Off-Line Permutation Routing on Circuit-Switched Fixed-Routing Networks

Abdou Youssef

Department of EECS, The George Washington University, Washington, D.C. 20052

Circuit-switched fixed routing (CSFR) is an increasingly popular communication model wherein there is between every source–destination pair a single path that is system-determined by a fixed-routing rule. This paper studies the new problem of off-line permutation scheduling on linear arrays, rings, hypercubes, and 2-dimensional arrays, assuming the CSFR model. Optimal permutation scheduling involves finding a minimum number of subsets of nonconflicting source–destination paths. Every subset of paths can be established to run in one pass. In this paper, optimal permutation scheduling on linear arrays is shown to be linear, and on rings, NP-complete. On hypercubes, the problem is NP-complete. However, we will give an $O(N \log N)$ algorithm that routes any permutation in two passes if the model is relaxed to allow for two routing rules, namely, the so-called e-cube rule and the e^{-1}-cube rule. This complexity is reduced to $O(N)$ hypercube-parallel time. Finally, an $O(N \log^2 N)$ bipartite-matching-based algorithm will be designed to schedule any permutation on $p \times q$ meshes/tori in q passes.

1. INTRODUCTION

The circuit-switched fixed-routing model has been adopted by several major computer companies that produce parallel computer systems. These systems include iPSC-2 and iPSC-860 by Intel, NCUBE/10 by nCUBE, and Symult 2010 by Ametek. In this routing model, the routing algorithm must follow for every source–destination pair a fixed, single path that is determined at manufacturing time according to some routing rule. We refer to this routing model as the *fixed-routing model* for short and to the rule whereby the single path is determined as the *fixed-routing rule*. The focus of this paper is the important problem of permutation routing on networks under the fixed-routing model.

When routing a permutation f [where every node i needs to send a single message to node $f(i)$], path conflicts often occur and cause communication overhead. There are three sources of communication overhead, namely, link conflict, node conflict, and path length. It was shown by Bokhari [2] that the impact of node conflict and path length is negligible in circuit-switched fixed-routing systems, while the impact of link contention is the most dominant. Therefore, this paper will assume throughout that the communication overhead is due to link contention only.

To minimize the communication overhead when routing a permutation, the permutation has to be scheduled. *Scheduling* a permutation consists of partitioning the set of nodes into m subsets $E_1, E_2, \ldots, E_m$, for some m, such that for every $i = 1, 2, \ldots, m$, the paths originating from the nodes in E_i do not conflict over links, i.e., they can be established simultaneously and their corresponding messages can be delivered in parallel. E_i represents then the set of source nodes that can send data to their destinations at time i, $i = 1, 2, \ldots, m$. *Optimal scheduling* is the process of finding a partition of minimum size m, and the partition is called an optimal schedule.

It should be pointed out that most of the research

NETWORKS, Vol. 23 (1993) 441–448

CCC 0028-3045/93/040441-08

efforts on permutation scheduling in graph networks have assumed packet switching [6, 15, 18, 19]. Permutation scheduling on circuit-switched fixed-routing networks is fairly new. This author addressed online self-routing of bit-permute-complement permutations on fixed-routing meshes [20], but for arbitrary permutations and for other networks, no effort has been made.

This paper will study off-line permutation scheduling on graph networks under the fixed-routing model, and give efficient scheduling algorithms for linear arrays, hypercubes, meshes, and tori. In the case of linear arrays, the algorithms are based on node coloring of certain intersection graphs. In hypercubes, the routing algorithm is based on routing in the well-known universal Benes multistage interconnection network [1]. Finally, scheduling on meshes and tori will be accomplished by using bipartite perfect matching in certain model graphs.

The paper is organized as follows: The next section will cover some fundamentals and overview relevant concepts. Section 3 will develop a node coloring formulation of permutation scheduling under the fixed-routing model. The following four sections will address the complexity of optimal permutation scheduling on the various networks and give optimal or suboptimal algorithms for each network.

2. PRELIMINARIES

This section will specify the fixed-routing rules on the networks under consideration. It will also overview intersection graphs and their node coloring complexity.

2.1. The Fixed-routing Rules

In linear arrays, there is only one path between any pair of nodes. Therefore, there can only be one fixed-routing rule. In the case of rings, we will consider the *clockwise rule* that selects the clockwise path. This forces the ring to be a directed graph. The *counter-clockwise rule* can be treated similarly. In meshes/tori, the row-column rule will be considered. In this rule, the source-destination path goes rowwise to the correct column and then columnwise to the destination.

In the k-cube, call the bits that differ in two nodes x and y of binary labels $x_{k-1} \ldots x_1x_0$ and $y_{k-1} \ldots y_1y_0$ the *changeable bits*. It is evident that a shortest path from x to y can be generated from x one node at a time by complementing every changeable bit exactly once in some order. We will define two fixed-routing rules: The *e-cube rule* selects the path corresponding to complementing the changeable bits from right to left. The *e^{-1}-cube rule* selects the path corresponding to complementing the changeable bits from left to right. The *e*-cube rule is implemented by Intel's iPSC2.

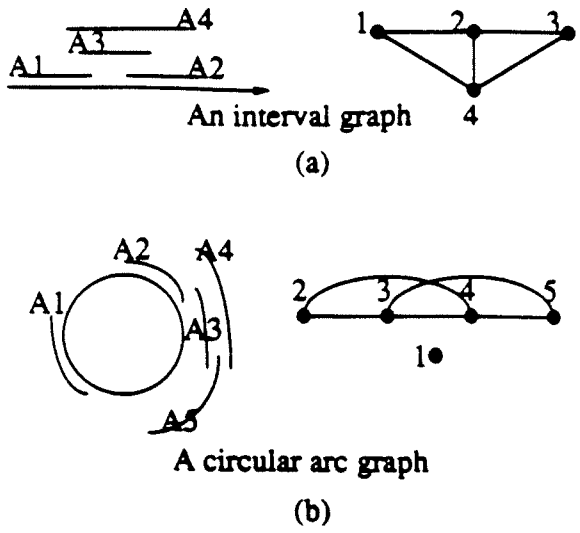

Fig. 1. Intersection graphs.

2.2. Intersection Graphs

Let $A_1, A_2, \ldots, A_n$ be n sets. The *intersection graph* modeled by these sets is the graph $G = (V,E)$ where $V = \{1,2, \ldots, n\}$ and $E = \{(i,j) \mid A_i \cap A_j \neq \emptyset\}$. If the sets are intervals on the real axis, the intersection graph is an *interval graph* [7, 8]. If the sets are arcs of a circle, the graph is a *circular arc graph* [4]. See Figure 1 for examples of intersection graphs.

Graph-theoretical problems related to these subclasses of graphs have received intense attention (see Golumbic [5] and the references therein for a thorough treatment of these graphs). The problem relevant to this paper is node coloring because optimal permutation scheduling on linear arrays and rings turn out to be equivalent to node coloring of interval graphs and circular arc graphs, respectively, as will be shown later. (A node coloring of a graph is the assignment of colors to nodes such that adjacent nodes have distinct colors.)

3. NODE COLORING FORMULATION

Let $G = (V,E)$ be a network with a fixed-routing rule s, and f a permutation of V to be scheduled on G. Our node-coloring formulation aims at constructing a graph $\Gamma_f(G)$ such that every schedule of f in G corresponds to a node coloring of $\Gamma_f(G)$ and vice versa. In particular, the chromatic number of $\Gamma_f(G)$ is equal to the minimum number of passes needed to route f in G, i.e., the size of the optimal schedule of f.

Let $\Gamma_f(G)$ be the intersection graph modeled by the

sets $A_i = \{e \mid e \text{ is an edge in the path } i \to f(i)\}$, for all $i \in V$, i.e., $\Gamma_f(G) = (V', E')$, where $V' = V$ and $(i,j) \in E'$ if and only if the s-determined paths $i \to f(i)$ and $j \to f(j)$ overlap (i.e., conflict) over at least one link in G.

Theorem 1. *(a) A partition $E_1, E_2, \ldots, E_m$ of V is a schedule of f in G if and only if the m-coloring of $\Gamma_f(G)$ derived by coloring the nodes in E_j with color j for $j = 1,2, \ldots, m$ is a correct node coloring.*

(b) The chromatic number of $\Gamma_f(G)$ is equal to the size of the optimal schedule of f.

Proof. (a) Assume that $E_1, E_2, \ldots, E_m$ is a schedule of f in G. Then, by definition, for every $j = 1,2, \ldots, m$, the paths $(i \to f(i))_{i \in E_j}$ are mutually nonoverlapping (i.e., nonconflicting), and, therefore, the corresponding nodes $i \in E_j$ are mutually nonadjacent in $\Gamma_f(G)$. Thus, all the nodes $i \in E_j$ can be colored with the same color, say color j. Conversely, if for every j the nodes in E_j can be colored with the same color in $\Gamma_f(G)$, then their corresponding paths are nonconflicting in G, and, therefore, $(E_j)_{1 \le j \le m}$ is a schedule of f in G.

(b) This immediately follows from part (a). ∎

Therefore, the complexity of scheduling a permutation f in a fixed-routing network G is the same as node coloring of the intersection graph $\Gamma_f(G)$.

4. PERMUTATION SCHEDULING ON LINEAR ARRAYS

We have to distinguish two cases: The first is when the edges are half-duplex, i.e., data can flow over a link in either direction but not simultaneously. The second case is when the links are full duplex, i.e., data can flow over a link in both directions simultaneously. We will treat the first case and then derive the second case as a corollary.

Observe that if G is the linear array L_N the graph $\Gamma_f(L_N)$ is an interval graph. This is so because two paths $i \to f(i)$ and $j \to f(j)$ conflict over links in L_N if and only if the real intervals $[i,f(i)]$ and $[j,f(j)]$ are overlapping. Note that an interval $[a,b]$ is the set $\{x \mid x \text{ is a real number and } a < x < b\}$. Note also that if $i > f(i)$, the interval $[i,f(i)]$ is simply taken to denote $[f(i),i]$.

Node coloring of interval graphs has been studied extensively as it has applications in VLSI channel routing. An optimal $\theta(N \log N)$ time algorithm to color interval graphs with a minimum number of colors has been found by Gupta et al. [7]. We will give a brief presentation of this algorithm and then conclude from it a refined $O(N)$ version for the case of permutation routing.

The algorithm in [7] is the following:

Procedure INTERVAL-COLOR

begin

1. Sort the end points of the intervals.
2. Let Q be a list of available colors, and m, an integer variable representing the maximum number of colors needed so far. Initialize Q to $\varnothing$ and m to 0.
3. Scan the sorted endpoints from left to right. When an endpoint x is reached do

if (x is a left end of an interval I) **then**
  if (Q is not empty) **then**
    delete a color c from Q and color I with c;
  else
    increment m and color I with color m;
else/* x is the right end of I*/
  insert the color of I into the list Q;
end

Observe that in the case $\Gamma_f(L_N)$ the $2N$ endpoints of the intervals are the integers $0,1, \ldots, N-1$, where every i occurs twice: in $[i,f(i)]$ and $[f^{-1}(i),i]$. Therefore, these endpoints are naturally sorted. This eliminates Step 1 above and reduces the complexity of the algorithm to $O(N)$, which is optimal. Accordingly, we present below an $O(N)$ refined code for permutation scheduling on linear arrays:

procedure LIN-SCHEDULE(*f*: in; *c*[0. .*N*3 −1]: out)
/*f a permutation to schedule, $c[i]$ = color of i */

```
begin
  integer m := 0;
  queue Q := ∅; /*m and Q are as before*/
  integer i; /*node index*/
  for i = 0 to N − 1 do
    case
      i = f(i):
        c[i] := 1;
      i < f(i) ∧ i < f^-1(i):
        c[i] :=COLOR(Q,m);
        c[f^-1(i)] :=COLOR(Q,m);
      f(i) < i < f^-1(i):
        c[f^-1(i)] := c[i];
        /*This was a combined step*/
        /*equivalent to adding the*/
        /*color of [f(i),i] to Q*/
        /*and giving a color from*/
        /*Q to [i,f^-1(i)]*/
      f^-1(i) < i < f(i):
        /*A similar combined step*/
        c[i] := c[f^-1(i)];
```

```
        i > f(i) ∧ i > f^{-1}(i):
          enqueue(c[i]);
          enqueue(c[f^{-1}(i)]);
      endcase
    endfor
end procedure
```

The function COLOR follows:

```
function COLOR(Q,m)
begin
  integer col;
  if Q ≠ ∅ then col =deque(Q); return(col);
  else m := m + 1; return(m);
end
```

Example. Consider the linear array of 7 nodes, and let f = [2 3 0 1 6 4 5] be a permutation to be routed [$f(0) = 2, f(1) = 3$ and so on]. We will assign a color to each of the seven nodes using LIN-SCHEDULE. After each color assignment, the value of m and the contents of Q will be shown. (Note that f^{-1} = [2 3 0 1 5 6 4].)

$0 < f(0) = 2 \wedge 0 < f^{-1}(0) = 2$
$c[0] := 1, m = 1, Q = \varnothing$
$c[f^{-1}(0)] = c[2] := 2, m = 2, Q = \varnothing$

$1 < f(1) = 3 \wedge 1 < f^{-1}(1) = 3$
$c[1] := 3, m = 3, Q = \varnothing$
$c[f^{-1}(1)] = c[3] := 4, m = 4, Q = \varnothing$

$2 > f(2) = 0 \wedge 2 > f^{-1}(2) = 0$
$Q = \{2\}$ (due to enqueue($c[2]$))
$Q = \{2,1\}$ (due to enqueue($c[f^{-1}(2)]$))
$m = 4$

$3 > f(3) = 1 \wedge 3 > f^{-1}(3) = 1$
$Q = \{2,1,4\}$ (due to enqueue($c[3]$))
$Q = \{2,1,4,3\}$ (due to enqueue($c[f^{-1}(3)]$))
$m = 4$

$4 < f(4) = 6 \wedge 4 < f^{-1}(4) = 5$
$c[4] := 2$ (deque(Q)), $Q = \{1, 4, 3\}$
$c[f^{-1}(4)] = c[5] := 1$ (deque(Q)), $Q = \{4, 3\}$
$m = 4$

$4 = f(5) < 5 < f^{-1}(5) = 6$
$c[f^{-1}(5)] = c[6] := c[5] = 1$
$m = 4, Q = \{4, 3\}$

$6 > f(6) = 5 \wedge 6 > f^{-1}(6) = 4$
$Q = \{4, 3, 1\}$ (due to enqueue($c[6]$))
$Q = \{4, 3, 1, 2\}$ (due to enqueue($c[f^{-1}(6)]$))
$m = 4$

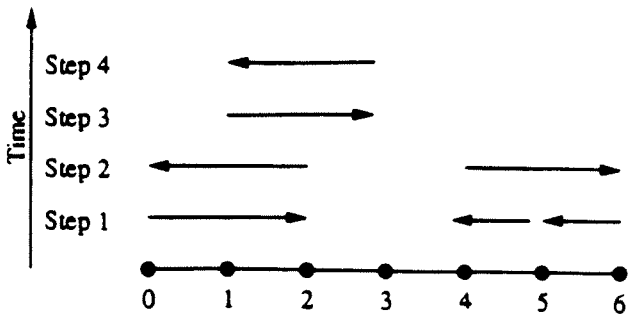

Fig. 2. Routing of f on the linear array L_7.

Therefore, routing f takes 4 time steps (see Fig. 2):

STEP 1. $0 \rightarrow 2, 5 \rightarrow 4, 6 \rightarrow 5$

STEP 2. $2 \rightarrow 0, 4 \rightarrow 6$

STEP 3. $1 \rightarrow 3$

STEP 4. $3 \rightarrow 1$.

The time complexity of LIN-SCHEDULE is clearly $O(N)$, which is optimal because the size of the input f is $O(N)$. The optimality of the resulting schedule is a direct result of the optimality of the number of colors of the algorithm by Gupta et al. [7]. Note that in the case where the links are full duplex the scheduling can be done in two phases: In the first phase, the nodes i whose destinations are to their right [i.e., $i < f(i)$] are colored with colors 1,2, ... In the second phase, the nodes i whose destinations are to their left [i.e., $f(i) < i$] are colored with colors 1,2, ... In both phases, the same procedure LIN-SCHEDULE can be used. The optimality of the resulting schedule is a consequence of the optimality of the half-duplex case.

5. PERMUTATION SCHEDULING ON RINGS

It will be shown that permutation scheduling on rings in NP-complete by reducing the problem of node coloring of circular arc graphs to the problem of scheduling of partial permutations on the clockwise ring.

Consider an instance I of node coloring in a circular arc graph. Let (x_1,x_2), (x_2,x_3), . . . ,(x_{2n-1},x_{2n}) be n clockwise arcs. Assume without loss of generality that no endpoint is shared by more than one arc. View the endpoints as nodes on the circle periphery. Scan the endpoints clockwise, starting with x_1, and relabel them by 0,1, . . . ,$2n - 1$ in the order they are scanned (see Fig. 3 for an example). View these nodes as a ring R_N of N nodes where $N = 2n$ and the edges are $\{(i,i + 1 \mod N) \mid i = 0,1, \ldots, N - 1\}$, and let f be the following partial permutation: For every arc (x_{2i},x_{2i+1}), if j is

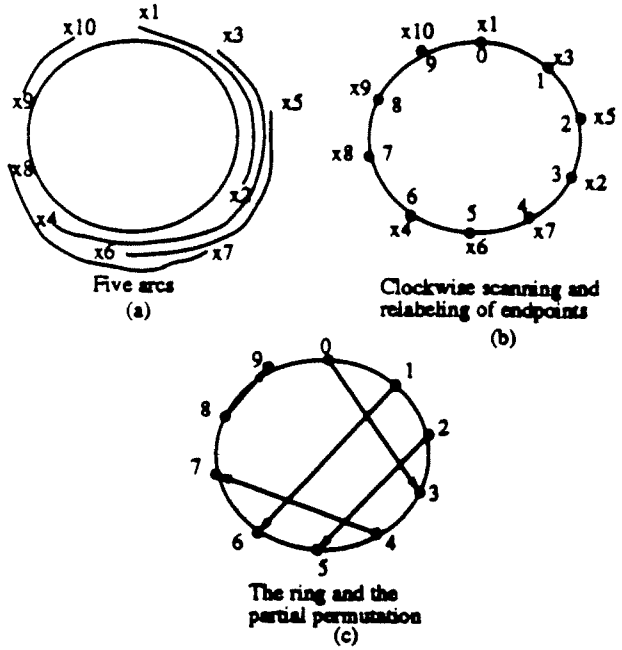

Fig. 3. Arcs-to-ring transformation.

the new label of x_{2i}, then $f(j)$ is the new label of x_{2i+1}. In other terms, the startpoint of each arc needs to send a message to the endpoint of that arc.

It can be easily seen that $\Gamma_f(R_N)$ is the same arc graph I. In particular, every schedule of f corresponds to an arc coloring of the circular arc graph and vice versa. Thus, node coloring reduces to scheduling of partial permutations on rings. In turn, scheduling of partial permutation on rings can be shown to reduce to scheduling of full permutations on rings. Since node coloring of circular arc graphs is NP-complete [4], it follows that permutation scheduling on fixed routing rings is NP-complete.

6. PERMUTATION SCHEDULING ON HYPERCUBES

In this section, the permutations that are routable in one pass (i.e., $m = 1$) under one of the rules, e^{-1}-cube and e-cube, will be characterized assuming that the links are full duplex. Using this characterization, it will be shown that the very useful Ω-realizable (resp., Ω^{-1}-realizable) permutations [10] are routable in one pass on the hypercube under the e^{-1}-cube (resp., e-cube) rule. Afterward, the Benes routing algorithm will be used to schedule arbitrary permutations in 2 passes on the hypercube.

Throughout this section, the hypercube under consideration is a k-cube of $N = 2^k$ nodes and the permutations are permutations of $\{0,1, \ldots, N-1\}$. We will also follow the notation that every node x has the binary label $x_{k-1} \ldots x_1 x_0$.

The following lemmas will lead to the desired characterization.

Lemma 1. *Let $s = s_{k-1} \ldots s_1 s_0$ and $d = d_{k-1} \ldots d_1 d_0$ be two nodes in the k-cube under the e^{-1}-cube rule. The path $s \rightarrow d$ goes through an edge (x,y) in the r-th dimension if and only if $x = d_{k-1} \ldots d_{r+1} s_r \ldots s_0$, $y = d_{k-1} \ldots d_r s_{r-1} \ldots s_0$ and $d_r = \overline{s_r}$.*

Proof. Assume that the path $s \rightarrow d$ goes through an edge (x,y) in the r-th dimension. Under the e^{-1}-cube rule, the path from s to d is found by complementing the changeable bits of s from left to right, where the changeable bits are the ones that differ in s and d. Therefore, when the path reaches x to cross the r-th dimension, all the bit positions $k-1, k-2, \ldots, r+1$ have been processed and the changeable bits among them complemented. Thus, the bits $k-1, k-2, \ldots, r+1$ agree in d and x. The remaining bits in x are still the same bits in s. This shows that $x = d_{k-1} \ldots d_{r+1} s_r \ldots s_0$. Since y is reached in the immediate next step by crossing the r-th dimension, it follows by the same line of argument that $y = d_{k-1} \ldots d_r s_{r-1} \ldots s_0$. The necessity of going through the r-th dimension can only be when $d_r = \overline{s_r}$.

The converse is easily established by following the above argument backward. ■

Lemma 2. *Two paths $s \rightarrow d$ and $s' \rightarrow d'$ conflict over a link in the k-cube under the e^{-1}-cube rule if and only if there exists an integer r, $0 \le r \le k-1$, such that $d_{k-1} \ldots d_r s_{r-1} \ldots s_0 = d'_{k-1} \ldots d'_r s'_{r-1} \ldots s'_0$ and $d_r = d'_r = \overline{s_r} = \overline{s'_r}$.*

Proof. Assume that the paths $s \rightarrow d$ and $s' \rightarrow d'$ conflict over a link. Let (x,y) be that link and assume it is in the r-th dimension for some r, $0 \le r \le k-1$. Using the previous lemma, it is concluded that $x = d_{k-1} \ldots d_{r+1} s_r \ldots s_0$, $x = d'_{k-1} \ldots d'_{r+1} s'_r \ldots s'_0$, $y = d_{k-1} \ldots d_r s_{r-1} \ldots s_0$, $y = d'_{k-1} \ldots d'_r s'_{r-1} \ldots s'_0$, $d_r = \overline{s_r}$ and $d'_r = \overline{s'_r}$. The last four equalities yield that $d_{k-1} \ldots d_r s_{r-1} \ldots s_0 = d'_{k-1} \ldots d'_r s'_{r-1} \ldots s'_0$ and $d_r = d'_r = \overline{s_r} = \overline{s'_r}$.

The converse can be established similarly using the previous lemma. ■

In the well-known paper by Lawrie [10], conflicting paths in Ω and Ω^{-1} networks were characterized. The following lemma is a brief restatement of the characterization of conflicting paths in Ω networks. The reader is referred to [10] for the proof.

Lemma 3. *Two paths $s \rightarrow d$ and $s' \rightarrow d'$ conflict over a link in Ω of 2^k inputs if and only if there exists an*

integer r, $0 \le r \le k-1$, *such that* $d_{k-1} \ldots d_r s_{r-1} \ldots s_0 = d'_{k-1} \ldots d'_r s'_{r-1} \ldots s'_0$.

We thus have this interesting theorem:

Theorem 2. *(a) Every permutation realizable by the* Ω *network without conflict is routable in one pass on the* k*-cube under the* e^{-1}*-cube rule.*

(b) Every permutation realizable by the Ω^{-1} *network without conflict is routable in one pass on the* k*-cube under the e-cube rule.*

Proof. Part (a) follows from the preceding lemmas and discussion. Part (b) follows from part (a) and the fact that the e-cube rule is the inverse of the e^{-1}-cube rule and Ω^{-1} is the inverse of Ω. ■

Since many interesting and frequently used permutations are realized by Ω and Ω^{-1}, it follows that many interesting permutations are routable in one pass on the fixed-routing hypercubes such as iPSC-2 which uses the e-cube rule.

As for arbitrary permutations, it was shown in [14] that optimal permutation scheduling on hypercubes is NP-complete. However, because of the importance of hypercubes and permutation routing, the problem cannot be left there. We will follow another approach based on Benes routing and the fact that every Benes network is identical to the network derived from concatenating an Ω^{-1} network with an Ω network, i.e., Benes $\equiv \Omega^{-1}\Omega$. We will next outline this approach that yields a 2-pass scheduler:

1. Use Lee's algorithm [11] which, for any given permutation f, finds the switch settings of $\Omega^{-1}\Omega$ to realize f.
2. Let g be the permutation realized by the switch settings of the Ω^{-1} part, and let h be the permutation realized by the switch settings of the Ω part. Clearly, $f = g \circ h$.
3. Route g in one pass on the hypercube under the e-cube rule, and then route h in one pass on the hypercube under the e^{-1}-cube rule. This is doable after Theorem 2.

Lee's algorithm takes $O(N \log N)$ sequential time [11], but $O(N)$ parallel time. The parallel algorithm in [11] can be shown to run on the k-cube with the same complexity $O(N)$. Thus, our hypercube scheduler takes $O(N \log N)$ sequential time and $O(N)$ parallel time on the k-cube. The parallel complexity can be further reduced to $O(\log N)$ parallel time on the hypercube for the interesting classes of permutations *bit-permute complement* (BPC) and *linear-complement* (LC) using the Benes routing algorithms in [16] and [3], respectively. The details of these Benes routing algorithms can be found in their respective references.

We thus have a fast cube scheduling algorithm that allows every permutation to run in two passes if the routing model allows for the two routing rules, namely, the e-cube rule and the e^{-1}-cube rule.

7. PERMUTATION SCHEDULING ON MESHES/TORI

Using bipartite matching, we will show that every permutation can be routed on $p \times q$ meshes/tori ($p \le q$) in q passes under the row-column routing rule. The following standard theorem about perfect matchings in bipartite graphs will be of prime use. The proof of this theorem can be found in [13].

Theorem 3. *Let* $G = (U,V,E)$ *be a bipartite graph such that for every subset* A *of* U*, we have* $|\Gamma(A)| \ge |A|$*, where* $\Gamma(A)$ *is the subset of nodes in* V *that are adjacent to nodes in* A*. Then,* G *has a perfect matching, that is, a matching of size* $= min(|U|,|V|)$.

Assume that the node in row x_1 and column x_2 is labeled x_1x_2, for all $x_1 = 0,1, \ldots, p-1$ and $x_2 = 0, 1, \ldots, q-1$. The main idea of scheduling is to select, in every pass, p paths $x_1x_2 \rightarrow f(x_1x_2)$ such that their sources belong to distinct rows and their destinations belong to distinct columns. Such paths do not conflict. This will be accomplished using perfect matching in a bipartite graph $G = (V_1,V_2,E)$ to be defined next. V_1 is the set of rows and V_2 is the set of columns. For every pair of nodes (x_1,y_2) in $V_1 \times V_2$, (x_1,y_2) is an edge in E of label y_1x_2 if $f(x_1x_2) = y_1y_2$. Thus, G is a bipartite multigraph. The following lemma will help us show that G has a perfect matching of size p.

Lemma 4. *Let* $G = (V_1,V_2,E)$ *be as just defined. For every subset* A *of* V_1*, we have* $|\Gamma(A)| \ge |A|$.

Proof. Let A be an arbitrary subset of V_1. For every node $x_1 \in V_1$, the number of edges incident to x_1 is q because row x_1 has q source nodes. Thus, the number of edges incident to nodes in A is $q|A|$. Similarly, for every node y_2 in V_2, the number of edges incident to y_2 is p because column y_2 has p destination nodes. Thus, the number of edges incident to nodes in $\Gamma(A)$ is $p|\Gamma(A)|$. Since all the edges incident to nodes in A are incident to nodes in $\Gamma(A)$, it follows that $p|\Gamma(A)| \ge q|A|$. This yields that $|\Gamma(A)| \ge q/p|A| \ge |A|$ because $q/p \ge 1$. ■

Using the previous lemma and Theorem 3, it is concluded that $G = (V_1,V_2,E)$ has a perfect matching M

of size equal to $min(|V_1|,|V_2|) = min(p,q) = p$. Every edge (x_2,y_2) of some label y_1x_2 in M corresponds to the path $x_1x_2 \rightarrow f(x_1x_2)$ that is $x_1x_2 \rightarrow y_1y_2$ consisting of row-path $x_1x_2 \rightarrow x_1y_2$ in row x_1, followed by the column-path $x_1y_2 \rightarrow y_1y_2$ in the column y_2. The p paths corresponding to the p edges of the matching M are mutually nonconflicting because every two such paths have their sources in two distinct rows and their destinations in two distinct columns due to the fact that M is a matching. Thus, all these paths can be established simultaneously in one pass.

By deleting M from G, we obtain a new bipartite graph that can be similarly shown to have a perfect matching. By repeating this process q times, we obtain q perfect matchings yielding a q-pass schedule for f under the row-column routing rule. We thus have a q-pass scheduling algorithm for meshes/tori.

Time complexity: The construction of $G = (V_1,V_2,E)$ takes $O(N)$ time. The bipartite matching algorithm takes $O(min(|V_1|,|V_2|)|E| = O(ppq)$ [17]. deleting M from G takes $O(p)$ time. Thus, the q iterations take $O(p^2q^2) = O(N^2)$ time, where N is the number of processors. Hence, the whole algorithm takes $O(N^2)$ time. This complexity can be brought down to $O(N \log^2 N)$ time by reducing perfect matching to permutation routing on Clos networks and using the Clos routing algorithm in [12].

8. CONCLUSIONS

This paper addressed off-line permutation scheduling on networks under the circuit-switched fixed routing model. Using node coloring of intersection graphs, it was shown that permutation scheduling on linear arrays takes linear time, and on rings it is NP-complete. The paper also developed a Benes-based algorithm that routes any permutation on the hypercube in two time steps when two routing rules are allowed. The complexity of the algorithm is the same as the Benes routing complexity, which is $O(N \log N)$ sequential time and $O(N)$ hypercube-parallel time. This time cost is very affordable for off-line scheduling. In addition, it was established that every Ω^{-1}-realizable permutation can be routed on the hypercube in one time step under the standard e-cube routing rule and that the routing need no scheduling. Finally, an $O(N \log^2 N)$ time algorithm to schedule arbitrary permutations on $p \times q$ meshes in q time steps was designed.

One conclusion that can be drawn from the paper is that the hypercube is the most efficient circuit-switched fixed-routing network among the representative standard networks considered in this paper. To allow for the highest efficiency and use of hypercubes, it is recommended that two fixed routing rules be implemented, namely, the e-cube rule and the e^{-1}-cube rule.

Several related topics deserve further investigation. First, the still-open inherent complexity of optimal permutation scheduling on meshes and tori need to be addressed. Second, other modes of routing on various networks under the fixed-routing model should be examined. Third, since the algorithms presented here are too costly for on-line permuting, faster permuting algorithms are needed even if they yield suboptimal schedules. One possible approach is *ad hoc* permuting that requires no scheduling. Rather, every node tries to establish the path to its destination and keeps trying until it succeeds. Our preliminary probabilistic analysis of ad hoc permuting indicates that the routing delay is surprisingly close to the optimal delay. Further examination of this approach and its performance is currently being undertaken.

REFERENCES

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

[2] S. H. Bokhari, Communication overheads on the Intel iPSC-2 hypercube. ICASE Interim Report 10 (May 1990).

[3] R. Boppana and C. S. Raghavendra, On self routing in Benes and shuffle exchange networks. *Proc. Int'l Conf. Par. Proc.* (Aug. 1988) 196–200.

[4] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, The complexity of coloring circular arcs and chords. *SIAM J. Alge. Discrete Methods* **1** (1980) 216–227.

[5] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York (1980).

[6] A. Gottlieb and C. P. Kruskal, Complexity results for permuting data and other computations on Parallel Processors. *J. ACM* **31**(2) (1984) 193–209.

[7] A. I. Gupta, D. T. Lee, and J. Y.-T. Leung, An optimal solution for the channel-assignment problem. *IEEE Trans. Comput.* **C-28**(11) (1979) 807–810.

[8] A. I. Gupta, D. T. Lee, and J. Y.-T. Leung, Efficient algorithms for interval graphs and circular-arc graphs. *Networks* **12** (1982) 459–467.

[9] F. Harari, On the group of the composition of two graphs. *Duke Math. J.* **26** (1959) 29–34.

[10] D. K. Lawrie, Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24** (1975) 1145–155.

[11] K. Y. Lee, A new Benes network control algorithm. *IEEE Trans. Comput* **C-36** (1987) 768–772.

[12] G. F. Lev, N. Pippenger, and L. G. Valiant, A fast parallel algorithm in permutation network. *IEEE Trans. Comput.* **C-30** (1981) 93–100.

[13] C. L. Liu, *Introduction to Combinatorial Mathematics*. Computer Science Series, McGraw-Hill, New York (1968).

[14] L. Liu, H.-A. Choi, and S. Rotenstreich, Simultaneous task migration on circuit-switched hypercube multiprocessors. Technical Report GWU-IIST-91-21, George Washington University (Sept. 1991).

[15] D. Nassimi and S. Sahni, An optimal routing algorithm for mesh-connected parallel computers. *J. ACM* **27**(1) (1980) 6–29.

[16] D. Nassimi and S. Sahni, A self-routing Benes network and parallel permutation algorithms. *IEEE Trans. Comput.* **C-30** (1981) 332–340.

[17] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, Englewood, NJ (1982).

[18] C. S. Raghavendra and V. K. Prasanna Kumar, Permutations on Illiac IV-type networks. *IEEE Trans. Comput.* **C-35**(7) (1986) 662–669.

[19] L. G. Valiant, A scheme for fast parallel communication. *SIAM J. Comput.* **11**(2) (1982) 350–361.

[20] A. Youssef, Online communication on circuit-switched fixed routing meshes. *Proceedings of the Int'l Parallel Processing Symposium*, California (March 1992).