Fast Barrier Synchronization in Wormhole k-ary n-cube Networks with Multidestination Worms[1]

Dhabaleswar K. Panda

Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Tel: (614)-292-5199, Fax: (614)-292-2911
E-mail: panda@cis.ohio-state.edu

Abstract: This paper presents a new approach to implement fast barrier synchronization in wormhole k-ary n-cubes. The novelty lies in using *multidestination* messages instead of the traditional single destination messages. Two different multidestination worm types, *gather* and *broadcasting*, are introduced to implement the report and wake-up phases of barrier synchronization, respectively. Algorithms for complete and arbitrary set barrier synchronization are presented using these new worms. It is shown that complete barrier synchronization in a k-ary n-cube system with e-cube routing can be implemented with $2n$ communication start-ups as compared to $2n \log_2 k$ start-ups needed with unicast-based message passing. This leads to an asymptotic improvement by a factor of $log_2 k$. Simulation results for different system and architectural parameters indicate that the new framework can reduce barrier synchronization cost considerably compared to the unicast-based scheme. For arbitrary set barrier, an interesting trend is observed where the synchronization cost keeps on *reducing* beyond a certain number of participating nodes. The framework demonstrates potential for supporting fast barrier synchronization in large wormhole-routed systems.

Keywords: Barrier Synchronization, Collective Communication, Wormhole Routing, Virtual Cut-through, Broadcasting, Multicasting, Meshes, k-ary n-cubes, and Path-based Routing.

[1]This research is supported in part by the National Science Foundation Grant # MIP-9309627. Accepted to be published in *Future Generation Computer Systems* Journal, a special issue on *High Performance Computer Architecture* consisting of best 10 papers from HPCA conference. A preliminary version of this paper has appeared in *Proceedings of International Symposium on High-Performance Computer Architecture* (HPCA '95), pp. 200-209.

Contents

1 Introduction

The wormhole-routing switching technique is becoming the trend in building future parallel systems due to its inherent advantages like low-latency communication and reduced communication hardware overhead [14]. Intel Paragon, Cray T3D, Ncube, J-Machine, and Stanford DASH are representative systems falling into this category. Such systems are being used for supporting either distributed-memory, shared-memory, or distributed-shared memory programming paradigms. In order to support these paradigms, the systems need fast implementation of barrier synchronization [13]. Reduced overhead for barrier synchronization alleviates the sequential bottleneck in a parallel program leading to better speedup. It also provides flexibility to achieve better overlapping of computation with synchronization [10].

Many hardware, software, and hybrid schemes have been proposed in the literature to efficiently implement barrier synchronization on shared-memory multiprocessors [1, 15] with bus and multistage interconnections. Topology- and routing-specific synchronization schemes have been proposed in [8, 19]. However, very little research has been done to implement fast barrier synchronization on popular k-ary n-cube networks with wormhole routing.

One easier way to implement barrier on a k-ary n-cube network is to use *software* message-passing. Traditionally, the wormhole-routed systems have supported only *point-to-point* (unicast) message passing mechanism [14]. This mechanism allows a message to have only a single destination. Using unicast-based *send* and *receive* message passing primitives, barrier synchronization can easily be achieved using a two-phase procedure: *report* and *wakeup* [20]. The processors *report* their respective arrival at a barrier synchronization point through steps of upward tree communication. The root node, after gathering the arrival information from all processors, broadcasts it during the *wakeup* phase. This phase involves several downward tree communication steps. This approach can be used to support *complete* (all processors participating) or *arbitrary* (only a fraction of processors participating) barrier synchronization. Either single barrier or multiple concurrent barriers can be supported in a system using this software approach.

For a k-ary n-cube system with k^n processors, this scheme requires $2n \log_2 k$ communication steps. Since synchronization messages do not need any data flits, the cost for each of the above communication steps for wormhole-routed systems can be approximated to $(t_s + d \times t_p)$. The component t_s is the communication startup time, t_p is the propagation time per hop, and d is the distance between the source and destination processors in the communication step. In current generation machines, t_s is around 1.0–35.0 microseconds while t_p is in the range of 5.0–15.0 nanoseconds. Hence, the cost of a communication step is dominated by t_s and the cost of any software-based synchronization scheme becomes proportional to the number of communication steps involved. This makes the barrier synchronization cost quite high. For example, with 1.0 microsec communication start-up time it will take 200.0 microsec to barrier synchronize 1K processors in a 32×32 system. It can be easily seen that such a software approach leads to high synchronization latency.

To alleviate such high latency, systems like Cray T3D and CM-5 use dedicated tree-based *synchronization networks* with barrier registers to provide fast synchronization. For k^n processors, such a tree consists of $n \log_2 k$ stages. For supporting m concurrent barriers, the tree requires m-bit wide links both in upward and downward directions. Considering a 10.0 microsec communication start-up time and 25 nanosec as the hardware delay per each stage of this tree, it will take only 10.5 microsec to barrier synchronize 1K processors in a 32×32 system. Such an approach provides very fast synchronization compared to the software-based scheme. However, it is not scalable. As the system size increases, more number of tree-levels are required. Similarly, more wires are needed to support additional concurrent barriers. The pipelining technique can be used to support multiple concurrent barriers over a synchronization tree with 1-bit wide links. However, this increases the overall synchronization latency. This organization also requires multiple registers at the intermediate nodes of the tree to distinguish between the barriers. Overall the hardware approach provides limitation for a given system to scale in size unless special care (expandability) has been taken during the design phase of the system.

The above comparison between hardware and software approaches raises a question whether fast barrier synchronization can be implemented on direct networks using message passing with architectural supports associated to each router. This will alleviate the need for a separate synchronization network and will provide easy scalability as the system size grows. With respect to message passing-based synchronization solutions, it can be seen that the above two-phase procedure has remain unchanged since the first generation of multicomputers were developed using packet-switched interconnection. As the switching technology has advanced towards the third generation wormhole-routed switching, the challenge leads to whether any intelligent synchronization mechanism blending well with the switching technique can be devised. The communication start-up time in current generation systems is also gradually reducing. Hence, if a mechanism can be developed to achieve barrier synchronization with less number of communication start-ups then the overall cost of barrier synchronization will be low and the future systems need not support a separate synchronization network.

In this paper, we take such a challenge and propose a fast, scalable, and cost-effective synchronization scheme for systems using wormhole direct networks. Our solution is based on using *multidestination* wormhole mechanism which was introduced by us recently [18]. We propose using two new worm types, *gather* and *broadcasting*, to implement fast barrier synchronization. These worms propagate through paths conformed to the base-routing of the system [18]. A special bit-string encoding of multidestination addresses [3] is used by these worms. This limits the size of these worms to only a few flits depending on the system size and the channel width. Deadlock-free propagation of these worms over the *data network* of a system is guaranteed using virtual cut-through technique and *minimal* architectural support at the router and processor-router interface. We present algorithms for both complete and arbitrary subset barrier synchronization with one process being mapped to a processor. It is shown that only a few communication steps are sufficient to implement barrier synchronization using the multidestination worms. The algorithms are evaluated through simulation experiments for different system size, topology, communication start-up time, and node delay. The results indicate that the multidestination approach is capable of implementing fast and scalable barrier synchronization compared to the unicast-based scheme.

The paper is organized as follows. An overview of the multidestination mechanism is presented in section 2. Two new worm types are introduced and used in section 3 to implement barrier synchronization on a linear array. In section 4, we develop algorithms for complete and arbitrary-set barrier synchronization on k-ary n-cube. Simulation experiments and results are presented in section 5. Finally, we conclude the paper with future research directions.

2 Overview of Multidestination Mechanism

In this section, we summarize wormhole message passing mechanism with multiple destinations [18]. Intrinsic benefits of this mechanism over unicast message passing are emphasized.

2.1 The Mechanism

A single destination wormhole message consists of a body and a header with its destination number. For a multidestination message, the header consists of multiple destinations and can span multiple flits. The *sender* node creates the list of destinations as an ordered list, depending on their intended order of traversal, and incorporate it into the header. There are different ways to encode these addresses [3] and these worms can have different functionality. To illustrate the concept, in this section, we use *all-destination* encoding format with broadcast/multicast functionality. In the next section, we propose using *bit-string* encoding and introduce alternative multidestination worm type like *gather*. It is to be noted that multidestination mechanism was first introduced by Ni [12]. However, it was proposed for multicasting/broadcasting in Hamiltonian path-based wormhole networks. In [18], we have proposed a generalized multidestination mechanism which can conform to the routing scheme of any wormhole network. We explain this concept later in this section. In this paper, in addition to multicast/broadcast operations, we show how to use multidestination mechanism for *gather/reduction* operations.

Let us first see the movement of a multidestination multicast/broadcast worm. Once such a worm is injected into the network, it is routed based on the address in the leading header flit corresponding to the first destination. As soon as the worm reaches the router of the first destination node, the flit containing this address is removed by the router. Now the worm is routed to the node whose address is contained in the next header flit (second destination in the ordered list). While the flits are being forwarded by the router of the first destination node to its adjacent router, they are also copied (consumed) flit-by-flit to the system buffer by the router-processor interface of the first destination. This process is carried out in each intermediate destination node of the ordered list. When the message reaches the last destination, it is not routed any further and gets completely consumed by the last destination node.

Figure 1 illustrates an example of this mechanism with 3 neighboring destinations. A typical multidestination worm format with all-destination encoding is shown in Fig. 1a. Figure 1b shows a snapshot of the system when flit A of the message has reached the router of node d3. It is assumed that the header flits have been appropriately consumed at the respective nodes. Figure 1c shows the snapshot when the worm moves ahead by one flit. Each flit is copied to the system buffer of the associated node and also propagated ahead to the next node. It is to be noted that such a mechanism is quite powerful and can deliver a message to multiple destinations much faster than using multiple unicast messages. This multidestination scheme is also quite general in the sense that unicast messages can always be implemented under it as a subset operation with only one destination.

2.2 Paths Conforming to Base Routing

The list of destinations in a multidestination worm is ordered by the sender node. This *ordering* depends on two parameters: 1) the *intended traversal* of the message to a set of destinations and

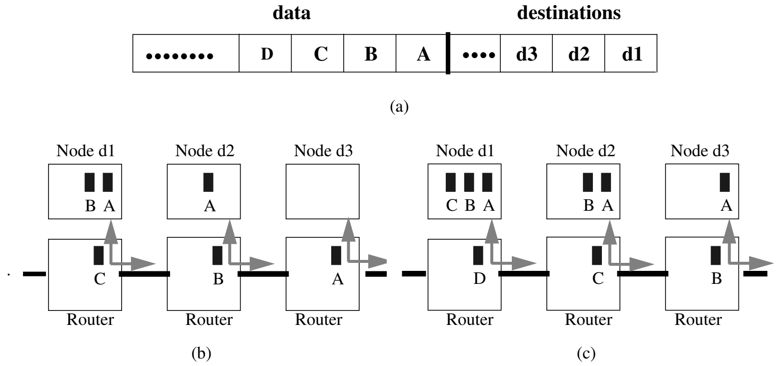


Figure 1: Multidestination wormhole mechanism with a multicast/broadcast worm: (a) a typical format with all destination encoding, (b) snapshot when flit A of the worm has reached the router of node d3 and (c) snapshot at the next flit movement.

2) the *feasibility* of the intended path to be traversed by the worm in the network. Obviously, the first parameter depends on the application and the second one is determined by the routing constraint of the underlying network. Consider a multidestination worm with n destinations from a source s with an ordered destination list $\{d_1, d_2, \ldots, d_{n-1}, d_n\}$ (the first and the last destinations are d_1 and d_n, respectively). The validity of this worm depends on the underlying routing scheme. Assume the network supports a deadlock-free routing scheme $\mathcal{R}$ which can be e-cube [6], planar-adaptive [4], turn-model [9], fully-adaptive [7], or any other routing scheme. The Base-Routing-Conformed-Path (BRCP) model, as defined in [18], is as follows:

Definition 1 *A multidestination worm with an ordered destination list* $\{d_1, d_2, \ldots, d_{n-1}, d_n\}$ *in a network supporting routing* $\mathcal{R}$ *conforms to this base routing iff the destination set* $\{d_1, d_2, \ldots, d_{n-1}\}$ *can be covered as intermediate nodes on one of the possible paths from* s *to* d_n *under the routing constraint* $\mathcal{R}$.

Figure 2 shows examples of multidestination worms under the BRCP model. For example, in an ecube system (assuming messages are routed first along row and then along column) [6], a multidestination worm can cover a set of destinations in row/column/row-column order. It is to be noted that a set of destinations ordered in a column-row manner will be an *invalid* path under the BRCP model for e-cube systems. Similarly, in a planar adaptive system [4], a multidestination worm can cover a set of destinations along any diagonal in addition to the flexibility supported by the ecube system. If the underlying routing scheme supports non-minimal west first turn model [9], it can provide further flexibility in covering a lot of destinations using a single worm. Hence, this model is quite general and can be used by any routing scheme. A similar but restricted method of using only row-path and column-path in e-cube meshes has also been independently proposed by Boppana, Chalasani, and Raghavendra [2].

2.3 Intrinsic Benefits

The significant benefit of the BRCP model comes from the fact that a message can be delivered to multiple destinations with the same overhead as that of sending it to a single destination, if the destinations can be grouped into a single worm under the BRCP model. Even the simplest e-cube routing scheme can take advantage of this model by grouping destinations on a

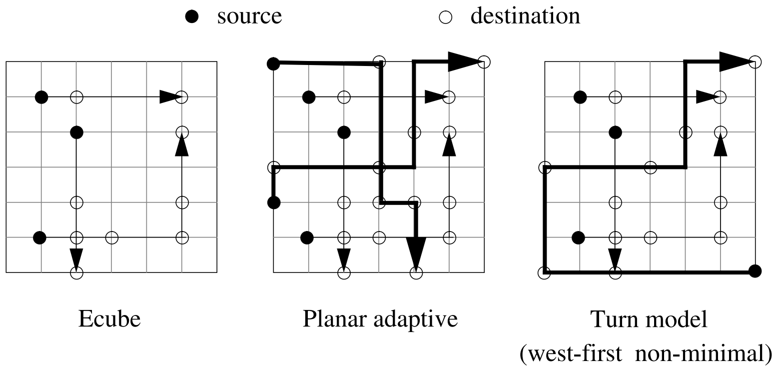


Figure 2: Examples of multidestination worms under the BRCP model conforming to different base routing schemes in a 2-D mesh. In addition to ecube paths, added flexibility for new paths under planar and west first routing scheme are shown in bold lines.

row/column/row-column. As the adaptivity of the base routing increases, more and more destinations can be covered by a single multidestination worm. As the number of destinations per source *increases* in operations like multicast and broadcast, *less* number of multidestination worms are needed to cover them compared to the unicast-based schemes [18]. If the set of destinations demonstrate some spatial organization (row, column, diagonal, or block) then a suitable scheme based on the underlying routing constraint can also be developed to cover them using as few multidestination worms as possible. Similarly, depending on the technological parameters (the ratio of communication start-up time to link-propagation time), a programmer/compiler can optimize on the grouping to implement the required communication operation with minimal time. Hence, this model opens up an entirely new framework together with its challenges for developing optimal algorithms to implement collective communication operations [13] with minimal time. In this paper, we emphasize on this framework by developing and evaluating algorithms for barrier synchronization which is an important collective communication operation.

3 Barrier Synchronization on a Linear Array

In this section, we introduce a new multidestination gather worm type. We demonstrate how this gather worm together with broadcast worm, discussed in the last section, can be used to implement fast barrier synchronization on a linear array. The necessary architectural modification to the processor-router interface logic to support movement of gather worm is presented. The associated performance benefits and design considerations under the current technology are discussed.

3.1 Synchronization with Gather and Broadcast Worms

We propose a scheme using a pair of multidestination gather and broadcast worms to implement fast barrier synchronization across processors on a linear array. The operation consists of two phases involving only two communication start-ups. The first phase implements *reporting* by using a gather worm. The second phase implements *wake-up* through a broadcast worm.

Consider a linear array of 6 processors as shown in Fig.3. Assume 4 processors (P0, P1, P2, and P4) participating in a barrier. The rightmost processor P4, after reaching its barrier point, can initiate a multidestination *gather* worm. The header of this worm consists an ordered destination list (P2, P1, and P0) with P0 as the final destination and P2, and P1 as intermediate destinations. As this worm propagates towards P0, it can *gather* information from processors P2 and P1 whether they have reached at the barrier or not. Assuming suitable architectural support is available at a router interface to provide information whether its associated processor has arrived at the barrier or not, a gather worm can collect such information on-the-fly without being consumed at intermediate destinations. As a gather worm propagates, it checks for the arrival information at every router interface of intermediate destinations. If the intermediate destination has already arrived at the barrier, the worm proceeds ahead. Otherwise, it gets blocked at the respective router interface until the associated processor arrives at the barrier and provides information to the router interface. If a processor is not participating in a barrier, the worm moves ahead unconditionally at its router interface. Such a worm gets consumed only at its final destination.

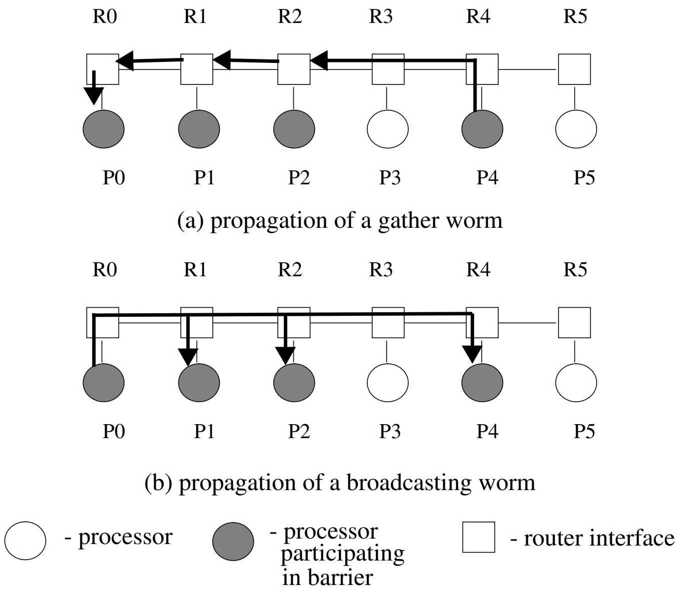
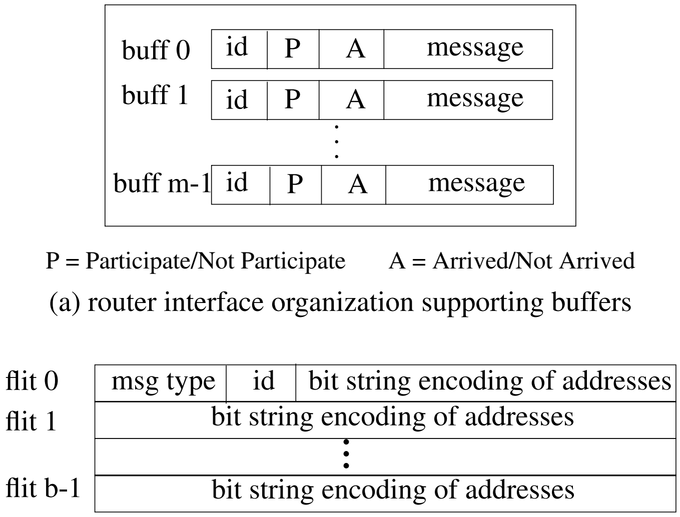


Figure 3: Two phase implementation of a barrier on a linear array: (a) propagation of a gather worm and (b) propagation of a broadcast worm.

In our example, the gather worm from P4, during its propagation, checks the arrival of processors P2 and P1 at the barrier and finally gets consumed at P0. With the consumption of the above gather worm, processor P0 has information that all other participating processors (P1, P2, and P4) have arrived at the barrier. Now to implement the wake-up phase, P0 initiates a *broadcast* worm with (P1 and P2) as intermediate destinations and P4 as the final destination. The propagation of this worm, as shown in Fig. 3b, is unconditional and similar to the movement of a multidestination multicast/broadcast worm as shown in Fig. 1. As this broadcast worm passes through the routers of the intermediate participating processors (P1 and P2), the associated router interfaces activate *wake-up* signals to the respective processors. The intermediate processors may be in wait state, busy looping, or doing computation of some other thread to detect this signal. The end processors, P0 and P4, get waken-up by themselves after sending and receiving the broadcast worm, respectively.

3.2 Architectural Supports

In order to implement the above gather worm type, we propose the following architectural supports at the router interface. Similar to the concept of barrier registers [5, 16], a set of buffers are provided at each router interface of the system. Figure 4a shows a typical router interface organization with m such buffers. Each buffer carries a few bits for *synchronization id*, a flag *participate/not participate* (P) to indicate whether the processor is participating at a given synchronization point or not, a flag *arrived/not arrived* (A) to indicate whether the processor has arrived at the barrier or not, and some space for holding an incoming gather worm. These buffers are accessible by the associated processor using memory-mapped I/O references [11]. It is to be noted that by supporting m buffers at every router interface, a system can implement m concurrent barriers for an application at a given point of time.



(a) router interface organization supporting buffers

(b) a typical message format

Figure 4: Architectural supports for implementing multidestination gather/broadcast worm: (a) synchronization buffers at a router interface and (b) a typical message format.

For supporting gather and broadcast worms together with the unicast worm, we propose a message format as shown in Fig. 4b. Each message carries a *msg type* field (2-3 bits) indicating whether it is a gather, broadcast, or unicast worm. The width of *synchronization id* field is a designer's option depending on how many buffers can be incorporated into the router interface. For a multiprogrammed system, some bits of this field should also identify *task id* so that different tasks running on overlapped partition of the system can barrier synchronize without any ambiguity. The destination addresses are encoded as a bit-string as proposed in [3]. For the linear array example being considered, six bits are sufficient to encode the addresses. Assuming processor 0 is identified as bit 0 of this address, the destination bit-string of gather worm initiated by P4 will be 000111. Similarly, the broadcast worm initiated by P0 will have a bit-string of 010110. In the next section, after proposing barrier synchronization algorithms for k-ary

n-cube systems, we analyze design issues (resource requirement, deadlock-freedom, maximum number of buffers that can be supported, etc.) of our scheme by considering the current trend in technology. We will show that, under the current technology, the multidestination addresses can be encoded into a few (1-3) flits.

3.3 Communication Sequence

Let us consider the communication sequence for the processors participating in a barrier using gather and broadcast worms. We assume static barriers [16] so that the processors participating in a barrier are known at compile time and the associated communication sequence can be generated before the program execution. A split-phase synchronization scheme [10, 15] with separate request and report phases is assumed. Figure 5 shows the communication sequence to implement a barrier on a linear array. Before the execution of barrier x, $0 \leq x < m$, each intermediate processor specifies its desire for participating in the barrier by grabbing buffer x at its router interface and making the associated *participating* flag 'on (1)' and *arrived* flag 'off (0)'. These flags are 'off (0)' by default and reflect masking behavior if the associated processor does not participate. The *arrived* flag of buffer x at an intermediate router is turned 'on (1)' when the associated processor reaches its barrier point after computation.

Leftmost Participating Processor	***Intermediate Participating Processor***	***Rightmost Participating Processor***
compute;	compute; P[buff(x)]=1; A[buff(x)]=0; compute;	compute;
arrive at barrier point x;	arrive at barrier point x;	arrive at barrier point x;
receive gather worm from rightmost processor with id=x;	A[buff(x)]=1;	***send*** gather worm to leftmost processor and intermediate destinations with id=x;
send broadcast worm to rightmost processor and intermediate destinations with id=x;	wait for wake-up; P[buff(x)]=0; A[buff(x)]=0;	***receive*** broadcast worm from leftmost processor with id=x;
compute;	compute;	compute;

Figure 5: Communication sequence of processors on a linear array while participating in a barrier using gather and broadcast worms. The barrier is assumed to have a *synchronization id=x*.

The rightmost participating processor initiates a gather worm with *synchronization id=x.* This gather worm, while passing through the router of an intermediate destination, checks for the *participate* and *arrived* flags of the associated buffer x. If the processor is not participating then the worm keeps on moving ahead. If the processor is participating and it has already arrived at the barrier then also the worm proceeds ahead. If the processor is participating and it has not arrived at the barrier then the worm gets blocked at the router interface until the *arrived* flag is set 'on (1)' by the processor. Finally this gather worm is consumed by the leftmost participating processor.

After the gather phase is over, the leftmost processor initiates a broadcast worm to the rightmost processor with intermediate processors as intermediate destinations. This worm, as it passes through the routers of intermediate destinations, wakes up the associated processors. The leftmost processor is done with the synchronization after sending the broadcast worm. Similarly, the rightmost processor is done when it receives the broadcast worm.

3.4 Performance Benefits

It can be seen that the above two-phase scheme just needs two communication steps to barrier synchronize any number of processors on a linear array. Using tree-based approach with unicast messages, the number of communication steps required are $2\lceil \log_2 d \rceil$, where d is the number of processors participating in a barrier. This leads to:

Theorem 1 *An arbitrary set barrier synchronization on a linear array can be implemented in two communication steps using a pair of multidestination gather and broadcast worms. With d processors participating, the multidestination mechanism reduces the number of communication steps by a factor of $\lceil \log_2(d) \rceil$ compared to the unicast-based scheme.*

Besides the communication start-up time, the number of links traversed by messages also contribute to the synchronization latency in both schemes. In section 5, we present simulation results for both schemes by taking into account communication start-up time, node delay, switch delay, system size, and the impact of multidestination encoding.

4 Synchronization on k-ary n-cube

In this section, we formulate barrier synchronization algorithms for k-ary n-cube systems based on the scheme developed for linear array in the last section. We consider both complete and arbitrary set barrier synchronization. The algorithms are illustrated with respect to 2D meshes and then generalized for higher dimensional systems.

4.1 Complete Barrier

Consider a $k \times k$ mesh as shown in Fig. 6. Complete barrier synchronization can be achieved by considering the mesh as a set of linear arrays. A basic scheme using four steps is shown in Fig. 6a. The first step uses gather worm in all rows to collect information towards the first column. The second step uses a gather worm on the first column. At the end of these two steps, the upper-left corner processor has information that all other processors have arrived at the barrier. Now it can initiate a broadcast worm along the first column. During the final step, the nodes in the first column initiate broadcast worms along their respective rows. This requires four communication steps to implement barrier synchronization on a $k \times k$ mesh. For any k-ary n-cube system, similar dimension-wise gather and broadcast can be done to implement barrier synchronization with $2n$ communication steps.

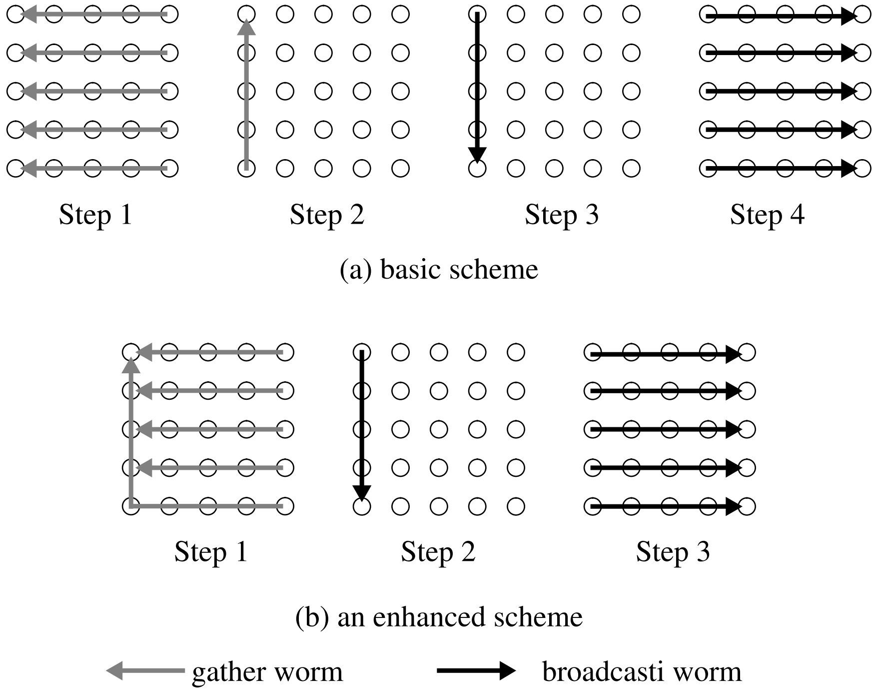


Figure 6: Complete barrier synchronization on a $k \times k$ mesh using (a) four step basic scheme and (b) an enhanced scheme with three steps.

It is to be noted that under this basic scheme, the multidestination gather and broadcast worms need to move along a single dimension of the system only. An enhancement to this basic scheme can be done as shown in Fig. 6b. The gather worm initiated by the bottom row does not stop at the leftmost node of this row but continues along the first column to gather information from the leftmost leaders of all other rows. This will combine the first two gather steps into a single one. Hence, using such *multidimensional* gather worm, complete barrier synchronization can be implemented on a 2D mesh in 3 steps. It is to be noted that such multidimensional worm conforms to the BRCP model discussed in section 2. Figure 7 shows how three dimension-wise gather steps can be combined into a single step on a 4-ary 3-cube system by using multidimensional gather worms. The header of such multidimensional worms is longer in size because they need to carry more destinations in their headers. Though such schemes are feasible, it requires additional logic at a router interface to wait for the arrival of multiple gather worms. Hence, in the remaining part of this paper, we focus and evaluate schemes using only single-dimensional worms in order to provide cost-effective and fast synchronization solutions.

By using unicast-based scheme, the number of communication steps required in a k-ary n-cube system to barrier synchronize k^n processors is $2\lceil\log_2(k^n)\rceil$. This leads to:

Theorem 2 *Complete barrier synchronization across k^n processors on a k-ary n-cube system can be implemented with $2n$ communication steps by using multidestination gather and broadcast worms. Compared to the unicast-based scheme, this leads to an improvement by a factor of $\frac{\lceil\log_2(k^n)\rceil}{n}$.*

An enhanced scheme using multidimensional gather worms reduces the complexity to $(n+1)$

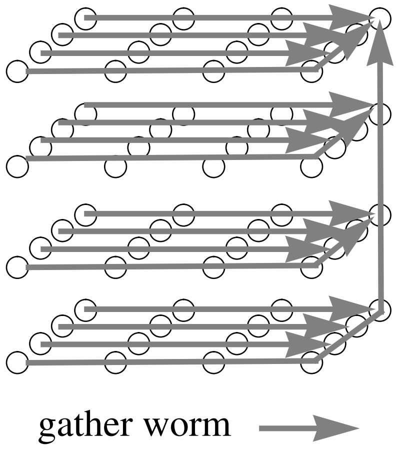


Figure 7: Single step gather operation on a 4-ary 3-cube system using multidimensional gather worms.

communication steps. Compared to the unicast-based scheme, this leads to an improvement by a factor of $\frac{2\lceil \log_2(k^n)\rceil}{(n+1)}$.

4.2 Arbitrary Set Barrier

The necessity of arbitrary set barrier synchronization arises under the following two circumstances: (a) the program is running on a contiguous partition of a system and it needs to barrier synchronize only a subset of processes (processors) or (b) the program is running on a discontiguous partition of a system and it needs to do complete barrier synchronization. In either cases, from the system's perspective, the situation leads to a subset barrier synchronization over a set of processors randomly distributed in the system.

Consider a subset of nodes in a 2D mesh trying to barrier synchronize as shown in Fig. 8. If all processors belong to a single task then this subset barrier can be implemented as a complete barrier by forcing and ordering all processors to participate in each barrier [15]. However, such an assumption is very restrictive. In this section, we propose a general scheme which allows for multiple subset barriers to be executed concurrently in a system and the operation for a given barrier involves only the processors participating in that barrier.

For a 2D mesh, the scheme uses 6 phases as shown in Fig. 8a-f. The first phase consists of using gather worms within rows. For every row having at least two participating processors, a gather worm is initiated by the rightmost participating processor and is consumed by the leftmost participating one. Let us designate the leftmost participating processors as *row leaders*. Now for every column having at least two row leaders, a gather worm can be initiated by the bottom row leader to gather information along that column and finally be consumed by the top row leader. Let these top row leaders be designated as *column leaders*. After these two phases, information needs only to be gathered from the column leaders. It can be easily seen that these column leaders fall into disjoint rows and columns based on the grouping done. Hence no further reduction can be achieved with multidestination worms on systems supporting e-cube routing. If the underlying routing supports adaptivity then further reduction can still be achieved by using adaptive paths as discussed in section 2. In this section, we emphasize on e-cube routing

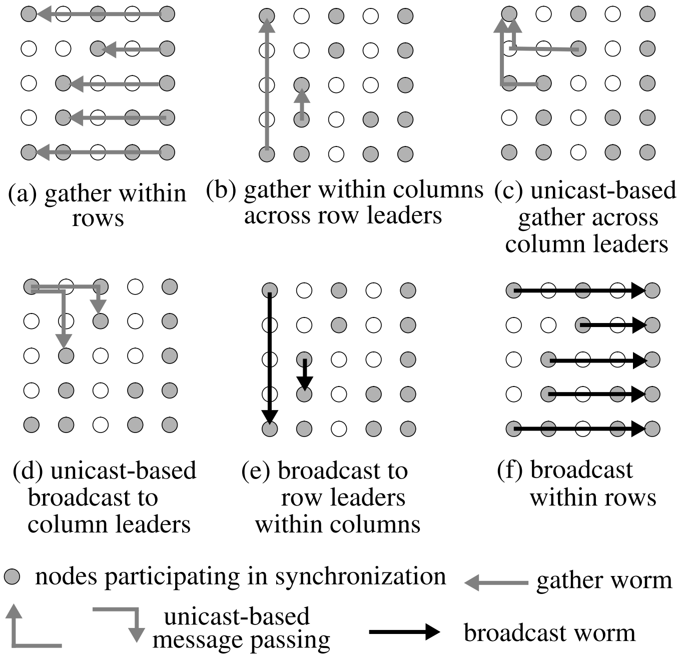


Figure 8: Barrier synchronizing an arbitrary subset of processors in a 2D mesh using multidestination gather and broadcasting worms together with unicast-based message passing.
only.

The third phase consists of unicast-based message passing. This phase uses reverse-umesh algorithm [20] to implement gather among column leaders in a tree-like manner using unicast-based message passing in every step. Once this reduction phase is over, the participating upper-left corner processor has information that all other processors have reached at their respective barriers. This completes the *report* phase of barrier synchronization. Now the *wake-up* phase of barrier synchronization can be achieved by a series of broadcast phases. The fourth phase involves broadcast from upper-left corner processor to column leaders in a tree-like manner using unicast-based umesh algorithm [20]. The fifth phase involves broadcast by column leaders to row leaders using broadcast worms. During the final step, the row leaders use broadcast worms in their respective rows to wake-up the associated processors.

In this six-phase scheme, every phase except the third and the fourth needs only one communication step. The number of communication steps needed in the third and the fourth phases depend on the number of column leaders involved. For an arbitrary set of p processors participating in a barrier, let's define the number of row leaders as L_1 and the number of column leaders as L_2. It can be easily seen that $L_2 \leq L_1 \leq p$. With L_2 column leaders, the reverse-umesh algorithm in the third phase and umesh algorithm in the fourth phase take $\lceil \log_2{(L_2)} \rceil$ communication steps each. It can be seen that as the number of participating nodes increases, more and more of them can be grouped together along rows and columns and it reduces the number of column leaders L_2 [18]. Hence this scheme allows to *reduce* synchronization cost as the number of participating nodes *increases*.

Table 1 illustrates the effect of multidestination gather and broadcast on the number of row and column leaders as well as the number of communication steps required for barrier synchronization. These numbers are derived for a 32x32 mesh and averaged over 40 iterations. It can be observed that as the number of processors participating in a barrier increases, better grouping gets achieved along rows and columns and the number of column leaders reduces drastically. If the set of participating processor is smaller in size then they are located sparsely in the system and they do not reduce the number of column leaders significantly. Hence, this leads to additional (maximum 4) number of communication steps than unicast-based scheme. Keeping this in mind, an optimization to the multidestination scheme is done as follows: the first and sixth phases are eliminated if $(\lceil \log_2(L_1) \rceil + 1) > \log_2 p$. Similarly, the second and fifth phases are eliminated if $(\lceil \log_2(L_2) \rceil + 1) > \lceil \log_2(L_1) \rceil$. This optimization ensures that the multidestination scheme always performs better than the unicast-based scheme. The number of communication steps required by this optimized approach is shown in parenthesis in Table 1. We use this optimized algorithm in the next section to demonstrate performance results through simulation.

Table 1: Average number of row leaders (L_1), column leaders (L_2), and communication steps required with multidestination scheme. The communication steps are compared with the steps required with unicast-based scheme for barrier synchronizing arbitrary set of processors on a 32x32 mesh supporting e-cube routing.

No. of Processors Participating (p)	No. of Row Leaders (L_1)	No. of Column Leaders (L_2)	Communication Steps for Multidestination Scheme $2\lceil \log_2(L_2) \rceil + 4$ (optimized case)	Communication Steps for Unicast-based Scheme $2\lceil \log_2 p \rceil$
16	13	10	12(8)	8
32	20	15	12(10)	10
64	28	17	14(12)	12
96	30	16	12(12)	14
128	31	15	12(12)	14
256	32	10	12(12)	16
384	32	7	10(10)	18
512	32	4	8(8)	18
640	32	4	8(8)	20
768	32	3	8(8)	20
896	32	2	6(6)	20
960	32	1	4(4)	20
1024	32	1	4(4)	20

The scheme proposed for 2D mesh can be generalized to k-ary n-cubes with $2(n+1)$ phases. The first n phases involve dimension-wise gathering of results along dimensions 1 to n. During this gathering process, the scheme produces the associated number of dimension leaders as $L_1, L_2, \ldots, L_n$. The $(n+1)$th phase involves tree-like reverse-umesh gathering among dimension-n leaders (L_n). The $(n+2)$nd phase involves tree-like broadcast to dimension-n leaders. The remaining n phases involve dimension-wise broadcast along dimensions n to 1. This leads to:

Theorem 3 *Arbitrary set barrier synchronization on a k-ary n-cube system can be implemented with $(\lceil \log_2(L_n) \rceil + 2n)$ communication steps by using multidestination scheme, where L_n is the number of dimension-n leaders after first n phases of dimension-wise gathering.*

Using unicast-based scheme, with p processors participating in a barrier, it takes $\lceil \log_2 p \rceil$ communication steps. As p increases, it can be observed that the multidestination scheme uses lesser number of communication steps. For very large p, the multidestination scheme only requires $2n$ steps leading to an asymptotic improvement by a factor of $\log_2 k$.

Here we have mostly emphasized on the number of communication steps. However, the overall synchronization time depends on other parameters like node delay, path length, and link propagation delay. In the next section, we present simulation results by taking all these parameters into account and study the scalability issues of these schemes with respect to different system and technological parameters.

4.3 Design Considerations

Before discussing the simulation results, let us consider the current technology to incorporate such multidestination worm-based synchronization framework for k-ary n-cube systems. Using single-dimensional paths, k bits are sufficient for a multidestination worm to represent its address in a bit-encoded fashion. One more additional flit is required to provide other fields. The current wormhole technology supports channel widths of $16 - 32$ bits. Hence, for a 32x32 system, the size of a multidestination gather/broadcast worm needed for barrier synchronization will be $2 - 3$ flits.

It can be seen that bit-string encoding is very powerful to make the message length quite small. With bit-string encoding, the router interface needs to absorb all flits before making a routing decision to the next destination. Since the message length is quite small, we propose a virtual cut-through approach instead of strict wormhole routing [14]. This alleviates the deadlock problem associated with multidestination gather worms for concurrent and overlapped arbitrary-set barriers. It is to be noted that multidestination broadcast worms may lead to deadlock due to limited number of consumption channels and it can be alleviated by using multiple consumption channels [2, 18]. The *message* field in synchronization buffers, as shown in Fig. 3a, is meant to hold a gather worm during its propagation if it gets blocked. On the current generation systems like IBM SP1/SP2, the router interface has 1K bytes of buffer to support buffered wormhole routing. Hence, the proposed framework can easily be implemented in near-future systems with a reasonable number of buffers at each router interface. With m buffers at each router interface, there will be flexibility to support m concurrent barriers for every program running in disjoint partition of the system.

5 Simulation Experiments and Results

To verify the effectiveness of our proposed schemes, we simulated them on a CSIM-based simulator test bed. This test-bed implements flit-level simulation and can be configured for different topological and technological parameters. We compared our scheme with unicast-based barrier synchronization algorithm [20]. The following parameters, representing the current trend in technology, were assumed: t_s (communication start-up time) as 1.0, 5.0, and 10.0 microseconds and t_p (link propagation time) as 5.0 nsec. For multidestination gather and broadcast worms, we used bit-string encoding with channel width as 32 bits. The node delay (t_{node}) was assumed to be 20.0 nsec for unicast-based message passing. Though the node delay for multidestination worm (using bit manipulation logic) will be similar to that of unicast messages, for completeness in analysis, we evaluated our schemes for three different values of node delay. In the simulation graphs, these values are indicated by 20.0 nsec (1.0x node delay), 30.0 nsec (1.5x node delay), and 40.0 (2.0x node delay). The processors were assumed to be arriving at a barrier simultaneously. For arbitrary set barrier synchronization, we randomly chose the participating processors in a system. Each experiment was repeated 40 times and the average synchronization latency was determined.

5.1 Complete Barrier

We evaluated our complete barrier barrier scheme against unicast-based scheme for the following 2D mesh configurations with e-cube routing: 4x4, 8x4, 8x8, 16x8, 16x16, 32x16, 32x32, 64x32, and 64x64. Evaluations were done for three values of t_s, reflecting the trend in current technology. Three different node delays were also considered. Figure 9 shows the comparison. It can be observed that the multidestination scheme outperforms unicast-based scheme for all system sizes. As the system size grows, the cost of synchronization increases very little for multidestination scheme compared to the unicast-based scheme. The costs for 64x32 and 64x64 configurations are higher because the worm length increases by 1 flit to accommodate bit-encoded addresses with 32-bit channel width. With 1.0 microsecond communication start-up time, the multidestination scheme with 1.0x node delay is able to barrier synchronize 4K processors in just 11.68 microseconds. For higher communication start-up, the cost with multidestination scheme remains practically 'constant' with increase in system size. This shows the effectiveness of multidestination scheme to perform low cost barrier synchronization in a large system in spite of higher communication start-up time.

5.2 Arbitrary Subset Barrier

We evaluated the cost of arbitrary subset barrier synchronization on 32x16, 32x32, and 10x10x10 systems with different number of participating processors. Communication start-up time of 1.0 microsecond with varying node delays for the multidestination scheme were assumed. Figure 10 shows the comparison. It can be seen that as the number of participating processors *increases* beyond certain number, the multidestination scheme is able to implement a barrier with *reduced* cost. This is the most *unique* strength of our proposed scheme. It shows the potential to implement dense barrier synchronization with reduced cost. Such crossover depends on system size, topology, communication start-up time, and node delay. For 1.0x node delay, the crossover was observed at 32 nodes for 32x16 system, 96 nodes for 32x32 system, and 256 nodes for 10x10x10 system. The crossover point is smaller for 2D mesh compared to 3D mesh. With less number of processors participating in a barrier, the multidestination scheme performs little worse due to longer paths traveled by multidestination worms. Beyond the crossover point, the multidestination scheme demonstrates steady reduction in cost over unicast-based scheme. For 75% nodes participating in a barrier, with the technological parameters considered, the multidestination scheme reduces the synchronization cost by a factor of 1.4-2.7 compared to the unicast-based scheme. For higher communication start-up time and system size, the improvement will be higher.

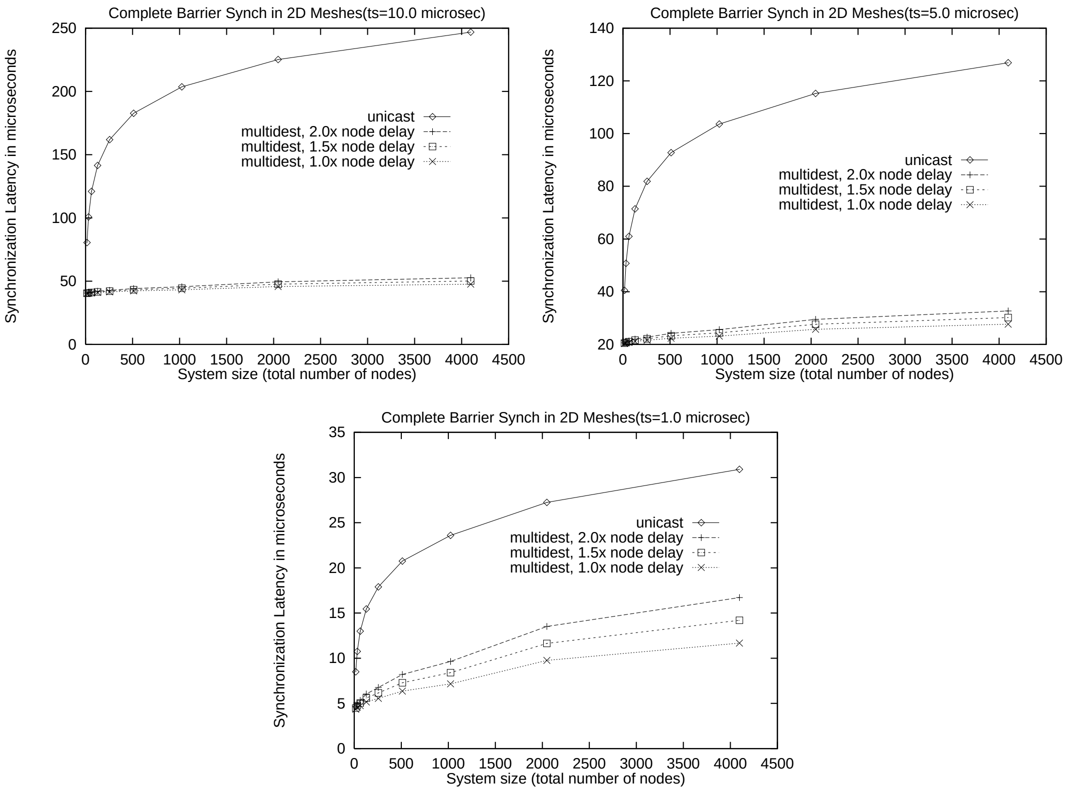


Figure 9: Comparison of unicast-based and multidestination-based schemes to implement complete barrier synchronization in 2D meshes with different communication start-up and node-delay time.

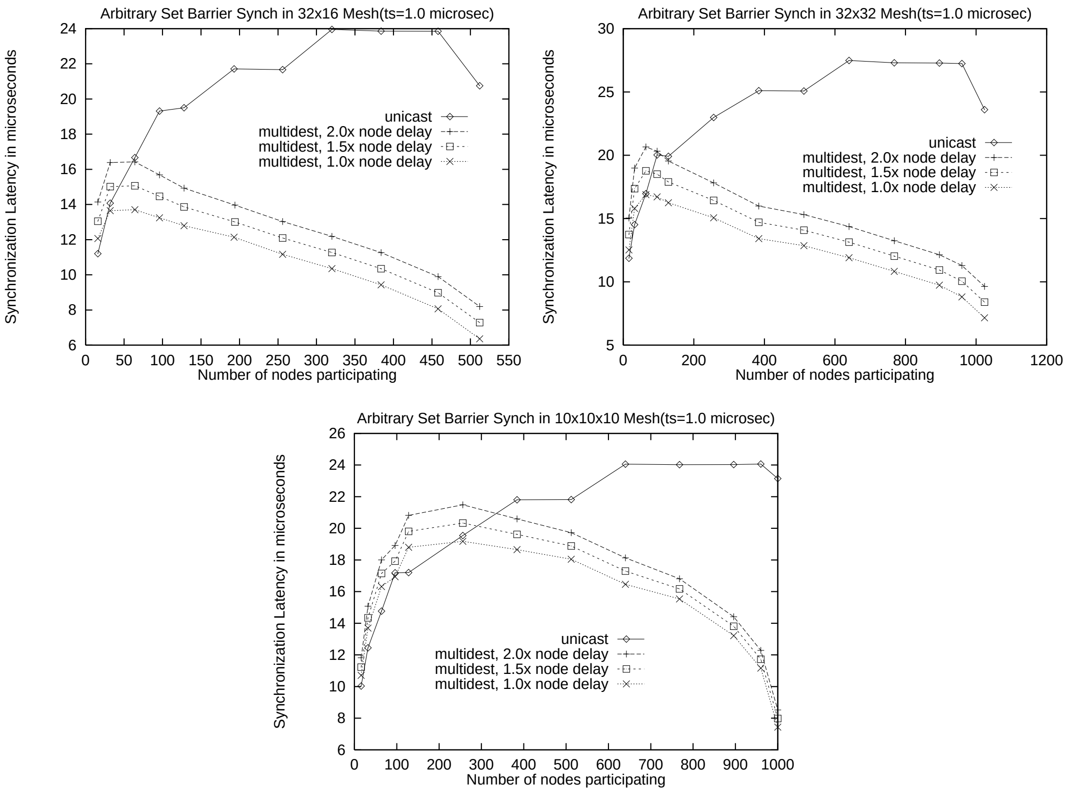


Figure 10: Comparison of unicast-based and multidestination-based schemes to implement arbitrary set barrier synchronization in 32x16, 32x32, and 10x10x10 meshes with different node delays.

5.3 Effect of Communication Start-up

We studied the effect of communication start-up time on the overall synchronization cost. Figure 11 shows the effect when 40% and 80% processors in a 32x32 mesh participated in a barrier, respectively. A range of communication start-up times were considered: 0.25, 0.5, 1.0, 2.0, and 5.0 microseconds. Link propagation and node delays were kept constant at 0.005 and 0.020 microseconds, respectively. For all values of communication start-up time, the multidestination scheme implemented the barrier with reduced cost. For 80% nodes participating, even with t_s=0.25 microsecond, the multidestination scheme outperformed the unicast-based scheme by a factor of 2.51. The improvement was by a factor of 2.0 for 40% nodes participating. This result shows that even if the communication start-up time decreases drastically in future, the multidestination scheme will be able to implement faster barrier synchronization than the unicast-based scheme.

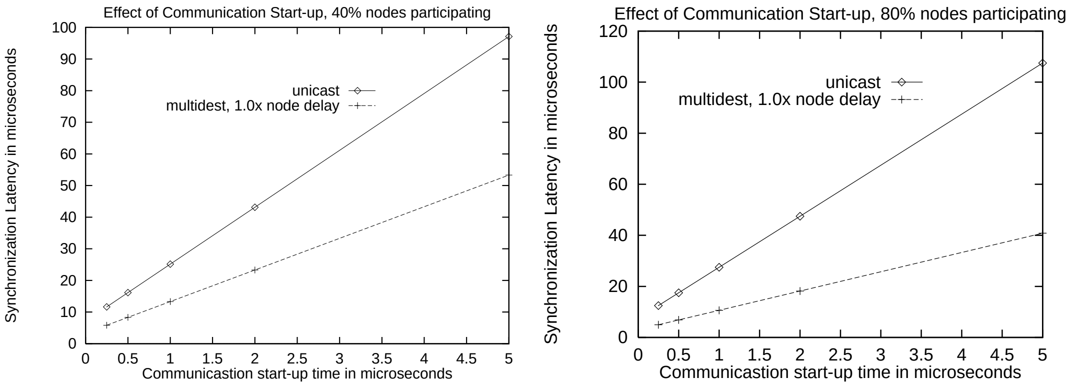


Figure 11: Effect of communication start-up on synchronization cost in a 32x32 mesh (a) 40% nodes participating and (b) 80% nodes participating.

5.4 Effect of Link Propagation and Node Delays

Finally, we studied the impact of link propagation and node delays on the scalability of multidestination scheme. We used three different pairs of node delay and link propagation delay values: (0.010, 0.0025), (0.020, 0.005), and (0.040, 0.01) microseconds. These parameters are shown in x-axis of Fig. 12 in a combined manner: 0.0125, 0.025, and 0.05 microseconds, respectively. It can be observed that even for the lowest values of node delay and link propagation delay, the multidestination scheme shows an improvement of 1.85 and 2.6 over unicast-based scheme for 40% and 80% nodes participating, respectively. It shows that with future improvement in communication link technology and node design, the multidestination scheme will be able to implement faster barrier synchronization than the unicast-based scheme.

6 Conclusions and Future Research

In this paper, we have presented a new approach to implement fast and scalable barrier synchronization in k-ary n-cube wormhole systems using multidestination gather and broadcast worms. The scheme uses message-passing with suitable architectural supports at the router interface

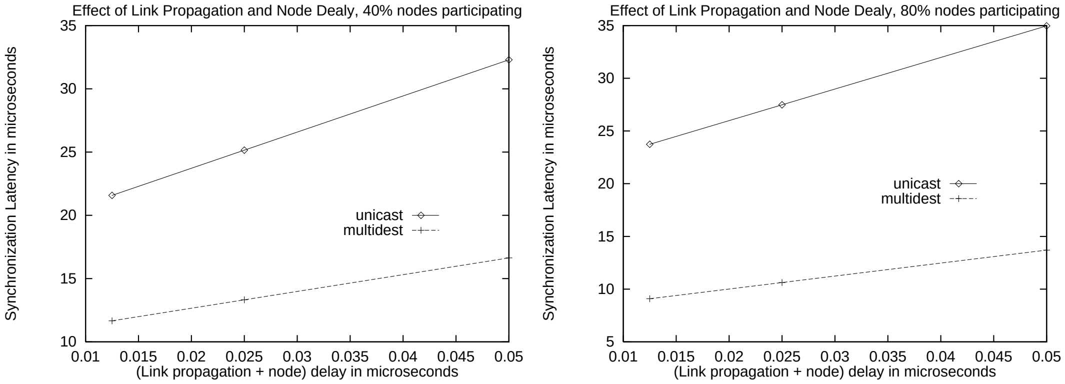


Figure 12: Scalability of multidestination scheme over unicast-based scheme with increase in link propagation and node delay in a 32x32 mesh (a) 40% nodes participating and (b) 80% nodes participating.

and alleviates the need of a separate barrier synchronization network. The necessary architectural supports and modifications to the router interface design are presented. Algorithms are developed and evaluated for both complete and arbitrary set barrier synchronization. With a communication start-up time of 1.0 microsecond, the scheme is capable of barrier synchronizing 4K processors, organized as a 64x64 mesh, in just 11.68 microseconds compared to 30.9 microseconds required by the unicast-based scheme. For arbitrary set barrier synchronization, the scheme demonstrates potential to *reduce* synchronization cost as the number of participating nodes *increases* beyond a certain number. Simulation studies with respect to various system, topology, and technological parameters indicate the superiority of the proposed approach over the conventional unicast-based scheme. The study indicates that the proposed scheme will be immensely beneficial on future k-ary n-cube wormhole-routed systems with larger k and lower n to provide fast barrier synchronization.

In this paper, we have studied the usage of multidestination mechanism for single barrier with simultaneous arrival. We will be evaluating the scheme for multiple barriers with asynchronous arrival and their impact on overall program execution time. We are also extending our work to other collective communication patterns like global-combine [17] and parallel-prefix. As the system size grows, the proposed scheme encounters more delay due to increased path length. We are working on alternative schemes to reduce such impact of path length. Since barrier synchronization is a basic mechanism for implementing distributed shared memory on direct networks, we are extending this framework to see how scalable distributed-shared-memory systems can be designed with low synchronization overheads including barriers, semaphores, and locks.

Acknowledgments: The author would like to thank the students of his *Parallel Architecture and Communication* (PAC) research group for providing comments, criticisms, and suggestions to this work. Special thanks go to Debashis Basak and Sanjay Singal for their help in designing the simulator.

References

[1] C. J. Beckmann and C. D. Polychronopoulos. Fast Barrier Synchronization Hardware. In *Proceedings of the Supercomputing '90, New York*, pages 180–189, Nov 1990.

[2] R. V. Boppana, S. Chalasani, and C. S. Raghavendra. On Multicast Wormhole Routing in Multicomputer Networks. In *Symposium on Parallel and Distributed Processing*, pages 722–729, 1994.

[3] C.-M. Chiang and L. M. Ni. Multi-Address Encoding for Multicast. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, May 1994.

[4] A. A. Chien and J. H. Kim. Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 268–277, 1992.

[5] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[6] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, pages 547–553, May 1987.

[7] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[8] A. Feldmann, T. Gross, D. O'Hallaron, and T. Stricker. Subset Barrier Synchronization on a Private Memory Parallel System. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 209–218, 1992.

[9] C. J. Glass and L. Ni. The Turn Model for Adaptive Routing. In *Proceedings of the International Symposium on Computer Architecture*, pages 278–287, 1992.

[10] R. Gupta. The Fuzzy Barrier: A Mechanism for the High Speed Synchronization of Processors. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 54–63, 1989.

[11] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1990.

[12] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free Multicast Wormhole Routing in Multicomputer Networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 116–124, 1991.

[13] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[14] L. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, pages 62–76, Feb. 1993.

[15] M. T. O'Keefe and H. G. Dietz. Hardware Barrier Synchronization: Dynamic Barrier MIMD (DBM). In *Proceedings of the International Conference on Parallel Processing*, pages I: 43–46, Aug 1990.

[16] M. T. O'Keefe and H. G. Dietz. Hardware Barrier Synchronization: Static Barrier MIMD (SBM). In *Proceedings of the International Conference on Parallel Processing*, pages I: 35–42, Aug 1990.

[17] D. K. Panda. Global Reduction in Wormhole k-ary n-cube Networks with Multidestination Exchange Worms. In *International Parallel Processing Symposium*, Apr 1995.

[18] D. K. Panda, S. Singal, and P. Prabhakaran. Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 131–145, 1994.

[19] J. A. Solworth and J. Stamatopoulos. Integrated Network Barriers for D-dimensional Meshes. In *Proceedings of the Working Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, 1992.

[20] H. Xu, P. K. McKinley, and L. Ni. Efficient Implementation of Barrier Synchronization in Wormhole-routed Hypercube Multicomputers. *Journal of Parallel and Distributed Computing*, 16:172–184, 1992.