On-line Communication on Circuit-Switched Fixed Routing Meshes

Abdou Youssef
Department of EECS
The George Washington University
Washington, D.C. 20052
youssef@gwusun.gwu.edu

Abstract. This paper addresses routing on the circuit-switched fixed routing $n \times n$ mesh M_n where paths follow the row-column rule. The paper presents first a self-routing algorithm that schedules any BPC permutation in $O(\log N)$ time ($N = n^2$) to run in n routing steps on M_n, and another self-routing algorithm that schedules in constant time any Ω- or Ω^1-realizable permutation to run in $2n$ routing steps on M_n. Finally, the paper shows that broadcasting and fan-in communication self-route in $\log N$ routing steps, that parallel prefix requires $\log^2 N$ routing steps, and that FFT requires $O(n)$ routing steps. In all but the FFT case, the circuit-switched mesh delivers better performance than its packet-switched counterpart.

Index Terms: Circuit Switching, Distributed Routing, Fixed Routing Rule, Mesh.

1 Introduction

The circuit-switched fixed routing model is increasingly becoming the dominant communication model for parallel computer systems. In this model, even if the network topology provides more than one path between pairs of nodes, the routing algorithm must follow a fixed path for every pair of nodes. The path is predetermined at manufacturing time according to some routing rule. We will refer to this routing model as the *fixed routing model*, and to the rule whereby the fixed path is selected as the *fixed routing rule*. An example of fixed routing rules is the mesh row-column rule which forces the path to first go row-wise to the column of the destination, and then go column-wise to the desired destination.

When routing in the fixed routing model, path conflicts often occur and cause communication overhead. There are three sources of communication overhead, namely, link conflict, node conflict, and path length. Bokhari has shown through extensive experimentation on hypercubes that the impact of node conflict and path length is negligeable in circuit-switched fixed routing systems, while the impact of link contention is the most dominant [1]. Therefore, this paper will assume throughout that communication overhead is due only to link contention.

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

The focus of this paper is efficient scheduling of permutations and other important communication patterns on the row-column fixed routing mesh.

Most of the research efforts on permutation scheduling in graph networks have assumed packet switching [2], [4], [6], [9]. Permutation scheduling on circuit-switched fixed routing networks is new.

This paper will develop three efficient self-routing algorithms for each of the three important classes of bit-permute-complement (BPC) permutations [5], Ω-realizable permutations, and Ω^{-1}-realizable permutations [3]. The mesh is $n \times n$, where n is a power of 2. The BPC self-routing algorithm determines a schedule of size n for each BPC permutation. The structure of the algorithm can be summarized as follows. When routing a BPC permutation, every node determines in $O(\log N)$ time the time step at which it is supposed to send its data to its destination. Afterwards, at every time step i, $i = 1, ..., n$, each node whose precomputed time step is equal to i sends its data. Thus, the algorithm spends $O(\log N)$ time to compute the schedule and n time steps to do the actual routing.

The two algorithms for the Ω and Ω^{-1} permutations route each permutation in two phases, where each phase requires n time steps. A small constant time is spent by the nodes to figure out the time step at which they should send their data. Thus, these two algorithms spend $O(1)$ time to schedule and 2n time steps to do the actual routing.

In addition to BPC and (Ω, Ω^{-1})-realizable permutations, this paper will address the mesh routing of common communication patterns: fan-out (i.e., broadcasting), fan-in (required by semi-group computations), and other patterns arising in divide-and-conquer algorithms such as parallel prefix and FFT. It will be shown that the fan-out and fan-in patterns can be done in optimal $\log N$ routing steps, that parallel prefix runs in $O(\log^2 N)$ time, and that FFT runs in $O(n)$ time.

This paper is organized as follows. The next section discusses some preliminary concepts. In Section 3 the self-routing algorithm for BPC permutations is developed. Section 4 presents the two self-routing algorithms for the Ω permutations and the Ω^{-1} permutations. Section 5 treats the mesh routing of the communication patterns that arise in certain parallel computations. Concluding remarks are presented in Section 6.

2 Preliminaries

Throughut the paper let $n = 2^k$ and $N = n^2$. Let also M_n denote the $n \times n$ row-column fixed routing mesh. The nodes of M_n are labelled row-wise 0 through $N-1$. Every node x is expressed in binary as $x_{2k-1}...x_1x_0$. Clearly, $x_{2k-1}...x_{k+1}x_k$ is the binary label of the row of x, and $x_{k-1}...x_1x_0$ is the binary label of the column of x.

Let π be a permutation of $\{0,1,...,2k-1\}$ and $a = a_{2k-1}...a_1a_0$ be a binary number. A BPC permutation $f_{\pi,a}$ [5] is a permutation of $\{0,1,...,N-1\}$ characterized by π and a such that $f_{\pi,a}(x_{2k-1}...x_1x_0) = (x_{\pi(2k-1)}...x_{\pi(1)}x_{\pi(0)}) \oplus a$, where $\oplus$ is the bit-wise exclusive or.

For every binary string $x_{m-1}...x_1x_0$ and every subset $A \subseteq \{0,1,...,m-1\}$, we denote by $[x_{m-1}...x_1x_0]_A$ the string derived by selecting from $x_{m-1}...x_1x_0$ the bits whose positions are elements of A. The relative order of the selected bits is the same as in the original string. For example, if $m = 8$ and $A = \{1,4,3,7\}$, then $[x_7...x_1x_0]_A = x_7x_4x_3x_1$. A useful alternative notation for $[x_{m-1}...x_1x_0]_A$ is $(x_i)_{i\in A}$ which does not require the bits x_j, $j \notin A$, to be defined.

3 Self-Routing of BPC

Let $f_{\pi,a}$ be a BPC permutation to be routed on M_n. Our permutation scheduling algorithm will find a schedule of n time steps such that at every time step exactly one node from each row will send and exactly one node in each column will receive. This one-source-per-row one-destination-per-column (1R-1C) criterion eliminates link conflict because link conflict between any two source-destination paths might occur only if the two sources are in the same row or the two destinations are in the same column.

The problem of routing $f_{\pi,a}$ is then to identify for each time step t one source from each row such that the (1R-1C) criterion holds. To achieve self-routedness, each node x should itself compute the time step $t(x)$ $(0 \le t(x) \le n-1)$ at which x sends to $f_{\pi,a}(x)$. The distributed method by which node x computes $t(x)$ should satisfy the (1R-1C) criterion.

To be able to compute $t(x)$ in a distributed fashion, an algebraic formula for $t(x)$ will be derived and certain fundamental properties will be identified and addressed. To that effect, let

$$\begin{aligned}
F &= \{\pi(i) \mid 0 \le i \le k-1 \ \&\ k \le \pi(i) \le 2k-1\} \\
F' &= \{\pi(i) \mid k \le i \le 2k-1 \ \&\ k \le \pi(i) \le 2k-1\} \\
F'' &= \{\pi(i) \mid k \le i \le 2k-1 \ \&\ 0 \le \pi(i) \le k-1\} \\
G &= \{i \mid 0 \le i \le k-1 \ \&\ k \le \pi(i) \le 2k-1\} \\
G' &= \{i \mid 0 \le i \le k-1 \ \&\ 0 \le \pi(i) \le k-1\}.
\end{aligned}$$

Clearly, $F \cup F' = \{k, k+1, ..., 2k-1\}$, $G \cup G' = \{0,1,...,k-1\}$, $F' \cup F'' = \{\pi(k), \pi(k+1), ..., \pi(2k-1)\}$, and $\pi(G) = F$, where $\pi(G) = \{\pi(i) \mid i \in G\}$. In particular, $|F| + |F'| = |G| + |G'| = |F''| + |F'| = k$. Also $|G| = |\pi(G)| = |F|$, concluding that $|G'| = |F'|$ and $|F| = |F''|$.

Let $(x_j)_{j\in F}$ and $(d_i)_{i\in G}$ be two strings. All the rows of labels $y_{2k-1}...y_{k+1}y_k$ such that $[y_{2k-1}...y_{k+1}y_k]_F = (x_j)_{j\in F}$ are said to form a *row group* denoted $RG((x_j)_{j\in F})$. In other terms, the labels of the rows in a row group agree in all the bit positions determined by the set F. Each row group has $2^{k-|F|}$ rows. Similarly, all the columns of labels $z_{k-1}...z_1z_0$ such that $[z_{k-1}...z_1z_0]_G = (d_i)_{i\in G}$ are said to form a *column group* denoted $CG((d_i)_{i\in G})$. This column group has $2^{k-|G|} = 2^{k-|F|}$ columns. There are $2^{|F|}$ disjoint row groups and $2^{|F|}$ disjoint column groups in M_n.

The following theorem proves that when routing $f_{\pi,a}$, there corresponds to each row group a column group such that the flow of data will be from each row group to its corresponding column group.

Theorem 1 *Assume that $f_{\pi,a}$ is to be routed on M_n. Let $(x_j)_{j\in F}$ be an arbitrary string and $(d_i)_{i\in G}$ be another string such that for all $i \in G$, $d_i = x_{\pi(i)} \oplus a_i$. Then the destination of every node of the row group $RG((x_j)_{j\in F})$ is a node of the column group $CG((d_i)_{i\in G})$. More precisely, each row of $RG((x_j)_{j\in F})$ has $2^{|F|}$ destinations in each of the $2^{k-|F|}$ columns of $CG((d_i)_{i\in G})$.*

Proof: Consider a row $y_{2k-1}\dots y_{k+1}y_k$ in $RG((x_j))_{j\in F}$, that is, $(y_j)_{j\in F} = (x_j)_{j\in F}$. Let $y_{2k-1}\dots y_k y_{k-1}\dots y_1 y_0$ be a node in that row. We need to show that its destination z is in the column group $CG((d_i)_{i\in G})$. Note that $z = z_{2k-1}\dots z_0 = f_{\pi,a}(y_{2k-1}\dots y_0) = (y_{\pi(2k-1)}\dots y_{\pi(0)}) \oplus a$.

For every $i \in G$ we have $\pi(i) \in F$ (because $\pi(G) = F$) and

$$\begin{aligned} z_i &= y_{\pi(i)} \oplus a_i \\ &= x_{\pi(i)} \oplus a_i \text{ (because for all } j \in F,\ y_j = x_j) \\ &= d_i \text{ (by definition of the string } (d_i)_{i\in G}) \end{aligned}$$

Consequently, the destination z is in the column group $CG((d_i)_{i\in G})$.

Consider next an arbitrary row $y_{2k-1}\dots y_{k+1}y_k$ in $RG((x_j)_{j\in F})$ and an arbitrary column $z_{k-1}\dots z_1 z_0$ in $CG((d_i)_{i\in G})$. The sources $y_{2k-1}\dots y_k y_{k-1}\dots y_0$ whose destinations are in column $z_{k-1}\dots z_1 z_0$ must satisfy the equality $(y_{\pi(k-1)}\dots y_{\pi(0)}) \oplus (a_{k-1}\dots a_0) = z_{k-1}\dots z_0$, which holds if and only if $y_{\pi(i)} \oplus a_i = z_i$ for all $i \in G'$ (because for all $i \in G$, z_i is already known to equal $y_{\pi(i)} \oplus a_i$). Therefore, the number of these sources is $2^{k-|G'|} = 2^{|G|} = 2^{|F|}$. Thus, each row of $RG((x_j)_{j\in F})$ has $2^{|F|}$ destinations in each of the $2^{k-|F|}$ columns of $CG((d_i)_{i\in G})$. □

Now we are in a position to give an alegraic formula that computes for every mesh node x the time step $t(x)$ at which node x should send its data to its destination. We will give the formula first then show that it satisfies the 1R-1C criterion.

For every mesh node x, $t(x)$ is defined to be:

$$t(x) = ([x]_{F'} \oplus [f_{\pi,a}(x)]_{G'}) \bullet [x]_{F''}$$

where $\bullet$ is the string concatenation operator. Note that $[f_{\pi,a}(x)]_{G'}$ is the string $(d_i)_{i\in G'}$ where $f_{\pi,a}(x) = d_{2k-1}\dots d_1 d_0$.

Theorem 2 *Let x and y be two distinct sources in the mesh M_n such that $t(x) = t(y)$. Then both x and y belong to two distinct rows and their destinations $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ belong to two distinct columns.*

Proof: If x and y belong to two distinct row groups, then their destinations belong to two distinct column groups. In particular, x and y belong to two distinct rows and their destinations belong to two distinct columns.

Assume then that x and y belong to the same row group. We will show that in this case $[x]_{F'} \neq [y]_{F'}$ and $[f_{\pi,a}(x)]_{G'} \neq [f_{\pi,a}(y)]_{G'}$. This will be done by showing that the opposite implies that $x = y$. To this effect, assume that

$$[x]_{F'} = [y]_{F'} \text{ or } [f_{\pi,a}(x)]_{G'} = [f_{\pi,a}(y)]_{G'}. \qquad (1)$$

Since $t(x) = t(y)$, it follows that $([x]_{F'} \oplus [f_{\pi,a}(x)]_{G'}) \bullet [x]_{F''} = ([y]_{F'} \oplus [f_{\pi,a}(y)]_{G'}) \bullet [y]_{F''}$, which yields that

$$[x]_{F'} \oplus [f_{\pi,a}(x)]_{G'} = [y]_{F'} \oplus [f_{\pi,a}(y)]_{G'} \qquad (2)$$

and

$$[x]_{F''} = [y]_{F''}. \qquad (3)$$

(1) and (2) imply that

$$[x]_{F'} = [y]_{F'} \qquad (4)$$

and

$$[f_{\pi,a}(x)]_{G'} = [f_{\pi,a}(y)]_{G'}. \qquad (5)$$

The fact that x and y are in the same row group implies that their destinations are in the same column group (after Theorem 1), that is,

$$[f_{\pi,a}(x)]_G = [f_{\pi,a}(y)]_G. \qquad (6)$$

(3) and (4) imply that $x_{\pi(i)} = y_{\pi(i)}$ for all $i = k, k+1, \dots, 2k-1$ because $F' \cup F'' = \{\pi(k), \pi(k+1), \dots, \pi(2k-1)\}$. Therefore, $x_{\pi(2k-1)}\dots x_{\pi(k+1)}x_{\pi(k)} = y_{\pi(2k-1)}\dots y_{\pi(k+1)}y_{\pi(k)}$ and hence

$$x_{\pi(2k-1)}\dots x_{\pi(k+1)}x_{\pi(k)} \oplus a_{2k-1}\dots a_{k+1}a_k = y_{\pi(2k-1)}\dots y_{\pi(k+1)}y_{\pi(k)} \oplus a_{2k-1}\dots a_{k+1}a_k.$$

Thus, the k most significant bits of both $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ are identical, implying that the nodes $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ are in the same row.

Also, (5) and (6) imply that $[f_{\pi,a}(x)]_{G\cup G'} = [f_{\pi,a}(y)]_{G\cup G'}$, that is, $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ are in the same column because $G \cup G' = \{0, 1, \dots, k-1\}$. We now have that both $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ are in the same row and the same column. Therefore,

$f_{\pi,a}(x) = f_{\pi,a}(y)$. This yields that $x = y$, which is the desired contradiction.

Consequently, $[x]_{F'} \neq [y]_{F'}$ and $[f_{\pi,a}(x)]_{G'} \neq [f_{\pi,a}(y)]_{G'}$. Since $F' \subseteq \{k, k+1, ..., 2k-1\}$, it follows that $x_{2k-1}...x_k \neq y_{2k-1}...y_k$, that is, x and y are in different rows. Similarly, since $G' \subseteq \{0, 1, ..., k-1\}$, it follows that $f_{\pi,a}(x)$ and $f_{\pi,a}(y)$ are in different columns. □

The previous theorem leads to this distributed permutation scheduling algorithm:

```
SELF-SCHEDULE-BPC($\pi, a$)
begin
/* every node is assumed to have $\pi$ and $a$ */
1. for  $x = 0$ to  $N - 1$ pardo/* scheduling */
        node $x$ computes $F'$, $G'$ and $F''$, and then
        $t(x) := ([x]_{F'} \oplus [f_{\pi,a}(x)]_{G'}) \bullet [x]_{F''}$;
   endfor
2. for $t = 0$ to $n - 1$ do   /* actual routing */
        forall node $x$ pardo
             if $t(x) = t$ then
                  node $x$ sends to node $f_{\pi,a}(x)$
                  at time step $t$;
             endif
        endforall
   endfor
end
```

Time complexity: Computing F', G', F'' and $t(x)$ takes $O(k) = O(\log N)$ time. Thus step (1.) takes $O(\log N)$ time. The inner **forall**-loop in step (2.) takes a constant time; therefore, step (2.) takes n time steps. Thus, the algorithm takes $O(\log N)$ time to schedule and n time steps to do the actual routing without link conflict.

As an illustration, consider again the example permutation $f_{\pi,a}$ of Figure 1-(a) where $n = 4$, $k = 2$, $a = 1010$ and $\pi(0) = 1$, $\pi(1) = 3$, $\pi(2) = 2$, and $\pi(3) = 0$. That is, $f_{\pi,a}(x_3x_2x_1x_0) = x_0x_2x_3x_1 \oplus 1010 = \overline{x_0}x_2\overline{x_3}x_1$, where $\overline{x_i}$ is the complement of x_i. In this example, $F = \{3\}$, $F' = \{2\}$, $F'' = \{0\}$, $G = \{1\}$, $G' = \{0\}$, and for every $x = x_3x_2x_1x_0$ we have: $[x]_{F'} = x_2$, $[x]_{F''} = x_0$, and $[f_{\pi,a}(x)]_{G'} = d_0 = x_{\pi(0)} \oplus a_0 = x_1 \oplus 0 = x_1$. Consequently, $t(x) = ([x]_{F'} \oplus [f_{\pi,a}(x)]_{G'}) \bullet [x]_{F''} = (x_2 \oplus x_1) \bullet x_0$. Thus,

$t(0000) = 00$, $t(0001) = 01$, $t(0010) = 10$, $t(0011) = 11$

$t(0100) = 10$, $t(0101) = 11$, $t(0110) = 00$, $t(0111) = 01$

$t(1000) = 00$, $t(1001) = 01$, $t(1010) = 10$, $t(1011) = 11$

$t(1100) = 10$, $t(1101) = 11$, $t(1110) = 00$, $t(1111) = 01$.

Hence, the source-destination paths that will be established at time $t = 00, 01, 10, 11$ are:

$t = 00$: $0000 \rightarrow 1010, 0110 \rightarrow 1111, 1000 \rightarrow 1000$ and $1110 \rightarrow 1101$

$t = 01$: $0001 \rightarrow 0010, 0111 \rightarrow 0111, 1001 \rightarrow 0000$ and $1111 \rightarrow 0101$

$t = 10$: $0010 \rightarrow 1011, 0100 \rightarrow 1110, 1010 \rightarrow 1001$ and $1100 \rightarrow 1100$

$t = 11$: $0011 \rightarrow 0011, 0101 \rightarrow 0110, 1011 \rightarrow 0001$ and $1101 \rightarrow 0100$.

Figure 1-(b,c,d,e) shows these paths at each time t. Observe that at any time step t the sources belong to distinct rows, their destinations belong to distinct columns, and the source-destination paths do not conflict over links.

4 Self-Routing of Ω- and Ω^{-1}-Realizable Permutations

The Omega (Ω) network and its inverse Ω^{-1} realize many interesting, widely used permutations [3]. Therefore, efficient algorithms that self-route the Ω permutations and the Ω^{-1} permutations on the fixed routing mesh M_n are of great importance. This section will develop a simple, self-routing algorithm for the Ω permutations and another similar algorithm for the Ω^{-1} permutations. Again n is assumed to be a power of 2, $n = 2^k$. The Omega network under consideration has N input (and output) terminals, where $N = n^2$, so that M_n and Omega are of the same size.

The algorithm for the Ω permutations is based on the idea that every Ω permutation f is routable on the mesh M_n in two phases: A column-wise phase followed by a row-wise phase. In the column-wise phase, every node x sends its message to the node that is in the same column of x and the same row of $f(x)$. In the row-wise phase, every node sends to the final destination the message received in the column-wise phase. The algorithm for the Ω^{-1} permutations works in the opposite order: The row-wise phase is first and the column-wise phase is second. Therefore, we will elaborate the first algorithm only.

For every column $x_{k-1}...x_0$, define the mapping $g_{x_{k-1}...x_0}$ such that

$$g_{x_{k-1}...x_0}(x_{2k-1}...x_0) = d_{2k-1}...d_k x_{k-1}...x_0$$

where $f(x_{2k-1}...x_0) = d_{2k-1}...d_0$. Using the characterization of Ω-realizable permutations derived by Lawrie in [3], it can be shown that $g_{x_{k-1}...x_0}$ is a permutation of size n for all $x_{k-1}...x_0$. Define also, for every row $d_{2k-1}...d_k$, the mapping $h_{d_{2k-1}...d_k}$ such that

$$h_{d_{2k-1}...d_k}(d_{2k-1}...d_k x_{k-1}...x_0) = d_{2k-1}...d_0$$

where $d_{2k-1}...d_0$ is the destination of some unique node $x_{2k-1}...x_0$ in column $x_{k-1}...x_0$. Again, every $h_{d_{2k-1}...d_k}$ can be shown to be a permutation of size n. The column-wise phase consists of simultaneous routing of all the $g_{x_{k-1}...x_0}$ in their corresponding columns. After this phase, every node is holding a single (intermediary) message. The row-wise phase consists of simultaneous routing of all the $h_{d_{2k-1}...d_k}$ in their corresponding rows, where the data routed are the intermediary messages. It should be clear that after executing the column-wise phase and then the row-wise phase, the messages are in their appropriate destinations.

The specifics of each phase are discussed next. In each phase, the communication consists of routing n separate permutations over n separate linear arrays. One simple approach is to sequentially route each permutation in its linear array in n time steps, that is, in step i only the i-th node of the linear array sends its message to its (intermediate or final) destination in the same array. This is done first in the column-phase in all the columns simultaneously (in n time steps), then it is done in the row-wise phase in all the rows simultaneously, again in n time steps, making the total number of steps equal to $2n$. The code below summarizes this self-routing algorithm.

```
SELF-ROUTE-$\Omega(f)$
begin
1. forall node $x_{2k-1}...x_0$ pardo
2.      node $x_{2k-1}...x_0$ computes its
        intermediary destination
        $d_{2k-1}...d_k x_{k-1}...x_0$ using its
        destination $f(x_{2k-1}...x_0) = d_{2k-1}...d_0$;
   endforall
   /* the column-wise phase next */
3. forall column $x_{k-1}...x_0$ pardo
4.      for $x_{2k-1}...x_k$ = 0...0 to 1...1 do
5.           node $x_{2k-1}...x_k x_{k-1}...x_0$ sends
             [its message, the final destination]
             to its intermediary destination;
        endfor
   endforall
   /* the row-phase next */
6. forall row $d_{2k-1}...d_k$ pardo
7.      for $x_{k-1}...x_0$ = 0...0 to 1...1 do
8.           node $d_{2k-1}...d_k x_{k-1}...x_0$
             sends its received message
             to the destination stored
             in the header of that message;
        endfor
   endforall
end
```

Note that step (1.) is the only step that can be called a scheduling step. Because step (2.) takes a small constant time and step (1.) is a parallel step, the scheduling step (1.) takes a small constant time. Step (4.) takes n time steps, and thus step (3.) takes n time steps due to the parallelism involved. Similarly, step (6.) takes n time steps. Therefore, as indicated earlier, this algorithm takes a constant time for scheduling and $2n$ steps for the actual routing.

5 Self-Routing of other Common Communication Patterns

Several communication patterns, other than single permutations, are very common and deserve special attention. These include fan-out communication (i.e., broadcasting), fan-in communication required in certain applications such as semi-group computations (e.g., addition of N numbers), and communication patterns in divide-and-conquer parallel algorithms. This section will address the routing of these patterns on the fixed routing mesh M_n.

5.1 Broadcasting

Broadcasting in a linear array of n nodes is explained first. Broadcasting in meshes follows as a two-phase process of row-wise broadcasting followed by column-wise broadcasting.

Suppose that a node x is to broadcast a message to all the nodes in a linear array L_n of n nodes. The broadcasting is, at least conceptually, a recursive process. In *the basis step*, node x sends its message to some arbitrarily selected node x' such that x and x' occupy two separate halves of L_n. For example, if $x < \lfloor \frac{n}{2} \rfloor$, then $x' = n - 1$; otherwise, $x' = 0$. In *the resursive step*, x broadcasts in its half of L_n, and x' broadcasts in its own half (i.e., the other half) of L_n, where these two sub-broadcasts are done simulataneoulsy since they do not conflict with one another over links. The number $C(n)$ of routing steps needed to carry out this process satisfies: $C(n) = C(\frac{n}{2}) + 1$. The term 1 in the right hand side represents the single time step needed by the basis step $x \rightarrow x'$. From this recurrence relation it follows that $C(n) = \log n$.

To broadcast from a node x in M_n, a linear array broadcast is first conducted in the row of x. Then, n simultaneous linear array broadcasts are conducted in the columns, where the broadcasting node in each column is the node at the intersection with the row of x. The number of routing steps of this process is $\log n + \log n$, which is $\log N$.

This broadcasting is faster than broadcasting on packet-switched $n \times n$ meshes because the latter brodcasting requires $2n - 1$ steps. This exhibits an area of superiority of circuit-switched meshes over their packet-switched counterparts.

5.2 Fan-in Communication

Fan-in communication arises in semi-group computations which involve evaluting a term of the form $A_0 * A_1 * ... * A_{N-1}$, where $*$ is an associative operator and every term A_i is stored in processor i. For example, $*$ can be 'addition', 'multiplication', boolean 'and', etc.

The fan-in communication pattern required in this problem is simply the opposite pattern of broadcasting. Therefore, it can be done in a self-routing fashion in $\log N$ routing steps using the mirror-image process of broadcasting.

5.3 Divide-and Conquer Communication

We will consider the subclass of divide-and-conquer algorithms which split their input into two equal halves, then call themselves recursively on each half, and finally merge the 2 subsolutions. The recursion can be a conceptual one while the actual implementation can be iterative. Whether we adopt the top-down recursive approach or the bottom-up iterative approach, the communication requirements are the same. However, for ease of analysis, the recursive formulation is adopted.

Assume that the input size is $N = n^2$ and that every processor stores one of the input items. Let $C(N)$ denote the number of routing steps required by such algorithms. One way to execute these algorithms is:

1) Do the splitting such that the resulting two halves of the data reside in the top $\frac{n}{2}$ rows and the bottom $\frac{n}{2}$ rows of M_n. Let $C_{splitting}$ denote the number of routing steps needed to do the splitting.

2) Call the algorithm recursively on each half. Each call executes on its $\frac{n}{2}$ rows. The two calls execute independently and simultaneously, costing $C(\frac{N}{2})$ routing steps.

3) Merge the two subsolutions into the final solution. Let $C_{merging}$ be the number of routing steps required by the merging.

$C(N)$ satisfies then the following relation: $C(N) = C(\frac{N}{2}) + C_{splitting} + C_{merging}$. In particular, if the splitting and merging take a constant number of routing steps (as in semi-group computations), then $C(N)$ is $O(\log N)$. If the splitting and merging take $O(\log N)$ routing steps (as in parallel prefix below where broadcasting is needed), then $C(N) = O(\log^2 N)$. If the splitting and merging take a constant number of permutations (as in many cases such as FFT), then $C(N) = C(\frac{N}{2}) + cn$ for some constant c. This yields that $C(N) = C(n) + c'n$ for some constant c'. $C(n)$ represents the number of routing steps of each call when the recursion has reached the level where the calls are to execute on single rows. Since routing a permutation of size n on a linear array of n nodes can be done sequentially in n routing steps, it follows that $C(n)$ satisfies the relation $C(n) = C(\frac{n}{2}) + n$. Therefore, $C(n) = 2n - 2$. This leads to $C(N) = O(n)$.

We will now apply this divide-and-conquer framework to parallel prefix and FFT. In the case of parallel prefix, the input is an array $A[0..N-1]$, and the desired output is also an array of N terms $A_{0:0}, A_{0:1}, ..., A_{0:N-1}$, where $A_{0:i} = A[0] * A[1] * ... * A[i]$ and $*$ is an associative operator. Processor i stores input $A[i]$ and is charged to compute $A_{0:i}$. Define $A_{j:i} = A[j] * A[j+1] * ... * A[i]$ for all $j < i$. In the recursive formulation, the first recursive call is on the subarray $A[0..\frac{N}{2} - 1]$ and results in the output $A_{0:0}, A_{0:1}, ..., A_{0:\frac{N}{2}-1}$. The second recursive call is on the subarray $A[\frac{N}{2}..N-1]$ and results in the output $A_{\frac{N}{2}:\frac{N}{2}}, A_{\frac{N}{2}:\frac{N}{2}+1}, ..., A_{\frac{N}{2}:N-1}$. By noting that $A_{0:i} = A_{0:\frac{N}{2}-1} + A_{\frac{N}{2}:i}$ for all $i \geq \frac{N}{2}$, it can be seen that the merging step requires every node in the second half of the mesh to receive $A_{0:\frac{N}{2}-1}$ from node $\frac{N}{2}-1$. Thus, the communicating part of the merging is a broadcast and takes $\log N$ steps. Since the splitting was trivial and required no routing, the relation for $C(N)$ is $C(N) = C(\frac{N}{2}) + \log N$. Making use of the fact that $C(2) = 1$, we derive that $C(N) = \log^2 N$. This is slower than the optimal case by only a factor of $\log N$, but it is much faster than $O(n)$ which is required by the same algorithm if the communication model is packet switching.

In the case of recursive FFT, a close inspection of the standard FFT recursive algorithm [8] shows that the splitting part and the mrging part require one BPC permutation each. The splitting BPC maps $x_{2k-1}x_{2k-2}...x_0$ to $\overline{x_{2k-1}}x_{2k-2}...x_0$, and the merging BPC is the perfect shuffle. It follows that $C(N) = O(n)$ as indicated earlier.

The above applications and analysis show that if the splitting and merging require constant or logarithmic numbers of routing steps on the fixed routing mesh, then the communication complexity is an order of magnitude faster on the circuit-switched mesh than on the packet-switch mesh. On the other hand, if the splitting and merging require data permuting, then no significant improvement is achieved by circuit switching.

6 Conclusions

This paper developed self-routing algorithms to schedule interesting classes of permutations on the circuit-switched fixed routing $n \times n$ mesh of $N = n^2$ nodes. It was shown that the bit-permute-complement permutations can be scheduled in $O(\log N)$ time to route in n time steps, and that the Ω- and Ω^{-1}-realizable permutations can be scheduled in constant time to route in $2n$ time steps. Other important communication patterns were also addressed. Broadcasting and fan-in communication were shown to self-route in $\log N$ routing steps, parallel prefix was shown to require $\log^2 N$ routing steps, and FFT was seen to require $O(n)$ routing steps.

In all but the case of FFT, the circuit-switched fixed routing mesh performed better than its packet-switched counterpart. It is true that in the case of permutation routing, the packet-switched mesh takes $O(n)$ steps. However, the delay incurred by the multiple hops that each message has to undergo to reach its destination contributes a significant constant factor to the communication delay in packet-switched meshes. This makes the n-step circuit-switched routing a more attractive alternative.

References

[1] S. H. Bokhari, "Communication Overheads on the Intel iPSC-2 Hypercube," ICASE Interim Report 10, May 1990.

[2] A. Gottlieb and C. P. Kruskal, "Complexity Results for Permuting Data and Other Computations on Parallel Processors," *Journal of the ACM*, Vol. 31, No. 2, pp. 193–209, April 1984.

[3] D. K. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Comput.*, C-24, pp. 1145–155, Dec. 1975.

[4] D. Nassimi ans S. Sahni, "An Optimal Routing Algorithm for Mesh-Connected Parallel Computers," *J. ACM*, vol. 27, No. 1, pp. 6-29, Jan. 1980.

[5] D. Nassimi ans S. Sahni, "A Self-Routing Benes Network and Parallel Permutation Algorithms," *IEEE Trans. Comput.*, C-30, pp. 332–340, May 1981.

[6] C. S. Raghavendra and V. K. Prasanna Kumar, "Permutations on Illiac IV-Type Networks," *IEEE Trans. Comput.*, Vol. C-35, No. 7, pp. 662–669, July 1986.

[7] C. Seitz et al. "The Architecture and Programming of the Ametek Series 2010 Multicomputer," in G. Fox, editor, *Proc. 3rd Conf. Hypercube Concurrent Architecures*, pp. 33–36, 1988.

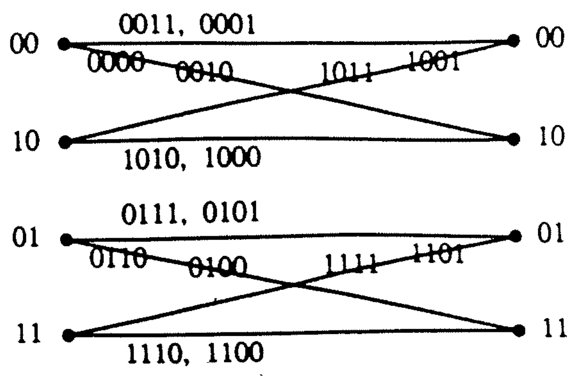
[8] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, C-20, pp. 153–161, Feb. 1971.

[9] L. G. Valiant, "A Scheme for Fast Parallel Communication," *SIAM J. Comput.*, Vol. 11, No. 2, pp. 350–361, May 1982.

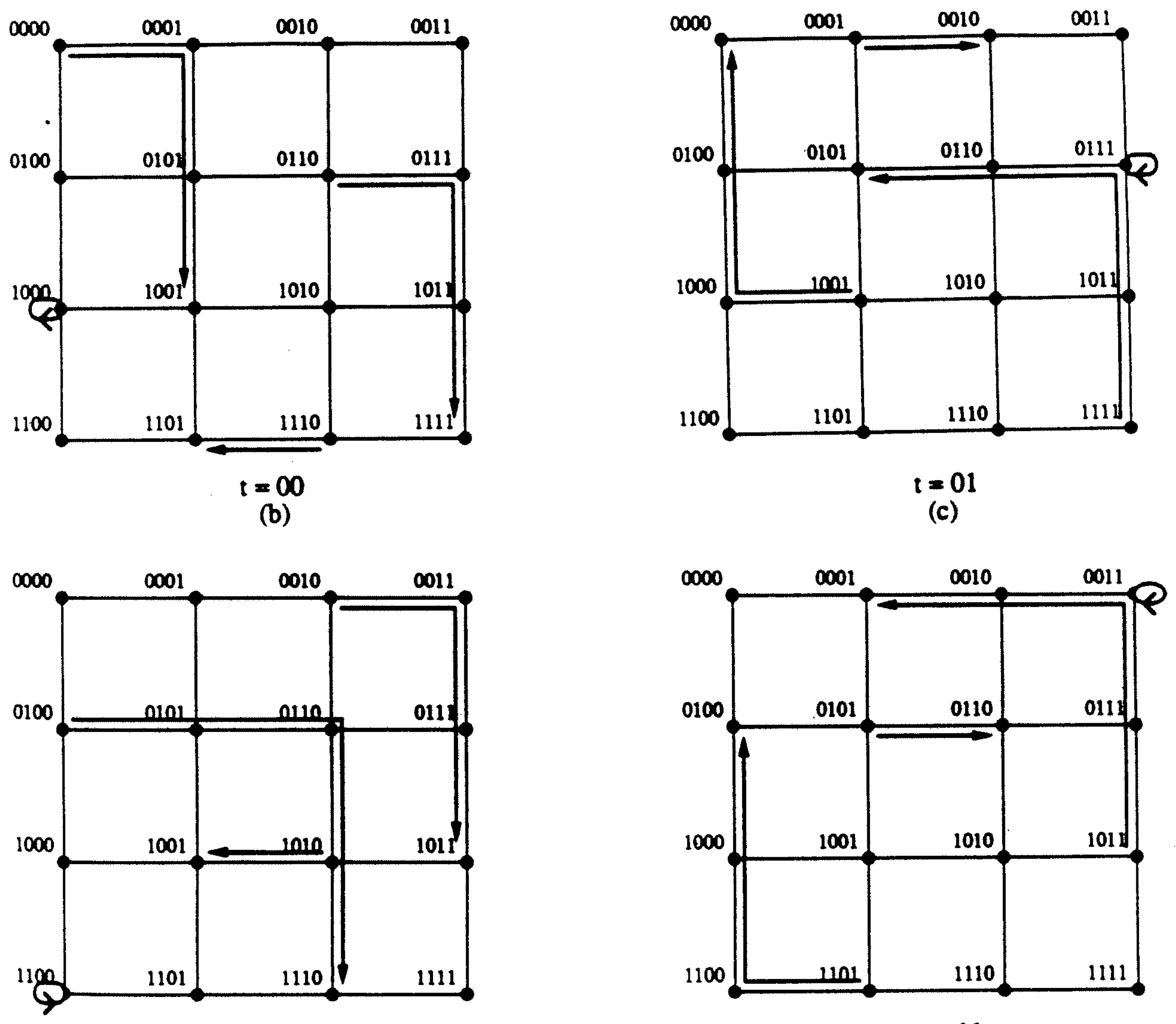
$n = 4,\ k = 2,\ a = 1010$

$\pi(0) = 1,\ \pi(1) = 3,\ \pi(2) = 2,\ \pi(3) = 0$

$f_{\pi,a}(x_3x_2x_1x_0) = \overline{x_0}x_2\overline{x_3}x_1$



(a)



The Bipartite Graph (a), and The Schedule (b,c,d,e) of a permutation $f_{\pi,a}$

Figure 1