Barrier Synchronization on Wormhole-Routed Networks

Yuzhong Sun, *Member*, *IEEE Computer Society*,
Paul Y.S. Cheung, *Senior Member*, *IEEE*, and Xiaola Lin

Abstract—In this paper, we propose an efficient barrier synchronization scheme on networks with arbitrary topologies. We first present a distributed method in building a barrier routing tree. The barrier messages can be delivered adaptively according to the hierarchy of the established barrier tree to void congestion and faulty nodes in the network. We then propose a new technique, called *bandwidth-preempting* technique, for a blocked barrier message to preempt a channel occupied by a data message so that the latency of a barrier message can be controlled without affecting much of the overall system performance. We also propose an analytical performance model and present simulation results for the performance evaluation of the proposed scheme. Performance evaluations show that the proposed scheme outperforms the existing algorithms for barrier synchronization.

Index Terms—Barrier synchronization, network topology, tree-based routing, data message, barrier message, bandwidth-preempting technique.

1 Introduction

A *barrier* is a synchronization point in a parallel program at which all processes involved must arrive before they can proceed further [1]. Barrier synchronization is a fundamental collective communication operation that is frequently used in parallel systems [2], [3]. It has been extensively studied in the literature and many schemes have been proposed for fast barrier synchronization through software, hardware, or their combination [4], [5], [6], [7], [8].

Barrier synchronization can be implemented in a *software-based* approach [6], [7], which has been used in IBM SP2, Intel Paragon, and other parallel systems. In the software method, a total exchange is performed by sending several phases of messages via the system message-passing routines. The software-based approach requires no special hardware support and it is easy to port to a new system. However, it incurs a high latency for barrier operations. To reduce the high software latency, several parallel systems, such as Cray T3D, have implemented dedicated hardware networks for barrier synchronization and other collective communication operations [9]. The latency in these systems is much lower compared with the software method, but at the expense of having multiple hardware networks. Thus a compromise is to provide hardware support for barrier synchronization in the existing data network [10].

In this paper, we focus on distributed-memory scalable computers with a wormhole routing mechanism. In a distributed-memory system, a barrier synchronization is usually performed in two phases. In the *reduction* phase, when a process in the barrier group arrives at the barrier, it informs a selected process in the group, called *center*. After the center receives the reduction information from all processes (including center itself), the *distribution* phase is initialized, in which the center sends a message to inform all other processes to proceed with the following computation.

A path-based approach based on multidestination worms is proposed for deadlock-free efficient barrier operations on wormhole networks in [8]. A *multidestination worm* is a message that carries multiple destination addresses so that it can be delivered to multiple destinations for multiple barrier processes with a single startup delay. The major problem with this scheme is that a message header has to carry the information of multiple destinations. Compared with the barrier message body that is very short only for synchronization, the message header is too long, resulting in waste of the network bandwidth.

A tree-based routing scheme for supporting barrier synchronization in 2D mesh is developed to avoid the problem in the multidestination worm method [5]. A *collective synchronization* (CS) tree is established at the barrier group creation time. The barrier message is then routed through the CS tree. For a barrier group, each router at a node of the CS tree records the input/output ports connected to its parent or child in the CS tree. No destination information is required in the barrier message after the CS tree is established. In [4], a reliable hardware barrier synchronization scheme is described. In this hardware method, each switch in the network is augmented with a barrier unit that operates on tiny barrier packets in several phases during a barrier synchronization. The *combine* (or reduction), *multicast* (or distribution), and *acknowledgement* packets for the barrier synchronization are delivered in different phases, respectively. The tree-based mutlicast algorithm in [11],

- *Y. Sun is with MxTek Networks, Inc., Albuquerque, NM 87110. E-mail: syzh@ieee.org.*
- *P.Y.S. Cheung and X. Lin are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong. E-mail: {cheung, xlin}@eee.hku.hk.*

Manuscript received 13 May 1999; revised 30 Nov. 2000; accepted 07 Dec. 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109805.

proposed for irregular topology, can be used in the multicast phase of the barrier scheme in [5]. However, congestion due to bursty traffic of data messages may occur to affect the barrier message latency in the proposed schemes in [4] and [5]. Also, the adaptive routing techniques for congestion avoidance and fault tolerance cannot be used because the barrier messages have to be routed strictly along the established routing tree. How to obtain the updated global information of the underlying network topology to initialize the routing tree may be another problem in using these approaches.

In the past, high-performance distributed-memory multiprocessor systems usually used regular networks, such as mesh or hypercube. Recently, irregular topologies are frequently used in connecting workstations in cluster and other distributed shared memory systems (DSMs) [12]. In this paper, we propose a new efficient barrier synchronization scheme for networks with arbitrary topologies. We first present a distributed method in building a barrier routing tree when the barrier operation is performed for the first time. The barrier synchronization tree is naturally formed by the barrier message flows. In the subsequent barrier operations, the barrier messages can be delivered adaptively according to the hierarchy of the established barrier tree to void congestion and faulty nodes in the network. We then propose a new technique, called the *bandwidth-preempting* technique, for a blocked barrier message to preempt a channel occupied by a data message, so that the latency of a barrier message can be controlled without affecting much of the overall system performance. We also propose an analytical performance model and conduct a simulation for the performance evaluation of the proposed barrier synchronization scheme. Performance evaluations show that the proposed scheme outperforms the existing algorithms for barrier synchronization.

The paper is organized as follows: Section 2 describes how to build a synchronization routing tree when the barrier synchronization is first performed. The preempting-bandwidth technique for barrier messages is proposed in Section 3. The performance of the proposed algorithm is evaluated and studied in Section 4, followed by the conclusions in Section 5.

2 Building a Barrier Synchronization Tree

The switch-based networks we consider are similar to those in [11]: The network consists of a set of switches connected in an arbitrary topology. Each switch has some ports that can be connected to processors (nodes) or other switches. The wormhole routing mechanism is adopted in transmission of the data messages. The unicast routing, such as the Autonet routing algorithm in [13], can be used for message passing.

As defined in MPI, the membership of a barrier group is fixed at the group creation time and every group member has complete information for all the members in the group [5]. A unique *group ID* identifies the group. We assume that one process is executed on one physical node and, thus, we will not distinguish between the two terms "process" and "node." A participating node in the barrier synchronization is referred to as a *member node*, or even *member*, and a message in barrier synchronization is referred as a *barrier message*. Also, the words "link" and "channel," as well as "switch" and "router," will be used interchangeably.

A tree-based routing scheme can be used for the barrier message passing. We first define a barrier synchronization routing tree as follows:

Definition 1. *A barrier synchronization tree (or BS tree) for a given barrier group is a tree in the network that is rooted at a selected member (or center) and connects all member nodes in the group together. For a BS tree, a branch node is defined as the node that has more than one child.*

Definition 2. *A barrier synchronization routing tree (or BSR tree) $T(V,E)$, for a given BS tree $t(V,E)$, is a routing tree in which the $V(T)$ only contains the member nodes and the branch nodes in $V(t)$ and a link (u,v) in $E(T)$, if and only if there exists a path $(u,u_1,\ldots u_2,u_k,v)$ in $t(V,E)$ and $u_i \notin V(T)$, $1 \leq i \leq k$. For convenience, the nodes in $V(t)$, but not in $V(T)$, are defined as the intermediate nodes in the BS tree t.*

In our proposed method, when a barrier synchronization is first executed, each member sends a reduction message to a selected center member after it arrives at the barrier. Any underlying deterministic routing mechanism can be used for the message transmission. The message passing paths of these reduction messages will form a BS tree rooted at the center. For simplicity, we only need to establish a BSR tree for the BS tree since a node in the BSR tree can deliver a message to its neighboring node by using the underlying unicast routing mechanism. Building a BSR tree in the network is to identify the nodes in the BSR tree, called *BSR node*, and record the parent-child relationship for each BSR node.

In order to establish such a BSR tree in a distributive fashion, each node in the BSR tree can determine its parent-child relationship according to the reduction messages traversing through it. Then, after the center receives all reduction messages and records the parent-child information (with no parent for center), it will send the distribution message to all members along the BSR tree. The first execution of the barrier synchronization, together with building up the BSR tree, is called *initialization*. Note that a barrier synchronization operation is usually performed many times in a parallel program. In the subsequent barrier synchronization execution, the established BSR tree can be used repeatedly for routing the reduction and distribution messages. The adaptive routing method can be used for the message transmission after the initialization.

In building a BSR tree, some additional information, shown in Fig. 2, needs to be carried in the reduction messages in addition to the original information in the message header, such as the source, destination, and other information.

In Fig. 1, the "Group ID" identifies the barrier synchronization group, the "Message type" indicates a reduction or distribution message, the "New BSR node" is the address of the new BSR node located during message passing, and the "BSR node tag" is used in identifying a branch node and will be explained later.

Group ID	Message type	New BSR node	BSR node tag

Fig. 1. The additional information in a reduction message.

Group ID	Parent/child, address	From/to channel	Other information

Fig. 2. An entry for a barrier group in the status table at a router.

At each router, there is a status table for a barrier group. Each item of the status table records the parent-child relationship, such as either parent or child and its address. It also includes other information, such as which physical channel goes to the parent or child (Fig. 2). The router design for the proposed method generally follows that of [4] and [5]. The status table in each router of a BSR node can be a register set and is organized as a fully-associative cache using the group ID as the cache tag.

The process of building a BSR tree is described as follows: Initially, only the members are marked as BSR nodes. Each member sends a reduction message, with its "BSR node tag" (BNT) being 0, toward the selected center member. When a reduction message M arrives at a router of node N, three cases can be identified as follows:

1. The BNT in M is equal to 0: Which means that no other reduction message has ever arrived at the router from the same physical channel from which M arrives. The source node in the message should be recorded as a child in the status table. We may further identify the following two cases:

   1.1. N is a non-BSR node: By checking the status table in the router, if the message is the first reduction message (with no other child in the table) arriving at the router, the BNT remains at 0. Otherwise, node N is marked as the BSR node. Because N is now found to have more than one child, the address of N is put into the "new BSR node" field of the message and the BNT is set to 1 to indicate that a new BSR node N has been located.

   1.2. N is a BSR node: BNT is set to 2 to indicate that message M will no longer be needed to provide information in subsequent operations in building the BSR tree.

   For example, in the network in Fig. 3, the reduction message from node 1, m_1, has its BNT equal to 0, initially. When it arrives at node 2, which is a non-BSR node, the source, node 1, in m_1 is recorded as a child in the status table at the router of node 2. Since m_1 is the first reduction message traveling through node 2, the BNT of m_1 remains at 0. m_1 is then forwarded to node 4. When m_1 arrives at node 4, if the reduction message, m_3, from node 3, has not arrived at node 4, the same operations as those in node 2 will be performed. On the other hand, if m_3 has arrived at node 4 before the arrival of m_1, node 4 should be now marked as a BSR node in addition to adding node 1 as the child of node 4. Also, the BNT in m_1 is set to 1 to indicate that a new branch node has been located and the address of node 4 is put into m_1 (its "new BSR node" field). The message m_1 will be forwarded to node 5 and so on.

   Now, let us consider the case that m_1 has BNT equal to 1 and arrives at node 5, which means that m_3 has arrived at node 5 before the arrival of m_1, hence, the router of node 5 has recorded node 3 in m_3 as its child. However, the child should now be changed from node 3 to node 4 (the new BSR node) because m_1 arrives at the same channel as m_3 (from node 4 to node 5). The BNT of m_1 remains 1 at node 5. When m_1 reaches node 7, again, the original child, node 3, in router of node 7, should now be changed to node 4. Obviously, if the reduction message from node 6, m_6, has already arrived at node 7, node 7 must have been marked as the BSR node since m_3 has also arrived at node 7. Thus, the BNT in m_1 should be set to 2. On the other hand, if m_6 has not arrived at node 7, the operations are the same as those performed in node 5. Using this example, it is easy to understand the operations in Case 2 below.

2. The BNT in M is equal to 1: Which means that another reduction message m has already arrived at the router from the same physical channel from which M arrives. The "new BSR node" in message M should be recorded as a child in the status table to replace the original child recorded from message m.

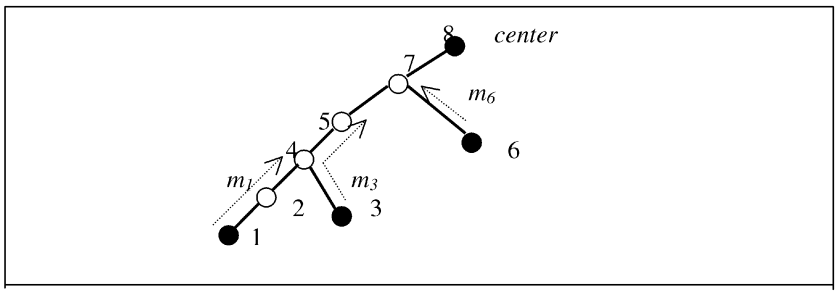


Fig. 3. An example of a BS tree.

Input: the membership information in a barrier group and a network topology.

Output: a successful barrier synchronization and a BSR tree.

Algorithm:

(1) Only the members are marked as BSR nodes for the barrier group, and each reduction message has its BNT being 0 initially.

(2) Each member sends a reduction message toward the center node using the underlying unicast routing mechanism, and the selected next node for forwarding is recorded as the parent in the status table, until the center receives the reduction messages from all the members.

When a reduction message *M* arrives at a router of node *N*,
Case 2.1: When the BNT in *M* is 0:
- a. The source node in *M* , *S*, is recorded as a child of *N* in the status table.
- b. Node *N* is a non-BSR node: If node *S* is the only child in the table, the BNT of *M* remains to be 0. Otherwise, node *N* is put into the "new BSR node" field in *M*, the node *N* is marked as BSR node, and the BNT of *M* is set to 1. Go to (2).
- c. Node *N* is a BSR node: BNT of *M* is set to 2.

Case 2.2: When the BNT in *M* is 1:
- a. The "new BSR node" in *M* is recorded as a child of *N* in the status table.
- b. If node *N* is a non-BSR node, the BNT of *M* remains to be 1. Otherwise, the BNT of *M* is set to 2.

(2) The center sends out distribution message to all members using the established BSR tree.

Fig. 4. The algorithm in building a BSR tree for barrier synchronization.

If node N is a non-BSR node, there should be no other reduction messages arriving from other previous channels, and the BNT remains to be 1. The BNT is set to 2, if N is a BSR node.

3. The BNT in M is equal to 2: Message M cannot provide any new information in building the BSR tree and, therefore, nothing needs to be changed in the status table and the message itself.

The formal description of the initialization is given in Fig. 4.

Theorem 1. *For any given barrier group on a network with arbitrary topology, a BSR tree can be established by the algorithm in Fig. 4.*

Proof. As described before, the purpose of building a BSR tree is to identify the BSR nodes and record the parent-child relationship in the router of each BSR node in the network. We will do the induction on the number of the members in the barrier group. When the number of members is 1, it is trivial. When the number of members is 2, a member can send a reduction message, with its BNT being 0, to another member (center) to establish the parent-child relation. The BSR tree for the two nodes can easily be built by using the algorithm in Fig. 4. Now, suppose that the theorem is true for any given barrier group with k members. We prove that it is also true for any given barrier group with $k+1$ members.

For any given barrier group with $k+1$ members, B_{k+1}, there exists a unique BS tree, as well as its *unique* corresponding BSR tree $T_{k+1}(V,E)$, which is formed by the paths of the reduction message passing in using the deterministic routing mechanism. Note that a link (u,v) in a BSR tree corresponds to a path from u to v in its corresponding BS tree. Now, consider a nonleaf BSR node p whose child has the maximum distance to the root in $T_{k+1}(V,E)$. Suppose that node c_1 is the child of p and, therefore, c_1 is a leaf BSR node and a member. Without loss of generality, we also assume that the reduction message from c_1 arrives at p after the arrival of the reduction messages from all other children (if any) of p. We further assume that node g is the parent of node p in $T_{k+1}(V,E)$.

In initialization, c_1 sends a reduction message M_1 with its BNT being 0, to p (toward the root). M_1 may travel through a number of intermediate nodes to reach p and its BNT remains 0 according to the operations of Step 2, Case 2.1b, in the algorithm. The following three cases can be discussed:

1. Node p has only one child, c_1 (see Fig. 5a): p must be a member, otherwise it would not be a BSR node in $T_{k+1}(V,E)$. For the barrier group $B_k = B_{k+1} - \{c_1\}$, by the induction hypothesis, a unique BSR, $T_k(V,E)$, can be built by using the algorithm in Fig. 4. Apparently, we must have

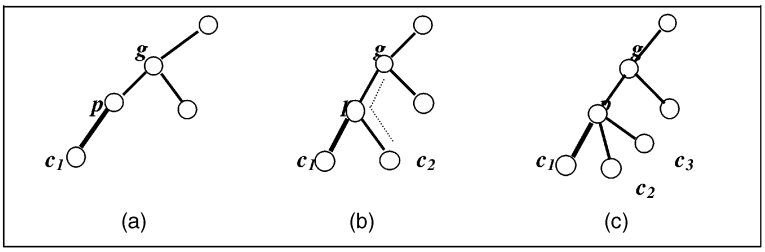


Fig. 5. An example of a BSR tree.

$$V(T_k) = V(T_{k+1}) - \{c_1\}$$

and

$$E(T_k) = E(T_{k+1}) - \{(c_1, p)\}.$$

When M_1 arrives at p, in terms of Step 2, Case 2.1, in the algorithm, node c_1 will be recorded as a child of p. Also, the BNT in M_1 is set to 2 because p is a BSR node and, hence, M_1 will no longer be needed to provide further information in building the BSR tree. Therefore, $T_{k+1}(V, E)$ can be built by the algorithm in Fig. 4, where

$$V(T_{k+1}) = V(T_k) + \{c_1\}$$

and

$$E(T_{k+1}) = E(T_k) + \{(c_1, p)\}.$$

2. Node p has two children, c_1 and c_2 (see Fig. 5b): If p happens to be a member, the proof is the same as that in Case 1. Now, consider that p is not a member. For the barrier group $B_k = B_{k+1} - \{c_1\}$, by the induction hypothesis, a unique BSR, $T_k(V, E)$, can be built by the algorithm in Fig. 4. The node p should not be in $T_k(V, E)$ because p is neither a member nor a branch node in its corresponding BS tree. Thus, we should have that

$$V(T_k) = V(T_{k+1}) - \{c_1\}$$

and

$$E(T_k) = E(T_{k+1}) - \{(c_1, p), (c_2, p), (p, g)\} + \{(c_2, g)\}.$$

When M_1 arrives at p, according to Step 2, Case 2.1, in the algorithm, node c_1 will be recorded as a child of p. Also, the BNT in M_1 is set to 1 (in Step 2, Case 2.1b) because p has another child, c_2, recorded. Node p will be put to the "new branch node" field in M_1. Then, M_1 is forwarded to g. It may go through some intermediate nodes and the BNT remains 1 (Step 2, Case 2.2b). When M_1 arrives at g, the p in M_1 will be recorded as the child of g to replace the original child c_2 and its BNT is set to 2 (Step 2, Case 2.2). Therefore, $T_{k+1}(V, E)$ can also be built by the algorithm in Fig. 5, where

$$V(T_{k+1}) = V(T_k) + \{c_1\}$$

and

$$E(T_{k+1}) = E(T_k) + \{(c_1, p), (c_2, p), (p, g)\} - \{(c_2, g)\}.$$

3. Node p has more than two children (see Fig. 5c): Note that, by our assumption in selecting c_1, the reduction messages from at least two children have already arrived before the arrival of the reduction message from c_1, therefore, node p should be included in $V(T_k)$. The proof for this case is the same as that in Case 1. □

After initialization, the subsequent barrier synchronizations will be performed by using the BSR tree. The router at each BSR node will send a reduction message to its parent only if it has received the reduction messages from all its children and the local processor. After the center (root) receives the reduction messages from all its children and its own processor, the distribution message will be sent from the center down to the members along the BSR tree until all members receive the message to complete the barrier synchronization.

Note that the message passing, in the algorithm in Fig. 4, can be implemented using the underlying unicast mechanism provided by the system. There are several advantages to using this method. First, a message's header for the synchronization may only contain one destination address to have a short message size. Second, after initialization, a message can be delivered adaptively to its destination parent or child since there may be many paths connecting a member node to its parent or child member in the BS tree. Finally, an intermediate node in the BSR tree can just work as usual as for date messages.

Tree-based routing in a wormhole system is susceptive to deadlock [14]. In our proposed scheme, we may build buffers inside the router for all barrier messages to support cut-through routing and, therefore, to avoid deadlocks in the tree-based routing [5]. It will be shown that the barrier messages are very short, thus, the amount of the buffers required in the router is quite small.

3 The Bandwidth-Preempting Technique

The barrier messages in our system are transmitted through the existing data network and, thus, the traffic of data messages and barrier messages is mixed. A small sized barrier message may be blocked by a long data message

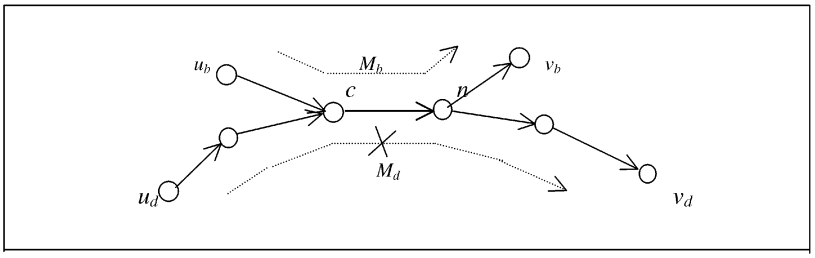


Fig. 6. Bandwidth-preempting for barrier message.

transmission in the contention for the same channel in the network. In this section, we propose a new bandwidth-preempting technique for a blocked barrier message to preempt a channel occupied by a data message so that the performance of the barrier synchronization can be greatly improved without affecting much of the overall system performance.

The underlying networks use wormhole routing, a technique that pipelines message transmission over a number of channels along its path to reduce the latency. In such a system, a packet is divided into a number of flits (flow control units). As the header (or the first) flit with routing information is routed toward its destination, the subsequent flits follow the routing path in pipeline fashion [15]. The basic idea of the proposed bandwidth-preempting method is to cut the data flits flow to preempt an occupied channel required for a barrier message transmission.

For example, in Fig. 6, suppose that the flits of data message M_d are occupying the channels of the path from u_d to v_d and barrier message M_b arrives at node c from node u_b toward node v_d. The M_b will be blocked because the channel (c, n) is being used by M_d. In our proposed method, the channel (c, n) will be preempted by interrupting the flit moving of the data message flow, so that it can be used by the barrier message M_b. In doing so, the flit of M_d at node c is stopped, channel (c, n) is then preempted for the barrier message M_b that consists usually of one or two flits. After the transmission of the short message M_b, channel (c, n) will be returned to data message M_d. In order to resume the transmission of M_d, we must preserve the path from n to v_d since the flit of M_d (nonheader flit) currently in c has no routing information. Thus, a sequence of the padding null flits will be generated at n and sent out following the current flit in n, to preserve the path from n to v_d, until the flit flow of M_d is resumed. At the destination of the data message, these padding null flits will be discarded. The padding technique is first proposed in [15] for deadlock-free and fault-tolerant compressionless routing.

We assume that a flit has 16 bits. Thus a barrier message may consist of only one or two flits. Each physical communication channel can be divided into a number of independent *virtual channels* sharing the physical channel bandwidth [16]. The bandwidth-preempting method, as well as the barrier tree algorithm in Section 2, requires necessary hardware supports in the switch architecture. In this section, we propose three schemes of switch architecture varying in complexity and performance.

In our first scheme, one $(k+1) \times (k+1)$ and one $(k+2) \times k$ crossbars are required, as shown in Fig. 7. In addition, a padding flit generator is added to the switch. The purpose of using the two crossbars is that data and barrier messages can be mixed and put into different input buffers within the switch.They may progress through the physical links between two neighboring nodes.

In Fig. 6, when channel (c, n) is required to be preempted for a barrier message, the router of current node c sends a *padding request* to the next node n for the data message M_d. The padding request contains information, such as which (virtual) channel will be preempted. The switch at the next node n starts the padding generator after receiving the request. The padding flits follow the current flit of the message M_d until a *padding end* message is received from node c when the preempting is over.

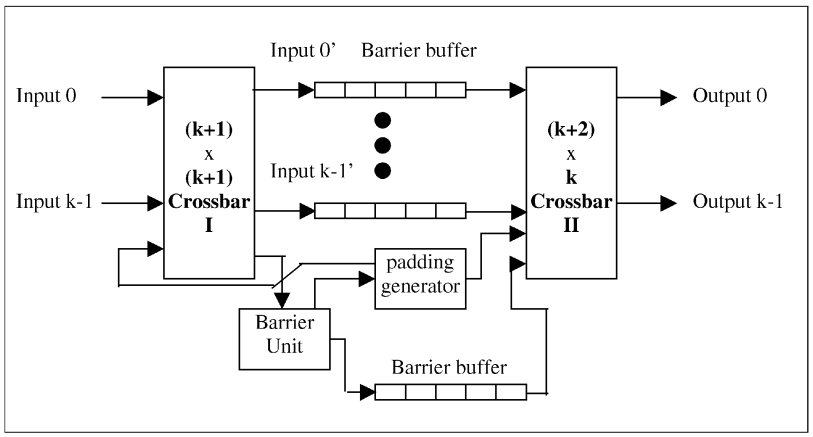


Fig. 7. A switch with preempting for barrier messages.

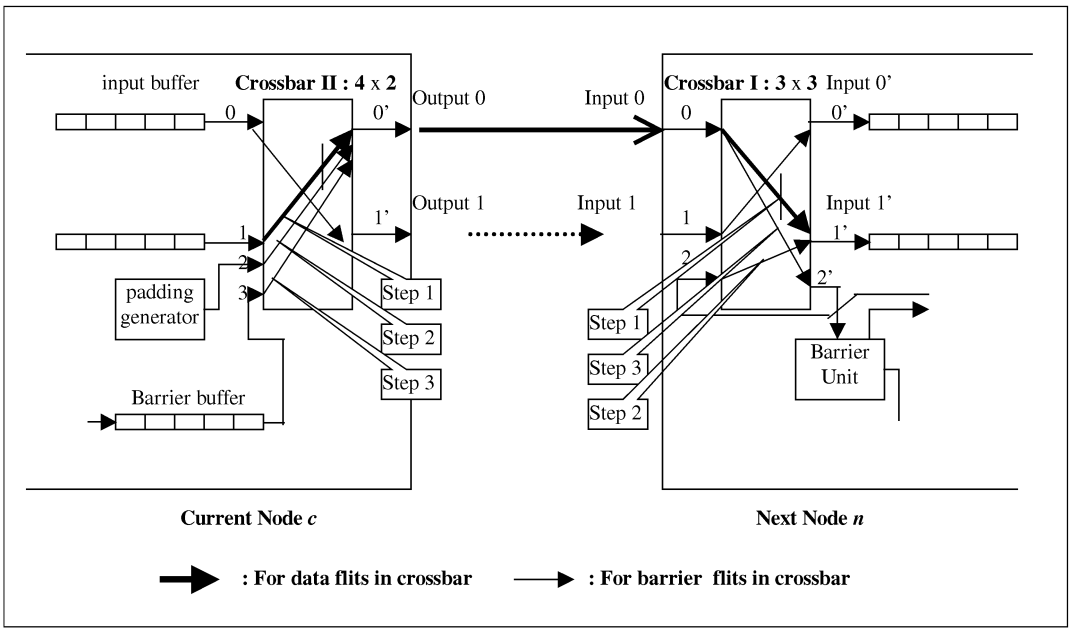


Fig. 8. Example for preempting a virtual channel for a barrier message.

The padding generator in the switch performs two functions: Producing padding requests and end messages and generating padding flits. A padding request, an end message, and padding flits are messages with fixed formats and contents. These messages can be directly prestored in some registers in the router.

The barrier unit is similar to that in [4], which contains a B-entry SRAM, or register set, and a controller for the padding generator. Entry i, in the SRAM, stores the current state of the barrier group i. Each entry should include the information, such as the barrier group ID, the preempting channel number, and the channel state. A barrier message can utilize any output port at the current node and any input port at the next node selected by the routing algorithm. In a switch, the barrier messages are stored only into the barrier buffer.

The zeroth to the $k-1$th input and output ports at crossbars I and II (Fig. 7) are used by data messages. The routing strategy for data messages at crossbar I is input port i is mapped to input port i', where $0 \leq i \leq k-1$, while barrier and padding messages can only use the k'th input port. Routing messages that cross crossbar II are determined by the routing algorithms according to the underlying network topologies. Data messages traverse the switch as if only a single crossbar were used.

An example for preempting a (virtual) channel is illustrated in Fig. 8. A virtual channel between two neighboring nodes in wormhole routing may be regarded as a pair of two flit buffers, respectively, at the switches of the two nodes [17]. Each of the virtual channels sends a flit, or a sequence of flits, in turn, either by cycling a token or by assigning a time slot among the virtual channels on a physical link. Fig. 8 illustrates an example of a barrier header flit at the current node that preempts a virtual channel from the current node to the adjacent next node. The connection in a crossbar is denoted by a pair of input and output ports in the crossbar. For simplicity, we only consider crossbar II (CN-II) at the current node c and crossbar I (CN-I) at the next node n.

We first consider the operations at the current node c. All virtual channels to node n are being assigned to the data messages. The router in node c decides (randomly or by priority) to preempt the virtual channel, that is, from $1 \rightarrow 0'$ at CN-II (node c) to $0 \rightarrow 1'$ at NN-I (node n), as shown in Fig. 8. The operations related to crossbar CN-II are:

1. Break down connection $1 \rightarrow 0'$ at CN-II.
2. Set a new connection, $2 \rightarrow 0'$ at CN-II, send a padding request generated by the padding generator to the next node, and break down the connection.
3. Set a new connection, $3 \rightarrow 0'$ at CN-II, and send flits of barrier messages.
4. If not any of the new barrier message uses the channel after an interval T, send a padding end message to the next node, break down connection $3 \rightarrow 0'$ at CN-II, and recover the original connection, $1 \rightarrow 0'$ at CN-II.

The time interval T may be specified by the compiler or the parallel program on the system if the barrier synchronization will be performed again in the near future. We then consider the operations at the next node n. The operations related to the preempting are performed on the crossbar NN-I as follows:

1. After receiving the padding request message from the current node, break down connection $0 \rightarrow 1'$ at CN-I.
2. Set a new connection, $2 \rightarrow 1'$ at CN-I. The padding flits are attached at the end of the message in the end of the input buffer $0'$ at the next node when the message proceeds.
3. Set a new connection, $0 \rightarrow 2'$ at CN-I, and put the received barrier message flits into the barrier unit at

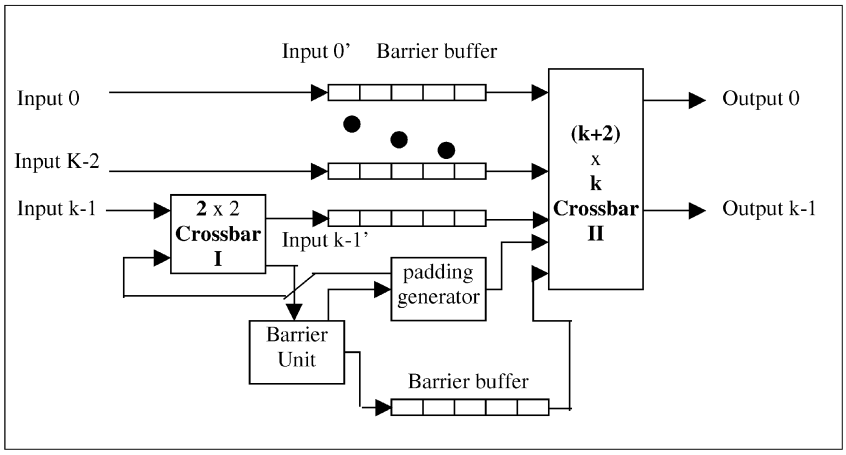


Fig. 9. The second proposed switch architecture.

the next node. Then, a virtual channel for the barrier message is established.

4. If the padding end message is received, connection $0 \rightarrow 2'$ at CN-I is broken down and the original connection, $0 \rightarrow 1'$ at CN-I, is recovered for data message.

To reduce the complexity of hardware of the first proposed switch architecture, we propose the second switch architecture in Fig. 9. Crossbar I of $(k+1) \times (k+1)$ in the first method (Fig. 7) is replaced by a 2×2 crossbar but at the expense of the capability of the preempting bandwidth. In this case, only a single fixed virtual channel between the two adjacent nodes is allowed to preempt for barrier messages. The routing for crossbar I is very simple in Fig. 9 because no routing in the crossbar is required. The barrier and padding messages are directly put to the barrier unit. The data messages will not traverse the additional 2×2 crossbar if the virtual channels they use are not preempted.

Without considering that the congestion occurred in the FIFO input buffers at the switch, we can further simplify the second switch architecture by removing the 2×2 crossbar in Fig. 9. The third proposed switch architecture is shown in Fig. 10. The capability of preempting is also further decreased. The preempting can only occur when there is at least one virtual channel on which the flit flow moves. Only in this case can a padding message be transferred successfully to the next router in cases of congestion. The next router breaks down the connection of the current virtual channel, according to the received padding message, establishing a new virtual channel for the barrier message identical to the operations in the first method.

It is important for a system to determine the bounds on critical communication operations, such as barrier synchronization on parallel systems [3], especially on the distributed shared memory systems and clusters with arbitrary network topologies. The proposed barrier scheme has the upper bound of barrier synchronization latency on networks with arbitrary topologies. By using the preempting technique, the bound of the proposed scheme can be independent of the data message traffic on the network. In our method, we may use the same underlying unicast routing for the barrier messages. In the BSR tree T described in Section 2, the distance of two nodes, x_i and x_j , is denoted by $D_T(x_i, x_j)$. If the minimum distance between x_i and x_j is known (i.e., by the fully adaptive minimum routing), Theorem 2 presents the upper bound of the latency of barrier synchronization.

Theorem 2. *For a barrier group $\{x_0, x_1, \ldots, x_p\}$, $0 \leq i \leq p$, on a network with arbitrary topology, if all operations of preempting*

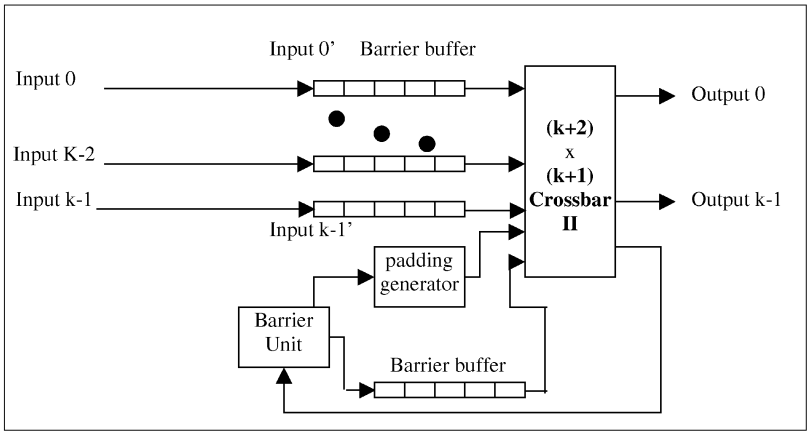


Fig. 10. The third proposed switch architecture.

for barrier messages are successful, the upper bound of the barrier operation is $2 \cdot T_b \cdot \max(D_T(x_i, center))$, $0 \leq i \leq p$, *where "center" is the center node and* T_b *is the latency for a barrier message to travel one hop.*

Proof. The proposed scheme consists of two phases. In the reduction phase, each member in the barrier group sends a reduction barrier message to the center member. The number of maximum hops in the BSR tree for these barrier messages is $\max(D_T(x_i, center))$, $0 \leq i \leq p$. In the distribution phase, the center sends a distribution barrier message to all members in the group along the BSR tree. The number of hops from the center member to the most remote members in the BSR tree is $\max(D_T(x_i, center))$, $0 \leq i \leq p$. By using the preempting technique, the waiting time due to contention for channel usage with data messages is a small constant and can be neglected. The latency T_b is a constant because the size of a barrier message is constant in our scheme. The overall latency for the barrier operation is therefore $2 \cdot T_b \cdot \max(D_T(x_i, center))$, which is independent of the data message traffic. □

4 Performance Evaluation

In networks with arbitrary topologies, the proposed bandwidth-preempting scheme for barrier synchronization should be able to outperform the schemes in [4] and [5], in the case of congestion, providing a controlled overhead for barrier synchronization. Intuitively, the performance of the proposed scheme should approach that of the schemes in [4] and [5] in cases of no congestion. The use of additional crossbars and the increase of the flit size of a barrier message have little effect on the overall system performance. The unicast routing for the additional crossbar is completed during the unicast routing for the second crossbar. No additional overhead is required. The location of the center member has an impact on the latency of barrier synchronization [18]. However, it is often difficult to determine the best location of a center on a network with arbitrary topology. Therefore, it is more feasible to find a barrier scheme that has the best average performance.

In this section, we first propose a simple analytical performance model to analyze the performance of the proposed scheme in the presence of congestion compared with the existing barrier schemes, such as those in [4]. We then present our simulation results to justify our analysis.

4.1 Analysis

In the following analytical performance model, the initialization overheads for barrier synchronization are not considered. Thus, we will first focus on the switch schemes proposed in Section 3. The following parameters are used in the simple analytical model, which are similar to those in [16]:

1. P_f: The probability that the flit flow over a virtual channel between two neighboring switches is blocked with uniform distribution.
2. P: The probability that a virtual channel between two neighboring switches serves a data message with uniform distribution.
3. T_p: The time for preempting a virtual channel between two neighboring switches.
4. T_b: The latency for a barrier message to traverse one hop.
5. Δ: The duration of congestion resulting from data messages over a virtual channel with uniform distribution.
6. k: The number of input ports equal to the number of output ports at a switch, assuming that $k > 1$.
7. T_s: Software startup, including send and receive for a processor.
8. L_{BS}: The level of the tree for the proposed barrier scheme.
9. L_{SO}: The level of the tree for the switched-based ordering algorithm [4], [11].

We also make the following assumptions similar to those in [16]:

1. Message destinations are uniformly and randomly distributed.
2. A message that arrives at its destination is consumed without waiting.
3. All barrier messages have the same length (one or two flits).
4. Each virtual channel is associated with a flit buffer that can store a whole barrier message.
5. Message blocking probabilities are independent.

For simplicity, we will analyze the member with the largest latency in the group in our scheme and the switch-based ordering barrier algorithm in [4]. In both schemes, such a member should be located at the lowest level of the tree. All communications for barrier synchronization in both schemes will be conducted along a barrier synchronization tree. The two schemes differ in their ways of establishing barrier trees and assigning virtual channels for barrier messages. Implementation of barrier synchronization consists of two independent communications phases: reduction and distribution. The effect of congestion in the distribution phase can be analyzed in the same way as that in the reduction phase. It is shown, in [19], that the increase and decrease rates of traffic in all the applications are especially steep, which implies that all nodes in the network are blocked and recovered almost at the same time. Thus, we assume that the members have the identical congestion interval, p. In our scheme, $T_b \geq T_p$ since a padding message has only one flit.

For the proposed three schemes of switch architecture in Section 3, as the switch architecture complexity decreases, its performance also decreases in handling congestion.

Case 1. The First Router Architecture (Fig. 7). In this case, any virtual channel between two neighboring switches can be preempted for a barrier message. The latency for the distribution phase is $2 \cdot (L_{BS} - 1) \cdot T_b$. Thus, the total latency T for a barrier synchronization operation is calculated by

$$T = T_s + 2(L_{BS} - 1) \cdot T_b + \sum_{i=1}^{2(L_{BS}-1)} (T_b + p^k \cdot T_p). \quad (1)$$

TABLE 1
Experimental Parameter Values

(t_s) Software overhead (send+recv.)	100, 1000 cycles
Time for flit to cross physical link	2 cycles
Time to route unicast packet at switch	2 cycles
Time for flit to cross switch crossbar	1 cycle
Time for flit to be read from input buffer	1 cycle
Time for reliability related handling of packet [8]	4 cycles
Time for arbitration between input and output	2 cycles

Case 2. The Second Switch Architecture (Fig. 9). Compared to Case 1, in which any output ports at a switch are allowed for preempting a barrier message, only a preoccupied input port connected to the 2×2 crossbar is permitted to serve the barrier message in the second switch scheme. This feature increases the probability of congestion over one hop in barrier synchronization from p^k to p. The total latency for barrier synchronization is

$$T = T_s + 2(L_{BS} - 1) \cdot T_b + \sum_{i=1}^{2(L_{BS}-1)} (T_b + p \cdot T_p). \quad (2)$$

Case 3. The Third Switch Architecture (Fig. 10). This case has the simplest hardware architecture but the worst performance among the three proposed switch schemes. The requirement for preempting a virtual channel in this scheme is that there must be at least one virtual channel over which the flit flow of data message is moving. The total latency is affected by Δ, which is the duration of congestion resulting from data messages over a virtual channel with uniform distribution. This means that a barrier message may have to wait for at least Δ time for preempting. The probability for such a congestion is determined by $p^k \cdot p_f$. The range of p_f is often dependent of the parallel applications and the underlying network. The latency for this scheme is given by

$$T = T_s + 2(L_{BS} - 1) \cdot T_b + \sum_{i=1}^{2(L_{BS}-1)} [T_b + p^k \cdot ((1 - p_f) \cdot T_p + p_f \cdot \Delta)]. \quad (3)$$

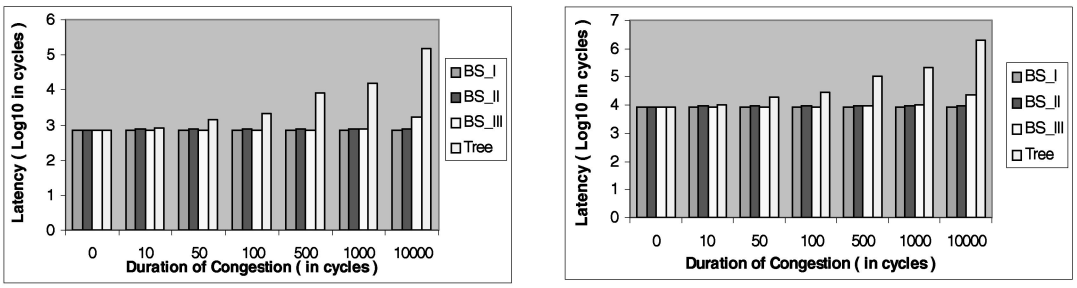
The latency of barrier synchronization in the scheme [4], however, is calculated as follows:

$$T = T_s + 2(L_{SO} - 1) \cdot T_b + \sum_{i=1}^{2(L_{SO}-1)} (T_b + p \cdot \Delta). \quad (4)$$

Checking the above four formulas, only the scheme in [4] and the third switch scheme in our scheme are affected by Δ. The latency of barrier synchronization in [4] is proportional to $p \cdot \Delta$ while the factor for Δ in the third switch scheme is $p^k \cdot p_f$. Generally, $1 > p \geq p_f$. For $k > 1$, we have $p \gg p^k \cdot p_f$. That is, when $\Delta \to \infty$, the latency for barrier synchronization in [4] approaches ∞ at a much faster speed than that of our third scheme. The latencies of the first and second proposed schemes increase by a fixed constant depending only on T_p. We usually have $T_p \ll \Delta$. For example, the time for preempting, T_p, may require 6~10 cycles as summarized in Table 1. We conclude that our schemes have much better performance than the schemes in [4] in the presence of congestion.

This analysis will be further verified in the following simulation. We also consider the special case that the network topology is a 2D mesh. In an $n \times n$ mesh, let $L_{BS} = L_{SO} = n - 1$. (4) can also describe the worst case in [5] for a 2D mesh.

Fig. 11 depicts the comparison between the proposed scheme and the algorithm in [4] for networks with arbitrary topologies and the comparison between the proposed scheme and the algorithm in [5] for a 2D mesh. In Fig. 11, *BS_I*, *BS_II*, and *BS_III* denote, respectively, the proposed three switch schemes. The scheme in [4] or [5] is specified by *Tree*. Fig. 11 illustrates the impact of the ranges of Δ and system sizes on performance of the scheme in [4] or [5] and



$L_{BS} = L_{SO} = 10$ (a)

$L_{BS} = L_{SO} = 128$ (b)

Fig. 11. Comparisons among the four analytical results: $T_s = 100$ cycles, $T_b = 16$ cycles, $p = 0.8$, $p_f = 0.2$, $k = 16$. (a) A switch-based irregular network. (b) A switch-based irregular network.

our schemes according to the simple analytical model. For simplicity, we assume that $L_{BS} = L_{SO}$. The analysis results show that our schemes outperform the scheme in [4] in cases of congestion. Fig. 11 shows that the proposed schemes remain especially stable in front of congestion. Furthermore, in the first and second switch schemes, the latency is almost independent of the traffic of data messages by using the preempting technique.

The tree-based routing in barrier operations may put the network into saturation when the number of members in a barrier group reaches a certain threshold value. This phenomenon implies that the P_f of each node may not be equal and a root's P_f may be much larger than that of the other node in the routing tree. It is especially difficult to calculate the dynamic distribution of P_f of nodes on networks with arbitrary topologies. The above analysis is to show the different performance profiles of the proposed schemes and the schemes in [4] and [5] in case of congestion.

4.2 Simulation on Networks with Arbitrary Topologies

A simulator is implemented to run the proposed first switch scheme in Fig. 7 and the scheme in [4] using the SO algorithm in [11]. The underlying network has arbitrary topology in which three ports in one switch are connected to the three different switches. The simulation is conducted to evaluate the performance of the two schemes for a range of system sizes, software startup overhead, number of members in a barrier group, number of members in congestion, and the congestion interval. A random number generator selects the members for a barrier group. The parameter values used in the simulation, the same as those in [4], are listed in Table 1. The cycle time is assumed to be 10 ns. We consider the systems with 300 and 1,200 nodes in cases of $t_s = 100$ and $t_s = 1,000$ cycles. To measure the effect of congestion on barrier synchronization, we randomly select a set of members in congestion in a barrier group. The congestion members cannot forward barrier messages but can receive barrier messages from other nodes until the congestion is gone. During the congestion, the congestion member can only receive barrier messages when all the input buffers of the member are not full. For simplicity, we assume that each selected member is simultaneously in a congestion state that lasts for the same time interval. Without loss of generality, we further assume that the congestion occurs at the beginning of the barrier synchronization and all members are assumed to arrive at the barrier at the same time.

Initialization Overhead. Initialization refers to establishing a barrier synchronization tree for barrier synchronization. The tree for barrier synchronization is statically determined with the global optimization in [4], [5], and [11]. The problem with the predetermined trees is how each node gets the updated global information on a network. The barrier trees cannot be changed during the barrier operation. To handle faulty components in the network, special fault-tolerant protocols have to be applied, increasing the overhead of barrier algorithms [4]. The proposed scheme builds a barrier tree in a distributed way. General adaptive routing algorithms can be directly applied after the initialization to avoid the faulty components. As shown in Fig. 12, when the number of members is small with respect to the network size, the initialization overhead is very small and can be neglected. However, as the number of members in the group increases gradually to approach the network size, the initialization overhead increases rapidly. In the worst case, the latency of the first round of a barrier operation is twice as large as that of the nonround operations. However, the latency of the first round, in the proposed scheme, is still much less than that of the scheme in [4].

Fig. 12 shows that the well-known star-based barrier schemes for networks with arbitrary topologies perform worse than the proposed barrier scheme without considering the initialization overhead. Tree-based routing may cause traffic convergence in networks with arbitrary topologies [13]. Adaptive routing is able to decrease the traffic convergence [11]. It is possible that, due to random barrier message collisions, several barrier messages destined for different nodes converge in certain nodes simultaneously. In the worst case, they may request the same output link from a node. Due to the limited length of input buffer in the switch, the switch may be blocked quickly. We assume that the members in the barrier group reach the barrier at the same time. The traffic for barrier synchronization is in bursts. The input buffers of switches in some nodes are quickly filled. This effect propagates back to form a saturation tree. When the network size is large and the number of members is also large (i.e., the number of members is greater than half of the network size), saturation may happen. For example, in Fig. 12c and d, when the number of members in a barrier group exceeds 720, the scheme in [4] enters saturation.

The proposed scheme does not enter saturation when the number of members is greater than 720, as shown in Fig. 12. Based on the flow of barrier messages in the reduction phase, the proposed scheme establishes the parent-child relations among as many members as possible, greatly decreasing the chances of message collision.

Effect of Startup Overhead. Startup overhead consists of the software startup and routing startup [9]. Routing startup refers to the time for a header flit to route for one hop from the current switch to its destination by using a routing algorithm. Software startup has a big impact on the latency of barrier synchronization. Similar to the schemes in [4], [5], the proposed scheme latency increases gradually as the startup increases, as shown in Fig. 12. Barrier synchronization schemes cannot fully eliminate the effect of software startup.

The proposed scheme can efficiently control the routing startup in case of no fault or the fault can be handled. The curves of the proposed scheme in Fig. 13 are quite flat and fully overlapped. However, the latency of the scheme in [4] is dependent on Δ. It suffers performance degradation of barrier synchronization when the underlying network is in congestion. The overhead for a flit to cross a congested switch is independent of the duration of congestion in one switch. In the simulation, it takes six cycles for a padding message to arrive at the neighboring switch of the current switch, two cycles for a flit to cross a physical link, two

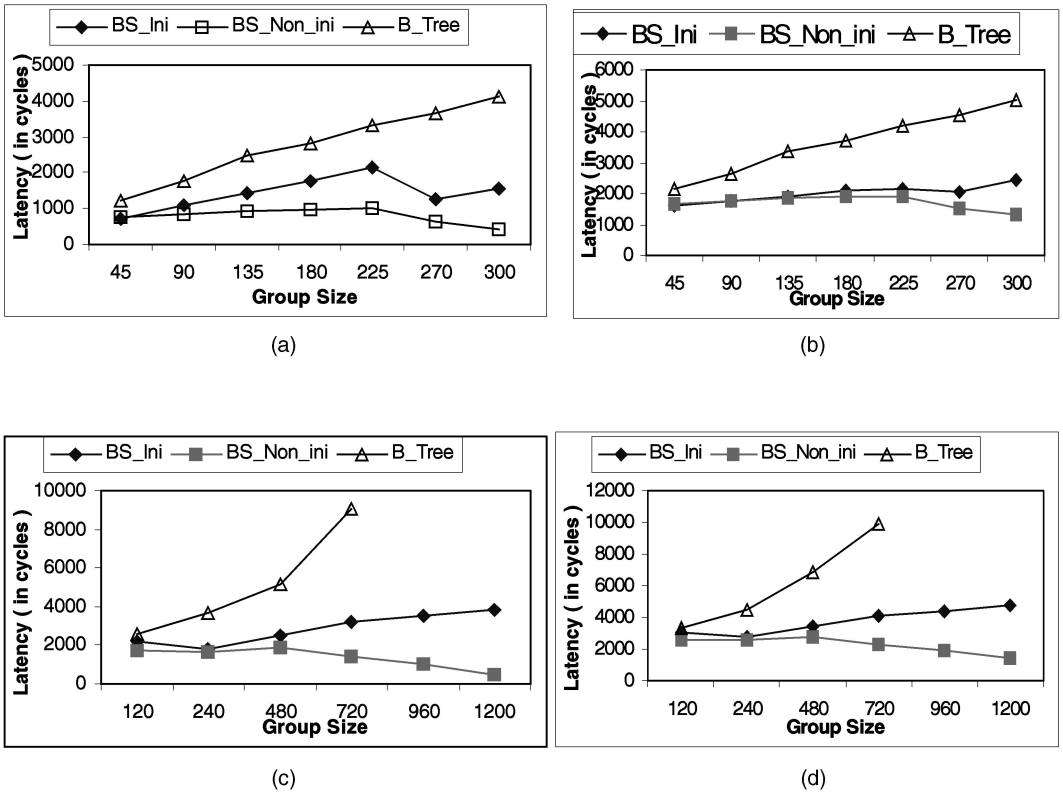


Fig. 12. Effects of startup and system size on latency of barrier synchronization. (a) Ts = 100 cycles, N = 300. (b) Ts = 1,000 cycles, N = 300. (c) Ts = 100 cycles, N = 1,200. (d) Ts = 1,000 cycles, N = 1,200.

cycles for a flit to cross the two crossbars, and two cycles for the destination barrier unit to preempt the specified virtual channel. Note that unicast routing for the two crossbars is not required because the barrier message is preempted by an established virtual channel.

Effect of Congestion in a Barrier Group. The effect of bursty traffic in barrier synchronization can be measured as the amount of participants congested simultaneously for an identical interval in a barrier group.

Fig. 13 illustrates the congestion impact on the scheme in [4] and the proposed scheme. We only consider congestion on barrier members. Different experiments were performed for the number of barrier members, the number of congested barrier members, and the congestion duration. When the duration of congestion is small, i.e., ten cycles, the two schemes show little change in the latencies. However, when the congestion duration increases gradually, the latencies of the two schemes have completely different tendencies. The proposed scheme displays a more graceful degradation in performance than the scheme in [4] in the presence of congestion. In Fig. 13, the noninitialization barrier operations are simulated. The two schemes are not sensitive to change in the number of congested members. Instead, they are sensitive to the duration of congestion. The latency of the scheme in [4] changes dramatically with the variation of congestion duration. On the other hand, the latency of our proposed scheme remains almost constant.

The proposed scheme can almost eliminate the impact of congestion from data messages by barrier synchronization in using the bandwidth-preempting technique. The latency of a barrier synchronization is determined only by the maximum number of hops between the center member and the leaf member in the barrier tree, which is independent of congestion from data messages. Thus, according to the

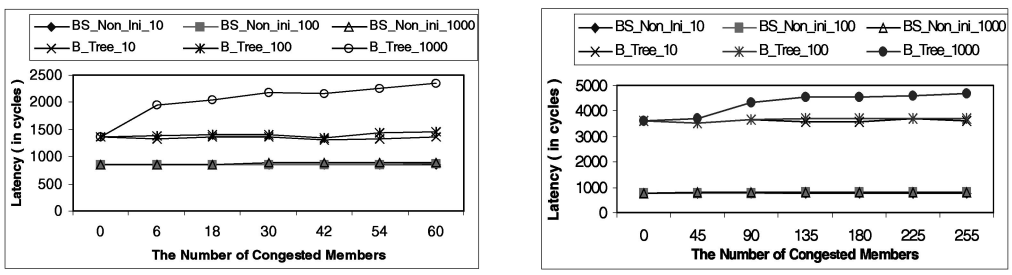
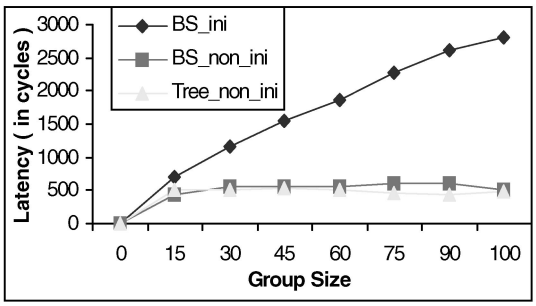
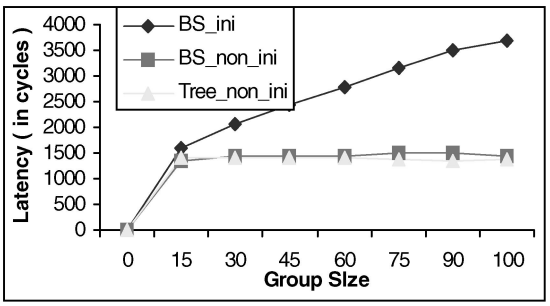


Fig. 13. Effect of congestion in an irregular network with 300 nodes in case of noninitialization turns with Ts = 100 cycles. (a) A barrier group with 60 members. (b) A barrier group with 255 members.

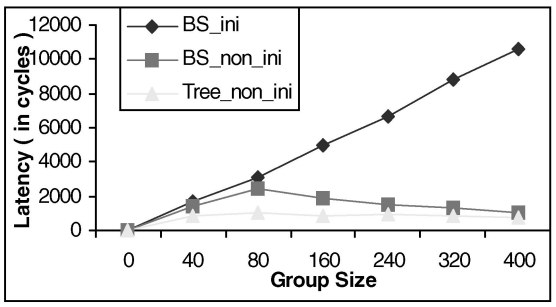


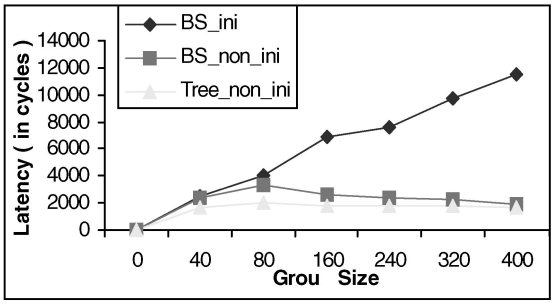
Num. of members	15	30	45	60	75	90	100
Calculate barrier trees (us) in [9]	267	920	1514	2628	3963	5792	7649

(a)



(b)





Num. of members	40	80	160	240	320	400
Calculate barrier trees (us) in [9]	3181	11,727	45,040	102,035	149,083	21,849

(c)

(d)

Fig. 14. Effects of startup and system size on latency of barrier synchronization. (a) Ts = 100 cycles. (b) Ts = 1,000 cycles. (c) Ts = 100 cycles. (d) Ts = 1,000 cycles.

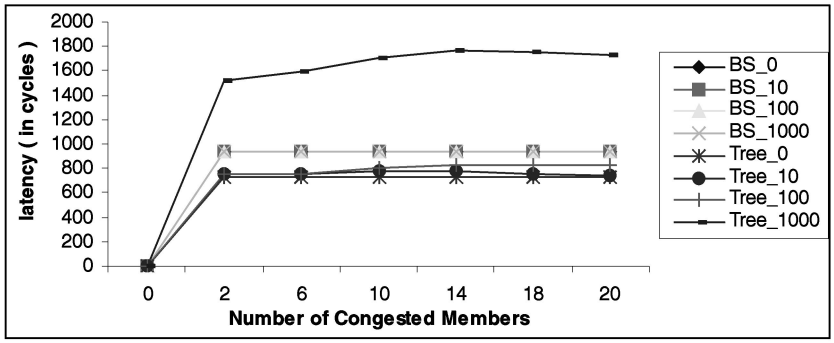
relative locations of members in a barrier group, the latency for the barrier operation can be effectively controlled in our barrier scheme.

4.3 Simulation on 2D Mesh Networks

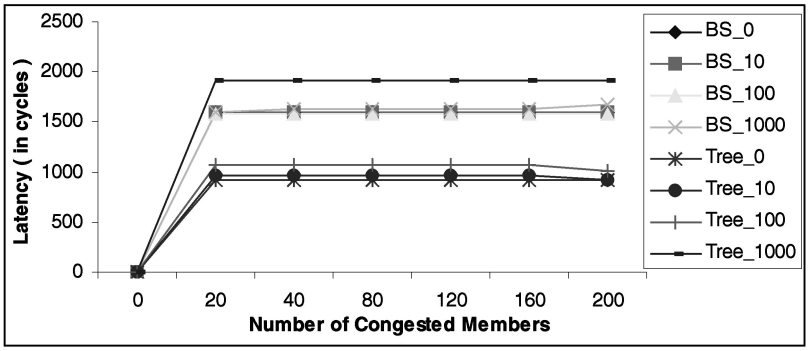
As a special case, a simulator is also developed to run our first scheme and the scheme in [5] for 2D mesh networks. A simulation was performed to evaluate the performances of the two schemes for a range of system size, software startup overhead, the member number for a barrier group, the number of congested members, and the congestion interval. The members in a barrier group are again selected by a random number generator. The values of the various parameters used in the experiments are the same as those shown in Table 1. The cycle time is assumed to be 10 ns. In the simulation, we consider the systems with 100 and 400 nodes, and $ts = 100$ and $ts = 1,000$ cycles. All simulation assumptions and environments are the same as those described before for networks with arbitrary topologies. The detailed simulation implementation and analysis for 2D meshes can be found in [20].

Initialization Overhead. We run the simulation program on an SGI Power Challenge which is a parallel machine with 8 nodes. The interaction of many different concurrent programs on the machine may complicate the measurement for the exact overhead in building the barrier tree. Compared to the initialization in [5], the initialization overhead in our scheme is much smaller. Fig. 9 shows the initialization overheads for the tree algorithm in [5], denoted by *Tree*, and our scheme, denoted by BS, in systems with 100 and 400 processors, respectively. The fast increase in the initialization overhead in our scheme results from the initialization reduction phase. In a 2D mesh, the effect of the center node as a bottleneck is significant, because of the small node degree, and each member sends a reduction message to the center node. However, the cost is justified because updating global network information is a time-consuming task. In the simulation, the deadlock-free X-Y routing in the 2D mesh is applied for simple implementation of the simulator. After the initialization, the proposed scheme has the performance approaching that in [5], as shown in Fig. 14.

Effect of Startup Overhead. Software startup has a large impact on the latency of barrier synchronization. Similar to the schemes in [4] and [5], the latency of the proposed scheme goes up in a stable way when the startup latency increases, as shown in Fig. 14. Similarly, the routing startup on 2D meshes can be effectively controlled in our scheme. However, the schemes in [5] have to incur a heavy loss in performance when congestion occurs.



(a)



(b)

Fig. 15. Effect of congestion at 20×20 mesh in case of noninitialization turns with Ts = 100 cycles. (a) A barrier group with 20 members. (b) A barrier group with 200 members.

Effect of Congestion in a Barrier Group. Fig. 15 describes the response to congestion from the scheme in [5] and our scheme. Without loss of generality, we again only consider the congestion in barrier members. Different experiments were performed for a range of the number of barrier members, the number of congested barrier members, and the congestion duration. We assume that all congested members have the same congestion duration. The congestion members are selected randomly in a barrier group. When the congestion duration is small, i.e., ten cycles, the two schemes show a little change in latency. However, when the congestion duration increases gradually, the latencies of the two schemes show completely different tendencies. The proposed scheme displays a much better stable performance than that of the scheme in [5] in the congestion state. Note that, in Fig. 15, we only consider the noninitialization barrier operations. The two schemes are not sensitive to the changes in the number of congested members. Instead, they are sensitive to the congestion duration. The latency of the scheme in [5] increases dramatically in the congestion duration. On the other hand, the latency of our scheme remains almost constant.

5 Conclusions

In this paper, we have proposed an efficient barrier synchronization scheme on networks with arbitrary topologies. First, we have shown how to build a BSR tree to reduce the traffic of a barrier operation. In our BSR tree method, a barrier message's header can only contain one destination address that is the same as the data message header. After the initialization, the established BSR tree can be used repeatedly for routing barrier messages adaptively. Furthermore, an intermediate node in the BSR tree can work just as usual as for date messages. A new technique, called the *bandwidth-preempting* technique, is then proposed for a blocked barrier message to preempt a channel occupied by a data message, so that the latency of a barrier message can be controlled without affecting much of the overall system performance. We have proposed three switch architecture schemes with differing complexity and performance to hardware support for the BSR tree and bandwidth-preempting function. Finally, we have also proposed an analytical performance model and presented simulation results for the performance evaluation of the proposed barrier synchronization scheme. Performance evaluations show that the proposed scheme outperforms the existing algorithms for barrier synchronization.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments and suggestions that helped us to improve the quality of this paper. Partial funding for this research was provided by Hong Kong RGC 7048/98E and HKU CRCG grant 337/062/0012.

REFERENCES

[1] J.M. Mellor-Crummery and M.L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multicomputer," *ACM Trans. Computer Systems,* vol. 9, no. 1, pp. 21-65, Feb. 1991.

[2] M. Kandemir, N. Shenoy, P. Banerjee, J. Ramanujam, and A. Choudhary, "Minimizing Data and Synchronization Costs in One-Way Communication," *Proc. Int'l Conf. Parallel Processing,* 1998.

[3] J.-S. Kim, S. Ha, and C.S. Jhon, "Efficient Barrier Synchronization Mechanism for the BSP Model on Message-Passing Architecture," *Proc. Int'l Parallel Processing Symp.,* Mar. 1998.

[4] R. Sivaram, C.B. Stunkel, and D.K. Panda, "A Reliable Hardware Barrier Synchronization Scheme," *Proc. Int'l Parallel Processing Symp.,* 1997.

[5] J.-S. Yang and C.-T. King, "Designing Tree-Based Barrier Synchronization on 2D Mesh Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 6, pp. 526-534, June 1998.

[6] H. Xu, P.K. McKinley, and L.M. Ni, "Efficient Implementation of Barrier Synchronization in Wormhole-Routed Hypercube Multicomputer," *J. Parallel and Distributed Computing,* vol. 16, pp. 172-184, Oct. 1992.

[7] K.B. Fan and C.-T. King, "Turn Grouping for Efficient Barrier Synchronization in Wormhole Mesh Networks," *Proc. 25th Int'l Conf. Parallel Processing,* Aug. 1997.

[8] D.K. Panda, "Fast Barrier Synchronization Wormhole k-ary n-cube Networks with Multi-Destination Worms," *Proc. Int'l Symp. High Performance Computer Architecture,* pp. 200-209, 1995.

[9] R.E. Kessler and J.L. Schwarzmeier, "Cray T3D: A New Dimension for Cray Research," *Proc. 38th IEEE Int'l Computer Conf.,* pp. 176-182, Spring 1993.

[10] S.L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Architecture Support Programming Languages and Operating Systems,* Sept. 1996.

[11] R. Kesavan, K. Bondalapati, and D.K. Panda, "Multicast on Irregular Switch-Based Networks with Wormhole Routing," *Proc. Third Int'l Symp. High Performance Computer Architecture (HPCA-3),* pp. 48-57, Feb. 1997.

[12] J. Protic, M. Tomasevic, and V. Milutinovic, *Distributed Shared Memory Concepts and Systems.* Los Alamitos, Calif.: IEEE Computer Society, 1998.

[13] M.D. Schroeder et al. "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE J. Selected Areas in Comm.,* vol. 9, no. 8, Oct. 1991.

[14] X. Lin, P.K. McKingley, and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2D Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 8, pp. 793-804, Aug. 1994.

[15] J.H. Kim, Z. Liu, and A.A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 3, pp. 229-244, Mar. 1997.

[16] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 2, pp. 194-205, 1992.

[17] J. Duato, "A Necessary and Sufficient Condition for Deadlock Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, pp. 1055-1067, 1995.

[18] Y.-J. Tsai, Y. Huang, and P.K. McKinley, "Performance Evaluation of Barrier Synchronization in ATM Network," *Proc. Int'l Conf. Computer Comm. and Networks,* Oct. 1996.

[19] F. Silla, M.P. Malumbres, J. Duato, D. Dai, and D.K. Panda, "Impact of Adaptive on the Behavior of Networks of Workstations Under Bursty Traffic," *Proc. Int'l Conf. Parallel Processing,* 1998.

[20] Y. Sun, P.Y.S. Cheung, and X. Lin, "Bandwidth-Preempting Barrier Synchronization on Wormhole-Routed 2D Networks," Technical Report HKU-EEE-99-002, 1999.

Yuzhong Sun received the PhD degree in computer engineering at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1997. This work was done when he was a research fellow in the Department of Electrical and Electronic Engineering at the University of Hong Kong. Recently, he has worked on multimedia servers. His research interests include interconnection networking, architectures for parallel and distributed computation, cluster computing, parallel algorithms, and RAID. He is a member of the IEEE Computer Society.

Paul Y.S. Cheung received the BSc(Eng) degree with first class honors and the PhD degree from the Imperial College of Science and Technology, University of London in 1973 and 1978, respectively. He joined the University of Hong Kong in 1980 and served as associate dean of engineering from 1991-1994 and dean of faculty of engineering from 1994-2000. He was the Asia Pacific Region Director of IEEE in 1995-96 and the Institute's Secretary of IEEE in 1997. He was a director of Pacific Century CyberWorks from 1999-2000. He is also a Director of Versitech, a technology transfer company of the University of Hong Kong, and a director of the Information Technology Entrepreneur Association, a nonprofit organization to promote I.T. entrepreneurs. Since October 2000, he has been with Pacific Century CyberWorks as senior vice-president in technology. His research interests include parallel computer architecture, Internet computing, VLSI design, signal processing, and pattern recognition. He is a senior member of the IEEE

Xiaola Lin received the BS and MS degrees in computer science from Peking University, Beijing, China, and the PhD degree in computer science from Michigan State University, East Lansing, in 1992. He is currently with the Department of Electric and Electronic Engineering, the University of Hong Kong. His research interests include parallel and distributed computing, design and analysis of algorithms, and high speed computer network.