THE PRODUCT-SHUFFLE NETWORK: Toward Reconciling Shuffles and Butterflies (Preliminary Version)

Arnold L. Rosenberg

Computer and Information Science Department
University of Massachusetts

COINS Technical Report 88-102

THE PRODUCT-SHUFFLE NETWORK: Toward Reconciling Shuffles and Butterflies (Preliminary Version)

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

December 14, 1988

Abstract

We introduce a new bounded-degree interconnection network for parallel architectures, the *Product-Shuffle* (*PS*) network. PS networks can simulate both butterfly- and shuffle-oriented interconnection networks very efficiently and have more communication power than either of those families. In addition to their simulation power, PS networks share many of the most desirable structural features of both Butterfly and deBruijn networks — which are among the most efficient of the bounded-degree Hypercube-derivative networks. Among these desirable features are pancyclicity, optimal diameter, and large complete-binary-tree subgraphs. PS networks also contain a number of computationally valuable subgraphs which butterfly- and shuffle-oriented cannot even simulate efficiently, including moderate size meshes and meshes of trees. Finally, PS networks enjoy VLSI layouts that consume only modestly more area than the best layouts of like-sized butterfly- and shuffle-oriented networks.

Keywords: Interconnection networks; parallel architectures; simulation; Hypercube-derivative networks; Butterfly networks; Cube-Connected Cycles networks; Shuffle-Exchange networks; deBruijn networks

1. HYPERCUBE-DERIVATIVE NETWORKS

1.1. Goals of the Study

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, are among today's dominant interconnection networks for massively parallel architectures. Indeed, architectures based on these networks have been built in both industry and academia.

Among these interconnection networks, the Hypercube is the clear favorite because of its efficiency on a broad class of algorithms [3, 5, 7, 18] and its structural uniformity that simplifies programming [17]. The major shortcoming of the Hypercube is its having high node-degrees.[1] The technological 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 CCC networks [13]. These networks were constructed with a certain important genre of Hypercube-algorithm, called *ascend-descend* algorithms [13], 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 [1] yield the smaller, shuffle-oriented bounded-degree "approximations" of the Hypercube, most notably the Shuffle-Exchange[2] [19] and deBruijn [6, 12] networks. Shuffle-oriented networks have only as many nodes as does the Hypercube, yet they avoid its large node-degrees; and, on certain computational tasks (including ascend-descend algorithms) they afford one computational efficiency (roughly) equal to that of the Butterfly and CCC.

Butterfly- and shuffle-oriented networks are roughly equivalent approximations of the Hypercube on a broad class of computational tasks, but it is not clear whether or not one of these network families majorizes the other on *general* computations. Confusingly enough, there is evidence that butterfly- and shuffle-oriented networks have incomparable strengths and weaknesses, and there is countervailing evidence that the two classes of networks are equivalent in power. On the one hand, deBruijn networks have the computationally useful properties of being *pancyclic*, i.e., of containing cy-

[1] Each node of the N-node Hypercube has degree (= number of neighbors) $\log_2 N$.

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

cles of all possible lengths [21], of having optimal diameters,[3] and of containing large complete binary trees as subgraphs; none of these facts is true of butterfly-oriented networks. On the other hand, the Butterfly network enjoys both node-transitivity and a recursive decomposition structure, which are not shared by shuffle-oriented networks; such symmetry and decomposability are quite useful in developing efficient algorithms for Butterfly networks. For instance, the efficient routing and sorting algorithms for the Butterfly network in [14] and [15], respectively, exploit the symmetry and recursive structure of the Butterfly, hence are not easily transported to any shuffle-oriented networks. However, none of the apparent algorithmic distinctions between the two families of networks is definitive: Recent work in [8] yields an algorithm for sorting on the deBruijn network that rivals the efficiency of the Reif-Valiant [15] algorithm for Butterflies. Even more, [8] presents techniques for efficiently simulating either of the Butterfly or deBruijn network on the other *in a work-preserving manner*. Counterbalancing the just-mentioned efficient simulations is recent work of the author [16], which uses a more demanding notion of simulation than does [8],[4] and which suggests that shuffle-oriented networks have strictly more communication power than do butterfly-oriented networks; this suggestion resides in the fact that we now know of simulations of butterfly-oriented networks by deBruijn networks that are *exponentially more efficient* than the best known simulations of deBruijn networks by butterfly-oriented networks. (Of course, this is, as yet, only circumstantial evidence.)

The present study is motivated in part by the unresolved questions implicit in the preceding paragraph: Which, if either, of the butterfly- and shuffle-oriented networks is the more powerful? Under what circumstances are the work-preserving simulations of [8] to be preferred to the more structured simulations of [16] (which yield algorithms that translate programs for the simulated class of networks to equivalent programs for the simulating class of networks) and vice versa? Further motivating our study is the fact that the constructions in both [8] and [16] suggest that *efficient* simulations of either of these network families by the other are likely to be rather sophisticated and complicated. There would be value, therefore, in having a genre of network which, while retaining most of the structural simplicity of both shuffle-oriented and butterfly-oriented networks, could be viewed as the "least upper bound" of the two families, in the sense of being able to simulate both families *simply and efficiently*. The (bounded-degree) *Product-Shuffle* (*PS*) network introduced in this paper is a candidate such network. We prove the following results about PS networks:

- *PS networks can simulate both butterfly- and shuffle-oriented networks of equal*

[3] The N-node deBruijn and Hypercube networks both have diameter exactly $\log_2 N$.

[4] We use the same notion of simulation here as in [16].

size, with only a factor-of-2 slowdown. This is considerably more efficent than the simulations in [8] and [16].

- Within our framework of highly structured simulations, *PS networks are strictly more powerful than either butterfly- or shuffle-oriented networks, by more than any constant factor.*

- PS networks inherit the three computationally most valuable structural properties of deBruijn networks, namely, *optimal diameter*, *pancyclicity*, and *containing large complete binary trees as subgraphs.*

- PS networks *contain moderate size mesh and mesh-of-trees networks as subgraphs*, a property shared by neither butterfly- nor shuffle-oriented network [2].

- PS networks *admit VLSI layouts that consume only modestly more area than the best layouts of like-sized butterfly- or shuffle-oriented networks.*

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 CCC networks. This ambiguity is justified by the fact that the deBruijn and Shuffle-Exchange networks can each simulate the other with only a factor-of-2 slowdown, and the same is true of the Butterfly and CCC networks. We focus on the deBruijn and Butterfly networks in the sections that follow, since they yield smaller constant factors in our simulations.

2. THE FORMAL FRAMEWORK

2.1. Interconnection Networks as Graphs

The most basic issue is how to view parallel architectures — really, arrays of identical processing elements (PEs) — and their underlying interconnection networks as undirected graphs:[5] 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.**

A. Notation and Terminology

[5] 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*.

- In the phrase "for all n," n always ranges over the positive integers. For all n, $Z_n =_{\text{def}} \{0, 1, \cdots, n-1\}$, and $\lambda_n =_{\text{def}} \lceil \log_2 n \rceil$.
- 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.
- For any string x, $|x|$ denotes the length of x.
- Given graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the *(direct) 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.
- The *degree* of node v of graph G is the number of edges of G incident to (i.e., involving) v. The *valence* of G is the maximum degree of any of its nodes.

B. The Graphs of Interest

Let n be a positive integer.

- The *order-n* **deBruijn graph** $\mathcal{D}(n)$ has node-set Z_2^n. $\mathcal{D}(n)$ has two types of edges: Node βx of $\mathcal{D}(n)$ ($\beta \in Z_2$ and $x \in Z_2^{n-1}$) is connected by a *shuffle* edge to node $x\beta$, and by a *shuffle-exchange* edge to node $x(1-\beta)$. See Fig. 1.
- 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 $\langle \ell,\ x\rangle \in V_n$. 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

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

  is connected by a *straight-edge* with node

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

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

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

  See Fig. 2.
- The *order-n* **Product-Shuffle graph (PS graph,** for short) $\mathcal{P}(n)$ is the product graph $\mathcal{D}(\lambda_n) \times \mathcal{D}(n)$.

In addition to the foregoing graphs, which are the focus of our study, the following graphs will also be of interest, since they all appear (at some size) as subgraphs of PS graphs (cf. Section 3.2).

- the n-node **cycle** $\mathcal{C}(n)$, whose nodes comprise the set Z_n and whose edges connect each node v with node $v + 1 \pmod n$
- the $m \times n$ **(toroidal) mesh** $\mathcal{M}(m, n)$, which is (isomorphic to) the product graph $\mathcal{C}(m) \times \mathcal{C}(n)$
- the height-n **complete binary tree** $\mathcal{T}(n)$, whose $2^n - 1$ nodes comprise the set $\bigcup_{k=0}^{n-1} Z_2^k$ of binary strings of length $\leq n$ and whose edges connect each node x of length $< n - 1$ with nodes $x0$ and $x1$
- the $m \times n$ **mesh of trees** $\mathcal{MT}(m, n)$ (m and n being powers of 2) [10] which is obtained from the $m \times n$ mesh by
  - eliminating all mesh edges
  - erecting a copy of the complete binary tree $\mathcal{T}(\log_2 m)$ along each column, using the column vertices as leaves of the tree
  - erecting a copy of the complete binary tree $\mathcal{T}(\log_2 n)$ along each row, using the row vertices as leaves of the tree

2.2. Simulation via Graph Embeddings

The next basic issue is how to represent the simulation of one array A by another array B. We assume that the PEs of the simulating array B are sufficiently numerous and powerful to simulate the PEs of the simulated array A step for step — so no delay is incurred because of computational steps. We restrict attention to simulations that honor the pulsed computation regimen of array A: Array B alternates (single) steps that simulate one *computation* step of array A, with (possibly multiple) steps that simulate one *communication* step of array A. The slowdown incurred by a simulation arises from having to simulate on the interconnection network underlying array B, communication steps that are tailored to the (possibly very different) structure of the interconnection network underlying array A. 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:[6]

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

Fundamental to our representing simulations by graph embeddings is our assessing the *delay* incurred by a simulation. We use two measures for this purpose. (Other meaningful measures exist but are not relevant here; cf. [3, 8.16].) 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:

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

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

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

The contention for communication links measured by *Congestion* can be resolved either by increasing the bandwidth of the links, at the cost of increased hardware and increased area, or via queuing of messages, at the cost of increased delay. The simulations presented here will have *Congestion* bounded by the constant 2, so it would be reasonable to double bandwidth in order to avoid contention.

The following simple inequality places our measures in perspective.

Proposition 1 *For any embedding (α, ρ) of a graph G into a graph H whose maximum node-degree is d,*

$$\mathit{Congestion}(\alpha, \rho) \leq d^{\mathit{Dilation}(\alpha,\rho)}$$

[6] 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 "$(v_i, v_{i+1}) \in \pi$".

Our primary interest here is algorithmic — we want to determine how efficiently one architecture H can simulate another architecture G on general computations — but our formal setting is purely graph-theoretic. The simulations we present here are so simple and efficient that their computational implications are immediate. In general, though, one would invoke (some analogue of) the following result to argue that the graph-theoretic notion of simulation outlined here captures the essence of the algorithmic problem.

Proposition 2 [11] *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.*

Finally, we can formalize our discussion at the end of the Introduction concerning the "equivalence" of the deBruijn and Shuffle-Exchange networks, on the one hand, and the Butterfly and CCC networks, on the other hand.

Proposition 3 *For all n:*
(a) *One can embed either of the order-n Shuffle-Exchange graph and the order-n de-Bruijn graph $D(n)$ in the other, with Dilation 2 and Congestion 2.*
(b) *One can embed either of the order-n CCC graph and the order-n Butterfly graph $B(n)$ in the other, with Dilation 2 and Congestion 2.*

3. STRUCTURAL PROPERTIES OF $P(n)$

3.1. Message-Passing Power

The *diameter* (maximum inter-node distance) of a graph H bounds above both the *Dilation* of any embedding into H and the time required for any single-node broadcast in H. Therefore, the following table places our simulation results in perspective and provides an interesting comparison of $P(n)$ with its "competitors." One noteworthy point is that PS graphs share diameter $\log_2 N$ with deBruijn graphs and Hypercubes.

Proposition 4 *For all n:*

	GRAPH	SIZE	VALENCE	DIAMETER
(a)	$D(n)$	$N = 2^n$	4	$\log N$
(b)	$B(n)$	$N = n2^n$	4	$2\log N \quad 2\log\log N$
(c)	$P(n)$	$N = 2^{n+\lambda_n}$	8	$\log N$

Proof Sketch. Parts (a,b) being well known, we concentrate on part (c). One can proceed from any node $\langle x, y\rangle$ of $\mathcal{P}(n)$ to any other node $\langle x', y'\rangle$ by

1. proceeding from node $\langle x, y\rangle$ to node $\langle x', y\rangle$ in at most $|x| = \lambda_n$ steps by mimicking the way one would proceed from node x to node x' in $\mathcal{D}(\lambda_n)$;
2. proceeding from node $\langle x', y\rangle$ to node $\langle x', y'\rangle$ in at most $|y| = n$ steps by mimicking the way one would proceed from node y to node y' in $\mathcal{D}(n)$. □

3.2. Computationally Important Subgraphs of $\mathcal{P}(n)$

We have just begun to investigate the simulation power of PS graphs, although their ability to simulate both Butterfly graphs and deBruijn graphs efficiently endows them automatically with considerable such power. However, even more dramatic is the fact that PS graphs contain a large number of computationally useful graphs *as subgraphs*, i.e., as graphs that can be simulated with no slowdown. Delimiting this collection of *perfectly* simulated graphs is still not complete, but the list is already impressive.

A. Cycles

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

Lemma 1 [21] *For all n, the order-n deBruijn graph $\mathcal{D}(n)$ is* pancyclic*; that is, for all $2 \leq k \leq 2^n$, the k-node cycle $\mathcal{C}(k)$ is a subgraph of $\mathcal{D}(n)$.*

PS graphs share this property, whose computational benefits are exploited in the simulations in [16] and in our Section 4.

Theorem 1 *For all n, the order-n PS graph $\mathcal{P}(n)$ is pancyclic.*

Proof Sketch. The case $n = 1$ being trivial, let $n \geq 2$ be any integer, and let $m \geq 1$ be any integer in $Z_{2^n+\lambda_n}$. We show that there is a cycle of length $m + 1$ in $\mathcal{P}(n)$.

Step 1. The integer $m + 1$ admits a unique representation in the form

$$m + 1 = c2^n + d$$

with $c < 2^{\lambda_n}$ and $d \leq 2^n$. By Lemma 1, we can find a cycle C of length $c + 1$ in $\mathcal{D}(\lambda_n)$, as well as a cycle D of length d in $\mathcal{D}(n)$. These cycles will be the "basis" for our length-$(m + 1)$ cycle in $\mathcal{P}(n)$.

Step 2. Use the cycle C in the natural way to select and order $c+1$ of the 2^{λ_n} copies of $D(n)$ that comprise $P(n)$; call the selected copies $D_0, D_1, \cdots, D_c$. Let x and y be nodes of $D(n)$ that are adjacent in the cycle D. For $i \in Z_{c+1}$, let x_i and y_i be the copies of the nodes x and y in copy D_i of $D(n)$.

Step 3. For $i \in \{1, 2, \cdots, c\}$, find a length-$2^n$ cycle (i.e., a Hamiltonian cycle) S_i in D_i in which the nodes x_i and y_i just defined are adjacent. The cycle S_i is guaranteed to exist by the following Claim, which follows from the fact that one can construct a Hamiltonian cycle in $D(n)$ from any Eulerian cycle in $D(n-1)$.

Claim. *For any pair of adjacent nodes r, s of $D(n)$, there is a length-2^n cycle in $D(n)$ in which nodes r and s are adjacent.*

Let u_i and v_i be two *other* nodes of D_i that are adjacent in the cycle S_i, chosen so that the cycle has the form

$$x_i, P_i, u_i, v_i, Q_i, y_i$$

where the P's and Q's are the intermediate paths that define the cycle.

Step 4. Say that c is even; simple modifications take care of odd c. A length-$(m+1)$ cycle in $P(n)$ can be described schematically as follows.

1. Trace the cycle D in D_0, from node x_0 to node y_0, leaving out the edge that connects the two nodes.
2. For $i = 1, \ldots, c/2$, cross to node y_{2i-1}, and follow path Q_{2i-1} to node v_{2i-1}; cross thence to node v_{2i}, and follow path Q_{2i} to node y_{2i}.
3. Cross the edge from node y_{2c} to node x_{2c}.
4. For $i = c/2, \ldots, 1$, proceed from node x_{2i} along path P_{2i} to node u_{2i}; cross thence to node u_{2i-1}, and follow path P_{2i-1} to node x_{2i-1}; cross thence to node x_{2i-2}.

The reader should be able to fill in remaining details, with the help of Fig. 3. □

B. Meshes

One corollary of the main result in [2] is that butterfly- and shuffle-oriented cannot simulate meshes with only constant slowdown (using our notion of simulation); it follows, of course, that they do not contain large meshes as subgraphs. In contrast, PS graphs contain moderate size meshes as subgraphs, as indicated in the following corollary of Theorem 1.

Corollary 1 *For all n, the order-n PS graph $P(n)$ contains the $n \times 2^n$ mesh $M(n, 2^n)$ as a subgraph.*

C. Complete Binary Trees

The complete binary tree is a very useful computational structure, most obviously for broadcasting, but also for simulations [2]. Thus, the following is one of the most useful properties of deBruijn graphs; cf. [12].

Lemma 2 *For all n, the order-n deBruijn graph $\mathcal{D}(n)$ contains the height-n complete binary tree $\mathcal{T}(n)$ as a subgraph, rooted at node $\vec{0}1$.*

While PS graphs cannot match the fact that the N-node deBruijn graph contains the $(N-1)$-node complete binary tree, they do come within a factor of 2 of matching it.

Theorem 2 *For all n, the order-n PS graph $\mathcal{P}(n)$ contains the height-$(n + \lambda_n - 1)$ complete binary tree $\mathcal{T}(n + \lambda_n - 1)$ as a subgraph.*

Proof Sketch. We find an instance of $\mathcal{T}(n + \lambda_n - 1)$ rooted at node $v_0 = \langle \vec{0}1, \vec{0}1 \rangle$ of $\mathcal{P}(n)$, as follows. We first invoke Lemma 2 to find a copy of $\mathcal{T}(n)$ in "copy $\vec{0}1$" of $\mathcal{D}(n)$, rooted at node v_0 and having leaves of the form $\langle \vec{0}1, x \rangle$ for some $x \in Z_2^n$. We then invoke Lemma 2 once for each "copy" of $\mathcal{D}(\lambda_n)$ (2^{n-1} times in all) to find a copy of $\mathcal{T}(\lambda_n)$ rooted at each of these leaves. □

To place Theorem 2 in perspective, the efficient embedding of complete binary trees in Butterfly graphs presented in [2] promises only constant (as oppose to unit) *Dilation* and utilizes only roughly one-eighth of the nodes of the host Butterfly.

D. Meshes of Trees

The mesh of trees network has been shown to be quite powerful computationally [10]. One can prove, using Lemma 2, that PS graphs contain at least moderate size such networks as subgraphs.

Proposition 5 *For all k, the order-2^k PS graph $\mathcal{P}(2^k)$ contains the $2^{k-1} \times 2^{2^k-1}$ mesh of trees $\mathcal{MT}(2^{k-1}, 2^{2^k-1})$ as a subgraph.*

Proof Sketch. Let $n = 2^k$. Let us denote by x_i, $1 \leq i \leq 2^{n-1}$, the i^{th} leaf of $\mathcal{T}(n)$ and by y_i, $1 \leq i \leq 2^{\lambda_n - 1}$, the i^{th} leaf of $\mathcal{T}(\lambda_n)$. By Lemma 2, $\mathcal{P}(n)$ contains the product graph $\mathcal{T}(\lambda_n) \times \mathcal{T}(n)$ as a subgraph. As a consequence, $\mathcal{P}(n)$ contains as a subgraph every tree of the form $\{y_i\} \times \mathcal{T}(n)$, as well as every tree of the form $\mathcal{T}(\lambda_n) \times \{x_i\}$. The union of all of these subgraphs is $\mathcal{MT}(2^{k-1}, 2^{2^k-1})$. □

3.3. An Area-Efficient VLSI Layout

In the previous subsections, we have shown that $\mathcal{P}(n)$ enjoys many of the computationally valuable structural properties of both $\mathcal{D}(n)$ and $\mathcal{B}(n)$. In Section 4, we shall prove that, in the sense of the current framework, $\mathcal{P}(n)$ is strictly more powerful than either $\mathcal{D}(n + \lambda_n)$ or $\mathcal{B}(n)$. Yet, surprisingly, $\mathcal{P}(n)$ admits a VLSI layout which is only modestly less area-efficient than the most efficient layout of either of the other two networks. The reader is referred to [4] for background on the formal framework and techniques of analysis for VLSI layouts.

Theorem 3 (a) *The N-node PS graph admits a VLSI layout of area*

$$O\left(\frac{N^2}{\log N \log\log N}\right).$$

(b) [20] *Any VLSI layout of the N-node Butterfly graph or the N-node deBruijn graph consumes area*

$$\Omega\left(\frac{N^2}{\log^2 N}\right).$$

Proof Sketch. Let us concentrate on $\mathcal{P}(n)$, so $N = 2^{n+\lambda_n}$. We use the layouts guaranteed by the following Lemma to obtain the desired layout of $\mathcal{P}(n)$.

Lemma 3 (a) [9] *The N-node deBruijn graph admits a VLSI layout of area*

$$O\left(\frac{N^2}{\log^2 N}\right).$$

(b) [4] *The N-node deBruijn graph admits a collinear VLSI layout of area*

$$O\left(\frac{N^2}{\log N}\right),$$

i.e., a layout in which all nodes are laid out in a line.

Stack 2^{λ_n} copies of the area-efficient layout of $\mathcal{D}(n)$ from Lemma 3(a), aligned so that, for each node v of $\mathcal{D}(n)$, all copies of v are lined up in the same vertical track. For each node v of $\mathcal{D}(n)$, allocate $2^{\lambda_n}/\lambda_n$ new vertical tracks, and use these tracks to route a collinear layout of $\mathcal{D}(\lambda_n)$ via the layout of Lemma 3(b), using the copies of v as nodes. Easily supplied details turn this schematic description into a layout of $\mathcal{P}(n)$, whose area satisfies the bound of Theorem 3(a). □

4. COMPARING $P(n)$, $D(n+\lambda_n)$, AND $B(n)$

4.1. $P(n)$ Simulating $D(n+\lambda_n)$ and $B(n)$

We now verify the most important property of PS graphs, namely, that it can efficiently simulate both butterfly- and shuffle-oriented graphs.

Theorem 4 *For all n, one can embed both the order-n Butterfly graph $B(n)$ and the order-$(n+\lambda_n)$ deBruijn graph $D(n+\lambda_n)$ in the order-n PS graph $P(n)$, with Dilation 2 and Congestion 2.*

It is interesting to compare the relative powers of Hypercubes and PS graphs by contrasting Theorem 4 with the currently known relationships between butterfly- and shuffle-oriented graphs, on the one hand, and Hypercubes, on the other hand:

- Every Butterfly and CCC graph *is a subgraph* of the smallest Hypercube that is big enough to hold it [7].
- Every known embedding of an N-node shuffle-oriented graph in a Hypercube has *Dilation* $\Omega(\log N)$ (which is as large as *Dilation* in the Hypercube can be).

Proof Sketch of Theorem 4. We consider first the (easier) problem of simulating deBruijn graphs on PS graphs. We prove a somewhat more general result than needed for the Theorem.

Lemma 4 *For all m and n, one can embed the order-$(m+n)$ deBruijn graph $D(m+n)$ in the product graph $D(m) \times D(n)$, with Dilation 2 and Congestion 2.*

Proof Sketch. Each node v of $D(m+n)$ is a length-$(m+n)$ binary string. Let $(v)_m$ denote the length-m prefix of this string, and let $[v]_n$ denote the length-n suffix of this string. The assignment function of the desired embedding is given by:

$$\alpha(v) = \langle (v)_m, [v]_n \rangle$$

for all $v \in Z_2^{m+n}$. The routing function ρ of the embedding realizes each edge of $D(m+n)$ of the form

$$(\beta x \gamma y, x \gamma y \delta)$$

$(\beta, \gamma, \delta \in Z_2;\ \delta \in \{\beta, 1-\beta\};\ x \in Z_2^{m-1}$ and $y \in Z_2^{n-1})$ via the following length-2 path in $\mathcal{D}(m) \times \mathcal{D}(n)$:

$$\begin{aligned} \alpha(\beta x \gamma y) &= \langle \beta x, \gamma y \rangle \\ &\hookrightarrow \langle x\gamma, \gamma y \rangle \\ &\hookrightarrow \langle x\gamma, y\delta \rangle \\ &= \alpha(x\eta y\varsigma). \end{aligned}$$

This embedding clearly has *Dilation* 2. The claimed *Congestion* follows from the facts that the first edge in the length-2 path identifies the edge of $\mathcal{D}(m+n)$ being simulated, up to the identity of δ, while the second edge identifies the edge being simulated, up to the identity of β. □

Now we turn to the simulation of Butterflies on PS graphs.

Lemma 5 [16] *For all n, one can embed the order-n Butterfly graph $\mathcal{B}(n)$ in $\mathcal{P}(n)$, with Dilation 2 and Congestion 2.*

Proof Sketch. We sketch the proof from [16]. By the pancyclicity of deBruijn graphs (Lemma 2), it suffices to embed $\mathcal{B}(n)$ in the product graph $\mathcal{C}(n) \times \mathcal{D}(n)$, with *Dilation* 2 and *Congestion* 2.

We label the nodes of $\mathcal{B}(n)$ with strings from Z_2^n via the following inductive procedure that is implicit in [1]; cf. Fig. 4.

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 (resp., the cross-edge) neighbor of node v on level $\ell + 1 \pmod n$ with the *shuffle* (resp., the *shuffle-exchange*) of $L(v)$.

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. 5. Produce the 2^n-node graph G_n from the isolated levels by identifying like-labelled nodes and eliminating self-loops. Our labelling procedure guarantees that:

Claim. *For any two consecutive 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 $\mathcal{C}(n) \times \mathcal{D}(n)$:

- Label the nodes of $\mathcal{B}(n)$ using the indicated procedure. Letting $L(v)$ denote the label assigned to node v, assign level-ℓ node v of $\mathcal{B}(n)$ to node $L(v)$ of copy ℓ of $\mathcal{D}(n)$.

- Route edge $e = (\langle \ell, x \rangle, \langle \ell', y \rangle)$ of $\mathcal{B}(n)$ within $\mathcal{C}(n) \times \mathcal{D}(n)$ via the length-2 path:

$$\begin{aligned} \langle \ell, L(\langle \ell, x \rangle) \rangle &\rightarrow \langle \ell, L(\langle \ell', y \rangle) \rangle \\ &\rightarrow \langle \ell', L(\langle \ell', y \rangle) \rangle \end{aligned}$$

  Thus, we first route within a copy of $\mathcal{D}(n)$ and then between copies.

Our embedding clearly has *Dilation* 2. The claimed *Congestion* is incurred since nodes in both $\mathcal{D}(n)$ and $\mathcal{B}(n)$ have two "successors" and two "predecessors." □

4.2. The Converse Simulations

In the framework of our strong notion of simulation, PS graphs are strictly more powerful than either Butterfly or deBruijn graphs, in the sense of the following result. Note how much stronger the result is for Butterfly graphs than for deBruijn graphs, both in terms of quantification and *Dilation*.

Theorem 5 (a) *For all n, any embedding of the order-n PS graph $\mathcal{P}(n)$ in any order-m Butterfly graph $\mathcal{B}(m)$, $m \geq n$, must have Dilation $\Omega(\log n)$.*
(b) *For all n, any embedding of the order-n PS graph $\mathcal{P}(n)$ in the order-$(n + \lambda_n)$ deBruijn graph $\mathcal{D}(n + \lambda_n)$ must have Dilation $\Omega(\log \log n)$.*

Proof Sketch. By Corollary 1, $\mathcal{P}(n)$ contains the $n \times n$ mesh as a subgraph. It is proved in [2] that any embedding of that mesh in *any* Butterfly graph must have Dilation $\Omega(\log n)$ and that any embedding of that mesh in a like-sized deBruijn graph must have Dilation $\Omega(\log \log n)$. □

The lower bounds of Theorem 5 grow faster than any constant, thus justifying our assertion about the power of PS graphs; however, each of these lower bounds is *exponentially smaller* than the best known analogous upper bound; cf. [16]. We do not know at this point whether to believe that the upper bounds can be lowered or that the lower bounds can be raised. Moreover, in common with the lower bounds of Theorem 5, the best known upper bounds for embedding PS graphs in deBruijn graphs are *exponentially smaller* than the corresponding bounds for embedding PS graphs in Butterfly graphs. We do not know if this coincidence suggests a gap in our knowledge or a difference in the powers of Butterfly and deBruijn networks.

Finding definitive answers to the question of how efficiently the families of graphs we have been discussing can simulate each other is an inviting challenge, but seems quite difficult.

ACKNOWLEDGMENTS. It is a pleasure to thank Seth Malitz for helpful comments and suggestions.
A portion of this research was supported by NSF Grant DCI-87-96236.

5. REFERENCES

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

2. 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.

3. 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.

4. S.N. Bhatt and F.T. Leighton (1984): A framework for solving VLSI graph layout problems. *J. Comp. Syst. Sci. 28,* 300-343.

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

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

7. 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.

8. R. Koch, F.T. Leighton, B. Maggs, S. Rao, A.L. Rosenberg (1988): Work-preserving simulations of fixed-connection networks. Typescript, MIT (1988); submitted for publication.

9. F.T. Leighton (1983): *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, MA.

10. F.T. Leighton (1984): Parallel computation using meshes of trees. *1983 Workshop on Graph-Theoretic Concepts in Computer Science,* Trauner Verlag, Linz, pp. 200-218.

11. F.T. Leighton, B. Maggs, S. Rao (1988): Universal packet routing algorithms. *29th IEEE Symp. on Foundations of Computer Science,* 256-269.

12. 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.

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

14. A.G. Ranade (1987): How to emulate shared memory. *28th IEEE Symp. on Foundations of Computer Science*, 185-194.

15. J.H. Reif and L.G. Valiant (1987): A logarithmic time sort for linear networks. *J. ACM 34*, 60-76.

16. A.L. Rosenberg (1988): Shuffle-like interconnection networks. Tech. Rpt. 88-84, Univ. Massachusetts; submitted for publication.

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

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

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

20. C.D. Thompson (1980): *A complexity theory for VLSI*. Ph.D. Thesis, CMU.

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

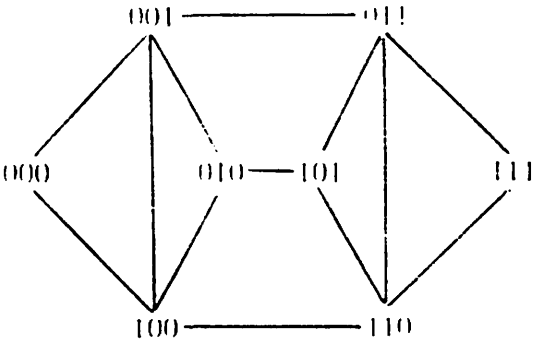

Figure 1: *The DeBruijn graph* $\mathcal{D}(3)$.

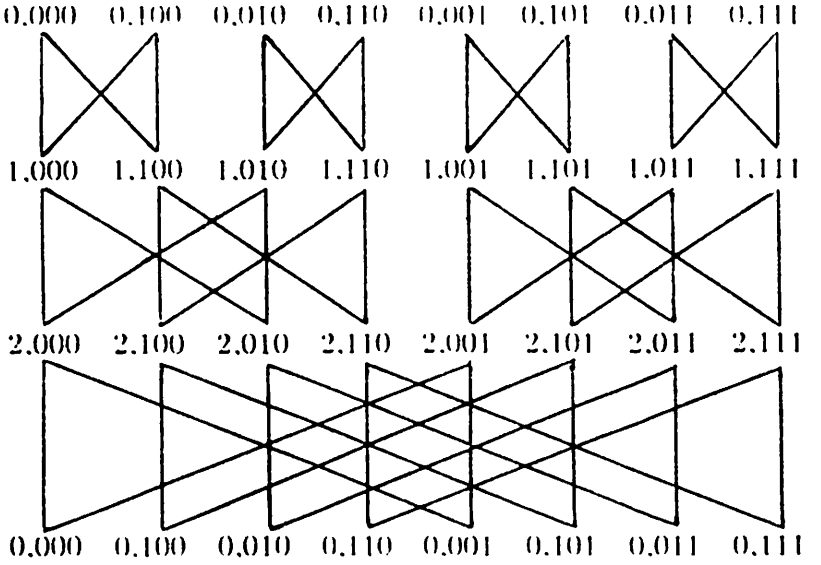

Figure 2: *The Butterfly graph* $\mathcal{B}(3)$ *with, level 0 replicated to aid visualization.*

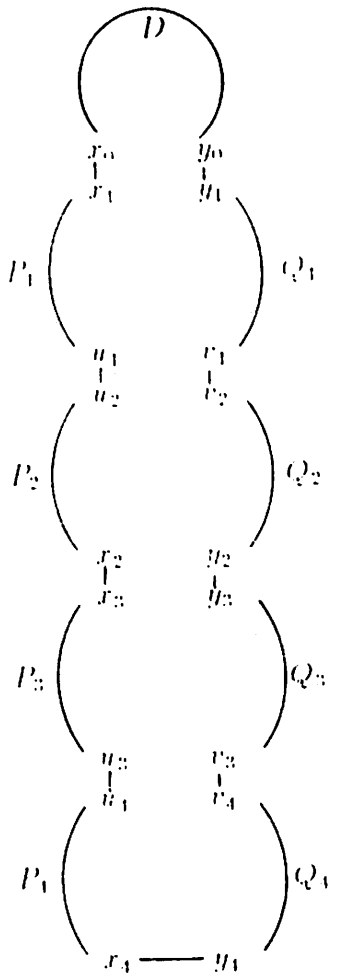

Figure 3: *A schematic view of the cycle constructed in Theorem 1.*

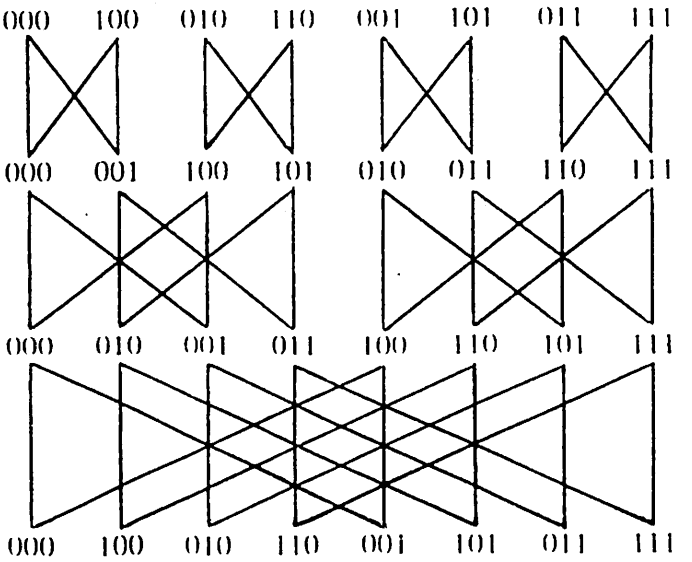

Figure 4: $B(3)$ *(with level 0 replicated) with the shuffle-oriented labelling.*

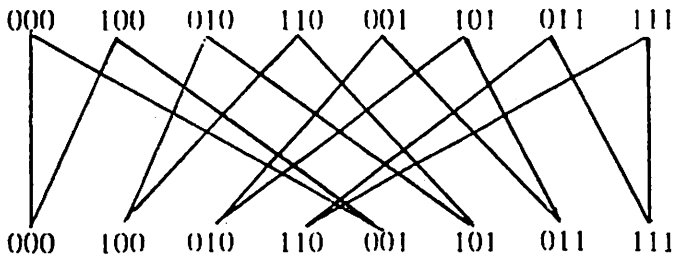

Figure 5: *Two levels of $B(3)$ with the shuffle-oriented labelling, permuted to help visualize the identification.*