Chapter 13
Embedding Interconnection Networks in Crossed Cubes

Emad Abuelrub

Abstract The hypercube parallel architecture is one of the most popular interconnection networks due to many of its attractive properties and its suitability for general purpose parallel processing. An attractive version of the hypercube is the crossed cube. It preserves the important properties of the hypercube and most importantly reduces the diameter by a factor of 2. In this chapter, we show the ability of the crossed cube as a versatile architecture to simulate other interconnection networks efficiently. We present new schemes to embed complete binary trees, complete quad trees, and cycles into crossed cubes.

Keywords Binary trees · crossed cubes · cycles · embedding · interconnection networks · quad trees

13.1 Introduction

Hypercube architectures are loosely coupled parallel processors based on the binary cube network. Parallel computers based on the hypercube topology have gained widespread acceptance in parallel computing. Recently, many machines based on the hypercube have been designed and made commercially available. The hypercube offers a rich interconnection topology with large bandwidth, logarithmic diameter, simple routing and broadcasting of data, recursive structure that is naturally suited to divide and conquer applications, and the ability to simulate other interconnection networks with minimum overhead [3, 13, 19]. Due to the popularity of the hypercube, many variations of the hypercube topology have been proposed to improve on its properties and computational power [7, 13, 17, 20]. Efe [6] proposed an attractive version of the hypercube called the crossed cube, where preliminary studies proved

E. Abuelrub (✉)
Department of Computer Science, Faculty of Science and IT,
Zarqa Private University, Zarqa 13132, Jordan
e-mail: abuelrub@zpu.edu.jo

S.-I. Ao and L. Gelman (eds.), *Electronic Engineering and Computing Technology*,
Lecture Notes in Electrical Engineering 60, DOI 10.1007/978-90-481-8776-8_13,

that the crossed cube preserves many of the attractive properties of the hypercube and more importantly reduces the diameter by a factor of 2 [1, 2, 4, 6, 8, 11, 15]. This implies that the crossed cube has an advantage over the hypercube when data communication is of major concern. It is well known that for parallel architectures, data communication cost dominates computation cost. Therefore, it is worthwhile to make comparative studies on crossed cubes and other interconnection networks, and explore the advantages provided by them. The problem of embedding one interconnection network into another is very important in the area of parallel computing for portability of algorithms across various architectures, layout of circuits in VLSI, and mapping logical data structures into computer memories.

The importance of binary trees comes from the fact that this class of structures is useful in the solution of banded and sparse systems, by direct elimination, and captures the essence of divide and conquers algorithms. Embedding binary trees into other interconnection networks attracted the attention of many researchers. Barasch et al. have considered embedding complete binary trees into generalized hypercubes [3]. Dingle and Sudborough considered simulation of binary trees and X-trees on pyramid networks [5]. Quad trees are becoming an important technique in the domain of image processing, computer graphics, robotics, computational geometry, and geographic information systems [16, 18]. This hierarchical structure is based on the principle of recursive decomposition, which is similar to divide and conquer methods. Mapping quad trees into other interconnection networks attracted the attention of many researchers [1]. The problem of embedding rings or cycles into other interconnection networks has been studied by many researchers. It is well known that rings can be embedded into hypercubes using cyclic Gray codes [19]. Latifi and Zheng [11] generalized the cyclic Gray code method to embed rings into twisted cubes. Many researchers have addressed the problem of embedding rings into hypercubes in the presence of faults. On the other hand, other researchers addressed the problem of embedding rings into fault-free and faulty topologies [8, 11, 12, 14] or the Hamiltonicity of such structures in fault-free and faulty environments [9, 10, 15].

The remainder of this chapter is organized as follows. In Section 13.2, we establish a few preliminary definitions and notations. Section 13.3 explains straight forward scheme to embed complete binary trees into the crossed cubes. Section 13.4 presents a recursive technique to embed complete quad trees. In Section 13.5, we extend the Gray code scheme to embed cycles into the crossed cube. Finally, Section 13.6 concludes the chapter and discusses some future possible work.

13.2 Definitions and Notations

In this chapter, we use undirected graphs to model interconnection networks. Each vertex represents a processor and each edge a communication link between processors. The embedding of a guest graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$ is an injective mapping f from V_G to V_H, where V_G, E_G and V_H, E_H are the vertex

and edge sets of G and H, respectively, and where $|V_H| \geq |V_G|$. We consider a complete binary tree of height $n-1$, a complete quad tree of height $n-1$, and a cycle of size n, denoted CB_n, CQ_n, and C_n, respectively, as guest graphs and a crossed cube of dimension n, denoted XQ_n, as a host graph. Two cost functions, dilation and expansion often measure the quality of an embedding. If u and v are two nodes in G, then the distance from u to v, $d = (u, v)$, is the length of the shortest path from u to v. The *Dilation* (D) is the maximum distance in H between the images of adjacent vertices of G, $D = \max\{d(f(u), f(v)), \text{where } u\text{–}v \in E_G\}$. The *Expansion* (E) is the ratio of the cardinality of the host vertex set to the cardinality of the guest vertex set, $E = |V_H|/|V_G|$. Minimizing each of these measurements has a direct implication on the quality of the simulation of the guest network by the corresponding host network. The dilation of an embedding measures how far apart neighboring guest processors are placed in the host network. Clearly, if adjacent guest processors are placed far apart in the host network, then there will be a significant degradation in simulation due to the long length of the communication path between them. The expansion of an embedding measures how much larger is the host network than the guest network during the simulation. We want to minimize expansion, as we want to use the smallest possible host network that has at least as many processors as in the guest network. In reality, we usually have a fixed size host network and we may have to consider many-to-one embedding for larger guest networks. When the size of the guest network is not equal to the size of the host network in terms of the number of processors, then we try to find the smallest host network that has at least as many processors as the guest network. Such a host network is referred to as the *optimal host network*. There is a trade off between dilation, which measures the communication delay, and expansion, which measures processor utilization, such that one can achieve lower expansion at a cost of greater dilation and vice versa.

A *hypercube* of dimension n, denoted by Q_n, is an undirected graph consisting of 2^n vertices labeled from 0 to $2^n - 1$ and such that there is an edge between any two vertices if and only if the binary representation of their labels differs in exactly one bit position. A *complete binary tree* of height $n-1$, denoted by CB_n, is an undirected graph consisting of $2^n - 1$ vertices and such that every vertex of depth less than $n-1$ has exactly two sons and every vertex of depth $n-1$ is a leaf. A *complete quad tree* of height $n-1$, denoted by CQ_n, is an undirected graph consisting of $(4^n - 1)/3$ vertices and such that every vertex of depth less than $n-1$ has exactly four sons and every vertex of depth $n-1$ is a leaf. A *cycle* of size n, denoted C_n, is an undirected graph consisting of n vertices labeled from v_1 to v_n, such that node v_i is a neighbor with node $v_{(I+1) \bmod n}$, $1 \leq i \leq n$. A *path* $(v_o, v_1, v_2, \ldots, v_{n-1})$ is a sequence of nodes such that each two consecutive nodes are adjacent. A path in a graph G is a *Hamiltonian path* if all its nodes are distinct and they span G. A cycle or a circuit is called a *Hamiltonian circuit* if it traverses every node of G exactly once.

The crossed cube is defined recursively as follows. Let G be any undirected labeled graph, then G^b is obtained from G by prefixing every vertex label with b. Two binary strings $x = x_1x_0$ and $y = y_1y_0$, each of length two, are *pair-related* if and only if $(x, y) \in \{(00, 00), (10, 10), (01, 11), (11, 01)\}$. Now, we define a *crossed*

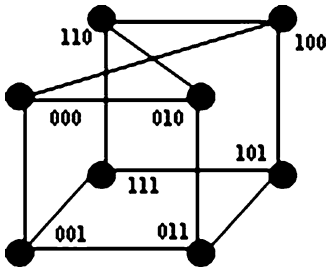

Fig. 13.1 The crossed cube XQ_3

cube of dimension n, denoted XQ_n, as an undirected graph consisting of 2^n vertices labeled from 0 to $2^n - 1$ and defined recursively as following:

1. XQ_1 is the complete graph on two vertices with labels 0 and 1.
2. For $n > 1$, XQ_n consists of two copies of XQ_{n-1} one prefixed by 0, ${XQ^0}_{n-1}$, and the other by 1, ${XQ^1}_{n-1}$. Two vertices $u = 0u_{n-2}\ldots u_0 \in {XQ^0}_{n-1}$ and $v = 1v_{n-2}\ldots v_0 \in {XQ^1}_{n-1}$ are adjacent, if and only if:
   (a) $u_{n-2} = v_{n-2}$, if n is even.
   (b) For $0 \le i \le \lfloor (n-1)/2 \rfloor$, $u_{2i+1}u_{2i}$ and $v_{2i+1}v_{2i}$ are pair-related.

Figure 13.1 shows crossed cubes of dimension 3. XQ_n is constructed recursively based on the construction of XQ_{n-1} by pasting together a copy of ${XQ^0}_{n-1}$ and the mirror image of ${XQ^1}_{n-1}$, then adding the appropriate links between the two copies according to the pair-related relationship.

13.3 Embedding Complete Binary Trees

This section describes our scheme to embed a complete binary tree CB_n into a crossed cube XQ_n with dilation two and unit expansion. Our scheme is based on the inorder labeling to embed CB_n into XQ_n in a straight forward way. The inorder embedding is constructed by Algorithm Embedding Complete Binary Tree (ECBT).

Algorithm ECBT

1. *Begin*
2. Label the nodes of the complete binary tree based on the inorder traversal using binary representation
3. Map each node of the complete binary tree to the node in the crossed cube with the corresponding binary representation
4. *End*

Theorem 13.1. *For all n, the inorder labeling of the complete binary tree embeds* CB_n *within the crossed cube* XQ_n *with dilation two.*

Prove of the theorems presented in this chapter is omitted due to the limited space. However, the reader can refer to [1] for more details. As an illustration to

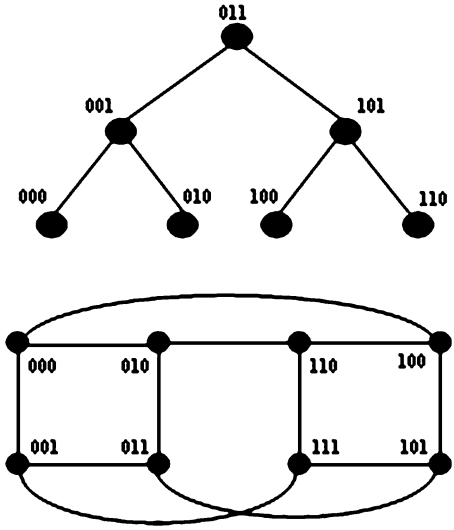

Fig. 13.2 The inorder embedding of CB_3 into XQ_3

the resulted embedding, in the lowest level, each edge from a left child to its parent is mapped to the corresponding crossed cube edge between the images of the two nodes, while the edge between a right child to its parent is mapped to a path of length two, from the right child to the left child and from the left child to the parent. In the higher level, each edge from a left child, or a right child, to its parent is mapped to the corresponding crossed cube edge between the images of the two nodes. In all higher levels, each edge from a left child, or a right child, to its parent is mapped to a path of length two. Notice that the inorder embedding is very simple and straight forward, as shown in Fig. 13.2.

13.4 Embedding Complete Quad Trees

This section describes our recursive scheme to embed a complete quad tree CQ_n into its optimal crossed cube XQ_{2n-1} with dilation two and unit expansion. We proceed in four steps. In the first step, CQ_n is decomposed into a four complete quad sub trees; a left complete quad sub tree ACQ_{n-1} with root a, a left middle complete quad sub tree BCQ_{n-1} with root b, a right middle complete quad sub tree CCQ_{n-1} with root c, a right complete quad sub tree DCQ_{n-1} with root d, and a root r. In the second step, XQ_{2n-1} is decomposed into four sub cubes ${XQ^{00}}_{2n-3}$, ${XQ^{01}}_{2n-3}$, ${XQ^{11}}_{2n-3}$, and ${XQ^{10}}_{2n-3}$. In the third step, ACQ_{n-1} is embedded into ${XQ^{00}}_{2n-3}$, BCQ_{n-1} is embedded into ${XQ^{01}}_{2n-3}$, CCQ_{n-1} is embedded into ${XQ^{11}}_{2n-3}$, DCQ_{n-1} is embedded into ${XQ^{10}}_{2n-3}$, and the root r is embedded into one of the unused nodes in ${XQ^{00}}_{2n-3}$. In the last step, we construct CQ_n by finding the paths $r \sim a$, $r \sim b$, $r \sim c$, and $r \sim d$, each of at most length two. The embedding process is continued recursively by decomposing the complete quad sub trees and

the crossed sub cubes, repeating the above steps, until we reach the leaves of the complete quad tree. At the bottom level of the complete quad tree, each complete quad sub tree with five nodes is mapped into a crossed sub cube of dimension 3. Next, we present Algorithm Embed Complete Quad Tree (ECQT) that uses a recursive divide and conquers technique to embed a complete quad tree CQ_n into its optimal crossed cube XQ_{2n-1}.

Algorithm ECQT

Let δ_i be the binary string of length n with a 1 *in position i and* 0 *in all other positions, θ_k be the binary string of length k with 0 in all positions, and $\oplus$ be the xor operator.*

1. *Begin*
2. Decompose CQ_n to ACQ_{n-1}, BCQ_{n-1}, CCQ_{n-1}, DCQ_{n-1}, and r
3. Decompose XQ_{2n-1} *to* ${XQ^{00}}_{2n-3}$, ${XQ^{01}}_{2n-3}$, ${XQ^{11}}_{2n-3}$, and ${XQ^{10}}_{2n-3}$
4. Map the quad sub trees into the crossed sub cubes as follows:
   (a) *Embed ACQ_{n-1} into ${XQ^{00}}_{2n-3}$, BCQ_{n-1} into ${XQ^{01}}_{2n-3}$, CCQ_{n-1} into ${XQ^{11}}_{2n-3}$, and DCQ_{n-1} into ${XQ^{10}}_{2n-3}$. a, b, c, and d will appear at addresses $000\theta_{2n-4}$, $010\theta_{2n-4}$, $110\theta_{2n-4}$, and $100\theta_{2n-4}$, respectively*
   (b) *Translate the embeddings in ${XQ^{00}}_{2n-3}$ and ${XQ^{10}}_{2n-3}$ by complementing the* $(2n-3)$th *bit of each node. Formally, if a tree node was mapped to address x then after the translation it will appear at address* $x \oplus \delta_{2n-3}$. *After the translation, the left root a and the right root d will appear at addresses $001\theta_{2n-3}$ and $101\theta_{2n-4}$, respectively. Therefore, the final position of a, b, c, and d are $001\theta_{2n-4}$, $010\theta_{2n-4}$, $110\theta_{2n-4}$, and $101\theta_{2n-4}$, respectively*
   (c) *Map the root r into the node with label* 0 *in ${XQ^{00}}_{2n-3}$*
5. *Construct CQ_n from ACQ_{n-1}, BCQ_{n-1}, CCQ_{n-1}, DCQ_{n-1}, CQ_n, and r by finding the four paths $r \sim a, r \sim b, r \sim c$, and $r \sim d$. The edges r-a and r-b of CQ_n are mapped to paths of length one in XQ_{2n-1}, while the edges r-c and r-d are mapped to paths of length two. The shortest paths from r to c and from r to d are $000\theta_{2n-4} - 010\theta_{2n-4} - 110\theta_{2n-4}$ and $000\theta_{2n-4} - 100\theta_{2n-4} - 101\theta_{2n-4}$, respectively*
6. *End*

Theorem 13.2. *For all n, Algorithm ECQT maps the complete quad tree* CQ_n *within the crossed cube* XQ_{2n-1} *with dilation two and unit expansion (Figs. 13.3 and 13.4).*

13.5 Embedding Cycles

Given a cycle C_{2^n} with 2^n nodes, consider the problem of assigning the cycle nodes to the nodes of the crossed cube XQ_n such that adjacency is preserved. Now, given any two adjacent nodes in the cycle, their images by this embedding should be neighbors in the crossed cube through some dimension i, where $1 \leq i \leq n$. We can view such an embedding as a sequence of dimensions crossed

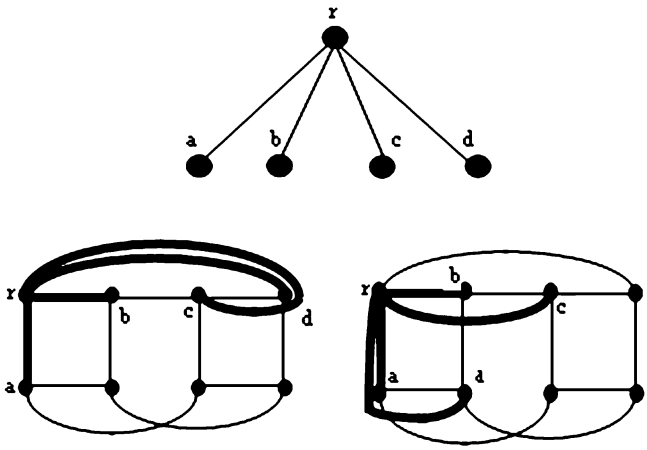

Fig. 13.3 Embedding CQ_1 into the sub cube XQ_3

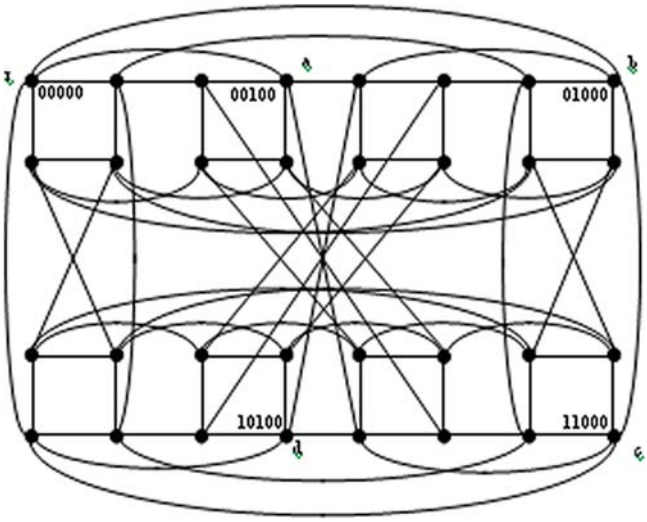

Fig. 13.4 Embedding CQ_3 into XQ_5

by adjacent nodes. We call such a sequence the *embedding sequence*, denoted by $ES = (d_1, d_2, \ldots, d_{2^n})$, where $d_i \in \{1, \ldots, n\}$ for all $1 \leq i \leq 2^n$. Figure 13.5 shows two different embeddings of the cycle C_{2^3} into the crossed cube XQ_3. It is more convenient to view the embedded cycle as will as the crossed cube in the way shown in Fig. 13.5. The embedding sequence of C_{2^3} is $ES = (1, 3, 1, 2, 1, 3, 1, 2)$. For example, in the first part of Fig. 13.5, notice that nodes 000 and 001 are connected by a link through dimension 1, 001 and 111 are connected by a link through

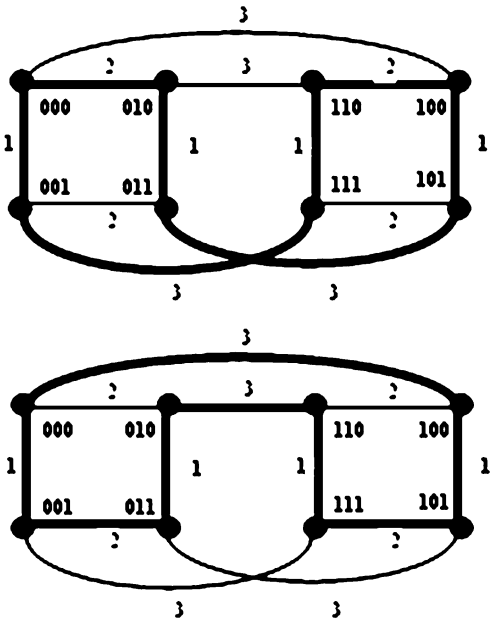

Fig. 13.5 The embedding sequence

dimension 3, 111 and 110 are connected by a link through dimension 1, 110 and 100 are connected by a link through dimension 2, and so on. The embedding sequence ES can be generated using Algorithm ES.

Algorithm ES

Let n be the dimension of the crossed cube and let the vertical bar be the concatenation operator.

1. *Begin*
2. $ES \leftarrow 1$
3. *For* $i \leftarrow 3$ *to n do*
4. $ES \leftarrow ES|i|ES$
5. $ES \leftarrow ES|2|ES|2$
6. *End*

The embedding sequence is generated by applying Algorithm ES on n, where n is the dimension of the crossed cube. The number of nodes in the crossed cube is equal to the number of nodes in the embedded cycle, which is 2^n nodes. Thus, the embedding sequence of the cycle C_{2^4} is ES $= (1, 3, 1, 4, 1, 3, 1, 2, 1, 3, 1, 4, 1, 3, 1, 2)$ and the embedding sequence of the cycle C_{2^5} is ES $= (1, 3, 1, 4, 1, 3, 1, 5, 1, 3, 1, 4, 1, 3, 1, 2, 1, 3, 1, 4, 1, 3, 1, 5, 1, 3, 1, 4, 1, 3, 1, 2)$. The embedding sequence corresponds to the extended binary-reflected Gray code embedding of a cycle into a crossed cube. The binary-reflected Gray code is the most common technique to embed a cycle into a fault-free hypercube. Notice that the same embedding sequence may result in different embeddings of C_2n into XQ_n depending on the crossed cube

node that initiates the cycle construction. Among all different embeddings, we are interested in one kind. The embedding when the node that initiates the cycle construction in the crossed cube is the upper leftmost node, node with label 0. This will not violate the generalization of the technique since the crossed cube is node and vertex symmetric [9], which means that we can relabel the nodes, where any node can be labeled as node 0, and hence initiates the construction of the cycle. In Fig. 13.5, the cycle is initiated by node 000 in the first part, while it is initiated by node 001 in the second part.

Theorem 13.3. *For every n, Algorithm ES will generate the embedding sequence to construct a cycle of size* 2^n *in a crossed cube of dimension n.*

Next, we present Algorithm Hamiltonian Circuit (HC) that uses a recursive divide and conquers technique to embed a cycle C_{2^n} into a crossed cube XQ_n.

Algorithm HC

1. *Begin*
2. *Partition* XQ_n *into* 2^{n-3} *disjoint crossed cubes, each of dimension 3*
3. *Embed the cycle* C_{2^3} *into each sub cube using the embedding sequence* $ES = (1, 3, 1, 2, 1, 3, 1, 2)$
4. *Connect the* 2^{n-3} *cycles, each of size 8, through the upper, or lower, links to come up with a cycle of size* C_{2^n}
5. *End*

Theorem 13.4. *For every n, Algorithm HC will embed a Hamiltonian cycle of size* 2^n *in a crossed cube of dimension n.*

Note the use of the upper links of dimension n when the embedding sequence is generated by node 0, while the lower crossed links of dimension n are used when the embedding sequence is generated by node 1, as shown in Fig. 13.6.

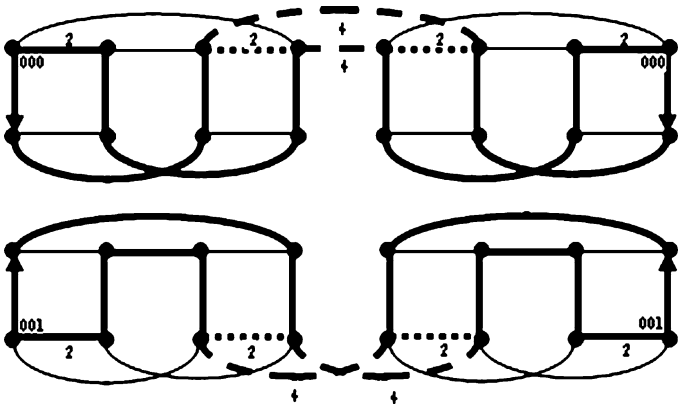

Fig. 13.6 The recursive construction of the cycle C_{2^n} in a fault-free environment

13.6 Conclusions and Future Work

This chapter has presented different schemes to show the ability of the crossed cube as a versatile architecture to simulate other interconnection networks efficiently. We present new schemes to embed complete binary trees, complete quad trees, and cycles into crossed cubes. A good problem will be to improve the dilation on embedding trees into crossed cubes. Another interesting problem is to generalize the schemes to embed trees and cycles in the presence of faults.

References

1. Abuelrub, E. Embeddings into crossed cubes. Proceedings of the World Congress on Engineering (WCE'2009), **1**, London (2009)
2. Abuelrub, E. The hamiltonicity of crossed cubes in the presence of faults. Eng. Lett. **16**(3), 453–459 (2008)
3. Barasch, L., Lakshmivarahan, S., Dhall, S.: Embedding arbitrary meshes and complete binary trees in generalized hypercubes. Proceedings of the 1st IEEE Symposium on Parallel and Distributed Processing, pp. 202–209, 1989
4. Chang, C., Sung, T., Hsu, L.: Edge congestion and topological properties of crossed cubes. IEEE Trans. Parall. Distrib. Syst. **11**(1), 64–80 (2006)
5. Dingle, A., Sudborough, I.: Simulating binary trees and x-trees on pyramid networks. Proceedings of the 1st IEEE Symposium on Parallel and Distributed Processing, pp. 210–219 (1989)
6. Efe, K.: The crossed cube architecture for parallel computation. IEEE Trans. Parall. Distrib. Syst. **3**(5), 513–524 (1992)
7. El-Amaway, A., Latifi, S.: Properties and performance of folded hypercubes. IEEE Trans. Parall. Distrib. Syst. **2**(1), 31–42 (1991)
8. Fan, J., Lin, X., Jia, X.: Optimal path embedding in crossed cubes. IEEE Trans. Parall. Distrib. Syst. **16**(12), 1190–1200 (2005)
9. Fu, J., Chen, G.: Hamiltonicity of the hierarchical cubic network. Theory Comput. Syst. **35**, 59–79 (2008)
10. Keh, H., Lin, J.: On fault-tolerance embedding of hamiltonian cycles, linear arrays, and rings in a flexible hypercube. Parall. Comput. **26**(6), 769–781 (2000)
11. Latifi, S., Zheng, S.: Optimal simulation of linear array and ring architectures on multiply-twisted hypercube. In: Proceedings of the 11th International IEEE Conference on Computers and Communications, 1992
12. Lee, S., Ho, H.: A 1.5 approximation algorithm for embedding hyperedges in a cycle. IEEE Trans. Parall. Distrib. Syst. **16**(6), 481–487 (June 2005)
13. Leighton, T.: Introduction to Parallel Algorithms and Architecture: Arrays, Trees, Hypercubes. Morgan Kaufmann, San Mateo, CA (1992)
14. Lin, J.: Embedding hamiltonian cycles, linear arrays, and rings in a faulty supercube. Int. J. High Speed Comput. **11**(3), 189–201 (2000)
15. Ma, M., Xu, J.: Panconnectivity of locally twisted cubes. Appl. Math. Lett. **17**(7), 674–677 (2006)
16. Markas, T., Reif, J.: Quad tree structures for image compression applications. Inform. Process. Lett. **28**(6), 707–722 (1992)
17. Preparata, F., Vuillemin, J.: The cube-connected cycles: a versatile network for parallel computation. Commun. ACM. **24**(5), 3000–3309 (1981)
18. Topi, L., Parisi, R., Uncini, A.: Spline recurrent neural network for quad-tree video coding. Proceedings of WIRN'2002, pp. 90–98 (2002)

19. Saad, Y., Schultz, M.: Topological properties of the hypercube. IEEE Trans. Comput. **37**(7), 867–872 (July 1988)
20. Youyao, L.: A hypercube-based scalable interconnection network for massively parallel computing. J. Comput. **3**(10) (October 2008)