ELSEVIER

Information Processing Letters 53 (1995) 277–286

Information Processing Letters

# Topological properties of the directional hypercube

Mounir Hamdi [1]

*Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*

Communicated by R.G. Dromey; received 1 April 1992; revised 1 September 1994

**Abstract**

The directional hypercube (Dcube) which is a cost-effective variation of the traditional binary hypercube is introduced and analyzed in this paper. It employs directional (simple duplex) links only. The Dcube is defined and its key properties are derived including diameter, bandwidth, and connectivity. The diameter is at most 2 greater than that for a hypercube of the same size; the bandwidth is 1/2 that of the hypercube; and the connectivity is optimal. The Dcube is shown to emulate the binary hypercube with at worst a factor of 4 degradation in time performance under any message distribution. A simple routing algorithm is demonstrated for the Dcube which requires only local information to route messages between nodes. Then, the concept of virtual channels has been added to the routing algorithm to make it deadlock-free.

*Keywords:* Fault tolerance; Parallel processing; Connectivity; Diameter; Node-disjoint path; Emulation; Routing

## 1. Introduction

The full duplex (bi-directional) links used to interconnect nodes in an interconnection network are more costly than directional links. Bi-directional link applications either replicate hardware to provide dedicated directional links or increase communication time complexity by time-sharing a link. In our analysis, we assume that each bi-directional link is implemented by using two dedicated directional links, this is the case in most practical implementations [15]; and that each switching node communicates with its neighboring nodes through input and output queues denoted by I and O. For a directional interconnection network, each switching node communicates with a neighboring node through either an input queue or an output queue as shown in Fig. 1. Thus, using directional links, there is a tremendous savings in the number of queues being used and the hardware associated with them. For more detailed analysis, please refer to [7]. Moreover, as can be seen from Fig. 1, the number of I/O pins per chip and the number of I/O ports per board required for each directional link will be half of that required by a bi-directional link. Hence, it would be less costly to build an interconnection network with directional links than with bi-directional links especially if the degree of the network is high such as the hypercube [17].

The architecture of hypercube interconnection network has achieved a marked popularity in the

[1] Email: hamdi@cs.ust.hk.


*SSDI* 0020-0190(94)00207-X

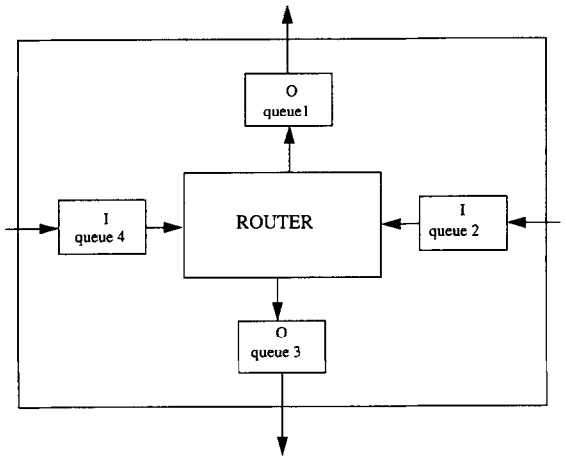


Fig. 1. The organization of a switching node for a directional interconnection network.

field of parallel computing for many reasons. It perfectly matches numerous algorithms such as divide-and-conquer algorithms and the classes of *descend* and *ascend* algorithms [14]. It provides a rich interconnection structure. Important interconnection networks such as the mesh, the pyramid, and the complete binary tree can be efficiently embedded in the hypercube [8,10,18]. It has a high bandwidth, thus many algorithms requiring a high transfer of data between sections of the network can be executed efficiently on the hypercube. Routing messages between nodes in the hypercube is particularly simple. On receiving a message, a node need only compare its address with the message's destination address. If the address differs in bit position $i$, the message should be sent on the link corresponding to dimension $i$. If the addresses are identical, the destination has been reached. If there are multiple bit differences, let $k$ be the rightmost bit (alternatively, leftmost) bit where the two addresses differ. The message should be sent on the link corresponding to dimension $k$ [20]. Several commercial multicomputers based on the hypercube interconnection network have been built [21].

The binary hypercube possesses one major liability despite its many advantages: the number of connections to each node is high. This makes it costly to build a hypercube interconnection network. Moreover, packaging technology places a limit on the number of I/O pins in an integrated circuit chip, and on the number of I/O ports a printed circuit board could have. This in turn places a limit on the number of hypercube nodes which can be placed in these units. To overcome this problem, we introduce in this paper an interconnection network which is identical to the hypercube except that each bi-directional link is replaced by a directional link in order to reduce the hardware cost and build larger network sizes. This interconnection network is referred to as a directional hypercube (Dcube). A related design choice has been independently made by Chou and Du [4]. They proposed two different schemes for the hypercube when directional links are utilized. However, their routing algorithms are not conveniently defined, and thus are far more complex than that of the traditional hypercube, possibly producing an intolerable overhead. The routing algorithm presented in this paper for the Dcube is as easy and efficient as that for the traditional hypercube. Moreover, they did not study important architectural aspects of this interconnection network such as its connectivity to give some insight into its fault-tolerance, its bandwidth to provide an indication of the congestion to be expected in such a network, and its hypercube emulation to relate the two networks together. All these aspects are treated in detail in this paper. Other directional interconnection networks have been proposed in [13].

This paper is organized as follows. In Section 2 we define the directional hypercube interconnection network. In Section 3 we present some of its key architectural characteristics. In Section 4 we present an efficient routing algorithm for the Dcube.

## 2. Directional hypercubes

A directional hypercube (Dcube) is a traditional binary hypercube with each bidirectional link replaced by a directional link. An $n$-Dcube is a Dcube with $N = 2^n$ PEs (Processing Elements)

where each PE has $n$ directional links. Links used for transmitting messages from a source PE to other PEs in the Dcube are referred to as OUT links at the source PE. An incoming link at a PE is referred to as an IN link. Thus associated with each link connected to a PE is a direction: IN or OUT. There are numerous ways of assigning directions to all of the links in the Dcube. The best assignment would be an assignment that retains the nice properties of the hypercube in the Dcube such as small diameter, strong connectivity, high bandwidth, and simple routing. Hence, assigning directions to the links of a Dcube is a crucial design choice.

The assignment function which we consider in this paper assigns the direction of each link depending only on the parity of the address of the link's source PE and the link number defined below. The address of a PE has an even parity if its binary representation has an even number of 1's, and it has an odd parity if its binary representation has an odd number of 1's. In a Dcube, half of the nodes have even parity and the other half have odd parity. Each link has an unsigned link number, $i$, which correspond to the bit position where the addresses of the two nodes connected to the link differ. The least significant bit of node addresses is assigned position 0 and $i$ satisfies $0 \leqslant i \leqslant n-1$. The link direction assignment function denoted by $DIR(A\,i)$ takes values IN or OUT where $A$ is the address of one of the PE's to which the link is connected, and $i$ is the link number.

**Definition 1.** $DIR(A, i)$ is defined in the following way:

- $DIR(A, 0) =$ OUT if $A$ has even parity.
- $DIR(A, 0) =$ IN if $A$ has odd parity.
- $DIR(A, i) = DIR(A, 0)$ if $i$ is even or $DIR(A, 0)'$ if $i$ is odd, where OUT′ = IN and IN′ = OUT.

Fig. 2 shows a Dcube of size 16 where the direction of each link is assigned by $DIR(A, i)$.

For each node $A$, we define an $n$-bit binary number, $MASK(A)$, which indicates the IN/OUT assignment determined by $DIR(A, i)$.

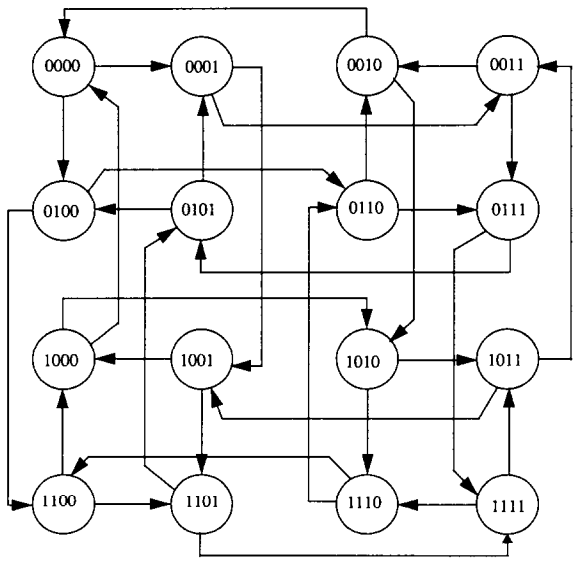


Fig. 2. A directional hypercube of size 16.

**Definition 2.** $MASK(A)$ is an $n$-bit binary number where the $i$th bit of $MASK(A)$ is 1 if the $i$th link of the PE whose address is $A$ is an OUT link, and 0 otherwise.

It follows that all of the PEs whose addresses have even parity have the same mask, and all of the PEs whose addresses have odd parity have the same mask. For example for $n$ even, $MASK(A) = 01\ldots0101$ if $A$ has even parity, and $MASK(A) = 10\ldots1010$ if $A$ has odd parity. The mask of each PE is very useful in routing messages in the Dcube, as will be seen in Section 4.

## 3. Characteristics of the Dcube

In this section we derive the diameter of the Dcube. Then, the message transfer capabilities (bandwidth) of the Dcube will be evaluated in terms of its bisection bandwidth, and compared to that of the hypercube. Its connectivity will also be evaluated as a measure of its fault tolerance. Finally, the efficiency of emulating the hypercube on the Dcube will be evaluated.

### 3.1. Diameter of the Dcube

The diameter of the Dcube is the maximum number of links that must be traversed in the shortest path between any two nodes. It is an important performance measure because it places a lower bound on the delay required to propagate information throughout the Dcube. Let $A$ (the source) and $B$ (the destination) be the addresses of two arbitrary PEs in an $n$-Dcube which differ in $L_d$ bit positions. From among these differences there are $L_I$ bits that correspond to IN links at $A$ and $L_O$ bits that correspond to OUT links at $A$, with $L_I + L_O = L_d \leqslant n$. Proceeding from $A$ to $B$, $L_I$ and $L_O$ will be alternatively decreased until they both equal zero (note that $L_I$ and $L_O$ at a certain node would correspond to $L_O$ and $L_I$ respectively at its neighboring node in the path to the destination because two neighboring nodes have different parities). There are certain primitive movements which are useful in illustrating the diameter of the Dcube. When we move along an OUT link of $A$ to resolve one of the $L_O$ differences, we refer to this as a *single link movement*. When we wish to resolve a single $L_I$ difference we define a *triple link movement* to achieve this.

**Definition 3.** A *triple link movement* is the traversal of three links to resolve a single $L_I$ difference between

$A = a_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0$ and

and

$B = \bar{a}_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0$

where $\bar{a}_{n-1}$ is the complement of $a_{n-1}$. Without loss of generality we assume the $L_I$ difference at link $n-1$. First we traverse any OUT link from $A$ along dimension $i$ to a node with address $I_j = a_{n-1}a_{n-2}\ldots\bar{a}_i\ldots a_1a_0$. Next we traverse the OUT link along the dimension where $A$ and $B$ differ leading to a node at address $I_k = \bar{a}_{n-1}a_{n-2}\ldots\bar{a}_i\ldots a_1a_0$. Finally, we traverse the OUT link along dimension $i$ to reach $B$. $I_j$ is guaranteed to have an OUT link to $I_k$ along the $n-1$ dimension since the parity of $I_j$ is different from that for $A$ and since the link at dimension $n-1$ is an IN link at $A$. $I_k$ is guaranteed to have an OUT link along dimension $i$ since $I_k$ and $A$ have the same parity and the link at dimension $i$ at $A$ is an OUT link.

This *triple link movement* is optimal as shown in the following proposition.

**Proposition 4.** *Resolution of a single $L_I$ difference between A and B requires a minimum of three link traversals.*

**Proof.** Let

$A = a_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0$ and

and

$B = \bar{a}_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0$

where the $L_I$ difference has been arbitrarily assumed at dimension $n-1$. Since the link between $A$ and $B$ is an IN link at $A$, we cannot go from $A$ to $B$ by using a single link. Thus, we must exist $A$ using an OUT link (at least $\lfloor n/2 \rfloor$ OUT links always exist at $A$) to an intermediate node $I_1 = a_{n-1}a_{n-2}\ldots\bar{a}_i\ldots a_1a_0$. We cannot go from $I_1$ to $B$ by using a single link since $I_1$ and $B$ differ in two bit positions. Hence, there is no path of length 1 or 2 from $A$ to $B$. □

Thus, there always exists a minimum length path from $A$ to $B$, where $A$ and $B$ are at a Hamming distance 1, which requires either 1 or 3 link traversals. Finally, we define a *double link movement* which resolves two $L_I$ differences.

**Definition 5.** A *double link movement* is the traversal of four links to resolve two $L_I$ differences between $A$ and $B$. This movement is the sequence of a *triple link movement* to an intermediate node $I_j$, which resolves one of the $L_I$ differences, followed by a *single link movement* to resolve the second $L_I$ difference. $I_j$ has an OUT link along the dimension corresponding to the second $L_I$ difference since $I_j$ and $A$ have different parities.

This *double link movement* is optimal for resolving two $L_I$ differences since the *triple link*

*movement* is optimal for resolving one $L_I$ difference leading to:

**Corollary 6.** *Resolution of two $L_I$ differences between A and B requires a minimum of four link traversals.*

**Proposition 7.** *The diameter of an n-Dcube is equal to $n+1$ if n is even and $n+2$ if n is odd.*

**Proof.** Only alternate steps in a path can reduce $L_I$ and the first step cannot reduce $L_I$, thus the shortest path between $A$ and $B$ should be $\geqslant 2L_I$. If $n$ is even, the maximum value of $L_I$ is $n/2$, hence the diameter is $\geqslant n$. If $n$ is odd, the maximum value of $L_I$ is $\lfloor n/2 \rfloor + 1$ and the diameter is $\geqslant n+1$. We refine these bounds by constructing a minimum length path for arbitrary $A$ and $B$ which meets or approaches these bounds. There are three types of movements that will be used to go from $A$ to $B$:

(1) *Single link movements* along an $L_O$ link as long as $L_O \neq 0$. Each of these movements takes one step and reduces $L_O$ by one.
(2) *Double link movements* when $L_O = 0$ and $L_I > 1$. Each of these movements takes 4 steps and reduces $L_I$ by 2.
(3) *Triple link movements* when $L_O = 0$ and $L_I = 1$. Each of these movements takes 3 steps and reduces $L_I$ by 1.

If $L_I = L_O$, only *single link movements* are required to go from $A$ to $B$, and thus the number of steps needed is equal to $2L_I$. When $L_I \neq L_O$, the above movements will be used to go from $A$ to $B$. We can always use *single link movements* until $L_O = 0$. Then if $L_I > 1$ we use *double link movements* to reduce $L_I$ to 0 or 1, and finally if $L_I = 1$ one *triple link movement* is used to get to $B$. Thus in the worse case $2(L_I - 1) + 3 = 2L_I + 1$ steps would be needed to go from $A$ to $B$, and the diameter is $\leqslant 2L_I + 1$. If $n$ is even, the maximum value of $L_I$ is $n/2$, and $n \leqslant$ diameter $\leqslant n+1$. The distance is equal to $n+1$ when $L_I = n/2$ and $L_O = n/2 - 1$, thus the diameter $= n+1$. If $n$ is odd, the maximum value of $L_I$ is $\lfloor n/2 \rfloor + 1$, and $n+1 \leqslant$ diameter $\leqslant n+2$. The distance is equal to $n+2$ when $L_I = \lfloor n/2 \rfloor + 1$ and $L_O = \lfloor n/2 \rfloor$, thus the diameter $= n+2$. □

### 3.2. Message capacity measures of the Dcube

The ability of an interconnection network to transfer a high volume of messages from one section of the network to another in one unit time is an important measure of performance. This factor is often used to set lower bounds on the time complexity of many parallel algorithms such as sorting and divide-and-conquer algorithms [19]. One measure of message transfer capability is *bisection bandwidth*. Bisection bandwidth is defined to be the maximum number of messages sent in one unit time from one half of the network to the other when the network is partitioned into two equal halves [9]. The bisection bandwidth of a hypercube of size $N$ is $N/2$ when it is partitioned into two equal halves along any dimension [8].

**Proposition 8.** *The bisection bandwidth of a Dcube of size N is $N/4$ when it is partitioned into two equal halves along any dimension.*

**Proof.** Let us construct an $n$-Dcube by connecting two $(n-1)$-Dcubes together, with the size of each of the $(n-1)$-Dcubes being $N/2$. There will be $N/2$ links connecting the two $(n-1)$-Dcubes. Among these $N/2$ links, $N/4$ links would be IN links for the first $(n-1)$-Dcube and OUT links for the second $(n-1)$-Dcube. The other $N/4$ links would be OUT links for the first $(n-1)$-Dcube and IN links for the second $(n-1)$-Dcube. This follows since in each $(n-1)$-Dcube, there are $N/4$ PEs who have addresses with even parity and $N/4$ PEs who have addresses with odd parity. Thus, the added links connected to PEs with different parities have different directions. Hence, the first $(n-1)$-Dcube can transfer $N/4$ messages in one unit time to the second $(n-1)$-Dcube, and the second $(n-1)$-Dcube can transfer $N/4$ messages in one unit time to the first $(n-1)$-Dcube. □

### 3.3. Connectivity of the Dcube

A strongly connected interconnection network is an interconnection network that has many disjoint paths between processor nodes [17] and hopefully approaches the upper bound deter-

mined by the degree of the nodes. These disjoint paths do not share any common nodes or links. The hypercube has always been recognized as a strongly connected graph. For a hypercube of size $N$, where $N = 2^n$, there are $n$ disjoint paths between any two nodes [3]. The following proposition shows that the Dcube is also a strongly connected network.

**Proposition 9.** *An n-Dcube has* $\lfloor n/2 \rfloor$ *disjoint paths between any two nodes, A and B, and none of these paths includes more than one node adjacent to either A or B.*

**Proof.** By induction on $n$. For the base case $n = 2$, the number of disjoint paths is obviously equal to 1 and no node is used more than once in this path. Now assume that Proposition 9 is true for $n = 3, 4, 5, \ldots, k$. For $n = k + 1$, let us construct a $(k+1)$-Dcube by connecting two $k$-Dcubes together. When $A \; xor \; B = k + 1$ ($A \; xor \; B$ is the bitwise Exclusive-OR of $A$ and $B$) then there are $\lfloor k + 1/2 \rfloor$ disjoint paths between them. This is shown by using the construct in [17] to first illustrate the existence of $k + 1$ disjoint paths in a binary hypercube. It is then easy to show that each of these paths is composed of links all directed in the same direction along the path. The construct for disjoint paths from $A$ to $B$ follows: these disjoint paths $p = 0, 1, 2, \ldots, \lfloor k + 1/2 \rfloor$ are chosen at each step $s = 0, 1, \ldots, n-1$ along links $2p + s \bmod n$ if the source node has even parity, or along the links $2p + 1 + s \bmod n$ if the source node has odd parity. This guarantees that each path (set of links) is composed of links all directed in the OUT direction.

When $A \; xor \; B < k + 1$ we can always find a dimension where $A$ and $B$ are within the same $k$-Dcube, $\Delta_1$. By the induction hypothesis we have $\lfloor k/2 \rfloor$ disjoint paths within $\Delta_1$. When $k$ is even $\lfloor k/2 \rfloor = \lfloor k + 1/2 \rfloor$ and we are done. When $k$ is odd we need to identify one more disjoint path. We will construct this path in the other, previously unused $k$-Dcube, $\Delta_2$. If $A$ has an OUT link into $\Delta_2$ and $B$ has an IN link from $\Delta_2$ then the last disjoint path is easily constructed in $\Delta_2$. If $A$ has an IN link into $\Delta_2$, then there exists a node $A_I$ in $\Delta_1$ which $A$ has an OUT link to and which has an OUT link to $\Delta_2$. By the induction hypothesis all but one OUT link of $A$ has been used in previous disjoint paths and the remaining unused OUT link is connected to $A_I$. If $B$ has an OUT link into $\Delta_2$, then there exists a node, $B_I$, in $\Delta_1$ for which $B$ has an IN link and which has an IN link from $\Delta_2$. By the induction hypothesis all but one IN link of $B$ has been used in previous disjoint paths and the remaining unused IN link is connected to $B_I$. Thus, there is always a disjoint path from $A$ into $\Delta_2$ and from $\Delta_2$ to $B$; therefore we can construct our final disjoint path in $\Delta_2$. Finally, the new path uses only one adjacent node of $A$ and $B$. □

### 3.4. Emulation of the hypercube on the Dcube

The commercial availability of hypercube parallel computers and their many interesting architectural properties have attracted extensive research on the design and implementation of parallel algorithms for these networks in numerous areas [2]. A very important property of any parallel network would be the emulation of the hypercube with a small degradation in time performance [18]. This means that all the algorithms that have been designed for the hypercube can be executed on the new network without making any changes on the algorithms themselves. This amounts to a big savings in time developing new software for the new architecture, taking advantage of all the efforts that have been put in designing algorithms for the hypercube. Now we will try to find the maximum number of steps needed by the Dcube to emulate a single step of the hypercube under any message distribution in the *worst* case. Hence, we assume that all the links in the hypercube are active. Since a hypercube of size $2^n$ has $n2^{n-1}$ bidirectional links, the maximum number of active messages at any single step is $n2^n$ if each bidirectional link can transmit two messages, one in each direction in just one step. Each bidirectional link can be regarded as two directional links with opposite directions. Thus in this context the hypercube has $n2^n$ directional links. Half of these links have the same direction as the related directional links in the Dcube and exist. The other half have an opposite direction and do not exist. It would require the Dcube at least 3 steps to transmit

messages on hypercube links that do not exist in the Dcube.

**Proposition 10.** *The Dcube can emulate the operations performed in one step by a hypercube of the same size in four steps in the worst case.*

**Proof.** Our emulation of the hypercube on the Dcube is done by separately emulating the movements along the hypercube links that have the same direction as the related directional links in the Dcube (e.g. exist in the Dcube) and the movements along the hypercube links that have opposite directions as the related directional links in the Dcube (e.g. do not exist in the Dcube), respectively, using the following steps:

1. Transmit all the messages on hypercube links that exist in the Dcube.
2. Transmit all the messages on hypercube links which do not exist in the Dcube in the following way:
   a. Transmit all messages originating in a processor with even parity and going through the $i$th link in the hypercube through the following links in the Dcube:
      i. Send the message through link $i-1$ to an intermediate processor $i_1$.
      ii. Send the message from $i_1$ to another intermediate processor $I_2$ through link $i$.
      iii. Send the message from $I_2$ to the destination processor through link $i-1$.
   b. Transmit all messages originating in a processor with odd parity and going through the $i$th link in the hypercube through the following links in the Dcube:
      i. Send the message through link $i+1$ to an intermediate processor $I_1$.
      ii. Send the message from $I_1$ to another intermediate processor $I_2$ through link $i$.
      iii. Send the message from $I_2$ to the destination processor through link $i+1$.

The steps in 2.a.i and 2.b.i, 2.a.ii and 2.b.ii, and 2.a.iii and 2.b.iii are each done in parallel in a total of three steps. It is easy to see that no link in the Dcube is used by more than one message within any step. Thus the Dcube in at *most* four steps can emulate any step in the hypercube under any message distribution. □

## 4. Routing on the Dcube

When a message is to be routed from one PE to another, the path it takes is determined by the routing algorithm. The routing algorithm is executed by the originating node and by every other node in the path to the destination. It is desirable that the routing algorithm be simple and require no complete knowledge of the entire network. It would be convenient by just knowing the source address and the destination address to obtain the exact and the shortest sequence of PEs the message must traverse to get to its destination [1]. One of the most important reasons for the popularity of the hypercube is the ease and effectiveness of message routing. One popular routing algorithm for the hypercube is given below as Algorithm 1 [20], known as $e$-cube routing.

**Algorithm 1.** (Send a message from PE1, whose address is $A$, to PE2, whose address is $B$ in hypercube using $e$-cube routing.)

```
If (A = B) Then
   Send message to local processor
Else
   Compute C = A xor B
   Starting with the most significant bit of C
      Let i be the bit number of the first 1 in C
   Send the message on link i
```

*A xor B* is the bitwise Exclusive-OR of $A$ and $B$. It will also be referred to as the relative address of $A$ and $B$.

A useful property of a message routing algorithm is that it does not *deadlock* [5]. Deadlock can occur if there is a cyclic dependency for resources. If two messages each hold resources required by the other to move, both messages will be blocked indefinitely. Typically, deadlock is avoided by the routing algorithm. It has been shown that $e$-cube routing, i.e. Algorithm 1, is deadlock-free [5].

A simple variant of Algorithm 1, described as Algorithm 2, works for the Dcube.

**Algorithm 2.** (Send a message from PE1 whose address is $A$ to PE2 whose address is $B$ in a Dcube.)

If ($A = B$) Then
  Send message to local processor
Else
Compute $C$ = A *xor* B *and MASK(A)*
  If ($C \neq 0$) Then
  Starting with the most significant bit of $C$
    Let $i$ be the bit number of the first 1 in $C$
    Send the message on link $i$
  Else
    Send the message on the most significant OUT link in PE1

We define *and* to be the bitwise AND of two binary numbers. One of the main and essential properties of any routing algorithm is its ability to route a message from a source to a destination through the shortest path between them. As it will be proved below, Algorithm 2 routes messages between any two nodes through the shortest path between them.

**Theorem 11.** *Given any two disjoint nodes whose addresses are A and B in a Dcube, Algorithm 2 routes messages between A and B through a shortest path between them.*

**Proof.** Let *reland* be the relative address of $A$ and $B$ *and*ed with $MASK(A)$. We say that *reland* sets bit $i$ if bit $i$ of *reland* is set to 1. There are two cases that could occur when routing messages between $A$ and $B$:

*Case* 1: At least one bit, $i$, is set in *reland*. The message would be sent through link $i$ of $A$ to a node which is one dimension closer to $B$, and the number of bits set in *reland* would decrease by 1.

*Case* 2: There is no bit set in *reland*. This means that either we have reached the destination or all the bit differences between $A$ and $B$ correspond to IN links in $A$. Then by using a *triple link movement*, the message would travel through three links to get one dimension closer to the destination which is the shortest path in this case as shown in Proposition 4.

Thus Algorithm 2 routes messages between any two nodes in the Dcube through the shortest path. □

Researchers have proposed a powerful routing method to avoid deadlock in interconnection networks using the concept of *virtual channels* [5]. Each group of virtual channels shares a physical communication channel (link); however, each virtual channel has its own queue [5]. In this case, deadlock is avoided through the routing algorithm by ordering the interconnection network virtual channels and requiring that messages request and use these virtual channels in strictly monotonic order [5].

To avoid message deadlock in the Dcube, we also use the concept of virtual channels. Thus, we split each Dcube physical channel into $(1/2)\log(N) + 3$ virtual channels where the Dcube is of size $N$. Hence, the virtual channels of each physical channel, $j$, are denoted by $C_0^j, C_1^j, C_2^j, \ldots, C_{(1/2)\log(N)+2}^j$ where we assume $C_0^j < C_1^j < C_2^j < \cdots < C_{(1/2)\log(N)+2}^j$. Moreover, $C_x^j < C_x^i$ and $C_y^i < C_x^j$ if $j < i$ and $y < x$. Then, to route a message from a source node to a destination node in the Dcube, the routing algorithm first selects the physical channel and then it chooses the virtual channel. We simply use Algorithm 2 for the selection of the physical channel. Virtual channels are chosen in the following way: initially, $C_{(1/2)\log(N)+2}$ channels are used. As long as the selected physical channels are chosen in a decreasing order, $C_{(1/2)\log(N)+2}$ virtual channels are chosen. When a physical channel corresponding to a dimension higher than the dimension of the last traversed physical channel must be used, the message will take the corresponding $C_{(1/2)\log(N)+1}$ virtual channel. Then, $C_{(1/2)\log(N)+1}$ virtual channels will be used as long as the physical channels are employed in a decreasing dimension order. When a physical channel corresponding to a dimension higher than the dimension of the last traversed physical channel must be used, the message will take the corresponding $C_{(1/2)\log(N)}$ virtual channel. This process continues until the message reaches its destination.

Obviously, by using the above described routing strategy, it is possible to establish an ordering between virtual channels. Hence, the routing al-

gorithm would be deadlock-free [5]. Now, it is necessary to prove that it will suffice with $(1/2)\log(N)+3$ virtual channels per physical channel to route a message from any source to any destination. First, each message is routed across useful dimensions in a decreasing order through the $C_{(1/2)\log(N)+2}$ virtual channels using *single link movements*. When we exhaust using all *single link movements* and the message has not reached the destination, then we have to use either a *double link movement* or a *triple link movement*. Remember that a *double link movement* is nothing but a *triple link movement* followed by a *single link movement*. Further, using a *triple link movement* to resolve a single $L_I$ difference at link $i$ between any two nodes, $A$ and $B$, involves crossing the following dimensions: $j, i, j$, where $j$ is the most significant OUT link in $A$. For more details, please refer to Section 3. Clearly, the sequence of physical channels $j, i, j$ are not ordered if they use the same virtual channels (i.e. $C_x^j$, $C_x^i$, $C_x^j$). Thus, we have to establish an order between them to avoid deadlock. If $A$ has an odd parity, then $j > i$, as it is the most significant bit in $A$. In this case, if the previous crossed virtual channel is $k$, then we should use the virtual channel $k$, then we should use the virtual channel $k-1$ of physical link $j$ ($C_{k-1}^j$). Next we use the virtual channel $k-1$ of physical link $i$ ($C_{k-1}^i$) followed by virtual channel $k-2$ of physical link $j$ ($C_{k-2}^j$). Hence, the sequence of physical channels $j, i, j$ would be crossed using the following virtual channels: $k-1$, $k-1$, $k-2$ which preserve the required order for deadlock-free routing (i.e. $C_{k-1}^j$, $C_{k-1}^i$, $C_{k-2}^j$). The next step after this *triple link movement* must be a *single link movement* across a link $l < j$ or else we have reached the destination. If we have to use a *single link movement*, then we use virtual link $k-2$ of physical link $l$ ($C_{k-2}^l$). Hence, the maximum number of virtual channel labels needed for this *double link movement* or *triple link movement* is 2 (i.e. $k-1$, $k-2$). This process continues until we reach the destination.

On the other hand, if $A$ has an even parity, then we may have two cases: either $i > j$ or $j > i$. If $i > j$, which may happen *only* for the first *triple link movement* after we exhaust all *single link movements*. In this case, if the previous crossed virtual channel is $k$, then we should use the virtual channel $k-1$ of physical link $j$ ($C_{k-1}^j$). Next we use the virtual channel $k-2$ of physical link $i$ ($C_{k-2}^i$) followed by virtual channel $k-2$ of physical link $j$ ($C_{k-2}^j$). Hence, the sequence of physical channels $j, i, j$ would be crossed using the following virtual channels: $k-1$, $k-2$, $k-2$ which preserve the required order for deadlock-free routing (i.e. $C_{k-1}^j$, $C_{k-2}^i$, $C_{k-2}^j$). The next step after this *triple link movement* must be a *single link movement* across a link $l < j$ or else we have reached the destination. If we have to use a *single link movement*, then we use virtual link $k-2$ of physical link $l$ ($C_{k-2}^l$). Hence, the maximum number of virtual channel labels needed for this *double link movement* or *triple link movement* is 2. If $j > i$, then if the previous crossed virtual channel is $k$, then we should use the virtual channel $k-1$ of physical link $j$ ($C_{k-1}^j$). Next we use the virtual channel $k-1$ of physical link $i$ ($C_{k-1}^i$) followed by virtual channel $k-2$ of physical link $j$ ($C_{k-2}^j$). Hence, the sequence of physical channels $j, i, j$ would be crossed using the following virtual channels: $k-1$, $k-1$, $k-2$ which preserve the required order for deadlock-free routing (i.e. $C_{k-1}^j$, $C_{k-1}^i$, $C_{k-2}^j$). The next step after this *triple link movement* must be a *single link movement* across a link $l < j$ or else we have reached the destination. If we have to use a *single link movement*, then we use virtual link $k-2$ of physical link $l$ ($C_{k-2}^l$). Hence, the maximum number of virtual channel labels needed for this *double link movement* or *triple link movement* is 2 (i.e. $k-1$, $k-2$). Thus, in all cases, whenever we need to use a *triple link movement* (*double link movement*), we must use 2 different virtual channel labels to preserve the channel ordering.

Therefore, in order to determine the maximum number of virtual channels per physical channel, we have to determine the maximum number of *triple link movements* needed to route a message between any two nodes in the Dcube. Referring to Section 3, we can easily deduce that the maximum number of *triple link movements* between any two nodes, $A$ and $B$, happens when $L_O < L_I$ and $L_O$ is equal to $\lfloor L_I/2 \rfloor + 1 = (1/4)\log(N) + 1$. Hence, the total number of virtual channels needed to route a message between

any two nodes in the Dcube in a deadlock-free manner is $1 + [(1/2)\log(N) + 2]$ where the first term, 1, corresponds to the number of virtual channels needed for the *single link movements*, and the second term, $(1/2)\log(N) + 2$, corresponds to the number of virtual channels needed for the *double link movements*, and consequently the *triple link movements*. Consequently, if the size of the Dcube is *very* large, this would require a large number of virtual channels per physical channel in the Dcube to achieve deadlock-free routing. This is one of the major drawbacks of the Dcube. It would be very interesting as a future research direction to find the optimum arrangements of directional links in the Dcube that would lead to the least number of virtual channels per physical channel to achieve deadlock-free routing.

## 5. Conclusion

We have presented a variation of the hypercube interconnection network where each bidirectional link is replaced by a directional link. The directional hypercube is shown to preserve the nice properties of the hypercube such as small diameter, high bandwidth, and strong connectivity. We have shown that the Dcube can emulate the hypercube with a small constant degradation in time performance under any message distribution. A simple routing algorithm has been derived for the Dcube requiring only local information to route messages between nodes, and it is just a variation of the original routing algorithm for the hypercube. Then, the concept of virtual channels has been added to the routing algorithm to make it deadlock-free. Thus, the Dcube may have some potential as a lower cost substitute for the binary hypercube in many applications.

## References

[1] D.P. Agrawal, V.K. Janakiram and G.C. Pathak, Evaluating the performance of multicomputer configurations, *IEEE Computer* **19** (1986) 23–37.

[2] S.G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[3] J.R. Armstrong and F.G. Gray, Fault diagnosis in a Boolean $n$-cube array of microprocessors, *IEEE Trans. Comput.* **30** (1981) 587–590.

[4] C.H. Chou and D.H.C. Du, Uni-directional hypercubes in: Supercomputing 90 (1990) 254–263.

[5] W. Dally and C. Seitz, Deadlock-free message routing in multiprocessor interconnection networks, *IEEE Trans. Comput.* **36** (1987) 547–553.

[6] S.A. Felperin, L. Gravano and J.L.C. Sanz, Routing techniques for massively parallel communication, *Proc. IEEE* **79** (1991) 488–503.

[7] M. Hamdi, Cost-effectiveness of directional interconnection networks for parallel computer systems, *IEEE Trans. Comput.*, submitted for publication.

[8] M. Hamdi and R.W. Hall, Compound graph networks for parallel image processing, in: *Proc. 1991 Workshop on Computer Architecture for Machine Perception* (1991) 365–377.

[9] L.H. Jamieson, D.B. Gannon and R.J. Douglass, eds., *The Characteristics of Parallel Algorithms* (MIT Press, Cambridge, MA, 1987).

[10] S.L. Johnsson, Communication efficient basic linear algebra computations on hypercube architectures, *J. Parallel Distributed Comput.* **4** (1987) 133–171.

[11] S. Konstantinidou and L. Snyder, Chaos router: A practical application of randomization in network routing, in: *Proc. Symp. on Parallel Algorithms and Architectures* (1990) 21–30.

[12] S. Konstantinidou and L. Snyder, Chaos router: Architecture and performance, in: *Proc. 18th Internat. Symp. on Computer Architecture* (1991) 212–221.

[13] A. Maxemchuck, Regular and mesh topologies in local and metropolitan area networks, *AT & T Tech. J.* **64** (1985) 1659–1686.

[14] F.P. Preparata and J. Vuillemin, The cube-connected cycles: A versatile network for parallel computations, *Comm. ACM.* **24** (1981) 300–309.

[15] D.A. Reed and R.M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing* (MIT Press, Cambridge, MA, 1987).

[16] D.A. Reed and G. Fox, Adaptive packet routing in a hypercube in: *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications* (1988) 334–341.

[17] Y. Saad and M.H. Schultz, Topological properties of hypercubes, *IEEE Trans. Comput.* **37** (1988) 867–872.

[18] Q.F. Stout, Mapping vision algorithms to parallel architectures, *IEEE Proc.* **76** (1988) 982–995.

[19] Q.F. Stout, Supporting divide-and-conquer algorithms for image processing, *J. Parallel Distributed Comput.* **4** (1987) 95–115.

[20] H. Sullivan and T.R. Bashkow, A large scale homogeneous, fully distributed parallel machine, I, in: *Proc. 4th Symp. on Computer Architecture* (1977) 105–117.

[21] A. Trew and G. Wilson, eds., *Past, Present, Parallel: A Survey of Available Parallel Computer Systems* (Springer, New York, 1991).