Efficient Nonblocking Switching Networks for Interprocessor Communications in Multiprocessor Systems

Fong-Chih Shao and A. Yavuz Oruç, *Senior Member, IEEE*

***Abstract*— The performance of a multiprocessor system depends heavily on its ability to provide conflict free paths among its processors. In this paper, we explore the possibility of using a nonblocking network with $O(N \log N)$ edges (crosspoints) to interconnect the processors of an N processor system. We combine Bassalygo and Pinsker's implicit design of strictly nonblocking networks with an explicit construction of expanders to obtain a strictly nonblocking network with $-765.18N + 352.8N \log N$ edges and $2 + \log(N/5)$ depth. We present an efficient parallel algorithm for routing connection requests on this network and implement it on three parallel processor topologies. The implementation on a parallel processor whose processing elements are interconnected as in the Bassalygo–Pinsker network requires $O(N \log N)$ processing elements, $O(N \log N)$ interprocessor links and it takes $O(\log N)$ steps to route any single connection request where each step involves a small number ($\approx$72) of bit-level operations. A contracted or folded version of the same implementation reduces the processing element count to $O(N)$ without increasing the link count or the routing time. Finally, we establish that the same algorithm takes $O(\log^3 N)$ steps on a perfect shuffle processor with $O(N)$ processing elements. These results improve the crosspoint, depth and routing time complexities of the previously reported strictly nonblocking networks.**

***Index Terms*—Bassalygo–Pinsker network, Clos network, Cantor network, extensive graph, expander, parallel routing algorithm, strictly nonblocking network.**

I. Introduction

THE PERFORMANCE of a multiprocessor system hinges critically on the intercommunication ability of its processors which is often impeded by the bandwidth of the network interconnecting them. The full crossbar network goes a long way to solve the bandwidth problem, but at the expense of $O(N^2)$ crosspoints for an N-processor system. What is desirable is a network which functions much like a crossbar but with fewer crosspoints. Such networks have been investigated extensively in communication switching and are commonly referred to as strictly nonblocking networks [3].

Formally, an $N \times N$ strictly nonblocking network is a directed acyclic graph with N source vertices, called inputs, and N sink vertices called outputs such that, given any idle input x and idle output y, the network always possesses a path from x to y which does not overlap with any of the already established paths between other inputs and outputs.[1] A number of strictly nonblocking networks comprising fewer than $O(N^2)$ crosspoints have been reported in the literature. The well known 3-stage Clos network [6] provides a strictly nonblocking network with $O(2^j N^{1+\frac{2}{j+1}})$ crosspoints and depth $2j + 1$. Improved versions of Clos network due to Cantor [5] give a strictly nonblocking network with $O(N \log^a N)$ crosspoints, $2 < a \leq 3$, and $O(\log^a N)$ depth. Cantor also provided the first strictly nonblocking network with $O(N \log^2 N)$ crosspoints and $O(\log N)$ depth [5]. Subsequent to these efforts, Bassalygo and Pinsker [4], [2] obtained a strictly nonblocking network with $O(N \log N)$ crosspoints and $O(\log N)$ depth. Nonetheless, unlike Clos and Cantor networks, Bassalygo and Pinsker's network relies on the existence of certain bipartite graphs with $O(N)$ edges, called *extensive graphs*. This makes their network implicit or nonconstructive as they were only able to prove that such graphs exist without an actual construction.

In this paper, we explore the possibility of explicitly constructing strictly nonblocking networks with $O(N \log N)$ crosspoints, $O(\log N)$ depth and $O(\log N)$ routing time. Here, routing generically refers to setting and abolishing paths between the idle inputs and idle outputs of a nonblocking network. We differentiate between two specific routing problems. The first routing problem deals with setting (or abolishing) a path between any idle (or busy) pair of inputs and outputs. The second routing problem deals with setting (or abolishing) paths between any two or more idle (or busy) pairs of inputs and outputs. We shall refer to these two problems as *single routing assignment problem* and *multiple routing assignment problem*, respectively.

Two recent efforts on routing problems in nonblocking networks are worth mentioning here. In [1], Arora et al. reported a greedy algorithm for setting paths between single and multiple pairs of idle inputs and outputs in multi-Beneš networks that encompass $O(N \log N)$ crosspoints and have $O(\log N)$ depth. Even though multi-Beneš networks meet our objective of constructing efficient nonblocking networks in order of complexity terms, the results reported in [1] have

Manuscript received March 8, 1994; revised July 5, 1994. This work was supported in part by the Graduate Studies and Research Board at the University of Maryland.

The authors are with the Department of Electrical Engineering and the Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA.

IEEE Log Number 9406349.

[1] It should be emphasized that strictly nonblocking networks are more powerful than baseline, omega, Beneš and other multistage networks in that the established connections in these networks never have to be rearranged or disrupted for new requests.

at least two drawbacks. First, multi-Beneš networks require expanders with large expansion coefficients in order to have a small constant factor in their routing time complexity. It is stated in [1] that one could generate expanders with large expansion coefficients randomly. However, this assumption makes multi-Beneš networks nonconstructive or implicit in contrast to our requirement that the construction be explicit.[2] The second drawback is that the time complexity of the greedy routing algorithm described in [1] depends on two key design parameters, namely, the in-degree (or out-degree) of each switch node d, and loading factor L. This relation was not spelled out exactly in [1], but the authors pointed out that to reduce the routing time one need to increase L and d. In particular, they stated that, to achieve a routing time of $100 \log N$, the values of d and L should approximately be 10 and 300, respectively. As we will see in Section II–B, d and L also affect the constant in the crosspoint complexity of multi-Beneš networks. More specifically, when $d = 10$ and $L = 300$ constructing a multi-Beneš network requires about $240,000N \log N$ crosspoints.

More recently, Lin and Pippenger [9] reported both deterministic and nondeterministic path selection algorithms for a strictly nonblocking network which resembles Cantor's network. Nonetheless this network exacts more than $O(N \log N)$ crosspoints. Moreover, it uses an $O(\log^5 N)$step path selection algorithm, or at best, a nondeterministic path selection algorithm requiring $O(\log^2 N)$ steps.

In this paper, these results are substantially improved. We provide a strictly nonblocking network construction with $-756.18N + 352.8N \log N$ crosspoints and $2 + \log(N/5)$ depth by combining an explicit expander construction of Galil, Alon and Milman [12] with the strictly nonblocking network introduced by Bassalygo and Pinsker [4]. We present $O(\log N)$ bit-step parallel algorithms to solve the single routing assignment problem for both setting and abolishing paths and the multiple assignment problem for abolishing paths on this new nonblocking network. Our algorithms can also be used to establish paths for multiple assignments but, at present, this requires $O(k \log N)$ bit-steps where k is the number of requests ((input, output) pairs) in the assignment. We realize our algorithms on a number of parallel processors consisting of $O(N)$ to $O(N \log N)$ processors and encompassing $O(N)$ to $O(N \log N)$ communication links. The implementation on a parallel processor whose processing elements are interconnected as in the Bassalygo-Pinsker network requires $O(N \log N)$ processing elements, $O(N \log N)$ interprocessor links and it takes $O(\log N)$ bit-steps to route any single connection request where each step involves a small number ($\approx$72) of bit-level operations. A contracted version of the same implementation reduces the processing element count to $O(N)$ without increasing the link count or the routing time. Finally, we establish that the same algorithm takes $O(\log^3 N)$ bit-steps on a perfect shuffle processor with $O(N)$ processing elements.

[2]It is possible to explicitly construct multi-Beneš networks by realizing the splitters in these networks in terms of explicit constructions of expanders. Nonetheless, [1] did not provide an analysis of the routing time complexity of multi-Beneš networks for such a realization.

The rest of the paper is organized as follows. Section II gives the definitions used in the paper and provides an explicit construction of the Bassalygo-Pinsker network. Section III describes the path-setting and path-abolishing algorithms. Section IV describes how these algorithms are implemented on a number of parallel processors, and the paper is concluded in Section V.

II. Basic Facts

In the first part of this section, we restate some definitions and facts mostly from [4] which will be used throughout the paper. In the second part we obtain an explicit construction of extensive graphs that are the building blocks of Bassalygo-Pinsker nonblocking networks.[3]

A. Definitions and Previous Results

Definition 1: Let $G = (A, B, E)$ be a bipartite graph with sets of vertices A, B and a set of edges E. The vertices in A are called the inputs of G and the vertices in B are called the outputs of G. A vertex y in B is called a neighbor of a vertex x in A if $(x, y) \in E$. Graph G is said to be d-homogeneous if each vertex in A is joined to exactly d neighbors in B.

Definition 2: Let $G = (A, B, E)$ be a bipartite graph, $\Gamma(X)$ denote the set of neighbors of a set of inputs $X \subseteq A$, and c be a positive number. G is called an (n, d, c)-expander if $|A| = |B| = n$, the degree of every vertex in G is d, and if $|\Gamma(X)| \geq (1 + c(1 - \frac{|X|}{n}))|X|$, for every set $X \subseteq A$, where $\frac{|X|}{n} \leq 1/2$.

Definition 3: $G = (A, B, E)$ is called an (m, r, α, β)-extensive graph if $|A| = m$, $|B| = r$, and if $|\Gamma(X)| \geq \beta r$ for every set $X \subseteq A$ such that $|X| = \alpha m$, where α and β are some positive fractions (i.e., $0 < \alpha, \beta < 1$) independent of m and r.

Definition 4: An $F_{a,m}$ network is a directed acyclic graph with m inputs and am outputs satisfying the following condition: For any number r $(r < m)$ of paths already established between the m inputs and the am outputs, any one of the remaining idle inputs can be connected to at least $\lceil \frac{am-r+1}{2} \rceil$ of the remaining idle outputs.

Lemma 1—(Bassalygo-Pinsker): An $F_{a,km}$ network can be constructed as shown in Fig. 1 by *(i)* cascading an $F_{a,m}$ network with an (am, kam, α, β)-extensive graph, *(ii)* replicating the cascade k times, and *(iii)* joining together the ith outputs of the k cascades, $1 \leq i \leq kam$, $\alpha = (a-1)/2a$, $\beta = (a+1)/2a$ and $\alpha + \beta = 1$.

We prove this lemma as the proof provides an insight into the construction of an $F_{a,km}$ network.[4]

Proof: The lemma is proved by induction. The basis of induction is a $\lceil (at + t)/2 \rceil$-homogeneous bipartite graph with t inputs and at outputs. In this case, given r established paths, each idle input can be connected to

$$\left\lceil \frac{at + t}{2} \right\rceil - r = \left\lceil \frac{at - r + 1}{2} + \frac{t - r - 1}{2} \right\rceil \geq \left\lceil \frac{at - r + 1}{2} \right\rceil$$

[3]The reader who is sufficiently familiar with [4] may skip Section II–A.

[4]It is worth noting that this lemma was stated without a proof in [4].

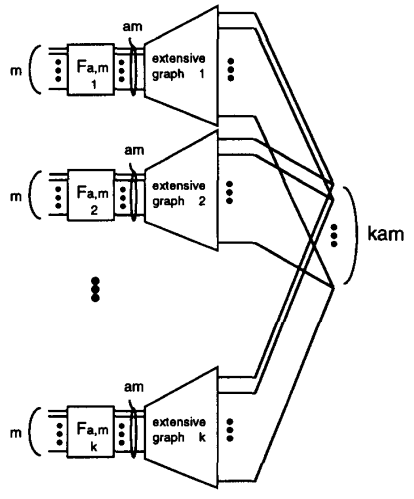


Fig. 1. An iterative construction of $F_{a,km}$ network. (The network has km inputs and kam outputs.)

idle outputs since $r \leq t-1$. Therefore, the first stage is an $F_{a,t}$ network. Now consider the $F_{a,km}$ network in Fig. 1. Let the $F_{a,m}$ networks be numbered $1, 2, \ldots, k$, the (am, kam, α, β)-extensive graphs also be numbered $1, 2, \ldots, k$ and suppose that there are r_i established paths within the ith $F_{a,m}$ network and $\sum_{i=1}^{k} r_i = r$. Without loss of generality, assume that there is at least one idle input in the first $F_{a,m}$ network so that $0 \leq r_1 \leq m-1$ and $0 \leq \sum_{i=1}^{k} r_i \leq km-1$. Let x denote this idle input. By the definition of an $F_{a,m}$ network, x can be connected to $\lceil \frac{am-r_1+1}{2} \rceil \geq \lceil \frac{am-m+2}{2} \rceil$ outputs of the first $F_{a,m}$ network, which are also the inputs of the first (am, kam, α, β)-extensive graph. The (am, kam, α, β)-extensive graph guarantees that every $\alpha am = \frac{am-m}{2}$ of its inputs are connected to at least $\beta kam = \frac{(a+1)kam}{2a} = \frac{kam+km}{2}$ of its outputs. So, the $\lceil \frac{am-m+2}{2} \rceil$ outputs of the first $F_{a,m}$ network can be connected to at least $\lceil \frac{kam+km}{2} \rceil$ outputs of the first (am, kam, α, β)-extensive graph. Since the number of established paths is at most r, the idle input x can therefore be connected to at least $\lceil \frac{kam+km}{2} \rceil - r = \lceil \frac{kam-r+1}{2} + \frac{km-(1+r)}{2} \rceil$ idle outputs. Because $0 \leq r \leq km-1$, it follows that x can be connected to at least $\lceil \frac{(kam-r+1)}{2} \rceil$ idle outputs. Therefore, by Definition 4, the construction in Fig. 1 is an $F_{a,km}$ network. □

The following theorem gives a strictly nonblocking network in terms of two $F_{a,N}$ networks.

Theorem (Bassalygo-Pinsker): Let G be a network obtained by tieing two $F_{a,N}$ networks back-to-back as shown in Fig. 2. G is an N-input strictly nonblocking network.

Proof: Consider the vertices in-between the two $F_{a,N}$ networks. Given r connections between any r inputs and any r outputs, each idle input on the left can reach at least $\frac{aN-r+1}{2}$ of these vertices, and each idle output on the right can also reach $\frac{aN-r+1}{2}$ of these vertices. Since $\frac{aN-r+1}{2} + \frac{aN-r+1}{2} > aN-r$, at least one of these vertices can be reached by both the idle input and idle output, and thus the statement follows. □

The two-stage construction of the $F_{a,km}$ network in Fig. 1 can be applied iteratively to obtain an $(s+1)$ stage F_{a,k^st} network for any $s \geq 1$. Stage 0 of an F_{a,k^st} network consists of k^s $\lceil at + t/2 \rceil$-homogeneous bipartite graphs, each with t inputs and at outputs. Stage 1 consists of k^s (at, kat, α, β)-extensive graphs, stage 2 consists of k^{s-1} $(kat, k^2at, \alpha, \beta)$-extensive graphs, and, in general, stage i consists of k^{s-i+1} $(k^{i-1}at, k^iat, \alpha, \beta)$-extensive graphs, where α and β are fixed as in Lemma 1. To complete the design of the F_{a,k^st} network, Bassalygo and Pinsker showed that there exist bipartite δ-homogeneous $(k^{i-1}at, k^iat, \alpha, \beta)$-extensive graphs where δ is given by

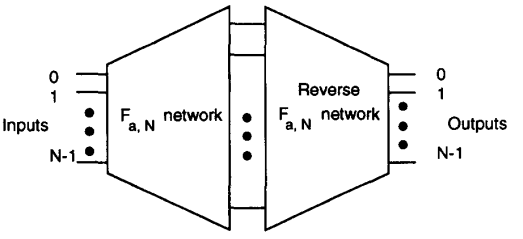


Fig. 2. A nonblocking network constructed with $F_{a,N}$ networks.

$$\delta = k + \left\lceil \frac{H(\alpha) + k(1-\alpha)H(\frac{1-\beta}{1-\alpha})}{\alpha \log(1/\beta)} \right\rceil \tag{1}$$

and $H(\cdot)$ is the binary entropy function, i.e., $H(x) = -x \log x - (1-x)\log(1-x), 0 < x < 1$.

It is easy to see that the F_{a,k^st} network constructed this way has k^st inputs and ak^st outputs. Furthermore, summing the number of edges in the homogeneous graphs in stage 0 and the extensive graphs in all the remaining stages, the number of edges in the F_{a,k^st} network is found to be

$$k^st\lceil (at+t)/2 \rceil + \sum_{i=1}^{s} \delta k^{i-1} atk^{s-i+1} \tag{2}$$

$$= k^st\lceil (at+t)/2 \rceil + \sum_{i=1}^{s} a\delta tk^s \tag{3}$$

$$= k^st\lceil (at+t)/2 \rceil + sa\delta tk^s \tag{4}$$

$$= N\lceil (at+t)/2 \rceil - \frac{a\delta}{\log k}(\log t)N + \frac{a\delta}{\log k} N \log N \tag{5}$$

where $N = tk^s$.

Even though this result does not lead to an explicit construction of an F_{a,k^st} network (as it relies on the existence of extensive graphs), it can be used to upper bound the number of edges in such a network. In particular, Bassalygo and Pinsker fixed $k = 4$, $a = 50/17$, $t = 17$ $\alpha = (a-1)/2a = 0.33$, and $\beta = (a+1)/2a = 0.67$ to show that there exists a $((50)4^{i-1}, (50)4^i, 0.33, 0.67)$-extensive graph for each i, $1 \leq i \leq \log_k(N/17)$. Each of these is a 23-homogeneous bipartite graph where $\delta = 23$ is worked out from (1). Substituting the values of k, a, and t in (4) and noting that $N = (17)4^s$, the number of edges in the $F_{50/17,N}$ network is determined as $\lceil 34 * 17 * 4^s + 1150s4^s \rceil = \lceil 67.65N \log N \rceil$.

B. Explicit Construction of Extensive Graphs

Bassalygo and Pinsker obtained an N-input strictly nonblocking network with $136N \log_4 N$ edges by cascading two

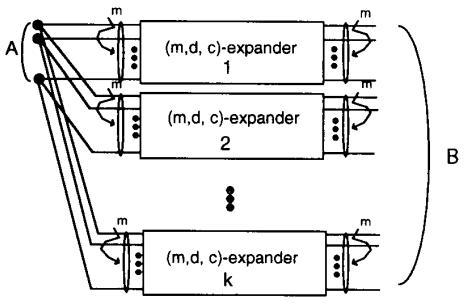


Fig. 3. Explicit construction of an (m, km, α, β)-extensive graph. (The network has m inputs and km outputs.)

$F_{50/17,N}$ networks back-to-back, where each $F_{50/17,N}$ network consists of $1+s = 1+\log_4(N/17)$ stages, and hence the two networks cascaded together have $2+2\log_4(N/17)$ stages. While this network has $O(N \log N)$ edges and $O(\log N)$ stages, it is only an existential construction, since the extensive graphs used in its construction were shown only to exist but not explicitly constructed.[5] In order to convert this network to an explicit form, we first obtain an explicit construction of Bassalygo and Pinsker's extensive graphs by using the following lemma.

Lemma 2: Let $G = (A, B, E)$ be a graph obtained by joining together the ith inputs of k (m, d, c)-expanders, $1 \leq i \leq m$, as shown in Fig. 3. G is an (m, km, α, β)-extensive graph, where $0 \leq \alpha \leq 1/2$ and $0 \leq \beta \leq \alpha(1 + c(1 - \alpha))$.

Proof: Let $X \subseteq A$ where $|X| = \alpha m \leq m/2$. The inputs in X can be joined to $(1 + c(1 - \alpha))|X|$ outputs of each of the (m, d, c)-expanders and thus can be joined to $k(1 + c(1 - \alpha))\alpha m = \alpha(1 + c(1 - \alpha))km$ outputs of G. This implies that the inputs in X can be joined to βkm outputs of G for any $\beta \leq \alpha(1 + c(1 - \alpha))$. Therefore, G is an (m, km, α, β)-extensive graph with $0 \leq \alpha \leq 1/2$ and $0 \leq \beta \leq \alpha(1 + c(1 - \alpha))$. □

Remark: It is noted that the degree of each input of G is kd (each input in A is connected to d outputs in each (m, d, c)-expander), and the degree of each output of G is d. In Fig. 3, the lines that connect the inputs in A to the inputs of the (m, d, c)-expanders do not represent edges; they are used merely to identify the connections between the inputs in A and the inputs of the expanders.

The main point of this lemma is that an (m, km, α, β)-extensive graph can be constructed by using k (m, d, c)-expanders. In the context of Lemma 1, this amounts to replacing each of the $(k^{i-1}at, k^iat, \alpha, \beta)$-extensive graphs in the ith stage of the $F_{a,N}$ network, by k $(k^{i-1}at, d, c)$-expanders subject to the following conditions:

i) $\beta \leq \alpha(1 + c(1 - \alpha))$, (from Lemma 2)
ii) $\alpha \leq 1/2$ (from Definition 2)
iii) $\alpha + \beta = 1$ (from Lemma 1)

Combining i) and iii) gives $c \geq \frac{(1-2\alpha)}{\alpha(1-\alpha)}$. Recalling from Lemma 1 that $\alpha = (a - 1)/(2a)$, ii) holds for any $a > 1$. Hence the above three conditions amount to the inequality: $c \geq \frac{(1-2\alpha)}{\alpha(1-\alpha)} = \frac{4a}{a^2-1}$, where $a > 1$. By solving this inequality for a, we obtain $a \geq \frac{2+\sqrt{4+c^2}}{c}$. Hence, obtaining an $(m, km, \frac{a-1}{2a}, \frac{a+1}{2a})$-extensive graph by Lemma 2 requires that $a \geq \frac{2+\sqrt{4+c^2}}{c}$. We set $a_{\min} = \frac{2+\sqrt{4+c^2}}{c}$.

Now returning to (5), we note that the factor in front of the $N \log N$ term is given by $\frac{a\delta}{\log k}$ or $\frac{adk}{\log k}$ since the extensive graphs used in the Bassalygo-Pinsker network have degree $\delta = dk$. Since $N \log N$ is the highest order term in (5), we seek to minimize $\frac{adk}{\log k} N \log N$ with respect to a, d and k, and subject to the constraint that $a \geq \frac{2+\sqrt{4+c^2}}{c}$. An additional constraint also imposed on a and k is that $k^{i-1}at$ and k^iat, $1 \leq i \leq \log_k(N/t)$, be both squares since all explicit constructions of (m, d, k)-expanders reported in the literature that we know of have a square number of inputs. This implies that k and at must both be squares. Under these conditions, it is obvious that $k = 4$ minimizes $\frac{adk}{\log k} N \log N$. As for $a_{\min}$, Fig. 4 shows that its value increases as c decreases. While we do not know of a close form relation between d and c, most explicit constructions of (m, d, c)-expanders suggest that d increases with increasing c as seen in Table I. Among these expanders, Galil *et al.*'s $(m, 9, 0.449)$-expander with $c = 0.412, d = 9$ and $a = 9.8$ yields the minimum value for $\frac{adk}{\log k}$ resulting in $\frac{adk}{\log k} N \log N = 352.8N \log N$ edges. Using two copies of $F_{9.8,N}$ networks and invoking (5) with $a = 9.8, d = 9, k = 4$ and $t = 5$ then gives a nonblocking network with $\lceil 54N + 705.6N \log_4(N/5) \rceil = \lceil -765.18N + 352.8N \log N \rceil$ edges and $2+2\log_4(N/5) = 2+\log(N/5)$ stages. Table II compares the edge-count and number of stages of this network with previously known nonblocking networks. It is seen that the edge-complexity of this network is about seven times higher than the edge-complexity of Bassalygo–Pinsker's nonblocking network construction, while they almost have the same depth. However, as it is already noted, the latter construction is not explicit, but it only points out the existence of a network with the edge and depth complexities stated in the table. As for the other entries in the table, the first three networks all have a higher order edge-complexity than the network described here. The only other nonblocking network construction listed in the table with $O(N \log N)$ edges and $O(\log N)$ depth is the multi-Beneš network [1], but the constant in its edge complexity is much larger than the constant that appears in the edge complexity of our network.[6]

III. Setting and Abolishing Paths

In this section we present parallel algorithms to set a path between any pair of idle inputs and idle outputs, and abolish paths between any number of pairs of busy inputs and

[5] The constant factor in the $N \log N$ expression was subsequently reduced to 53.4 in [2] by refining the notion of extensive graphs and choosing the values of the parameters more judiciously.

[6] The constant 240 000 in the edge complexity of the multi-Beneš network is worked out from an informal remark in the second paragraph of the second column in [1, p. 156]. The paper does not provide an expression for either the edge complexity or the depth of the network. We inferred from its description that it has $8d^2LN \log N$ edges and $2(\log NL) + 1$ stages where d is the in-degree (and out-degree) of the switches in the network and L is the loading factor. They together determine the routing time of the network. The authors stated that choosing $d < 10$ and $L < 300$ leads to a routing time of $100 \log N$, and since no lower bound were given for d and L, this implies that the requisite multi-Beneš network could have as many as $8 * 10^2 * 300 * N \log N = 240,000N \log N$ edges.

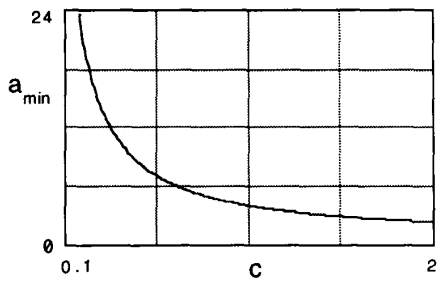


Fig. 4. Values of $a_{\min}$ with respect to c.

TABLE I
VARIOUS EXPANDERS AND THEIR DEGREES AND EXPANSION COEFFICIENTS

Expander	Degree (d)	Expansion coefficient (c)	$a_{\min}$
Margulis [10]	5	Not known	Not known
Gabber & Galil [7]	5	$(2-\sqrt{3})/4$	59.7
Gabber & Galil [7]	7	$(2-\sqrt{3})/2$	29.9
Galil *et al.* [12]	9	0.412	9.8
Galil *et al.* [12]	13	0.465	8.7
C. Y. Lee [8]	33	0.868	4.8

TABLE II
COST AND DEPTH OF VARIOUS NONBLOCKING NETWORKS

Network	Cost	Depth
Clos–Cantor [5]	$N\log^a N, 2 < a \leq 3$	$\log^a N, 2 < a \leq 3$
Cantor [5]	$O(N\log^2 N)$	$2\log(N+2)-1$
Pippenger & Lin [9]	$O(N\log^2 N)$	$2\log N+\log\log N)-3$
Multi-Beneš [1]	$240\,000N\log N+O(N)$	$2(\log 300N)+1$
Bassalygo & Pinsker [2]	$53.4N\log N+O(N)$	$2+\log(N/17)$
This paper's network	$352.8N\log N+O(N)$	$2+\log(N/5)$

outputs in a Bassalygo-Pinsker network. In the description of these algorithms, we will combine the extensive graphs whose outputs are merged together in each stage of the $F_{9.8,N}$ network into a single bipartite graph of degree dk as illustrated in Fig. 5 for $t = 5, a = 9.8$, and $k = 4$. This simplifies the representation of the $F_{9.8,N}$ network without altering its structure. We further illustrate the construction of a nonblocking Bassalygo–Pinsker network in Fig. 6 for $N = 80, t = 5, a = 9.8$, and $k = 4$. This network comprises three different types of graphs. The trapezoidal boxes marked with B_1 are 5-input, 49-output, 27-homogeneous graphs, the rectangular boxes marked with B_2 are merged blocks of four (49, 196, 0.449, 0.551)-extensive graphs and the rectangular boxes marked with B_3 represent merged blocks of four (196, 784, 0.449, 0.551)-extensive graphs.

The single assignment routing problem for a Bassalygo-Pinsker network includes two main tasks: setting paths and abolishing paths. First, we formalize the path-setting problem. Let x be an idle input which requests to be connected to an idle output, say y. A free path between x and y (a path between x and y comprising unused switching vertices) will be established by traversing the left and right $F_{a,N}$ networks separately. That is, traversals from x to the idle outputs of the left $F_{a,N}$ network will be combined with the traversals from y to the idle inputs of the right $F_{a,N}$ network to determine the free paths between x and y.

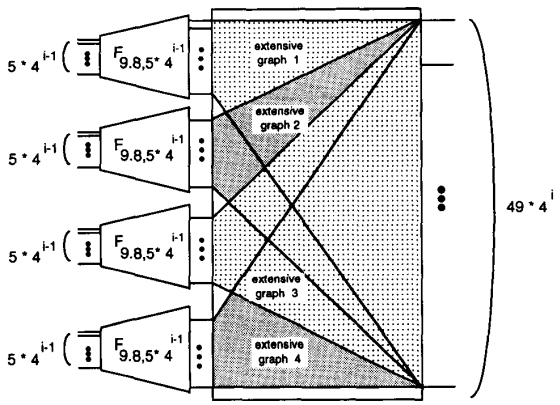


Fig. 5. Restructured $F_{9.8,5*4^i}$ network.

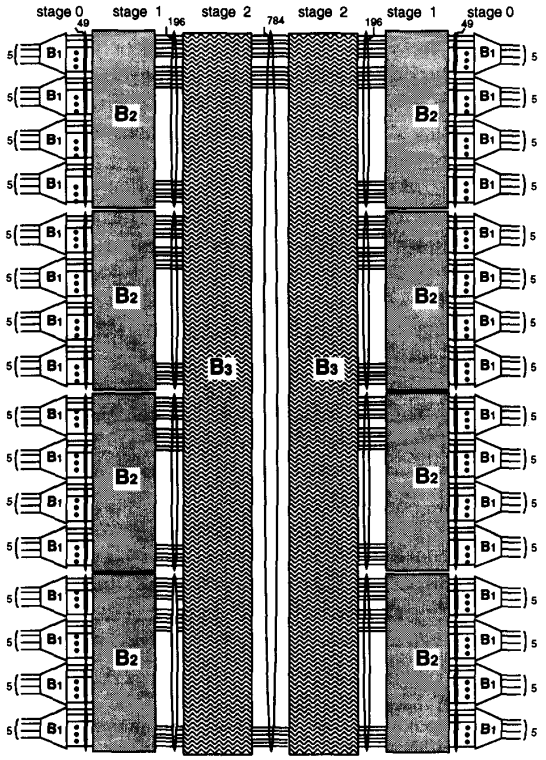


Fig. 6. 80-input Bassalygo-Pinsker network.

The path abolishing problem (single routing assignment version) for a Bassalygo–Pinsker network is concerned with dismantling an established path between a busy input and its busy output pair. The paths between inputs and outputs must be abolished after the transactions between them are over because leaving the edges on these paths busy invalidates the nonblocking property of a Bassalygo Pinsker network (any nonblocking network for that matter). We also note that the paths that need to be abolished never overlap, and hence they can be abolished in parallel without any additional time penalty.

A. Path Representation

We will represent the paths between the inputs and outputs of each $F_{a,N}$ network in terms of sequences of vertices where each vertex identifies an (output, input) pair (i.e., a link between two consecutive stages) of the network. This can be viewed as collapsing the outputs of each stage with the inputs of the succeeding stage without altering their original ordering. We denote the ith vertex between the $j-1$th stage and jth stage by (i,j) and call (i,j) the ID of this vertex, $0 \leq i \leq aN$ and $1 \leq j \leq \log_k(N/t)$. In particular, $(i,0), 0 \leq i \leq aN-1$, denote the input vertices and $(i, 1+\log_k(N/t)), 0 \leq i \leq aN-1$, denote the output vertices.

A vertex in an $F_{a,N}$ network is said to be occupied if it falls on a path established between a busy input and a busy output; it is called unoccupied otherwise. The status of the vertices in an N-input $F_{a,N}$ network with parameters a, t, d and k will be represented by an $aN \times (2+\log_k(N/t))$ status matrix P, where each entry $P[i,j]$ is a triplet $(\mu_{i,j}, b_{i,j}, S_{i,j})$, $0 \leq i \leq aN$, $0 \leq j \leq 1+\log_k(N/t)$, and[7]

i) $\mu_{i,j}$ is a location to store the ID of one of the neighbors of vertex (i,j);

ii) $b_{i,j}$ is a binary variable which represents the status of vertex (i,j) (vertex (i,j) is occupied if $b_{i,j}=1$ and it is unoccupied if $b_{i,j}=0$);

iii) $S_{i,j}$ is a dk-element vector, where $S_{i,j}[r]$ contains the ID of the rth successor of vertex (i,j), $1 \leq r \leq dk$.

For an $F_{a,N}$ network with fixed parameters a, t, d and k, the entries in $S_{i,j}$ are fixed by the structure of the specific extensive graph that is used to construct that network. The value of $b_{i,j}$ is updated after each request to establish a path or each request to abolish a path has been completed. Thus, $S_{i,j}$ and $b_{i,j}, 0 \leq i \leq aN, 0 \leq j \leq 1+\log_k(N/t)$ collectively represent the current state of the $F_{a,N}$ network.

B. Path-Setting Algorithm

Given an idle input $(I_x, 0)$ of the left $F_{a,N}$ network and an idle output $(I_y, 0)$ of the right $F_{a,N}$ network (equivalently, an idle input-output pair of the entire network containing the left and right $F_{a,N}$ networks), the path-setting algorithm consists of three phases: 1) path-claiming phase, 2) pivot-selection phase, and 3) path-tracing phase. Before the execution of these three phases, variables $\mu_{i,j}$ except $\mu_{I_x,0}$ and $\mu_{I_y,0}$, in all entries of the status matrices associated with the two $F_{a,N}$ networks are initialized with an invalid vertex ID.

1) Path Claiming Phase: During the path-claiming phase, we mark all free paths between $(I_x,0)$ and $(I_y,0)$ which are vertex-disjoint with the already established paths in the two $F_{a,N}$ networks by linking the vertices along the free paths with the variables $\mu_{i,j}$. This phase consists of $(2+\log_k(N/t))$ steps for each $F_{a,N}$ network (one step for each value of j). During the jth step, each vertex (i,j), $0 \leq i \leq aN-1$, with its variable $\mu_{i,j}$ containing a valid vertex ID in stage j of the $F_{a,N}$ network broadcasts its ID to its dk successors specified by variable $S_{i,j}$. Each vertex $(i,j+1)$, $0 \leq i \leq aN-1$, in stage $j+1$ keeps only the ID of one of its predecessors, and stores it in variable $\mu_{i,j+1}$ if its variable $b_{i,j+1}=0$ (an unoccupied vertex) and discards all the ID's from its predecessors if its variable $b_{i,j+1}=1$ (an occupied vertex).

[7]Note that each $F_{a,N}$ network has only N vertices in its input stage even though matrix P allocates a column of aN entries for these vertices. This is done to simplify the notation in our discussion.

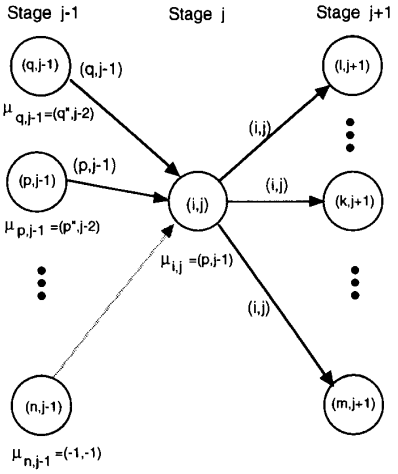


Fig. 7. Demonstration of the jth iteration of path-claiming phase.

Fig. 7 illustrates these activities. Vertex (i,j) which has received the ID from vertex $(p,j-1)$ in the last step broadcasts its ID to all its successors, i.e., vertices $(l,j+1)$, $(k,j+1)$ and $(m,j+1)$. These vertices then store in the same step, this ID in $\mu_{l,j+1}$, $\mu_{k,j+1}$ and $\mu_{m,j+1}$ if their corresponding variables $b_{l,j+1}$, $b_{k,j+1}$, $b_{m,j+1}$ are equal to 0. We note that vertex $(n,j-1)$ does not transmit its ID to vertex (i,j) as indicated by the dashed line because its variable $\mu_{n,j-1}$ does not contain a valid ID.

It follows that upon applying the path-claiming phase to the left $F_{a,N}$ network, all its idle output vertices that have free paths to the chosen idle input $(I_x,0)$ can be determined. Likewise, upon applying the same procedure to the right $F_{a,N}$ network, all its idle input vertices that have free paths to the chosen idle output $(I_y,0)$ can also be determined. It is noted that a vertex may have several successors as well as several predecessors. The path-claiming procedure assigns one predecessor to each vertex.

2) Pivot Selection Phase: We call each idle vertex common to both the left and right $F_{a,N}$ networks a pivot vertex if it can be reached by a path from both input $(I_x,0)$ and output $(I_y,0)$ that is determined in the path claiming phase. That the Bassalygo–Pinsker network is nonblocking ensures that there exists at least one pivot vertex for any given idle input of the left $F_{a,N}$ network and any given idle output of the right $F_{a,N}$ network. The pivot-selection phase uses a backward traversal from pivot vertices on the input side of the right $F_{a,N}$ network toward its output $(I_y,0)$ to locate a free path. This traversal takes $1+\log_k(N/t)$ steps. More specifically, during step j, $0 \leq j \leq \log_k(N/t)$, the vertices in stage $1+\log_k(N/t)-j$ of the right $F_{a,N}$ network activated in the previous step send their ID's to their neighbors as specified in $\mu_{i,1+\log_k(N/t)-j}$. The vertices in stage $1+\log_k(N/t)-j-1$ that receive any ID's from stage $1+\log_k(N/t)-j$ retain only one of

these ID's and then the same step is repeated between the vertices in stage $1 + \log_k(N/t) - j - 1$ and those in stage $1 + \log_k(N/t) - j - 2$ and so on. This phase generates a free path, (linked by variables $\mu_{i,j}$) between output $(I_y, 0)$ of the right $F_{a,N}$ network and one of the pivot vertices on its input side.

3) Path Tracing Phase: Once the pivot-selection phase is completed, all that remains to be done is to establish a free path by tracing it back from output vertex $(I_y, 0)$ through the pivot vertex in the center to the input vertex $(I_x, 0)$ of the combined network. This path tracing phase takes $2 + \log_k(N/t)$ steps on each of the left and right $F_{a,N}$ networks. We start out with the right $F_{a,N}$ network and make the idle output $(I_y, 0)$ as the only marked vertex. In the jth step, the marked vertex (i, j) in stage j of the right $F_{a,N}$ network sets variable $b_{i,j} = 1$ to indicate that vertex (i, j) is occupied and transfers its ID to its neighbor specified by variable $\mu_{i,j}$ so that the edge between them is activated. The unique vertex in stage $j + 1$ which receives this vertex ID becomes the marked vertex in stage $j + 1$ for the following step. After $2 + \log_k(N/t)$ steps, a particular pivot vertex in the center stage is marked. The same process is then repeated for the left $F_{a,N}$ network for another $2 + \log_k(N/t)$ steps starting with the chosen pivot vertex as marked vertex. At the end of this phase, a path is formed between $(I_x, 0)$ and $(I_y, 0)$ and the request is served.

C. Path-Abolishing Algorithm

The algorithm to abolish a path is much simpler. Given a busy input $(I_x, 0)$ of the left $F_{a,N}$ network and a busy output $(I_y, 0)$ of the right $F_{a,N}$ network, abolishing the path between them only takes one phase which consists of $2 + \log_k(N/t)$ steps on each $F_{a,N}$ network. At the beginning of this algorithm, two vertices associated with input $(I_x, 0)$ and input $(I_y, 0)$ are marked. In the jth step, marked vertex (i, j) in stage j of the $F_{a,N}$ network checks its dk successors specified in the dk-element vector $S_{i,j}$ to see which one is occupied. It then sends out its ID to the occupied successor in stage $j+1$. The successor vertex then becomes marked vertex in stage $j+1$ and resets its busy variable to 0 to indicate that it is no longer occupied. The same step is now repeated in stage $j + 1$ and so on until all the edges on the path are marked free. At the end of this phase, the originally established path is abolished and the request is completed. As stated before, this algorithm can be extended to handle multiple requests at no additional time penalty since the busy paths are all disjoint and therefore can be abolished in parallel.

IV. Realization and Performance

In this section we discuss the implementation and performance of the path-setting and path-abolishing algorithms on parallel processors with three different topologies. The first two of these implementations are derived directly from the topology of the Bassalygo-Pinsker network and the third is based on the perfect shuffle network.

A. Direct Realization

In this case, all phases of the routing algorithm associated with the two $F_{a,N}$ networks presented in the previous section

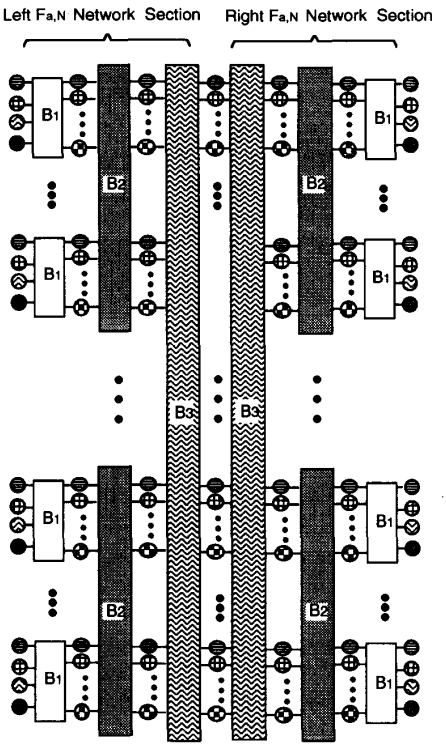


Fig. 8. A Bassalygo–Pinsker (BP)-processor.

are mapped directly onto a parallel processor with $2N + 2aN \times \log_k(N/t) + aN$ processors that are interconnected exactly the same way as the vertices are connected in the Bassalygo-Pinsker network.[8] This is illustrated in Fig. 8 for the 80-input Bassalygo-Pinsker network. The boxes marked with B_1, B_2 and B_3 represent the interprocessor communication links that correspond to the links marked with the same labels in Fig. 6. We shall refer to this parallel processor realization as a BP-processor since its topology is patterned after the Bassalygo–Pinsker network.

Each processor in a BP-processor except those associated with the input vertices and output vertices has $2dk$ communication links connecting it to its neighboring processors. Since d and k are constants, the number of communication links for each processor is a constant and the total number of communication links is

$$2dkN + 2dkaN \log_k(N/t) = O(N \log N). \quad (6)$$

The path-setting algorithm described in the previous section can be realized on the BP-processor using either a centralized or a distributed processing scheme. In the centralized scheme, we assume that a master control unit initiates the various phases of the routing algorithm. To form a path, between an idle input x and an idle output y, the master control unit activates the processor associated with input x and the processor associated with output y. Each of these two processors then

[8]Note that each $F_{a,N}$ network has only N input vertices. Thus the first stage of the parallel processor for each $F_{a,N}$ network consists of only N processors and each of the remaining stages encompasses aN processors.

simultaneously initiates a path-claiming phase. Once the path-claiming phase is completed, the processors in the center stage invoke a pivot-selection phase. After this phase is completed, the processor associated with output y then invokes a path tracing phase. At the end of this phase, a path is formed between input x and output y.

In the distributed scheme, the request for a connection between an idle input and an idle output arrives directly at the processor associated with the idle input and this request must be transmitted to the processor associated with the idle output. This is accomplished by broadcasting the destination address of the idle output via its processor to the processors associated with all the idle outputs. The processor (associated with an idle output) whose destination address matches the broadcast address is then activated to initiate the three phases of the path-setting algorithm. These three phases are also carried out by the processor associated with the idle input. The rest of the realization proceeds as in the centralized scheme.

It follows that all three phases of the path setting algorithm can be completed in $4 * (2 + \log_k(N/t)) = O(\log N)$ steps on a BP-processor under the centralized scheme and in $6 * (2 + \log_k(N/t)) = O(\log N)$ steps under the distributed scheme. Similarly, it can be shown that the path abolishing algorithm can also be realized on the same parallel processor in $2 + \log_k(N/t) = O(\log N)$ steps under the centralized scheme and $3 * (2 + \log_k(N/t)) = O(\log N)$ steps under the distributed scheme.

B. Indirect Realizations

The parallel processor realization just described can be simplified by combining some of the processors together and restructuring the communication links between them so as to maintain the connectivities in the original topology of the BP-processor. This can lead to a variety of realizations with centralized routing schemes for the Bassalygo-Pinsker network. One possibility is to combine the processors for the right $F_{a,N}$ network with the corresponding processors for the left $F_{a,N}$ network (see Fig. 9(a)). This results in halving the number of processors in the original topology. The path-claiming phase of the routing algorithm can be executed on this contracted BP-processor in a tandem fashion.

A more radical contraction is achieved by collapsing all the processors into a single column of aN processors (all processors in the same row are contracted into a single processor), and restructuring the communication links so that if any two processors have a direct communication link before the contraction, they have a direct communication link after the contraction as well. Fig. 9(b) depicts this contraction graphically. Suppose that the aN processors in this realization are numbered $0, 1, 2, \ldots, aN - 1$. Then row i of the status matrices associated with the two $F_{a,N}$ networks now resides in processor i. After the contraction, each vertex in a stage of the BP-processor is connected to dk successors in the succeeding stage. Therefore, each processor in the contracted BP-processor has $dk(2 + \log_k(N/t))$ communication links connecting it to the other processors. Thus, the contracted BP-processor consists of $O(N)$ processors and a total of $O(N \log N)$ communication links.

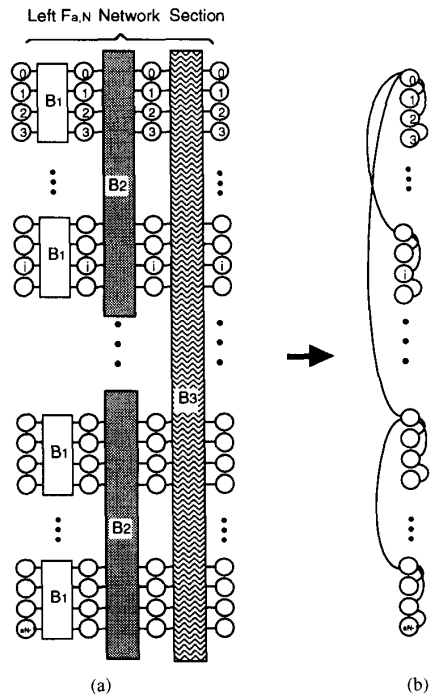


Fig. 9. Two contractions of the BP-processor. Processors marked with the same number in (a) are collapsed into a single processor in (b).

Now consider the execution of the path-setting algorithm on this contracted BP-processor. During the jth step of the path-claiming phase, processor i, $0 \leq i \leq aN - 1$, with its variable $\mu_{i,j}$ containing a valid ID broadcasts its ID to its dk neighboring processors specified by variable $S_{i,j}$. Processor $i, 1 \leq i \leq aN$ stores any one of the ID's it receives in $\mu_{i,j+1}$ if $b_{i,j+1} = 0$ and discards all the ID's if its variable $b_{i,j+1} = 1$. After two path-claiming phases for the left and right $F_{a,N}$ networks, each processor knows whether or not it is a pivot vertex. During the jth step of the pivot-selection phase, the marked processor i transfers its ID to the processor specified by variable $\mu_{i,j}$ and then updates the $\mu_{i,j}$ with the ID it received during the last step. The processor which receives any valid ID's then stores only one of these ID's temporarily and becomes a marked processor for the next step. After $1 + \log_k(N/t)$ steps, a unique free path linked $\mu_{i,j}$ is found from one of the pivot vertices to the idle output pair. Combining this path with the paths found in the left $F_{a,N}$ network during the path-claiming phase, we trace back a free path from the idle output pair to the idle input. In the jth step of the path-tracing phase for right $F_{a,N}$ network, the marked processor i sets variable $b_{i,j} = 1$ to indicate that vertex (i, j) of the right $F_{a,N}$ network is occupied and transfers its ID to the processor specified by variable $\mu_{i,j}$ so that the specified processor can use this ID to activate the corresponding edge in the right $F_{a,N}$ network. The processor which receives this vertex ID then becomes a marked processor for the next step. The process for the left $F_{a,N}$ network is the same as the process of the right $F_{a,N}$ network. At the end of this phase, a free path is established and the request is completed.

It follows from the ongoing discussion that the path-setting algorithm can be implemented in $5 * ((2 + \log_k(N/t))) = O(\log N)$ steps on the contracted BP-processor. The path-abolishing algorithm can similarly be implemented with a time complexity of $2 * (2 + \log_k(N/t)) = O(\log N)$ steps.

Now that the realization of the routing algorithm is reduced to data broadcasting on a single column of $O(N)$ processors, the same algorithm can be realized on other parallel computers consisting of $O(N)$ processors. In particular, we can realize this routing algorithm on a perfect shuffle processor using the data broadcast algorithm of Nassimi and Sahni [11]. Consider a perfect shuffle processor with aN processors, and suppose that processor $P(i)$ contains an index register $W(i)$ and data register $D(i)$. Nassimi and Sahni described an algorithm, called random access write (RAW) that broadcasts the contents of $D(i)$ in processor $P(i)$ to processor $P(W(i))$, $0 \leq i \leq aN - 1$. If two or more processors attempt to broadcast to the same processor, that is, if $W(i_1) = W(i_2) = \cdots = W(i_r) = i$, then $P(i)$ receives its data from $P(j)$, where $j = \text{Min}_{1 \leq k \leq r}\{i_k\}$. This algorithm takes $O(\log^2 N)$ steps to execute on a perfect shuffle processor. The various phases of the routing algorithms described in Section 3 can be broken down into a sequence of steps each of which amounts to executing the Nassimi and Sahni's data broadcast algorithm. To see this, consider the path-claiming phase of the path-setting algorithm. In the BP-processor realization, each processor within a stage broadcasts its own ID to its dk successors in the next stage. On the receiving end, each processor keeps only one of the ID's that reach it. This broadcasting of ID's between the processors in consecutive stages can be performed by iterating the Nassimi and Sahni's RAW algorithm dk times, where, during each iteration, all active processors send their ID's to one of their successors. Since each iteration takes $O(\log^2 N)$ steps and a total of dk iterations are needed to complete the broadcast of the ID's for all active processors during each step of path-claiming algorithm and since the entire algorithm encompasses $O(\log N)$ steps, the path-claiming phase can be completed in $O(\log^3 N)$ steps on a perfect shuffle processor with $O(N)$ processors using Nassimi and Sahni's algorithm. It should be noted that this realization increases the time complexity from $O(\log N)$ to $O(\log^3 N)$ when compared to the fully-contracted BP-processor, but it only requires $O(N)$ communication links as compared to $O(N \log N)$ communication links for the fully-contracted BP-processor.

The steps in both direct and indirect realizations of our algorithm involve broadcasting, updating vertex ID's and checking binary variables. In Section III-A, we stated that a switching vertex in a Bassalygo–Pinsker network will be given or assigned a pair (i, j) as its ID. For an N-input Bassalygo-Pinsker network this implies that the ID of each vertex takes up $O(\log N)$ bits, and hence broadcasting and updating ID's would require $O(\log N)$ bit-steps. Fortunately, the bit-level complexity of the steps in our algorithms can be reduced to $O(1)$ by noting that each vertex in Bassalygo–Pinsker network has only $2dk$ neighbors and thus it suffices to use $2\log(dk) = O(1)$ bits to identify the neighbors of a vertex. Therefore, broadcasting a vertex ID reduces to setting a single-bit flag and identifying the neighbor that sets a single-bit flag reduces to encoding a $\log(dk)$-bit address which can be done in $O(\log^2(dk)) = O(1)$ bit-steps. Recalling that the three phases of the path-setting algorithm requires $4*(2+\log_4(N/5))$ steps, the total bit-level time complexity of this algorithm will be $\approx 4 * \lceil \log(dk) \rceil^2 * (2 + \log_4(N/t)) = 72 \log N + 120.82$ when $d = 9$, $k = 4$ and $t = 5$.

TABLE III
PROCESSOR, TIME, AND LINK COMPLEXITIES OF VARIOUS PARALLEL PROCESSOR REALIZATIONS OF THE ROUTING ALGORITHM

Realization	No. of processors	Execution time	No. of Links
BP-processor	$O(N \log N)$	$O(\log N)$	$O(N \log N)$
Contracted BP-processor	$O(N)$	$O(\log N)$	$O(N \log N)$
Perfect shuffle processor	$O(N)$	$O(\log^3 N)$	$O(N)$

V. CONCLUDING REMARKS

In this paper, we first described a strictly nonblocking network with $-765.18N + 352.8N \log N$ crosspoints and $2 + \log(N/5)$ depth by combining Bassalygo and Pinsker's implicit nonblocking network construction with Galil et al.'s expanders. We then presented algorithms to set and abolish paths on this network. For each new request each of these algorithms takes $O(\log N)$ steps, where each step involves broadcasting and checking a constant number of bits on a parallel processor with $O(N \log N)$ processing elements interconnected by a topology that is identical to the Bassalygo–Pinsker network. We also established that the same algorithms can be realized on an N-processor computer with $O(N \log N)$ communication links in $O(\log N)$ steps if the processing elements are interconnected by a contracted Bassalygo–Pinsker network and in $O(\log^3 N)$ steps if they are interconnected by a perfect shuffle network. These results are summarized in Table III. It is worth noting that the constants hidden in the routing time complexities are reasonably small ($\approx$72). In contrast, the routing algorithm described in [1] achieves a constant factor of 100 but at the expense of a very large constant factor in the crosspoint complexity of the nonblocking network used (see Table II).

While these results are rewarding, it will be worthwhile to further reduce the constant 352.8 in the crosspoint expression of the nonblocking network described in the paper. This would require new constructions of expanders with lower densities and larger expansion coefficients. Another direction for further research is to extend the path setting algorithm of this paper to handle multiple connection assignments. Such assignments can be handled by iteratively applying the algorithm given in this paper, but this is likely to lead to excessive routing time when the number of requests gets very large. These problems and other related questions will be dealt with in detail elsewhere.

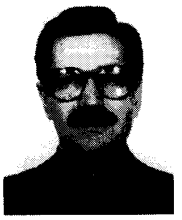
REFERENCES

[1] S. Arora, T. Leighton and B. Maggs, "On-line algorithms for path-selection in a nonblocking network," in *Proc. ACM Symp. Theory of Comput.*, 1990, pp. 149–158.
[2] L. Bassalygo, "Asymptotically optimal switching circuits," *Problems of Inform. Transmission*, vol. 17, pp. 206–211, July–Sept. 1981.

[3] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965.
[4] L. Bassalygo and M. Pinsker, "Complexity of optimum nonblocking switching without reconnections," *Problems of Inform. Transmission*, vol. 9, pp. 64–66, Jan.–Mar. 1974.
[5] D. G. Cantor, "On non-blocking switching networks," *Networks*, vol. 1, pp. 366–377, 1971.
[6] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, vol. 32, pp. 406–425, Feb. 1953.
[7] O. Gabber and Z. Galil, "Explicit constructions of linear sized superconcentrators," *J. Comput. Syst. Sci.*, vol. 22, pp. 407–420, 1981.
[8] C. Y. Lee, "Networks for fast efficient unicast and multicast communications," Ph.D. Dissertation, Univ. Maryland, Dec. 1992.
[9] G. Lin and N. Pippenger, "Parallel algorithms for routing in nonblocking networks," in *Proc. ACM Symp. Theory of Comput.*, 1991, pp. 272–277.
[10] G. A. Margulis, "Explicit construction of concentrators," *Problems of Inform. Transmission*, vol. 9, pp. 325–332, 1973.
[11] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers," *IEEE Trans. Comput.*, vol. C-30, pp. 101–107, Feb. 1981.
[12] Z. Galil, N. Alon, and V. D. Milman, "Better expanders and superconcentrators," *J. Algorithms*, vol. 8, pp. 337–347, 1987.

Fong-Chih Shao received the B.Sc. degree in communications engineering from National Chiao-Tung University, XingChu, Taiwan, in 1981, and the M.Sc. degree in electrical engineering from National Taiwan University, Taipei, in 1983.

From May 1983 to June 1990, he served as an assistant researcher at Nuclear Energy Council of Executives, Yuan, Taiwan. Since 1990, he has been a Ph.D. student in the Department of Electrical Engineering, University of Maryland, College Park. His research interests include computer communication networks, parallel processing, computer architecture, computer algorithms, and signal processing.



A. Yavuz Oruç (S'81–M'81–SM'92) received the B.Sc. degree in electrical engineering from the Middle East Technical University, Ankara, Turkey, in 1976, the M.Sc. degree in electronics form the University of Wales, Cardiff, U.K., in 1978, and the Ph.D. degree from Syracuse University, Syracuse, NY, in 1983.

Since January 1988, he has been an associate professor in the Department of Electrical Engineering, University of Maryland, College Park. Prior to joining the University of Maryland, he was on faculty of the Department of Electrical, Computer, and Systems Engineering at Rensselaer Polytechnic Institute, Troy, NY. His research interests include parallel computer and communication systems.

Dr. Oruç is a member of IEEE Communications, Computer, and Information Theory Societies.