A New Decomposition Algorithm for Rearrangeable Clos Interconnection Networks

Hyun Yeop Lee, Frank K. Hwang, and John D. Carpinelli, *Senior Member, IEEE*

***Abstract*—We give a new decomposition algorithm to route a rearrangeable three-stage Clos network in $O(nr^2)$ time, which is faster than all existing decomposition algorithms. By performing a row-wise matrix decomposition, this algorithm routes all possible permutations, thus overcoming the limitation on realizable permutations exhibited by many other routing algorithms. This algorithm is extended to the fault tolerant Clos network which has extra switches in each stage, where it provides fault tolerance under faulty conditions and reduces routing time under submaximal fault conditions.**

***Index Terms*— Interconnection networks, multiprocessor systems, routing algorithm, fault tolerance.**

I. Introduction

CLOS interconnection networks [1] have been gaining attention due to their potential uses in data networks and computing systems. The general Clos network is shown in Fig. 1. The three-stage Clos network consists of two symmetrical outer stages of rectangular switches, with an inner stage of square switches. The first stage contains r switches, each of which has n inputs and m outputs. Each switch is a simple crossbar switch which can realize any mapping of its inputs onto its outputs on a one-to-one basis. The second stage consists of $mr \times r$ switches, each of which receives exactly one input from each first-stage switch. The output stage has $rm \times n$ switches, each of which receives exactly one input from each second stage switch. The number of inputs to the network is $N = nr$.

Various routing schemes for this type of network have been reported in the literature [2]–[10]. In general, matrix decomposition algorithms are not as fast as graph coloring algorithms in the computational complexity. However, the former have the advantage of addressing the problem directly, without the need of converting to a graph-theoretic problem. Thus, they could be the right choices in many practical applications when the problem size is not too large and the computational complexity has to be balanced by the overhead. For example, the decomposition matrix algorithm proposed in this paper has been coded and used by the AT&T DACS IV-2000 switch group.

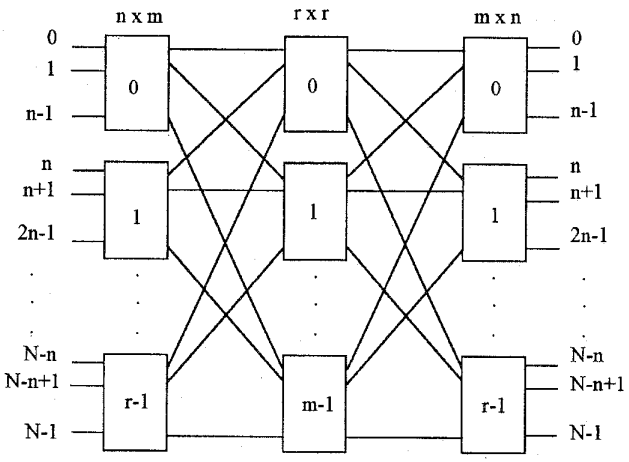


Fig. 1. A general ordinary Clos network.

Unfortunately, it has been shown that many of the early matrix decomposition algorithms are incomplete, except Neiman's algorithm [2], which is correct in principle, but gives no details for implementation. (It was stated in [5] that Neiman's algorithm can be implemented in $O(nr^4)$ time.)

Recently, Gordon and Srikanthan [8] introduced an algorithm, referred to as the (GS) algorithm in this paper, which uses two nouvelle matrices, called the specification matrix and the count matrix (the count matrix is not strictly necessary but helps in describing the algorithm). However, the GS algorithm was suspected to be incomplete by Chiu and Siu [9], and Lee and Carpinelli [12].

On the other hand, efforts have been devoted to making the interconnection networks fault tolerant including Clos networks. A single fault in the interconnection network can cause a severe degradation in performance unless measures are provided to make the network tolerant of such faults.

With developments in very large scale integration (VLSI) technology, large scale multiprocessor systems with fault-tolerant interconnection networks have become feasible. Nassar [11] has introduced the fault tolerant Clos (FTC) network by adding extra switches in each stage of the Clos network. Lee and Carpinelli [12] have given an algorithm for routing FTC networks. They also showed that the extra switches in

Paper approved by A. Jajszczyk, the Editor for Switch Theory and Fabrics of the IEEE Communications Society. Manuscript received November 9, 1994; revised June 8, 1995 and February 5, 1996. This paper was presented in part at the Conference on Information Science and Systems, Princeton University, NJ, March 1994.

H. Y. Lee is with Hyundai Electronics Co., Ichon, Korea (email: hylee@hei.co.kr).

F. K. Hwang was with AT&T Bell Labs, Murray Hill, NJ 07974 USA. He is now with the Department of Applied Mathematics, Chiaotung University, Hsinchu, Taiwan ROC (email: fhwang@iris2.math.nctu.edu.tw).

J. D. Carpinelli is with the Department of Electrical and Computer Engineering, the New Jersey Institute of Technology, Newark, 07102-1982 USA (email: carpinelli@njit.edu).

Publisher Item Identifier S 0090-6778(96)08594-7.

FTC networks can improve the run time of the algorithm significantly when the system displays few or no faults.

In this paper, we give a new decomposition algorithm, also based on the specification matrix, which requires $O(nr^2)$ time. Thus, it is faster than all existing decomposition algorithms. This algorithm is extended to FTC networks.

II. The GS Algorithm

It is customary to assume that the necessary connections involve every input and every output, and thus are representable by a permutation

$$P = \begin{bmatrix} 0 & 1 & \cdots & i & \cdots & N-1 \\ \pi(0) & \pi(1) & \cdots & \pi(i) & \cdots & \pi(N-1) \end{bmatrix}$$

where input i is to be connected to output $\pi(i)$, $0 \le i \le N-1$, and $N = rn$. Since each switch is assumed to be nonblocking between its inlets and its outlets, we may transform P into a mapping between input switches and output switches. For example, for $r = 4$ and $n = 3$

$$P = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 2 & 10 & 3 & 5 & 6 & 11 & 7 & 1 & 9 & 4 & 0 & 8 \end{bmatrix}$$

is transformed to

$$P^* = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 0 & 3 & 1 & 1 & 2 & 3 & 2 & 0 & 3 & 1 & 0 & 2 \end{bmatrix}.$$

Gordon and Srikanthan introduced an $r \times n$ matrix $S = (s_{ij})$, called the *specification matrix*, where s_{ij} is the jth element in the ith row of S. The rows of S are indexed by input switches and columns by center switches, while entries represent output switches. Thus, $s_{ij} = e$ implies that a connection from the ith input switch to the eth output switch is routed through the jth center switch. Clearly, the routing represented by S is feasible if and only if each column is *complete*, i.e., containing each output exactly once. S is called complete if every column is complete. While the initial S may not be complete, the GS algorithm gives a method to yield a complete S through iterative swapping of two entries in the same row.

Gordon and Srikanthan also introduced another $r \times n$ matrix $C = (c_{ej})$, called the *count matrix*, whose rows are indexed by output switches and columns by center switches, while c_{ej} is the number of occurrences of entry e in column j of S. Let S_j (respectively, C_j) denote column j of S (respectively, C). Then S_j is complete if and only if C_j is all ones. From the P^* given before, we have

$$S = \begin{bmatrix} 0 & 3 & 1 \\ 1 & 2 & 3 \\ 2 & 0 & 3 \\ 1 & 0 & 2 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

We now describe the GS algorithm.

Step 1) Initialize by setting $j = u = 0$.

Step 2) Find the minimum j such that S_j is not complete. If no such j exists, then stop.

Step 3) Find a row e of C such that $c_{ej} = 0$.

Step 4) Find C_k with $c_{ek} \ge 2$. Search a row i in the order of u, $u+1$, $\cdots$ such that $s_{ik} = e$.

Step 5) Swap s_{uj} with s_{uk}. Set $u = i + 1$. Go to Step 2).

Note that e and k are chosen arbitrarily if there is more than one choice. On the other hand, i is chosen by following a specific order controlled by the variable u. Suppose that e is chosen in a similar way to i, namely, by setting up a variable w and letting the search follow the order w, $w+1$, $\cdots$ (mod r). However, the following counterexample shows that such a choice of e can lead to infinite looping. Since an arbitrary choice includes this specific choice, the GS algorithm can also encounter infinite looping. Given the matrix

$$S = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 2 & 1 & 4 \end{bmatrix}$$

the GS algorithm can perform the following swaps in the order shown: $s_{10} \leftrightarrow s_{11}$; $s_{40} \leftrightarrow s_{42}$; $s_{10} \leftrightarrow s_{11}$; $s_{40} \leftrightarrow s_{42}$. This returns to the original matrix and thus represents a potential infinite loop. An expanded description of this example is included in the appendix. This example will also be used to demonstrate our proposed algorithm.

Chiu and Siu [9] modified the GS algorithm by adding the control variable w, along with a few other perturbations. But there is no guarantee that these perturbations can avoid infinite looping.

III. A New Decomposition Algorithm

We give a new decomposition algorithm which also swaps elements of S until S is complete. For a given element e, a column of S is *e-excessive* if it contains more than one e, and *e-deficient* if it does not contain e (there are at most $n-1$ e-deficient columns). An element e is called *balanced* if there exists no e-excessive column, and *unbalanced* otherwise.

At an iteration step, suppose that $e, 0 \le e < r-1$, is the minimum unbalanced element (if all but one elements are balanced, the remaining element is also balanced). Then there exists an e-excessive column S_i and an e-deficient column S_j. Arbitrarily select a row k with $s_{ki} = e$. Let $s_{kj} = e_1$. If $e < e_1$, swap s_{ki} with s_{kj} and the iteration step, labeled *simple swap*, is completed. If $e > e_1$, select another row k' with $s_{k'i} = e$. Let $s_{k'j} = e_1'$. If $e < e_1'$, swap $s_{k'i}$ with $s_{k'j}$ and the iteration step, labeled *next simple swap*, is again completed. If $e > e_1'$, swap s_{ki} with s_{kj}. Let k_1 be the row such that $s_{k_1 i} = e_1$ (recall that e_1 was balanced), swap $s_{k_1 i}$ with $s_{k_1 j} = e_2$. If $e < e_2$, the iteration step, labeled *successive swap*, is completed. If $e > e_2$, let k_2 be the row such that $s_{k_2 i} = e_2$ and swap $s_{k_2 i}$ with $s_{k_2 j}$, and so on. Since $e, e_1, e_2, \cdots$ are all distinct, a successive swap iterative step must be completed in at most e swappings.

At the end of an iteration step, either the minimum number of balanced element increases to $e+1$, or it remains at e, but the number of e-deficient columns is decreased by one. Hence, the number of iteration steps is at most

$$\sum_{e=0}^{r-2} (n-1) = (r-1)(n-1)$$

and the number of swappings is at most

$$\sum_{e=0}^{r-2}(n-1)(e+1) = (n-1)\binom{r}{2} = O(nr^2).$$

We now show that our algorithm can be implemented in a way such that each iteration step requires constant time. We need to do some preprocessing which consists of construction of the following three types of sets.

1) (j,e), $j = 0,1,\cdots,n-1$, $e = 0,1,\cdots,r-1$, is the set of rows $\{i\}$ such that $s_{ij} = e$.
2) $0(e)$, $e = 0,1,\cdots,r-1$, is the set of columns $\{j\}$ such that S_j does not contain e.
3) $2(e)$, $e = 0,1,\cdots,r-1$, is the set of columns $\{j\}$ such that S_j contains e at least twice.

By going through all s_{ij} once, and assigning i to (j,e) if $s_{ij} = e$, the nr sets (j,e) can be constructed in $O(nr)$ time. We will also keep a count $|(j,e)|$ on the cardinality of (j,e). At the beginning, put all columns in $0(e)$. Remove j from $0(e)$ whenever $|(j,e)| \geq 1$, and assign j to $2(e)$ whenever $|(j,e)|$ reaches 2. In this way $0(e)$ and $2(e)$ are constructed along with (j,e). We now state the proposed algorithm using these sets. Initialize by setting $e = 0$.

Step 1) If $2(e)$ is empty, i.e., $|2(e)| = 0$, set $e = e+1$. Stop if $e = r$; otherwise repeat Step 1).

Step 2) If $2(e)$ is not empty, i.e., $|2(e)| > 0$, take its first element j. Also, take the first element k of $0(e)$.

Step 3) (Simple Swap) Set i to be the first element of (j,e). If $e < s_{ik}$, swap s_{ij} with s_{ik}. Suppose $s_{ik} = e'$. Remove i from (j,e) and (k,e'), and add i to (j,e') and (k,e). If $|(j,e)| = 1$, remove j from $2(e)$. If $|(j,e')| = 1$, remove j from $0(e')$. If $|(j,e')| = 2$, add j to $2(e')$. If $|(k,e')| = 0$, add k to $0(e')$. If $|(k,e')| = 1$, remove k from $2(e')$. Remove k from $0(e)$ and go to Step 1).

Step 4) (Next Simple Swap) If $e > s_{ik}$, repeat Step 3) on the second element i' of (j,e). If $e > s_{i'k}$, go to Step 5).

Step 5) (Successive Swap) Divide into substeps.

A) Set $u = e$. Remove k from $0(u)$. If $|(j,u)| = 2$, remove j from $2(u)$.

B) Set $v = s_{ik}$. Swap s_{ij} with s_{ik}. Remove i from (j,u) and (k,v) and add i into (j,v) and (k,u).

C) Suppose $e < v$. If $|(k,v)| = 0$, add k to $0(v)$. If $|(k,v)| = 1$, remove k from $2(v)$. If $|(j,v)| = 1$, remove j from $0(v)$. If $|(j,v)| = 2$, add j to $2(v)$. Go to Step 1).

D) Suppose $e > v$. Set $u = v$ and go to Step 5B).

Adding an element to a set and choosing or removing the first (or second) element from a set take $O(1)$ time. Since k and i are the first elements of their sets, Steps 5A), 5B), and 5D) each take $O(1)$ time. Note that i is a unique element in (k,v) in Step 5B). Removing a specific but generally positioned element from an r-set takes $O(r)$ time. This occurs in Step 5C) when we remove k from $2(v)$ and j from $0(v)$. But the looping of Step 5) occurs only between Step 5B) and Step 5D). Therefore Step 5) takes $O(r)$ time and the algorithm $O(nr^2)$ time.

We illustrate this algorithm with an example. Constructing three types of sets of rows or columns from the S matrix

$$S = \begin{bmatrix} 0^* & 2^* & 4 \\ 1 & 3 & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \begin{matrix} 0(0) = \{1,2\} & 2(0) = \{0\} \\ 0(1) = \{\} & 2(1) = \{\} \\ 0(2) = \{\} & 2(2) = \{\} \\ 0(3) = \{0\} & 2(3) = \{1\} \\ 0(4) = \{0\} & 2(4) = \{2\} \end{matrix}$$

$$\begin{matrix} (0,0) = \{0,2,3\} & (1,0) = \{\} & (2,0) = \{\} \\ (0,1) = \{1\} & (1,1) = \{4\} & (2,1) = \{3\} \\ (0,2) = \{4\} & (1,2) = \{0\} & (2,2) = \{1\} \\ (0,3) = \{\} & (1,3) = \{1,2\} & (2,3) = \{2\} \\ (0,4) = \{\} & (1,4) = \{3\} & (2,4) = \{0,4\}. \end{matrix}$$

Simple swap: $e = 0$, $j = 0$, $k = 1$, $i = 0$, $e' = 2$

$$S = \begin{bmatrix} 2 & 0 & 4 \\ 1 & 3 & 2 \\ 0^* & 3 & 3^* \\ 0 & 4 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad \begin{matrix} 0(0) = \{2\} & 2(0) = \{0\} \\ 0(1) = \{\} & 2(1) = \{\} \\ 0(2) = \{1\} & 2(2) = \{0\} \\ 0(3) = \{0\} & 2(3) = \{1\} \\ 0(4) = \{0\} & 2(4) = \{2\} \end{matrix}$$

$$\begin{matrix} (0,0) = \{2,3\} & (1,0) = \{0\} & (2,0) = \{\} \\ (0,1) = \{1\} & (1,1) = \{4\} & (2,1) = \{3\} \\ (0,2) = \{4,0\} & (1,2) = \{\} & (2,2) = \{1\} \\ (0,3) = \{\} & (1,3) = \{1,2\} & (2,3) = \{2\} \\ (0,4) = \{\} & (1,4) = \{3\} & (2,4) = \{0,4\}. \end{matrix}$$

Simple swap: $e = 0$, $j = 0$, $k = 2$, $i = 2$, $e' = 3$

$$S = \begin{bmatrix} 2 & 0 & 4 \\ 1 & 3 & 2 \\ 3 & 3 & 0 \\ 0 & 4 & 1 \\ 2^* & 1^* & 4 \end{bmatrix} \quad \begin{matrix} 0(0) = \{\} & 2(0) = \{\} \\ 0(1) = \{\} & 2(1) = \{\} \\ 0(2) = \{1\} & 2(2) = \{0\} \\ 0(3) = \{2\} & 2(3) = \{1\} \\ 0(4) = \{0\} & 2(4) = \{2\} \end{matrix}$$

$$\begin{matrix} (0,0) = \{3\} & (1,0) = \{0\} & (2,0) = \{2\} \\ (0,1) = \{1\} & (1,1) = \{4\} & (2,1) = \{3\} \\ (0,2) = \{4,0\} & (1,2) = \{\} & (2,2) = \{1\} \\ (0,3) = \{2\} & (1,3) = \{1,2\} & (2,3) = \{\} \\ (0,4) = \{\} & (1,4) = \{3\} & (2,4) = \{0,4\}. \end{matrix}$$

Successive swap: $e = 2$, $j = 0$, $k = 1$, $i = 4$, $e' = 1$, $u = 2$, $v = 1$

$$S = \begin{bmatrix} 2 & 0 & 4 \\ 1^* & 3^* & 2 \\ 3 & 3 & 0 \\ 0 & 4 & 1 \\ 1 & 2 & 4 \end{bmatrix} \quad \begin{matrix} 0(0) = \{\} & 2(0) = \{\} \\ 0(1) = \{1\} & 2(1) = \{0\} \\ 0(2) = \{\} & 2(2) = \{\} \\ 0(3) = \{2\} & 2(3) = \{1\} \\ 0(4) = \{0\} & 2(4) = \{2\} \end{matrix}$$

$$\begin{matrix} (0,0) = \{3\} & (1,0) = \{0\} & (2,0) = \{2\} \\ (0,1) = \{1,4\} & (1,1) = \{\} & (2,1) = \{3\} \\ (0,2) = \{0\} & (1,2) = \{4\} & (2,2) = \{1\} \\ (0,3) = \{2\} & (1,3) = \{1,2\} & (2,3) = \{\} \\ (0,4) = \{\} & (1,4) = \{3\} & (2,4) = \{0,4\}. \end{matrix}$$

Simple swap: $j = 0$, $k = 1$, $i = 1$, $u = 1$, $v = 3$

$$S = \begin{bmatrix} 2 & 0 & 4 \\ 3 & 1 & 2 \\ 3^* & 3 & 0^* \\ 0 & 4 & 1 \\ 1 & 2 & 4 \end{bmatrix} \quad \begin{matrix} 0(0) = \{\} & 2(0) = \{\} \\ 0(1) = \{\} & 2(1) = \{\} \\ 0(2) = \{\} & 2(2) = \{\} \\ 0(3) = \{2\} & 2(3) = \{0\} \\ 0(4) = \{0\} & 2(4) = \{2\} \end{matrix}$$

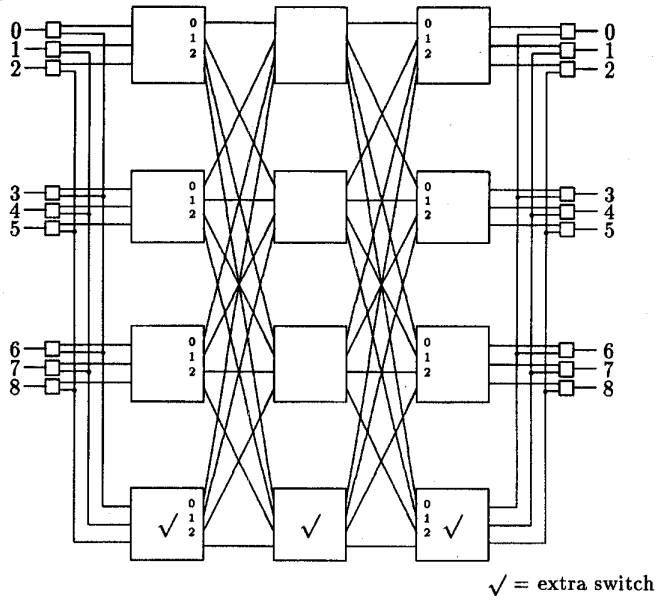


Fig. 2. A fault-tolerant 9 × 9 Clos network with one extra switch in each stage

$$\begin{array}{lll}(0,0)=\{3\} & (1,0)=\{0\} & (2,0)=\{2\}\\ (0,1)=\{4\} & (1,1)=\{1\} & (2,1)=\{3\}\\ (0,2)=\{0\} & (1,2)=\{4\} & (2,2)=\{1\}\\ (0,3)=\{2,1\} & (1,3)=\{2\} & (2,3)=\{\}\\ (0,4)=\{\} & (1,4)=\{3\} & (2,4)=\{0,4\}.\end{array}$$

Successive swap: $e=3$, $j=0$, $k=2$, $i=2$, $e'=0$, $u=3$, $v=0$

$$S=\begin{bmatrix}2 & 0 & 4\\ 3 & 1 & 2\\ 0 & 3 & 3\\ 0^* & 4 & 1^*\\ 1 & 2 & 4\end{bmatrix}\quad\begin{array}{ll}0(0)=\{2\} & 2(0)=\{0\}\\ 0(1)=\{\} & 2(1)=\{\}\\ 0(2)=\{\} & 2(2)=\{\}\\ 0(3)=\{\} & 2(3)=\{\}\\ 0(4)=\{0\} & 2(4)=\{2\}\end{array}$$

$$\begin{array}{lll}(0,0)=\{3,2\} & (1,0)=\{0\} & (2,0)=\{\}\\ (0,1)=\{4\} & (1,1)=\{1\} & (2,1)=\{3\}\\ (0,2)=\{0\} & (1,2)=\{4\} & (2,2)=\{1\}\\ (0,3)=\{1\} & (1,3)=\{2\} & (2,3)=\{2\}\\ (0,4)=\{\} & (1,4)=\{3\} & (2,4)=\{0,4\}.\end{array}$$

Simple swap: $j=0$, $k=2$, $i=3$, $u=0$, $v=1$

$$S=\begin{bmatrix}2 & 0 & 4\\ 3 & 1 & 2\\ 0 & 3 & 3\\ 1 & 4 & 0\\ 1^* & 2 & 4^*\end{bmatrix}\quad\begin{array}{ll}0(0)=\{\} & 2(0)=\{\}\\ 0(1)=\{2\} & 2(1)=\{0\}\\ 0(2)=\{\} & 2(2)=\{\}\\ 0(3)=\{\} & 2(3)=\{\}\\ 0(4)=\{0\} & 2(4)=\{2\}\end{array}$$

$$\begin{array}{lll}(0,0)=\{2\} & (1,0)=\{0\} & (2,0)=\{3\}\\ (0,1)=\{4,3\} & (1,1)=\{1\} & (2,1)=\{\}\\ (0,2)=\{0\} & (1,2)=\{4\} & (2,2)=\{1\}\\ (0,3)=\{1\} & (1,3)=\{2\} & (2,3)=\{2\}\\ (0,4)=\{\} & (1,4)=\{3\} & (2,4)=\{0,4\}.\end{array}$$

Simple swap: $e=1$, $j=0$, $k=2$, $i=4$, $e'=4$

$$S=\begin{bmatrix}2 & 0 & 4\\ 3 & 1 & 2\\ 0 & 3 & 3\\ 1 & 4 & 0\\ 4 & 2 & 1\end{bmatrix}\quad\begin{array}{ll}0(0)=\{\} & 2(0)=\{\}\\ 0(1)=\{\} & 2(1)=\{\}\\ 0(2)=\{\} & 2(2)=\{\}\\ 0(3)=\{\} & 2(3)=\{\}\\ 0(4)=\{\} & 2(4)=\{\}\end{array}$$

$$\begin{array}{lll}(0,0)=\{2\} & (1,0)=\{0\} & (2,0)=\{3\}\\ (0,1)=\{3\} & (1,1)=\{1\} & (2,1)=\{4\}\\ (0,2)=\{0\} & (1,2)=\{4\} & (2,2)=\{1\}\\ (0,3)=\{1\} & (1,3)=\{2\} & (2,3)=\{2\}\\ (0,4)=\{4\} & (1,4)=\{3\} & (2,4)=\{0\}.\end{array}$$

IV. Extended Algorithms for FTC Networks

The Clos network can be made fault-tolerant as shown in Fig. 2 by adding extra switches in all stages and multiplexers/demultiplexers before the first stage and after the last stage [11]. For the FTC networks, the following fault model is assumed.

1) Any switch can fail
2) Any interstage link can fail
3) The failure rate of multiplexers, demultiplexers and external links is negligible as compared to that of the switches.

The faults are assumed to occur independently, and the faulty components are unusable. The FTC network achieves fault tolerance by redirecting inputs to multiplexers, demultiplexers and extra switches in the outer stages when there is a fault in the outer stage switches. If there is a fault in a middle stage switch, inputs bound to a faulty switch are directed to one of the extra switches in the middle stage. FTC networks with y extra switches in each outer stage and x extra switches in the center stage can tolerate up to $2y+x$ switch failures, as long as each stage contains no more faulty switches than it has spare switches. A link failure may be modeled as a failure of either of the switches it connects.

The S matrix is basically the same as in the ordinary Clos network except that the spares are considered in the matrix. The extra y switches in each of the first and third stages add an additional y rows to the S matrix, which provide ny extra spare elements. These spares are denoted as α, and can be considered as wild cards during the routing process; this eases

the swapping of a pair of elements. The extra x switches in the second stage, on the other hand, add x extra columns to the S matrix, which provide rx extra spare elements. These spares are represented as β. The α spares correspond to the paths created by multiplexers/demultiplexers along with extra switches in outer stages, and may swap with any element in the same column except β spares. The β spares correspond to the paths generated by extra switches in the second stage, and these spares can be swapped with any element in the same row except α spares. The spares α and β also can be swapped only if the resulting number of α spares in each column remains the same. Violating these rules would correspond to physically changing the connections and switch locations in the FTC network, which is prohibited. As noted in Section II, the rows of S are indexed by input switches and columns by center switches, while entries represent output switches.

Fault checking is performed prior to the execution of the proposed algorithm. If there is a fault in any nonspare switch, an available spare switch within that stage must be assigned in place of the faulty switch. However, when a first stage spare switch is faulty, one of the α rows in the matrix is not used. When a second stage spare switch is faulty, one of the β columns of the matrix is removed. If a third stage switch is faulty, its corresponding value is not used. Once the network is reconfigured, the algorithm is applied as if there is no fault in the network and switches are set accordingly.

In the FTC network, a column j, $0 \le j < n + x$ of S is called *e-excessive* for a given element e if it contains more than one e in the rows k, $0 \le k < r$, and *e-deficient* if it does not contain e in the rows k, $0 \le k < r$. When $e = \alpha$ (respectively, β), a column j, $0 \le j < n + x$ of S is called *e-excessive* if it contains more than y e's (respectively, x e's) in the rows k, $0 \le k < r + y$, and *e-deficient* if it contains less than y α's (respectively, x β's) in the rows k, $0 \le k < r + y$. An element e is called *balanced* if there exists no e-excessive column, and *unbalanced* otherwise. Note that α needs to be balanced to completely decompose the S matrix, but β may be unbalanced because of its flexibility. The previous algorithm for ordinary Clos networks can be easily extended to FTC networks which also swaps elements of S until S is complete.

As an iteration step, suppose that e, $0 \le e < r - 1$, is the minimum unbalanced element. Then there exists an e-excessive column S_i and an e-deficient column S_j. Select arbitrarily a row k with $s_{ik} = e$. If there is any spare α (or β) in the extra column i (or extra row k), swap s_{ik} with α (or β). This is labeled *wild swap* since α or β can be used as a wild card during routing. Otherwise, let $s_{jk} = e_1$. If $e < e_1$ or $e = \beta$, swap s_{ik} with s_{jk} and the iteration step, labeled *simple swap*, is completed. If $e > e_1$, select another row k' with $s_{ik'} = e$. Let $s_{jk'} = e_1'$. If $e < e_1'$ or $e = \beta$, swap $s_{ik'}$ with $s_{jk'}$ and the iteration step, labeled *next simple swap*, is again completed. If $e > e_1'$, swap s_{ik} with s_{jk}. Let k_1 be the row such that $s_{ik_1} = e_1$ (recall that e_1 is balanced), swap s_{ik_1} with $s_{jk_1} = e_2$. If $e < e_2$ or $e = \beta$, the iteration step, labeled *successive swap*, is completed. If $e > e_2$, let k_2 be the row such that $s_{ik_2} = e_2$ and swap s_{ik_2} with s_{jk_2}, and so on.

To speed up the algorithm, we again need to do some preprocessing. By going through all of S once, construct sets $0(e)$, $2(e)$, and (j, e). We now state the algorithm for FTC networks using these sets.

Algorithm: Initialize by setting $e = 0$.

Step 1) If $2(e)$ is empty, i.e., if the cardinality of $2(e)$ is zero ($|2(e)| = 0$), set $e = e + 1$. Stop if $e = r$; otherwise repeat Step 1).

Step 2) If $2(e)$ is not empty, i.e., $|2(e)| > 0$, take its first element j. Also, take the first element k of $0(e)$.

Step 3) (Wild Swap) Set i to be the first element of (j, e). If u is an element of (j, α), i.e., $u \in (j, \alpha)$, $u \ge r$, swap s_{ij} with s_{uj}. Remove i from (j, e), u from (j, α) and add i to (j, α), u to (j, e). If $i \in (v, \beta)$, $v \ge n$, swap s_{ij} with s_{iv}. Remove i from (j, e) and (v, β) and add i to (j, β) and (v, e). If $|(j, e)| = 1$ in both cases, remove j from $2(e)$. Go to Step 1. If no spares are available, go to Step 4).

Step 4) (Simple Swap) Set i to be the first element of (j, e). If $e < s_{ik}$, or $s_{ik} = \beta$, swap s_{ij} with s_{ik}. Suppose $s_{ik} = e'$. Remove i from (j, e) and (k, e'), and add i to (j, e') and (k, e). If $|(j, e)| = 1$, remove j from $2(e)$. If $|(j, e')| = 1$, remove j from $0(e')$. If $|(j, e')| = 2$, add j to $2(e')$. If $|(k, e')| = 0$, add k to $0(e')$. If $|(k, e')| = 1$, remove k from $2(e')$. Remove k from $0(e)$. Go to Step 1).

Step 5) (Next Simple Swap) If $e > s_{ik}$, or $s_{ik} = \beta$, repeat Step 4 on the second element i' of (j, e). If $e > s_{i'k}$ or $s_{i'k} = \alpha$, go to Step 6).

Step 6) (Successive Swap) Divide into substeps.

A) Set $u = e$. Remove k from $0(u)$. If $|(j, u)| = 2$, remove j from $2(u)$.

B) Set $v = s_{ik}$. Swap s_{ij} with s_{ik}. Remove i from (j, u) and (k, v), and add i to (j, v) and (k, u).

C) Suppose $e < v$ or $v = \beta$. If $|(k, v)| = 0$, add k to $0(v)$. If $|(k, v)| = 1$, remove k from $2(v)$. If $|(j, v)| = 1$, remove j from $0(v)$. If $|(j, v)| = 2$, add j to $2(v)$. Go to Step 1).

D) Suppose $e > v$ or $v = \alpha$. Set $u = v$ and go to Step 6B).

We again illustrate the algorithm by starting from element zero in the S matrix when $n = 3$, $r = 4$ and $x = y = 1$. In this example no switches are faulty; this is done to illustrate the use of fault-tolerance hardware under no-fault conditions. Constructing three types of sets of rows or columns from the S matrix

$$S = \begin{bmatrix} 2 & 2 & 2 & \beta \\ 1 & 0^* & 3 & \beta \\ 1 & 0^* & 3 & \beta^* \\ 1 & 0 & 3 & \beta \\ \alpha & \alpha^* & \alpha & \end{bmatrix}$$

$0(0) = \{0, 2, 3\}$ $2(0) = \{1\}$
$0(1) = \{1, 2, 3\}$ $2(1) = \{0\}$
$0(2) = \{3\}$ $2(2) = \{\}$
$0(3) = \{0, 1, 3\}$ $2(3) = \{2\}$

$(0, 0) = \{\}$ $(1, 0) = \{1, 2, 3\}$ $(2, 0) = \{\}$
$(0, 1) = \{1, 2, 3\}$ $(1, 1) = \{\}$ $(2, 1) = \{\}$
$(0, 2) = \{0\}$ $(1, 2) = \{0\}$ $(2, 2) = \{0\}$
$(0, 3) = \{\}$ $(1, 3) = \{\}$ $(2, 3) = \{1, 2, 3\}$
$(0, \beta) = \{\}$ $(1, \beta) = \{\}$ $(2, \beta) = \{\}$
$(0, \alpha) = \{4\}$ $(1, \alpha) = \{4\}$ $(2, \alpha) = \{4\}$
$(3, \beta) = \{0, 1, 2, 3\}$.

For elements $s_{ij} = 1$ (*wild swap*)

$$S = \begin{bmatrix} 2 & 2 & 2 & \beta \\ 1^* & \alpha & 3 & \beta \\ 1 & \beta & 3 & 0 \\ 1^* & 0 & 3 & \beta^* \\ \alpha^* & 0 & \alpha & \end{bmatrix}$$

$0(0) = \{0,2\}$ $2(0) = \{\}$
$0(1) = \{1,2,3\}$ $2(1) = \{0\}$
$0(2) = \{3\}$ $2(2) = \{\}$
$0(3) = \{0,1,3\}$ $2(3) = \{2\}$

$(0,0) = \{\}$
$(0,1) = \{1,2,3\}$
$(0,2) = \{0\}$
$(0,3) = \{\}$
$(0,\beta) = \{\}$
$(0,\alpha) = \{4\}$

$(1,0) = \{3,4\}$
$(1,1) = \{\}$
$(1,2) = \{0\}$
$(1,3) = \{\}$
$(1,\beta) = \{2\}$
$(1,\alpha) = \{1\}$
$(3,0) = \{2\}$

$(2,0) = \{\}$
$(2,1) = \{\}$
$(2,2) = \{0\}$
$(2,3) = \{1,2,3\}$
$(2,\beta) = \{\}$
$(2,\alpha) = \{4\}$
$(3,\beta) = \{0,1,3\}$.

For an element $s_{ij} = 3$ (*wild swap*)

$$S = \begin{bmatrix} 2 & 2 & 2 & \beta \\ \alpha & \alpha & 3^* & \beta \\ 1 & \beta & 3 & 0 \\ \beta & 0 & 3 & 1 \\ 1 & 0 & \alpha^* & \end{bmatrix}$$

$0(0) = \{0,2\}$ $2(0) = \{\}$
$0(1) = \{1,2\}$ $2(1) = \{\}$
$0(2) = \{3\}$ $2(2) = \{\}$
$0(3) = \{0,1,3\}$ $2(3) = \{\}$

$(0,0) = \{\}$
$(0,1) = \{2,4\}$
$(0,2) = \{0\}$
$(0,3) = \{\}$
$(0,\beta) = \{3\}$
$(0,\alpha) = \{1\}$

$(1,0) = \{3,4\}$
$(1,1) = \{\}$
$(1,2) = \{0\}$
$(1,3) = \{\}$
$(1,\beta) = \{2\}$
$(1,\alpha) = \{1\}$

$(2,0) = \{\}$
$(2,1) = \{\}$
$(2,2) = \{0\}$
$(2,3) = \{1,2,3\}$
$(2,\beta) = \{\}$
$(2,\alpha) = \{4\}$

$(3,0) = \{2\}$
$(3,1) = \{3\}$
$(3,2) = \{\}$
$(3,3) = \{\}$
$(3,\beta) = \{0,1\}$.

Again, for an element $s_{ij} = 3$ (*wild swap*)

$$S = \begin{bmatrix} 2 & 2 & 2 & \beta \\ \alpha & \alpha & \alpha & \beta \\ 1 & \beta^* & 3^* & 0 \\ \beta & 0 & 3 & 1 \\ 1 & 0 & 3 & \end{bmatrix}$$

$0(0) = \{0,2\}$ $2(0) = \{\}$
$0(1) = \{1,2\}$ $2(1) = \{\}$
$0(2) = \{3\}$ $2(2) = \{\}$
$0(3) = \{0,1,3\}$ $2(3) = \{2\}$

$(0,0) = \{\}$
$(0,1) = \{2,4\}$
$(0,2) = \{0\}$
$(0,3) = \{\}$
$(0,\beta) = \{3\}$
$(0,\alpha) = \{1\}$

$(1,0) = \{3,4\}$
$(1,1) = \{\}$
$(1,2) = \{0\}$
$(1,3) = \{\}$
$(1,\beta) = \{2\}$
$(1,\alpha) = \{1\}$

$(2,0) = \{\}$
$(2,1) = \{\}$
$(2,2) = \{0\}$
$(2,3) = \{2,3,4\}$
$(2,\beta) = \{\}$
$(2,\alpha) = \{1\}$

$(3,0) = \{2\}$
$(3,1) = \{3\}$
$(3,2) = \{\}$
$(3,3) = \{\}$
$(3,\beta) = \{0,1\}$.

Simple swap: $e = 3$, $j = 2$, $k = 1$, $i = 2$, $e' = \beta$

$$S = \begin{bmatrix} 2 & 2 & 2 & \beta \\ \alpha & \alpha & \alpha & \beta \\ 1 & 3 & \beta & 0 \\ \beta & 0 & 3 & 1 \\ 1 & 0 & 3 & \end{bmatrix}$$

$0(0) = \{0,2\}$ $2(0) = \{\}$
$0(1) = \{1,2\}$ $2(1) = \{\}$
$0(2) = \{3\}$ $2(2) = \{\}$
$0(3) = \{0,3\}$ $2(3) = \{\}$

$(0,0) = \{\}$
$(0,1) = \{2,4\}$
$(0,2) = \{0\}$
$(0,3) = \{\}$
$(0,\beta) = \{3\}$
$(0,\alpha) = \{1\}$

$(1,0) = \{3,4\}$
$(1,1) = \{\}$
$(1,2) = \{0\}$
$(1,3) = \{2\}$
$(1,\beta) = \{\}$
$(1,\alpha) = \{1\}$

$(2,0) = \{\}$
$(2,1) = \{\}$
$(2,2) = \{0\}$
$(2,3) = \{3,4\}$
$(2,\beta) = \{2\}$
$(2,\alpha) = \{1\}$

$(3,0) = \{2\}$
$(3,1) = \{3\}$
$(3,2) = \{\}$
$(3,3) = \{\}$
$(3,\beta) = \{0,1\}$.

Note, that a successive swap iterative step will be completed in at most $e+y$ swaps because of the additional y α elements. As in the ordinary case, either the minimum balanced element increases to $e+1$, or it remains at e at the end of an iteration step, but the number of e-deficient columns is decreased by 1. Hence the number of iteration steps is at most

$$\sum_{e=0}^{r-2}(n+x-1) = (r-1)(n+x-1)$$

and the number of swappings without no wild swaps is at most

$$\sum_{e=0}^{r-2}(n+x-1)(e+y+1) = O(r(n+x)(r+y)).$$

However, the wild swap can suppress at most $rx+ny$ simple, next simple, or successive swaps. Thus, as the number of extra rows and columns increases, the algorithm has more chances to suppress the possible swaps and, as a result, can significantly improve the run time especially when there are few or no faults in the FTC network. Note, that Nassar [11] suggested an algorithm for FTC networks, but the run time of his algorithm increases as the number of spares switches increases.

V. Conclusion

This paper has presented a new routing algorithm for Clos networks. The proposed algorithm, which proceeds row-wise, can assure correct routing in contrast to the GS algorithm which runs column-wise. The time complexity of the preprocessed algorithm for the Clos network is $O(nr^2)$. This algorithm can easily be applied to FTC networks by introducing the wild swap, which treats extra switches as wild cards. The extra switches in the FTC network provide extra spare elements, which helps to improve the run time of the algorithm as well as the reliability.

APPENDIX
EXPANDED COUNTEREXAMPLE TO GORDON AND SRIKANTHAN'S ALGORITHM

The following counterexample shows that such a choice of e can lead to infinite looping. Since an arbitrary choice includes this specific choice, the GS algorithm can also encounter infinite looping. Elements selected for swapping are marked with an asterisk

First iteration: $j = 0,\ w = 0,\ u = 0,\ e = 3,\ k = 1,\ i = 1$

$$S = \begin{bmatrix} 0 & 2 & 4 \\ 1^* & 3^* & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 2 & 1 & 4 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

Second iteration: $j = 0,\ w = 4,\ u = 2,\ e = 4,\ k = 2,\ i = 4$

$$S = \begin{bmatrix} 0 & 2 & 4 \\ 3 & 1 & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 2^* & 1 & 4^* \end{bmatrix} \quad C = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

Third iteration: $j = 0,\ w = 0,\ u = 0,\ e = 1,\ k = 1,\ i = 1$

$$S = \begin{bmatrix} 0 & 2 & 4 \\ 3^* & 1^* & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 4 & 1 & 2 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Forth iteration: $j = 0,\ w = 2,\ u = 2,\ e = 2,\ k = 2,\ i = 4$

$$S = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 2 \\ 0 & 3 & 3 \\ 0 & 4 & 1 \\ 4^* & 1 & 2^* \end{bmatrix} \quad C = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

This results in the original matrix and value of u, thus realizing an infinite loop.

REFERENCES

[1] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, vol. 32, no. 2, pp. 406–424, Mar. 1953.

[2] V. I. Neiman, "Structure *et* commande optimales des réseaux de connexion sans blocage," *Annales des Telecommun.*, pp. 232–238, July/Aug. 1969.

[3] H. R. Ramanujam, "Decomposition of permutation networks," *IEEE Trans. Comput.*, vol. C-22, no. 7, pp. 639–643, July 1973.

[4] M. Kubale, "Comments on decomposition of permutation networks," vol. C-31, no. 3, p. 265, Mar. 1982.

[5] A. Jajszczyk, "A simple algorithm for the control of rearrangeable switching networks," *IEEE Trans. Commun.*, vol. COM-33, no. 2, pp. 169–171, Feb. 1985.

[6] C. Cardot, "Comments on a simple algorithm for the control of rearrangeable switching networks," *IEEE Trans. Commun.*, vol. COM-34, no. 4, p. 395, Apr. 1986.

[7] F. K. Hwang, "Control algorithms for rearrangeable Clos networks," *IEEE Trans. Commun.*, vol. COM-31, no. 8, pp. 952–954, Aug. 1983.

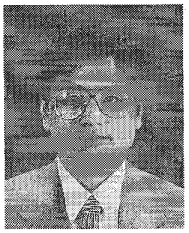
[8] J. Gordon and S. Srikanthan, "Novel algorithm for Clos-type networks," *Electron. Lett.*, vol. 26, no. 21, pp. 1772–1774, Oct. 1990.

[9] Y. K. Chiu and W. C. Siu, "Comment: Novel algorithm for Clos-type networks," *Electron. Lett.*, vol. 27, no. 6, pp. 524–526, Mar. 1991. Reply by J. Gordon and S. Srikanthan, p. 526.

[10] J. D. Carpinelli and A. Y. Oruc, "A nonbacktracking decomposition algorithm for routing on Clos networks," *IEEE Trans. Commun.*, vol. 41, no. 8, pp. 1245–1251, Aug. 1993.

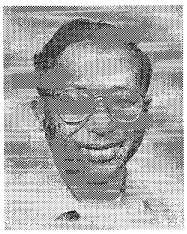
[11] H. Nassar and J. D. Carpinelli, "Design and performance of a fault tolerant Clos network," in *Conf. Inform. Sci. Syst.*, Johns Hopkins University, Mar. 1995, pp. 810–815.

[12] H. Y. Lee and J. D. Carpinelli, "Routing algorithms in fault tolerant Clos networks," in *Conf. Inform. Sci. Syst.*, Princeton University, NJ, Mar. 1994, pp. 227–231.



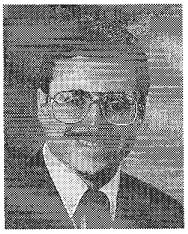
Hyun Yeop Lee received the B.S. degree in electronics engineering from Yonsei University, Seoul, Korea, in 1980, and the M.S. and Ph.D. degrees in electrical and computer engineering from the New Jersey Institute of Technology, Newark, NJ, in 1990 and 1994, respectively.

From 1980 to 1987, he was a R&D Engineer at the Agency for Defense Development, Daejeon, Korea. In 1995, he joined Hyundai Electronics Co. in Ichon, Korea, where he is presently engaged in the development of SMP servers. His research interests include interconnection networks, computer architecture, and parallel and distributed systems.



Frank K. Hwang received the B.A. degree in english literature from National Taiwan University in 1960, the M.B.A. degree from Baruch School in 1964, and the Ph.D. degree in statistics from North Carolina State University in 1968.

He had been a Researcher in the Mathematics Center of Bell Laboratories until his retirement in January 1996. He is a Professor in the Department of Applied Mathematics, Chiaotung University, Hsinchu, Taiwan, ROC. He has co-authored and co-edited six books and published about 300 papers.



John D. Carpinelli (S'81–M'86–SM'92) received the B.E. degree in electrical engineering from the Stevens Institute of Technology, Hoboken, NJ, in 1983, and the M.E. degree in electrical engineering and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1984 and 1987, respectively.

In 1986, he joined the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology, Hoboken, NJ, where he is currently an Associate Professor and Director of the computer engineering program. His primary research interests include interconnection networks, multiprocessor system design, and computer assisted instruction.