Hypercube Vs Cube-Connected Cycles: A Topological Evaluation

Srikanth Kambhatla

Oregon Graduate Institute
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 91-006

March, 1991

HYPERCUBE Vs CUBE-CONNECTED CYCLES: A TOPOLOGICAL EVALUATION

Srikanth Kambhatla
Oregon Graduate Institute of Science and Technology
Beaverton, OR 97006

ABSTRACT

Hypercubes and cube-connected cycles differ in the number of links per node which has fundamental implications on several issues including performance and ease of implementation. In this paper, we evaluate these networks with respect to a number of parameters including several topological characterizations, fault-tolerance, various broadcast and point-to-point communication primitives. In the process we also derive several lower bound figures and describe algorithms for communication in cube-connected cycles. We conclude that while having lower number of links per node in a CCC might not degrade performance drastically (especially for lower dimensions) as compared to a hypercube of a similar size, this feature has several consequences which substantially aid its (VLSI and non-VLSI) implementation.

Keywords: topological evaluation, Cube-connected cycles, communication primitives, fault-tolerance, implementation issues.

1. Objective

The hypercube topology [1] has been a popular interconnection network due to many reasons. These include [2]:

- It subsumes a wide variety of interconnection patterns including meshes, trees, rings and X-trees.
- Its diameter is logarithmically related to the number of nodes.
- The algorithm for routing messages is fairly simple.
- There exist alternate paths between any pair of nodes enabling fault-tolerance and congestion-avoidance.

However, one of the major problems with this topology is that the number of links per node increases (logarithmically) with the number of nodes in the network. It is believed that VLSI communication networks are wire limited and that the cost of a network is not dependent on the number of switches required but is rather dependent on the wiring density required to construct the network [3]. The Cube Connected Cycles (CCC), proposed by Preparata and Vuillemin [4], retains most of the desirable properties of the hypercube while restricting the number of connections per node to three.

Larger number of connections per node in the hypercube potentially make it more efficient for communication tasks and more fault-tolerant than the CCC. On the other hand, a fixed bound on

the number of connections per node in the CCC makes the topology particularly scalable. In this paper, we perform an evaluation of these two topologies with respect to a number of parameters including several topological characterizations, fault-tolerance and various communication primitives. In the process we also derive several lower bound figures and describe algorithms which achieve these communication primitives.

Rest of the paper is organized as follows. The next section gives the basic definitions and notations used in the paper. Section 3 evaluates the hypercube and the CCCs with respect to several topological characterizations. In section 4 we study the fault-tolerance aspects of the topologies. Section 5 concentrates on the bounds for various communication primitives in the two systems. Section 6 gives the related work while section 7 concludes the paper.

2. Preliminaries

An n-dimensional hypercube graph is an undirected graph consisting of $k = 2^n$ vertices labeled from 0 to $2^n - 1$ such that there is an edge between any two vertices if and only if their binary numbers (addresses) differ by one and only one bit [5].

A CCC is obtained by replacing each node in a hypercube graph by a cycle of nodes (and adding connections among these nodes according to the definition given below). The number of processors in each cycle is greater than or equal to the dimension of the hypercube. A CCC(d,k) can be described by two parameters: the dimension of the underlying hypercube *d*, and the number of processors per cycle *k*, with $k \geq d$. If the CCC is based on a d-cube, we say that the *size* of the CCC is d. In this paper we assume $d = k = n$.

The address of a processor is represented as a tuple (c,p) where c is the cycle number and p is the position of the processor within a cycle. The number of cycles in the system is 2^d and, the total number of processors is $k2^d$. There is a link between two processors with identities (i,j) and (k,m) if and only if a) j=m and i differs from k in the jth least significant bit or b) i=k and $|j-m| = 1$ or $|j-m| = d-1$. CCC(3,3) is shown in figure 1. We see that each processor has exactly three links irrespective of the dimension of the base cube.

A1 gives a routing algorithm for the CCC.

3. Topological Characterizations

For an n dimensional hypercube, each node has n links. Since there are 2^n nodes in the system, we have a total of $\frac{n2^n}{2}$ links. A CCC which is based on an n dimensional cube has $n2^n$ processors each having three links resulting in a total of $\frac{3n2^n}{2}$ links. We notice that for the same dimension there are n times more processors in the CCC but only 3 times more links.

Bisection width

Wire density on the chip is important in VLSI computing systems because the speed of the machine is limited by wire delays and the majority of the power consumed by a machine is used to drive wires [3]. *Bisection width* [6] is commonly used as a measure of network cost. The bisection width of a network is the minimum number of wires cut when the network is divided

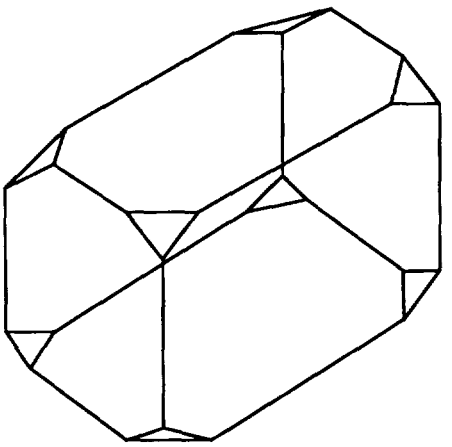

Figure 1. CCC(3,3).

Algorithm A1: Routing algorithm from any source to any destination

1. Compute the difference vector of the cube connection.
2. If the bit number corresponding to the current node number in the cycle is part of the difference vector, route the message along the lateral connection of the current cycle.
3. Else, find the closest path within the cycle to the next bit position to be changed in the difference vector, and route the message along that direction.
4. If current node = destination, quit. Else goto step (2).

into two equal halves. The Bisection width (B) of an n-cube turns out to be 2^{n-1} or N/2 where N = 2^n is the total number of nodes. In the case of a CCC, B = 2^{d-1} = N/2n. Thus, B for a given number of nodes is lesser for CCC by a factor of n.

Diameter and Average distance

Average distance between nodes and the diameter of the graph are some of the important factors determining the (communication) performance of the topology. In an n-cube there are exactly $\binom{n}{i}$ nodes a distance i away. This gives us an average distance of $\frac{1}{n}\sum_{i=1}^{n} i\binom{n}{i}$, which equals approximately 1.9 for a 3-cube and 2 for a 4-cube. However, this average distance is 3.25 for CCC(3,3). This is not surprising because the average distance increases when the number of links per node is small. The diameter for an n-cube is n and we show (see result 4 and corollary 3 below) that the diameter of a CCC(d,d) is 2d. This shows that the diameter for a CCC of size d is twice that of a d-cube.

Result 1. *The maximum number of hops required in order to reach a destination node from a source is 2*n.*

Proof: The proof is by induction.

Basis: The result trivially holds for a CCC of size 1, where we have two nodes connected to each other. We consider a CCC of size 2. A construction of the graph giving different nodes at increasing distances results in figure 1 for node (0,0). It is clear from the graph above that it cannot take more than 2*n (= 4) hops from the source to the destination. Construction of a similar graph for a CCC of size 3 also gives the same result.

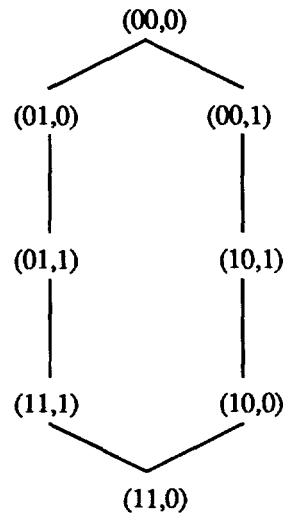

Figure 2. Nodes at increasing distances from (0,0).

Inductive step: Assume that the result holds for a CCC of size n. We now consider a CCC of size (n+1). From result 2 (see below), we notice that the CCC can be constructed from 2 CCC's of size n. We call the two CCCs of size n C1 and C2 for ease of referance. When the source is a new node (after the formation of the CCC of size n+1 from C1 and C2), we consider the following cases. Without any loss of generality let us assume that this node is an addition to C1.

1. From the new node to any other node in C1. It takes one hop from the new node to an old node in the same cycle. We have now entered a CCC of size n. By our hypothesis, it takes a maximum of 2*n steps to reach any other node within the same CCC of size n. Also, to reach any other new node within C1, it takes exactly one hop from a corresponding node which is within C1 but is next to the new (destination) node. Thus, the total number of hops is not more than 1 + 2*n + 1 = 2*(n + 1).

2. From a new node in C1 to any other node in C2. It takes exactly 2 hops to reach an old node of C2 (1 jump & 1 shift). From there, it is a maximum of 2*n hops to any other node. Thus, for any old node in C2, the number of hops is a maximum of 2*(n+1). We now consider any other new (destination) node in C2. By Fig. 1 and similar figures for CCC of size 3, 2*n steps is required only for a node obtained by bit-reversal of the cube address, keeping the cycle address same. In particular, we

notice that due to the cycle interconnection, there are two old nodes 1 hop away from any new node. Thus, from any old node, one of these two old nodes can be reached in less than (2*n) hops. Therefore, the number of hops is a maximum of 2 + i + 1 (where $i < 2*n) \leq 2*(n + 1)$. □

Corollary 1. The node most distant from any given node in a CCC can be obtained by a bit-reversal of the cube address while keeping the node number within the cycle constant. □

Corollary 2. The diameter of a CCC (which is also the minimal depth of a spanning tree within a CCC) is 2n. □
It may be noted that this is a better estimate than those previously given in literature [7, 8].

Scalability

Scalability is another important issue while choosing a topology. It is well-known that two n-cubes can be combined to form an (n+1)-cube. However, the number of links for each node increases correspondingly. This may be considered a drawback because existing nodes have to be designed considering possible future expansion (i.e. support should be provided for an appropriate number of I/O ports).

Result 2. *CCC's can be inductively constructed from smaller CCC's.*
Proof: Suppose, we have two CCC's of size n. In order to construct a CCC with (n+1) nodes in the cycle, we follow the following two steps:

1. Add one node to each cycle in both CCCs of size n.

2. The interconnections for the new node are simply obtained by connecting the new nodes of corresponding cycle. It may be noted that this satisfies the conditions of inductive architectures as defined by Lipovski and Malek [9]. □

We see that the nodes in a CCC have a constant number of links and that the nodes are not affected by an increase in the size of the CCC. We consider this to be a significant advantage over the hypercube.

Embeddings

Mapping of other topologies onto our current topology is of interest because of the following reasons [5]:

1) Some algorithm may have been developed for another architecture for which it fits perfectly. Then one might wish to implement the same algorithm with little additional programming overhead on the current topology.

2) A given problem may have a well-defined structure which leads to a particular pattern of communication. Mapping the structure may therefore result in substantial savings in communication time.

Efficient mappings of topologies onto hypercubes have been studied extensively in the literature [5, 10, 11]. These topologies include rings, linear arrays, grids and binary trees. Since a hypercube is logically present in a CCC [4], all these mappings hold in the CCC by default.

VLSI implementation issues

It becomes necessary to re-evaluate a topology for VLSI implementation for a number of reasons. These include [12]:

1) The spatial distribution of the processors play an important role on the total chip area, while it is not a constraint on design when the implementation is not in VLSI.

2) The signal propagation delay directly depends on the length of the interconnection because of the additional capacitive components.

3) Regularity of the network decides the layout cost.

4) Existance of redundancy in the network and such factors as the presence of long interconnects (which increase the chip failure probability) determine the device yield.

It has been shown in [4] that the VLSI layout for hypercube requires $O(N^2)$ area which is larger than $O(\frac{N^2}{\log^2 N})$ required for a CCC. If we assume O(1) layout cost per processor, the area efficiency (defined as the ratio of the area occupied by the processors to the total chip area) is O(1/N) for a hypercube and $O(\log^2 N/N)$ for the CCC. Coupled with increased area efficiency, we get an increased power efficiency (which is defined analogous to the area efficiency) assuming a constant steady state power dissipation per unit area. While the lengths of the interconnects in the topologies of interest are not as good as some other topologies (meshes for example), we have redundant paths (discussed below) in both CCCs and hypercubes giving them good fault-tolerance properties.

4. Fault-Tolerance

There are two aspects of fault-tolerance in networks: tolerance to faults in links and tolerance to faults in the nodes. Link failures can be tolerated when there exist redundant paths in the topology. It has been proved that there are n edge-disjoint paths from any source to any destination in an n-cube, h of these paths are of length h and (n-h) are of length (h+2), where h is the hamming distance between the source and destination nodes [5].

Result 3. *There exist three disjoint paths between any two nodes in a CCC, for $r > 2$.*
Proof: Let the address of the source node be (Sn...S0,i) and that of the destination node be (dn...d1,j). The two links to the source node within the cycle result in the following three starting nodes: 1) (Sn...S0, (i+1) mod n), 2) (Sn...S0,i) and 3) (Sn...S1, (i-1) mod n). From these nodes we attempt to reduce the hamming distance between the cube addresses of the source and destination in a clockwise direction (left to right with wrap-around) starting with the bit at the current position in the cycle.

Without any loss of generality, we will consider the paths from nodes 1) and 2) above and show that they are disjoint. All other cases are analogous. For ease of referance we write ((i+1) mod n) simply as (i+1). In the next step in the routing (when the nodes take the lateral connection out of the cycle) we get the following nodes:

$$(Sn...S0, i+1) \rightarrow (Sn...S\bar{i+1}..S0, i+1)$$

$$(Sn...S0, i) \rightarrow (Sn..S\bar{i}..S0,i)$$

If the bit at position i+1 is not in the distance vector (obtained by taking an exclusive-or of the cube addresses of source and destination nodes), clearly, the path followed by 1) will be different from all other (because bit Si+1 will be different in the nodes in their paths). Same applies to the bit at position i.

If on the other hand, the bits at position i+1 and i are set in the distance vector, we see that a clockwise sequence of changes will all be different in the cube address until the last but one step, at which point they will differ in their cycle position. For example, consider the case when the source node is (000,1) and the destination node is (011,1). The distance vector is 011. We consider the starting points (000,2) and (000,1). The paths from these starting points are given below:

$$(000,2) \rightarrow (100,2) \rightarrow (100,1) \rightarrow (110,1) \rightarrow (110,0) \rightarrow (111,0) \rightarrow (111,2) \rightarrow (011,2) \rightarrow (011,1)$$
$$(000,1) \rightarrow (010,1) \rightarrow (010,0) \rightarrow (011,0) \rightarrow (011,1)$$

It may be noted that since the bit at position 2 was not set in the distance vector, the nodes in the two paths do not meet till the last step where they differ in the cycle positions. The paths in other cases also do not clash when a clockwise changes are followed. Since at each step in the routing we either change the cube address or the cycle address, we cannot reach the same intermediate node address in the same number of hops when starting from two different nodes. □

Corollary 3. *The disjoint paths are not of the same length.*
Proof: Result follows from the construction. □

Strategies have been developed for increasing the fault-tolerance which involve the presence of extra links in the network. These strategies include the presence of an extra stage (in a multistage version of the cube interconnect) [13] and folding of the cubes to improve certain other communication properties [14].

Tolerance of node failures requires the presence of redundant nodes so that the integrity of the topology can be maintained by reconfiguration. Maintaining the topology is critical in applications which depend heavily on pipelined execution because the pipeline is broken on a failure. The scheme proposed in [15] involves the addition of extra links and an additional processor per cycle in order to achieve reconfiguration on a failure. Other such *strongly* fault-tolerant CCC structures include the cubical ring connected cycles of Banerjee [16, 17]. Recent papers [18, 19, 20, 21] discuss the issue of spare allocation in fault-tolerant hypercubes. Work has also been reported on fault-diagnosis in n-cubes defining limits on the number of faults for isolation of nodes and diagnosis strategies [22, 23].

5. Broadcast and Point-to-point Communications

We now evaluate the topologies with respect to common data communication patterns in parallel numerical computations which include [24, 25]:

1) *Singlenode broadcasting*: which involves transfer of the same data from one processor to all other processors in the network.

2) *Multinode broadcasting*: which involves concurrent broadcasting from all nodes to all other nodes in the network.

3) *Scattering*: which is singlenode broadcasting with the difference that different packets are transferred to different nodes.

4) *Total exchange*: which involves transfer of a different packet from each node to every other node.

Broadcasting finds applications in a variety of linear algebra algorithms [26], such as matrix-vector multiplication, matrix-matrix multiplication, LU-factorization, and also in database queries and transitive closure algorithms [27]. Scattering and total exchange operations occur in transposing a matrix and the conversion between different data structures [26].

Johnsson and Ho [25] describe broadcast and personalized communications in the hypercube. They give lower bounds and also describe algorithms which achieve the communication tasks. In this section, we give lower bounds and algorithms for the communication primitives mentioned above, for the single I/O (where each processor can send or receive at most one message at a time) and multiple I/O (where each processor can simultaneously do I/O operations on all the ports available) cases for the cube-connected cycles architecture. We then compare the topologies with respect to these lower bounds. Our results for CCC are summarized in table 1.

Table 1.
Lower bounds for communication patterns in CCC

Communication task	Single I/O	Multiple I/O
Single node broadcast	2n	2n
Multinode broadcast	(N-1)	(N-1)/3
Scatter	(N-1)	(N-1)/3
Total exchange	-	(Nn)/2

Single-node broadcast

Result 4. *A lower bound for the number of time steps required for single node broadcast with single I/O is 2*n for a CCC of size n. Further, this bound does not change for multiple I/O.*
Proof: Follows from result 1. Even in the case of multiple I/O, since a node exists at a distance 2n (corollary 1), 2n timesteps are still required. □

Result 5. *Algorithm A2 satisfies the following properties:*
1. All the nodes receive the message.

Algorithm A2 (single node broadcast with single I/O)

```
If (originator)
        send the message along the lateral connection;
        send the message left (or right) in the cycle;
else If (msg received from a lateral connection)
        send the message to left (or right) in the cycle;
else If (msg received from a horizontal connection)
        send the message to the lateral connection.
```

2. It achieves the lower bound as defined in result 4.
Proof: The intuition behind the algorithm is figure 1 and a corresponding figure for a CCC of size 3. We see that the adjacent nodes for the source node are two nodes in the same cycle (n1 and n2) and one (n3) in another cycle. In the next step of the broadcast, n3 has two adjacent nodes in the same cycle (the node on the lateral connection has already received the message), while for n1 and n2 the adjacent nodes in the same cycle have received the message, leaving only the node in the lateral direction yet to receive the message. Thus, at any level, the nodes either have a lateral neighbour which has not yet received the message or, two neighbours in the same cycle which have not received the message, but not both.

Hence, in our algorithm, a node which gets a message from its lateral neighbour need only send it to its neighbours in the same cycle, and similarly a node receiving a message from its neighbors in the cycle need only propagate it to its neighbor in the lateral direction.

From construction, the algorithm follows the distance graphs (analogous to figure 1). Thus, all the nodes eventually get the message. Further, the lower bound is also achieved (because the diameter of the distance graph is 2*n). □

Algorithm A3 (single node broadcast with multiple I/O)

```
If (originator)
        simultaneously send horizontally and laterally;
else if (rcvd laterally)
        simultaneously send to left and right;
else send in the lateral direction.
```

Result 6. *Algorithm A3 also achieves the broadcast and the lower bound.*
Proof: Similar to that of result 5. The only difference is the simultaneity in sends, which does not effect the pattern of message propagation. Hence the lower bound is not disturbed. □

Multinode broadcast

Result 7. *A lower bound for multinode broadcast with single I/O is (n*(2**n) - 1). For multiple I/O case, the lower bound is (n*(2**n) - 1)/3.*
Proof: The total number of nodes in a CCC is N = n*(2**n). Each node has to receive (N-1) messages. In the multiple I/O case, the node can receive at most three messages at any given time. □

Result 8. *The requirements for a single global ring in CCC of size > 3, which is simple and elementary, are:*
1. The hamming distance between any two consecutive nodes in the ring is 1.
2. The difference in changes in dimensions between cube addresses of consecutive nodes forming the ring is 1.
Proof: Condition (1) is necessary for the connectivity of consecutive nodes. The motivation for condition (2) is a bit more involved. If a single global ring is to include all the nodes in a CCC, all the nodes of the cycle need to be traversed while crossing between two cube connections. If the ring is to be simple and elementary, it becomes necessary that the entry and exit points from the cycle be adjacent. If this were so, one can traverse the nodes of the cycle between the entry and exit points. This means that the bit positions which change between the consecutive cube addresses should be adjacent to each other. This gives us the condition (2). For a CCC of size ≤ 3, it is possible to include all the nodes in the cycle in any combination of entry and exit points of the global ring at each cycle. □

Corollary 4. The standard reflected gray code ring embedding of a cube is not adequate for embedding a ring in CCC.
Proof: The standard binary reflected gray code does not satisfy the condition (2) of result 11. An example is the sequence 0011 --- 0010 --- 0110 which occurs when a ring is being embedded in a 4-cube. The successive changes are in dimensions 0 and 2. □

Result 9. *A ring can be embedded such that at least two nodes of each CCC cycle are part of the global ring.*
Proof: This global ring is achieved by traversing a reflected gray code routing pattern, but every time a CCC cycle is encountered, the intermediate nodes are traversed before the appropriate node in the cycle is reached such that the next cube address can be reached; at which point, we exit the cycle. □

Result 10. *We can have several partial rings which together include all the processors of the CCC.*
Proof: We can attempt different ring embeddings (which differ only in the paths taken within the cycles) which are sequentially composed so that each ring contains some of the nodes and all the rings collectively exhaust all the nodes in the CCC. Thus the final global ring is neither elementary nor simple. □

Result 11. *A4 satisfies the following properties:*
1. It is correct i.e. it implements a multinode broadcast.
*2. It takes O(n**N) time steps.*
Proof:
Correctness:

> At the end of step (1), all the nodes in the (partial) global ring exchange messages. The nodes in the (partial) global ring of step (2) have 2 nodes common with every cycle in the partial global ring of the previous step. This is so because the entry points and the exit points of each cycle in both the partial embeddings are the same. This enables the nodes of step (2) to obtain the information from all other nodes in the CCC. However, we have some nodes that participated in step (1) (and were not present in step (2)) that do not have

Algorithm A4: Multinode Broadcast

A4 involves the following steps:

1. We follow the reflected gray code on the cube addresses. When cycles are reached, we uniformly traverse the cycles in an increasing dimension (mod n).

2. We follow the reflected gray code on the cube addresses. When cycles are reached, we uniformly traverse the cycles in a decreasing dimension (mod n).

3. We follow the reflected gray code on the cube addresses. When cycles are reached, we uniformly traverse the cycles in an increasing dimension (mod n).

messages from the nodes involved only in step (2). This necessitates a repeat of step (1). Again, since some nodes are common, a total dispersion of information can be effected.

Performance:

Suppose that r1 nodes per ring are involved in step (1). Thus, step (1) takes r1*(2**n) time steps. Of these r1 nodes per ring, two are involved in step (2) as well. Thus the number of nodes in step (2) per ring are (n - r1 + 2). The number of time steps is ((r1 - 2)/2)*(n - r1 + 2)*(2**n). The factor (r1-2)/2 arises because, each of the common nodes needs to relay information of those cycle nodes which participated in step (1) but not in step (2). Finally, step (3) requires ((n - r1 + 2 -2)/2)*(r1)*(2**n) steps, by the same reasoning. This gives us a total of: [(r1)/2 + (n-r1+1)(r1-1) + 1/2]* (2**n). If we assume that on an average, in different CCC sizes, we have an average split of half the nodes in each cycle, i.e. (r1) = (n/2), the number of time steps becomes [(n**2)/4 + (n/2) + 1]*(2**n) = O([n**2]*[2**n]) = O(n*N), where N = n*(2**n) is the number of nodes in the CCC. □

Single node Scattering

Result 12. *A lower bound for single node scattering using single I/O is (n-1); the lower bound is (n-1)/3 for multiple I/O.*

Proof: The source node needs to send (n-1) messages. With multiple I/O the source node can potentially send 3 messages at a time. □

Algorithm A5: (Single node) Scattering - single I/O

For single node scattering, we follow the routing algorithm A1, sending data for the farthest node first and pipelining the messages to other nodes.

Algorithm A6: Scattering - multiple I/O

For the multiple I/O case, we use a modification of the single node broadcast algorithm and pipe different messages. The modification is essentially in not using the horizontal/ lateral sends when they are not required.

Total Exchange

Result 13. *A lower bound for total exchange with multiple I/O is (N*n)/2.*
Proof: If we make a cut on the nth dimension of a CCC of size n, the number of processors on each partition = n*(2**(n-1)). The number of messages which need to be sent across the cut (per processor) = n*(2**(n-1)). Therefore, total number of messages = (n*(2**(n-1))) ** 2. The number of messages which can be transmitted per unit of time = 2**(n-1), which implies a lower bound of (2**(n-1))*(n**2) = (n*N)/2 timesteps. □

Algorithm A7: Total Exchange (multiple I/O)

A7 is recursive and follows the following steps for a CCC of size n:

1. Total exchange in CCC of cube dimension n-1 simultaneously; also simultaneously exchange messages along the nth dimension.

2. Do total exchange with the new messages in the CCCs of cube dimension (n-1).

When the recursion comes down cube dimension 1, we have a cycle. At this point, we do a total exchange within the ring (by rotation).

6. Related Work

Several topological evaluations (of different topologies) have been attempted in literature [3, 12, 8]. These have generally tended towards evaluation of certain specific characteristics of interest: dimensionality [3] or communication [8] or VLSI implementation [12], for example. Hypercubes and CCCs differ in the number of links per node which has very fundamental implications on several parameters. The goal of this paper has been to perform an evaluation of these two topologies in order to study the effects of these parameters.

7. Conclusions

In this paper, we have studied different results obtained for a CCC with those existing in hypercube in the literature. Table 2 summarizes the results of the study.

A major characteristic of the CCC is constant number of links per node, which reduces the bisection width (it is a factor of n lesser than that of the hypercube). This coupled with its regular topology (enabling efficient layouts, higher area and power efficiencies) makes it a better candidate for VLSI implementations. A fixed number of links per node implies that the architecture is

Table 2. Comparision of topologies

Parameter	Hypercube	CCC
relative #links	3	1
relative #nodes	1	n
Bisection width	N/2	N/(2n)
average distance (size 3)	1.9	3.25
diameter	n	2n
scalability	Not inductive	inductive
embeddings	Most topologies	Most topologies
chip area	$O(N^2)$	$O(\frac{N^2}{\log^2 N})$
area efficiency	$O(\frac{1}{n})$	$O(\frac{\log^2 N}{N})$
power efficiency	$O(\frac{1}{n})$	$O(\frac{\log^2 N}{N})$
redundant paths	n	3
singlenode broadcast	n-1	2n
multinode broadcast	(N-1)/n	(N-1)/3
(singlenode) scatter	(N-1)/n	(N-1)/3
total exchange	N/2	nN/2

highly scalable which is a significant advantage for its (VLSI or non-VLSI) implementation.

Higher number of links per node increases tolerance to link failures. In order to maintain *strong* fault-tolerance however, one needs to incorporate spare nodes so that topological integrity can be maintained.

The lower bounds for communication primitives in the table are for multiple I/O model of communication (i.e. when maximum allowable parallelism can be extracted). These results can therefore be viewed as an upper bound on performance during evaluation. On a comparision of these results, we notice that in most of the cases the lower bound figures for the hypercube are lower by a factor of n/3. The difference in performance would therefore be significant only when n is large. With the present communication technology (like the Direct Connect Module of Intel) more hops in a path do not contribute much overhead to the overall communication delay. Thus, the performance degradation in CCCs for communication primitives may not be significant (as compared to a hypercube of an equivalent dimension) especially for lower dimensions.

The contributions of this paper have been two-fold:

1) In order to compare the topologies with respect to communication issues, we presented lower bounds and algorithms achieving various forms of broadcasts and personalized communications in CCC (which were compared with existing results for the hypercubes). In each communication primitive, we assumed two models of I/O communication. The lower bound results for CCC are summarized in table 1.

2) We performed a topological evaluation of hypercubes and the CCCs with respect to a number of parameters and conclude that while the lower number of links in each node might not be too much of a performance hinderance (especially for lower dimensions) as

compared to a hypercube of a similar size, they have several positive effects which substantially aid its (VLSI and non-VLSI) implementation.

References

1. H. Sullivan and T. R. Bashkow, "A Large Scale Homogeneous, Fully Distributed Parallel Machine," *Proceedings of the Fourth Symposium on Computer Architecture*, pp. 105-117, 1977.

2. K. Ghose and K. R. Desai, "The Design And Evaluation Of The Heirarchical Cubic Network," *Proceedings of the 1990 International Conference on Parallel Processing*, pp. I355-I362.

3. W. J. Dally, *A VLSI Architecture For Concurrent Data Structures*, Ph.D. Thesis, Department of Computer Science, California Institute of Technology, 1986.

4. F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Communication Network For Parallel Computation," *Communications of the ACM*, vol. 24, 5, pp. 300-309, 1981.

5. Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, vol. 37, 3, pp. 867-872, 1988.

6. *A Complexity Theory of VLSI*, Computer Science Department, Carnegie-Mellon University, Technical Report CMU-CS-80-140, 1980.

7. D. S. Meliksetian and C. Y. R. Chen, *Optimal Routing Algorithm and Other Communication Issues of the Cube-connected Cycles*, Department of Electrical And Computer Engineering, Syracuse University tech report TR 90-4, 1990.

8. L. D. Wittie, "Communication Structures For Large Networks Of Microcomputers," *IEEE Transactions on Computers*, vol. C-30,4, pp. 264-273, 1981.

9. G. J. Lipovski and M. Malek, *Parallel Computing theory and comparisions*, John Wiley & Sons, 1987.

10. A. Wagner, "Embedding Arbitrary Binary Trees in a Hypercube," *Journal of Parallel and Distributed Computing*, pp. 503-520, 1989.

11. A. Y. Wu, "Embedding of Tree Networks into Hypercubes," *Journal of Parallel And Distributed Computing*, pp. 238-249, 1985.

12. P. Mazumder, "Evaluation of Three Interconnection Networks For CMOS VLSI Implementation," *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 200-207, 1986.

13. G. B. Adams and H. J. Siegel, "The Extra Stage Cube: A Fault-tolerant Interconnection Network for Supersystems," *IEEE Transactions on Computers*, vol. C-31, pp. 443-454,

1982.

14. S. B. Choi and A. K. Somani, "The Generalized Folding-Cube," *Proceedings of the 1990 International Conference on Parallel Processing*, pp. I-372-I-375, 1990.

15. N. Tzeng, S. Bhattacharya, and P. Chuang, "Fault-tolerant Cube-connected Cycles Structures Through Dimensional Substitution," *Proceedings of the 1990 International Conference on Parallel Processing*, pp. I-433-I-440, 1990.

16. P. Banerjee, "The Cubical Ring Connected Cycles: A Fault-tolerant Parallel Computation Network," *IEEE Transactions on Computers*, vol. C-37, pp. 632-636, 1988.

17. P. Banerjee, S. Kuo, and W. K. Fuchs, "Reconfigurable Cube-connected Cycles Architecture," *Proceedings of the 16th International Symposium on Fault-tolerant Computing*, pp. 286-291, 1986.

18. M. S. Alam and R. G. Melhem, "An Efficient Modular Spare Allocation Scheme and Its Application to Fault-Tolerant Binary Hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, 1, pp. 117-126, 1991.

19. P. Banerjee, J. T. Rahmeh, C. B. Stunkel, V. S. Nair, K. Roy, and J. A. Abraham, "An Evaluation of System Level Fault-Tolerance on the Intel Hypercube Multiprocessors," *Proceedings of the International Fault-Tolerant Computing Symposium*, pp. 362-367, 1988.

20. S. C. Chau and A. L. Liestman, "A Proposal for a Fault_Tolerant Binary Hypercube Architecture," *Proceedings of the International Fault-Tolerant Computing Symposium*, pp. 323-330, 1989.

21. S. Dutt and J. P. Hayes, "An Automorphic Approach to the Design of Fault-Tolerant Multiprocessors," *Proceedings of the International Fault-Tolerant Computing Symposium*, pp. 496-503, 1989.

22. J. R. Armstrong and F. G. Gray, "Fault Diagnosis in a Boolean N-Cube Array of Microprocessors," *IEEE Transactions on Computers*, vol. C-30,8, pp. 587-590, 1981.

23. A. Kavianpour and K. H. Kim, "Diagnostic Power of Four Basic System-level Diagnosis Strategies for Hypercubes," *Proceedings of the 1990 International Conference on Parallel Processing*, pp. I-267-I-271, 1990.

24. Y. Saad and M. H. Schultz, "Data Communications in Parallel Architectures," *Parallel Computing*, vol. 11, pp. 131-150, 1989.

25. S. L. Johnsson and C. T. Ho, "Optimum Broadcasting and Personalized Communications in Hypercubes," *IEEE Transactions on Computers*, vol. 38, 9, pp. 1249-1268, 1989.

26. S. L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," *Journal of Parallel and Distributed Computing*, vol. 4, pp. 133-172, 1987.

27. S. A. Browning, *The Tree Machine: A Highly Concurrent Computing Environment*, Technical Report 1980:TR:3760, Computer Science, California Institute of Technology, 1980.