A Practical Fast Parallel Routing Architecture for Clos Networks

S.Q. Zheng
Dept. of Computer Science
University of Texas at Dallas
Richardson, TX 75083, USA
sizheng@utdallas.edu

Ashwin Gumaste
School of Information Tech.
Indian Institute of Technology
Bombay, India
ashwin@it.iitb.ac.in

Enyue Lu
Dept. Math and CS
Salisbury University
Salisbury, MD 21801, USA
ealu@salisbury.edu

ABSTRACT

Clos networks are an important class of switching networks due to their modular structure and much lower cost compared with crossbars. For routing I/O permutations of Clos networks, sequential routing algorithms are too slow, and all known parallel algorithms are not practical. We present the algorithm-hardware codesign of a unified fast parallel routing architecture called distributed pipeline routing (DPR) architecture for rearrangeable nonblocking and strictly nonblocking Clos networks. The DPR architecture uses a linear interconnection structure and processing elements that performs only shift and logic AND operations. We show that a DPR architecture can route any permutation in rearrangeable nonblocking and strictly nonblocking Clos networks in $O(\sqrt{N})$ time. The same architecture can be used to carry out control of any group of connection/disconnection requests for strictly nonblocking Clos networks in $O(\sqrt{N})$ time. Several speeding-up techniques are also presented. This architecture is applicable to packet and circuit switches of practical sizes.

Categories and Subject Descriptors: B.4.3 [Interconnections]: Parallel I/O; C.2.6 [Internetworking]: Routers

General Terms: Algorithms; Design

Keywords: Clos network, circuit switching, packet switching, permutation routing, rearrangeable nonblocking, strictly nonblocking, parallel algorithm, parallel architecture, pipelining.

1. INTRODUCTION

Switching networks serve as the core of network switches and routers, and the communication subsystems in high-performance parallel computer systems. An $N \times N$ switching network simultaneously connects up to N input/output pairs, which form a (partial) I/O permutation, using link-disjoint (or conflict-free) paths within the network. A switching network is *strictly nonblocking* if there always exists a connection path from any idle input to any idle output in the presence of existing connections, regardless of how the existing connection paths were selected. A switching network is *rearrangeable nonblocking* if connections for any (partial) I/O permutation can be established under the condition that rearrangement of existing connections is allowed. Strictly nonblocking switching networks are suitable to be used as switches in circuit switching networks, whereas rearrangeable nonblocking networks are more cost-effective for implementing packet switches for packet switching networks. The routing problem of a nonblocking switching network is to find conflict-free paths for I/O permutations. The focus of this paper is on the design and analysis of a practical fast parallel architecture for routing permutations in rearrangeable and strictly nonblocking Clos switching networks[6].

A 3-stage $N \times N$ Clos network can be constructed from smaller crossbar switches arranged in three stages so that it can be rearrangeable nonblocking or strictly nonblocking depending on the number of middle-stage modules. Its cost, measured by the number of crossing points, is $O(N^{1.5})$, which is significantly less than that of an $N \times N$ crossbar. Due to its modular structure, the generalized Clos networks can be recursively constructed with more than three stages and cost lower than $O(N^{1.5})$ [3]. However, unlike crossbars for which routing is trivial, the routing problem for Clos networks is time consuming. The best known sequential algorithm for rearrangeable Clos network has time complexity $O(N \log N)$ [15]. For strictly nonblocking Clos networks, we have not seen any algorithm that routes $O(N)$ new connections in the presence of $O(N)$ existing connections in less than $O(N^{1.5})$ time in the worst case. In order to speed up routing, parallel algorithms must be used. For rearrangeable 3-stage Clos networks, the best parallel time complexity is $O(\log^2 N)$ if the number of middle stage modules is a power of 2 [8]. This result is also reported in [11, 12] in different forms. For an arbitrary number of middle stage modules, the best parallel time known is $O(\log^3 N)$ according to [8]. These parallel time complexities are derived on the abstract parallel random access machine (PRAM) model [9] or a completely connected multiprocessor system of N processors supporting constant time communications of complicated patterns, which are either unrealistic or too expensive to build. We have not seen practical fast parallel algorithms for routing an arbitrary set of connections in strictly nonblocking Clos networks.

In this paper, we present the algorithm-hardware codesign of a fast parallel routing architecture called distributed pipeline routing (DPR) architecture. We first present a unified PRAM algorithm for routing in rearrangeable and

ANCS'06, December 3–5, 2006, San Jose, California, USA.

strictly nonblocking Clos networks, and then describe the DPR architecture that directly implements this algorithm. The attractive features of the DPR architecture include its simplicity and very small constant associated with its time complexity. It uses a linear interconnection structure and $O(N)$ processing elements that performs only shift and logic AND operations. A permutation routing task is decomposed into groups of simple Boolean logic operations, and these operations are distributed over processing elements in a pipelined fashion. Pipelining is further utilized at different levels to improve performance. We show that a DPR architecture can route any permutation in rearrangeable nonblocking and strictly nonblocking Clos networks in $O(\sqrt{N})$ time. The same architecture can be used to carry out control of any group of connection/disconnection requests for strictly nonblocking Clos networks in $O(\sqrt{N})$ time. Several speeding-up techniques are also presented. Considering that $\sqrt{N} \leq \log_2^2 N$ for $N \leq 65,536$, our architecture is applicable to packet and circuit switches of practical sizes.

2. ROUTING IN CLOS NETWORKS

The Clos network $C(n_1, r_1, m, n_2, r_2)$ is an $r_1 \cdot n_1 \times r_2 \cdot n_2$ 3-stage network, where the first stage S_1 consists of r_1 (input) crossbars $S_1(i)$, $0 \leq i \leq r_1 - 1$, of size $n_1 \times m$, the last stage S_3 consists of r_2 (output) crossbars $S_3(j)$, $0 \leq j \leq r_2 - 1$, of size $m \times n_2$, and the middle stage S_2 has m crossbars $S_2(k)$, $0 \leq k \leq m-1$, of size $r_1 \times r_2$. Each input crossbar $S_1(i)$ has a connection to each middle stage crossbar $S_2(k)$. Similarly, each middle stage crossbar $S_2(k)$ has a connection to each output crossbar $S_3(j)$. Each component crossbar is called a module. Figure 1 shows the structure of $C(n_1, r_1, m, n_2, r_2)$.

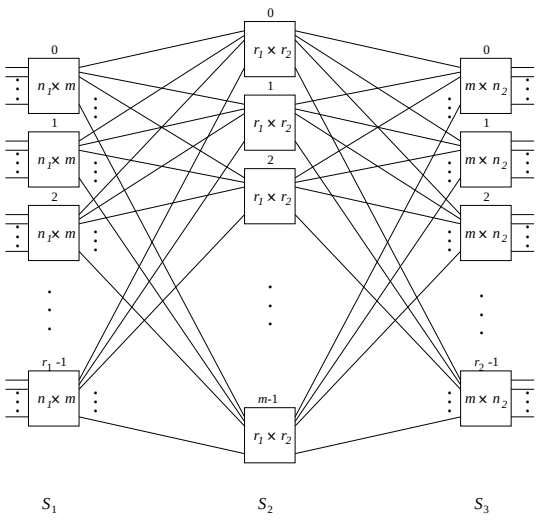

Figure 1: Three-stage Clos network $C(n_1, r_1, m, n_2, r_2)$.

When $n_1 = n_2 = n$ and $r_1 = r_2 = r$, the network $C(n_1, r_1, m, n_2, r_2)$ is called a *symmetric 3-stage Clos network*, denoted by $C(n, m, r)$. We are particularly interested in $N \times N$ symmetric Clos networks $C(n, m, r)$, where $N = nr$. It is known that $C(n, m, r)$ is rearrangeable nonblocking if and only if $m \geq n$ [1] and $C(n, m, r)$ is strictly nonblocking if and only if $m \geq \min\{2n - 1, nr\}$ [6]. Thus, rearrangeable and strictly nonblocking $C(n, m, r)$s with $m = O(n)$ have $O(N^{1.5})$ crossing points. Figure 2 gives a 3-dimensional view of $C(n, 2n - 1, n)$, where each module is represented by a plane.

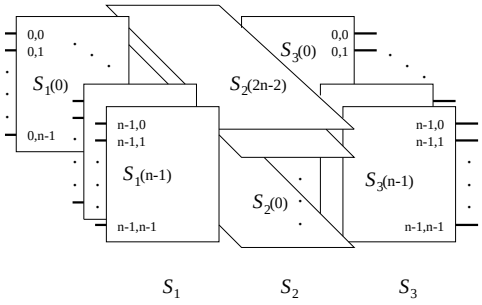

Figure 2: A 3-dimensional view of $C(n, 2n - 1, n)$.

For $C(n, m, r)$, the $N = nr$ inputs are denoted by a set $I = \cup_{0 \leq i \leq r-1} IG_i$, where $IG_i = \{I_{i,k} | 0 \leq k \leq n - 1\}$ is the i-th *input group* that consists of inputs of input module $S_1(i)$, with $I_{i,k}$ being the k-th input of $S_1(i)$. Similarly, the set of N outputs is $O = \cup_{0 \leq j \leq r-1} OG_j$, where $OG_j = \{O_{j,k} | 0 \leq k \leq n-1\}$ is the j-th *output group*, which consists of outputs of output module $S_3(j)$, with $O_{j,k}$ being the k-th output of $S_3(i)$. The connections between I and O of $C(n, m, r)$ can be characterized by a mapping π from I to O. For a (partial) permutation, each input in I is to be connected to at most one output in O and each output in O is to be connected to at most one input in I. We represent I/O connections by a bipartite graph $G(V_1, V_2; E)$, which has r nodes in each of V_1 and V_2, and a set E of edges between V_1 and V_2. More specifically, the nodes in V_1 are labeled v'_i (for input group IG_i) and the nodes in V_2 are labeled v''_j (for output group OG_j), $0 \leq i, j \leq r - 1$. There is a one-to-one correspondence between each I/O connection in an I/O mapping π and an edge in E: the I/O connection from $I_{i,j}$ to $O_{p,q}$ is corresponding to an edge between node v'_i and node node v''_p. Clearly, $G(V_1, V_2; E)$ is a multigraph of degree at most n. We denote the set of edges in E that connect the same pair of nodes v'_i and v''_j by $E_{i,j}$, i.e. $E_{i,j} = \{e | e = (v'_i, v''_j) \in E\}$. We call $G(V_1, V_2; E)$ an *I/O mapping graph*.

EXAMPLE 1. *Consider the following I/O mapping:*

$$\begin{pmatrix} (0,0)^* & (0,3) & (1,0) & (1,1) & (1,2) & (1,3) \\ (0,2)^* & (2,4) & (4,1) & (3,3) & (0,3) & (1,2) \end{pmatrix}$$

$$\begin{pmatrix} (1,4) & (2,0)^* & (2,4)^* & (3,1)^* & (4,1)^* & (4,2) \\ (0,1) & (4,0)^* & (3,0)^* & (2,2)^* & (3,2)^* & (3,4) \end{pmatrix}$$

in which (x, y) and (x', y') in the same column indicates input $I_{x,y}$ is mapped to output input $O_{x',y'}$ in strictly nonblocking $C(5, 9, 5)$. A column marked with $*$ corresponds to an existing I/O connection and a column without $*$ indicates a new connection to be established. The corresponding I/O mapping graph is shown in Figure 3.

■

For routing in rearrangeable nonblocking $C(n, m, r)$ used as a cell switch, no connection is assumed existing.

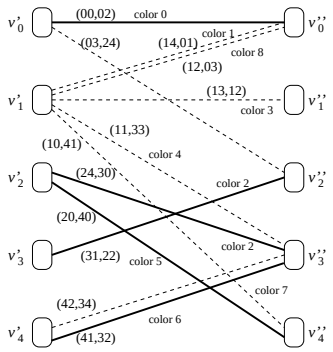

Figure 3: An I/O mapping graph of strictly nonblocking $C(5,9,5)$, where a solid edge corresponds to an existing I/O connection, and a dashed edge corresponds to an I/O connection to be established. $(xy, x'y')$ is a connection from $I_{x,y}$ to $O_{x',y'}$.

The routing problem for $C(n,m,r)$ can be solved by properly coloring edges of $G(V_1, V_2; E)$ with at most m colors. We say that the edges are *properly colored* if no two edges incident at the same node have the same color. A proper coloring of edges in $G(V_1, V_2; E)$ translates into conflict-free I/O connections within $C(n,m,r)$ as follows: if an edge between v'_i and v''_j is assigned color k, then its corresponding I/O connection is established by letting it to take the unique path through the middle stage module $S_2(k)$. Hence, we can reduce the problem of routing connections in $C(n,m,r)$ to the following two graph coloring problems:

Weak Edge Coloring Problem: Given an I/O mapping graph $G(V_1, V_2; E)$, properly color edges in E. If we can find a weak edge-coloring of $G(V_1, V_2; E)$ using at most c_1 different colors, we call this coloring *a (weak)*[1] *c_1-edge coloring* of $G(V_1, V_2; E)$.

Strong Edge Coloring Problem: Given an I/O mapping graph $G(V_1, V_2; E)$ with a subset E^c of properly colored edges of E, color edges in $E - E^c$ without changing the colors of edges in E^c so that all edges in E are properly colored. If we can find a strong edge-coloring of $G(V_1, V_2; E)$ using at most c_2 different colors, we call this coloring *a strong c_2-edge coloring* of $G(V_1, V_2; E)$.

Note that the degree of any I/O mapping graph $G(V_1, V_2; E)$ of $C(n,m,r)$ is n. Let $\Delta(G)$ denote the degree of a bipartite multigraph G. The following two lemmas guarantee that an m-edge coloring is always possible for any $G(V_1, V_2; E)$ of a rearrangeable nonblocking $C(n,m,r)$, and a strong m-edge coloring is always possible for any $G(V_1, V_2; E)$ of a strictly nonblocking $C(n,m,r)$.

LEMMA 1. *[2] Every bipartite multigraph G has a $\Delta(G)$-edge coloring.*

LEMMA 2. *[12] Any multigraph G has a strong $(2\Delta(G) - 1)$-edge coloring.*

[1]The definition of weak edge-coloring is the same as the definition of edge-coloring in graph theory. Thus we omit "weak" in the rest of this paper.

3. A PARALLEL ROUTING ALGORITHM

In this section, we present a parallel algorithm for edge coloring of bipartite multigraphs of degree n. This algorithm can be used to find m-edge coloring of bipartite multigraphs of degree n, where $m \geq n$, and strong m-edge coloring of bipartite multigraphs of degree n, where $m \geq 2n-1$. The approach of solving routing problems by graph edge-coloring has been used extensively (e.g. [5, 7, 10, 8, 12, 14]). Unlike existing work, which aimed at finding theoretically fastest algorithms, our objective is to design a fast algorithm *COLORING* that is feasible for hardware implementation.

There are r^2 processing elements $PE_{i,j}$, $0 \leq i,j \leq r-1$. Let $Color1[0..r-1, 0..m-1]$ be an $r \times m$ Boolean array such that $Color1[i,k] = 1$ if and only if color k is available for any edge incident at node v'_i in V_1. Similarly, we define $Color2[0..r-1, 0..m-1]$ as an $r \times m$ Boolean array such that $Color2[j,k] = 1$ if and only if color k is available for any edge incident at node v''_j in V_2. As input of *COLORING*, we are given a bipartite multigraph $G(V_1, V_2; E)$ of degree n with E^c being the subset of colored edges. We use $E_{i,j}$ to denote the subset of edges of E with one end node incident at v'_i and the other incident at v''_j.

algorithm *COLORING*$(G(V_1, V_2; E))$
begin
  for $k = 0$ **to** $m-1$ **do**
    for all $PE_{i,j}$, $0 \leq i,j \leq r-1$, **do in parallel**
      $c_{i,j,k} := (i + j + k) \bmod m$;
      if there is an uncolored edge e in $E_{i,j}$
      and $Color1[i, c_{i,j,k}] \wedge Color2[j, c_{i,j,k}] = 1$
      then begin
        select one uncolored edge e from $E_{i,j}$;
        assign color $c_{i,j,k}$ to edge e and mark e
        "colored";
        $Color1[i, c_{i,j,k}] := 0; Color2[j, c_{i,j,k}] := 0$
      end
end

THEOREM 1. *Let $G(V_1, V_2; E)$ be any bipartite multigraph such that $|V_1| = |V_2| = r$ and its degree is n, and E^c be a subset of edges that are properly colored with no more than m colors. If $m \geq 2n-1$, then algorithm COLORING colors the edges in $E^{uc} = E - E^c$ properly without changing the colors of edges in E^c. If $m \geq n$ and $E^c = \emptyset$, then algorithm COLORING properly colors the edges in E.*

PROOF. The algorithm has m iterations. The proof consists of the following simple facts:

(i) In iteration k, one uncolored edge, if any, in each $E_{i,j}$, $0 \leq i,j, \leq r-1$, is selected. Note that such a selected edge may or may not be colored in the iteration, and it is colored if and only if $Color1[i, c_{i,j,k}] \wedge Color2[j, c_{i,j,k}] = 1$ and the color used is $(i + j + k) \bmod m$.

(ii) In iteration k, if two edges, one in $E_{i,j}$ and one in $E_{p,q}$ are assigned the same color, then $i \neq p$ and $j \neq q$. This is because $(i+j+k) \bmod m = (p+q+k) \bmod m$ only if $i \neq p$ and $j \neq q$.

(iii) For each uncolored edge, all m colors are tried.

(iv) After m iterations all uncolored edges are assigned colors and no two edges incident at a common node are

assigned the same color because of (i), (ii) and (iii), and Lemma 1 and Lemma 2.

This completes the proof of the theorem. □

Using the nr-processor EREW PRAM abstract parallel machine model, each iteration of the outer for-loop of *COLORING* can be easily carried out in $O(1)$ time. Similarly, using an nr-processor parallel computer system with nr processors connected as a complete graph, each iteration can be carried out in $O(1)$ time. The total time required by *COLORING* is $O(m)$. Since for $C(n, m, r)$ to be rearrangeable nonblocking and strictly nonblocking, $m \geq n$ and $m \geq 2n - 1$, respectively, we have the following result:

COROLLARY 1. *For an $N \times N$ rearrangeable or strictly nonblocking $C(n, m, r)$ network with $m = O(n) = O(\sqrt{N})$, algorithm COLORING takes $O(\sqrt{N})$ time to route any I/O mapping on an N-processor EREW PRAM or a completely connected N-processor parallel computer system in $O(\sqrt{N})$ time.*

PRAM model is not realistic. A completely connected multiprocessor system requires $O(N^2)$ interconnection complexity, making it unscalable and too expensive to build. In the next section, we present a parallel processing architecture for algorithm *COLORING*. This architecture has linear interconnection complexity and can achieve the same performance as PRAM and a completely connected N-processor parallel computer system.

4. DISTRIBUTED PIPELINING ROUTING ARCHITECTURE

In this section, we introduce our hardware routing architecture for $C(n, m, r)$ networks. We name our architecture as the *Distributed Pipelining Routing* (DPR) architecture. We consider a subclass (rearrangeable or strictly) nonblocking Clos networks $C(n, m, r)$ such that $n = r$. The reason of choosing $n = r$ is because for such networks our DPR design is optimally balanced. Our design can be extended to perform routing in nonblocking $C(n, m, r)$ networks of arbitrary valid values of n, m and r. For brevity, we omit this extension in this paper.

4.1 Basic Hardware Features

The design of DPR for (rearrangeable or strictly) nonblocking Clos networks $C(n, m, n)$ is based on the structure of $G(V_1, V_2; E)$ and algorithm *COLORING*. It has the following features:

(1) Corresponding to each node v_i' in V_1 (i.e. each input group IG_i of $C(n, m, n)$), there is a ring IR_i of n *processing elements* (PEs) denoted by $IP_{i,j}$, $0 \leq j \leq n-1$. $IP_{i,j}$ is connected to $IP_{i,(j-1) \bmod n}$ by a unidirectional link. Similarly, there is a ring OR_i of n PEs denoted by $OP_{i,j}$, $0 \leq j \leq n-1$, corresponding to each node v_i'' in V_2 (i.e. each output group OG_i) such that $OP_{i,j}$ is connected to $OP_{i,(j-1) \bmod n}$ by a unidirectional link. IR_i (resp. OR_i) is called *input ring* (resp. *output ring*) i.

(2) A request for connection from input $I_{i,p}$ to output $O_{j,q}$ is received by $IP_{i,p}$. $IP_{i,p}$ forms a 3-tuple $(p, j, \_)$, where $\_$ is a blank field to be filled. After routing, a 3-tuple (p, j, c) is returned to $I_{i,p}$, where c corresponds to a color, indicating that a connection from $I_{i,p}$ to $O_{j,q}$ is going to go through middle stage module $S_2(c)$. Note that upon receiving (p, j, c), $I_{i,p}$ can make a connection from $I_{i,p}$ to output $O_{j,q}$ by *self-routing*. $IP_{i,j}$ is responsible for finding c for I/O pair $(I_{i,p}, O_{j,q})$. We call such a 3-tuple a *connection addition request token* (CAR token). CAR tokens of each input group IG_i flow around ring IR_i in pipelined fashion.

(3) Each $IP_{i,j}$ in ring IR_i has a 1-bit cell denoted by $c'_{i,j}$, a circuit capable of carrying out simple Boolean logic operations, a couple of working registers, and a small amount of memory. In IR_i, the last processing element, $IP_{i,n-1}$, has $m - n$ additional cells. A total of m cells form an m-cell ring, with 1 cell in each of the first $n - 1$ PEs, and $m - n + 1$ cells in the last PE, $IP_{i,n-1}$. This m-cell ring is called an *input circular shift register* (ICSR). For easy reference, we denote this m-cell ring by $ICSR_i$.

(4) Each $OP_{i,j}$ in ring OR_i has a 1-bit cell denoted by $c''_{i,j}$. The last processing element, $OP_{i,n-1}$, in OR_i has $m-n$ additional cells. A total of m cells form an m-cell ring, with 1 cell in each of the first $n - 1$ PEs, and $m - n + 1$ cells in the last PE, $OP_{i,n-1}$. This m-cell ring is called an *output circular shift register* (OCSR), and we denote it by $OCSR_i$.

(5) Each $IP_{i,j}$ is associated with a modulo-m circular counter $CC_{i,j}$. Thus, there are n circular counters in each IR_i, and a total of $N = n^2$ such counters. (For convenience, we also assume that each $OP_{i,j}$ is associated with a modulo-m circular counter $CC_{i,j}$. We denote the counters for IPs as $CC^1_{i,j}$s and counters for OPs as $CC^2_{i,j}$s.) All counters are incremented by a common system clock. The initial value of $CC_{i,j}$ is set $(i + j) \bmod m$.

(6) Each $IP_{i,j}$ is connected to $OP_{j,i}$ by a 1-bit bidirectional link. Thus, Both $IP_{i,j}$ and $OP_{j,i}$ can access $c''_{j,i}$.

(7) All IPs, OPs, $ICSR$s and $OCSR$s are under the control of a common system clock.

The architecture of input rings IR_i and output rings QR_j for $C(5, 9, 5)$ are shown in Figure 4. Figure 5 shows the overall structure of our DPR architecture.

4.2 Hardware Algorithm

Based on the architecture presented in the previous section, we have a hardware algorithm *HARD-COLORING* that implements algorithm *COLORING* of Section 3. The algorithm iterates continuously with each iteration consisting of three phases:

algorithm *HARD-COLORING*
begin
  repeat
    Phase 1;
    Phase 2;
    Phase 3
  for ever
end

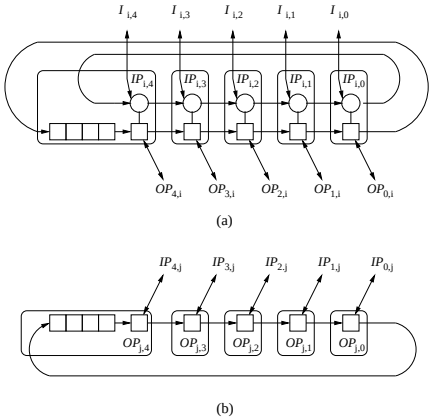

Figure 4: The input and output rings for $C(5, 9, 5)$. (a) IR_i. (b) OR_j.

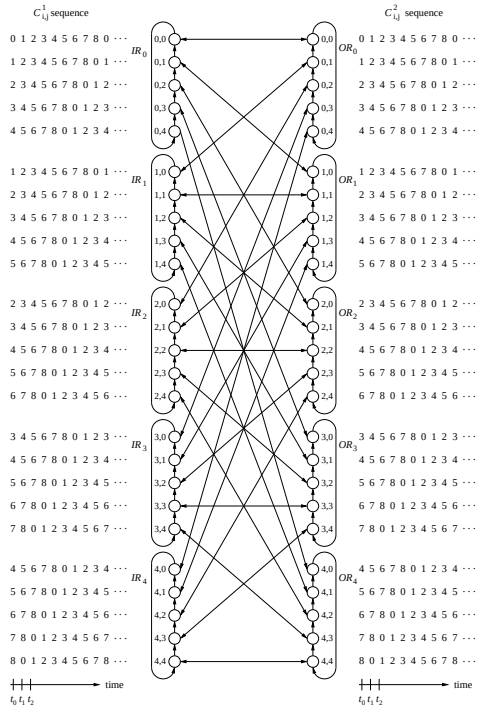

Figure 5: DPR architecture for Clos network $C(5, 9, 5)$. The sequence of values of each $CC^1_{i,j}$ and $CC^2_{i,j}$ is given on the left side and right side of $IP_{i,j}$ and $OP_{i,j}$, respectively.

We call each iteration of *HARD-COLORING* a *routing cycle*. We now present the details for each of the three phases of a routing cycle.

4.2.1 Phase 1 - Distribution of I/O Edges

In IR_i, $IP_{i,j}$ is responsible for processing all I/O connection requests from input group IG_i to output group OG_j. We call $IP_{i,j}$ the *agent for I/O connections from IG_i to OG_j*. Each I/O connection from input group IG_i to output group OG_j corresponds to an edge from v'_i to v''_j in the corresponding I/O mapping graph $G(V_1, V_2; E)$, and all CAR tokens from IG_i to OG_j correspond to the set $E^{uc}_{i,j} = E^{uc} \cap E_{i,j}$ of new edges of $G(V_1, V_2; E)$ to be colored (note: for reconfigurable nonblocking Clos networks, $E^{uc} = E$). The purpose of this phase is to prepare CAR tokens, and distribute them to their respective agent PEs. The operations of this phase are given below:

for all $IP_{i,p}$, $0 \le i, p \le n-1$, **do in parallel**
  if $I_{i,p}$ receives a request to be connected to $O_{j,q}$
  then prepare a CAR token $(p, j, \_)$
for $k = 1$ **to** $n-1$ **do**
  for all $IP_{i,j}$, $0 \le i, j \le n-1$, **do in parallel**
    if $IP_{i,j}$ has a CAR token $(p, j', \_)$ such that$p \neq j$
    then send $(p, j', \_)$ to $IP_{i,(j-1) \bmod n}$

4.2.2 Phase 2 - Color Assignment

This phase directly implements the algorithm *COLORING*. Boolean circular shift registers $ICSR_i$s (resp. $OCSR_i$s) are used to represent array $Color1$ (resp. $Color2$), and each $ICSR_i$ (resp. $OCSR_i$) is used to represent subarray $Color1[i, 0..m-1]$ (resp. $Color2[i, 0..m-1]$). The operations of this phase are as follows:

for $k = 0$ **to** $m-1$ **do**
begin
  for all $IP_{i,j}$, $0 \le i, j \le n-1$, **do in parallel**
  begin
    if $c'_{i,j} \wedge c''_{j,i} = 1$ and there is a CAR token $(p, j, \_)$ with its third field unfilled
    then assign $CC^1_{i,j}$ value (i.e. a color number) as the value of the third field of the token to obtain $(p, j, CC^1_{i,j})$, and set $c'_{i,j} := 0$ and $c''_{j,i} := 0$;
  increment $CC_{i,j}$ by 1;
  end
  for all $ICSR_i$ and all $OCSR_i$, $0 \le i \le n-1$, **do in parallel**
    perform a circular shift operation;
end

4.2.3 Phase 3 - Redistribution of I/O Edges

The purpose of this phase is to move each CAR token $(j'$, p, $c_{j',p})$ in IR_i at the end of Phase 2 back to $IP_{i,j'}$, where the token originally came from in Phase 1. Suppose that a token $T = (j', p, c_{j',p})$ is currently in $IP_{i,j}$, and to be sent back to $IP_{i,j'}$. We define T's *current forward distance to its destination* as $j - j'$ if $j' \le j$, and $n - j' + j$ if $j' > j$. The current forward distance of T to its destination is the minimum number of hops in IR_i for T to reach its destination $IP_{i,j}$. The operations of this phase are given below:

for $k = 1$ **to** $n - 1$ **do**
  for all $IP_{i,j}$, $0 \le i, j \le n-1$, **do in parallel**
    if $IP_{i,j}$ has a token T such that its forward distance to its destination is k
    then send T to $IP_{i,(j-1) \bmod n}$

EXAMPLE 2. *Consider the I/O mapping and colors for existing connections in Example 1, as shown in Figure 3. For IR_1 of the DPR for $C(5, 9, 5)$, there are five connection requests: $(I_{1,0}, O_{4,1})$, $(I_{1,1}, O_{3,3})$, $(I_{1,2}, O_{0,3})$, $(I_{1,3}, O_{1,2})$, and $(I_{1,4}, O_{0,1})$. In Phase 1, five CAR tokens, $(0, 4, \_)$, $(1, 3, \_)$, $(2, 0, \_)$, $(3, 1, \_)$, $(4, 0, \_)$, are prepared as shown in Figure 6(a). After Phase 1, all CAR tokens in IR_1 are received by their agent IPs, and Phase 2 starts using the configuration of Figure 6(b). Colors available are 1, 3, 4, 7 and 8. Colors 0, 2, 5, and 6 can not be used. The process of Phase 2 is shown in Figure 6(c). There are 9 iterations. In the figure, two rows of numbers are associated with each iteration. The first row contains values of c' cell, denoted by (x), or CC^1 counter and c' cell, denoted by $y(x)$, in IR_1. The second row contains values of the corresponding c'' cells and counters CC^2 in ORs. If there is a CAR token of empty "color" field in $IP_{1,j}$, and $c'_{1,j} \wedge c''_{j,1} = 1$, the current value of $CC^1_{1,j}$ is assigned to the CAR token, and both of $c'_{1,j}$ and $c''_{j,1}$ are reset to 0. In Phase 3, the filled CAR tokens are sent back to their origins as shown in Figure 6(d). Figure 3 shows that the colors assigned to the edges of new connections from IG_1 do not conflict with colors of the edges of the existing connections.*

■

4.3 Analysis of *HARD-COLORING*

We show that our hardware algorithm *HARD-COLORING* is equivalent to algorithm *COLORING* of Section 3 for $C(n, m, n)$. The following fact is obvious.

FACT 1. *At the end of Phase 1, all CAR tokens corresponding to edges in $E^{uc}_{i,j}$ of $G(V_1, V_2; E)$ are sent to their agent $IP_{i,j}$ in pipelined fashion. Note that for rearrangeable $C(n, m, n)$, $E^{uc} = E$.*

Denote the value of circular counter $CC^1_{i,j}$ and $CC^2_{i,j}$, $0 \le i, j \le n-1$, during the k-th iteration of Phase 2 of *HARD-COLORING* by $CC^1_{i,j}(k)$ and $CC^2_{j,i}(k)$, respectively. Since $CC^1_{i,j}(0) = CC^2_{i,j}(0) = (i + j) \bmod m$, and $CC^1_{i,j}$ and $CC^1_{i,j}$ are incremented at the end of each iteration, the following statement is true.

FACT 2. *Right before iteration k of Phase 2, $CC^1_{i,j}(k) = CC^2_{i,j}(k) = c_{i,j,k} = (i + j + k) \bmod m$, where $c_{i,j,k}$ is the local variable of $PE_{i,j}$ in the k-th iteration of COLORING.*

As noted earlier, the Boolean array $Color1[0..n-1, 0..m-1]$ (resp. $Color2[0..n-1, 0..m-1]$) used in algorithm *COLORING* is represented by $ICSR_i$s (resp. $OCSR_i$s), with $ICSR_i$ (resp. $OCSR_i$) representing subarray $Color1[i, 0..m-1]$ (resp. $Color2[i, 0..m-1]$) for $C(n, m, n)$. Denote the value of $c'_{i,j}$ and $c''_{i,j}$ in iteration k of *HARD-COLORING* by $c'_{i,j}(k)$ and $c''_{i,j}(k)$, respectively, and define that $c'_{i,j}(0)$ and $c''_{i,j}(0)$ correspond to $Color1(i, c_{i,j,0})$ and $COLOR2(i, c_{i,j,0})$ of iteration 0 of *COLORING*, respectively.

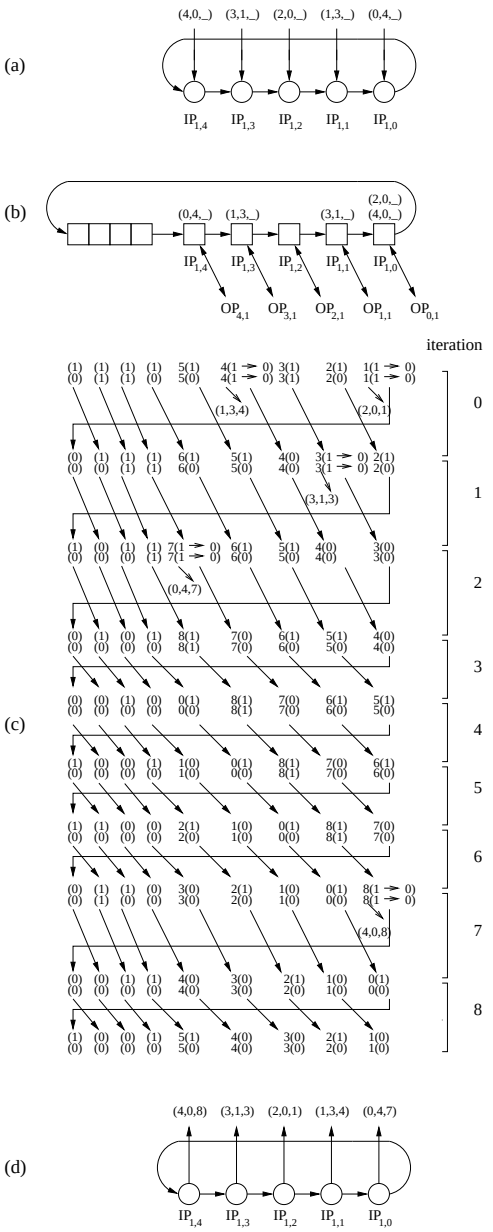

Figure 6: Operations of input ring IR_1 of $C(5, 9, 5)$ on Example 1. (a) Configuration after CAR preparation in Phase 1. (b) Configuration at the end of Phase 1. (c) Process of Phase 2. (d) Configuration after Phase 3.

LEMMA 3. *In iteration k of HARD-COLORING, $c'_{i,j}(k)$ and $c''_{i,j}(k)$ accessed by $IP_{i,j}$ correspond to $Color1[i, c_{i,j,k}]$ and $Color2\ [j, c_{i,j,k}]$ accessed by $PE_{i,j}$ in the k-th iteration of COLORING, respectively.*

PROOF. Since the proofs for $c'_{i,j}(k)$ and $c''_{i,j}(k)$ are the same, we only consider $c'_{i,j}(k)$. *HARD-COLORING* implements *COLORING* for $r = n$. In the k-th iteration, $PE_{i,j}$ in *COLORING* only accesses $Color1[i, c_{i,j,k}]$ of array $Color1$. Similarly, in the k-th iteration, $IP_{i,j}$ in *HARD-COLORING* only accesses $c'_{i,j}(k)$ of c' cells. By definition of $c'_{i,j}(0)$, the claim is obviously true for $k = 0$. Let $J = (j + 1) \bmod m$. Assume that the claim is true for k' and consider the case of iteration $k' + 1$. Let $J = (j + 1) \bmod m$. $PE_{i,j}$ accesses $Color1[i, c_{i,j,k'+1}]$, which was accessed by $PE_{i,J}$ in iteration k'. $IP_{i,j}$ accesses $c'_{i,j}(k' + 1)$. By the shifting operation performed at the end of iteration k', $c'_{i,j}(k' + 1)$ is $c'_{i,J}(k')$, which implies that it was accessed by $IP_{i,J}$ in iteration k'. By the hypothesis, $c'_{i,j}(k' + 1)$ accessed by $IP_{i,j}$ corresponds to $Color1[i, c_{i,j,k'+1}]$ accessed by $PE_{i,j}$ in the k-th iteration of *COLORING*, respectively. This completes the induction. □

Based on Facts 1 and 2, Lemma 3 and Theorem 1, Phase 2 exactly implements algorithm *COLORING*. By the correctness of *COLORING*, Phase 2 of *HARD-COLORING* properly colors uncolored edges in $G(V_1, V_2; E)$ for $C(n, m, n)$.

Since all I/O connections considered are partial or full permutations, no two CAR tokens are destined for the same PE in each iteration of Phase 3 of *HARD-COLORING* in Phase 3 of *HARD-COLORING*. Scheduled by their forward distances, all tokens are sent back to their origins in pipelined fashion without conflict after $n - 1$ iterations.

Let us consider the time complexity of *HARD-COLORING* now. Obviously, Phase 1 takes $n - 1$ steps, with each step taking $O(1)$ time. Phase 2 takes m steps, with each step taking $O(1)$ time if all tokens in each $IP_{i,j}$ can be found in $O(1)$ time. This can be easily done by linking all tokens in each $IP_{i,j}$ as a queue in arbitrary order. While Phase 3 also takes $n - 1$ steps, we must ensure that each step takes $O(1)$ time. In $IP_{i,j}$, by maintaining an $(n-1)$-element array with its d-th element containing a pointer to the token, if any, with forward distance d, finding the token with forward distance k in the k-th iteration of Phase 3 can be done in $O(1)$ time (note: the token with forward distance 0 remains in $IP_{i,j}$ in the remaining iterations of Phase 3). The total number of steps is $2n + m - 2$. Since $O(m) = O(n)$, where $n = \sqrt{N}$ for rearrangeable or strictly nonblocking $C(n, m, n)$ networks, we have the following result:

THEOREM 2. *For an $N \times N$ rearrangeable or strictly nonblocking $C(n, m, n)$ network with $m = O(n) = O(\sqrt{N})$, algorithm HARD-COLORING takes $O(\sqrt{N})$ steps to route any I/O mapping in one routing cycle on the DPR architecture.*

Since the pair $IP_{i,j}$ and $OP_{j,i}$ are connected by a link, the value of $CC^2_{j,i}$ is implied by the value of $CC^1_{i,j}$, $CC^2_{j,i}$ is redundant; its introduction is meerly for the purpose of easy analysis. To see this, consider the control architecture for $C(5, 9, 5)$ shown in Figure 5. The sequence of values of each $CC^1_{i,j}$ and each $CC^2_{i,j}$s is shown on the left and right side of the figure, respectively.

In the DPR architecture, routing decisions are made in parallel in different IRs and ORs. The elements of these rings can be distributed in the input ports. The interconnection of the DPR architecture is very sparse. It is easy to see that the total number of interconnections connecting IPs and OPs in the DPR architecture for $C(n, m, n)$ is $3N$, a linear function of the inputs (or outputs).

5. ADDITIONAL FEATURES

5.1 Connection Tear-Down

Algorithm *HARD-COLORING* finds conflict-free I/O paths for new connections in a Clos network. When an input $I_{i,p}$ wants to tear down a connection for the I/O pair $(I_{i,p}, O_{j,q})$, it can simply stop using its connection path. However, in order to make used internal links of $C(n, m, n)$ by connection $(I_{i,p}, O_{j,q})$ available for other connections, the current state of the network must be updated.

For a rearrangeable Clos network for (slotted) packet switching, this can be done by simply setting all cells in $ICSR_i$s and $OCSR_i$s to 1 concurrently. For circuit switching, the situation can be more complicated if input $I_{i,p}$ immediately wants to establish a connection to another output $O_{j',q'}$ after tearing down its current $(I_{i,p}, O_{j,q})$ connection, and, furthermore, several inputs may want to tear down their current connections and establish new connections. We can slightly modify the three phases of algorithm *HARD-COLORING* to cope with such situations, making the DPR design for strictly nonblocking $C(n, m, n)$ complete.

In Phase 1, each $IP_{i,p}$ can prepare at most two request tokens $(p, j, c)^{\#}$ and $(p, j', \_)$, where $(p, j, c)^{\#}$ is for deleting its current connection from $I_{i,p}$ to $O_{j,q}$ going through middle stage module $S_2(c)$ (an edge in $G(V_1, V_2; E)$ connecting v'_i and v''_p with color c) and $(p, j', \_)$ is for adding a new connection from $I_{i,p}$ to $O_{j',q'}$. We name $(p, j, c)^{\#}$ as a *connection deletion request token* (CDR token). Symbol #, which can be represented by a binary bit, is used to distinguish CDR tokens from CAR tokens. There are four possibilities for each input: no CDR token and no CAR token, a CDR token but no CAR token, no CDR token but a CAR token, and a CDR token and a CAR token. Then, CDR token $(p, j, c)^{\#}$ (if any) is sent to its agent $IP_{i,j}$, and CAR token $(p, j', \_)$ (if any) is sent to its agent $IP_{i,j'}$. Tokens from all $IP_{i,j}$s in IR_i are sent to their destinations (agents) in pipelined fashion. Phase 2 then is modified to have two subphases. Subphase 2.1, called *color erase subphase*, consists of m steps as follows:

for $k = 0$ **to** $m - 1$ **do**
begin
  for all $IP_{i,j}$, $0 \le i, j \le n - 1$, **do in parallel**
  begin
    if there is a CDR token $(j', j, k)^{\#}$ and $k = CC_{i,j}$
then set $c'_{i,j} := 1$ and $c''_{j,i} := 1$ and
    discard token $(j', j, k)^{\#}$;
    increment $CC_{i,j}$ by 1;
  end
  for all $ICSR_i$ and all $OCSR_i$, $0 \le i \le n - 1$,
  do in parallel
    perform a circular shift operation;
end

Subphase 2.2 is *color assignment subphase*, which is ex-

actly the same as Phase 2 of the original *HARD-COLORING*. This modified algorithm has $2m + 2n - 2$ steps in total. In summary, we have the following result:

THEOREM 3. *For an $N \times N$ strictly nonblocking $C(n, m, n)$ network with $m = O(n) = O(\sqrt{N})$, DPR architecture can satisfy any group of connection/disconnection requests in $O(\sqrt{N})$ steps.*

5.2 Speedup of Color Erase and Color Assignment

For strictly nonblocking $C(n, m, n)$, $m \geq 2n - 1$. Then, each routing cycle of the modified *HARD-COLORING* for $C(n, 2n - 1, n)$ takes $6n - 4$ steps. We introduce a scheme called *doubled shift registers*, particularly designed for strictly nonblocking $C(n, 2n - 1, n)$, that can reduce the number of each subphases of Phase 2 to n, resulting $4n - 2$ steps in a routing cycle.

Instead of using one $ICSR_i$ in each IR_i, we use two $ICSR_i$s, $ICSR_i^1$ and $ICSR_i^2$, each having n cells. Similarly, we use two n-cell $OCSR_i$s, $OCSR_i^1$ and $OCSR_i^2$, in each OR_i. The j-th cell of $ICSR_i^1$ (resp. $OCSR_i^1$) and the j-th cell of $ICSR_i^2$ (resp. $OCSR_i^2$) reside in the same IP (resp. OP). The j-th cell of $ICSR_i^1$ and the j-th cell of $ICSR_i^2$ are respectively connected to the i-th cell of $OCSR_j^1$ and the i-th cell of $OCSR_j^2$. The shift operations of $ICSR_i^1$s, $ICSR_i^2$s, $OCSR_i^1$s and $OCSR_i^2$s are performed synchronously. $ICSR_i^1$ and $OCSR_i^1$ are used to maintain the availability of colors 0 through color $n - 1$, and $ICSR_i^2$ and $OCSR_i^2$ are used to maintain the availability of colors n through color $2n - 2$. Color $2n - 1$ is never used, and the values of its corresponding cells are always enforced to be 0 (i.e. unavailable). Each $IP_{i,j}$ has two modulo-n circular counters $CC_{i,j}^{1,1}$ and $CC_{i,j}^{1,2}$ such that $CC_{i,j}^{1,2} = (CC_{i,j}^{1,1} + n) \bmod 2n$. The modified input ring IR and output ring OR structures for $C(5, 9, 5)$ are shown in Figures 7. For convenience, we assume that there are two modulo-n circular counters $CC_{i,j}^{2,1}$ and $CC_{i,j}^{2,2}$ in $OP_{i,j}$. For Clos network $C(5, 9, 5)$, the sequence of values of each $(CC_{i,j}^{1,1}, CC_{i,j}^{1,2})$ and each $(CC_{i,j}^{2,1}, CC_{i,j}^{2,2})$ is given on the left side and right side of $IP_{i,j}$ and $OP_{i,j}$, respectively, in Figure 8.

With this modified architecture, two pairs of cells, $(ICSR_{i,j}^1, OCSR_{j,i}^1)$ and $(ICSR_{i,j}^2, OCSR_{j,i}^2)$ can be accessed by $IP_{i,j}$ for color status. Thus, up to two CDR tokens and two CAR tokens can be processed by $IP_{i,j}$ in each step of Phase 2 of *HARD-COLORING*. Special digital logic circuit associated with working registers must be designed to handle different combinations of 1 and 2 tokens. In order to access two tokens stored in the memory of each $IP_{i,j}$, 2-port memory or simple memory interleaving can be used. To further reduce the effective memory access time, these techniques can be combined with latency hiding, which overlaps memory access with shifting $ICSR_i^1$s, $ICSR_i^2$s, $OCSR_i^1$s, and $OCSR_i^2$s. It is easy to verify that, by using $CC_{i,j}^{1,1}$, $CC_{i,j}^{1,2}$, $CC_{i,j}^{2,1}$, and $CC_{i,j}^{2,2}$ (where $CC_{i,j}^{2,1}$ and $CC_{i,j}^{2,2}$ are implied by $CC_{i,j}^{1,1}$ and $CC_{i,j}^{1,2}$, respectively) to access "colors", this scheme reduces the number of steps used in Phase 2 by half.

5.3 Overlapping Phases

A rearrangeable Clos network can be used as switching fabric in packet routers as follows. A packet received in an

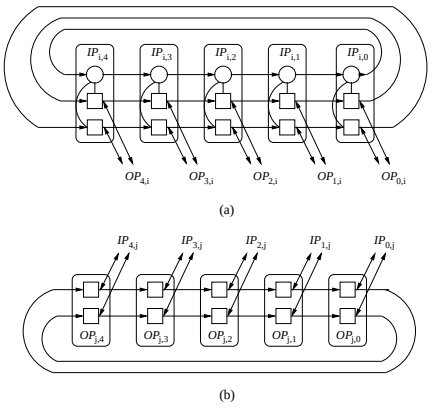

Figure 7: The input and output rings with doubled shifter registers for $C(5, 9, 5)$. (a) IR_i. (b) OR_j.

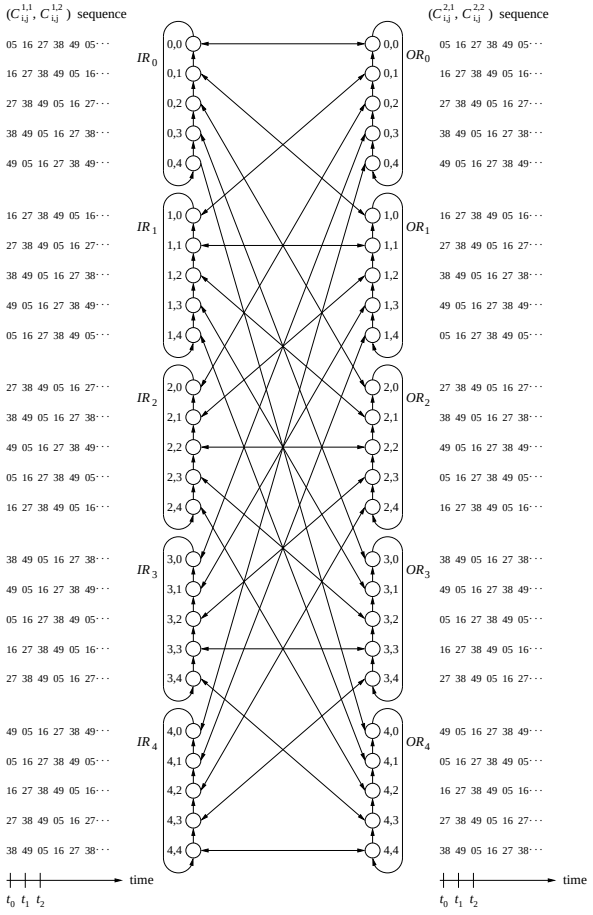

Figure 8: DPR architecture with doubled shifter registers for Clos network $C(5, 9, 5)$. The sequence of values of each $(CC_{i,j}^{1,1}, CC_{i,j}^{1,2})$ and each $(CC_{i,j}^{2,1}, CC_{i,j}^{2,2})$ is given on the left side and right side of $IP_{i,j}$ and $OP_{i,j}$, respectively.

input port is divided into fixed size cells, which are scheduled by arbitration circuits (collectively called *cell scheduler*) so that for each cell slot the connections for a (partial) I/O permutation can be established. A set of cells forming a permutation are transmitted over the switching fabric in a cell slot. In output ports, cells are then reassembled into packets. To meet the stringent timing requirement for each of these tasks, a pipelined approach can be adopted.

We implement an input ring IR_i by three rings, IR_i^1, IR_i^2 and IR_i^3, one corresponding to a phase in the 3-phase *HARD-COLORING* algorithm as shown in Figure 9. Then, phases in consecutive routing cycles can be overlapped to achieve improved performance. The principle used is *two-dimensional pipelining*. Let $IP_{i,j}^s$, $1 \leq s \leq 3$, denote the j-th IP in ring IR_i^s. IPs in IRs are connected as a two-dimensional torus. Within each IR, CAR tokens are moved or processed in pipelined fashion using vertical links. As soon as a CAR token reaches its agent $IP_{i,j}^1$, it is passed to $IP_{i,j}^2$ of IR_i^2 using the horizontal link between them. Similarly, as soon as a CAR token is assigned a color in $IP_{i,j}^2$, it is passed to $IP_{i,j}^3$ using the horizontal links connecting the two IPs. Considering arbitration and cell transmission as two additional phases, and, for simplicity, assuming that each phase takes time that is equal to a cell slot, we obtain a timing diagram of all pipelined operations in cell switching as shown in Figure 10. Clearly, using this two-dimensional pipeline, each routing cycle takes n steps effectively for for $C(n, n, n)$.

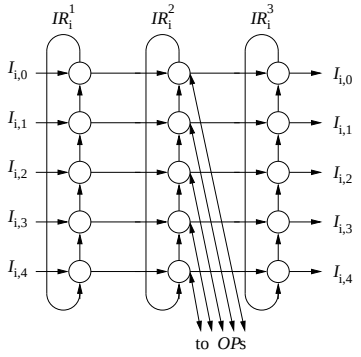

Figure 9: Pipelined input ring IR_i structure that overlaps phases of 3 consecutive routing cycles for routing rearrangeable $C(n, n, n)$.

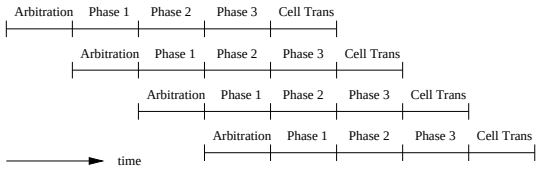

Figure 10: Timing diagram of pipelined cell switching using rearrangeable $C(n, n, n)$.

For routing strictly nonblocking $C(n, 2n-1, n)$ using doubled shift registers (refer to Section 5.2), each of Phase 1 and Phase 3 takes $n-1$ steps, and each of the two subphases of Phase 2 takes n steps. We can implement an input ring IR_i by four rings, IR_i^1, $IR_i^{2.1}$, $IR_i^{2.2}$ and IR_i^3 as shown in Figure 11(a), and implement an output ring OR_j by two rings $OR_j^{2.1}$ and $OR_j^{2.2}$ as shown in Figure 11(b). IR_i^1s are used for executing Phase 1, IR_i^3s are used for executing Phase 3, $IR_i^{2.1}$s and $OR_i^{2.1}$s are used for executing Phase 2.1 (first subphase of Phase 2), and $IR_i^{2.2}$s and $OR_i^{2.2}$s are used for executing Phase 2.2. Then, phases of consecutive routing cycles can be performed in an overlapped way so that any group of disconnection/connection requests can be processed in $4n$ steps. Similar to cell switching discussed above, this performance can be maintained if arbitration operations for resolving output contentions are included as an additional phase. Furthermore, due to pipelining, groups of disconnection/connection requests can be sampled more frequently as every n steps, and every request group can be satisfied in n steps effectively.

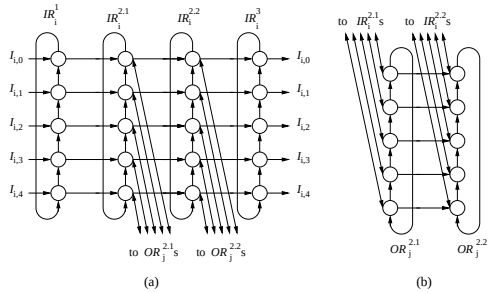

Figure 11: Overlapping routing cycles for routing strictly nonblocking $C(n, 2n-1, n)$. (a) Pipelined input ring IR_i structure. (b) Pipelined output ring OR_j structure.

5.4 Phaseless Approach

We present another scheme for improving the routing performance of $C(n, 2n-1, n)$ with doubled shift registers. Each input ring IR and output ring OR in the corresponding DPR architecture has exactly n IPs and n OPs, respectively. Each $ICSR$ has exactly n cells, one in each IP. Similarly, each $OCSR$ has exactly n cells, one in each OP. Thus, the four phases (the two subphases of Phase 2 for processing CDRs and CARs are counted as two phases) can be interleaved in a synchronous way. We call this scheme the *phaseless scheme*. Each step in this scheme is called a *composed step*. A composed step performs four *substeps*, each corresponding to one step in each (sub)phase of the phased approach. In the phaseless scheme, a token is available for processing in the next (sub)phase as soon as it is processed in a substep. Thus, in the best case, a token can be completely processed in $O(1)$ time. For example, to make a connection from $I_{i,j}$ to $O_{j,i}$, its CAR token $(j, j, \_)$ has to go through at least the entire Phase 1 and the first subphase of Phase 2 in the phased approach. But it may be completely processed in one composed step (which has 4 substeps).

In the phased approach, there is a well-defined schedule for the tokens to go through its ring IR without conflict. In the phaseless scheme, tokens in the same $IP_{i,j}$ can be put into a priority queue, and they are sent to $IP_{i,(j-1) \bmod n}$ in the order enforced by the queue. The simplest queue is the FIFO queue.

6. CONCLUDING REMARKS

We presented an efficient parallel algorithm *COLORING* for routing Clos networks, and a hardware implementation *HARD-COLORING* of this algorithm. The resulting hardware solution is a high-speed DPR architecture that employs distributed and parallel processing. Pipelining is entensively used at different levels. The overall structure of this architecture is simple and scalable because of its linear interconnection complexity. Our algorithm takes $O(\sqrt{N})$ steps for optimized nonblocking $C(n, m, r)$ networks. The constants associated with the time complexity is very small. Using phase-overlapping method, our architecture effectively takes exactly $\sqrt{N}$ steps, each requiring simple data movement and Boolean logic operations.

The best known parallel routing algorithm for rearrangeable $N \times N$ Clos network $C(n, m, r)$ has time complexity $O(\log^2 N)$, when m is a power of 2. For arbitrary eligible m, the best known parallel time complexity is $O(\log^3 N)$. The constants associated with these complexities are large, and the parallel machine models used for deriving these complexities are either unrealistic or impractical. To the best of our knowledge, for strictly nonblocking Clos networks, our algorithm is so far the best.

Let us compare the relative magnitudes of N, $\log_2^2 N$, $\log_2^3 N$ and $\sqrt{N}$ in the following table:

N	$\log_2^2 N$	$\log_2^3 N$	$\sqrt{N}$
16	16	64	4
64	36	216	8
256	64	512	16
1,024	100	1,000	32
4,096	144	1,728	64
16,384	196	2,744	128
65,536	256	4,096	256

In general, comparing two time complexities $O(f(N))$ and $O(g(N))$ by comparing $f(N)$ and $g(N)$ does not make sense. In our case, however, such a comparison reveals that the actual performance of our DPR architecture is superior over known parallel algorithms for the following reasons. First, our DPR architecture is a realistic model with linear interconnection complexity, whereas PRAM is not physically implementable and a completely connected multiprocessor system is not scalable because of its $O(N^2)$ interconnection complexity. Second, the time complexity of DPR architecture is measured by the number of steps performed. By our analysis, this number is no more than $c \cdot \sqrt{N}$, where c is a single-digit constant. In the worst case, each step consists of a shift operation, a logical AND or reset operation, and a memory access. In contrast, known unrealistic parallel algorithms involve complex shared-memory or distributed memory data communication operations and arithmetic operations, and the coefficients associated with their time complexities are large. Third, since an $N \times N$ reconfigurable or strictly nonblocking Clos network has at least $N^{1.5}$ crosspoints to be controlled and considerable additional hardware is required to integrate it into a network switch, it is quite safe to say that $N = 65,536$ is an upper bound for practical switch sizes according to the current and foreseeable hardware technologies. Therefore, we claim that for practical switch sizes N the actual performance of our solution is better than the known $O(\log^2 N)$-time and $O(\log^3 N)$-time parallel algorithms.

For cell/packet switching, we considered using a separate hardware called scheduler to find permutations by resolving contentions. A hardware scheduling algorithm such as the ones of time complexity $O(\log^2 N)$ in [13] can be used. In [4], the relatively scalable schedulers are defined. A scheduler is relatively scalable with respect to a switching network if its interconnection complexity is not larger than the interconnection complexity of its associated nonblocking switching network. The scheduler of [13] are not scalable with respect to Clos networks It remains a challenging open problem of designing a high-performance scalable scheduler for Clos networks. One possible way is to incorporate contention resolution functions into our routing architecture.

7. REFERENCES

[1] V. E. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.

[2] J. Bondy and U. Murty. *Graph Theory with Applications*. American Elsevier, MacMillan, New York, London, 1976.

[3] J. Bondy and U. Murty. *The Mathematical Theory of Nonblocking Switching Networks*. World Scientific, River Edge, NJ, USA, 1998.

[4] S. Z. C. Li and M. Yang. Scalable schedulers for high-performance switches. In *Proceedings of Workshop on High Performance Switching and Routing*, pages 198–202. IEEE, April 2004.

[5] J. Carpinelli and A. Y. Oruc. Applications of matching and edge-coloring algorithms to routing in clos networks. *Networks*, 24:319–326, September 1994.

[6] C. Chen and A. Frank. A study of non-blocking switching networks. *Bell System Technical Journal*, 32:406–424, March 1953.

[7] C. Chen and A. Frank. On programmable parallel data routing networks via cross-bar switches for multiple element computer architectures. *Lecture Notes In Computer Science*, 24:338 – 369, 1974.

[8] N. P. G. F. Lev and L. G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Comput.*, 30:93–100, February 1981.

[9] J. Jaja. *Introduction to Parallel Algorithms*. Addison-Wesley, Reading, MA, 1992.

[10] O. Kariv and H. Gabow. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput.*, 11(1):117–129, 1982.

[11] T. Lee and S. Liew. Parallel routing algorithms in benes-clos networks. *IEEE Trans. on Comm.*, 50:1841–1847, 2002.

[12] E. Lu and S. Q. Zheng. Parallel routing algorithms for nonblocking electronic and photonic switching networks. *IEEE Trans. on Parallel and Distributed Systems*, 16(8):702–713, August 2005.

[13] V. A. N. McKeown, A. Mekkittikul and J. Walrand. Achieving 100% throughput in an input queued switch. *IEEE Trans. on Comm.*, 47:1260–1267, 1999.

[14] N. Nassimi and S. Sahni. Parallel algorithms to set up the benes permutation network. *IEEE Trans. on Comput.*, 31(2):148–154, February 1982.

[15] K. O. R. Cole and S. Schirra. Edge coloring bipartite multigraphs in $o(e \log d)$ time. *Combinatorica*, 21(1):5–12, 2001.