3.1 The Hypercube

In this section, we define the hypercube and explain why it is such a powerful network for parallel computation. Among other things, we will show how the hypercube can be used to simulate all of the networks discussed in Chapters 1 and 2. In fact, we will find that the hypercube contains (or nearly contains) all of these networks as subgraphs. This material is both surprising and important because it demonstrates how all of the parallel algorithms discussed thus far can be directly implemented on the hypercube without significantly affecting the number of processors or the running time. Hence, we will quickly understand one of the main reasons why the hypercube is so powerful.

We begin the section with some definitions and a brief discussion of the hypercube's simplest properties in Subsection 3.1.1. In Subsection 3.1.2, we show that the hypercube is Hamiltonian, and we explain the correspondence between Hamiltonian cycles in the hypercube and Gray codes. We also prove that any N-node array (of any dimensionality) is a subgraph of the N-node hypercube (assuming that N is a power of 2).

In Subsection 3.1.3, we describe several embeddings of an $(N-1)$-node complete binary tree in an N-node hypercube. Although the $(N-1)$-node complete binary tree is not a subgraph of the N-node hypercube, we will find that a complete binary tree can be simulated very efficiently on the hypercube.

More generally, we will show that the N-node hypercube can efficiently simulate any binary tree in Subsection 3.1.4. In particular, we will show how to grow an arbitrary M-node binary tree in an on-line fashion in an N-node hypercube so that neighboring nodes in the tree are nearby in the hypercube and so that at most $O(M/N+1)$ tree nodes are mapped to each hypercube node with high probability. The analysis of the tree-growing algorithm involves an interesting relationship between one-error-correcting codes and hypercubes that has numerous applications. We also define the hypercube of cliques in Subsection 3.1.4 and prove that it is computationally equivalent to the hypercube.

In Subsection 3.1.5, we show how to efficiently simulate a mesh of trees on the hypercube. As a consequence, we will find that all of the algorithms described in Chapter 2 can be implemented without significant slowdown on a hypercube of approximately the same size.

We conclude in Subsection 3.1.6 with a brief survey of some related

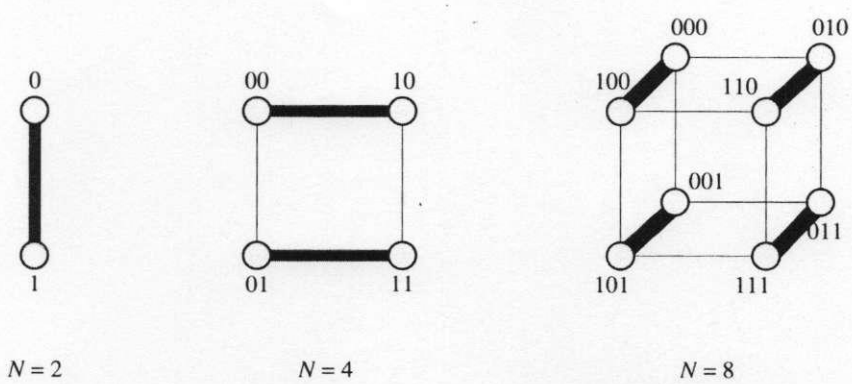

Figure 3-1 *The N-node hypercube for $N = 2$, 4, and 8. Two nodes are linked with an edge if and only if their strings differ in precisely one bit position. Dimension 1 edges are shown in boldface.*

network containment and simulation results for the hypercube.

3.1.1 Definitions and Properties

The *r-dimensional hypercube* has $N = 2^r$ nodes and $r2^{r-1}$ edges. Each node corresponds to an r-bit binary string, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit. As a consequence, each node is incident to $r = \log N$ other nodes, one for each bit position. For example, we have drawn the hypercubes with 2, 4, and 8 nodes in Figure 3-1.

The edges of the hypercube can be naturally partitioned according to the dimensions that they traverse. In particular, an edge is called a *dimension k edge* if it links two nodes that differ in the kth bit position. We will use the notation u^k to denote the neighbor of node u across dimension k in the hypercube. In particular, given any binary string $u = u_1 \cdots u_{\log N}$, the string u^k is the same as u except that the kth bit is complemented. More generally, we will use the notation $u^{\{k_1,k_2,\ldots,k_s\}}$ to denote the string formed by complementing the k_1th, k_2th, ..., and k_sth bits of u. For example, $0011010^2 = 0111010$ and $0011010^{\{3,4\}} = 0000010$ in a 128-node hypercube.

The dimension k edges in a hypercube form a perfect matching for each k, $1 \leq k \leq \log N$. (Recall that a perfect matching for an N-node graph is a set of $N/2$ edges that do not share any nodes.) Moreover, removal of the dimension k edges for any $k \leq \log N$ leaves two disjoint copies of an

$\frac{N}{2}$-node hypercube. Conversely, an N-node hypercube can be constructed from two $\frac{N}{2}$-node hypercubes by simply connecting the ith node of one $\frac{N}{2}$-node hypercube to the ith node of the other for $0 \leq i < \frac{N}{2}$. For example, see Figure 3-2.

In addition to a simple recursive structure, the hypercube also has many of the other nice properties that we would like a network to have. In particular, it has low diameter ($\log N$) and high bisection width ($N/2$). The bound on the diameter is easily proved by observing that any two nodes $u = u_1 u_2 \cdots u_{\log N}$ and $v = v_1 v_2 \cdots v_{\log N}$ are connected by the path

$$u_1 u_2 \cdots u_{\log N} \rightarrow v_1 u_2 \cdots u_{\log N} \rightarrow v_1 v_2 u_3 \cdots u_{\log N}$$
$$\rightarrow \cdots \rightarrow v_1 v_2 \cdots v_{\log N - 1} u_{\log N} \rightarrow v_1 v_2 \cdots v_{\log N}.$$

The bound on bisection width is established by showing that the smallest bisection consists of the edges in a single dimension. The proof follows as a special case of Theorem 1.21 from Section 1.9.

As an interesting aside, it is worth noting that a hypercube can be bisected by removing far fewer than $\frac{N}{2}$ nodes, even though $\frac{N}{2}$ edges are required to bisect the N-node hypercube. For example, consider the partition formed by removing all nodes with size $\left\lfloor \frac{\log N}{2} \right\rfloor$ and $\left\lceil \frac{\log N}{2} \right\rceil$. (The *size*, or *weight*, of a node in the hypercube is the number of 1s contained in its binary string.) A simple calculation reveals that removal of these nodes forms a bisection with $\Theta(N/\sqrt{\log N})$ nodes, which is the best possible. The details of these and some related results are left to the exercises (see Problems 3.3–3.7).

It is also worth noting that the hypercube possesses many symmetries. For example, it is *node* and *edge symmetric*. In other words, by just relabelling nodes, we can map any node onto any other node, and any edge onto any other edge. More precisely, for any pair of edges (u, v) and (u', v') in an N-node hypercube H, there is an automorphism σ of H such that $\sigma(u) = u'$ and $\sigma(v) = v'$. (An *automorphism* of a graph is a one-to-one mapping of the nodes to the nodes such that edges are mapped to edges.) In fact, there are many such automorphisms. For example, let $u = u_1 u_2 \cdots u_{\log N}$, $u' = u'_1 u'_2 \cdots u'_{\log N}$, k be the dimension of (u, v), and k' be the dimension of (u', v'). Then for any permutation π on $\{1, 2, \ldots, \log N\}$ such that $\pi(k') = k$, we can define an automorphism σ with the desired property by setting

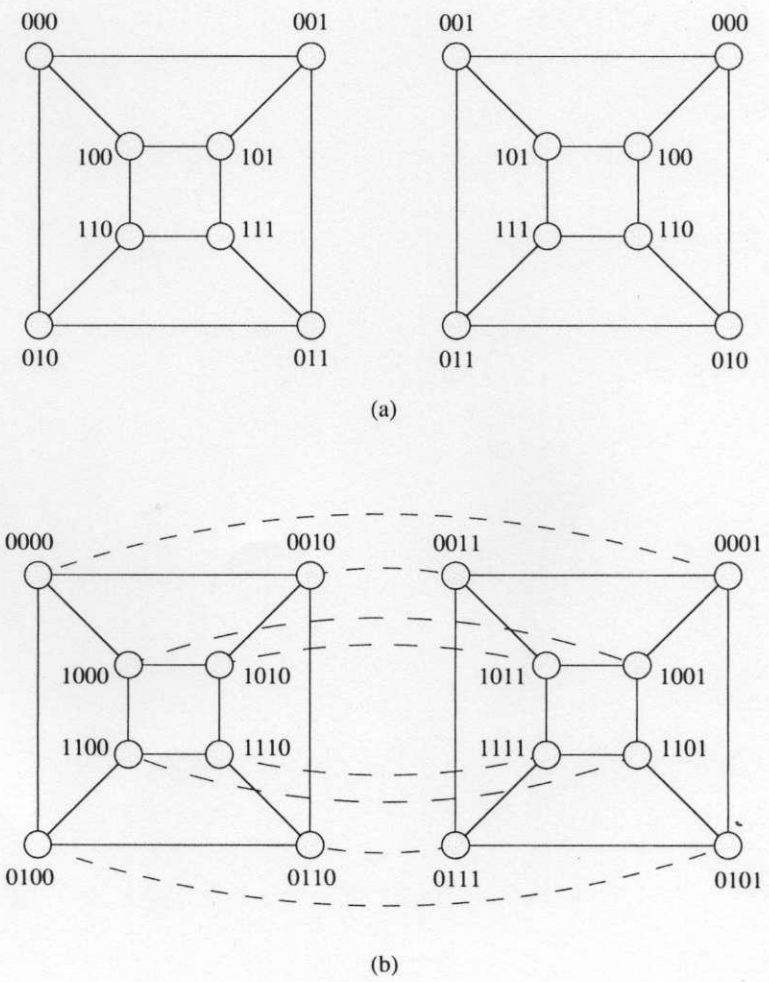

Figure 3-2 *Construction of a four-dimensional hypercube (b) from two three-dimensional hypercubes (a). Dashed edges form a matching between the two three-dimensional cubes.*

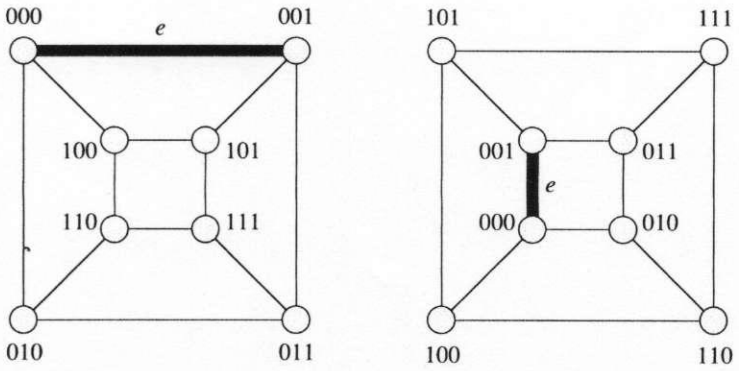

Figure 3-3 *Two labellings of the 8-node hypercube. By relabelling appropriately, we could have mapped edge $e = (000, 001)$ to any position in the network.*

$$\sigma(x_1 x_2 \cdots x_{\log N}) = (x_{\pi(1)} \oplus u_{\pi(1)} \oplus u'_1) \mid (x_{\pi(2)} \oplus u_{\pi(2)} \oplus u'_2) \mid \cdots \mid (x_{\pi(\log N)} \oplus u_{\pi(\log N)} \oplus u'_{\log N}). \quad (3.1)$$

(Here and throughout Chapter 3, we use the notation $\alpha \mid \beta$ to denote the concatenation of α and β.) It is a simple exercise to check that σ is an automorphism of the hypercube with the desired properties. (See Problem 3.10.)

As an example, we have illustrated two labellings of the 8-node hypercube in Figure 3-3. In the example, we have mapped the edge $(000, 001)$ to edge $(110, 100)$ using the automorphism

$$\sigma(x_1 x_2 x_3) = (x_1 \oplus 1) \mid (x_3 \oplus 1) \mid x_2.$$

In general, we can rearrange the dimensions of the edges in any order that we want (by varying π) without altering the network. (See Problems 3.11–3.13.) We will use such symmetries routinely in the chapter to simplify explanations.

3.1.2 Containment of Arrays

One of the most interesting properties of the N-node hypercube is that it contains every N-node array as a subgraph. This result holds true even for high-dimensional arrays and even if wraparound edges are allowed. For example, the embedding of a 4×4 array in a 16-node hypercube is shown