

1994


# A Greedy Hypercube-Labeling Algorithm

D. Bhagavathi  
*Old Dominion University*

C. E. Grosch  
*Old Dominion University*

S. Olariu  
*Old Dominion University*

Follow this and additional works at: [https://digitalcommons.odu.edu/computerscience\\_fac\\_pubs](https://digitalcommons.odu.edu/computerscience_fac_pubs)

 Part of the [Databases and Information Systems Commons](#), [Software Engineering Commons](#), and the [Theory and Algorithms Commons](#)

---

## Repository Citation

Bhagavathi, D.; Grosch, C. E.; and Olariu, S., "A Greedy Hypercube-Labeling Algorithm" (1994). *Computer Science Faculty Publications*. 111.  
[https://digitalcommons.odu.edu/computerscience\\_fac\\_pubs/111](https://digitalcommons.odu.edu/computerscience_fac_pubs/111)

## Original Publication Citation

Bhagavathi, D., Grosch, C. E., & Olariu, S. (1994). A greedy hypercube-labeling algorithm. *Computer Journal*, 37(2), 124-128.  
doi:10.1093/comjnl/37.2.124

---

# A Greedy Hypercube-Labeling Algorithm

D. BHAGAVATHI, C. E. GROSCH AND S. OLARIU

*Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162, USA*

---

Due to its attractive topological properties, the hypercube multiprocessor has emerged as one of the architectures of choice when it comes to implementing a large number of computational problems. In many such applications, Gray-code labelings of the hypercube are a crucial prerequisite for obtaining efficient algorithms. We propose a greedy algorithm that, given an  $n$ -dimensional hypercube  $H$  with  $N = 2^n$  nodes, returns a Gray-code labeling of  $H$ , that is, a labeling of the nodes with binary strings of length  $n$  such that two nodes are neighbors in the hypercube if, and only if, their labels differ in exactly one bit. Our algorithm is conceptually very simple and runs in  $O(N \log N)$  time being, therefore, optimal. As it turns out, with a few modifications our labeling algorithm can be used to recognize hypercubes as well.

*Received October 26, 1993; accepted December 6, 1993*

---

## 1. INTRODUCTION

Advances in VLSI in the last decade have made it possible to build massively parallel machines featuring tens of thousands of processing elements. Communication among processing elements in a multiprocessor machine is typically achieved by some message passing protocol and the number of edges traversed is often a reliable measure of the delay incurred. It has been recognized (Hwang and Briggs, 1984; Bertsekas and Tsitsiklis, 1989) that in a multiprocessor system the time spent for interprocessor communications is often a significant portion of the overall time needed to solve a given computational problem.

Some of the key parameters of the interconnection topology of the multiprocessor system that affect the communication delay are the diameter, the degree, the symmetry and the connectivity of the underlying structure. It soon became apparent that the above parameters cannot be all optimized at the same time (Stone, 1971; Sietz, 1984). Attempts to address one or the other of these parameters have resulted in an array of different topologies that have appeared in the literature (Seitz, 1984; Brown *et al.*, 1985; Omondi and Brock, 1987; Abram and Padmanabhan, 1989; Sur and Srimani, 1992). Each of these has a number of virtues and failings and is, as a rule, suitable for a limited domain of practical applications.

Among the commercially available multiprocessor machines, the hypercube (also known as  $n$ -cube, Cosmic cube, Boolean cube) stands out as one of the most attractive and versatile, with a vast range of applications (Preparata and Vuillemin, 1981; Quinn, 1984; Bertsekas and Tsitsiklis, 1989). The hypercube has a highly regular graph structure that results in very powerful interconnection features and in the capability to embed many of the classical architectures (Saad and Schule, 1985; Bertsekas and Tsitsiklis, 1989; Stone, 1990).

Not surprisingly, hypercubes and hypercube-like interconnection networks have received a lot of attention

over the past years because they offer a large bandwidth, logarithmic diameter, and a high degree of fault tolerance (Abrams and Padmanabhan, 1989; Bertsekas and Tsitsiklis, 1989; Sur and Srimani, 1992).

Topological properties of hypercubes have been extensively studied (Bhat, 1980; Quin, 1984; Bertsekas and Tsitsiklis, 1989). Consider an  $n$ -dimensional hypercube  $H$  with  $N = 2^n$  nodes. The following properties are both well-known and easy to prove from scratch:

- (h1)  $H$  is a connected bipartite graph;
- (h2)  $H$  has precisely  $n2^{n-1}$  edges;
- (h3) its diameter is  $n$ .

In a hypercube the processors are located at the nodes and the interconnections between processors are defined by the edges. It is worth noting that, as a consequence, the distance between any two processors is at the most  $n$ . Moreover, both the diameter and the degree of nodes in the hypercube increase very slowly with respect to the number of nodes in the graph (i.e. logarithmically in the number of nodes). In addition, the  $n$ -dimensional hypercube has the remarkable property (h4) that we state next:

- (h4) its nodes can be labeled by integers in the range 0 to  $2^n - 1$  such that two nodes are adjacent in the hypercube if, and only if, their labels differ in exactly one bit.

The Hamming distance (Bertsekas and Tsitsiklis, 1989) between two processors in the hypercube is the number of bits in which their labels differ. Therefore, the hypercubes have the property that two processors are connected by a direct link if and only if their Hamming distance is 1. A Gray-code labeling of the vertices of a given hypercube is a labeling for which the Hamming distance between two vertices is 1 if and only if the vertices are adjacent in the hypercube (see Figure 1). Gray-code labelings are a crucial prerequisite in many applications. For example, for the purpose of embedding several architectures onto a hypercube it is

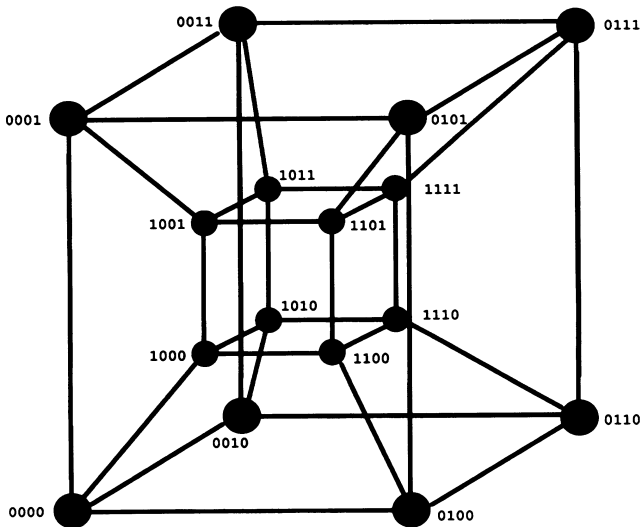


FIGURE 1. A labeled four-dimensional hypercube.

necessary to obtain such a labeling of the nodes of a hypercube (Stone, 1990).

Recently, the availability of powerful and flexible computing devices (e.g. transputers) has made the incremental construction of various interconnection topologies a reality. Not surprisingly, the problem of building hypercubes incrementally from small building blocks has received a good deal of attention in the literature (Jane et al., 1992; Sur and Srimani, 1992). Once such a construction is complete, it is necessary to inform every processor of the newly created hypercube about its identity *within* the hypercube. Since virtually all known hypercube algorithms assume that every processor knows its own coordinates in the machine, this initialization must be done at power-up time.

The purpose of this note is to report a very simple greedy algorithm that returns a Gray-code labeling of the nodes of a hypercube, thus solving the initialization problems mentioned above. Specifically, our algorithm uses depth-first search to guide the greedy choice of labels based on local conditions only. What results is a labeling algorithm that, with an  $n$ -dimensional hypercube with  $N=2^n$  nodes as input, runs in  $O(N \log N)$  time. In view of the property (h2) mentioned above, this algorithm is optimal.

Along similar lines, the problem of recognizing graphs that are hypercubes turns out to be of considerable import. Bhat (1980) has proposed such an algorithm: with a graph  $G$  with  $N=2^n$  nodes as input his algorithm determines whether the graph is a hypercube in  $O(N \log N)$  time. As it turns out, with a few modifications our hypercube-labeling algorithm can be used to recognize hypercubes as well. The details of our recognition algorithms will be presented in an upcoming paper.

The remainder of this paper is organized as follows: Section 2 presents basic technical results that are at the heart of our algorithm; Section 3 discussed the details

of the proposed labeling algorithm; finally, Section 4 summarizes the results.

## 2. BASICS

We assume standard terminology compatible with [1]. The shortest distance  $d_G(x, y)$  between vertices  $x$  and  $y$  in a graph  $G$  represents the smallest number of edges that have to be traversed to reach  $y$  from  $x$ . Consider an  $n$ -dimensional hypercube  $H$  with  $N=2^n$  nodes, and let  $u$  be an arbitrary node of  $H$ .

Partition the nodes of  $H$  into sets  $N_i(u)$  ( $0 \leq i \leq n$ ) defined as follows:

$$N_i(u) = \{v | d_H(u, v) = i\}. \tag{1}$$

An easy inductive argument confirms that any labeling of the vertices of  $H$  that satisfies (h4) must be such that

$$N_i(u) = \{v | \text{label}(u) \text{ and label}(v) \text{ differ in exactly } i \text{ bits}\}. \tag{2}$$

As a consequence of (2), no vertices in  $N_i(u)$  are adjacent. Furthermore, since  $N_i(u) \cap N_j(u) = \emptyset$  whenever  $i \neq j$ , every node of the hypercube must belong to precisely one of the sets  $N_i(u)$ . For further reference, we also take note of the following technical result.

LEMMA 1 Let  $v, w$  be distinct vertices in  $N_i(u)$ . The following statements must be satisfied:

- (1.1)  $v, w$  have at most one common neighbor in  $N_{i+1}(u)$ ;
- (1.2)  $v, w$  have a common neighbor in  $N_{i+1}(u)$  if, and only if, they have a common neighbor in  $N_{i-1}(u)$ .

*Proof* To settle (1.1) let  $z$  be a common neighbor of  $v$  and  $w$  in  $N_{i+1}(u)$ . Since  $v$  and  $z$  are adjacent, they disagree in exactly one bit. Let this be the  $k$ th bit. Note that since  $z \in N_{i+1}(u)$ ,  $z$  disagrees with  $u$  in all the bits on which  $v$  and  $u$  disagree, plus the  $k$ th bit. As a consequence,  $u$  and  $v$  agree on the  $k$ th bit.

Similarly,  $w$  and  $z$  being adjacent, they must disagree in exactly one bit, say the  $l$ th bit. Since  $z \in N_{i+1}(u)$ ,  $z$  and  $u$  disagree in all bits on which  $w$  and  $u$  disagree, plus the  $l$ th bit. Again,  $w$  and  $u$  must agree on the  $l$ th bit. It follows that  $v$  and  $w$  must agree on all bits, with the exception of the  $k$ th and  $l$ th bit.

Now assume that  $v$  and  $w$  have a common neighbor  $z'$  in  $N_{i+1}(u)$ . Repeating the above argument, we find distinct subscripts  $p$  and  $q$  such that  $v$  and  $w$  differ in the  $p$ th and  $q$ th bit. Therefore, it must be the case that  $p=k$  and  $q=l$  or vice versa, and so  $z'$  coincides with  $z$ . This completes the proof of (1.1).

To prove (1.2), we first assume that  $v$  and  $w$  have a common neighbor  $z$  in  $N_{i+1}(u)$ . The proof of (1.1) guarantees that we can write

$$v = a_1 a_2 \dots a_{k-1} a_k a_{k+1} \dots a_{l-1} \bar{a}_l a_{l+1} \dots a_n$$

and

$$w = a_1 a_2 \dots a_{k-1} \bar{a}_k a_{k+1} \dots a_{l-1} \bar{a}_l a_{l+1} \dots a_n$$

such that  $u$  and  $v$  agree on  $a_k$  and disagree on  $a_l$  and such that  $u$  and  $w$  disagree on  $\bar{a}_k$  and agree on  $\bar{a}_l$ .

But now, the node

$$z' = a_1 a_2 \dots a_{k-1} a_k a_{k+1} \dots a_{l-1} \bar{a}_l a_{l+1} \dots a_n$$

is adjacent to both  $v$  and  $w$  (differing from both in one bit); furthermore,  $z'$  disagrees with  $u$  in  $i-1$  bits and so  $z'$  belongs to  $N_{i-1}(u)$ .

Conversely, let  $v$  and  $w$  have a common neighbor  $z'$  in  $N_{i-1}(u)$ . Write

$$z' = b_1 b_2 \dots b_n$$

From  $z'$  we obtain both  $v$  and  $w$  by permuting bits  $b_p$  and  $b_q$ , ( $1 \leq p \neq q \leq n$ ), and so

$$v = b_1 b_2 \dots b_{p-1} \bar{b}_p b_{p+1} \dots b_{q-1} b_q b_{q+1} \dots b_n$$

and

$$w = b_1 b_2 \dots b_{p-1} b_p b_{p+1} \dots b_{q-1} \bar{b}_q b_{q+1} \dots b_n$$

Note that since  $v$  and  $w$  are in  $N_i(u)$ , it must be that  $u$  and  $z'$  agree on both  $b_p$  and  $b_q$ . But now, the node

$$z = b_1 b_2 \dots b_{p-1} \bar{b}_p b_{p+1} \dots b_{q-1} \bar{b}_q b_{q+1} \dots b_n$$

is adjacent to both  $v$  and  $w$  and differs from  $u$  in exactly  $i+1$  bits, confirming that  $z \in N_{i+1}(u)$ . This completes the proof of Lemma 1.  $\square$

### 3. THE ALGORITHM

We assume a hypercube  $H=(V, E)$  with  $N=2^n$  nodes specified by its adjacency lists. That is, for every node in the hypercube we assume a linked list containing all the nodes adjacent to it. As a preprocessing step, we select a distinguished node  $u$  in  $H$  and perform a breadth-first search (Aho *et al.*, 1974) with  $u$  as anchor; when this step is done, every node  $v$  in  $H$  knows its shortest distance  $d_H(u, v)$  from  $u$ . For every node  $v$  define

$$\text{level}(v) = i \text{ iff } d_H(u, v) = i. \quad (3)$$

Note that once this information is available, the partition defined in (1) is also, implicitly, known. The basic idea of our algorithm is to proceed from this partition by assigning labels in a way consistent with (2).

We let the assignment of labels be performed greedily, based on local information only. To guide the assignment of labels we traverse the hypercube in a way reminiscent of depth-first search (Aho *et al.*, 1974). For each node  $v$  of  $H$  we maintain the following variables:  $\text{mask}(v)$ ,  $\text{label}(v)$ , and  $\text{level}(v)$ . Initially, all these variables are initialized to  $00 \dots 0$ , and the nodes are labeled 'new' (i.e. unprocessed). As the algorithm progresses, more and more nodes become 'old' (i.e. processed). The details are spelled out by procedure Hypercube that we present next.

**Procedure** Hypercube( $H$ );

{Input: an  $n$ -dimensional hypercube;

Output: a labeling of the nodes of  $H$  such that two nodes are adjacent if and only if their labels differ in one bit;}

0. **begin**

1. choose an arbitrary node  $u$  in  $H$ ;
2. perform a breadth-first search of  $H$

starting at  $u$  and record for every node  $v$ ,  $\text{level}(v)$  as in (3);

3. **for** all nodes  $v$  of  $H$  **do**
4.      $\text{mask}(v) \leftarrow \text{label}(v) \leftarrow 00 \dots 0$ ;
5.     mark  $v$  "new";
6.     mark  $u$  "old";
7.     Label\_Node( $v$ )
8. **end**; {Hypercube}

The following procedure invoked from within Hypercube processes the nodes in depth-first fashion. Initially, all nodes are marked 'new'; during the execution of the algorithm the nodes become 'old' and they receive, at that moment, their permanent label. To help with the assignment of labels, every node  $v$  that has just been marked 'old' scans its adjacency list, node by node, performing the following:

- For every node  $w$  adjacent to  $v$  marked 'new' with  $\text{level}(w) > \text{level}(v)$ , the procedure Label\_Node is invoked recursively with  $w$  as a parameter, resulting in a labeling of  $w$ ;
- For every node  $z$  adjacent to  $v$  marked 'new' with  $\text{level}(z) < \text{level}(v)$  and such that  $\text{label}(z) = 00 \dots 0$   $v$  writes its own label into  $\text{label}(z)$ . This will be used later to help assign  $z$  its permanent label. Specifically, consider the moment when Label\_Node is invoked with  $z$  as a parameter. Assume that the closest pending recursive call of Label\_Node involved some vertex  $x$  as a parameter (put differently,  $z$  is a neighbor of  $x$  with  $z$  'new' and  $\text{level}(z) > \text{level}(x)$ ). Note that in any labeling of the nodes compatible with (h4),  $x$  and  $z$  differ in one bit; this bit can be determined easily knowing the label of  $x$  and the label of some neighbor of  $z$ , conveniently stored, temporarily, in  $\text{label}(z)$ . The details are spelled out as follows.

**Procedure** Label\_Node( $v$ );

0. **begin**

1.     **for** all 'new' neighbors  $w$  of  $v$  **do**
2.         **if**  $\text{level}(w) > \text{level}(v)$  **then**
3.             **if**  $\text{label}(w) \neq 00 \dots 0$  **then**
4.                  $\text{temp} \leftarrow \text{label}(w)$ ;
5.                  $\text{label}(w) \leftarrow [\text{temp AND } \text{mask}(v)] \text{ OR } \text{label}(v)$ ;
6.                  $\text{mask}(w) \leftarrow \text{mask}(v) \leftarrow [\text{temp XOR } \text{label}(v)]$   
                    OR  $\text{mask}(v)$ ;
7.                 mark  $w$  'old';
8.             **else** {now  $\text{label}(w) = 00 \dots 0$ }
9.                 update  $\text{mask}(v)$  by toggling an arbitrary 0;
10.                  $\text{label}(w) \leftarrow \text{mask}(w) \leftarrow \text{mask}(v)$ ;
11.                 Label\_Node( $w$ )
12.             **else** {now  $\text{level}(w) < \text{level}(v)$ }
13.                 **if**  $\text{label}(w) = 00 \dots 0$  **then**
14.                      $\text{label}(w) \leftarrow \text{label}(v)$   
                           {record  $v$ 's label}

15. **end**; {Label\_Node}

We are now in a position to state the following result that proves the correctness of our algorithm and argue for its running time.

**THEOREM 1.** With an  $n$ -dimensional hypercube  $H$  with

$N = 2^n$  nodes as input, procedure Hypercube correctly produces a labeling of the nodes of  $H$  such that two nodes are adjacent if and only if their labels differ in one bit; furthermore, the running time of this procedure is bounded by  $O(N \log N)$ .

*Proof* To argue for the correctness of our algorithm we note that when line 1 of Label\_Node is encountered, either:

- for all 'new' nodes  $w$  adjacent to  $v$ , with  $\text{level}(w) > \text{level}(v)$ ,  $\text{label}(w) = 00 \dots 0$ ; or else
- for all 'new' nodes  $w$  adjacent to  $v$ , with  $\text{level}(w) < \text{level}(v)$ ,  $\text{label}(w) \neq 00 \dots 0$ .

Otherwise, choose a counterexample  $v$  with  $\text{level}(v)$  as small as possible. (We note that, in fact,  $\text{level}(v) = i$  amounts to saying that  $v \in N_i(u)$ .) We shall distinguish between the following two cases.

*Case 1*  $v$  has a neighbor  $v'$  marked 'old' with  $\text{level}(v') > \text{level}(v)$ .

Let  $w$  be any neighbor of  $v$  marked 'new', with  $\text{level}(w) > \text{level}(v)$ . It is easy to confirm that  $\text{level}(w) = \text{level}(v')$ . Note that (1.2) guarantees that  $v'$  and  $w$  have a common neighbor  $z$  with  $\text{level}(z) > \text{level}(v') = \text{level}(w)$ .

However, the recursive call Label\_Node( $v'$ ) during which  $v'$  was marked 'old' (and also labeled) could not end until  $z$  was also marked 'old'. Consequently, lines 12–14 in the procedure guarantee that  $\text{label}(w)$  cannot be  $00 \dots 0$ .

*Case 2* no neighbor  $v'$  of  $v$  with  $\text{level}(v') > \text{level}(v)$  is 'old'.

Consequently,  $v$  is still 'new' and  $\text{label}(v) = 00 \dots 0$ . Let  $w$  be a neighbor of  $v$  marked 'new' with  $\text{level}(w) > \text{level}(v)$  and such that  $\text{label}(w) \neq 00 \dots 0$ . Now lines 12–14 in the procedure guarantee the existence of a node  $z'$  marked 'old' with  $w$  and  $z'$  adjacent and such that  $\text{level}(z') > \text{level}(w)$ . Let  $z$  be the node adjacent to  $z'$  that called Label\_Node with  $v'$  as a parameter. Trivially,  $\text{level}(w) = \text{level}(v)$ .

Since  $w$  and  $z$  have a common neighbor (i.e.  $z'$ ) (1.2) guarantees that they have a neighbor  $y$  with  $\text{level}(y) = \text{level}(w) - 1 = \text{level}(v)$ . Finally, note that  $v$  and  $y$  have a common neighbor (namely  $w$ ), they must also have a common neighbor  $x$  with  $\text{level}(x) = \text{level}(v) - 1$ . But now,  $x$  has two neighbors,  $v$  and  $y$ , both 'new' and such that  $\text{label}(v) = 00 \dots 0$ , and  $\text{label}(y) \neq 00 \dots 0$ , contradicting our choice of  $v$ .

Next, an easy inductive argument shows that for every 'old' node  $v$ , the number of 0's in  $\text{mask}(v)$  denotes the number of 'new' nodes  $w$  adjacent to  $v$ , with  $\text{level}(w) > \text{level}(v)$ .

Furthermore, note that  $\text{temp}$  in line 4 and  $\overline{\text{mask}}(v)$  differ in exactly one bit and therefore,  $\text{label}(w)$  contains precisely one more 1 bit than  $\text{label}(v)$ . It follows easily that  $\text{label}(w) = i + 1$  if and only if  $\text{level}(w) = i + 1$ , and that all nodes of the hypercube receive distinct labels. This settles the correctness.

To argue for the complexity, note that every node of

the hypercube will eventually receive a permanent label: this happens when the node is labeled 'old'. Lines 4–6 in procedure Label\_Node guarantee that computing the permanent label of a node in the hypercube takes  $O(\log N)$  time and is done once only. Consequently, the overall time needed to label all the nodes in the hypercube is  $O(N \log N)$ .

Similarly, the time needed to perform the breadth-first and depth-first searches (Aho *et al.*, 1974) is also  $O(N \log N)$ . The conclusion follows.  $\square$

#### 4. CONCLUSION

The hypercube has emerged as one of the most versatile topologies in use today; this is due, in part, to its regular interconnections and to the fact that many other architectures can be embedded efficiently onto the hypercube. A basic prerequisite to efficient hypercube algorithms is a certain labeling of the vertices of the hypercube, such that two vertices (processors) are adjacent if and only if their binary representations differ in exactly one bit.

The motivation for the work presented in this paper stems from the fact that the availability of powerful and flexible computing devices (e.g. transputers) makes it possible to incrementally construct new interconnection networks from simple building blocks. In particular, the problem of building hypercubes incrementally from small transputers, for example, has received a good deal of attention in the literature (Jane *et al.*, 1992; Sur and Srimani, 1992). Once such a construction is complete, it is necessary to inform every processor of the newly created hypercube about its identity *within* the hypercube. Ideally, this initialization operation should be done in parallel.

In this paper we have presented a solution to the initialization problem mentioned above. Specifically, given an  $n$ -dimensional hypercube with  $N = 2^n$  nodes our algorithm returns a Gray-code labeling of the hypercube, that is, a labeling of its nodes with binary strings of length  $n$  such that two nodes are neighbors in the hypercube if, and only if, their labels differ in exactly one bit. Our algorithm is conceptually very simple and runs in  $O(N \log N)$  time being, therefore, optimal.

The problem of recognizing graphs that are hypercubes turns out to be of considerable import. Bhat [4] has proposed such an algorithm: with a graph  $G$  with  $N = 2^n$  nodes as input his algorithm determines whether the graph is a hypercube in  $O(N \log N)$  time. With a few modification our hypercube-labeling algorithm can be used to recognize hypercubes as well. The details of our recognition algorithm will be presented in an upcoming paper.

#### ACKNOWLEDGEMENTS

The authors would like to thank an anonymous reviewer for his/her constructive comments. The third author was supported, in part, by the National Science Foundation under grant CCR-8909996.

## REFERENCES

- Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974) *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.
- Abram, S. and Padmanabhan, K. (1989) An analysis of the twisted cube topology. *Proc. 1989 Int. Conf. on Parallel Processing*, I, 116–120.
- Bertsekas, D. P. and Tsitsiklis, J. N. (1989) *Parallel and distributed computation*. Prentice-Hall, Englewood Cliffs, NJ.
- Bhat, K. V. S. (1980) On the complexity of testing a graph for  $n$ -cube. *Inf. Process. Lett.*, **11**, 16–20.
- Brown, C. M., Ellis, C. S., Feldman, J. A., LeBlanc, T. J. and Peterson, G. L. (1985) Research with the butterfly multicomputer. In *Computer Science and Computer Engineering Research Review*. University of Rochester, Rochester, NY.
- Hwang, K. and Briggs, F. A. (1984) *Computer architecture and Parallel Processing*. McGraw-Hill, New York.
- Jane, M., Fawcett, R. and Mawly, T. (1992) *Transputer Applications—Progress and Prospects*. IOS Press, Amsterdam.
- Omondi, A. R. and Brock, J. D. (1987) Implementing a dictionary on a hypercube. *Proc. 1987 Int. Conf. on Parallel Processing*, 707–709.
- Preparata, F. P. and Vuillemin, J. (1981) The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, **24**, 300–309.
- Quinn, M. J., *Designing efficient algorithms for parallel computers*. McGraw-Hill, New York.
- Saad, Y. and Schultz, M. H. (1988) Topological properties of hypercubes. *IEEE Trans. Comput.*, **37**, 867–872.
- Scott, D. S. and Brandenburg, J. (1988) Minimal mesh embeddings in binary hypercubes. *IEEE Trans. Comput.*, **37**, 1284–1288.
- Sietz, C. L. (1985) The cosmic cube. *Commun. ACM*, **28**, 22–33.
- Sietz, C. L. (1984) Concurrent VLSI architectures. *IEEE Trans. Comput.*, **C-33**, 1247–1265.
- Stone, H. S. (1990) *High-Performance Computer Architecture*, 2nd edn. Addison-Wesley, Reading, MA.
- Sur, S. and Srimani, P. K. (1992) Incrementally extendible hypercubes. *Proc. Int. Phoenix Conf. on Computers and Communications*, 1–7.
- Weber, H. C. (ed.) (1992) *Image Processing and Transputers*. IOS Press, Amsterdam.
- Wu, A. Y. (1985) Embeddings of tree networks into hypercubes. *J. Parallel Distr. Comput.*, **2**, 238–249.
- Wu, C.-L. and Feng, T.-Y. (eds) (1984) *Interconnection Networks for Parallel and Distributed Processing*. IEEE Computer Press, New York.