Generalized Connection Networks for Parallel Processor Intercommunication

CLARK D. THOMPSON

Abstract—**A generalized connection network (GCN) is a switching network with N inputs and N outputs that can be set to pass any of the N^N mappings of inputs onto outputs. This paper demonstrates an intimate connection between the problems of GCN construction, message routing on SIMD computers, and "resource partitioning." A GCN due to Ofman [7] is here improved to use less than $7.6N \log N$ contact pairs, making it the minimal known construction.**

Any GCN construction leads to a new algorithm for the broadcast of messages among processing elements of an SIMD computer when each processing element is to receive one message. Previous approaches to message broadcasting have not handled the problem in its full generality. The algorithm arising from this paper's GCN takes $8 \log N$ (or $13N^{1/2}$) routing steps on an N element processor of the perfect shuffle (or mesh-type) variety.

If each resource in a multiprocessing environment is assigned one output of a GCN, private buses may be provided for any number of disjoint subsets of the resources. The partitioning construction derived from this paper's GCN has $5.7N \log N$ switches, providing an alternative to "banyan networks" with $O(N \log N)$ switches but incomplete functionality.

Index Terms—**Array processors, connection networks, message broadcasting, parallel algorithms, parallel processing, resource partitioning, SIMD machines.**

I. Introduction

A GENERALIZED connection network (GCN) is a switching network with N inputs and N outputs capable of implementing any mapping of inppts onto outputs. In other words, each output may be connected to any one of the inputs for a total of N^N different connection patterns. Thus, a GCN is more powerful than the connection networks of Beizer [2] and Benes [3] *et al.*, for a connection network handles only one-to-one mappings of inputs onto outputs ($N!$ settings).

In many situations, two parameters of a GCN design are of paramount importance: its delay and the number of contact pairs used in its construction. The delay of a GCN is defined to be the maximum number of contact pairs separating any input–output pair. There exists a tradeoff between these two parameters, as evidenced by Table I (all logarithms in this paper are base 2).

(The delays quoted for the last two entries are derived from Pippenger's proof technique [10]; it may be possible to improve these delays without affecting the asymptotic number of contact pairs.) This paper's construction is seen to be a small improvement on Ofman's construction [7], on which it is based.

Any GCN construction leads to an algorithm for the transfer of data among processing elements of an SIMD (single instruction stream multiple data stream) computer. This data transfer is modeled as the routing of messages, each originating at a processing element and destined for some subset of the other processing elements. There have been many papers treating particular message routing patterns on particular networks (Stone [12], Siegel [11], Orcutt [8], etc.). The algorithm based on the GCN of this paper performs near-optimally on any message broadcasting pattern in which each processing element receives one message on several popular SIMD interconnection networks. For an N element computer (N a power of 2), the algorithm requires $13N^{1/2}$ routing steps on a square mesh-type array, $8 \log N$ routing steps on the perfect shuffle, PM2I, and WPM2I networks, and $4 \log N$ routing steps on the Cube (see Section III for descriptions of these networks). All other known GCN constructions lead to slower routing algorithms.

Finally, any GCN construction applies to the partitioning of multiprocessor systems in the sense of Goke and Lipovski [4]. If each resource is assigned one output of a GCN, proper switch settings provide a private conductive path for each of any number of disjoint subsystems. The banyan networks originally proposed for this task do not implement all partitions when $O(N \log N)$ switches are employed. When used for partitioning, $\frac{1}{4}$ of this paper's GCN can be omitted, so that unrestricted partitioning may be obtained with $5.7N \log N$ switches. No other known GCN construction leads to smaller partitioners.

The new GCN construction is described in Section II, its application to message routing is elaborated in Section III, and its related partitioning network is derived in Section IV.

II. A GCN Construction

A GCN may be represented as a graph with one edge for each contact pair (SPST switch). The N input terminals become N vertices, as do the N output terminals. Other vertices are added as necessary as "tie-posts" for switch leads. For example, an $N \times N$ crosspoint switch is a GCN with N^2 contact pairs; its graph is the complete bipartite graph on $2N$ vertices and N^2 edges.

The setting of a GCN can be formalized as a sequence of N integers, one for each output vertex: $j_1, j_2, \cdots, j_N$ where $j_k = i$ iff output number k is connected to input number i (each output is connected to exactly one input). For example, if $N = 4$ a GCN setting might be (3, 3, 4, 1): input 3 is

Manuscript received August 10, 1977; revised April 10, 1978. This work was supported in part by the National Science Foundation under Grant MCS 75-222-55 and the Office of Naval Research under Contract N00014-76-C-0370, NR 044-422.

The author is with the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

0018-9340/78/1200-1119$00.75

TABLE I
COMPARISON OF CURRENT GCN's

Description	# of contact pairs	delay
N × crosspoint	N^2	1
Masson and Jordan's GCN [6]	$O(N^{5/3})$	3
Ofman's GCN [7]	10N log N	5 log N
Pippenger's GCN [9]	7.6N log N	O(N log N)
This paper's GCN	7.6N log N	3.8 log N
Non-constructive upper bound on # of contact pairs [10]	3.8N log N	$(O(\log^2 N))$
Lower bound on # of contact pairs [10]	1.9N log N	$(O(\log^2 N))$

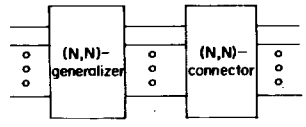

Fig. 1. Schema for a GCN construction.

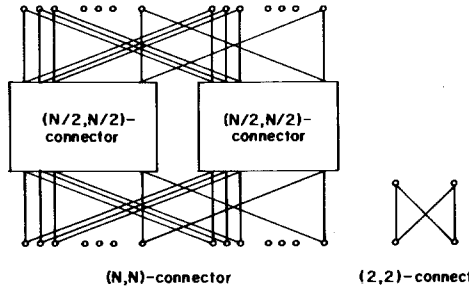

Fig. 2. Beizer's connection network (recursive construction).

connected to outputs 1 and 2, input 4 to output 3, and input 1 to output 4.

A graph is a GCN iff it contains a subgraph with proper connectivity for each of the N^N possible j_k sequences. The edges in these subgraphs correspond to the switches that should be closed to realize each GCN setting.

A GCN construction may be obtained from the schema shown in Fig. 1 (Ofman [7]).

The left-hand network produces the correct number of copies of each of the inputs, which are then permuted to the proper outputs by the right-hand network. For the j_k sequence (3, 3, 4, 1), the left-hand network (the generalizer) must have two copies of input 3 somewhere on its outputs, one copy each of inputs 4 and 1, and no copies of input 2. These signals are connected to the proper GCN outputs by the right-hand connection network.

It is now necessary to examine connection and generalization networks in more detail.

A. Connection Networks

An (N, N)-connection network is a switching network with N inputs and N outputs capable of passing any of the $N!$ one-to-one mappings (permutations of inputs onto outputs). This is, of course, strictly less powerful than a GCN, in which the same input may be connected to more than one output at a time (in terms of the j_k notation developed above, connection networks operate on sequencies $j_1, \cdots, j_N$ in which the j_k's are distinct).

Beizer [2] published the $4N \log N - 2N$ construction of Fig. 2 in which an N-input connection network is synthesized from 2 $N/2$-input connectors and $4N$ additional contact pairs.

The proper switch settings for any desired connection pattern may be found by the method of Waksman [14] in $O(N \log N)$ time on a serial computer, the best result known. Thus, it would seem that lengthy preprocessing time will be required for each GCN setting. In some cases, it may be feasible to tabulate precomputed GCN settings, although it would seem necessary to store about one bit per switch setting or $O(N \log N)$ bits in all. This, of course, limits the practical application of networks involving Beizer's connector. Unfortunately, all known GCN's with $O(N \log N)$ contact pairs contain a Beizer-style connector.

It should be noted that this connector construction is symmetric about a horizontal axis. In fact, the top $\log N$ stages and the bottom $\log N$ stages comprise Omega networks (see Lawrie [5]) that share a common level of switches.

B. Generalization Networks

An (N, N)-generalizer passes input i to m_i different outputs where $\sum m_i = N$ and $m_i \geq 0$. Thus, it provides a particular number of copies of each input somewhere among the outputs. The existence of (N, N)-generalizers with $O(N)$ switches has been demonstrated nonconstructively by Pippenger [10]. Construction of a generalizer can be accomplished by the schema shown in Fig. 3, due to Ofman [7].

The left-hand network routes all important inputs to its uppermost output lines. More precisely, if p of the inputs will appear on some output of the generalizer, they must appear on lines k_1 through k_p of the hyperconcentrator. The right-hand network is responsible for producing the correct number of copies of each of its inputs, but there must exist some integer p such that $k_1, k_2, \cdots, k_p$ will appear in the output at least once, while $k_{p+1}, k_{p+2}, \cdots, k_N$ will be ignored. For example, an (8, 8)-generalizer setting might be $m_i = (3, 0, 1, 0, 0, 2, 1, 1)$: input 1 to be connected to three outputs; inputs 2, 4, 5 to none; inputs 3, 7, 8 to one; and input 6 to two. The corresponding hyperconcentrator–infrageneralizer decomposition would have $p = 5$, with inputs 1, 3, 6, 7, 8 appearing on lines k_1 through k_5 in some arbitrary order.

Ofman demonstrates that the network shown in Fig. 4 is an infrageneralizer.

Ofman's infrageneralizer is a little more powerful than the infrageneralizer defined above. An infrageneralizer requires its "live" input signals to appear on the first p of its input lines; Ofman's network allows the p signals to appear in any consecutive sequence of its N input lines (wraparound is permitted, so that (k_{N-1}, k_N, k_1) is a consecutive sequence of inputs). Secondly, infrageneralizers need only connect the correct number of outputs to each live input, but Ofman's network connects outputs to inputs in order. The first

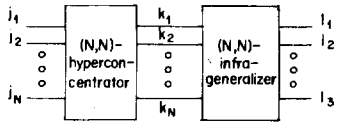

Fig. 3. Schema for a generalizer construction.

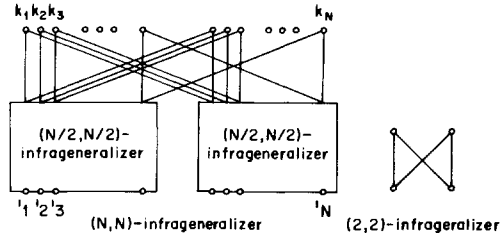

Fig. 4. Ofman's infrageneralizer (recursive construction).

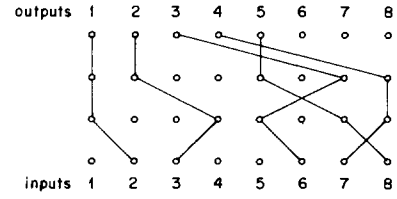

Fig. 5. An upside-down hyperconcentrator set for (2, 3, 6, 7, 8).

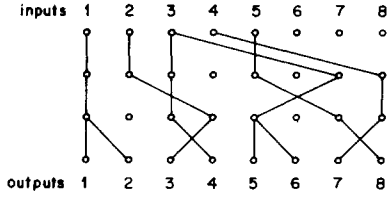

Fig. 6. An infrageneralizer set for (2, 1, 3, 1, 1, 0, 0, 0).

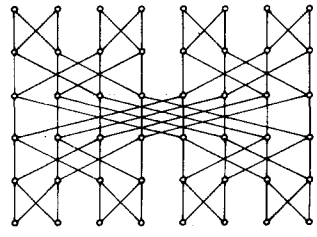

Fig. 7. An (8, 8)-generalizer.

member of the consecutive sequence of inputs is connected to the leftmost outputs; the second live input is connected to outputs just to the right of these; and the last live input is connected to the rightmost outputs.

The proper switch settings for Ofman's network may be obtained recursively by using the upper switches to give each half-sized infrageneralizer some of the live input signals. The output specifications outlined above determine which input signals are needed by each half-sized network. These signals form a consecutive subsequence of the original sequence of live inputs by virtue of the ordering of the outputs.

An (N, N)-connection network could be used for hyperconcentration since a hyperconcentrator merely permutes its inputs. This is, in fact, Ofman's approach, yielding an (N, N)-generalizer with $6N \log N$ contact pairs. Ofman's construction can be improved by using fewer switches in the hyperconcentrator portion. Somewhat surprisingly, Ofman's infrageneralizer is an "upside down" hyperconcentrator—the direction of signal flow through the network is reversed by turning inputs into outputs and vice versa. This equivalence will be verified by the demonstration of a correspondence between any desired hyperconcentration function and an infrageneralizer function. A hyperconcentration setting may be specified by a list of p integers $(n_1, n_2, \cdots, n_p)$ with $1 \leq n_1 < n_2 < \cdots < n_p \leq N$ corresponding to the indices of the inputs whose signals are to appear in the first p output lines. The corresponding infrageneralizer function is that input i should appear on m_i output lines where $m_i = n_i - n_{i-1}$, $n_0 = 0$, and $n_{p+1} = n_{p+2} = \cdots = n_N = N$. Ofman's infrageneralizer will connect input i to outputs $n_{i-1} + 1$ through n_i; if switches are opened to disconnect all but output number n_i for $1 \leq i \leq p$, then the required hyperconcentration function is implemented by the reversed infrageneralizer.

An example should clarify matters. A (8, 8)-hyperconcentrator setting for $n_i = (2, 3, 6, 7, 8)$ corresponds to a (8, 8)-infrageneralizer setting for $m_i = (2, 1, 3, 1, 1, 0, 0, 0)$. In other words, the problem of finding the proper switch settings to bring inputs 2, 3, 6, 7, and 8 to outputs 1, 2, 3, 4, and 5 (a hyperconcentration) may be solved by setting Ofman's infrageneralizer to route input 1 to outputs 1 and 2; input 2 to output 3; input 3 to outputs 4, 5, and 6; input 4 to output 7; and input 5 to output 8. See Figs. 5 and 6.

Since Ofman's (N, N)-infrageneralizer has $2N \log N$ contact pairs, an (N, N)-generalizer can be built with $4N \log N$ contact pairs by attaching an infrageneralizer to a reversed infrageneralizer (a hyperconcentrator). Since the last stage of the hyperconcentrator is identical to the first stage of the infrageneralizer, the combined functionality of these two levels of switches may be obtained with a single one, eliminating $2N$ contact pairs. The (8, 8)-generalizer obtained in this way is illustrated in Fig. 7.

C. The Complete GCN Construction

The astute reader will have noticed that the (N, N)-generalizer of Section II-B is quite similar to the (N, N)-connection network of Section II-A. In fact, one merely needs to "unshuffle" the inputs and outputs of this (N, N)-connection network to make the two networks identical. Then, when concatenating the generalization and connection networks to obtain a GCN, the first stage of the latter can be combined with the last stage of the former. This eliminates $2N$ contact pairs, yielding (for $N = 8$) Fig. 8.

The number of contact pairs in this GCN is easily counted: the generalization network "front end" has $4N \log N - 2N$ contact pairs, as does the connection network. When the two networks are concatenated, $2N$ contact pairs are eliminated, so that the complete GCN has

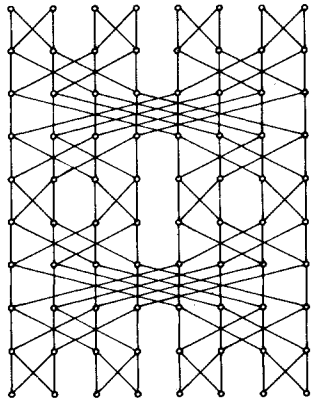

Fig. 8. An (8, 8)-GCN.

$8N \log N - 6N$ contact pairs. If further optimization is desired, $O(N)$ contact pairs may be stripped from the connector (see Waksman [14]).

The $7.6N \log N$ GCN construction claimed in the Introduction is derived from the $3.8N \log N$ connection network of Benes [3] (N is a power of 3; three-way branching is used throughout). These three-way connection networks have the same property as the two-way Beizer networks used until now in this paper. The first half (front end) may be set to act as a hyperconcentrator, and the second half can be used as an infrageneralizer. Thus a $7.6N \log N$ contact pair GCN may be formed by concatenating two of these connectors. This new GCN does not seem to be any easier to "set up" than the two-way branching constructions derived above, limiting its applications in a similar fashion.

III. Message Broadcasting

An SIMD computer may be considered to consist of three major parts: a central control unit, the processing elements, and an interconnection network. Each PE (processing element) operates on data in its own local memory according to the dictates of the central control unit. Data enter and leave this local memory via the interconnection network, which typically connects each PE to one of several neighboring PE's. For example, in a mesh-type computer, each PE has at most four neighbors. The situation may be depicted in Fig. 9, where the boxes are PE's and the lines are possible connections.

Note that PE's on the edges have fewer than four neighbors. The interconnection network of this mesh-connected computer may be in one of four states: U, D, L, R (Up, Down, Left, Right). When in the U position, each PE may receive one message per time unit from its downward-adjacent neighbor.

The message broadcasting problem may now be broadly stated. Initially, each PE has generated a message of interest to some (possibly empty) subset of the other PE's. Each PE is to receive exactly one interesting message. How long does it take to deliver all the messages, as a function of the total number of PE's and their interconnection pattern? Time is measured in the number of (parallel) unit-distance message routings. For simplicity, assume that no time is spent on

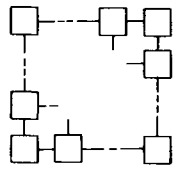

Fig. 9. A mesh-connected computer.

selecting which message (of possibly several) will be sent from each PE. This assumption is valid on a computer with a sufficiently powerful control unit (each PE is explicitly told which message to send), and is nearly valid when routing decisions are made locally (for example, by examination of "routing tags" on the messages). The algorithms of this paper will place at most two messages in a PE at a time, so these routing decisions should not be time-consuming.

It should be noted at this juncture that the routing algorithms presented in this paper will require substantial preprocessing time. As indicated in Section II-A, computation of settings for Beizer's connection network is a time-consuming process, but must be performed for each distinct message broadcasting pattern. Thus, the algorithms outlined below will be of most use when the broadcasting pattern is known at compile time. In more dynamic situations, "sorting" of destination tags is a more viable alternative (see Batcher [1] and Thompson and Kung [13]).

The next three subsections will solve the message broadcasting problem for several different interconnection networks. In all cases, N is assumed to be a power of 2.

A. Message Broadcasting on the Mesh-Connected Computer

A lower bound for the message broadcasting problem on the mesh-connected computer may easily be derived. A square N-element computer of the type depicted in Fig. 9 has $N^{1/2}$ processing elements on a side. If a message broadcasting pattern involves the exchange of messages between the upper-leftmost PE and the lower-rightmost PE, at least $4(N^{1/2} - 1)$ time units will be required. This result follows immediately from the observation that the interconnection network must be in each of its four states (U, D, L, R) for at least $(N^{1/2} - 1)$ time units. Any general algorithm for message broadcasting must be able to handle this particular message exchange, leading to a lower bound of $4(N^{1/2} - 1)$ time units.

It is not known whether more complicated broadcasting patterns require more than $4(N^{1/2} - 1)$ time units. However, a large number of patterns can be completed in that amount of time—the so-called Omega permutations (Orcutt [8]). Also, any one-to-one pattern (each message of interest to exactly one PE) can be accomplished in about $6N^{1/2}$ time when N is very large (Thompson and Kung [13]). Indeed, for these one-to-one patterns, $7(N^{1/2} - 1)$ time is sufficient for any N, as indicated later in this subsection. The main algorithm of this subsection demonstrates that no broadcast pattern need take more than $13N^{1/2} - 16$ time units.

A relationship between the graph form of a GCN and a message routing algorithm may be drawn in the following way. Each vertex of a GCN corresponds to a PE, and each

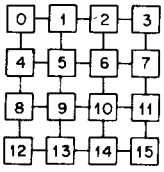

Fig. 10. Row-major indexing of a 4 × 4 mesh-connected computer.

edge to the routing of a single message from one PE to another PE. Furthermore, each of the N input vertices must correspond to a different PE, as must each of the output vertices. A message routing pattern is now exactly analogous to a GCN setting: messages (signals) originate at PE's (input vertices) and are routed to other PE's (intermediate vertices) until they reach their destinations (output vertices).

The vertex–PE correspondence is most easily made by indexing both vertices and PE's; corresponding vertex–PE pairs have the same index (from 0 to $N-1$). On mesh-connected computers, the row-major indexing scheme illustrated in Fig. 10 will be used throughout this paper.

If the 16 nodes on each level of the (16, 16)-GCN built according to Section II are numbered from left (0) to right (15), then the corresponding routing algorithm may be drawn as in Fig. 11. Note that each "stage" of the GCN corresponds to a possible interchange of messages between pairs of PE's. The first stage's interchange pairs are (0, 1), (2, 3), $\cdots$, (14, 15); this is easily seen to call for one time unit of the L interconnection setting and one of R. The second stage's pairs are (0, 2), (1, 3), (4, 6), $\cdots$, (13, 15)—two L's and two R's. For convenience, the notation (Lx, Ry) is used to denote x time units of the L setting and y time units of R.

In general, this approach on an N element computer would require 3($L1$, $R1$) routings, 4($L2$, $R2$) routings, 4($L4$, $R4$) routings, $\cdots$, 4($LN^{1/2}/2$, $RN^{1/2}/2$) routings, 4($D1$, $U1$) routings, 4($D2$, $U2$) routings, $\cdots$, 4($DN^{1/2}/4$, $UN^{1/2}/4$) routings, and 2($DN^{1/2}/2$, $UN^{1/2}/2$) routings. Note that there are four routings of every type in the list except the first and the last. The time required to complete these routings may be shown to be

$$2 * 4\left(\sum_{1 \le i \le (\log N)/2} 2(N^{1/2})/2^i\right) - 2 - 2N^{1/2} = 14N^{1/2} - 18.$$

The summand is the time taken by the routing instruction ($LN^{1/2}/2^i$, $RN^{1/2}/2^i$), which is issued four times. The leading factor of 2 accounts for the analogous ($DN^{1/2}/2^i$, $UN^{1/2}/2^i$) instructions. The "missing" ($L1$, $R1$) instruction saves 2 time units; the two missing ($DN^{1/2}/2$, $UN^{1/2}/2$) instructions save $2N^{1/2}$ time units.

The natural GCN numbering scheme used until now is not optimal. Another scheme may easily be derived in which an ($LN^{1/2}/2$, $RN^{1/2}/2$) instruction is "missing" in the stead of the previously missing ($L1$, $R1$) routing. This will save $N^{1/2} - 2$ time units, so that the optimized message broadcasting algorithm takes just $13N^{1/2} - 16$ time units. One of the numberings that leads to this result is obtained from the binary representation of the natural sequence by exchanging the least significant bit with the (log N)/2th least significant bit. For example, with $N = 16$ the numbering of GCN nodes from left to right on each stage would be (0, 2, 1, 3, 4, 6, 5, 7, 8, 10, 9, 11, 12, 14, 13, 15), and the first stage of GCN simulation would involve interchanges between PE pairs (0, 2), (1, 3), $\cdots$, (13, 15).

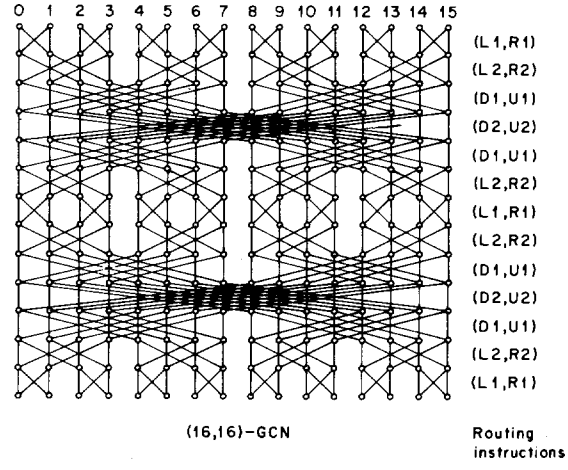

Fig. 11. Routing on a 4 × 4 mesh-connected computer.

Other GCN constructions may, of course, be simulated on a mesh-connected computer. However, none seems to lead to faster message broadcasting algorithms. The $N \times N$ crosspoint offers no structure to the problem: the simulation of its first and only stage calls for potential data movement from each PE to every other. Masson and Jordan's GCN [6] is little better because potential data movement in its three stages occurs in "neighborhoods" of $O(N^{2/3})$ or $O(N^{1/3})$ PE's. A simulation of Ofman's GCN would entail a simulation of this paper's GCN, for the latter is contained in the former. Pippenger's GCN has $O(N \log N)$ delay; hence it would require at least $O(N \log N)$ time to simulate. And the three-way branching GCN alluded to at the end of Section II involves interchanges among triplets of PE's, which is not a natural operation on a square mesh-connected computer (although it might be applicable to a "triangular mesh"-connected computer).

If a particular message distribution pattern happens to be one-to-one (each message goes to exactly one PE), then the full power of a GCN simulation is not required. Instead, a simulation of the connection network imbedded in the last half of the GCN can be accomplished in $7N^{1/2} - 8$ time units, using the natural correspondence scheme. There would be two routings of every type except ($DN^{1/2}/2$, $UN^{1/2}/2$).

B. Message Broadcasting on a Perfect Shuffle Computer

The perfect shuffle interconnection (Stone [12]) is nicely suited for message broadcasting. As demonstrated below, a GCN may be simulated, and thus any message broadcasting pattern implemented, in 8 log $N - 7$ time units.

Let the PE's of a perfect shuffle computer be numbered from 0 to $N-1$. Each index can be represented in log $N = m$ binary bits, $b_m b_{m-1} \cdots b_3 b_2 b_1$. The perfect shuffle interconnection network has just three settings, so that PE $b_m \cdots b_1$ is connected to $b_m b_{m-1} \cdots b_2 \overline{b_1}$

("exchange"), to $b_{m-1}b_{m-2}\cdots b_2b_1b_m$ ("shuffle"), and to $b_1b_mb_{m-1}\cdots b_3b_2$ ("unshuffle"). Fig. 12 illustrates a perfect shuffle computer; the "shuffle" connections may be visualized by reversing the direction of the "unshuffle" arrows.

A lower bound for the message broadcasting problem on this computer can be obtained through consideration of the time necessary to send a message from PE 0 to PE $N-1$. In binary notation, $N-1$ has $(\log N-1)$ "1" bits. The "exchange" setting is the only one that allows communication between PE's with different numbers of "1" bits, but it only connects PE's with a bit-difference (Hamming distance) of 1. So $\log N-1$ exchange connections intervene between PE 0 and PE $N-1$. Furthermore, at least one shuffle or unshuffle must be performed between each pair of exchanges (or else the second exchange connects the same PE pairs as the first). Thus, $\log N-2$ shuffle or unshuffle connections intervene between PE 0 and PE $N-1$, leading to a lower bound for message broadcasting of $2\log N-3$ time units.

A good algorithm for message broadcasting results from the careful (if nonobvious) numbering of the nodes of Section II's GCN. Let the input nodes be labeled naturally: 0 (left) through $N-1$ (right). The labelings of the next $\log N-1$ rows of GCN nodes are obtained by unshuffling the binary representation of the labels of the previous row. For example, if $N=8$, the first row is (0, 1, 2, 3, 4, 5, 6, 7), the second row is (0, 4, 1, 5, 2, 6, 3, 7), and the third row is (0, 2, 4, 6, 1, 3, 5, 7). The $(\log N)$th through the $(2\log N-1)$th rows are labeled by shuffling the indices in the previous row. In the present example, the fourth row is (0, 4, 1, 5, 2, 6, 3, 7) and the fifth row is (0, 1, 2, 3, 4, 5, 6, 7). The $(2\log N)$th through the $(4\log N-3)$th rows are labeled identically to the second through the $(2\log N-1)$th rows (e.g., the sixth through the ninth rows are identical to the second through fifth rows), while the output row (the $(4\log N-2)$th) is numbered naturally.

This GCN numbering may be motivated by considering the corresponding perfect shuffle network settings. In the example above, the first two rows are (0, 1, 2, 3, 4, 5, 6, 7) and (0, 4, 1, 5, 2, 6, 3, 7). Thus, after the first stage of GCN simulation, each of PE 0 and PE 4 should have one of the messages originally in PE 0 and PE 1; PE 1 and PE 5 should have messages from either PE 2 or PE 3; PE's 2 and 6 should have messages from PE's 4 and 5; and PE's 3 and 7 should have messages from PE's 6 and 7. This result may be obtained with only two unit-distance routing steps: an exchange and an unshuffle. The exchange transmits messages between PE's 0 and 1, PE's 2 and 3, PE's 4 and 5, and PE's 6 and 7. At this point each PE has two messages, one of which is selected to be sent out on the unshuffle connection, while the other is ignored (destroyed). After "unshuffling," each message is in the proper PE (that is, PE 0 and PE 4 have messages that originated in either PE 0 or PE 1, etc.). Succeeding stages of the GCN simulation are handled similarly. The complete GCN simulation consists of $(\log N-1)$ repetitions of (exchange, unshuffle), $(\log N-1)$ repetitions of (exchange, shuffle), $(\log N-1)$ repetitions of (exchange, unshuffle), $(\log N-1)$ repetitions of (exchange, shuffle), and one final exchange, for a total of $8\log N-7$ time units.

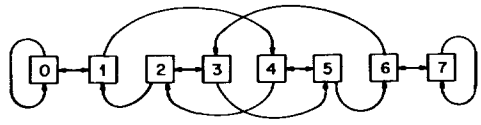

Fig. 12. Exchange (straight) and unshuffle (curved) connections on an eight-PE perfect shuffle computer.

This GCN numbering is the best possible, as may be seen from the following considerations. Each stage of the GCN consists of $N/4$ complete bipartite graphs on four nodes. The shuffle and unshuffle network connections are not in themselves sufficient to simulate any stage of the GCN since, for example, PE 0 is only connected to itself. Thus, at least one exchange step must be executed during the simulation of each GCN stage. However, a shuffle or an unshuffle must occur between consecutive exchange steps (if not, the second exchange is superfluous). Since there are $4\log N-3$ stages in this paper's GCN construction, a simulation requires $4\log N-3$ exchanges interlarded with $4\log N-4$ shuffles or unshuffles. This subsection's numbering and associated routing algorithm realizes this minimum.

For the special case of one-to-one message distribution patterns, $4\log N-3$ time units are sufficient to simulate the last half of the GCN (a connection network).

C. Message Broadcasting on Cube, PM2I, and WPM2I Computers

The nomenclature of this section is due to Siegel [11]. The Cube network is similar to the one implemented in Staran, the PM2I network is similar to Feng's Data Manipulator, while the WPM2I is Siegel's brainchild. As before, let the PE's be numbered from 0 to $N-1$ in $m=\log N$ bits: $b_mb_{m-1}\cdots b_2b_1$.

The Cube has m settings, where setting i connects $b_m\cdots b_1$ to $b_m\cdots b_{i+1}\bar{b}_ib_{i-1}\cdots b_1$. Using the natural left (0) to right $(N-1)$ numbering for the nodes on each level of the GCN, it should be clear that simulation of any stage of the GCN takes only one time unit, so that at most $4\log N-3$ time units are required by any message distribution pattern on the Cube. A lower bound is also immediate: $\log N$ time units are necessary for PE 0 to communicate with PE $N-1$, that is, one time unit for each bit that is of different value in their indices.

The PM2I network has $2\log N=2m$ settings, corresponding to the addition or subtraction mod N of 2^i for $0\le i<m$. The WPM2I connections are similar to those of the PM2I network, except any "carry" or "borrow" will "wrap around" to bit b_{i-2}. Lower bounds for message broadcasting on these computers are implicit in the work of Siegel [11]. The PM2I (or WPM2I) takes at least $\log N$ (or $(\log N)/2$) time to perform a "shuffle," which is a legal message broadcasting pattern. Upper bounds are also easily obtained. Either network can simulate a naturally numbered GCN in two time units per stage (one "addition" and one "subtraction"), giving a total of $8\log N-6$ time units for worst-case message broadcasting.

Of course, these bounds are cut almost in half for the special case of one-to-one message distribution patterns. Only $2\log N-2$ time units are required for a Cube simula-

tion of a connection network (4 log $N - 4$ time units on the PM2I or WPM2I), using the natural numbering scheme.

IV. Partitioning

The use of switching networks in the partitioning of a multiprocessing system is treated in Goke and Lipovski [4]. They propose connecting N resources to a network flexible enough to provide private buses for disjoint "subsystems" of the resources. For example, if a particular terminal, processing unit, and memory device are to be formed into an independent subsystem, the partitioning network is instructed to form a private connection between their respective I/O ports. The partitioning networks considered in this paper will merely connect appropriate I/O ports; management of the bus thereby created for each subsystem will be the responsibility of the member resources. The most straightforward partitioning network is based on an N by $N/2$ crosspoint switch: each of the N resources can be independently connected to any of $N/2$ internal buses. While this network is simple to configure and has only constant delay, it requires $O(N^2)$ switches. Another network considered by Goke and Lipovski is an (N, N)-connector whose inputs are connected to its outputs. Although this device has only $O(N \log N)$ switches, its delay may be $O(N \log N)$. Goke and Lipovski settled on "banyan networks" with $O(N \log N)$ switches and $O(\log N)$ delay, but incomplete functionality (not all partitions could be achieved). It should be clear that a GCN provides unrestricted freedom of connection between any of its N outputs. This paper's GCN construction thus immediately gives a complete partitioning network with $O(N \log N)$ switches and $O(\log N)$ delay.

Actually, a GCN is an unnecessarily complex partitioning network. The resources will only be connected to the outputs of the GCN, so that the ordering of the inputs is completely arbitrary. In terms of Section II's construction, this implies that the hyperconcentrator "front end" is superfluous and may be removed. Thus, a partitioner can be built with $8N \log N - 2N \log N = 6N \log N$ contact pairs. If further optimization is desired, Waksman's connector [14] may be used. Also, half the inputs to the infrageneralizer may be removed, since at most $N/2$ subsystems can have more than one resource. For example, Fig. 13 is an (8)-partitioner.

Setup algorithms for this network are relatively time-consuming, limiting its practicality (banyan networks can be essentially self-configuring in $0(\log N)$ time). When a new subsystem with k resources ($k > 1$) comes into existence, it is assigned the leftmost unused infrageneralizer input and the k leftmost unused connector inputs. The infrageneralizer can be configured in $0(\log N)$ time since it is a banyan. However, the connector setting may need radical changes for which the best known algorithm (Waksman [14]) requires $0(N \log N)$ time on a serial computer. Heuristic approaches to connector setting may mitigate this problem, but the author is forced to conclude that this partitioner is of little use in real-time or rapidly changing computational environments.

The three-way branching construction mentioned at the end of Section II leads to another partitioner. The concatenation of a $3.8N \log N$ contact pair connector with a $1.9N \log N$ infrageneralizer (a copy of the second half of the connector) produces a $5.7N \log N$ partitioner. Of course, the problem of setting the connector is a limiting factor in the practicality of such a partitioner.

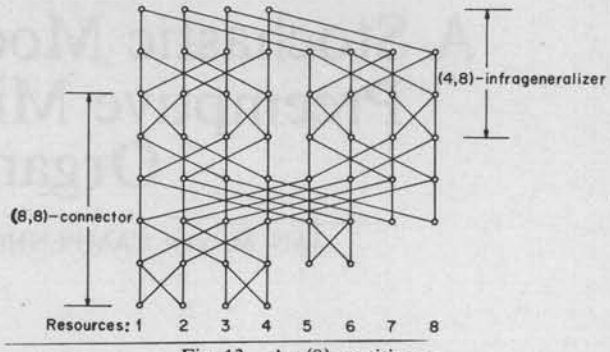

Fig. 13. An (8)-partitioner.

Acknowledgment

The author is indebted to N. Pippenger for several stimulating discussions, and to H. T. Kung for much selfless advice and help.

References

[1] K. E. Batcher, "Sorting networks and their applications," in *1968 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 32, pp. 307–314.
[2] B. Beizer, "The analysis and synthesis of signal switching networks," in *Proc. Symp. on Math. Theory of Automata*, 1962, pp. 563–576.
[3] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965, pp. 113–135.
[4] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annu. Comput. Architecture Conf.*, Gainsville, FL, 1973, pp. 21–28.
[5] D. H. Lawrie, "Memory-processor connection networks," Dep. Comput. Sci., Univ. Illinois, Urbana, Rep. 557, 1973.
[6] G. M. Masson and B. W. Jordan, Jr., "Generalized multistage connection networks," *Networks*, vol. 2, pp. 191–209, 1972.
[7] J. P. Ofman, "A universal automaton," *Trans. Moscow Math. Soc.*, vol. 14, 1965 (translation published by Amer. Math. Soc., Providence, RI, 1967, pp. 200–215).
[8] S. E. Orcutt, "Implementation of permutation functions on Illiac IV-type computers," *IEEE Trans. Comput.*, vol. C-25, pp. 929–936, Sept. 1976.
[9] N. J. Pippenger, "The complexity theory of switching networks," M.I.T. Res. Lab. of Electronics, Rep. TR-487, pp. 42–43, 1973.
[10] ——, "Generalized connectors," IBM Res. Rep. RC-6532, 1977.
[11] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Trans. Comput.*, vol. C-26, pp. 153–161, Feb. 1977.
[12] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.
[13] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 263–271, Apr. 1977.
[14] A. Waksman, "A permutation network," *J. Ass. Comput. Mach.*, vol. 15, pp. 159–163, 1968.

Clark D. Thompson received the B.S. degree in chemistry and the M.S. degree in computer science (computer engineering) from Stanford University, Stanford, CA, in 1975.

He was awarded an NSF Fellowship in 1976 to study computer science at Carnegie-Mellon University, Pittsburgh, PA. His research interests center on computational complexity, especially for parallel models of computation.