ticularly important since it means that all of the algorithms described in Chapter 2 can be implemented on these networks with only a constant factor loss in efficiency.

We conclude in Subsection 3.2.4 with a review of network containment results analogous to those proved for the hypercube in Subsections 3.1.2–3.1.6. Among other things, we find that the butterfly, cube-connected-cycles, and Beneš network contain linear arrays and complete binary trees with constant dilation, but that any embedding of higher-dimensional arrays requires logarithmic dilation, which is the worst possible. We also prove a general result that every N-node connected network contains an N-node linear array with dilation 3.

3.2.1 Definitions and Properties

In what follows, we describe the butterfly, a variant of the butterfly called the wrapped butterfly, the cube-connected-cycles, and the Beneš network. All four networks have a similar structure, and all four are computationally equivalent.

The Butterfly

The r-dimensional butterfly has $(r+1)2^r$ nodes and $r2^{r+1}$ edges. The nodes correspond to pairs $\langle w, i\rangle$ where i is the *level* or *dimension* of the node $(0 \le i \le r)$ and w is an r-bit binary number that denotes the *row* of the node. Two nodes $\langle w, i\rangle$ and $\langle w', i'\rangle$ are linked by an edge if and only if $i' = i+1$ and either:

1) w and w' are identical, or

2) w and w' differ in precisely the i'th bit.

If w and w' are identical, the edge is said to be a ***straight edge***. Otherwise, the edge is a ***cross edge***. For example, see Figure 3-19. In addition, edges connecting nodes on levels i and $i+1$ are said to be ***level*** $i+1$ ***edges***.

The butterfly and hypercube are quite similar in structure. In particular, the ith node of the r-dimensional hypercube corresponds naturally to the ith row of the r-dimensional butterfly, and an ith dimension edge (u, v) of the hypercube corresponds to cross edges $(\langle u, i-1\rangle, \langle v, i\rangle)$ and $(\langle v, i-1\rangle, \langle u, i\rangle)$ in level i of the butterfly. In effect, the hypercube is just a folded up butterfly (i.e., we can obtain a hypercube from a butterfly by merging all butterfly nodes that are in the same row and then removing the extra copy of each edge). Hence, any single step of N-node hypercube

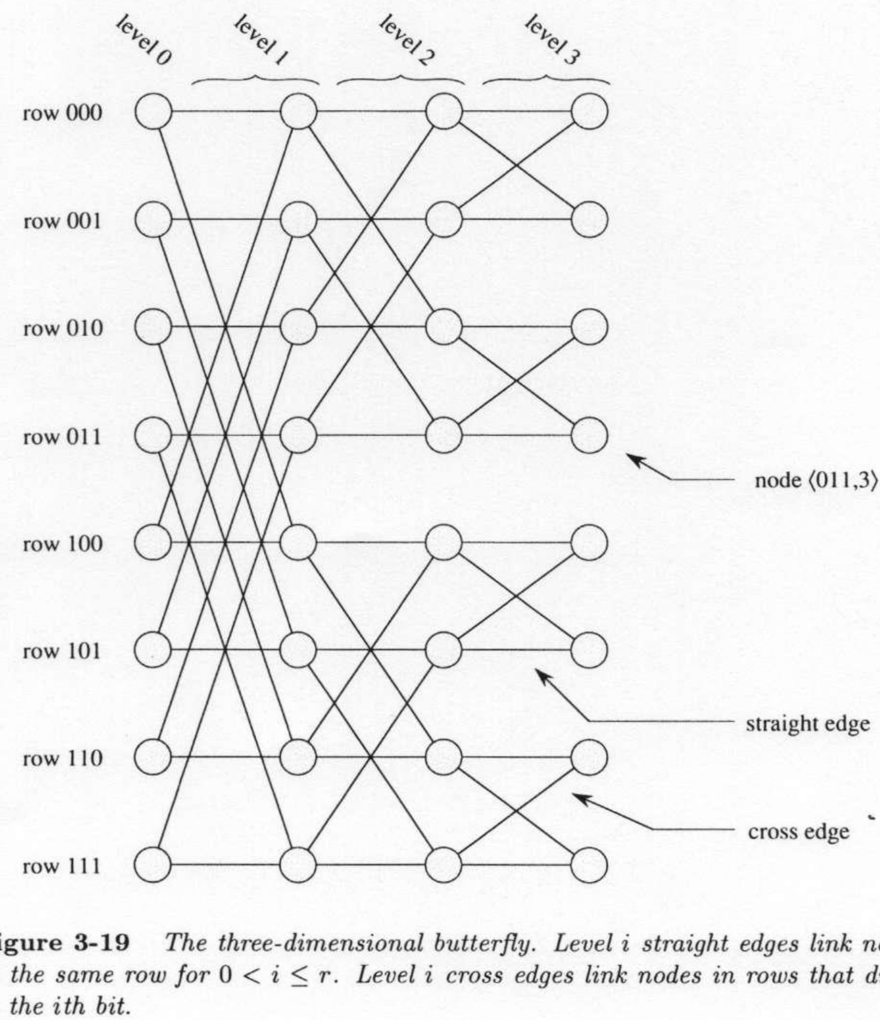

Figure 3-19 *The three-dimensional butterfly. Level i straight edges link nodes in the same row for $0 < i \leq r$. Level i cross edges link nodes in rows that differ in the ith bit.*

calculation can be simulated in $\log N$ steps on an $N(\log N + 1)$-node butterfly by having the ith row of the butterfly simulate the operation of the ith node of the hypercube for each i.

Because of the great similarity between the butterfly and the hypercube, the butterfly has several nice properties. First, it has a simple recursive structure. For example, it can be seen from Figure 3-19 that one r-dimensional butterfly contains two $(r-1)$-dimensional butterflies as subgraphs. (Just remove the level 0 nodes of the r-dimensional butterfly. Alternatively, we could remove the level r nodes, as is done in Figure 3-20, although it takes a little longer to realize that the resulting graph is simply two $(r-1)$-dimensional butterflies.)

Another useful property of the r-dimensional butterfly is that the level 0 node in any row w is linked to the level r node in any row w' by a unique path of length r. The path traverses each level exactly once, using the cross edge from level i to level $i+1$ if and only if w and w' differ in the $(i+1)$st bit. For example, see Figure 3-21. As a simple consequence of this fact, we can see that the N-node butterfly has diameter $O(\log N)$.

Like the hypercube, the butterfly also has a large bisection width. In particular, the bisection width of the N-node butterfly is $\Theta(N/\log N)$. To construct a bisection of this size, simply remove the cross edges from a single level. To show that $\Omega(N/\log N)$ is a lower bound on the bisection width of the network, we can apply the same technique used to prove Theorem 1.21 in Section 1.9. (For example, see Problem 3.89.)

The Wrapped Butterfly

For computational purposes, the first and last levels of the butterfly are sometimes merged into a single level. In particular, node $\langle w, 0\rangle$ is merged into node $\langle w, r\rangle$ for each w. The result is an r-level graph with $r2^r$ nodes, each of degree 4. Two nodes $\langle w, i\rangle$ and $\langle w', i'\rangle$ are linked by an edge if and only if $i' \equiv i+1 \bmod r$ and either $w = w'$ or w and w' differ in the i'th bit. Such edges are called *level i'* edges. To distinguish between this structure and the unmerged butterfly of Figure 3-19, we will refer to the former as a *wrapped butterfly* and the latter as an *ordinary butterfly*. For example, a three-dimensional wrapped butterfly is illustrated in Figure 3-22.

At first glance, it might seem that the wraparound edges could make the wrapped butterfly more powerful than the ordinary butterfly from a computational point of view. This turns out not to be the case, however. In fact, the relationship between the butterfly and wrapped butterfly is

[L]

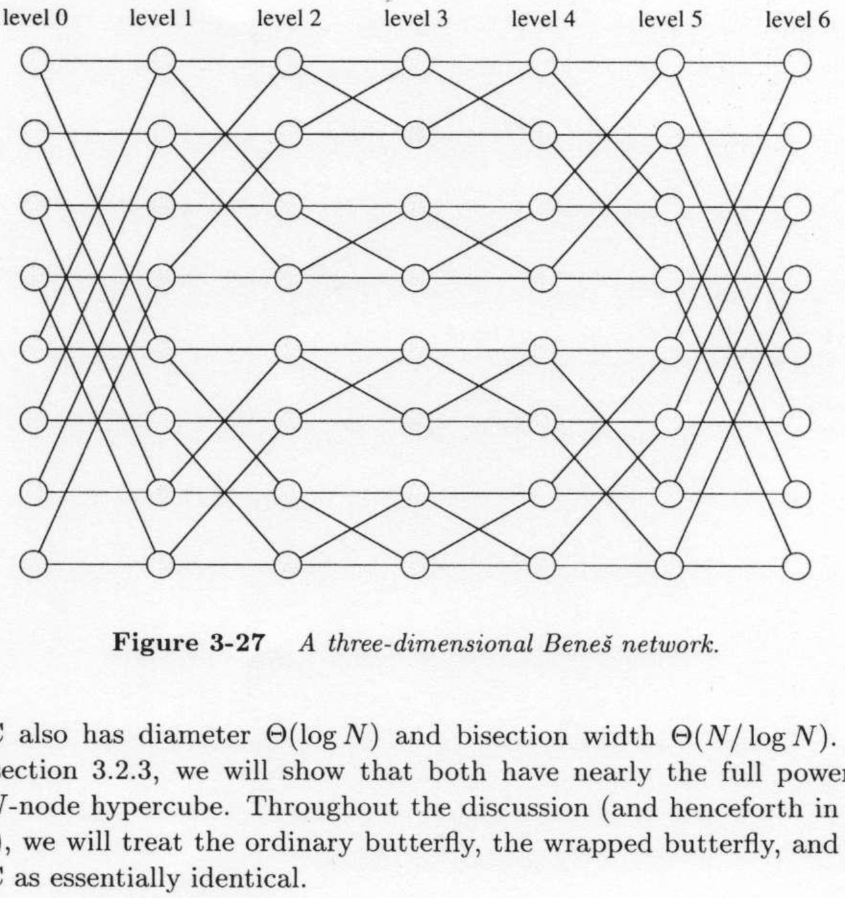

Figure 3-27 *A three-dimensional Beneš network.*

CCC also has diameter $\Theta(\log N)$ and bisection width $\Theta(N/\log N)$. In Subsection 3.2.3, we will show that both have nearly the full power of an N-node hypercube. Throughout the discussion (and henceforth in the text), we will treat the ordinary butterfly, the wrapped butterfly, and the CCC as essentially identical.

The Beneš Network

Many other variations of the butterfly have also been proposed in the literature, and we will mention several of them in Section 3.8. There is one additional variation, however, called the Beneš network, that is worthy of special attention. The *Beneš network* consists of back-to-back butterflies, as shown in Figure 3-27. Overall, the r-dimensional Beneš network has $2r + 1$ levels, each with 2^r nodes. The first and last $r + 1$ levels in the network form an r-dimensional butterfly. (The middle level of the Beneš network is shared by these butterflies.)

Not surprisingly, the Beneš network is very similar to the butterfly, in terms of both its computational power and its network structure. Indeed, at first glance, the network hardly seems worth defining at all.

The reason for defining the Beneš network is that it is an excellent example of a *rearrangeable network*. A network with N inputs and N outputs is said to be rearrangeable if for any one-to-one mapping π of the inputs to the outputs, we can construct edge-disjoint paths in the network linking the ith input to the $\pi(i)$th output for $1 \le i \le N$. In the case of the r-dimensional Beneš network, we can have *two* inputs for each level 0 node and *two* outputs for every level $2r$ node, and still connect every permutation of inputs to outputs with edge-disjoint paths. (In this case, $N = 2^{r+1}$.) For example, we illustrated the paths for the mapping $\binom{1\,2\,3\,4\,5\,6\,7\,8}{6\,4\,5\,8\,1\,2\,3\,7}$ in an 8-input Beneš network in Figure 3-28. For ease of illustration, each node of the Beneš network is displayed as a 2×2 switch that connects its two incoming edges from the left to the two outgoing edges on the right in one of two ways (crossing or straight through).

A quick inspection of Figure 3-28 reveals that every edge of the Beneš network must be used to form the edge-disjoint paths connecting the inputs to the outputs, no matter what permutation is used. Under the circumstances, it seems extraordinary that we can find edge-disjoint paths for any permutation. Nevertheless, the result is true, and it is even fairly easy to prove, as we show in the following theorem.

THEOREM 3.10 *Given any one-to-one mapping π of 2^{r+1} inputs to 2^{r+1} outputs on an r-dimensional Beneš network, there is a set of edge-disjoint paths from the inputs to the outputs connecting input i to output $\pi(i)$ for $1 \le i \le 2^{r+1}$.*

Proof. The proof is by induction on r. If $r = 1$, the Beneš network consists of a single node (i.e., a single 2×2 switch) and the result is obvious. Hence, we assume that the result is true for an $(r-1)$-dimensional Beneš network and try to establish the induction.

The key to the induction is to observe that the middle $2r - 1$ levels of an r-dimensional Beneš network comprise two $(r-1)$-dimensional Beneš networks. Hence, it will be sufficient to decide whether each path is to be routed through the upper sub-Beneš network or the lower sub-Beneš network. For example, the path from input 2 to output 4 in Figure 3-28 is routed through the lower subnetwork, while the path from input 1 to output 6 is routed through the upper subnetwork.

The only constraints that we have on whether paths use the upper or lower subnetworks are that paths from inputs $2i - 1$ and $2i$ must use different subnetworks for $1 \le i \le 2^r$, and that paths to outputs $2i - 1$ and $2i$ must use different subnetworks. This is because each switch on

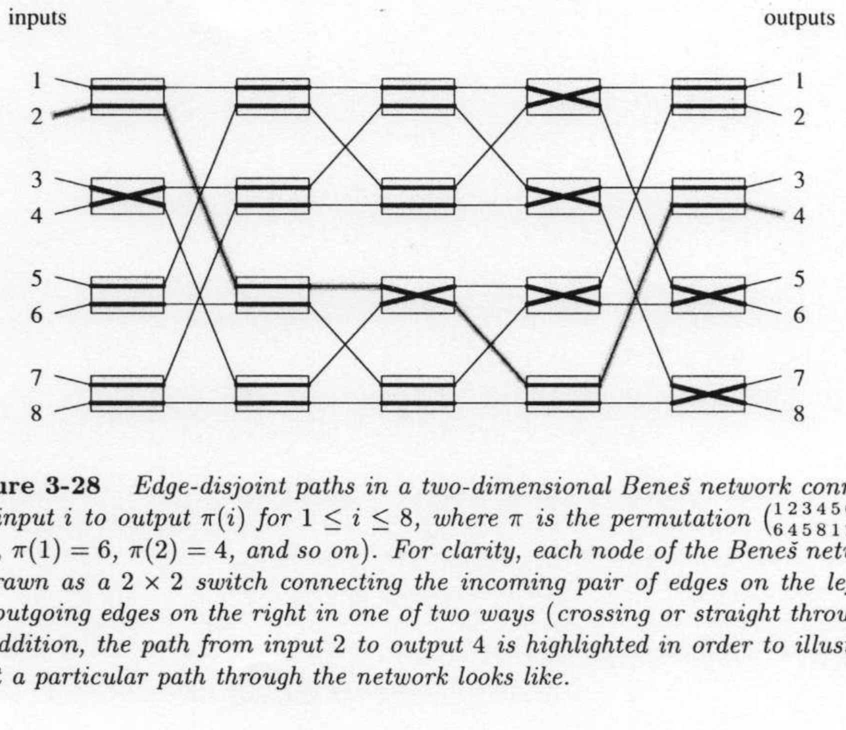

Figure 3-28 *Edge-disjoint paths in a two-dimensional Beneš network connecting input i to output $\pi(i)$ for $1 \le i \le 8$, where π is the permutation $\binom{1\,2\,3\,4\,5\,6\,7\,8}{6\,4\,5\,8\,1\,2\,3\,7}$ (i.e., $\pi(1) = 6$, $\pi(2) = 4$, and so on). For clarity, each node of the Beneš network is drawn as a 2×2 switch connecting the incoming pair of edges on the left to the outgoing edges on the right in one of two ways (crossing or straight through). In addition, the path from input 2 to output 4 is highlighted in order to illustrate what a particular path through the network looks like.*

the first and last levels of the Beneš network has precisely one connection to each of the upper and lower subnetworks. Thus the paths sharing a node at the first level must go to different subnetworks, and the paths sharing a node at the last level must come from different subnetworks.

Fortunately, these constraints are easy to satisfy. For example, the paths for the permutation in Figure 3-28 were constructed as follows. First, we decided to route the path from input 1 to output 6 through the upper subnetwork. This meant that the path from input 3 to output 5 had to use the lower subnetwork (since the paths to outputs 5 and 6 must use different subnetworks). This meant that the path from input 4 to output 8 had to use the upper subnetwork (since the paths from inputs 3 and 4 must use different subnetworks). Continuing in like fashion, we find that the path from input 8 to output 7 uses the lower subnetwork, the path from input 7 to output 3 uses the upper subnetwork, and that the path from input 2 to output 4 uses the lower subnetwork. The last choice on the path from input 2 closes the loop in the constraint imposed by the initial decision to route the path from input 1 through the upper subnetwork. To complete the first-level choices, we route the path from

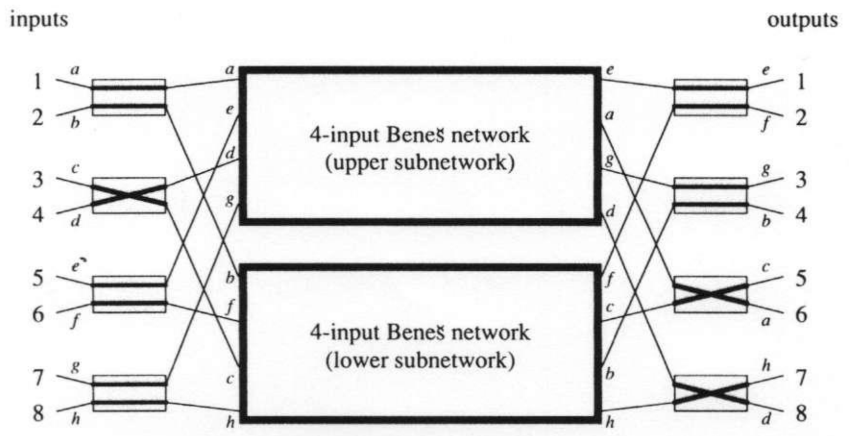
inputs
outputs
4-input Beneš network (upper subnetwork)
4-input Beneš network (lower subnetwork)
1
2
3
4
5
6
7
8
a
b
c
d
e
f
g
h

Figure 3-29 *The first step in finding the edge-disjoint paths for the permutation* $\binom{1\,2\,3\,4\,5\,6\,7\,8}{6\,4\,5\,8\,1\,2\,3\,7}$. *The paths* $1 \to 6$, $4 \to 8$, $5 \to 1$, *and* $7 \to 3$ *(labelled a, d, e, and g, above) are routed through the upper subnetwork, while the paths* $2 \to 4$, $3 \to 5$, $6 \to 2$, *and* $8 \to 7$ *(labelled b, c, f, and h, above) are routed through the lower subnetwork. At the recursive stage, the upper subnetwork routes the permutation* $\binom{1\,2\,3\,4}{2\,1\,4\,3}$ *and the lower subnetwork routes the permutation* $\binom{1\,2\,3\,4}{3\,1\,2\,4}$.

input 5 to output 1 through the upper subnetwork and the path from input 6 to output 2 through the lower subnetwork. For example, these choices are illustrated in Figure 3-29. We can now complete the routing of the paths by using induction on the upper and lower subnetworks.

But how do we know for sure that we can always assign each path to the upper or lower subnetwork in a way that satisfies all of the constraints? The answer is surprisingly easy. We start by routing the first path (e.g., the path from input 1) through the upper subnetwork. We next satisfy the constraint generated at the output by routing the corresponding path (e.g., the path to output 5 in the previous example) through the lower subnetwork. We next satisfy the constraint newly generated at the input by routing the appropriate path (e.g., the path from input 4 in the previous example) through the upper subnetwork. We keep on going back and forth through the network, satisfying constraints at the inputs by routing through the upper subnetwork and satisfying constraints at the outputs by routing through the lower subnetwork. Eventually we will close the loop by routing a path through the lower subnetwork (in response to an output constraint) that shares an input switch with the first path that was routed. Since the first path that was routed used the upper subnetwork, the original input constraint is now also satisfied. If any additional paths need to be routed, we continue as before, starting over again with an arbitrary unrouted path. In this way, all paths can be assigned to the upper or lower subnetworks without conflict (i.e., we can set the switches at the first and last levels of the Beneš network so that both ends of every path are connected to the same subnetwork). The remainder of the path routing and switch setting is handled by induction in the subnetworks. Hence, we have established the inductive hypothesis, thereby proving the theorem. ■

In the case that each level 0 node of the r-dimensional Beneš network has just one input and each level $2r$ node has just one output, then the paths from the inputs to the outputs can be constructed so as to be node-disjoint (instead of only edge-disjoint). For example, we have illustrated node-disjoint paths through a two-dimensional Beneš network for the permutation $\binom{1\,2\,3\,4}{3\,1\,4\,2}$ in Figure 3-30. The proof of this result is identical to the proof of Theorem 3.10, except that the paths from inputs i and $i + 2^{r-1}$ must use different subnetworks for $1 \le i \le 2^{r-1}$ (instead of the paths from inputs $2i - 1$ and $2i$) and the paths to outputs i and $i + 2^{r-1}$ must use different subnetworks for $1 \le i \le 2^{r-1}$. For example, the path from input 1 to output 3 is routed through the upper subnetwork in Figure 3-30, whereas the path from input 3 and the path to output 1 both use the lower subnetwork. The remainder of the proof is left as an easy exercise. (See Problem 3.100.) For ease of reference, we state the result formally in the following theorem.

THEOREM 3.11 *Given any one-to-one mapping of π of 2^r inputs to 2^r outputs in an r-dimensional Beneš network (one input per level 0 node and one output per level $2r$ node), there is a set of node-disjoint paths from the inputs to the outputs connecting input i to output $\pi(i)$ for $1 \le i \le 2^r$.*

Theorems 3.10 and 3.11 have many important applications. For example, we will use them in the next subsection to show that an N-node butterfly can simulate any other N-node bounded-degree network with only $O(\log N)$ slowdown. Of course, similar results are also true for the hypercube, as well as the other hypercubic networks that we will study.

The only drawback to Theorems 3.10 and 3.11 is that we do not know how to set the switches on-line. In other words, each switch needs to be

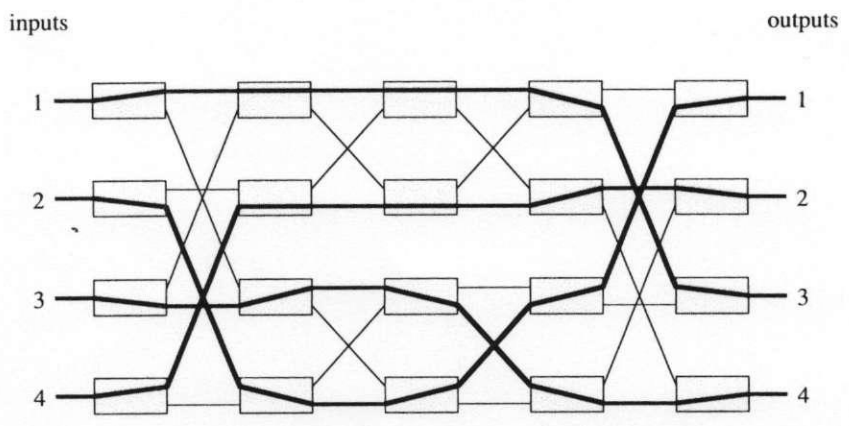

Figure 3-30 *Node-disjoint paths in a two-dimensional Beneš network connecting input i to output $\pi(i)$ for $1 \le i \le 4$ where π is the permutation $\binom{1\,2\,3\,4}{3\,1\,4\,2}$. Edges contained in the paths are shaded. Because of the node-disjoint property of the paths, precisely one path passes through each switch in the network.*

told what to do by a global control that has knowledge of the permutation being routed. We will describe numerous methods for overcoming this difficulty in Sections 3.4 and 3.5. For now, however, we will be content with the off-line nature of the result.

3.2.2 Simulation of Arbitrary Networks ⋆

Like the hypercube, the butterfly can simulate many networks without slowdown. In fact, we will discuss in the next few subsections how all of the algorithms described in Chapters 1 and 2 can be implemented on a butterfly with only a constant factor loss in efficiency. Before we do this, however, it is useful to prove the following broader result: The N-node butterfly can simulate any N-node bounded-degree network with only an $O(\log N)$-factor slowdown. As a consequence, we will have shown that the butterfly is *universal* in the sense that it can simulate anything with comparable hardware (i.e., processors and wires) with only an $O(\log N)$-factor slowdown (the least possible, in general). Similar results also hold for the hypercube and other related networks.

The key step in showing that the butterfly can efficiently simulate any other bounded-degree network is to show that the N-node butterfly can route any N-packet permutation in $O(\log N)$ steps provided that the permutation is known in advance. In other words, given an N-node butterfly