Computer Science
Technical Report

Colorado State University

Node-to-Set Vertex Disjoint Paths in Hypercube Networks*

Shahram Latifi†, Hyosun Ko‡, and Pradip K Srimani§

Technical Report CS-98-107

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*To appear in the Proceedings of **1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '98)**, Las Vegas, July 13–16, 1998
†Department of Electrical Engineering, University of Nevada, Las Vegas, NV 89154
‡Department of Electrical Engineering, University of Nevada, Las Vegas, NV 89154
§Department of Computer Science, Colorado State University, Ft. Collins, CO 80523

Node-to-Set Vertex Disjoint Paths in Hypercube Networks*

Shahram Latifi†, Hyosun Ko‡, and Pradip K Srimani§

1 Introduction

Design features for an efficient interconnection topology include properties like low degree, regularity, small diameter, high connectivity, efficient routing algorithms, high fault-tolerance, low fault diameter etc. Since more and more processors must work concurrently these days in a multiple processor environment, the criterion of high fault tolerance and computing node disjoint paths has become increasingly important. One of the most efficient interconnection network has been the well known binary n-cubes or hypercubes; they have been used to design various commercial multiprocessor machines [1]. The fault tolerance of an interconnection network is usually measured by the *vertex connectivity* of the underlying graph as well as the *fault diameter*. Vertex connectivity of an n-cube (which is a n-regular graph) is n and the corresponding fault diameter is $n+1$ (the fault-free diameter is n) [2]. It is also important to compute the set of node disjoint paths given a source node and a set of k distinct destination nodes. Menger's theorem guarantees the existence of n such paths between a source and a set of n destination nodes in a hypercube Q_n of dimension n; but, it's non trivial to identify those paths and even more so to make those paths as minimal as possible. Our purpose in this paper is to design a simple algorithm to compute such paths in Q_n and to show that the length of each of the paths is upper bounded by $n + 1$. This result was previously known [3]; but the proof in [3] is complicated. Our algorithm is much simpler and it is much easier to argue the correctness of the bound on length. We expect the technique would be useful to prove similar results for other structured graphs.

*To appear in the Proceedings of **1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '98)**, Las Vegas, July 13–16, 1998

†Department of Electrical Engineering, University of Nevada, Las Vegas, NV 89154

‡Department of Electrical Engineering, University of Nevada, Las Vegas, NV 89154

§Department of Computer Science, Colorado State University, Ft. Collins, CO 80523

2 Hypercube Q_n of Dimension n

Hypercubes are introduced under different names (n-cube, binary n-cube, Boolean n-cube, etc.). A hypercube Q_n, of order n, is defined to be a symmetric graph $G = (V, E)$ where V is the set of 2^n vertices, each representing a distinct n-bit binary number and E is the set of symmetric edges such that two nodes are connected by an edge iff the Hamming distance between the two nodes is 1, i.e., the number of positions where the bits differ in the binary labels of the two nodes is 1. Links (edges) are also labeled such that link i, $0 \leq i \leq n-1$, connects two nodes which differ in the i-th bit position, with the rightmost bit being in the 0-th position. For example, in Q_3, the node 010 is connected to node 110 by link 2, to node 000 by link 1 and to node 011 by link 0. Any sub-cube Q_{n-m} of a Q_n may be described by $(n - m)$ fixed bits and m arbitrary bits each denoted by "x". For example, a sub-cube Q_2 of Q_5 consisting of 4 nodes $\{00100, 00110, 01100, 01110\}$ is denoted as $0x1x0$. Similarly, $x^{n-1}0$ denotes the $(n-1)$ dimensional sub-cube of Q_n with all nodes with the last bit 0. The *weight* of a node v in Q_n is defined to be the number of 1's in the binary label of v and the Hamming distance of two nodes u and v is denoted by $H(u, v)$.

3 One to Many Disjoint Paths in Q_n

Given a hypercube Q_n, a source node $s = (00..0)$ and a set of ℓ distinct destination nodes $D = \{d_1, d_2, \cdots, d_\ell\}$, $1 \leq \ell \leq n$, the objective in this section is to design an algorithm to compute node disjoint paths from the source node to the destination nodes.

Remark 1 *Q_n is node symmetric [2] and hence we can consider the identity node (the all zero node $(00..0)$) to be the source node for our purpose without any loss of generality. We also note that Q_n (denoted also by x^n) consists of two sub-cubes $x^{n-1}1$ and $x^{n-1}0$. A node u in Q_n is denoted as $u = (u_{n-1}u_{n-2}\cdots u_0)$ where u_i*

denotes the i-th bit of the node label, $0 \leq i \leq n-1$ (the dimensions are numbered 0 through $n-1$ from least significant bit (LSB) position to most significant bit (MSB) position.

Definition 1 *A path $P(u,v)$ in Q_n between two nodes u and v is denoted by a sequence of link labels. For example, the path $P(0001,1010) = 0001 \longrightarrow 0000 \longrightarrow 0010 \longrightarrow 1010$ in Q_4 is denoted by $P(0001,1010) = (0,1,3)$. Note that $P(1010,0001) = (0,1,3)$ denotes a different path e.g., $1010 \longrightarrow 1011 \longrightarrow 1001 \longrightarrow 0001$, between the same two nodes. Since the sequence of link labels is ordered, we apply the sequence $P(u,v)$ starting at node u and reaching node v. The length of a path is defined to be the number of links in the path.*

Definition 2 *Given two nodes u and v in Q_n, the* first Hamming distance *path $FHP(u,v)$ is the minimal (shortest) path obtained by flipping the non matching bits in ascending order of dimensions. For example in Q_4, $FHP(0001,1010) = (0,1,3) = 0001 \rightarrow 0000 \rightarrow 0010 \rightarrow 1010$.*

Definition 3 *Consider an arbitrary set of $(n-1)$ distinct nodes $V = \{v^1, v^2, \cdots v^{n-1}\}$ in Q_n. $\mathcal{N}_V(v^i)$, $1 \leq i \leq n-1$, is defined to be an adjacent node $v' = v'_{n-1}v'_{n-2}\cdots v'_1v'_0$ of v^i such that $v'_{n-1}\cdots v'_1 0 \notin V$ and v^i is obtained as follows: (1) if possible, v' is obtained from v^i by inverting the first "1" (in v^i) in descending dimension order, or (2) if v' cannot be obtained by (1), then v' is obtained from v^i by inverting the first "0" (in v^i) in descending dimension order. $\eta_V(v^i)$ is defined to be the bit position of v^i that is reversed to get v'.*

Example 1: Consider $V = \{0101, 0111, 0110\}$ in Q_4; we have $\mathcal{N}_V(0111) = 0011$ and $\eta_V(0111) = 2$. As another example, consider $V = \{0101, 0001, 0100\}$in Q_4; we get $\mathcal{N}_V(0101) = 1101$ and $\eta_V(0101) = 3$. Note that in the second example $\mathcal{N}_V(0101)$ cannot be obtained by reversing any of the "1" bits of the node 0101. $\diamond$

Lemma 1 *Given any set V of $(n-1)$ distinct nodes in Q_n, $\mathcal{N}_V(v)$, $v \in V$, always exists.*

Proof : Consider an arbitrary node $v \in V$. Since $v \in Q_n$, node v has n distinct neighbors in Q_n and $|V| = n-1$, $\mathcal{N}_V(v)$ always exists. $\square$

Our strategy in designing the algorithm is to use a recursive approach – we trace a path to one of the destination nodes restricted to one of the two component sub-cubes and then map the rest of the destination nodes onto the other sub-cube and then repeat the process.

- Consider two sub-cubes $x^{n-1}0$ and $x^{n-1}1$ of Q_n. Of n destination nodes, consider the set $\mathcal{S}_1$ of those nodes that are in $x^{n-1}1$ and choose one, say $v \in \mathcal{S}_1$ (resolve ties arbitrarily), closest to the source node s and trace a shortest (FHP) path from node s to node v. Note that since the FHP is computed in ascending order of the dimensions, all points in this path are in$x^{n-1}1$.
- We delete node v from the set D (also from $\mathcal{S}_1$) and map the rest of the nodes in $\mathcal{S}_1$ to the sub-cube $x^{n-1}0$ (note that the source node s is in $x^{n-1}0$).
  - If for any node $y = y_{n-1}\cdots y_1y_0$ in $\mathcal{S}_1 - \{v\}$, there does not exist any node $w = w_{n-1}\cdots w_1w_0$ in $D - \{v\}$ such that $y_{n-1}\cdots y_1 0 = w = w_{n-1}\cdots w_1 0$, then use $y_{n-1}\cdots y_1 0$ as the new destination node in $x^{n-1}0$ (corresponding to the node y);
  - otherwise, use the node $w_{n-1}\cdots w_1 0$ as the new destination node in $x^{n-1}0$ (corresponding to the node y), where $w = \mathcal{N}_{D-\{v\}}(y)$.
- We need to map the nodes in $D - \mathcal{S}_1$. If $\mathcal{S}_1 = \emptyset$, then we choose one, say $v \in D$, farthest to the source node, and go from node v to v' by traversing link 0 and then trace a shortest (FHP) path from node s to the node v' (all points in the path are in $x^{n-1}1$). For each remaining node $y = y_{n-1}\cdots y_1y_0$ in $D - \mathcal{S}_1$, use $y_{n-1}\cdots y_1 0$ as the new destination node in $x^{n-1}0$ (corresponding to the node y).
- We now have a set of $(n-1)$ destination nodes and a source node in $x^{n-1}0$ (which is a $n-1$ dimensional hypercube); invoke the algorithm recursively.

Example 2: Consider the source node $s = (00000)$ and the set of destination nodes $D = \{01100, 11100, 01010, 00010, 01110\}$ in Q_5. Note that at this stage of the algorithm the set $\mathcal{S}_1$ is empty. The nodes 01110 and 11100 both have the maximum weight of 3. We choose 01110 arbitrarily, and compute the path $P(00000, 01110) = \{0, 3, 2, 1\}$. At the second stage of recursion we have a source node (0000) and a set of destination nodes

Destination Nodes	Path from Source to Destination
01100	23
11100	432
01010	31
00010	1
01110	03210

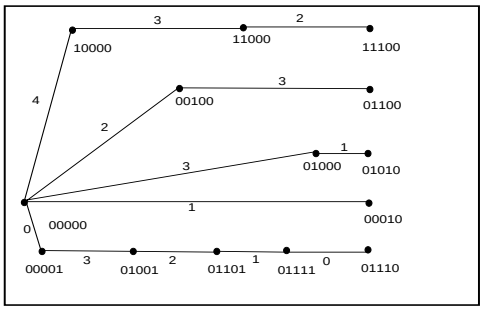

Figure 1: Stepwise Execution of the Algorithm in Q_5

$\{0110, 1110, 0101, 0001\}$. At this stage, $\mathcal{S}_1$ consists of two nodes $\{0101, 0001\}$ of which the node 0001 is closer to the source and we compute the path $P(0000, 0001) = \{0\}$. For the other node 0101, we go to its image 01000 in $x^3 00$ and take the images of the other nodes as well. So, at the third stage of recursion, we have a source node 000 and a set of destination nodes $\{011, 111, 010\}$. Farthest node in $\mathcal{S}_1$ is 111 and we compute the path $P(000, 111) = \{4, 3, 2\}$. For the other node 011 in the set $\mathcal{S}_1$ we cannot simply take its image in $x^2 000$ since it will then coincide with the other remaining destination node; hence, we go to its nearest neighbor 000 (by following the path $\{3, 2\}$) which happens to be source node. See Figure 1 for details; note that the paths in the table are given as link sequences from the source node to the destination nodes. $\diamond$

Example 3: Consider the source node $s = (0000000)$ and the set of destination nodes $D = \{0001100, 0101001, 0111011, 1010111, 1100010, 1110000, 1110010\}$. At this stage, $\mathcal{S}_1$ consists of 3 nodes $\{0101001, 0111011, 1010111\}$ of which 0101001 is closest to the source node and hence we trace the path $P(s, 0101001) = (0, 3, 5)$. At the second stage of recursion, we have six destination nodes

Destination Nodes	Path from Source to Destination
0001100	23
0101001	035
0111011	34510
1010111	64210
1100010	156
1110000	564
1110010	4516

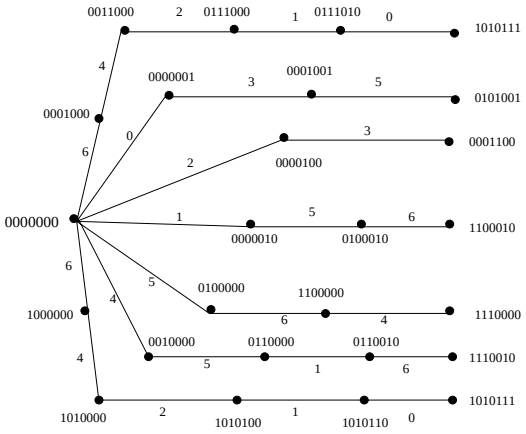

Figure 2: Stepwise Execution of the Algorithm in Q_7

$\{000110, 011101, 101011, 110001, 111000, 111001\}$ and a source node $s = (000000)$ in Q_6. $\mathcal{S}_1$ consists of 4 nodes $\{011101, 101011, 110001, 111001\}$ of which 110001 is closest to the source node and hence we trace the path $P(000000, 110001) = (1, 5, 4)$ – note that this is also the path $P(0000000, 1100010)$ in the original problem. The rest of the computation is similar. See Figure 2 for the details; note that the paths in the table are given as link sequences from the source node to the destination nodes. $\diamond$

We now can present the pseudo-code of the algorithm that generates the disjoint paths in Q_n given a source node $s = 00..0$ and n destination nodes. $\boldsymbol{P}[\,]$ and $\boldsymbol{Q}[\,]$ are two array variables of ordered sets (sequences) to store the generated link label sequences for the node disjoint paths. $\boldsymbol{D^k}$ variables are set variables to hold the destination nodes at each stage of recursion. We use one primitive function $\mathbf{II}$: $\boldsymbol{x}\,\mathbf{II}\,\boldsymbol{y}$, where both x and y are ordered set variables, returns a new sequence obtained by appending the sequence y to the sequence x. See Figure **??** for the complete pseudo-code.

Lemma 2 *The n paths, generated by the algorithm are*

Function Node_Disjoint_paths (k, D^k, P[1..k], Q[1..k-1])

(0) **if** $k = 1$ **then exit**;
(0) $j = k$; $D^{k-1} = \emptyset$;
(1) Compute $\mathcal{S}_1 = \{y \mid y \in D^k \wedge y \in x^{k-1}1\}$;
(2) **if** $\mathcal{S}_1 \neq \emptyset$ **then**
(3) **begin**
(4) Select v such that $w(v) = \min_{u \in \mathcal{S}_1}\{w(u)\}$;
(5) Compute the path $FHP(s, v)$; $P[j] = FHP(s, v) \amalg Q[j]$;
(6) For each $y \in \{\mathcal{S}_1 - v\}$ **do**
() $j = j - 1$;
(7) **if** $y_{n-1}y_{n-2}\cdots y_1 0 \notin \{D^k - v\}$
(8) **then**
(9) **begin**
(10) $D^{k-1} = D^{k-1} \cup y_{n-1}y_{n-2}\cdots y_{n-k+1}$;
(11) $P[j] = \{n-k\} \amalg Q[j]$;
(13) **end**
(14) **else**
(15) **begin**
() Compute the node $u = \mathcal{N}_{\{D^k - v\}}(y)$; $x = \eta_{\{D^k - v\}}(y)$;
() $D^{k-1} = D^{k-1} \cup u_{n-1}u_{n-2}\cdots u_{n-k+1}$:
() $P[j] = \{n-k, x\} \amalg Q[j]$;
() **end**;
() **end**
() **if** $\mid D^k - S_1 \mid \neq 0$ **then**
( ) **begin**
() **if** $j = k$ **then** /* $\mathcal{S}_1 = \emptyset$ *\
() **begin**
(17) Select v such that $w(v) = \max_{u \in D^k}\{w(u)\}$; $D^k = D^k \cap \{v\}$;
(18) Compute the path $\rho = FHP(s, v_{n-1}v_{n-2}\cdots v_{n-k+1}1)$; $P[j] = \{k\} \amalg \rho \amalg Q[j]$;
end;
(19) For each $y = y_{n-1}y_{n-2}\cdots y_{n-k} \in \{D^k - S_1\}$ **do**
(20) $D^{k-1} = D^{k-1} \cup y_{n-1}y_{n-2}\cdots y_{n-k+1}$;$j = j - 1$; $P[j] = \{k\} \amalg Q[j]$
( ) **end**
() Node_Disjoint_paths ($k-1$,D^{k-1}, P[1..k-1], P[1..k-1]);

mutually node-disjoint.

Proof : The node-disjoint property follows from the fact that at each stage of recursion, we are computing the path from **one** destination node to the source node (at any stage the hypercube has two sub-cubes: the 0-sub-cube and the 1-sub-cube say; the path is traced only in the 1-sub-cube) and mapping the other destination nodes to distinct nodes in the 0-sub-cube (intermediate nodes on the path are always chosen from the 1-sub-cube). Thus, the paths computed at any stage of recursion cannot intersect with each other. Also note that the nearest neighbor node always exists since there are always one fewer node to have conflict with than the degree of a node. Note that in step 2 only the closest node to the source is considered so that the path will not go through any other destination node at that stage. Similarly for the path computed for a node in $D - S_1$, the path consists of nodes in the 1-sub-cube while all destination nodes are in the 0-sub-cube. □

Remark 2 *The length of a path, from a destination node to the source node, generated by the CSR algorithm, can be greater than its distance to the source node iff we pay a penalty by flipping a "0" to a "1" in generating the FHP at some stage(s) of the algorithm; each time we pay a penalty, the path length increases by 2.*

Assume that for a given set D of n destination nodes, our CSR algorithm generates a path of length $\geq n+2$ in Q_n. Let the specific destination node be v (in D) such that $w(v) = m$.

Lemma 3 $2 \leq m \leq n-2$.

Proof : m cannot be 1 since for the path to node v to be of length $\geq n+2$ the node must enter the set $\mathcal{S}_1$ at some stage of recursion to pay the penalty.

$m = n-1$ is not possible since in that case there is only one zero in the bit string of node v and there is only one penalty possible and hence the length of the path can be at most $n-1+2 = n+1$. □

Assume that for the path we pay x number of penalties (i.e., we flip an original "0" to "1" x times) – each penalty adds 2 to the path length – thus, we must have

$$m + 2x > n + 1$$

Now, let us see what happens to the node v when we execute the algorithm; assume $v = 000...11..1$ without any loss of generality with all the m ones at the end. For us to pay the penalty, there must be at least "m" destinations which will coincide with the label of the troublesome node after changing one of its "m" 1's to 0 as well as there is one node that is already mapped to the source node; so, for the first penalty for node v, there are at least $m+2$ nodes (including v). Consider the scenario of next penalty – the node v (in its reduced version in a sub-cube of the original cube) has still m 1's (penalty means one 0 was flipped to 1) – for us to penalty, there again must be m other offending nodes – only one of them can be from the offending nodes of the previous stage due to the special 1 in the node v because of previous penalty. Repeat the argument to get

$$m + 2 + (x-1)(m-1) \leq n$$

or

$$(m-1)x \leq n-3$$

Combining the two inequalities we see that in order for a path to node v to have a length greater than $n+1$, we must have $m = 2$. If $m = 2$, then the minimum value of n is 7 in order to satisfy the inequalities and the corresponding x is 4. This is impossible in the execution of the algorithm CSR since m must be bigger than x (for node v to pay penalty at at any stage, v must be in $\mathcal{S}_1$ for that stage, i.e., the rightmost bit of v must be "1"; also note that a "0" is flipped to "1" (as a penalty) in a left to right fashion and the algorithm proceeds on a right to left fashion); similarly, larger values of n and x with $m = 2$ are also ruled out since m must be bigger than x. Thus, we can state the following theorem.

Theorem 1 *The maximum length of any path traced by the algorithm CSR in a hypercube Q_n is $n+1$.*

4 Conclusion

We have proposed a simple and efficient algorithm to compute the n vertex disjoint paths in a hypercube Q_n of dimension n, given a source node and an arbitrary set of at most n destination nodes; our algorithm is computationally much simpler than that of [3].

References

[1] C. L. Seitz. The cosmic cube. *Communications ACM*, 28(1):22–33, January 1985.

[2] Y. Saad and M. H. Shultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, July 1988.

[3] M. A. Rabin. Efficient dispersal of information for security. *Journal of ACM*, 36(2):335–348, 1989.