Shuffle-Oriented Interconnection Networks

Arnold L. Rosenberg

Computer and Information Science Department
University of Massachusetts

COINS Technical Report 88-84

SHUFFLE-ORIENTED INTERCONNECTION NETWORKS

Arnold L. Rosenberg
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

September 27, 1988

Abstract

The boolean Hypercube and its butterfly-oriented and shuffle-oriented derivatives have become the dominant interconnection networks for massively parallel architectures. It is known that for a large class of algorithms, the bounded-degree derivatives can equal the speed of the high-degree Hypercube; thus, these algorithms do not afford one a basis for selecting among these derivatives. In this paper, we propose a graph-theoretic framework for comparing the communication powers of competing interconnection networks by seeing how efficiently one can simulate the other *on general computations*. Within this framework, we present new, surprisingly efficient simulations by shuffle-oriented networks of like-sized butterfly-oriented networks: these simulations incur a slowdown that is *doubly* logarithmic in the size of the simulated network, i.e., exponentially faster than the anticipated logarithmic slowdown. The simulation mappings, which can be computed in linear time, also afford one an algorithmic technique for translating programs for butterfly-oriented architectures into equivalent programs for shuffle-oriented architectures, the latter programs incurring the indicated slowdown factor.

1. INTRODUCTION

1.1. Background

The boolean Hypercube and its bounded-degree derivatives, such as the *butterfly-oriented* Butterfly and Cube-Connected-Cycles (CCC) networks and the *shuffle-oriented*

Shuffle-Exchange and deBruijn networks,[1] have become the dominant interconnection networks for massively parallel architectures. Indeed, architectures based on these networks have been built both in industry and academia.

Among these interconnection networks, the Hypercube is the clear favorite because of its demonstrated communication power [7, 11, 13, 15, 23], its structural uniformity that simplifies programming [2], and its robustness in the face of faults [4, 12]. The major shortcoming of the Hypercube is its having high node-degrees.[2] The technical difficulties attendant to implementing such high-degree networks have led to the development of several butterfly-oriented bounded-degree "approximations" of the Hypercube, most notably the Butterfly and Cube-Connected-Cycles networks [21]. These networks were constructed with a certain popular genre of Hypercube-algorithm, called *ascend-descend* algorithms,[3] in mind and so can simulate the Hypercube with little or no slowdown on a large, important class of computational problems. Yet, in a sense, butterfly-oriented networks just replace one implementational problem with another, since they use $N \cdot \log_2 N$ nodes (processors) to simulate the N-node Hypercube. Further, algebraic, transformations of these large networks [3] yield the smaller, shuffle-oriented bounded-degree "approximations" of the Hypercube, most notably the Shuffle-Exchange[4] [16, 19, 25] and deBruijn [9, 20] networks. Shuffle-oriented networks share the size[5] of the Hypercube, yet avoid its large node-degrees; and, they afford one computational efficiency (roughly) equal to that of the Butterfly and CCC, on certain computational tasks, including ascend-descend algorithms (cf. Section 4.1). Indeed, many in the parallel computation community posit the computational equivalence of the four Hypercube-derivative networks we have mentioned explicitly, based on their performance on ascend-descend algorithms.

This paper is part of a research program [3, 5, 6, 11] dedicated to investigating this asserted equivalence analytically (as opposed to empirically) in the context of arbitrary computational tasks (as opposed to a restricted class of algorithms). A similar goal motivates [24] (which concentrates on evaluating competitors of the Hypercube) and [10] (which concentrates on comparing the Shuffle-Exchange network with the Hypercube). The comparison in [24] is purely qualitative, hence only peripherally related to

[1] These networks are defined formally in Section 2.1.

[2] Each node of the N-node Hypercube has degree $\log_2 N$.

[3] Ascend-descend algorithms exercise the Hypercube's communication facilities in a stylized way: at each time step, all interprocessor communication is across one Hypercube dimension. Many well-known, efficient algorithms reside in this class, for such tasks as sorting, computing convolutions (using the FFT algorithm [1, Ch. 7]), and matrix operations; see [21] for details.

[4] The Shuffle-Exchange network can also be derived directly from the Hypercube, by a geometric transformation [17].

[5] The *size* of a network is its number of nodes.

our study. The comparison in [10] focusses on a notion of the "cost effectiveness" of a network, which is measured by combining topological properties like diameter, bisection width, and degree of connectivity. In contrast, we are aiming for a more computation-oriented comparison. Therefore, we use the theory of graph embeddings to model the notion of one interconnection network *simulating* another on a general computational task. A result of Leighton [17] (Proposition 2.1) justifies our treating what is basically an algorithmic problem in a purely graph-theoretic framework. Within this framework, we present new, surprisingly efficient simulations by shuffle-oriented networks of like-sized butterfly-oriented networks. Simulations based on the same approach appear in [13], wherein Hypercubes simulate like-sized complete binary tree networks with only constant slowdown; in [11], wherein Hypercubes simulate like-sized butterfly-oriented networks with only constant slowdown; in [8, 23], wherein Hypercubes efficiently simulate meshes; and in [5], wherein Butterfly networks simulate complete binary tree networks with constant slowdown, X-trees with doubly logarithmic slowdown, and meshes with logarithmic slowdown, all simulations being optimal.

Our approach to simulation has an unexpected dividend relating to the problem of programming architectures that are based on the simulating interconnection networks. We assume that the processing elements of the simulating architecture are at least as powerful as those of the simulated architecture: Our approach yields an algorithmic technique for translating any program P for the simulated architecture into a functionally equivalent program Q for the simulating architecture, that is slower than P by at most the simulation slowdown factor. This fact is especially valuable when the simulating network is more cost-effective to build, but is harder to program, than the simulating network. This is likely to be the case with the networks of interest here, since the Butterfly and CCC networks enjoy structural uniformities (such as node-transitivity) which enhance programmability, but which are not shared by the deBruijn and Shuffle-Exchange networks [3].

After noting the effective equivalence of the Shuffle-Exchange and deBruijn networks (Proposition 3.1) and of the Butterfly and CCC networks (Proposition 3.2) in Section 3.1, and reviewing some salient properties of shuffle-oriented networks in Section 3.2, we establish our main results in Section 4:

- We present a graph-theoretic demonstration (to our knowledge, the first such) that an architecture based on the Shuffle-Exchange network can execute any ascend-descend algorithm as fast as can an architecture based on the Hypercube or the CCC network. This improves earlier claimed execution times by a factor of 2.

- We prove that an architecture based on a shuffle-oriented network can simulate

an architecture based on a *like-sized* butterfly-oriented network, on a general computational task, with only *doubly logarithmic* slowdown due to communication delays, rather than the expected logarithmic slowdown.[6]

Since there are substantial differences in the costs of implementing the networks we study here, we hope that our results might help one to evaluate the cost/performance tradeoffs in selecting among these networks.

2. THE FORMAL FRAMEWORK

2.1. The Graphs of Interest

The most basic issue is how to view parallel architectures — really, arrays of identical processing elements (PEs) — as undirected graphs:[7] We assume a *pulsed* model of computation, wherein *computation* steps alternate with (point-to-point) *communication* steps between adjacent PEs. Thus, formally, we have the **nodes** of the graph represent the **PEs** of the array, and the **edges** of the graph represent the inter-PE **communication links**.

We use the following notation throughout. For any positive integer n, $Z_n =_{\text{def}} \{0, 1, \cdots, n-1\}$. For any set S and positive integer k, S^k denotes the set of all length-k strings of elements of S, and $|S|$ denotes the cardinality of S.

Let n be a positive integer. The *order-n* **deBruijn graph** $\mathcal{D}(n)$ is the graph whose nodes comprise the set Z_2^n. $\mathcal{D}(n)$ has two types of edges: Each *shuffle* edge connects a node of the form βx, where $\beta \in Z_2$ and $x \in Z_2^{n-1}$, to node $x\beta$; each *shuffle-exchange* edge connects a node of the form βx, where $\beta \in Z_2$ and $x \in Z_2^{n-1}$, to node[8] $x(\beta \oplus 1)$; see Fig. 1. Thus, $\mathcal{D}(n)$ has 2^n nodes and roughly 2^{n+1} edges.[9]

Let n be a positive integer. The *order-n* **Shuffle-Exchange graph** $\mathcal{S}(n)$ is a graph whose nodes comprise the set Z_2^n. $\mathcal{S}(n)$ has two types of edges: Each *shuffle* edge connects a node of the form βx, where $\beta \in Z_2$ and $x \in Z_2^{n-1}$, to node $x\beta$; each *exchange* edge connects a node of the form $x\beta$, where $\beta \in Z_2$ and $x \in Z_2^{n-1}$, to node $x(\beta \oplus 1)$; see Fig. 2. Thus, $\mathcal{S}(n)$ has 2^n nodes and roughly $3 \cdot 2^{n-1}$ edges.[9]

[6] That is, the slowdown when simulating an N-node architecture is $O(\log \log N)$ rather than the exponentially larger $O(\log N)$.

[7] An undirected graph G is specified by a set V_G of *nodes* and a set E_G of two-element subsets of V_G called *edges*.

[8] $\oplus$ denotes addition modulo 2.

[9] The exact number of edges involves a small additive constant that depends on the parity of n.

Let n be a positive integer. The *order-n* **Butterfly graph** $\mathcal{B}(n)$ has node-set $V_n = Z_n \times Z_2^n$; we call ℓ the *level* of node $\langle \ell, \vec{\delta} \rangle \in V_n$. The edges of $\mathcal{B}(n)$ are of two types: For each $\ell \in Z_n$ and each $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, the node

$$\langle \ell, \delta_0\delta_1 \cdots \delta_{n-1} \rangle, \text{ on level } \ell \text{ of } \mathcal{B}(n),$$

is connected by a *straight-edge* with node

$$\langle \ell', \delta_0\delta_1 \cdots \delta_{m-1} \rangle, \text{ on level } \ell' =_{\text{def}} \ell + 1 \pmod{n} \text{ of } \mathcal{B}(n);$$

and it is connected by a *cross-edge* with node

$$\langle \ell', \delta_0\delta_1 \cdots \delta_{\ell-1}(\delta_\ell \oplus 1)\delta_{\ell+1} \cdots \delta_{n-1} \rangle \text{ of } \mathcal{B}(n);$$

see Fig. 3. Thus, $\mathcal{B}(n)$ has $n2^n$ nodes and $n2^{n+1}$ edges.

Let n be a positive integer. The *order-n* **Cube-Connected-Cycles graph**, (*CCC*, for short), $\mathcal{C}(n)$ has node-set $V_n = Z_n \times Z_2^n$; we call ℓ the *level* of node $\langle \ell, \vec{\delta} \rangle \in V_n$. The edges of $\mathcal{C}(n)$ are of two types: For each $\ell \in Z_n$ and each $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, the node

$$\langle \ell, \delta_0\delta_1 \cdots \delta_{n-1} \rangle, \text{ on level } \ell \text{ of } \mathcal{C}(n),$$

is connected by a *level* edge with node

$$\langle \ell, \delta_0\delta_1 \cdots \delta_{\ell-1}(\delta_\ell \oplus 1)\delta_{\ell+1} \cdots \delta_{n-1} \rangle, \text{ on level } \ell \text{ of } \mathcal{C}(n);$$

and it is connected by a *straight-edge* with node

$$\langle \ell', \delta_0\delta_1 \cdots \delta_{m-1} \rangle, \text{ on level } \ell' = \ell + 1 \pmod{n} \text{ of } \mathcal{C}(n);$$

see Fig. 4. Thus, $\mathcal{C}(n)$ has $n2^n$ nodes and $3n2^{n-1}$ edges.

The following graph is not of direct interest but is useful in our simulations.

The *length-n* **ring graph** $\mathcal{R}(n)$ is the graph whose nodes comprise the set Z_n and whose edges connect each node v with node $v + 1 \pmod{n}$.

2.2. Simulation via Graph Embeddings

The next basic issue is how to represent the simulation of one array by another. We assume that the PEs of the simulating array are sufficiently numerous and sufficiently powerful to simulate the PEs of the simulated array step for step — so no delay is incurred because of any computational step. We restrict attention to simulations that honor the pulsed computation regimen of the simulated array: The simulating array alternates (single) steps that simulate one *computation* step of the simulated array, with (possibly multiple) steps that simulate one *communication* step of the simulated array. The slowdown incurred by a simulation arises from having to simulate communication steps that are tailored to the structure of the simulated interconnection network on the simulating network. This delay results both from mismatched adjacency structures and from congested communication lines. Our formal notion of simulation resides in the following notion of graph embedding. An *embedding* of the graph G in the graph H is specified by:

- a one-to-one *assignment* α of the nodes of G to the nodes of H:

$$\alpha : V_G \to V_H$$

- a *routing* ρ of each edge of G along a distinct path in H:[10]

$$\rho : E_G \to \text{Paths}(H)$$

Fundamental to our representing simulations by graph embeddings is our assessing the *delay* incurred by a simulation. We use three measures for this purpose. Say that we have an embedding (α, ρ) of G in H:

- The *Dilation* of the embedding is the maximum amount that the routing ρ "stretches" any edge of G:

$$\mathit{Dilation}(\alpha, \rho) = \max_{(u,v)\in E_G} \text{Length}(\rho(u, v))$$

[10] A *path* in H from node $u \in V_H$ to node $v \in V_H$ is a sequence π of nodes

$$u = v_0, v_1, \ldots, v_{\ell-1}, v_\ell = v$$

such that, for each $0 \leq i < \ell$, $(v_i, v_{i+1}) \in E_H$; we say that the path π has *length* ℓ. By abuse of notation, we write "$(v_i, v_{i+1}) \in \pi$".

- The *(edge) Congestion* of the embedding is the maximum number of edges of G that ρ routes over a single edge of H:

$$Congestion(\alpha, \rho) = \max_{e \in E_H} |\{e' \in E_G \ : \ e \in \rho(e')\}|$$

- The *Dynamic (edge) Congestion* of the embedding is the maximum number of edges of G that ρ routes over a single edge of H *that must transmit signals in the same time step,* when H simulates a single communication step of G.

Both *Congestion* and *Dynamic Congestion* measure contention for communication links, which can be resolved either by increasing the bandwidth of the links, thereby incurring the penalty of increased hardware and increased area, or via queuing of messages, thereby incurring increased delay. In our present situation, wherein *Dynamic Congestion* will be bounded by $O(1)$, while *Congestion* will be bounded only by $O(\log \log N)$, one might opt to handle *Dynamic Congestion* via increased bandwidth and *Congestion* via message queuing.

The reader can verify the following simplifying inequalities.

Lemma 2.1. *For any embedding (α, ρ) of a graph G into a graph H whose maximum node-degree is d,*

$$\frac{Congestion(\alpha, \rho)}{Dilation(\alpha, \rho)} \leq Dynamic\ Congestion(\alpha, \rho) \leq Congestion(\alpha, \rho) < d^{Dilation(\alpha, \rho)}$$

Our primary interest here is algorithmic: we want to determine how efficiently one architecture H can simulate another architecture G on general computations. Recent, as-yet unpublished results of Leighton [17] demonstrate that the purely graph-theoretic notion of simulation outlined here captures the essence of the algorithmic problem, in the following formal sense.

Proposition 2.1. [17] *Say that one can embed the graph G in the graph H, with Dilation D and Congestion C. Then the architecture H can simulate T steps of the architecture G on a general computation in $O(C + D)T$ steps.*

We have not mentioned the *Expansion*-cost of our embeddings [22], i.e., the ratio $|V_H|/|V_G|$ (a measure of efficiency of resource utilization) even though one can sometimes decrease the *Dilation* of an embedding by increasing its *Expansion* [14]. We do not know if such a tradeoff can occur in the present setting, so we simplify our study by considering only minimum-*Expansion* embeddings; i.e., we always embed a given graph G in the smallest H that can hold it. Thus, our embeddings have almost unit *Expansion*, but often not exactly unit, since, for example, we often embed a graph with $m2^m$ nodes in a graph with 2^n nodes.

3. USEFUL BACKGROUND

We review a number of known properties of shuffle-oriented and butterfly-oriented interconnection networks, which are useful in our study.

3.1. Two Simplifying Equivalences

We have thus far used the term "shuffle-oriented networks" to refer ambiguously to the deBruijn and Shuffle-Exchange networks, and the term "butterfly-oriented networks" to refer ambiguously to the Butterfly and Cube-Connected-Cycles networks. We now justify this ambiguity by noting that, to within small constant factors, the deBruijn and Shuffle-Exchange networks are "equally efficient," as are the Butterfly and Cube-Connected-Cycles networks, in the following formal sense.

Proposition 3.1. *For all positive integers n:*
(a) *one can embed the order-n Shuffle-Exchange graph $S(n)$ in the order-n deBruijn graph $D(n)$ with Dilation 2 and Congestion 2;*
(b) *one can embed the order-n deBruijn graph $D(n)$ in the order-n Shuffle-Exchange graph $S(n)$ with Dilation 2 and Congestion 2.*

Proposition 3.2. *For all positive integers n:*
(a) *one can embed the order-n CCC graph $C(n)$ in the order-n Butterfly graph $B(n)$ with Dilation 2 and Congestion 2;*
(b) *one can embed the order-n Butterfly graph $B(n)$ in the order-n CCC graph $C(n)$ with Dilation 2 and Congestion 2.*

These results are part of folklore. We present a proof of Proposition 3.1 both because of the result's central importance to our study and to indicate how such a proof looks in our framework. The almost-identical proof of Proposition 3.2 is left to the reader.

Proof of Proposition 3.1. For the embeddings of both parts (a) and (b), we use the identity map for α, and we route each shuffle edge of $S(n)$ (resp., of $D(n)$) along the corresponding shuffle edge of $D(n)$ (resp., of $S(n)$). For part (a), we route each shuffle-exchange edge of $D(n)$ along the obvious corresponding (shuffle edge)-(exchange edge) path in $S(n)$. For part (b), we route each exchange edge of $S(n)$ along the obvious corresponding (shuffle-exchange edge)-(shuffle edge) path in $D(n)$. □

We simplify our study by restricting attention henceforth to whichever of the de-Bruijn and Shuffle-Exchange networks, on the one hand, and CCC and Butterfly networks, on the other hand, makes a particular proof simpler. We point out in places how constant factors would vary if we used $S(n)$ instead of $D(n)$ or vice versa.

Our next result is useful in our simulation of general computations on butterfly-oriented neworks.

It is well known that $\mathcal{D}(n)$ is *Hamiltonian* in that it contains $\mathcal{R}(2^n)$ as a subgraph. The following stronger property seems to have been lost in the mists of time.

Proposition 3.3. [27] *For all positive integers n, the order-n deBruijn graph $\mathcal{D}(n)$ is* pancyclic*; that is, for all integers $2 \leq k \leq 2^n$, the order-k ring graph $\mathcal{R}(k)$ is embeddable in $\mathcal{D}(n)$ with unit Dilation, i.e., as a subgraph.*

Proposition 3.3 is true also for "base-d deBruijn graphs," whose nodes are strings over Z_d [18].

Since the *diameter* (maximum inter-node distance) of a graph H bounds above the *Dilation* of any embedding into H, the following known result places the costs of our simulations in perspective.

Proposition 3.4. *For all positive integers n:*

GRAPH	SIZE	DIAMETER
$\mathcal{Q}(n)$	2^n	n
$\mathcal{D}(n)$	2^n	n
$\mathcal{S}(n)$	2^n	$2n-1$
$\mathcal{B}(n)$	$n2^n$	$2n-2$
$\mathcal{C}(n)$	$n2^n$	$3n-4$

4. SIMULATING THE BUTTERFLY

4.1. Simulating Ascend-Descend Algorithms

We slightly generalize the definition of ascend-descend algorithm from [21], to the following implicit form. An *order-n ascend-descend algorithm* operates on a 2^n-element sequence $\langle d_1, d_2, \cdots, d_{2^n} \rangle$ of data. At each time step t, there is an integer $k(t) \in Z_n$ such that the algorithm operates at that step on all 2^{n-1} pairs of data of the form $\langle d_{a2^{k(t)+1}+b}, d_{a2^{k(t)+1}+b+2^{k(t)}} \rangle$, where $a \in Z_{2^{n-k(t)-1}}$ and $b \in Z_{2^{k(t)}}$; for each t, $k(t+1) \in \{k(t)-1, k(t)+1\}$. See [21] for examples.

We claim that shuffle-oriented architectures can execute ascend-descend algorithms as fast as Hypercubes or butterfly-oriented architectures can. A result that is weaker than ours in two respects has been known for years: The weaker result uses a restricted

definition of the class of algorithms, which forces the integer $k(t)$ to proceed monotonically either from 0 to $n-1$ or from $n-1$ to 0; it further has the shuffle-oriented network use a computational strategy that incurs a slowdown of a factor of 2 whereas our simulation incurs no slowdown. An algorithmic proof of the more limited result appears in [21] and [26]. An algebraic proof of our more general result appears in [3]. We present here the first graph-theoretic development of the general result.

Proposition 4.1. [3] *For all positive integers n, the order-n Shuffle-Exchange architecture $\mathcal{S}(n)$ can execute any order-n ascend-descend algorithm just as quickly as can the n-dimensional Hypercube $\mathcal{Q}(n)$ or the order-n CCC architecture $\mathcal{C}(n)$.*

Proof Sketch of Proposition 4.1. Details fleshing out this sketch should be inferrable from the proof of Lemma 4.1 in Section 4.2.

It is clear that, given any T-step order-n ascend-descend algorithm, one can devise a T-level analog G_T of the CCC architecture that will execute the algorithm in precisely T steps. Specifically, G_T will be built in levels that are connected serially (as are the levels of the CCC, but without the wraparound of the latter); each level ℓ of G_T will be a copy of level $k(\ell)$ of $\mathcal{C}(n)$: The level edges at this level of $\mathcal{C}(n)$ connect precisely those pairs of data that are combined at this step of the algorithm.

Now take any two consecutive levels, ℓ and $\ell+1$, of G_T, and label their nodes with strings from Z_2^n, via the following inductive procedure.

1. Label an arbitrary unlabelled level-ℓ node with any string from Z_2^n that has not yet been used as a label at level ℓ. (Initially, one has total choice.)
2. Repeat the following steps until no new nodes get labelled.
   (a) For each unlabelled node v that is connected to a node u having label $L(u)$ by a *straight-edge*, label v with the *shuffle* of $L(u)$.
   (b) For each unlabelled node v that is connected to a node u having label $L(u)$ by a *level edge*, label v with the *exchange* of $L(u)$.
3. If some nodes remain unlabelled, repeat Step 1.

One can now verify that each string from Z_2^n appears as the label of precisely one node at each level of G_T.

Now, form the 2^n-node graph $\mathcal{G}$ by identifying like-labelled nodes from the two levels of G_T and connecting two nodes of $\mathcal{G}$ whose preimages were connected in G_T, removing any self-loops that were created by the identification. One verifies easily from our labelling procedure that

Claim. *The graph $\mathcal{G}$ is isomorphic to the order-n Shuffle-Exchange graph $S(n)$.*

It follows from this Claim that $S(n)$ can simulate each step of the ascend-descend algorithm in a single step, whence the result. □-Proposition 4.1

A factor of 2 slowdown occurs if we substitute either $D(n)$ for $S(n)$ or $B(n)$ for $C(n)$ in Proposition 4.1: The butterfly structure of $D(n)$ and $B(n)$ (manifest in the shuffle-exchange edges of the former and the cross-edges of the latter) necessitates two steps for mimicking the traversal of a single dimension in the Hypercube.

4.2. Simulating General Computations

While proving our main simulation result, we set off a series of fundamental lemmas that are likely to be useful also in other contexts.

To simplify notation in the sequel, let[11] $\Lambda(n) =_{\text{def}} \lceil \log n \rceil$.

Theorem. *For all positive integers n, one can embed the order-n Butterfly graph $B(n)$ in the order-$(n+\Lambda(n))$ deBruijn graph $D(n+\Lambda(n))$, with Dilation $O(\Lambda(n))$, Congestion $O(\Lambda(n))$, and Dynamic Congestion $O(1)$.*

Proof. We develop our embedding in three steps.

1. We embed $B(n)$ in the product graph[12] $\mathcal{R}(n) \times D(n)$, with *Dilation* 2 (Lemma 4.1).

2. We embed $\mathcal{R}(n) \times D(n)$ in $D(\Lambda(n)) \times D(n)$ with unit *Dilation*, i.e., as a subgraph (Lemma 4.2).

3. We embed $D(\Lambda(n)) \times D(n)$ in $D(n + \Lambda(n))$, with *Dilation* $2\Lambda(n) + 1$, via an embedding that has small *Dynamic Congestion* (Lemma 4.3).

In the course of the third embedding, we prove the amusing fact that n disjoint copies of $D(n)$ can be embedded simultaneously in $D(n + \Lambda(n))$, with *Dilation* $2\Lambda(n) + 1$.

Lemma 4.1. *For all positive integers n, one can embed the order-n Butterfly graph $B(n)$ in $\mathcal{R}(n) \times D(n)$ with Dilation 2, Congestion 2, and Dynamic Congestion 2.*

[11] All logarithms are to the base 2.

[12] Given graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the *product graph* $G \times H$ has node-set $V_G \times V_H$. Let u and v be nodes of G, and let x and y be nodes of H. Then $(\langle u, x\rangle, \langle v, x\rangle)$ is an edge of $G \times H$ just when (u, v) is an edge of G; and $(\langle u, x\rangle, \langle u, y\rangle)$ is an edge of $G \times H$ just when (x, y) is an edge of H.

Proof of Lemma 4.1. We begin by labelling the nodes of $\mathcal{B}(n)$ with strings from Z_2^n. Our *shuffle-oriented* labelling, which shares motivation with the labelling in Proposition 4.1, is implicit in [3], where n copies of $\mathcal{D}(n)$ are found lurking in $\mathcal{B}(n)$; see Fig. 5. The labelling rules are:

1. Label node $\langle 0, \vec{0} \rangle$ of $\mathcal{B}(n)$ with the string $\vec{0}$.
2. If level-ℓ node v ($\ell \in Z_n$) is labelled with string $L(v)$, then label
   - the *straight-edge* neighbor of node v on level $\ell + 1 \pmod n$ with the *shuffle* of $L(v)$.
   - the *cross-edge* neighbor of node v on level $\ell + 1 \pmod n$ with the *shuffle-exchange* of $L(v)$.

Claim. *Each level-ℓ of $\mathcal{B}(n)$ is assigned a unique label.*

Verification. There is a unique "downward" path in $\mathcal{B}(n)$ from node $\langle 0, \vec{0} \rangle$ to each node on level $n-1$, i.e., a path that increases level-number at each step: to wit, every level-0 node of $\mathcal{B}(n)$ is the root of a complete binary tree whose leaves are all of the level-$(n-1)$ nodes. Moreover, the unique path from node $\langle 0, \vec{0} \rangle$ to each level-$(n-1)$ node is recorded in the label of the node: the k^{th} edge of the path is a straight-edge (resp., a cross-edge) just when the k^{th} symbol of the label is a 0 (resp., a 1). It follows that the labels of all level-$(n-1)$ nodes are distinct. But, the same must now be true of every level of $\mathcal{B}(n)$ since, according to rule 2, the label of an arbitrary node u at level ℓ is just the $(n-\ell-1)$-fold shuffle of the label of the level-$(n-1)$ node reached by following $n-\ell-1$ straight-edges from u. $\square$-Claim

Now, isolate any two consecutive levels of the labelled $\mathcal{B}(n)$, together with the 2^{n+1} edges that connect the levels; cf. Fig. 6. Produce a 2^n-node graph G_n from the isolated levels by identifying like-labelled nodes and removing the self-loops from the nodes labelled $\vec{0}$ and $\vec{1}$. Our remarks about how labels of adjacent nodes are either the shuffle or the shuffle-exchange of each other renders the following claim transparent.

Claim. *For all choices of adjacent levels of $\mathcal{B}(n)$, the graph G_n is isomorphic to $\mathcal{D}(n)$.*

The Lemma is now direct: To embed $\mathcal{B}(n)$ in $\mathcal{R}(n) \times \mathcal{D}(n)$:

- Label the nodes of $\mathcal{B}(n)$ as indicated. Assign the level-ℓ node of $\mathcal{B}(n)$ that gets label x to node x of copy ℓ of $\mathcal{D}(n)$.
- Consider a straight-edge (resp., a cross-edge) e of $\mathcal{B}(n)$ that connects the node labelled x on level ℓ with the node labelled y on level $\ell + 1 \pmod n$. Route edge e within $\mathcal{R}(n) \times \mathcal{D}(n)$ via the length-2 path consisting of

- the shuffle edge (resp., the shuffle-exchange edge) that connects node x and node y in copy ℓ of $\mathcal{D}(n)$,
- the edge that connects the instances of node y in copies ℓ and $\ell+1 \pmod n$ of $\mathcal{D}(n)$.

The validity and efficiency of the embedding should be obvious. □-Lemma 2

We state and prove Lemmas 4.2 and 4.3 in a somewhat more general context than is needed for the Theorem.

Lemma 4.2. *For all positive integers m and n, one can embed $\mathcal{R}(m) \times \mathcal{D}(n)$ in $\mathcal{D}(\Lambda(m)) \times \mathcal{D}(n)$ with unit Dilation, i.e., as a subgraph.*

Proof of Lemma 4.2. By Proposition 3.3, $\mathcal{D}(\Lambda(m))$ is pancyclic, hence contains $\mathcal{R}(m)$ as a subgraph. The Lemma follows. □-Lemma 4.2

Lemma 4.3. *For all positive integers m and n, one can embed $\mathcal{D}(m) \times \mathcal{D}(n)$ in the order-$(m+n)$ deBruijn graph $\mathcal{D}(m+n)$, with Dilation $2m+1$, Congestion $8m+2$, and Dynamic Congestion 4.*

Proof of Lemma 4.3. We build on the following amusing fact.

Lemma 4.4. *For all positive integers m and n, one can embed 2^m node-disjoint copies of $\mathcal{D}(n)$ in $\mathcal{D}(m+n)$ with Dilation $2m+1$, and Congestion at most $4m+2$.*

Proof of Lemma 4.4. Let us be given 2^m copies of $\mathcal{D}(n)$. Label each with a distinct string $x \in Z_2^m$, so we can talk about copy x of $\mathcal{D}(n)$, which we denote $\mathcal{D}_x$.

For each x, we embed $\mathcal{D}_x$ in $\mathcal{D}(m+n)$ as follows. We assign node v of $\mathcal{D}_x$ to node xv of $\mathcal{D}(m+n)$. We route edge $(\beta v, v\gamma)$ of $\mathcal{D}_x$ ($\beta, \gamma \in Z_2$; $\gamma \in \{\beta, \beta \oplus 1\}$; $v \in Z_2^{n-1}$) via the following length-$(2m+1)$ path in $\mathcal{D}(m+n)$ that connects nodes $x\beta v$ and $xv\gamma$: Let $x = \xi_1\xi_2\cdots\xi_m$.

$$
\begin{array}{rcll}
x\beta v & = & \xi_1\xi_2\xi_3\cdots\xi_m\beta v & \\
 & \leftrightarrow & \xi_2\xi_3\cdots\xi_m\beta v\gamma & \text{shuffle/shuffle-exchange} \\
 & \leftrightarrow & \xi_3\cdots\xi_m\beta v\gamma\xi_1 & \text{shuffle/shuffle-exchange} \\
 & \cdots & & \\
 & \leftrightarrow & v\gamma\xi_1\xi_2\xi_3\cdots\xi_{m-1}\xi_m & \text{shuffle/shuffle-exchange} \\
 & \leftrightarrow & \xi_m v\gamma\xi_1\xi_2\xi_3\cdots\xi_{m-1} & \text{shuffle} \\
 & \cdots & & \\
 & \leftrightarrow & \xi_2\xi_3\cdots\xi_m v\gamma\xi_1 & \text{shuffle} \\
 & \leftrightarrow & \xi_1\xi_2\xi_3\cdots\xi_m v\gamma & \text{shuffle} \\
 & = & xv\gamma. &
\end{array}
$$

In the first $m+1$ of these edges, the choice between the shuffle and the shuffle-exchange edges of $\mathcal{D}(m+n)$ is dictated by whether the symbol being rotated to the end of the string needs to be complemented (calling for the shuffle-exchange edge) or not (so the shuffle edge is needed).

The claimed *Dilation* of the embedding is clear from our description of the routing procedure. Note that some of the routing paths are longer than necessary. (For instance, a length-1 path will suffice when x, β, and v are all 0's, and $\gamma = 1$.) This extra length does not affect *Dilation*, which is a worst-case measure; and, it will allow us to *decrease* the *Dynamic Congestion* of the final embedding that builds on this one.

To see that the *Congestion* of the embedding is at most $4m+2$, say that we are told that an edge of $\mathcal{D}(m+n)$ is being used as the k^{th} edge in a routing path. Then: if $k > m+1$, that information completely specifies x, v, and γ; if $k \leq m+1$, that information completely specifies v, β, and γ, and all but one bit of x. It follows that each edge of $\mathcal{D}(m+n)$ can be used to route at most two edges in each of the $2m+1$ steps of the routing. The bound follows.

We remark in passing that the *Dilation* of the embedding increases to $3m+2$ when one substitutes the Shuffle-Exchange graph for the deBruijn graph. □-Lemma 4.4

Return to Proof of Lemma 4.3. We now show how to extend the embedding of Lemma 4.4 to yield the desired embedding of $\mathcal{D}(m) \times \mathcal{D}(n)$ in $\mathcal{D}(m+n)$.

The assignment portion of the embedding is straightforward: We assign node $\langle x, v \rangle$ of $\mathcal{D}(m) \times \mathcal{D}(n)$ ($x \in Z_2^m$, $v \in Z_2^n$) to node xv of $\mathcal{D}(m+n)$. Thus, the subgraph $\{x\} \times \mathcal{D}(n)$ of $\mathcal{D}(m) \times \mathcal{D}(n)$ plays the role of $\mathcal{D}_x$. This assignment allows us to invoke the embedding of Lemma 4.4 to route the edges of $\mathcal{D}(m) \times \mathcal{D}(n)$ that connect nodes in different copies of $\mathcal{D}(m)$, i.e., that derive from edges of $\mathcal{D}(n)$. To complete the embedding, we need, therefore, specify only how to route the edges of $\mathcal{D}(m) \times \mathcal{D}(n)$ that connect nodes in different copies of $\mathcal{D}(n)$, i.e., that derive from edges of $\mathcal{D}(m)$.

We accomplish this second routing task by a technique that is almost identical to the technique of Lemma 4.4. Every edge that we must route in this second task connects a node u in some copy, $\mathcal{D}_x$, of $\mathcal{D}(n)$ *to the corresponding node* u in another copy, $\mathcal{D}_y$, where x and y are adjacent nodes of $\mathcal{D}(m)$. Thus, the path that realizes the edge must connect two nodes of the form $\beta z v$ and $z \gamma v$ ($\beta, \gamma \in Z_2$; $\gamma \in \{\beta, \beta \oplus 1\}$; $z \in Z_2^{m-1}$;

$v \in Z_2^n$). We realize this path as follows. Let $z = \varsigma_1\varsigma_2\cdots\varsigma_{m-1}$.

$$
\begin{array}{rcll}
\beta z v & = & \beta\varsigma_1\varsigma_2\cdots\varsigma_{m-1}v & \\
 & \leftrightarrow & \varsigma_1\varsigma_2\cdots\varsigma_{m-1}v\varsigma_1 & \text{shuffle/shuffle-exchange} \\
 & \leftrightarrow & \varsigma_2\cdots\varsigma_{m-1}v\varsigma_1\varsigma_2 & \text{shuffle/shuffle-exchange} \\
 & \cdots & & \\
 & \leftrightarrow & \varsigma_{m-1}v\varsigma_1\varsigma_2\cdots\varsigma_{m-1} & \text{shuffle/shuffle-exchange} \\
 & \leftrightarrow & v\varsigma_1\varsigma_2\cdots\varsigma_{m-1}\gamma & \text{shuffle/shuffle-exchange} \\
 & \leftrightarrow & \gamma v\varsigma_1\varsigma_2\cdots\varsigma_{m-1} & \text{shuffle} \\
 & \leftrightarrow & \varsigma_{m-1}\gamma v\varsigma_1\varsigma_2\cdots\varsigma_{m-2} & \text{shuffle} \\
 & \cdots & & \\
 & \leftrightarrow & \varsigma_2\cdots\varsigma_{m-1}\gamma v\varsigma_1 & \text{shuffle} \\
 & \leftrightarrow & \varsigma_1\varsigma_2\cdots\varsigma_{m-1}\gamma v & \text{shuffle} \\
 & = & z\gamma v. &
\end{array}
$$

As in the embedding of Lemma 4.4, the choice between the shuffle and shuffle-exchange edges in the first m edges of the path is dictated by whether or not the symbol being rotated to the end of the string needs to be complemented.

It is clear that the presented mappings constitute an embedding of $\mathcal{D}(m) \cdot \mathcal{D}(n)$ in $\mathcal{D}(m+n)$, in that the assignment and routing functions are both one-to-one. It remains, therefore, only to assess the cost of the resulting embedding.

The *Dilation* of the embedding is the maximum of the *Dilations* of the two stages. The routing of edges *within* copies of $\mathcal{D}(n)$ (cf. Lemma 4.4) incurs *Dilation* $2m+1$; the just-presented routing of edges *between* copies of $\mathcal{D}(n)$ incurs *Dilation* $2m$.

The *Congestion* of the embedding is the sum of the *Congestions* of the two stages of the embedding. The routing of edges *within* copies of $\mathcal{D}(n)$, in Lemma 4.4, incurs *Congestion* $4m+2$. We show now that routing edges *between* copies of $\mathcal{D}(n)$ incurs *Congestion* $8m$. To wit, say that we are told that an edge of $\mathcal{D}(m+n)$ is being used as the k^{th} edge in a routing path. Using the notation in our routing algorithm above: that information completely specifies the strings v and z, but it generally leaves both β and γ unspecified. It follows that each edge of $\mathcal{D}(m+n)$ can be used to route at most 4 edges in each of the $2m$ steps of the routing. The bound follows.

To determine the *Dynamic Congestion* of the embedding, assume that when a single step of a parallel algorithm traverses edges of $\mathcal{D}(m) \times \mathcal{D}(n)$, the associated routing paths within $\mathcal{D}(m+n)$ are traversed *in lockstep*. Under this assumption, at each step of the simulation, we know that edge e of $\mathcal{D}(m+n)$ is being used as the k^{th} edge in a routing path, since the same is true of *all* edges of $\mathcal{D}(m+n)$ that are being used at that step. It follows that the "uncertainty" about which edge of $\mathcal{D}(m) \times \mathcal{D}(n)$ is being routed

across a given edge of $\mathcal{D}(m+n)$ at a given step – which quantity bounds the *Dynamic Congestion* of the embedding from above – is less than the *Congestion* of the embedding by the factor $2m$. □-Lemma 4.3

Return to Proof of the Theorem. By composing the embeddings of Lemmas 4.1, 4.2, and 4.3, we obtain an embedding of $\mathcal{B}(n)$ in $\mathcal{D}(n + \Lambda(n))$. By the analyses of the Lemmas, we see that the *Dilation* of the composite embedding $4\Lambda(n) + 2$, its *Congestion* is $\leq 24\Lambda(n) + 4$, and its *Dynamic Congestion* is $\leq 12 + o(1)$. □-Theorem

ACKNOWLEDGMENTS. It is a pleasure to express thanks to several people for helpful comments and suggestions: to my perennial collaborators, Sandeep Bhatt, Fan Chung, Lenny Heath, and Tom Leighton; to Avraham Lempel and Tuvi Etzion of the Technion; and to UMass graduate students Fred Annexstein, Marc Baumslag, and Seshaiyen Raghuram.
A portion of this research was done while visiting the Department of Computer Science, the Technion, Haifa, Israel; a portion was supported by NSF Grant DCI-87-96236.

5. REFERENCES

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman (1974): *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

2. S.B. Akers and B. Krishnamurthy (1986): A group-theoretic model for symmetric interconnection networks. *Intl. Conf. Parallel Processing,* 216-223.

3. F. Annexstein, M. Baumslag, A.L. Rosenberg (1987): Group-action graphs and parallel architectures. Tech. Rpt. 87-133, Univ. Massachusetts; submitted for publication.

4. B. Becker and H.U. Simon (1986): How robust is the n-cube? *27th IEEE Symp. on Foundations of Computing,* 283-291.

5. S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, A.L. Rosenberg (1988): Optimal simulations by Butterfly networks. *20th ACM Symp. on Theory of Computing,* 192-204; submitted for publication.

6. S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1988): Efficient embeddings of trees in hypercubes. Tech. Rpt., Univ. Massachusetts; submitted for publication. See also, Optimal simulations of tree machines. *27th IEEE Symp. on Foundations of Computer Science* (1986) 274-282.

7. R.M. Chamberlain (1988): Gray codes, Fast Fourier Transforms and hypercubes. *Parallel Computing 6,* 225-233.

8. M.Y. Chan (1988): The embeddings of grids into optimal hypercubes. Typescript, Univ. Hong Kong.

9. N.G. DeBruijn (1946): A combinatorial problem. *Proc. Akademe Van Weteschappen 49,* Part 2, 758-764.

10. R. Ginosar and D. Egozi (1987): Topological comparison of Perfect Shuffle and Hypercube. Typescript, The Technion.

11. D.S. Greenberg, L.S. Heath and A.L. Rosenberg (1988): Optimal embeddings of butterfly-like graphs in the Hypercube. Tech. Rpt., Univ. Massachusetts; submitted for publication.

12. J. Hastad, F.T. Leighton, M. Newman (1987): Reconfiguring a hypercube in the presence of faults. *19th ACM Symp. on Theory of Computing.*

13. I. Havel and P. Liebl (1973): Embedding the polytomic tree into the n-cube. *Časopsis pro Pěstování Matmatiky 98,* 307-314.

14. J.-W. Hong, K. Mehlhorn, A.L. Rosenberg (1983): Cost tradeoffs in graph embeddings. *J. ACM 30,* 709-728.

15. L. Johnsson (1985): Basic linear algebra computations on hypercube architectures. Tech. Rpt., Yale Univ.

16. D. Kleitman, F.T. Leighton, M. Lepley, G.L. Miller (1983): An asymptotically optimal layout for the shuffle-exchange graph. *J. Comp. Syst. Sci. 26,* 339-361.

17. F.T. Leighton (1988): Personal communication.

18. A. Lempel (1971): m-ary closed sequences. *J. Combin. Th.(A) 10,* 253-258.

19. D.S. Parker (1980): Notes on shuffle-exchange type switching networks. *IEEE Trans. Comp., C-29,* 213-222.

20. D.K. Pradhan and M.R. Samatham (1988): The deBruijn multiprocessor network: A versatile parallel processing and sorting network for VLSI. *IEEE Trans. Comp.,* to appear.

21. F.P. Preparata and J.E. Vuillemin (1981): The cube-connected cycles: a versatile graph for parallel computation. *C. ACM 24,* 300-309.

22. A.L. Rosenberg (1981): Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science: Proceedings of the International Workshop WG80,* Bad Honnef, Germany (H. Noltemeier, ed.) *Lecture Notes in Computer Science 100,* Springer-Verlag, New York 150-176.

23. Y. Saad and M.H. Schultz (1988): Topological properties of hypercubes. *IEEE Trans. Comp. 37,* 867-872.

24. C. Stanfill (1987): Communications architecture in the Connection Machine system. Tech. Rpt. HA87-3, Thinking Machines Corp.

25. H. Stone (1971): Parallel processing with the perfect shuffle. *IEEE Trans. Comp., C-20,* 153-161.

26. J.D. Ullman (1984): *Computational Aspects of VLSI.* Computer Science Press, Rockville, MD.

27. M. Yoeli (1962): Binary ring sequences. *Amer. Math. Monthly 69,* 852-855.

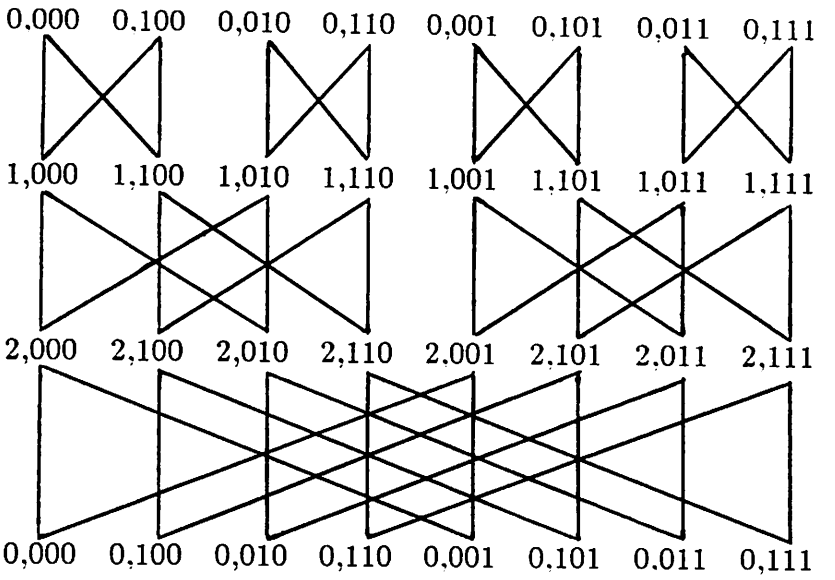

Figure 3: *The Butterfly graph $B(3)$ with, level 0 replicated to aid visualization.*

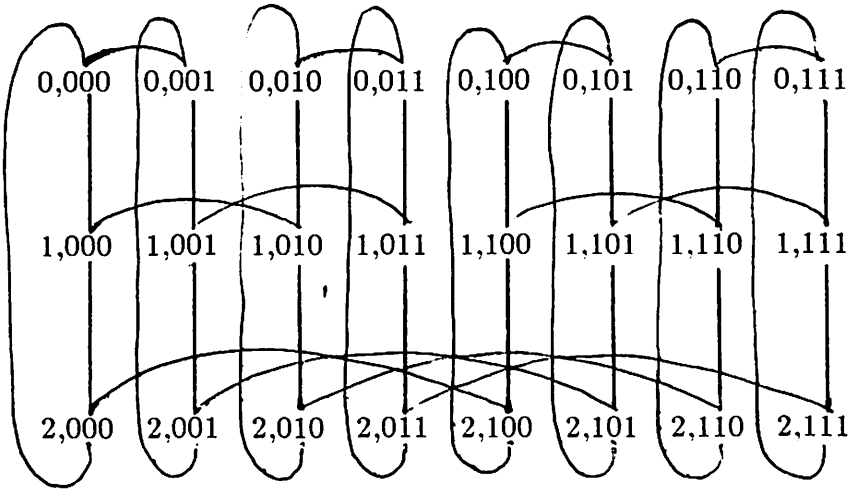

Figure 4: The Cube-Connected-Cycles graph $C(3)$.

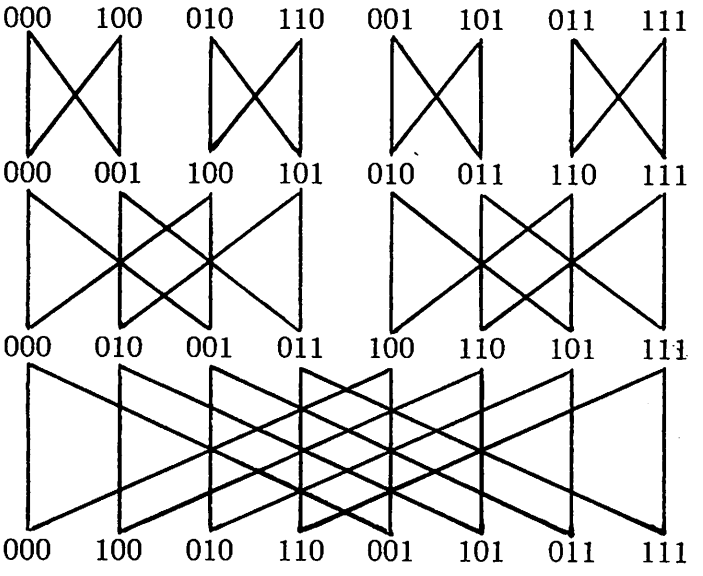

Figure 5: $\mathcal{B}(3)$ *(with level 0 replicated) with the shuffle-oriented labelling.*

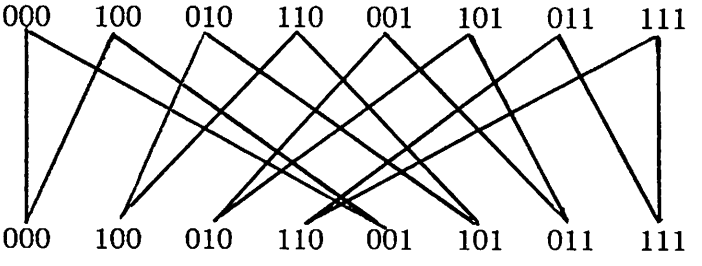

Figure 6: *Two levels of* $\mathcal{B}(3)$ *with the shuffle-oriented labelling, permuted to help visualize the identification.*