EFFICIENT EMULATIONS OF INTERCONNECTION NETWORKS: A Formal Framework and a Case Study

Arnold L. Rosenberg

Computer and Information Science Department
University of Massachusetts

COINS Technical Report 89-40

EFFICIENT EMULATIONS OF INTERCONNECTION NETWORKS: A Formal Framework and a Case Study

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

ABSTRACT. We propose a graph-theoretic framework for comparing the communication powers of competing interconnection networks by seeing how efficiently one of the networks can emulate the other *on general computations* (including pipelining). As a case study illustrating the framework, we present three surprisingly efficient emulations by shuffle-oriented networks (such as the **Shuffle-Exchange** and **deBruijn** networks) of butterfly-oriented networks (such as the **Butterfly** and **Cube-Connected-Cycles** networks). Each emulation is efficient in its own way. (In order to minimize constant factors, we concentrate on the **Butterfly** and **deBruijn** networks.)

- We emulate the order-n **Butterfly** network on the (like-sized) order-$(n + \lceil \log n \rceil)$ **deBruijn** network, incurring a slowdown that is only logarithmic in n, hence *doubly* logarithmic in the size of the emulated network; this is exponentially faster than the anticipated slowdown.
- For any $m \leq n + \lceil \log n \rceil$, we emulate the order-$n$ **Butterfly** network on the order-m **deBruijn** network, incurring a slowdown proportional to $n2^{n-m} \log n$. This emulation is a *work-preserving* derivative of the preceding one, in that the *slowdown* $\times$ *processor-count* product is preserved for any choice of m.
- We emulate the order-n **Butterfly** network on the order-$2n$ **deBruijn** network, incurring a slowdown that is only a factor of 2. In fact, we can emulate roughly $2^n/n$ such **Butterfly** networks simultaneously.

1. INTRODUCTION

1.1. Goals of the Study

Which is the most powerful Hypercube-derivative network?

This apparently simple question is very hard to answer definitively, because there are numerous meaningful notions of "powerful," and not all notions lead to the same answer. If we restrict attention to certain classes of computations, such as convolutions, then all networks in the class have roughly the same power. If we insist on using the networks' structures in a straightforward way, then the butterfly-like networks will appear to be more powerful than the shuffle-oriented networks, since they (seem to) admit pipelining more gracefully. Other groundrules lead to other answers.

Our goal here is to present a formal notion of the *emulation* of one interconnection network by another, which will allow us to study questions of this sort rigorously. The framework we present derives from several sources. It somewhat generalizes the framework enunciated in [6] and used in [1-3, 5, 7] and elsewhere; that framework has recently been shown in [4] to exclude certain useful emulation techniques. It is somewhat less general than the full framework of [4]; our emulations do not require the full power of that approach. The framework we present is simple and tractable, yet formal and rigorous.

1.2. Summary of Results

In order to illustrate the framework, we apply it to our motivating question, adducing certain relevant, though not definitive, results. Specifically, we present three surprisingly efficient emulations by shuffle-oriented networks (such as the **Shuffle-Exchange** and **deBruijn** networks) of butterfly-oriented networks (such as the **Butterfly** and **Cube-Connected-Cycles** networks). Each emulation is efficient in its own way. (In order to minimize constant factors, we concentrate on the **Butterfly** and **deBruijn** networks.)

- We emulate the order-n **Butterfly** network on the (like-sized) order-$(n + \lceil \log n \rceil)$ **deBruijn** network, incurring a slowdown that is only logarithmic in n, hence *doubly* logarithmic in the size of the emulated network; this is exponentially faster than the anticipated slowdown.
- For any $m \leq n + \lceil \log n \rceil$, we emulate the order-$n$ **Butterfly** network on the order-m **deBruijn** network, incurring a slowdown proportional to $n2^{n-m} \log n$. This emulation is a *work-preserving* derivative of the preceding one, in that the *slowdown* $\times$ *processor-count* product is preserved for any choice of m.
- We emulate the order-n **Butterfly** network on the order-$2n$ **deBruijn** network, incurring a slowdown that is only a factor of 2. In fact, we can emulate roughly $2^n/n$ such **Butterfly** networks simultaneously.

These results originate in [1, 4, 7]; related results appear in [2, 3, 5].

2. THE FORMAL FRAMEWORK

2.1. Interconnection Networks as Graphs

The Structure of Networks.

In common with the related sources, we view parallel architectures and their underlying interconnection networks as undirected graphs.[1] Formally,

- the **nodes** of the graph represent **PEs** of the architecture;
- the **edges** of the graph represent inter-PE **communication links**.

Computations on Networks.

We assume a **pulsed** model of computation, wherein **computation steps** alternate with (point-to-point) **communication steps** between adjacent PEs.

The Networks of Interest.

For each positive integer n, the *order-n* **deBruijn** network $\mathcal{D}(n)$ is the graph whose 2^n nodes comprise the set of length-n bit strings. $\mathcal{D}(n)$ has two types of edges: *Shuffle* edges connect nodes

$$0x \leftrightarrow x0 \quad \text{and} \quad 1x \leftrightarrow x1,$$

and *shuffle-exchange* edges connect nodes

$$0x \leftrightarrow x1 \quad \text{and} \quad 1x \leftrightarrow x0,$$

for every length-$(n-1)$ bit string x. See Fig. 1.

For each positive integer n, the *order-n* **Butterfly** network $\mathcal{B}(n)$ is the graph whose $n2^n$ nodes comprise n *levels*, each comprising the 2^n length-n bit strings. $\mathcal{B}(n)$ has two types of edges: *Straight-edges* connect nodes of the form

$$\langle \ell,\ \delta_0\delta_1\cdots\delta_{n-1}\rangle \leftrightarrow \langle (\ell+1 \bmod n),\ \delta_0\delta_1\cdots\delta_{n-1}\rangle,$$

and *cross-edges* connect nodes of the form

$$\langle \ell,\ \delta_0\delta_1\cdots\delta_{n-1}\rangle \leftrightarrow \langle (\ell+1 \bmod n),\ \delta_0\delta_1\cdots\delta_{\ell-1}\bar{\delta}_\ell\delta_{\ell+1}\cdots\delta_{n-1}\rangle,$$

where $\ell \in \{0, 1, \ldots, n-1\}$, and the δ_i are bits. See Fig. 2.

[1]An undirected graph $\mathcal{G}$ is specified by a set $V_\mathcal{G}$ of *nodes* and a set $E_\mathcal{G}$ of two-element subsets of $V_\mathcal{G}$ called *edges*.

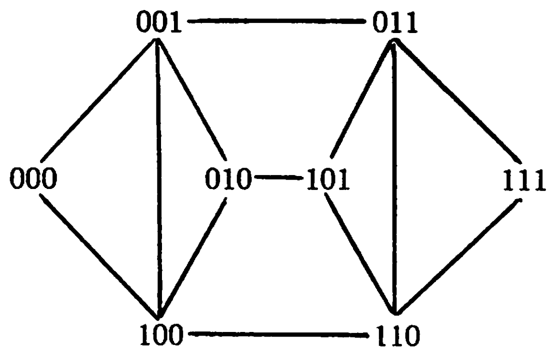

Figure 1: *The DeBruijn graph $\mathcal{D}(3)$.*

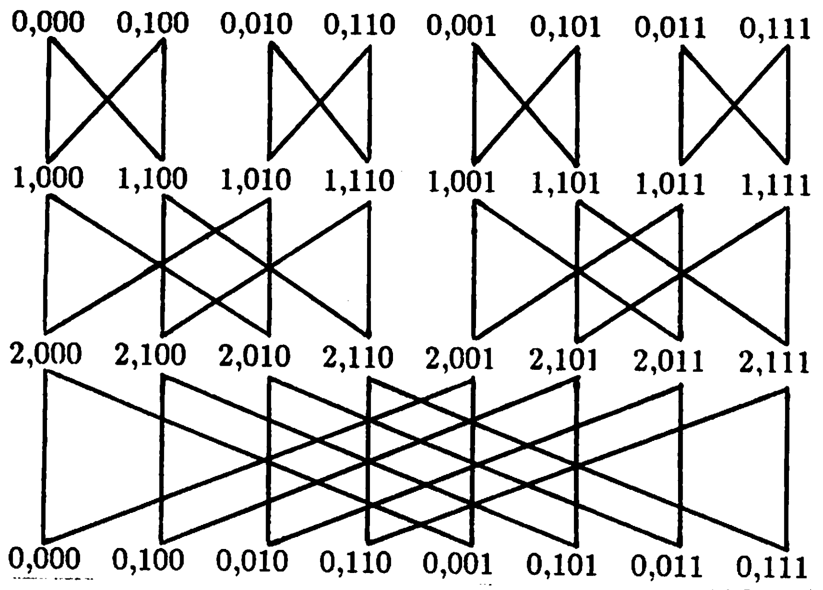

Figure 2: *The Butterfly graph $\mathcal{B}(3)$; level 0 is replicated to aid visualization.*

2.2. A Formal Notion of Emulation

The Components of an Emulation.

Our formal notion of architecture $\mathcal{H}$ *emulating* architecture $\mathcal{G}$ has the following components.

- We **assign** each PE of $\mathcal{G}$ to one PE of $\mathcal{H}$.
  The **assignment** *is accomplished with the (possibly many-to-one)* **assignment map** α.
- We **route** each *link* of $\mathcal{G}$ along a *routing-path* in $\mathcal{H}$.
  The **routing** *is specified by the* **routing function** ρ.
- We have $\mathcal{H}$ **emulate** $\mathcal{G}$ via alternating **computation** and **communication** *phases*:
  - During each **computation phase**:
    Each $\mathcal{H}$-PE executes one computation step for each $\mathcal{G}$-PE assigned to it by the **assignment map** α.
  - During each **communication phase**:
    Each "message" from $\mathcal{G}$-PE v to $\mathcal{G}$-PE w is routed (using the **routing function** ρ) from node $\alpha(v)$ of $\mathcal{H}$ to node $\alpha(w)$ of $\mathcal{H}$.
    (If α assigns both v and w to the same node of $\mathcal{H}$ [i.e., $\alpha(v) = \alpha(w)$], then this communication is "internal" to the $\mathcal{H}$-PE $\alpha(v)$.)

Sources of Slowdown.

In order to have a meaningful notion of the *efficiency of an emulation*, we must understand the sources of slowdown in our stylized type of emulation. We have isolated three major sources.

1. The **Load Factor** of the **assignment** is the largest number of $\mathcal{G}$-PEs that α assigns to any one $\mathcal{H}$-PE.
   Load Factor incurs slowdown because every PE of $\mathcal{H}$ must emulate one computation step of each of its assigned $\mathcal{G}$-PEs during each computation phase of the emulation.
2. The **Dilation** of the **routing** is the length of the longest ρ-routing path (in $\mathcal{H}$) used to emulate a single communication link in $\mathcal{G}$.
   Dilation incurs slowdown because every message that crosses link e in Architecture $\mathcal{G}$ must traverse path $\rho(e)$ in Architecture $\mathcal{H}$.

3. The **Congestion** of the **routing** is the largest number of ρ-paths crossing a single edge of $\mathcal{H}$.
   Congestion incurs slowdown because the messages that want to cross a congested edge must be queued up. (For simplicity, we are giving each edge of $\mathcal{H}$ unit capacity; endowing edges with some other constant capacity will reduce our calculated slowdown due to **Congestion** by that constant factor.)

In many emulations, it is possible to *amortize* the effects of **Congestion**, by *orchestrating* the communication phase of the emulation. When this is possible (as it is in our emulations; see [7] for details), we can replace **Congestion** as a source of slowdown by

$$\textbf{Dynamic Congestion} = \textbf{Congestion}/\textbf{Dilation}.$$

When this is possible, after all considerations, we end up with:

$$\boxed{\textbf{Slowdown} = \textbf{Load Factor} + \textbf{Congestion}}$$

3. EFFICIENT EMULATION OF BUTTERFLIES

The order-$(n + \lceil \log_2 n \rceil)$ **deBruijn** *network can emulate the order-n* **Butterfly** *network, with slowdown $O(\log n)$.*

This is exponentially faster than previous emulations.

The strategy of this emulation is to decompose the task into a sequence of simpler emulations:

1. *Emulatee:* the order-n **Butterfly**
   Emulator: the *product network*[2] (length-n **Cycle**) $\times$ (order-n **deBruijn**)
   Slowdown Factor: 2

2. *Emulatee:* (length-n **Cycle**) $\times$ (order-n **deBruijn**)
   Emulator: (order-$\lceil \log_2 n \rceil$ **deBruijn**) $\times$ (order-n **deBruijn**)
   Slowdown Factor: 1

3. *Emulatee:* (order-$\lceil \log_2 n \rceil$ **deBruijn**) $\times$ (order-n **deBruijn**)
   Emulator: order-$(n + \lceil \log_2 n \rceil)$ **deBruijn** network
   Slowdown Factor: $2 \log_2 n + O(1)$

[2]The nodes of the *product* $\mathcal{G} \times \mathcal{H}$ of graphs $\mathcal{G}$ and $\mathcal{H}$ are ordered pairs $\langle u, x \rangle$, where u is a node of $\mathcal{G}$ and x is a node of $\mathcal{H}$. Every edge $u \leftrightarrow v$ of $\mathcal{G}$ engenders a set $\{\langle u, x \rangle \leftrightarrow \langle v, x \rangle\}$ of edges of $\mathcal{G} \times \mathcal{H}$; every edge $x \leftrightarrow y$ of $\mathcal{H}$ engenders a set $\{\langle u, x \rangle \leftrightarrow \langle u, y \rangle\}$ of edges of $\mathcal{G} \times \mathcal{H}$.

We shall sketch the first and third of these emulations, the second being immediate from the following result of Yoeli [8]:

$\boxed{\text{The } \textbf{deBruijn} \text{ network contains cycles of every length.}}$

3.1. Emulating the Butterfly on the (Cycle $\times$ deBruijn)

Our emulation depends on the following interesting fact.

> *The order-n* **Butterfly** *network "contains" a copy of the order-n* **deBruijn** *network between every two consecutive levels.*

One can verify instances of this fact visually; cf. Figs. 3 and 4.

3.2. Emulating the Product of deBruijn Networks on a Big deBruijn Network

Let $\lambda =_{\text{def}} \lceil \log_2 n \rceil$. We emulate the product network $\mathcal{D}(\lambda) \times \mathcal{D}(n)$ on $\mathcal{D}(n+\lambda)$, as follows.

- We **assign** node $\langle x, y \rangle$ of $\mathcal{D}(\lambda) \times \mathcal{D}(n)$ to node xy of $\mathcal{D}(n+\lambda)$.
- We **route** links of $\mathcal{D}(\lambda) \times \mathcal{D}(n)$ in $\mathcal{D}(n+\lambda)$ as follows.
  - We route the link
    $$\langle x, y \rangle \leftrightarrow \langle x, y' \rangle$$
    within copy x of $\mathcal{D}(n)$ along the length-$(2\lambda+1)$ path
    $$xy \leftrightarrow \cdots \leftrightarrow xy'$$
    in $\mathcal{D}(n+\lambda)$.
  - We route the link
    $$\langle x, y \rangle \leftrightarrow \langle x', y \rangle$$
    between copies x, x' of $\mathcal{D}(n)$ along the length-2λ path
    $$xy \leftrightarrow \cdots \leftrightarrow x'y$$
    in $\mathcal{D}(n+\lambda)$.
- We **orchestrate** the link-routings to get **Dynamic Congestion** $O(1)$.

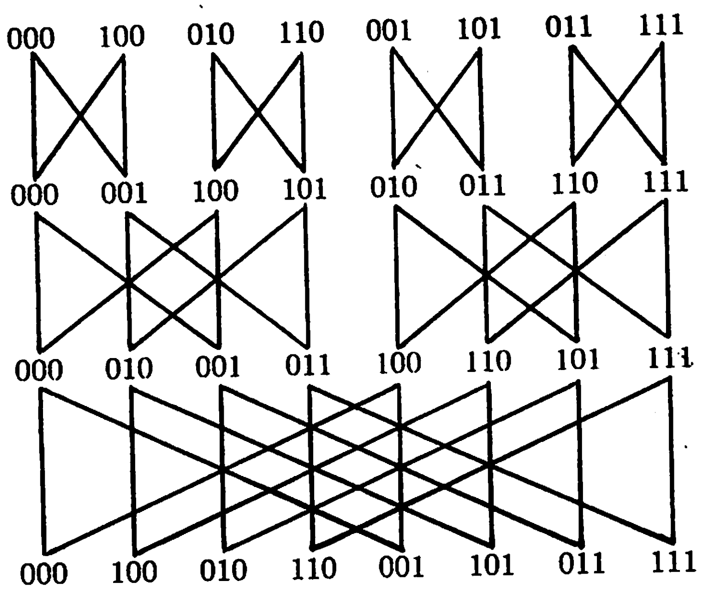

Figure 3: The order-3 **Butterfly** network, with a "shuffle-oriented" node-labelling (deriving from [1]).

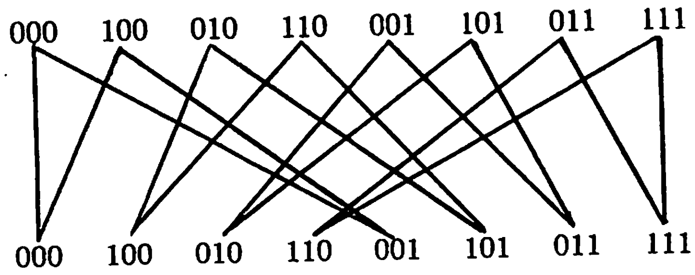

Figure 4: Two consecutive levels of the order-3 **Butterfly** network, permuted so that like-labelled columns line up. Collapsing the levels yields the **deBruijn** network.

We illustrate the routing of intra-copy links, the routing of inter-copy links being similar. Every intra-copy link connects a node of the form xy (x of length λ and y of length n) to a node of the form $x[y]_{n-1}\delta$, where $[y]_m$ denotes the length-m suffix of y, and $\delta \in \{0,1\}$.

$$\boxed{xy \leftrightarrow \cdots \leftrightarrow xv\delta \quad (x = \xi_1\xi_2\cdots\xi_\lambda;\ v = [y]_{n-1})}$$

$$\begin{array}{rcll}
\xi_1\xi_2\xi_3\cdots\xi_\lambda\delta v & \leftrightarrow & \xi_2\xi_3\cdots\xi_\lambda\delta v\delta & \text{shuffle(-exchange)} \\
 & \leftrightarrow & \xi_3\cdots\xi_\lambda\delta v\delta\xi_1 & \text{shuffle(-exchange)} \\
 & \cdots & & \\
 & \leftrightarrow & \delta v\delta\xi_1\xi_2\xi_3\cdots\xi_{\lambda-1} & \text{shuffle(-exchange)} \\
 & \leftrightarrow & v\delta\xi_1\xi_2\xi_3\cdots\xi_{\lambda-1}\xi_\lambda & \text{shuffle(-exchange)} \\
 & \leftrightarrow & \xi_\lambda v\delta\xi_1\xi_2\xi_3\cdots\xi_{\lambda-1} & \text{unshuffle} \\
 & \cdots & & \\
 & \leftrightarrow & \xi_2\xi_3\cdots\xi_\lambda v\delta\xi_1 & \text{unshuffle} \\
 & \leftrightarrow & \xi_1\xi_2\xi_3\cdots\xi_\lambda v\delta & \text{unshuffle}
\end{array}$$

Finally, we indicate why our emulation suffers only the claimed slowdown.

1. Our **assignment** is one-to-one, so our emulation has unit **Load Factor**.

2. Our link-routing engenders **Congestion** $O(\log n)$, because of the structure of the **deBruijn** network.

3. By orchestrating our routing, we ensure that

$$\textbf{Slowdown} = \textbf{Load Factor} + \textbf{Congestion}$$

These three facts guarantee that our emulation engenders only **Slowdown** $O(\log n)$.

4. A WORK-PRESERVING PROCESSOR-TIME TRADEOFF

The result in the previous section has a **deBruijn** network emulating an equal size **Butterfly** network with *modest slowdown* but *no savings in processor-count.* We now build on that result to have a **deBruijn** network emulate a smaller **Butterfly** network with *increased slowdown* but *commensurate decrease in processor-count.* This yields a **processor-time tradeoff** which is **work preserving**, in the sense that

$$\boxed{\text{The } \textbf{slowdown} \times \textbf{processor-count} \text{ product is constant.}}$$

Our tradeoff takes the following form.

For all $m \leq n + \lceil \log_2 n \rceil$, the order-m **deBruijn** *network can emulate the order-n* **Butterfly** *network, with slowdown $O(n2^{n-m} \log n)$.*

The tradeoff follows from the fact that small **deBruijn** networks can rather efficiently emulate big ones:

For all m, n, the order-n **deBruijn** *network can emulate the order-$(m + n)$* **deBruijn** *network, with slowdown 2^m.*

This auxiliary emulation proceeds as follows.

- We **assign** node z of $\mathcal{D}(m + n)$ to node $[z]_n$ of $\mathcal{D}(n)$, a 2^m-to-1 mapping.
- We **route** link
$$z \leftrightarrow [z]_{m+n-1}\delta$$
of $\mathcal{D}(m + n)$ — all links have this form — via the link
$$[z]_n \leftrightarrow [z]_{n-1}\delta$$
in $\mathcal{D}(n)$.

Since the **routing** in our emulation maps links to links, it engenders no slowdown. Thus, the slowdown in our emulation results solely from the **Load Factor** of the **assignment**, which is 2^m.

5. OPTIMAL-SPEED EMULATION OF SMALL BUTTERFLIES

The order-2n **deBruijn** *network can emulate the order-n* **Butterfly** *network, with just a factor-of-2 slowdown.*

At first blush, this emulation seems less surprising than the former ones, because the **deBruijn** network consists of a collection of butterflies:

$$\begin{array}{cc} 0x & 1x \\ \\ x0 & x1 \end{array}$$

However, there are two (related) problems with these butterflies, which prevent an easy emulation.

1. Some of these butterflies degenerate.
2. The **deBruijn** network lacks the *levelled* structure of the **Butterfly** network.

If one is willing to use the **deBruijn** network to emulate only a *small* **Butterfly** network, then there is a simple way to overcome these problems: by appropriatelyinterspersing cyclic rotations of the string $100\cdots 0$ with the bits of the **Butterfly**'s node "address-strings":

- We **assign** node
$$\langle 1, \delta_0\delta_1\cdots\delta_{n-1}\rangle$$
of $\mathcal{B}(n)$ to node
$$\delta_1 0 \delta_2 0 \cdots \delta_{n-2} 0 \delta_{n-1} 1 \delta_0 0$$
of $\mathcal{D}(2n)$.
  - The "active" bit-position is at the left; the index to this position is numbered from the right.
  - The position of the "1" in each cyclic rotation indicates which **Butterfly** level is being emulated.
  - The interspersed bits keep butterflies from degenerating.
- We **route** link
$$\langle 1, \delta_0\delta_1\cdots\delta_{n-1}\rangle \leftrightarrow \langle 2, \delta_0\hat{\delta}_1\cdots\delta_{n-1}\rangle$$
of $\mathcal{B}(n)$ ($\hat{\delta}_1 \in \{\delta_1, \bar{\delta}_1\}$) via the length-2 path
$$\begin{aligned} \delta_1 0 \delta_2 0 \cdots \delta_{n-2} 0 \delta_{n-1} 1 \delta_0 0 &\leftrightarrow 0 \delta_2 0 \cdots \delta_{n-2} 0 \delta_{n-1} 1 \delta_0 0 \hat{\delta}_1 \\ &\leftrightarrow \delta_2 0 \cdots \delta_{n-2} 0 \delta_{n-1} 1 \delta_0 0 \hat{\delta}_1 0 \end{aligned}$$
in $\mathcal{D}(2n)$.
  - Two **deBruijn** moves change both the "active" bit-position and its index.

Since this emulation has unit **Load Factor**, the entire slowdown results from **Dilation**; hence, the slowdown is 2.

Satish Rao (MIT) has generalized this solution, bringing the size of the emulated **Butterfly** closer to the size of the emulating **deBruijn** network, at the cost of increasing the slowdown factor.

ACKNOWLEDGMENTS. It is a pleasure to express thanks to my collaborators on [1, 4, 7], on which the current paper is based: Fred Annexstein, Marc Baumslag, Sandeep Bhatt, Fan Chung, and Tom Leighton.
A portion of this research was supported by NSF Grants DCI-87-96236 and CCR-88-12567.

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. M. Baumslag and A.L. Rosenberg (1989): Processor-time tradeoffs for Cayley-graph interconnection networks. In preparation, Univ. Massachusetts.

3. 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.

4. R. Koch, F.T. Leighton, B. Maggs, S. Rao, A.L. Rosenberg (1989): Work-preserving emulations of fixed-connection networks. *21st ACM Symp. on Theory of Computing,* Seattle, WA.

5. A. Raghunathan and H. Saran (1989): Is the shuffle-exchange better than the butterfly? Typescript, Univ. California, Berkeley.

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

7. A.L. Rosenberg (1988): Shuffle-like interconnection networks. Tech. Rpt. 88-84, Univ. Massachusetts.

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