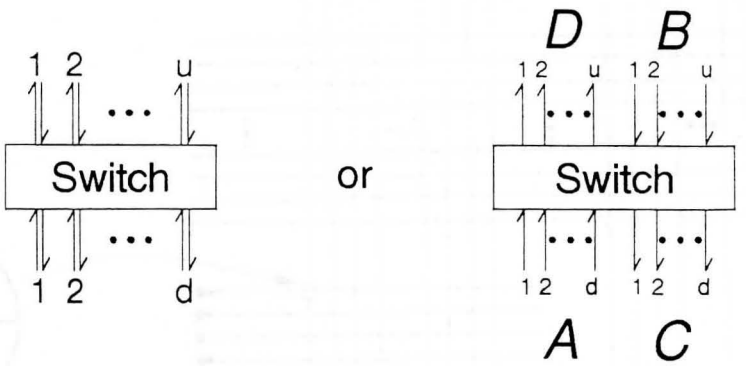

Figure 1: An LCA switch

Each bi-directional link actually represents two wires, each wire carrying information in a opposite directions. Let the uppers be partitioned into two groups, B and D, which represent wires carrying information to and from the switch respectively and the downers be also partitioned into two groups, A and C, which represent wires carrying information to and from the switch. In order to achieve the connections between the groups A, B, C, D, they need to be connected by crosspoint switches detailed in Figure 2. In Figure 2 each intersection of two lines represents a crosspoint.

The switch operates in two modes. In the first mode, data on the inputs B and A are used to set the internal crosspoints of the switch. How this data is actually interpreted by the switch in order to setup the internal crosspoints is dependent on the routing strategy, and is outlined later as different routing strategies are considered. In the second mode, information is passed through the switch using the internal crosspoint setup.

2.2 Parameter Description

LCA networks are parameterized by $(u, d, n, l, \overline{SP})$-tuples. There are n PE's, and l levels (stages) of switches in the network. $\overline{SP}$ is a vector of size l, each vector element describing the permutations between stages, the manner in which the switches in adjacent stages are connected. Each switch in level i, where $0 \leq i \leq l-1$, has d bi-directional downers, links which connect to switches in the next lower level, level $i+1$, and u bi-directional uppers which connect to switches in the next higher level, level $i-1$ (see Figures 1 and 3). Thus, switches in level i are only connected to switches in levels $i-1$ and $i+1$. The exceptions are level 0 (the highest level in the network), in which case

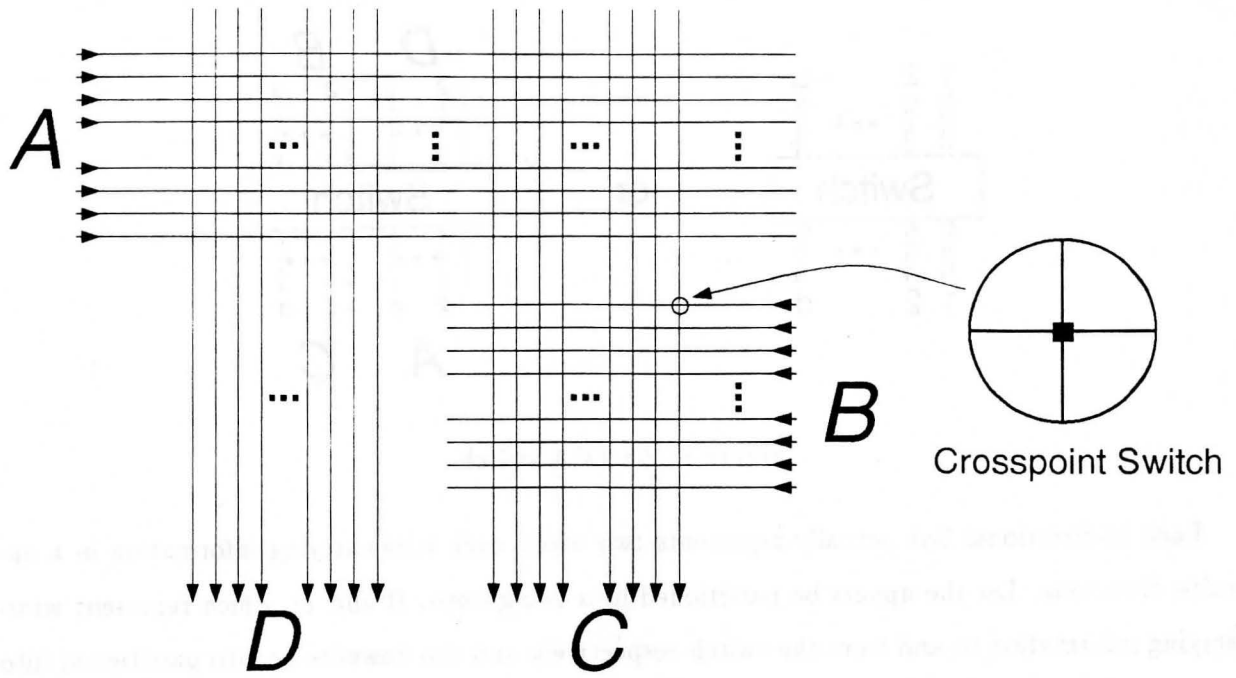

Figure 2: LCA switch showing crosspoints

the uppers do not connect to anything, and level $l-1$ (the lowest level in the network), in which case the downers connect to the PE's. A switch accepts up to $u+d$ inputs and switches them to $u+d$ outputs. Any upper and downer can connect to any upper or downer, including itself.

When $d > u$, the number of switches per stage decreases as the number of stages grows; $d = u$, the number of switches per stage is constant; $d < u$, the number of switches per stage increases as the number of stages grows. In this paper, we are primarily interested in tuples where $d \geq u$ since they represent networks that are highly scalable.

When $u = 1$, the network is a d-ary tree. Figure 4 shows $(2, 3, 9, 2, 2-3$ *bipartite*), an LCA network that has two levels of switches, a bottom level comprising nine PE's and a stage permutation that connects every three switches in level i with two switches in level $i-1$ with a 2-way shuffle (denoted *2-3 bipartite*).

2.3 Full Connection

Certain tuples represent *fully connected* networks, *i.e.* any two PE's can communicate using the network and all switch downers and uppers, except those at the highest level, are utilized. These

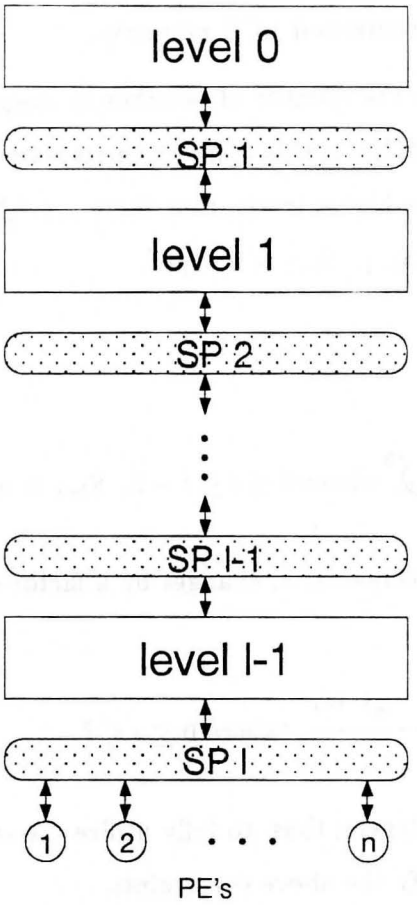

Figure 3: The LCA network structure

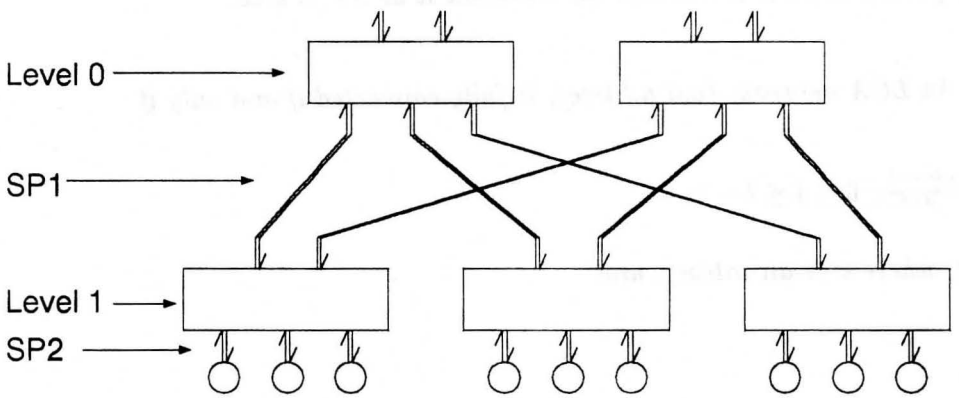

Figure 4: An example LCA network: $(2,3,9,2,2-3$ *bipartite*$)$

tuples are of interest because the networks they describe are regular. We first make some observations on the number of switches required per level in order to utilize all uppers and downers, and then prove a theorem about fully connected LCA networks.

Let S_i, where $0 \leq i \leq l-1$, be the number of switches in stage i. Given that n is the number of PE's, $S_{l-1} = \frac{n}{d}$ since each switch has d pins connecting to lower levels. The $\frac{n}{d}$ switches in stage $l-1$ each have u pins connecting to higher levels, thus there are $\frac{u \cdot n}{d}$ pins going up from stage $l-1$. Repeating the same reasoning for S_{l-1}, $S_{l-2} = u \cdot n/d^2$.

In general,

$$S_i = \frac{u}{d} S_{i+1}, \text{ where } 0 \leq i \leq l-2, \; S_{l-1} = n/d. \tag{1}$$

Recursively substituting, as i decreases, S_i changes by a factor of u/d:

$$S_i = n \cdot \frac{u^{l-i-1}}{d^{l-i}}, \text{ where } 0 \leq i \leq l-1. \tag{2}$$

The S_i's, u, d, l and n are all integers; thus, to fully utilize the switch pins, we are interested in finding parameter values that satisfy the above constraints.

LCA networks where $d > u$ and all uppers of a switch in stage i connect to one switch in stage $i-1$ resemble (d/u)-ary trees, each link consisting of u bi-directional links. In this event, we say that the stage permutation is a tree and we represent it as $\overline{SP} = tree$.

Theorem 1 *An LCA network, (u,d,n,l,tree), is fully connected if and only if*

1. $S_i = n \cdot \frac{u^{l-i-1}}{d^{l-i}}$, $0 \leq i \leq l-1$,
2. $u = k \cdot d$, *where* k *is an integer, and*
3. $S_0 = 1$.

Proof:

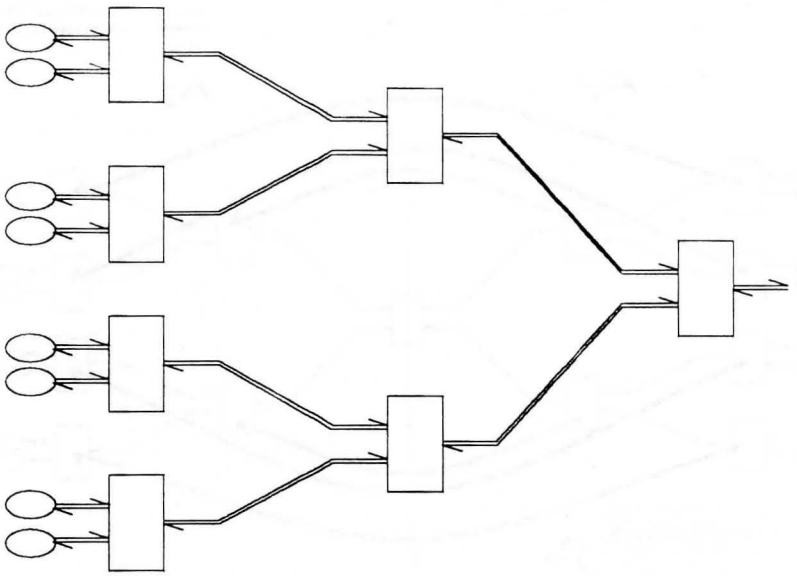

Figure 5: An LCA network, A

($\Rightarrow$) Property 1 follows from (1). Since S_i must be integer for $0 \leq i \leq l-1$, property 2 must hold. If $S_0 = j$, $j > 1$, then the network would consist of j disjoint subtrees by the definition of $\overline{SP} = tree$. Thus, $S_0 = 1$, property 3.

($\Leftarrow$) Properties 1 and 2 insure that the uppers of stage i and the downers of stage $i-1$, $1 < i < l-1$, are fully utilized. The downers of stage $l-1$ are fully utilized by the n PE's, and the uppers of stage 0 are not utilized. $S_0 = 1$ guarantees that subtrees of the topmost switch are connected, as per the definition of $\overline{SP} = tree$. $\square Q.E.D.$

2.4 Relationship to MIN's

LCA networks can also be viewed in another fashion. Let A be the an LCA network (see Figure 5); let A_1 comprise the PE's, the switches and the upward-going wires of the bi-directional links. Let A_2 comprise duplications of the PE's, the switches (except those in level 0) and the downward-going wires of the bi-directional links. Unfold A_2 in a mirror-like fashion (see Figure 6). 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 6). These are virtual links, and are costless to traverse; removing these links shows LCA networks' topological relationship to MIN's. LCA networks are different because communication in MIN's can only occur in a forward direction; however, in the folded LCA, A,

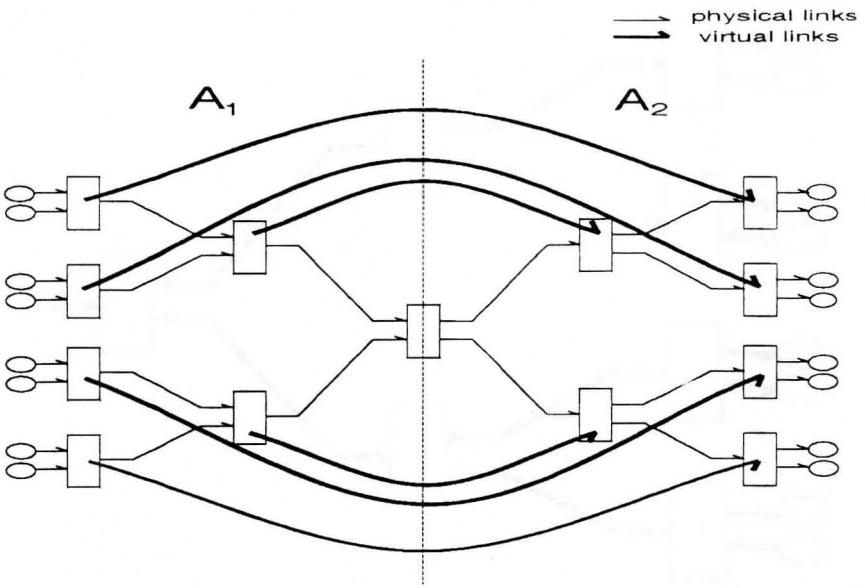

Figure 6: A unfolded into A_1 and A_2 showing virtual links

communication can move forward and backward.

The added functionality provided by the virtual links allows certain connections to be made without utilizing a switch in every stage, *i.e.* some middle stages need not be traversed. Given that the switch that is the lowest common ancestor of two communicating PE's belongs to some stage k, to connect the PE's, $1 + 2(k-1)$ stages in A can be skipped by utilizing the virtual links between A_1 and A_2.

3 Permutation Routing

In this section, some observations concerning permutation routing on LCA networks are made, and then the permutation routing algorithm is explained by means of an example LCA network where $d = 2, u = 1, S_0 = 1$, and $\overline{SP} = tree$, *i.e.* a binary tree with one switch in the top stage. To develop the permutation routing algorithm, LCA networks where the stage permutation is simple and homogeneous is considered, *i.e.* $\overline{SP} = tree$. The algorithm is analyzed, and then extended to cases where $d > 2$ and $u > 1$.

3.1 Path Intersection Graphs and Vertex Coloring

Finding the minimum number of passes needed to route a permutation can be formulated as a vertex coloring problem. Minimizing the number of passes utilizing the full bi-directional functionality of the uppers and downers of the switches results in a highly complex vertex coloring formulation. As such, for simplicity, the switch functionality is restricted such that no two connections (source-destination pairs) in the same pass may utilize the same switch. This results in the following vertex coloring formulation.

Each permutation of n elements is routed using n connections, each connecting two PE's. Let $s(P)$ be the set of connections in a permutation P. Routing a permutation can potentially take many passes through the network. A permutation routing algorithm determines the schedule for the connections, *i.e.* which connections get routed in each pass. Each connection is effected using a unique path; if c is a connection, let $path(c)$ represent its path. Two paths *intersect* if they contain a common switch. If the paths of two connections intersect, then the connections cannot be routed in the same pass.

A *k-vertex coloring* of a graph G consists of assigning k colors to the vertices of G; G is called *k-colorable*. A vertex coloring is *proper* if no two distinct adjacent vertices have the same color. The *chromatic number*, $\chi(G)$, of G, is the minimum k for which G is properly k-colorable [3].

To determine the schedule for a permutation P, we first construct its *Path Intersection Graph (PIG)*. Let $PIG(P)$ have vertex set V and edge set E. Every vertex v, where $v \in V$, represents a connection $c \in s(P)$. Each edge e, where $e \in E$, represents a conflict between two connections, *i.e.* the paths of two connections intersect. Thus, given a permutation P, and two connections $c_1, c_2 \in s(P)$, $\forall$ vertices $v_1, v_2 \in V$ $(v_1 \neq v_2)$, $\exists e \in E$ iff $\text{path}(c_1) \cap \text{path}(c_2)$. Let the minimum number of passes required to route P be p. Then $p = \chi(PIG(P))$. A connection is scheduled for the pass corresponding to its color.

3.2 Permutation Routing Algorithm

This section shows how to determine a permutation's schedule on an LCA network where $d = 2$ and $u = 1$. The algorithm repeatedly proceeds in three phases for each pass: the first two phases determine the switch settings necessary to route the connections in the pass and set the switches; in

the third phase, those connections route their data through the network. The algorithm schedules $O(p \log n)$ passes, and the settings for each pass take $O(\log n)$ steps to determine and set (p is the minimum number of passes that an algorithm with global knowledge schedules).

First, however, the *lowest common ancestor (LCA)* of two PE's is defined and routing between two PE's is described. The PE-to-PE routing is used in the permutation routing algorithm.

3.2.1 PE-to-PE Routing

Let the *LCA switch* of a connection (source-destination pair) be the switch at greatest depth which counts both PE's of the connection among its descendants. In a binary tree, each switch in level i has a *parent* link to a switch in level $i-1$, and *left* and *right* links to switches in level $i+1$. The level of the switch is determined as follows. Every PE has a label which consists of a binary number. Since there are l levels in the tree, each label consists of l digits. Given a permutation, each PE has a source address, $S = s_{l-1}s_{l-2}\ldots s_0$, and a destination address, $D = d_{l-1}d_{l-2}\ldots d_0$. Let i be the smallest i such that $s_{l-1}s_{l-2}\ldots s_{i+1} = d_{l-1}d_{l-2}\ldots d_{i+1}$. If $s_{l-1} \neq d_{l-1}$, then $i = l-1$. The LCA is in level $l-i-1$.

To route between two PE's labelled S and D, traverse up i parent links towards the root of the tree; now we are level $i+1$. Next, we traverse downwards using $d_i d_{i-1}\ldots d_0$. At level j, if $d_j = 0$, follow the left link; if $d_j = 1$, follow the right link. The switches' crosspoint setting algorithm follow these rules when routing packets calling for setting switches are received.

3.2.2 Permutation Routing: $d = 2$ and $u = 1$

The algorithm uses the LCA network to collect and disseminate information about the schedule. It requires the switches to have a contention mechanism that arbitrarily chooses between routing packets in case of contention, *i.e.* when two incoming routing packets both want to use the outgoing link.

Let the *LCA level* of a connection, or a PE (and its destination), be the level of the LCA switch of two communicating PE's. The algorithm gives connections with higher LCA level (closer to the top of the tree) priority over those with lower LCA level (closer to the bottom of the tree). The algorithm repeatedly proceeds in three phases.

In phase one, our permutation routing algorithm begins by attempting to schedule PE's with the highest (closest to the top of the tree) LCA level h. PE's with LCA level h send a routing packet to their parent switches in level $l-1$. Any switch which receives two routing packets deals with the contention by arbitrarily selecting one packet to "win," the other packet "loses;" this process is termed *competition* and is arbitrated by the switch's contention mechanism. In the next step, the winning packets proceed to their parent switches in level $l-2$, and PE's with LCA level $h+1$ send packets to their parent switches in level $l-1$. Now, competition occurs both at switches in levels $l-1$ and $l-2$. In lockstep fashion, sending routing packets up the tree and competing at switches repeats until packets from PE's with LCA level h reach their LCA switch; at this point packets of all LCA levels reach their LCA switch.

Phase two resolves conflicts between the winning packets that are at their LCA switch, and sends routing packets that set the switches according to the contention resolutions. There are potentially two winning packets at the highest LCA level, one from each subtree of the LCA switch. Let the source and destination addresses of one packet be S_1 and D_1; likewise let S_2 and D_2 be the addresses of the other packet. Each packet generates and sends two packets: one packet towards its destination and another towards its source in order to remove conflicting packets at lower LCA levels in both subtrees. The two packets going to one subtree, say S_1 and D_2 are concatenated into one packet; likewise for S_2 and D_1. When the concatenated packets are received at switches in level $h+1$, wire-contention is resolved with any winning packets waiting at the level $h+1$ switches. On the way down to the sources, the S_i's are understood to require the upward-going wire of the bi-directional uppers and downers; and the D_i's require the downward-going wires. The crosspoint switches are set following the crosspoint setting algorithm rules described in the PE-to-PE routing section. Any waiting packets which do not contend with packets from higher LCA levels repeat, in lockstep manner, the same process of creating two packets, one for the source and one for the destination. Any packet can hold at most two addresses, those of the PE's requiring the upwards and downwards wires of the bi-directional link. All other packets would have lost due to competition. This repeats until the packets from LCA level h reach their sources and destinations. Each PE that receives an S_i packet knows that it is to send data in phase three, and each PE that receives a D_i packet knows it is to receive data in phase three.

In phase three, winning connections passing their data through the network. Afterwards, the

phases are repeated; repetition ends when all PE's have been scheduled and routed.

The pseudo-code that each PE executes is as follows:

```
while ∃ unscheduled PE's
{
    ** phase one **
    h = highest LCA level of unscheduled PE's
    for level = 0 to l
        if PE's LCA level = level and PE not routed then
            compete one level (attempt to route to LCA level)
    if PE wins (reaches LCA level) then
    {
        ** phase two **
        for level = 0 to l
            if PE's LCA level = h or ≤ level then
                compete one level (while returning to source and going to destination)
        ** phase three **
        if PE receives routing packet then
            send data
    }
}
```

Assuming that resolving competition at switches using their contention mechanism takes one step, determining winning PE's for one pass takes $O(log\ n)$ steps since at worst a pass contains a connection with LCA level 0, necessitating traversing the height of the tree two times (up during phase one and down during phase two) in lockstep fashion with connections having other LCA levels. The algorithm is not guaranteed to produce a schedule with the minimum number of passes. Intuitively, this is because when choosing between competing PE's, non-optimal choices could be made, *i.e.* a choice could be made for PE's with LCA level h that precludes a PE with an LCA level lower than h from inclusion in the same pass.

3.3 Analysis

Claim: given a permutation, if p is the minimum number of passes required to route the permutation on a given network, our algorithm schedules $O(p\ log\ n)$ passes, where p is the minimum number of

passes and algorithm with global knowledge schedules.

In this section, we prove correctness of the algorithm and time bounds for $d = 2$ and $u = 1$, a binary tree. In the course of the algorithm, no two paths in the same pass can ever use the same wire of any bi-directional link. This is because competing connections have the same LCA level, thus they have the same LCA switch; the contention mechanism eliminates the possibility of two connections using the same wire on the way up the tree, and on the way down the tree. Two connections travelling in opposite directions may simultaneously utilize the switch that is their LCA since it has bi-directional links.

Lemma 1 *Given a set of connections that have the same LCA level, h, the permutation routing algorithm schedules the minimum number of passes required to route the set.*

Proof: Only those connections which utilize the same wire in a bi-directional link are not allowed to be routed in the same pass. The LCA switches are those that are in level h. With respect to the set, there are two subsets per LCA switch: those connections which go up the left subtree of the LCA switch and those which go up the right subtree. No two connections of one subset of one LCA switch may be routed in the same pass since they both need at least the upwards wire linking their subtree to the root. The minimum number of passes needed to route the entire set is the cardinality of the maximum sized subset. It is obvious that the algorithm performs thusly due to the competition as connections progress up the tree. $\Box Q.E.D.$

Theorem 2 *Given a permutation, on an LCA network where $d = 2$, $u = 1$, $\overline{SP} = tree$ and $S_0 = 1$, the permutation routing algorithm schedules at most $O(p \log n)$ passes, where p is the minimum number of passes an algorithm with global knowledge schedules.*

Proof: Let c be the set of connections routed in one pass of a given permutation. Assume there is an algorithm, A', which computes the "minimum" schedule such that the number of passes is a minimum. Due to the choices made in competition, at worst our algorithm, A, might not route any two connections with different LCA levels in the same pass. Thus, the algorithm would route connections with different LCA levels in different passes. There are at most $O(log\ n)$ different LCA levels among the LCA levels in c, therefore, at very worst, A could turn each pass in the "minimum" schedule into $O(log\ n)$ different passes.

Since A considers all connections with the same LCA level at the same time, by Lemma 1, A schedules $O(p\ log\ n)$ passes in the worst case. $\Box Q.E.D.$

3.4 Permutation Routing: $d > 2$ and $u > 1$

We still assume $d > u$ and now consider LCA networks where $d > 2$ and $u > 1$. In these cases PE-to-PE routing uses digits which are base d, not base 2. Since $\overline{SP} = tree$, a network with $u > 1$ can be mapped to a (d/u)-ary tree, where each link has *capacity* u, *i.e.* each link may simultaneously make u connections. Thus each pass takes $O(log_{d/u}\ n)$ steps (the height of the tree), to determine and set the switch settings.

Theorem 3 *Given a permutation, on an LCA network where $d > u$, $\overline{SP} = tree$ and $S_0 = 1$, the permutation routing algorithm schedules at $O(dp\ log\ n)$ passes in the worst case, where p is the minimum number of passes an algorithm with global knowledge schedules.*

Proof: In the permutation routing algorithm, up to $d + u$ connections may compete at a switch; the switch contention mechanism resolves contention among these connections. When connections are competing for the downers of their LCA switches, in the worst case $d - 1 + u$ connections compete for one downer. This is because the algorithm arbitrarily chooses winners among competing connections. Due to "bad" choices in the worst case, the algorithm could use $d - 1 + u$ passes to route the connections.

An algorithm with global knowledge can choose winners "wisely," and in the best case routes the $d - 1 + u$ connections in one pass. The ratio between the number of passes an algorithm with global knowledge schedules in the best case to the number of passes our algorithm schedules in the worst case is $\frac{1}{d-1+u}$. Since $d+1-u < 2d$, our algorithm schedules $O(dp\ log_{d/u}\ n)$ passes. $\Box Q.E.D.$

4 Conclusion

We have introduced Lowest Common Ancestor interconnection networks, which comprise identical switches with bi-directional uppers and downers arranged in a hierarchical structure. These networks are parameterized by $(u, d, n, l, \overline{SP})$-tuples. We investigated LCA networks where $d > u$

and $S_0 = 1$ (the number of switches in the highest level) because they are highly scalable, and cost-effective.

To show the usefulness of LCA networks for SIMD computers, we developed a permutation routing algorithm that schedules $O(dp\ log_{d/u}\ n)$ passes in the worst case for fully connected LCA networks with $\overline{SP} = tree$, *i.e.* to within $O(d\ log_{d/u}\ n)$ of a schedule that an algorithm with global knowledge obtains; each pass takes $O(log_{d/u}\ n)$ steps to determine and set the switch settings. The algorithm uses the network itself to collect and disseminate global information.

Because the analysis of the number of passes the algorithm schedules was not tight, and based on simulated permutation routings, we conjecture that the number of passes is actually closer to $O(dp)$. In addition to more in-depth analysis of the algorithm's performance, possible future research includes investigating permutation routing on networks represented by tuples where $d < u$, $d = u$, and $d > u$ and $S_0 \neq 1$, *i.e.* finding LCA networks that minimize p for a given permutation.

References

[1] V.E. Benes, *Mathematical theory of connecting networks and telephone traffic,* Academic, New York, 1965.

[2] T. Blank, R. Tuck, *Personal communications,* MasPar Computer Corporation, 1991.

[3] J.A. Bondy and U.S.R. Murty, *Graph theory with applications,* Elsevier Science Publishing, 1976.

[4] C. Clos, *A study of non-blocking switching networks,* Bell System Technical Journal, Vol. 32, 1953, pp. 406-424.

[5] A.M. Despain and D.A. Patterson, *X-Tree: A tree structured multiprocessor computer architecture,* Proc. Fifth Int. Symp. Comp. Architecture, April 1978, pp. 144-151.

[6] T. Feng, *A survey of interconnection networks,* Computer, Vol. 14, December 1981, pp. 12-27.

[7] J.R. Goodman and C.H. Sequin, *Hypertree: A multiprocessor interconnection topology,* IEEE Transactions on Computers, C-30, December 1981, pp. 923-933.

[8] J. Harris and D. Smith, *Simulation experiments of a tree organized multicomputer,* Proc. 6th Annual Symp. on Comp. Arch., IEEE, April 1979, pp. 83-89.

[9] E. Horowitz and A. Zorat, *A divide and conquer computer,* CS Dept. Technical Report, USC, July 1979.

[10] E. Horowitz and A. Zorat, *The binary tree as an interconnection network: Applications to multiprocessor systems and VLSI,* Proc. of the Workshop on Interconnection Networks, April 21-22, 1980, pp. 1-10.

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

[12] B.L. Menezes and R. Jenevein, *The KYKLOS multicomputer network: Interconnection strategies, properties, and applications,* IEEE Transactions on Computers, Vol. 40, No. 6, June 1991, pp. 693-705.

[13] D. Nassimi and S. Sahni, *Parallel algorithms to set up the Benes permutation network,* IEEE Trans. Comput., Vol. C-31, No. 2, February 1982, pp. 148-154.

[14] H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing,* Lexington Books, 1985.

[15] C.W. Wu and T. Feng, *The reverse-exchange interconnection network,* IEEE Trans. Comput., Vol. C-29, No. 9, September 1980, pp. 801-811.