Hybrid Algorithms for Complete Exchange in 2D Meshes

N. S. Sundar D. N. Jayasimha[†] [1] D. K. Panda P. Sadayappan

Dept. of Computer & Information Science
The Ohio State University
Columbus, OH 43210
(sundar,panda,saday)@cis.ohio-state.edu

[†]Intel Corporation, MS RN2-02
2200 Mission College Blvd
Santa Clara, CA 95052-8119
djayasim@mipos2.intel.com

[1]Work done while at the Dept. of Computer and Information Science, The Ohio State University.

Abstract

Parallel algorithms for several common problems such as sorting and the FFT involve a personalized exchange of data among all the processors. Past approaches to doing complete exchange have taken one of two broad approaches: direct exchange or the indirect message-combining approaches. While combining approaches reduce the number of message startups, direct exchange minimizes the volume of data transmitted.

This paper presents a family of hybrid algorithms for wormhole-routed 2D meshes that can effectively utilize the complementary strengths of these two approaches to complete exchange. The performance of hybrid algorithms using Cyclic Exchange [24] and Scott's Direct Exchange [21] are studied using analytical models, simulation and implementation on a Cray T3D system. The results show that hybrids achieve lower completion times than either pure algorithm for a range of mesh sizes, data block sizes and message startup costs. It is also demonstrated that barriers may be used to enhance performance by reducing message contention, whether or not the target system provides hardware support for barrier synchronization. The analytical models are shown useful in selecting the optimum hybrid for any given combination of system parameters (mesh size, message startup time, flit transfer time and barrier cost) and the problem parameter (data block size).

Contents

1 Introduction

The all-to-all personalized (complete) exchange is a global collective communication operation in distributed memory multicomputers in which every processor sends a unique block of data to every other processor in the system. This pattern arises in many important problems such as multidimensional FFT, matrix transposition and sorting. It is one of the collective communication primitives defined by the MPI [11] message passing interface standard. Since complete exchange is communication intensive, its effective implementation in wire-sparse topologies like 2D meshes is a challenging problem.

The principal issues that need to be addressed in an effective solution to the complete exchange problem are those of *high message startup overhead*, the *high volume of data* that needs to be transmitted over the network and *contention among messages* for the mesh links. If every processor in an n-processor mesh directly delivers its block of data to every other processor in the system, each processor needs to execute at least $n-1$ message startups sequentially. As many modern multicomputers have a high message startup time, which is often orders of magnitude higher than the time to transmit a byte over a link, this factor strongly influences the completion time.

The total volume of data that the network has to carry is at least $n(n-1)m$ bytes in an n-processor system, where m is the size of each data block. Since this is quadratic in the system size, it is another significant factor that affects performance. In a 2D mesh, the limited bisection bandwidth implies that the central links should be optimally used to achieve good performance.

Furthermore, contention is a significant factor in wormhole-routed networks because wormhole routing reserves network links. When a message is forced to wait for a link to become available, the links which it holds are not released, which can cause other messages to block. So, significant network bandwidth may become unusable and message latency increases. Thus, contention is a critical factor that needs to be managed.

Past approaches to performing complete exchange on 2D meshes have taken one of two distinct approaches. The *direct exchange* approach involves each processor directly sending a block of data to every other processor in the system. For example, the *Optimal* algorithm for hypercubes [14], in which every processor exchanges data with processor $myid \oplus i$ in step i, can be adapted for meshes. Thakur *et al.* [28, 27] evaluate algorithms that take this approach. Scott [21] has proven that, in an $a \times a$ mesh, the number of contention-free communication steps, each of which overlaps messages between several processor pairs, must be at least $\frac{1}{4}a^3$. He also presents a methodology for developing direct exchange algorithms that realize this lower bound when a is a multiple of 4.

In contrast to direct exchange, an alternative approach is to combine the data blocks that a processor P has for all processors in a submesh and send them as a single message to one processor Q within that submesh. Q then forwards the data blocks towards their respective destinations in subsequent phases. This *combining* or *indirect* approach typically takes a divide-and-conquer strategy. For example, Bokhari and Berryman [9] developed two algorithms that use this approach called Binary Exchange and Quadrant Exchange. Binary Exchange recursively divides the mesh into halves while Quadrant Exchange organizes the mesh in quadrants. The latter has been shown to be faster than the former. Sundar *et al.* [24] proposed another combining algorithm called Cyclic Exchange, which recursively divides the mesh into quadrants. (Such an algorithm was also proposed independently by Gupta *et al.* [13].) This was demonstrated to be faster than Quadrant Exchange for a wide range of mesh sizes and data block sizes.

The direct exchange and the indirect (combining) approach have complementary strengths. Direct Exchange minimizes the volume of data that must be transmitted sequentially over the network but has a large number of message startups. Combining approaches reduce the number of startups considerably, but move larger volumes of data over the network since some blocks are retransmitted several times. This tradeoff suggests that both these schemes may be integrated to exploit their complementary strengths.

This paper presents a family of algorithms for complete exchange on wormhole-routed 2D meshes, derived by hybridizing Cyclic Exchange and Scott's Direct Exchange. These hybrids are formulated by replacing the last K phases in the recursive structure of Cyclic Exchange with Direct Exchange. The value of K that minimizes completion time will depend on system parameters (mesh size, message startup time, flit transfer time and barrier cost) and the problem parameter (data block size). Analytical models are presented for the hybrids, which may be used to pick the best hybrid for a given combination of parameters when barriers are used (see below). A rule of thumb is also described, which can be used to quickly predict the best hybrid for the given parameters with good accuracy, in the absence of barriers. The analytical models are validated with a simulation study and an implementation on the Cray T3D multicomputer, whose results confirm that the hybrids are the fastest algorithms for a large range of system and problem parameters. A unique feature of the paper is that link contention is quantified accurately with an analytical model, unlike many past studies which model contention in an approximate manner. The paper also investigates the use of barriers, implemented either in hardware or software, to reduce contention, thereby improving performance. A counter-intuitive experimental observation is presented and explained, where higher message startup costs actually *reduce* completion time.

The paper is organized as follows. Section 2 describes Direct Exchange [21] and presents an analytical model which is applicable when the algorithmic steps execute in synchrony (as in an SIMD architecture or in an MIMD architecture with barrier synchronization between steps). Section 3 presents Cyclic Exchange and presents analytical models for it, both with and without the assumption of synchrony. The motivation for hybridizing and the structure of the hybrid algorithms are discussed in Section 4, which also uses the analytical model to prove theorems that provide useful insights into the behavior of the hybrids. Section 5 presents the results of a simulation study of hybrid algorithms, with and without the use of barriers. The simulation results are in close correspondence to the analytical predictions and show that the proposed hybridizing scheme is effective for a wide range of system and problem parameters. Section 6 discusses the influence that message contention has on finish time and presents evidence that use of barriers can enhance performance by reducing contention. Since the target system may or may not provide hardware support for barrier synchronization, section 7 focuses on the effectiveness of barrier usage in both cases. The results of the implementation of the hybrid algorithms on the Cray T3D multicomputer are presented in Section 8 and they are shown to confirm the trends seen with the analytical model and the simulation study. Section 9 points to previous related literature and shows how this paper differs from them. Finally, Section 10 summarizes the work.

Table 1 explains the notation commonly used in the rest of the paper.

Table 1: Symbols and notation frequently used in this paper.

Symbol	Explanation	Symbol	Explanation
CE	Cyclic Exchange	DE	Direct Exchange
m	size (in bytes) of a data block	M	size (in bytes) of a message in CE
b	Cost of barrier synchronization	t_{mem}	time to copy a byte within local memory
t_p	Time for a flit to cross a mesh link	t_s	Message startup overhead
K	the number of steps in a hybrid algorithm that are carried out with Direct Exchange		
p	logarithm of the mesh size i.e. mesh size is $2^p \times 2^p$		

2 Direct Exchange

There are several algorithms [21, 28, 27] for complete exchange that schedule a direct communication between every pair of processors. The main issues in this approach are the large number of message startups and contention for the central links of the mesh, which form a bottleneck. In an $a \times a$ mesh, a lower bound can be derived for the number of contention-free communication steps needed, by noting that there are $\frac{a^2}{2}$ nodes on either half of the mesh, so that $\frac{a^4}{4}$ data blocks need to cross the central links, but there are only a central links; therefore there need to be at least $\frac{a^3}{4}$ sequential steps.

2.1 The methodology of direct exchange

Scott [21] presented a scheme to generate complete exchange algorithms which achieve the above lower bound when a is a multiple of 4. The central idea behind the scheme is a set of contention-free communication patterns for complete exchange on a one-dimensional chain of processors, which is then extended to two dimensions by the use of cross-products of these patterns in a contention-free manner. The following is a description of this methodology.

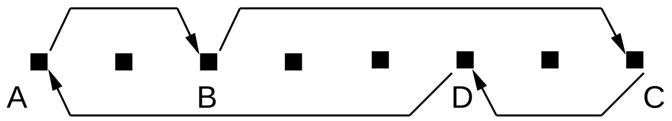

Figure 1: Scott's Direct exchange on a 1D chain of processors.

On a chain of a processors, numbered $0, 1, \ldots, a-1$, define the antipodal node of processor x as $(a-1)-x$. Designate two nodes in the left half of the chain as A and B. Let their corresponding antipodal nodes be C and D, respectively. A communication pattern in which A sends to B, B to C, C to D and D to A, would utilize the central link in both directions (see Figure 1). By varying A and B over the $\frac{a}{2}$ nodes in the left half of the mesh (such that they are always distinct), $\frac{a}{2}(\frac{a}{2}-1)$ *regular* steps are generated, which collectively do a complete exchange except for communication among the antipodal nodes themselves. The latter is achieved through a set of $\frac{a}{2}$ *special* steps, in each of which a node on the left side exchanges a data block with its antipodal node on the right side. Thus, a total of $\frac{a^2}{4}$ steps are needed (if a were even).

On a 2D mesh, complete exchange can be carried out by executing cross products of all the (regular and special) steps in the vertical and horizontal dimension. Figure 2 shows one such cross product. To include communication among processors in the same row (or the same column), it is necessary to assume that, in the special steps, there are two processors that are "actively sending to themselves". If every cross product were executed sequentially, this would result in $\frac{a^4}{16}$ steps, which is suboptimal compared to the lower bound of $\frac{a^3}{4}$. The key to deriving an optimal scheme is to observe that, while in a 4×4 mesh, all the processors are busy in every step, that is not the case with $4k \times 4k$ meshes for $k > 1$. In fact, the 1D schemes would involve only four processors in each row (and in each column),

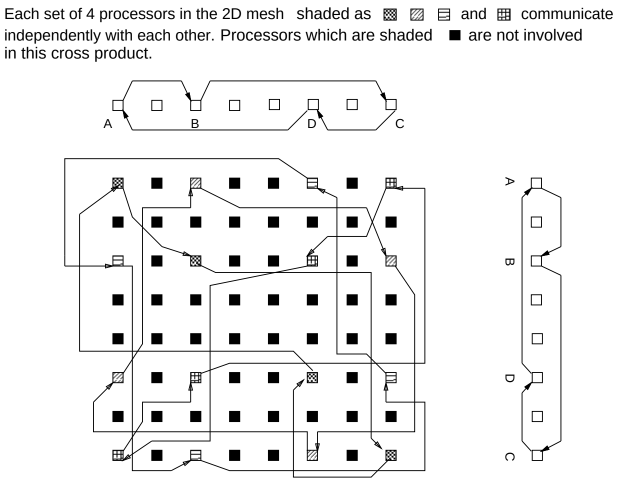

Figure 2: A cross product of two 1D patterns in Scott's Direct Exchange.

so that k separate cross products can be scheduled within a step, as long as they don't contend among one another. Scott presents a method based on the tournament scheduling problem, to choose the k cross products that constitute each step. Thus, the number of steps is reduced by a factor of $k = a/4$, so that the total number of steps is optimal.

In the following, any of the 2D algorithms derived by this methodology will be called Direct Exchange.

2.2 An analytical model for Direct Exchange

Each step in Direct Exchange has no link contention. However, when they are executed on an MIMD architecture, the steps are not synchronous, so that link contention will result. This is because not all processors will finish a given step together, even if they started it together. Besides, some processors are idle in steps that involve the special phases, so that they will attempt to begin the next step and thus contend with processors executing the current step.

Contention is difficult to quantify and model in Direct Exchange, in part because it depends on the sequence of cross products that get scheduled. Furthermore, our simulation results (presented in Section 5) show that controlling the contention by imposing a barrier between successive steps (so that the steps execute synchronously) results in better completion time, for a large range of mesh sizes and problem sizes. Hence, an analytical model is presented under the assumption that the steps are executed synchronously, with a barrier between successive steps.

On an $a \times a$ mesh, there are $\frac{a^3}{4}$ steps. Each step incurs a message startup cost and a message transmission cost. The time for a message transmission in a wormhole-routed mesh has two components: one for the header to establish a path (t_h) and one for the rest of the message to follow in pipeline fashion, the latter being typically dominant. The first term depends on the number of hops the message needs to travel. Thus, the header terms for the longest messages within each step can be determined, once the schedule of cross products is fixed, and their sum T_h can be found. This yields the completion time

$$T_{DE} = \frac{a^3}{4} t_s + \frac{a^3}{4} m t_p + T_h + (\frac{a^3}{4} - 1) b \tag{1}$$

The cost due to startups is quite high, though the transmission time component is optimal.

3 Cyclic Exchange

In contrast to the direct exchange approach, the combining approach gathers several data blocks that a given processor P has for an entire submesh and sends them in a single message to a processor Q within that submesh. Q then forwards the appropriate data blocks to the various submeshes within that submesh. This gives a recursive structure to the combining algorithms known today.

The Cyclic Exchange (CE) algorithm [24] works on a 2D circuit-switched or wormhole-routed mesh. The algorithm is described for a $2^p \times 2^p$ mesh but it can be adapted to non-square meshes with non-power-of-two number of processors to each side.

3.1 Cyclic Exchange algorithm

We first present the underlying idea for a one-dimensional array of 2^p nodes. Figure 3 illustrates it for $p = 3$. The communication occurs in p phases such that, in phase i ($i = p-1, \ldots, 0$), processors that are 2^i hops away exchange data between themselves. Consider a system with $p = 3$, that is 2^3 processors. In the first phase, every processor communicates with its counterpart (4 hops away) in the other half of the linear array. In the second phase, each processor exchanges with its counterpart (2 hops away) in the other quarter of the same half. The last phase finally swaps data among neighboring nodes. The size of the messages remains constant at $4m$ bytes in each phase, m being the size of a data block.

For a 2D mesh, CE interleaves this pattern in both dimensions. Assume initially that the communication is organized in subphases that execute synchronously. The algorithm for two dimensions is illustrated in Fig. 4 for a 4×4 mesh. In the initial phase, the entire mesh is involved and the messages

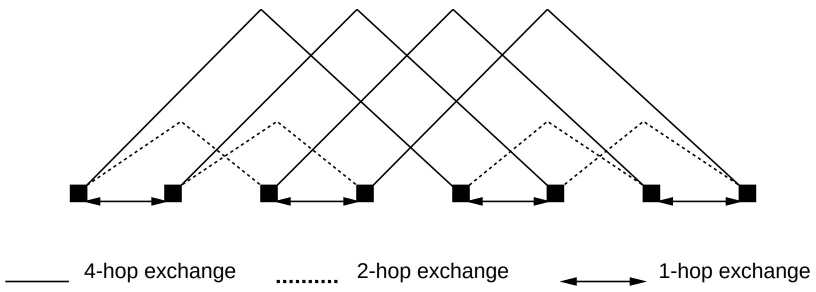

Figure 3: Cyclic Exchange on a linear array of 8 nodes.

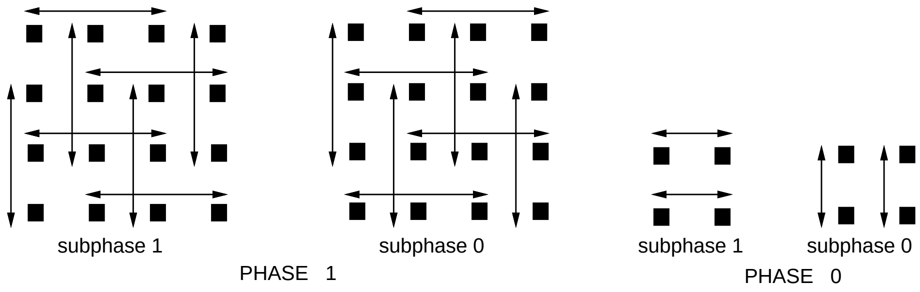

Figure 4: Cyclic Exchange on a 4×4 mesh, if the processors execute in lockstep.

are of length $2^{2p-1}m$ bytes. Subsequent phases recursively divide the mesh into quadrants and operate on each quadrant independently. Phase i (where $i = 1, 0$) operates on a $2^{i+1} \times 2^{i+1}$ submesh and has 2^i subphases because that many processors have to communicate across the central link in each row (except for phase $i = 0$ which has two subphases). Phase $i = p - 1$ is performed first.

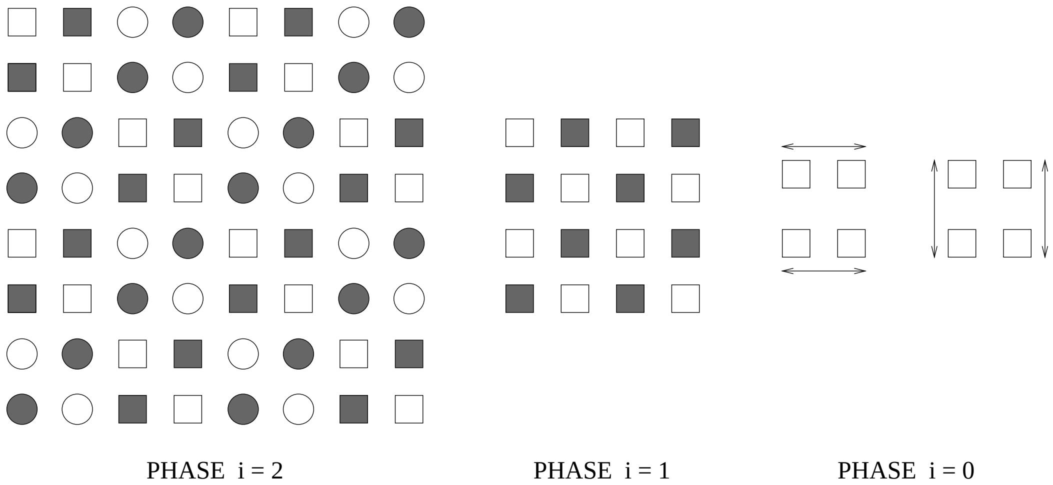

Figure 5: The Cyclic Exchange Algorithm for an 8×8 mesh.

In an MIMD system, the subphases will not execute synchronously. Different communications will get overlapped, both within and across phases. The algorithm behavior under these conditions is illustrated in Fig. 5 with an 8×8 mesh. Three phases are required; during phase i (where $i = 2, 1, 0$), each processor communicates with two others that are 2^i hops away, one in the same column and one in the same row. Thus, communications occur independently in submeshes of size $2^{i+1} \times 2^{i+1}$. A checkerboard pattern is used to interleave communication so that, in each phase, in the first step, half the processors communicate horizontally (the white nodes) and the other half communicate vertically (the black nodes). In the next step, white nodes communicate vertically and black nodes communicate horizontally. For example, in Figure 5, processors marked with a white circle first communicate horizontally to the similarly marked processor in the same row, and then communicate vertically to the similarly marked processor in the same column.

Every processor has an array of 2^{2p} data blocks. In every communication, two data quadrants of the data array are sent to the partner node, so that message size is constant throughout the algorithm at $2^{2p-1}m$ bytes. The data need to be organized appropriately so that, in each communication, the set of data blocks to be sent are contiguous in the data array. This necessitates two kinds of movement of data blocks: reorder between phases and data movement within each phase. Our implementation of Cyclic Exchange uses a scheme that overlaps all intraphase data movement and half the reordering with communication, by using a user buffer which is $3.2^{2p-2}m$ bytes in size. Details of this scheme are provided in Appendix A, along with the Cyclic Exchange algorithm. A high-level outline of the algorithm showing the major communication steps and the data movement is presented in Figure 6.

```
{ SPMD program executed by the processor in
  row r and column c. }
algorithm CyclicExchange(r, c).
begin
        for i ← p − 1 downto 0 do { Phase i }
                HorizProc ← (r, c ± 2^i)
                VertProc ← (r ± 2^i, c)
                if ( r mod 2 = c mod 2 ) or ( i = 0 ) {White Node}
                        Partner1 ← HorizProc; Partner2 ← VertProc
                else {Black Node}
                        Partner2 ← HorizProc; Partner1 ← VertProc
                endif
                Step1: Exchange 2^(2p−1) data blocks with Partner1
                        Perform intra-phase data movement of 2^(2p−2)m bytes
                Step2: Exchange 2^(2p−1) data blocks with Partner2
                if ( i > 0 ) perform interphase data reordering of 2^(2p−1)m bytes
        endfor
end
```

Figure 6: The Cyclic Exchange Algorithm for Complete Exchange.

3.2 Analytical models for Cyclic Exchange

In an MIMD implementation of Cyclic Exchange, in general, messages will contend for mesh links. The pattern of contention among messages can be predicted and modeled. However, when mesh size p and data block size m exceed certain limits, contention becomes high and difficult to model analytically. In fact, as our simulation results show (Section 5), the contention may be so high in this parameter range that imposing a barrier between successive steps or phases *decreases* completion time by eliminating contention among messages that belong to different steps/phases. When barriers are imposed between phases, it is feasible to analytically model the algorithm's behavior for those parameter ranges. Accordingly, we use a three-part model: one part without barriers which is valid for small mesh size p and data block size m and two others with barriers between phases or steps which is valid for large p and large m. A guideline for deciding which part of the model to use is also formulated (see Condition 4 in this section).

In the absence of barriers between steps, there are two kinds of message contention in general in Cyclic Exchange. One is contention among messages originating from nodes of the same color. For example, in any phase, all the white nodes on the left side of any given row will send a message each to their partners on the right side; all these will contend for the central link of the row. This phenomenon is called same-color contention. Therefore, in phase j, which happens on a $2^{j+1} \times 2^{j+1}$ submesh, a message from the far left white node of a row will reach its destination with a latency of

$$T_{sc}^{(j)} = t_h^{(j)} + 2^{j-1} M t_p \tag{2}$$

where $t_h^{(j)}$ is the time for header movement for one message (since the header movements of other white messages are overlapped with data transmissions) and $M = 2^{2p-1}m$ is the message size when data blocks are m bytes long.

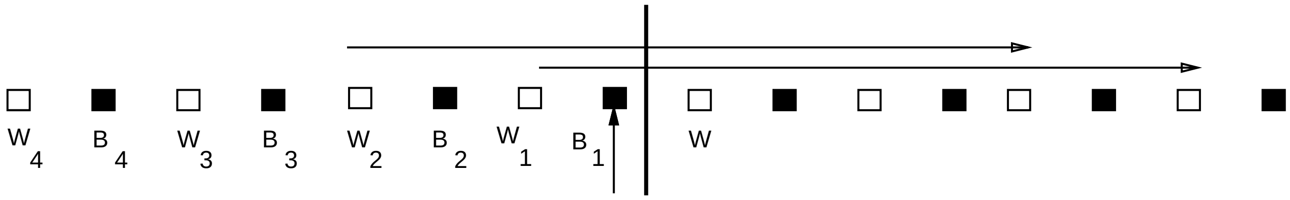

Figure 7: The topmost row of a 16×16 mesh at the time when the last flit in the horizontal message of W_1 has reached its destination. The black node B_1 has also received its last flit and is performing a startup. It cannot block W_2 but, if the startup time were small, it can block W_4, causing multicolor contention.

The other kind of message contention is contention among messages from nodes of differing colors. For example, the message from a white node (call it a white message) on the far left of a given row may be so delayed by other white messages from the same half of the row that a black node in that half may finish the reception of its first (vertical) message, perform the message startup for the next (horizontal) communication and initiate the horizontal communication, thus colliding with the white message and delaying it further. In Figure 7 which illustrates this situation, the last flit from the white node labeled W_1 has reached its destination, and simultaneously the black node B_1 has received the last flit of its first (vertical) message. The next (horizontal) message of B_1 cannot block W_2 as the latter's message has followed that of W_1 in pipeline fashion. However, it can block the other white nodes depending on how long the intervening message startup is. Since all the black nodes in this row receive their vertical messages at the same time, all of them can contend with and delay W_4, except for the immediately adjacent black node B_4. Of course, the other white nodes will also delay W_4 by same-color contention. Thus, the time when the partner of W_4 receives its message is

$$T_{mc}^{(j)} = t_h^{(j)} + (2^j - 1)Mt_p \tag{3}$$

Note that, in an 8×8 mesh, multicolor contention cannot happen because there are only two white nodes on each half of a row. Therefore, it cannot happen on smaller meshes either. Similarly, if barriers were imposed between successive steps in each phase, multicolor contention would be eliminated. On a 16×16 mesh, whether a black node can delay a white node in the same half depends on the relative magnitudes of the message startup time t_s and the time to transmit one message Mt_p. If

$$t_s < Mt_p \text{ and } p >= 4, \tag{4}$$

multicolor contention can be expected to be significant. Since this is the case for 32×32 and larger meshes for practically any value of m, for reasonable values of t_s and t_p, it is better to impose a barrier for large p.

The remainder of this section presents analytical models for Cyclic Exchange for three cases: barrier synchronization between steps (and between phases too) , barrier synchronization between phases only and no barrier synchronization at all.

3.2.1 Model with barrier between steps

This model is applicable when successive steps have a barrier between them, so that multicolor contention is eliminated. Within each step, the message startups are overlapped. Thus, with $2p$ steps, the total startup time is

$$S = 2pt_s$$

In phases that involve 8×8 or larger meshes, messages from nodes of the same color will contend for central links. For example, horizontal messages from white nodes on one half of a row will all contend for the central link of that row. Thus, each step in phase j will involve a time of $2^{j-1}Mt_p$. The phases involving 4×4 and smaller meshes have no such contention. Thus, the sum of the non-overlapped transmission times is

$$\sum_{j=1}^{p-1} 2.2^{j-1} M t_p + \underbrace{2Mt_p}_{j\,=\,0}$$

ignoring the times for header movement in each message. This evaluates to

$$T = 2^{3p-1} m t_p$$

Since only half the data array is reordered between successive phases, the total time spent in reordering is

$$R = (p-1)2^{2p-1} t_{mem}$$

The fourth component of the completion time is the barrier overhead of

$$B = (2p-1)b$$

So, the completion time can be expressed as the sum of these four components $S+T+R+B$, that is

$$T_{CE}^{barr} = 2pt_s + 2^{3p-1} m t_p + (p-1)2^{2p-1} t_{mem} + (2p-1)b \tag{5}$$

3.2.2 Model with barrier between phases

If the condition 4 holds true, multicolor contention becomes significant and causes a large variation in the times when different processors finish each phase. This leads to considerable contention among messages belonging to successive phases, thus affecting the completion time. Interphase message contention can be eliminated by imposing a barrier between successive phases. This results in an additional cost of $p-1$ barriers, but if the barrier cost were low, the reduction in contention can decrease the completion time.

Each phase can be analyzed independently in the presence of barriers, as all processors begin a phase together. In 16×16 and larger meshes, multicolor contention will delay the message reception of processors such as W (see figure 7) considerably, so that the highest latency is $T_{mc}^{(j)}$ in phase j. However, it is only the nodes farthest from the center of the mesh that will face this high degree of contention; others will finish their message reception sooner. Therefore, in the next step of the same phase, far fewer processors will be involved in contention. As the exact number N_j of contending processors will depend on the dynamics of flit movement in the previous communication, it is difficult to give a general closedform formula to estimate it. However, for specific mesh sizes, N_j can be determined. For example, for a 16×16 mesh in phase 3, $N_3 = 2$ and for a 32×32 mesh in phase 4, $N_4 = 5$. Thus, the time for phase j can be written as

$$T_j = t_s + reorder_j + \left\{ \begin{array}{cc} T_{mc}^{(j)} + N_j & \text{if } j \geq 3 \\ 2T_{sc}^{(j)} & \text{else} \end{array} \right\}$$

The completion time is the sum of all phase times, including the barrier cost is

$$\sum_{j=0}^{j=p-1} T_j + (p-1)b$$

3.2.3 Model with no barriers

In the absence of barriers, all processors execute asynchronously so that different phases may overlap. The model for this case has to consider the critical path i.e. the sequence of message transmissions that take the longest to complete. Accordingly, for each phase j ($p-1 \geq j \geq 0$), define $earliest_j$ as the time when phase j can be completed earliest, and $latest_j$ as the time when phase j is completed by the last processor(s) (i.e. when both the horizontal and vertical messages have been received and reordering has been performed). Recurrence equations can be defined for both these sets of times. The time $latest_0$ gives the completion time for the algorithm.

To derive the recurrence equation for $earliest_j$, note that the processors in the initial phase $j = p-1$ that face no contention in message reception in either dimension will finish first. In the last phase $j = 0$, all messages occur within 2×2 meshes, so that there is no contention here either. In all other phases, the processors that finished earliest in previous phases will be able to initiate a transmission first but will face same-color contention. Thus,

$$earliest_j = t_s + reorder_j + \left\{ \begin{array}{ll} earliest_{j+1} + 2 * T_{free}^{(j)} & \text{if } j = p-1 \text{ or } j = 0 \\ earliest_{j+1} + 2 * T_{cont}^{(j)} & \text{else} \end{array} \right\} \tag{6}$$

for $p-1 \geq j \geq 0$. Here $T_{free}^{(j)} = t_h^{(j)} + Mt_p$ is the latency for a contention-free transmission and $T_{cont}^{(j)}$ is either $T_{sc}^{(j)}$ or $T_{mc}^{(j)}$ depending on condition 4. The reorder term is $2^{2p-1}mT_{mem}$ only if $j > 0$. In the last phase $j = 0$, it is zero.

The last processor to finish a phase would have faced contention in receiving messages in either dimension, unless it were the 2×2 submesh phase, or if it were receiving from a processor that finished the previous phase so late that, in the current phase, it faced no contention. Thus,

$$latest_j = t_s + reorder_j + \left\{ \begin{array}{ll} latest_{j+1} + 2 * T^{(j)}_{free} & \text{if } latest_{j+1} > earliest_{j+1} + T^{j}_{contention} \\ latest_{j+1} + 2 * T^{(j)}_{free} & \text{if } j = 0 \\ latest_{j+1} + 2 * T^{j}_{cont} & \text{else} \end{array} \right\} \quad (7)$$

The reorder term is defined in the same way as above.

4 Hybridizing combining and direct algorithms

To motivate the hybridizing scheme, consider the behavior of Direct Exchange (DE) and Cyclic Exchange (CE), both with barriers between steps. The completion time for DE for a $2^p \times 2^p$ mesh can be expressed as (Equation 1):

$$2^{3p-2}t_s + 2^{3p-2}mt_p + (2^{3p-2} - 1)b$$

The completion time for CE is (Equation 5)

$$2pt_s + 2^{3p-1}mt_p + (p-1)2^{2p-1}mt_{mem} + (2p-1)b$$

Thus, the startup component is much higher with DE but the transmission time component is optimal. With CE, the startup component is much lower but its transmission volume is double that of DE and it also incurs a reordering overhead. This suggests that both the schemes can be combined to exploit their complementary benefits.

Consider an execution of CE on a $2^p \times 2^p$ mesh, with p phases. In phase j, where $p-1 \geq j \geq 0$, CE operates on independent submeshes of size $2^{j+1} \times 2^{j+1}$. The communication performed in a $2^{j+1} \times 2^{j+1}$ submesh for phases j and lower is equivalent to performing complete exchange on a mesh of that size with an effective data block size of $2^{2(p-1-j)}m$. Thus, in the recursive structure of CE, successive phases operate on smaller submeshes with increasing effective block sizes. A decrease in the mesh size implies that startup overhead becomes less significant. The increased block size implies a higher transmission volume. Both these factors favor DE. So, at some point in the sequence, it may become more advantageous to perform a single direct exchange within each submesh instead of carrying out the remaining phases of CE. That is, at some phase $j = K-1$, it may be beneficial to change to DE and replace the last K phases of CE with a single DE operation, which is executed in every $2^K \times 2^K$

submesh independently. The effective block size in the DE stage of such a hybrid algorithm is $2^{2p-2K}m$.

Thus, a family of $p+1$ hybrid algorithms may be generated for a $2^p \times 2^p$ mesh, each of which performs $p-K$ phases of CE, for some K in the range $0 \ldots p$, and then a single DE operation in each $2^K \times 2^K$ submesh. Thus, $K = 0$ corresponds to pure CE while $K = p$ corresponds to pure DE. Since phases involving larger meshes require more startups, use of combining in these phases helps in minimizing the overall number of message startups. When the importance of minimizing startup costs decline, due to the decrease in submesh size, the change to DE helps in decreasing transmission volume. Thus, this scheme of hybridizing exploits the benefits of both the combining and the direct exchange approaches.

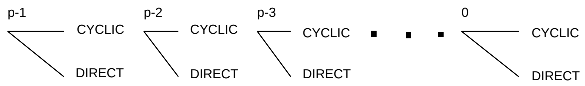

Figure 8: The decision tree for switching to Direct Exchange in a hybrid scheme.

The choice of a suitable K that minimizes the completion time among the various possible K-values in the range $0 \ldots p$ depends on system factors such as mesh size, startup cost, etc. and also on the problem size m. It may appear that the decision to change to DE could be made on a phase-by-phase basis. That is, at the start of a given phase j, CE or DE may be chosen for that phase by comparing the completion times of CE and DE for that submesh size and effective block size. This is equivalent to comparing just two paths in the decision tree, one being *DIRECT*, the other being pure *CYCLIC* (see Figure 8). This ignores other paths in the tree of the form *(CYCLIC,...,CYCLIC,DIRECT)*, which may be faster for the given submesh size and effective block size. Thus, a *local decision* is not possible. All the different paths in the tree need to be compared before deciding the best path.

One way to do this analysis is to traverse the tree bottom-up. Since the submesh size and effective block size can be computed for phase 0, the decision whether to perform that phase with CE or DE can be made. If CE is decided upon, all the p phases need to be CE. Otherwise, the tree node for phase 1 is examined and a decision needs to be made then, and so on. The key observation is that, if a phase π is best performed using CE, then the previous phases $\pi+1, \pi+2, \ldots, p-1$ also need to be done using CE. But, if π is best performed with DE, that does not imply anything about the decisions 'up' the tree.

An equivalent approach to analyze all the paths of the tree is to develop a quantitative model of the hybrid scheme, with K as a dependent parameter. This would allow the computation of the completion times of all possible hybrids and choose the best one. The model is derived below under the assumption that both CE and DE are performed with barriers between steps. As described in section 3.2, we have also developed models for CE that either do not impose barriers at all or impose them only between phases (not steps).

The completion time of the hybrid algorithm can be analytically modeled in terms of the primary factors - startup and transmission time - and the secondary factors - reorder and barrier time. The CE component of the hybrid algorithm has $p - K$ phases, each with two message startups, so that the time spent in message startups in the CE phase is $S^{CE} = 2(p - K)t_s$. The time taken by message transmissions is

$$T^{CE} = \left\{ \begin{array}{cc} (2^p - 2^K)2^{2p-1}mt_p & \text{if } K > 0 \\ 2^{3p-1}mt_p & \text{else} \end{array} \right\}$$

The CE component also has a data reordering overhead of $R^{CE} = \max((p - K - 1)2^{2p-1}mt_{mem}, 0)$ and has a penalty of $B^{CE} = (p - K - 1)b$ in barrier synchronization. The DE component of the hybrid algorithm operates on a $2^{K+1} \times 2^{K+1}$ mesh and hence has 2^{3K-2} steps. So, the time spent by it in message startups is $S^{DE} = 2^{3K-2}t_s$. Message transmissions in the DE component take a time of $2^{3K-2}2^{2p-2K}mt_p$ if $K > 0$, since the size of each message in the DE component is $2^{2p-2K}m$. In general, the message transmission time can be expressed as

$$T^{DE} = \left\{ \begin{array}{cc} 2^{2p+K-2}mt_p & \text{if } K > 0 \\ 0 & \text{else} \end{array} \right\}$$

Between the CE component, if any, and the DE component, the data array needs to be reordered so as to extract the first message. This overhead amounts to

$$R^{DE} = \left\{ \begin{array}{cc} 2^{2p-2K}mt_{mem} & \text{if } K < p \\ 0 & \text{else} \end{array} \right\}$$

Once this is done, the data for subsequent messages can be extracted in parallel with the previous communications while the data for the executing processor itself can be extracted in parallel with the last communication. If $t_{mem} \leq t_p$, these addtional reordering times would be completely subsumed in the transmission time and hence would not contribute to R^{DE}. The barrier synchronizations between steps impose an additional cost of $B^{DE} = (2^{3K-2} - 1)b$. The time incurred by each of the components of the hybrid algorithm can then be expressed as the sum of these terms i.e. $T^{CE}_{hybrid} = S^{CE} + T^{CE} + R^{CE} + B^{CE}$

and $T^{DE}_{hybrid} = S^{DE} + T^{DE} + R^{DE} + B^{DE}$, so that

$$T_{hybrid} = \left\{ \begin{array}{cl} 2pt_s + 2^{3p-1}mt_p + (p-1)2^{2p-1}mt_{mem} + (p-1)b & \text{if } K = 0 \\ 2^{3p-2}t_s + 2^{3p-2}mt_p + (2^{3p-2}-1)b & \text{if } K = p \\ \{2(p-K) + 2^{3K-2}\}t_s + (2^{3p-1} - 2^{2p+K-2})mt_p + & \\ \{(p-K-1)2^{2p-1} + 2^{2p-2K}\}mt_{mem} + (p-K-1+2^{3K-2})b & \text{else} \end{array} \right\} \tag{8}$$

This expression is a function of K and the value K_{opt} for which it attains the minimum can be computed by evaluating this expression for each of the $p+1$ possible values of K, that is $0, \ldots, p$. Thus, the best algorithm in the family of hybrid algorithms can be chosen analytically.

The analytical model can be used to predict the variation of K_{opt} when the underlying system parameters and the problem size vary. The following theorems are based on the analytical model and offer insight into the trends exhibited by K_{opt} when a single parameter is changed, with all others held constant.

Theorem 1: Increasing m cannot decrease K_{opt}, all other factors being equal (that is, the DE component of the hybrid will increase or stay the same when data block size is increased).

Proof: Consider the finish time of the hybrid for a given m and $K > 0$, denoted as $T_{hybrid}(m, K)$. Since K_{opt} is the value of K which results in the least completion time, it follows that

$$\forall K \in \{0, \ldots, p\} : T_{hybrid}(m, K_{opt}) \leq T_{hybrid}(m, K) \tag{9}$$

It will be shown that

$$\forall m' > m : \forall K < K_{opt} : T_{hybrid}(m', K_{opt}) < T_{hybrid}(m', K) \tag{10}$$

In other words, choosing $K < K_{opt}$ for $m' > m$ cannot be an optimal choice as it results in a larger completion time than just retaining the old optimal value K_{opt}. The theorem follows from this.

The inequality 10 is proven by considering three distinct cases. For $K_{opt} = 0$, there is no $K < K_{opt}$, so that it is vacuously true. For $0 \leq K < K_{opt} < p$, we have

$$\begin{aligned} T_{hybrid}(m', K_{opt}) = {} & \{2(p - K_{opt}) + 2^{3K_{opt}-2}\}t_s + (2^{3p-1} - 2^{2p+K_{opt}-2})m't_p \\ & + \{(p - K_{opt} - 1)2^{2p-1} + 2^{2p-2K_{opt}}\}m't_{mem} + (p - K_{opt} - 1 + 2^{3K_{opt}-2})b \end{aligned}$$

$$= T_{hybrid}(m, K_{opt}) + (2^{3p-1} - 2^{2p+K_{opt}-2})(m'-m)t_p + \{(p-K_{opt}-1)2^{2p-1} + 2^{2p-2K_{opt}}\}(m'-m)t_{mem} \quad (11)$$

A similar expression can be derived for $T_{hybrid}(m', K)$ as

$$\begin{aligned} T_{hybrid}(m', K) = {} & \{2(p-K) + 2^{3K-2}\}t_s + (2^{3p-1} - 2^{2p+K-2})m't_p \\ & + \{(p-K-1)2^{2p-1} + 2^{2p-2K}\}m't_{mem} + (p-K-1+2^{3K-2})b \end{aligned}$$

which can be rewritten as

$$T_{hybrid}(m', K) = T_{hybrid}(m, K) + (2^{3p-1} - 2^{2p+K-2})(m'-m)t_p + \{(p-K-1)2^{2p-1} + 2^{2p-2K}\}(m'-m)t_{mem} \quad (12)$$

Since $K < K_{opt}$, the cofficient of $(m'-m)t_p$ for $T_{hybrid}(m', K_{opt})$ in equation 11 is smaller than that for $T_{hybrid}(m', K)$ in equation 12. Similarly, the coefficient of $(m'-m)t_{mem}$ is also smaller with $T_{hybrid}(m', K_{opt})$ than with $T_{hybrid}(m', K)$. From these observations and from inequality 9, it follows that $T_{hybrid}(m', K) > T_{hybrid}(m', K_{opt})$.

Finally, for the case where $0 \leq K < K_{opt} = p$, we have

$$T_{hybrid}(m', K_{opt}) = T_{hybrid}(m, K_{opt}) + 2^{3p-2}(m'-m)t_p$$

Here again, the cofficient of $(m'-m)t_p$ is smaller than that for $T_{hybrid}(m', K)$, while there is no term containing t_{mem} unlike equation 12. Inequality 10 follows from this. Q.E.D. #

Theorem 2: Increasing t_s cannot increase K_{opt}, all other factors being equal (that is, the DE component of the hybrid will decrease or stay the same when the message startup time increases).

Proof: Let $T_{hybrid}(t_s, K)$ be the completion time of a hybrid algorithm where all parameters except t_s and K are held constant. From the definition of K_{opt}, it follows that

$$\forall K \in \{0, \ldots, p\} : T_{hybrid}(t_s, K_{opt}) \leq T_{hybrid}(t_s, K) \quad (13)$$

It will be shown that

$$\forall t'_s > t_s : \forall K > K_{opt} : T_{hybrid}(t'_s, K_{opt}) < T_{hybrid}(t'_s, K) \quad (14)$$

From this, the theorem follows by the same reasoning as in Theorem 1.

The inequality 14 is proven by considering three distinct cases. For $K_{opt} = p$, there is no $K > K_{opt}$, so that it is vacuously true. For $0 < K_{opt} < K < p$, we have

$$T_{hybrid}(t'_s, K_{opt}) = T_{hybrid}(t_s, K_{opt}) + \{2(p - K_{opt}) + 2^{3K_{opt}-2}\}(t'_s - t_s) \tag{15}$$

and

$$T_{hybrid}(t'_s, K) = T_{hybrid}(t_s, K) + \{2(p - K) + 2^{3K-2}\}(t'_s - t_s) \tag{16}$$

Consider the difference of the coefficients of the $t'_s - t_s$ term in the two expressions above: $(2^{3K-2} - 2^{3K_{opt}-2}) + 2(K_{opt} - K)$. For $K > K_{opt} \geq 1$, this difference is positive. So, the coefficient in expression 16 is larger than that in expression 15. Together with inequality 13, this implies that $T_{hybrid}(t'_s, K) > T_{hybrid}(t'_s, K_{opt})$.

Finally, the case $K_{opt} = 0$ is considered.

$$T_{hybrid}(t'_s, K_{opt}) = T_{hybrid}(t_s, K_{opt}) + 2p(t'_s - t_s)$$

Here too, the coefficient of the $t'_s - t_s$ term is smaller than that in expression 16. Hence $T_{hybrid}(t'_s, K) > T_{hybrid}(t'_s, K_{opt})$. Q.E.D. #

Theorem 3: For sufficiently large m, increasing p cannot decrease K_{opt}, all parameters except p being equal (that is, the DE component of the hybrid will increase or stay the same when the mesh size is increased, provided the block size is suffciently large).

Proof: Let $T_{hybrid}(p, K)$ be the completion time of a hybrid algorithm where all parameters except p and K are held constant. It needs to be shown that

$$\exists m_c \text{ s.t. } \forall m > m_c : \forall p' > p : \forall K < K_{opt} : T_{hybrid}(p', K_{opt}) < T_{hybrid}(p', K) \tag{17}$$

By the same reasoning as in Theorem 1, this would imply the theorem.

The inequality above is proven by considering three distinct cases. For $K_{opt} = 0$, there is no $K < K_{opt}$, so that the inequality is vacuously true. For the case $0 < K < K_{opt} \leq p$, we have

$$\begin{aligned} T_{hybrid}(p', K) - T_{hybrid}(p', K_{opt}) = {} & \{(2^{3K-2} - 2^{3K_{opt}-2}) + 2(K_{opt} - K)\}t_s + (2^{2p'+K_{opt}-2} - 2^{2p'+K-2})mt_p \\ & + \{(2^{2p'-2K} - 2^{2p'-2K_{opt}}) + 2^{2p'-1}(K_{opt} - K)\}mt_{mem} + \{(2^{3K-2} - 2^{3K_{opt}-2}) + (K_{opt} - K)\}b \end{aligned}$$

Since $K < K_{opt}$, the coefficients of mt_p and mt_{mem} are strictly positive. Also, in the coefficients of t_s

and b, the terms involving $K_{opt} - K$ are positive. The only negative component in the above expression is $(2^{3K-2} - 2^{3K_{opt}-2})(t_s + b)$. Note that this component does not involve m while two of the possitive terms do involve it. Thus, for a sufficiently large m, the above difference will be strictly positive i.e. $T_{hybrid}(p', K_{opt}) < T_{hybrid}(p', K)$.

Finally, in the case $0 = K < K_{opt} \leq p$, the difference would be

$$T_{hybrid}(p', 0) - T_{hybrid}(p', K_{opt}) = (2K_{opt} - 2^{3K_{opt}-2})t_s + 2^{2p'+K_{opt}-2}mt_p + (K_{opt} - 2^{3K_{opt}-2})b \\ + (2^{2p'-1}K_{opt} - 2^{2p'-2K_{opt}})mt_{mem}$$

The terms involving m are strictly positive and those not involving it are nonpositive. So, for sufficiently large m, the difference would be strictly positive i.e. $T_{hybrid}(p', K_{opt}) < T_{hybrid}(p', K)$. Q.E.D. #

Theorem 4: Increasing b cannot increase K_{opt}, all other factors being equal (that is, the DE component of the hybrid will decrease or stay the same when the cost of a barrier synchronization is increased).

Proof: Let $T_{hybrid}(b, K)$ be the completion time of a hybrid algorithm where all parameters except b and K are held constant. From the definition of K_{opt}, it follows that

$$\forall K \in \{0, \ldots, p\} : T_{hybrid}(b, K_{opt}) \leq T_{hybrid}(b, K) \tag{18}$$

It will be shown that

$$\forall b' > b : \forall K > K_{opt} : T_{hybrid}(b', K_{opt}) < T_{hybrid}(b', K) \tag{19}$$

From this, the theorem follows by the same reasoning as in Theorem 1.

The inequality 19 is proven by considering three distinct cases. For $K_{opt} = p$, there is no $K > K_{opt}$, so that it is vacuously true. For $0 < K_{opt} < K < p$, we have

$$T_{hybrid}(b', K_{opt}) = T_{hybrid}(b, K_{opt}) + (p - K_{opt} - 1 + 2^{3K_{opt}-2})(b' - b) \tag{20}$$

and

$$T_{hybrid}(b', K) = T_{hybrid}(b, K) + (p - K - 1 + 2^{3K-2})(b' - b) \tag{21}$$

For $K > K_{opt} > 0$, $2^{3K-2} - K > 2^{3K_{opt}-2} - K_{opt}$. So, the coefficient of the $b' - b$ term is bigger in expression 21 than in expression 20. This observation, together with inequality 18 show that

$T_{hybrid}(b', K_{opt}) < T_{hybrid}(b', K)$.

Finally, considering the case $K_{opt} = 0$, we have

$$T_{hybrid}(b', K_{opt}) = T_{hybrid}(b, K_{opt}) + (p-1)(b' - b)$$

where the coefficient of the $b' - b$ term is again smaller than with expression 21. Hence, $T_{hybrid}(b', K_{opt}) < T_{hybrid}(b', K)$. Q.E.D. #

Table 2: Variation in completion times of CE and DE, and also in K_{opt}, as the parameters of complete exchange are varied one at a time. Among CE and DE, the one whose finish times increases the most is indicated with $\Uparrow$.

Factor being varied	CE Finish Time	DE Finish Time	K_{opt}
$m \uparrow$	$\Uparrow$	$\uparrow$	$\uparrow$
$t_s/t_p \uparrow$	$\uparrow$	$\Uparrow$	$\downarrow$
$p \uparrow$	$\Uparrow$	$\uparrow$	$\uparrow$
$b \uparrow$	$\uparrow$	$\Uparrow$	$\downarrow$

These trends are summarized in Table 2.

In the following section, these trends predicted by the model are validated with a simulation study.

5 Simulation results and discussion

The family of hybrid algorithms needs to be evaluated to find the least completion time among the hybrids. The evaluation, in general, may be carried out using analysis, simulation or measurement of an implementation on a multicomputer. All the three avenues were pursued as they have their own merits.

Analytical models were developed for Direct Exchange (DE) and Cyclic Exchange (CE). The model for DE assumes barriers between steps (Section 2.2). Three models were developed for CE: one that assumes no barriers, one assuming a barrier between steps, and one with barrier between phases. The former two were implemented and studied, whereas the latter was used primarily for its simplicity of analysis. These were, in turn, used to develop a quantitative model of the hybrid algorithms. This model can be used to predict the best hybrid in the family for specific combinations of system factors and the problem size.

The predictions of the analytical model can be verified either with a simulation or with an implementation on a multicomputer. Simulation has the advantage that the architectural parameters can be systematically varied, which would not be possible in a real machine. An implementation, however, would provide concrete proof that the scheme is practically useful. So, both these methods were employed. The results of the simulation study are presented in this section and those of the implementation in Section 8.

The performance of CE, DE and their hybrids, with and without the use of barriers, were studied using simulation. The simulator was written in C, using the CSIM simulation package [20]. It models a wormhole-routed network with k-ary n-cube topology, at the flit level in detail. The simulator reports the completion time, the average latency of all messages and the average ideal (contention-free) latency.

Several parameters were varied for the simulation study using a combinatorial design. The system parameters are p, a measure of the size of the $2^p \times 2^p$ mesh; t_s, the message startup cost; and b, the barrier cost. The problem size is the number m of bytes in each data block. The hybrids were investigated under three cases:

- without barriers,
- with barriers that utilize the hardware support of the underlying architecture to reduce barrier cost, if such support is available
- with barriers implemented in software, should the underlying machine not have support for barrier synchronization.

The following values were held constant: t_p at $0.012\mu s$ and t_{mem} at $0.007\mu s$.

The objective of the study was to compare hybrid algorithms with different K values and determine:

- if the hybrid algorithms perform better than the pure algorithms for some parameter values,
- the best value of K for a given set of system parameters and problem size, and
- the accuracy of the analytical models.

The following tables and plots summarize the results of the study.

5.1 Results with hardware barrier

For 16×16 and larger meshes, the high degree of contention degrades performance. This contention can be controlled by imposing a barrier between successive phases of CE or successive steps of DE.

Table 3: The actual (top row) and predicted (bottom row) completion times for various hybrids with hardware barriers, with barrier cost of $1\mu s$. The numbers in bold correspond to the best K for that block size.

a. 16×16 mesh, $t_s/t_p = 150$

K	Data Block Size (in bytes)					
	8	32	128	1024	4096	16384
0	248	924	3627	28859	115366	461396
	248	923	3627	28858	115366	461396
1	**239**	886	3472	27614	110386	441474
	240	886	3473	27615	110387	441475
2	246	**808**	3055	24029	95938	383576
	245	807	3054	24028	95937	383574
3	555	1019	**2878**	20221	79682	317529
	554	1019	2877	20220	79682	317528
4	3574	3869	5049	**16059**	**53807**	**204802**
	3578	3873	5053	16063	53812	204806
5	—					
	—					

b. 32×32 mesh, $t_s/t_p = 7000$

K	Data Block Size (in bytes)					
	8	32	128	1024	4096	16384
0	**2570**	7719	28313	220530	879560	3515680
	2567	7716	28310	220527	879558	3515678
1	2614	**7646**	27774	215633	859720	3436072
	2612	7644	27772	215631	859719	3436070
2	3441	8133	**26903**	202089	802726	3205276
	3437	8129	26899	202085	802723	3205272
3	12694	16997	34211	**194875**	745721	2949107
	12691	16994	34208	194872	745718	2949104
4	89030	92656	107159	242516	**706596**	2562920
	89032	92658	107160	242517	706598	2562922
5	707210	709570	719007	807087	1109077	**2317037**
	707250	709610	719047	807127	1109117	2317077

This is found to improve performance, as demonstrated below. This solution is attractive because many modern multicomputers, such as the Cray T3D, offer hardware support for barriers, so that the cost of a barrier is low. This subsection focuses on the performance of hybrid algorithms using such fast hardware barriers. The cost of a barrier is held fixed at $b = 1\,\mu s$.

Table 3 presents the data for hybrids with barriers between successive steps for two mesh sizes and two startup ratios. The startup ratio of 7000 is representative of systems available widely today, while the ratio of 150 is a low value that may be expected in future systems. The top row for each K gives the actual completion time of the simulation while the bottom row provides the completion times predicted by the analytical model. Note that $K = 0$ corresponds to CE while $K = p - 1$ corresponds to DE.

It can be seen from Table 3(a) for a 16×16 mesh that there is very close correspondence between analytically predicted and observed results. The model predicts that, for a fixed mesh size and startup time, a hybrid algorithm should take longer to finish as m increases, due to the increase in transmission time component. This is indeed the case. Theorem 1 derived from the model also predicts that, as m increases, the best hybrid would tend to have an increased DE component i.e. the best K, denoted K_{opt}, would tend to increase. This is also confirmed. Table 3(b) confirms these trends for a larger system size of 32×32. The model has been verified to be accurate for a large range of mesh size, block size and startup ratios, with and without barriers.

Table 4 illustrates the variation of K_{opt} with m, t_s and p. Increasing m while holding the other two constant leads to an increase in K_{opt}, as already noted. If t_s is increased, all other factors being equal, K_{opt} tends to decrease, which is in accordance with Theorem 2. Finally, if p alone were increased, K_{opt} tends to increase, favoring DE, as stated in Theorem 3. This is in agreement with the predictions of the model.

Table 4: The K of the best hybrid with hardware barriers, with barrier cost of $1 \mu s$.

a. 8×8 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	1	1	2	3	3	3
1000	0	0	1	2	3	3
7000	0	0	0	1	2	3

b. 16×16 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	1	2	3	4	4	4
1000	0	1	2	3	4	4
7000	0	0	1	2	3	4

c. 32×32 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	2	3	4	5	5	5
1000	1	2	3	4	5	5
7000	0	1	2	3	4	5

5.2 Results with software barriers

The results and analysis presented so far assume a constant barrier cost of $1\,\mu s$. If the target machine does not provide hardware support for barriers, or if the processors involved in the complete exchange are not contiguous, the barrier would have to be implemented in software. Barrier synchronization algorithms are usually tree-structured, with a reporting stage and a wakeup stage. On a $2^p \times 2^p$ mesh, each of these stages take a time of $2pt_s$, for a total of $4pt_s$. This higher cost of barriers will affect the completion times of the algorithms. However, the impact on CE will be much lower than on DE, as the former employs far fewer barriers. This, in turn, implies that the hybrid with lower DE component will be favored when the barrier cost is high i.e. K_{opt} would tend to decrease (as proved in Theorem 4). This subsection presents and analyzes the performance of hybrid algorithms with software barriers.

For Direct Exchange (DE) on a $2^p \times 2^p$ $(p > 1)$ mesh, there are 2^{3p-2} steps so that the total cost of barriers is $(2^{3p-2} - 1).4pt_s$. For Cyclic Exchange (CE), the recursive structure implies that the parameter p decreases in successive phases. The total barrier cost is $2.\{4pt_s + 4(p-1)t_s + \ldots + 4.2.t_s\}$, since there is there is no need for barriers in the last 2×2 phase. The factor of 2 arises because there are two barriers in each phase. This sum evaluates to $\{p(p+1) - 2\}4t_s$. The barrier cost for hybrids for $0 < K < p$ can be calculated along similar lines. (Note that Theorem 4 would still hold true with this modified cost model for barrier synchronization since the number of barriers in DE is much higher than with CE.)

The completion times of the hybrids with software barriers were determined and the value of K_{opt} found, for each combination of p, m and the t_s/t_p ratio. The results are presented in Table 5 and exhibit similar variation with p, m and t_s/t_p as the hardware barrier case (Table 4), thus confirming the analytically proven theorems.

A software barrier is more expensive than a hardware barrier. Since DE uses more barriers than CE, it would be expected that the DE component of the best hybrid i.e. K_{opt} would decline when software barriers replace hardware barriers. Also, this decrease should be more marked at large values of m and p, as these conditions lead to a high K_{opt}. Furthermore, as software barrier cost rises with t_s, K_{opt} may be expected to decline at high t_s values also.

A comparison of Table 5(a) with Table 4(a) shows that K_{opt} values tend to decrease when software barriers replace hardware barriers, especially for large t_s. Comparing all three subtables of Table 5 and Table 4 reveals that the decrease of K_{opt} with software barriers is more pronounced for larger meshes. Thus, the predictions are borne out by the simulation study.

To summarize, hybrid algorithms do quite well when barriers are used to control contention and, in fact, are the optimal algorithms for a large range of parameters. Also, the analytical model is useful in choosing the best K for a given set of parameters.

5.3 Results without barrier

The performance of the hybrids in the absence of barriers in both the CE and DE components is reported in this section.

Table 6 presents the K for the best hybrid without barriers for three different mesh sizes and startup

Table 5: The K of the best hybrid with software barriers.

a. 8×8 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	1	1	1	2	3	3
1000	0	0	1	1	2	2
7000	0	0	0	1	1	2

b. 16×16 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	1	2	3	3	4	4
1000	0	1	1	2	2	3
7000	0	0	1	1	1	2

b. 32×32 mesh

	Data Block Size (in bytes)					
t_s/t_p	8	32	128	1024	4096	16384
150	1	1	2	3	4	5
1000	1	1	1	2	3	4
7000	0	1	1	1	2	3

ratios. Though all the combinations of factors were studied, we present the data for only some selected combinations. The optimal algorithm for a large range of parameters is a hybrid algorithm, thus establishing that hybridizing is quite useful in the absence of barriers also.

Note that, although system size, block size and startup cost are varied by several orders of magnitude, the best hybrid tends to be $K = 3$. This is contrary to the trend in the case of hybrids with barriers, where K_{opt} tends to increase with m and p, thus favoring DE. This surprising pattern suggests a rule of thumb to pick the best hybrid for the nonbarrier case: choose $K_{opt} = 3$ for $p > 2$. The reason for this is discussed in the next section.

6 Impact of contention on finish time

Wormhole routing reserves mesh links. When a message is forced to wait for a link to become available, the links which it holds are not released, causing other messages which need those links to block. This may have the effect of increasing message latency enormously and degrading performance. This section quantifies the degree of contention and explores its impact on the finish time of complete exchange.

Table 6: The K-value of the best hybrid without barriers.

		Data Block Size (in bytes)					
mesh	t_s/t_p	8	32	128	1024	4096	16384
8×8	150	1	2	3	3	3	3
	1000	0	0	1	3	3	3
	7000	0	0	0	2	3	3
16×16	150	2	3	3	3	3	3
	1000	0	2	3	3	3	3
	7000	0	0	1	3	3	3
32×32	150	3	3	3	3	3	3
	1000	2	3	3	3	3	3
	7000	0	1	3	3	3	3

The degree of contention in the execution of an algorithm is measured by the ratio of average actual latency of all messages in the algorithm to the average ideal (contention-free) latency of all messages. This ratio, called the *Contention Factor*, is meaningful only when all the messages in the algorithm are of the same length. This is the case for both the pure algorithms Cyclic Exchange (CE) and Direct Exchange (DE), though not for the hybrids. Thus, Figure 9 plots the contention factors only for the pure algorithms.

Figure 9 clearly shows the steep increase in contention in DE as m increases. This happens for a range of startup costs, from $1\mu s$ to $50\mu s$. In contrast, the contention factor for CE remains remarkably stable even when m varies by 4 orders of magnitude. This accounts for the observation in Section 5.3 that, in the absence of barriers, the best hybrid algorithm always has 3 or fewer phases replaced by Direct Exchange (DE). This is due to the fact that, as m increases, contention in DE increases steeply, thus offsetting any savings due to lower transmission volume. Thus, the best hybrid does not have a large DE component, even for large m.

Another striking feature of the plot in Figure 9 is that the contention factor for DE is lower when startup time is higher i.e. higher message startup cost has the effect of decreasing network contention. This is because increasing the startup time increases the interval between injection of messages into the network by each processor. This has the effect of decreasing the offered load (number of messages per unit time) on the network, thus leading to lower latency.

This leads to the counter-intuitive observation that *increasing* the startup cost can *decrease* the completion time by reducing contention. This is illustrated in Table 7a, which presents the completion times for DE with and without barriers for several values of m and t_s/t_p. In the absence of barriers,

Table 7: Comparison of completion times with and without hardware barriers for Direct Exchange and Cyclic Exchange on a 32×32 mesh.

a. Direct Exchange

		Data Block Size (in bytes)					
t_s/t_p	barrier	8	32	128	1024	4096	16384
150	yes	33828	36187	45625	133705	435695	1643654
	no	3798	20650	73020	549395	2190788	8727527
7000	yes	707210	709570	719007	807087	1109077	2317037
	no	86942	87787	91457	367076	2114656	8867861

b. Cyclic Exchange

		Data Block Size (in bytes)					
t_s/t_p	barrier	8	32	128	1024	4096	16384
150	yes	1748	6897	27491	219708	878738	3514858
	no	2527	10035	40067	320365	1281384	5125464
7000	yes	2570	7719	28313	220530	879560	3515680
	no	2786	9426	44042	321269	1282289	5126368

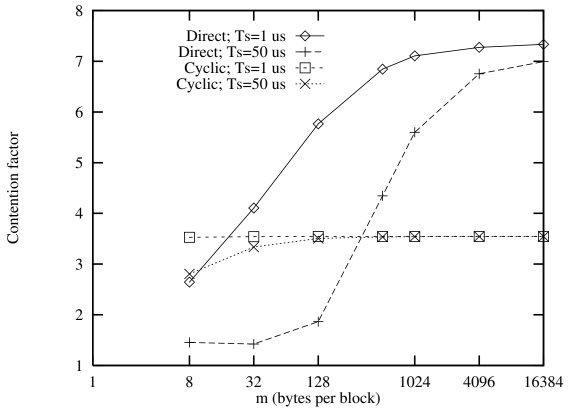

Figure 9: Variation of contention factor with block size for Cyclic Exchange and Direct Exchange in the absence of barriers. Contention factor is defined as the ratio of average actual latency of all messages in the algorithm to the average ideal (contention-free) latency.

for $m = 1024$ or 4096, increasing the startup ratio from 150 to 7000 decreases the completion time for DE, although it has a large number of startups. This seemingly strange behavior can be explained by noting the strong influence of contention on message latencies in DE. This is seen to happen with CE also, but only for $m = 32$ (Table 7b).

The high level of contention in the no-barrier version also explains why it is beneficial to impose barriers. The decrease in finish time due to reduction or the elimination in contention often offsets the increase due to the cost of the barriers. This is illustrated in Table 7. The performance of DE improves when hardware barriers are imposed for $m \geq 128$ at $r = 150$ and for $m \geq 4096$ at $r = 7000$. For low m, contention is not high; so, the improvement in performance by eliminating contention is outweighed by the cost of barriers. At higher values of m, the high degree of contention makes barriers useful. Also, barriers are more beneficial at lower startup costs, because contention is higher in that regime. For CE, the barrier version performs better than the nonbarrier version for a large range of m and for widely differing startup costs. This is consistent with the fact that the contention factor for CE hardly varies with m or t_s/t_p.

7 Influence of barriers on K_{opt}

Table 7 validates the claim that using barriers can potentially improve performance. The degree to which they enhance performance depends on the cost of the barriers, which in turn is influenced by the availability of hardware support for barrier synchronization. This subsection investigates the effectiveness of barrier usage in the presence and absence of hardware support for barrier synchronization and also the impact on the K-value of the best hybrid. As noted earlier, the data for 8×8 mesh assume that barriers are used only between phases, while they are employed between successive steps (not phases) for larger meshes.

7.1 Impact of hardware barriers

Table 8 presents data on the best hybrid to use and whether or not it uses hardware barriers, for several combinations of p, m and t_s/t_p. The barrier cost is fixed at $1\mu s$.

Table 8(a) shows that using hardware barriers is advantageous at larger values of m for 8×8 meshes. Furthermore, the value of m at which it becomes better to use barriers tends to increase with t_s/t_p. This is explained by noting that large t_s values serve to reduce contention, as discussed in Section 6, so that barriers are not as useful.

Table 8: The best K and hardware barrier usage for various p, m and t_s/t_p. The letter "h" denotes the use of hardware barriers while "n" indicates absence of barriers.

a. 8×8 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	1 n	2 n	3 n	3 h	3 h	3 h
1000	0 n	0 n	1 n	3 n	3 h	3 h
7000	0 n	0 n	0 n	2 n	3 n	3 h

b. 16×16 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	1 h	2 h	3 h	4 h	4 h	4 h
1000	0 h	1 h	2 h	3 h	4 h	4 h
7000	0 n	0 n	1 h	2 h	3 h	4 h

c. 32×32 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	2 h	3 h	4 h	5 h	5 h	5 h
1000	1 h	2 h	3 h	4 h	5 h	5 h
7000	0 h	1 h	2 h	3 h	4 h	5 h

From Table 8(b), it can be seen that it is better to employ barriers for 16×16 meshes except for $t_s/t_p = 7000, m \leq 32$. This exception arises because, at large t_s and small m, the degree of contention is too small to justify the use of barriers. However, the barrier version takes only 3.7% extra time for $m = 8$ and 7.1% more for $m = 32$. Finally, the data of Table 8(c) for 32×32 meshes show that it is uniformly better to use hardware barriers.

Thus, use of hardware barriers leads to better performance in most cases; for relatively small mesh sizes, high t_s and small m, it increases the finish time by only a small percentage. So, if the target system provides hardware support for barrier synchronization, it is a good strategy to use it in all cases.

7.2 Impact of software barriers

When barriers need to be implemented in software because of lack of architectural support for barrier synchronization, the decision on their usage is not as clearcut as with hardware barriers. This is illustrated further in the following subsection.

Table 9 presents the choice of best K and whether or not it uses software barriers, for several combinations of p, m and t_s/t_p. Part (a) shows that software barriers are useful only when t_s is low and m is high for 8×8 meshes. At higher values of t_s, the cost of a software barrier is so high that it

Table 9: The best K and software barrier usage for various p, m and t_s/t_p. The letter "s" denotes the use of software barriers while "n" indicates absence of barriers.

a. 8×8 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	1 n	2 n	3 n	3 n	3 s	3 s
1000	0 n	0 n	1 n	3 n	3 n	3 n
7000	0 n	0 n	0 n	2 n	3 n	3 n

b. 16×16 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	2 n	3 n	2 s	3 s	4 s	4 s
1000	0 n	2 n	3 n	2 s	2 s	3 s
7000	0 n	0 n	1 n	3 n	1 s	2 s

c. 32×32 mesh

t_s/t_p	8	32	128	1024	4096	16384
150	1 s	1 s	2 s	3 s	4 s	5 s
1000	2 n	1 s	1 s	2 s	3 s	4 s
7000	0 n	1 n	3 n	1 s	2 s	3 s

is better to dispense with it. Similarly, at low m, the level of contention is not high enough to accept the penalty of barriers.

It is observed in Table 9(b) for 16×16 meshes also that software barriers are beneficial at higher m. The crossover point at which barriers become useful is seen to shift towards higher m, as t_s increases, because large t_s tends to reduce contention, thus making barriers less useful.

The same trend is also seen in part (c) for 32×32 meshes. Contrasting parts (b) and (c) reveals that the crossover point tends to decrease as p increases, as the level of contention is higher in larger meshes and barriers reduce it.

Summarizing, barriers can be effective even when they need to be implemented in software. Software barriers are most useful for larger meshes, lower startup costs and higher data block sizes. As the overall trend is towards systems with lower message startup time and increasing number of processors, and towards larger applications, barriers may be expected to be useful for complete exchange even in the absence of architectural support for them.

Table 10: The K of the best performing hybrid on the Cray T3D. Note that 8×8 mesh is embedded in a $8 \times 4 \times 4$ torus.

logical mesh size	Data block size							
	8	32	128	256	512	1024	4096	16384
2×2	0	0	0	1	1	1	1	1
4×4	0	0	1	1	2	2	2	2
8×8	0	1	2	2	3	3	3	3

8 Experimental results

The family of hybrid algorithms were implemented on the Cray T3D system at the Ohio Supercomputer Center. It is a distributed memory machine with the interconnect being a 3-dimensional torus. This installation of T3D has 128 processing nodes, structured as $8 \times 4 \times 4$. The message-passing model is supported on the T3D with a native implementation of the Parallel Virtual Machine (PVM) [12] communication library. So, the hybrids were implemented using PVM. Certain factors had to be taken care of in the implementation; these are described in the Appendix.

The K_{opt} of the hybrids that had the best performance for a given mesh size and block size are presented in Table 10. For a 2×2 mesh, CE does best for block sizes $8 \leq m \leq 128$, but for larger m, DE fares better. Thus, a hybrid model that switches from one to the other as appropriate would have the best performance for the entire range of m. For 4×4 meshes, CE does best for $8 \leq m \leq 32$. For $128 \leq m \leq 256$, the best algorithm is neither CE nor DE, but a hybrid algorithm; for larger m, DE fares better. Similarly, for 8×8 meshes also, the hybrid model switches smoothly from CE to DE through various intermediate hybrids.

These observations are consistent with the trends seen in the simulation study and the theorems derived from the analytical model. CE tends to do well at lower block sizes since the savings in message startup costs offset the higher volume of data transmission. DE fares well at larger block sizes since the volume of data handled by the network becomes important in this regime. The DE component tends to increase when the mesh size increases and the block size is held constant, as observed in the simulation study and the analytical model.

Thus, the experimental results from the implementation of the hybrid scheme on the Cray T3D multicomputer confirms and validates the thesis that exploiting the complementary strengths of combining and direct algorithms leads to lower execution times across a broad range of mesh sizes and block sizes.

9 Related Work

Considerable work has been done to find good solutions for the problem of performing complete exchange efficiently, since it occurs in numerous practically relevant applications such as FFT and parallel sorting. A review of past literature and their differences in approach with this work is provided in this section.

The complete exchange problem has been extensively studied on various topologies such as hypercubes [14, 6, 7, 5, 1, 19, 26] and tori [23]. Solutions have also been formulated for linear arrays and rings [10] and other architectures [16, 18] with circuit-switching. Several studies have focussed on the 2D mesh architecture [9, 21, 28, 27, 24].

Another class of studies has aimed at an efficient solution to complete exchange on specific parallel computers [29, 6, 19]. Many papers have analyzed the problem in the context of specific applications such as parallel sorting [2, 1, 15, 22].

Efficient implementation of complete exchange is a significant issue for designers of collective communication libraries. The question of performing collective communication efficiently has been addressed in [3, 4]. A survey of collective communication algorithms, including those for complete exchange, is given in [17].

Multiphase algorithms have been proposed for the hypercube [7] and these same algorithms have been implemented on a 2D mesh [8].

This paper has several features that distinguish it from past studies. It hybridizes two algorithms that have been specifically formulated for meshes: Cyclic Exchange and Direct Exchange. It also provides an analytical model, unlike earlier works, which helps in choosing the best solution out of a family of hybrid algorithms. The predictions of the analytical model have been verified by both simulation and implementation on a Cray T3D to be accurate. One of the unique features of the paper is the accurate modeling of message contention on a 2D mesh when barrier synchronization is employed, in contrast to many previous studies where the time loss due to contention is quantified in an approximate manner. It also demonstrates that link contention is an important factor in determining the finish time of complete exchange. This leads to two other novel observations. The first is that the finish time can be reduced for many parameter ranges by controlling contention through deploying barrier synchronizations between steps. The other is the counter-intuitive observation that increased message startup overhead can reduce the completion time by decreasing message contention. These observations are quantified and studied with the analytical model and further confirmed with a simulation study.

10 Conclusions

This paper has proposed a scheme to hybridize the best known combining algorithm (Cyclic Exchange) and the best direct algorithm (Direct Exchange) for complete exchange in a way that effectively exploits their complementary merits. Analytical models have been proposed that quantify the influence of various factors on the completion time of CE (without barriers, with barriers between phases and with barriers between steps), DE (with barriers between steps) and their hybrids. The best hybrid for specified sets of system parameters and problem size can be chosen using the analytical model (when barriers are used) or a thumb rule (when barriers are not used). The hybrids were shown to perform better than previously known efficient algorithms for complete exchange for a large range of system and problem parameters, through both a simulation study as well as implementation on a Cray T3D multicomputer.

This paper also brings out the strong influence that link contention among messages has on the performance of complete exchange algorithms. In the case of CE, analytical models are developed that identify patterns of message contention and quantify them. These are used to motivate the use of barriers between algorithmic steps to control contention. The predictions of the analytical model are validated with a simulation study, which further confirm the impact of contention on performance. The direct exchange approach is found to suffer from a large degree of contention in the absence of barriers. This in turn motivates the use of barriers, with or without support for them at the hardware level, to control contention and improve performance. The conditions at which use of hardware and software barriers becomes beneficial are characterized and explained. The counter-intuitive phenomenon of decreased completion time with increased startup cost is identified and explained.

Acknowledgments

The implementation of the hybridization scheme was done on the Cray Research Inc. T3D at the Ohio Supercomputer Center. We are grateful to the staff and consultants there for their technical assistance.

References

[1] B. Abali, F. Özgüner, and A. Bataineh. Balanced parallel sort on hypercube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):572–581, 1993.

[2] David A. Bader, David R. Helman, and Joseph JáJá. Practical parallel algorithms for personalized communication and integer sorting. Technical Report UMIACS TR 95-101, Institute for Advanced Computer Studies, University of Maryland, 1995. `http://www.umiacs.umd.edu/research/EXPAR/papers/3548.html`.

[3] Vasanth Bala, Jehoshua Bruck, Robert Cypher, Pablo Elustondo, Alex Ho, Ching-Tien Ho, Shlomo Kipnis, and Marc Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *Proceedings of the 8th International Parallel Processing Symposium*, pages 835–844, 1994.

[4] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Building a high-performance collective communication library. *Proceedings of the Scalable High Performance Computing Conference*, pages 835–844, 1994.

[5] D.P. Bertsekas, C. Özveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[6] Shahid H. Bokhari. Complete exchange on the iPSC-860. Technical Report 91-4, ICASE, NASA Langley Research Center, January 1991.

[7] Shahid H. Bokhari. Multiphase complete exchange on a circuit switched hypercube. *Proceedings of the International Conference on Parallel Processing*, pages I–525 – I–529, 1991. Also appeared as NASA ICASE Report 91-5. ICASE, NASA Langley Research Center, 1991.

[8] Shahid H. Bokhari. Multiphase complete exchange on Paragon, SP2, and CS-2. *IEEE Parallel & Distributed Technology*, 4(3):45–59, Fall 1996.

[9] Shahid H. Bokhari and H. Berryman. Complete exchange on a circuit switched mesh. *Proceedings of the Scalable High Performance Computing Conference*, pages 300–306, 1992.

[10] V.V. Dimakopoulos and N.J. Dimopoulos. Optimal total exchange in linear arrays and rings. *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks.*, pages 230–237, 1994.

[11] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical report, University of Tennessee, Knoxville, TN, June 1995. Version 1.1. `http://www.mcs.anl.gov/mpi/mpi-report-1.1/mpi-report.html`.

[12] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine — A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[13] Satya Gupta, Stuart Hawkinson, and Bruce Baxter. A binary interleaved algorithm for complete exchange on a mesh architecture. Technical report, Intel Corporation, 1993. Personal Communication.

[14] S. Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[15] M. Kaufmann, J.F. Sibeyn, and T. Suel. Derandomizing algorithms for routing and sorting on meshes. *Proceedings of the 5th Symposium on Discrete Algorithms*, pages 69–79, 1994.

[16] Y.-D. Lyuu and E. Schenfeld. Total exchange on a reconfigurable parallel architecture. *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, pages 2–10, 1993.

[17] Philip K. McKinley, Yih-jia Tsai, and David F. Robinson. A survey of collective communication in wormhole-routed massively parallel computers. Technical Report MSU-CPS-94-35, Michigan State University, June 1994. `ftp://ftp.cps.msu.edu/pub/crg/PAPERS/msu-cps-94-35.ps.Z`.

[18] S.R. Öhring and S.K. Das. Efficient communication in the folded Petersen interconnection networks. *Proceedings of the 6th International Parallel Architectures and Languages Europe Conference*, pages 25–36, 1994.

[19] T. Schmiermund and S.R. Seidel. A communication model for the intel iPSC/2. Technical Report CS-TR 9002, Dept. of Computer Science, Michigan Tech. University, April 1990.

[20] Herbert Schwetman. Introduction to process-oriented simulation and CSIM. *Proceedings of the Winter Simulation Conference*, 1990.

[21] David S. Scott. Efficient all-to-all communication patterns in hypercube and mesh topologies. *Proceedings of the 6th Conference on Distributed Memory Concurrent Computers*, pages 398–403, 1991.

[22] T. Suel. Routing and sorting on meshes with row and column buses. Technical Report UTA//CS-TR-94-09, Dept. of Computer Sciences, University of Texas at Austin, October 1994.

[23] Young-Joo Suh and Sudhakar Yalamanchili. Algorithms for all-to-all personalized exchange in 2D and 3D tori. *Proceedings of the 10th International Parallel Processing Symposium*, pages 808–814, April 1996.

[24] N.S. Sundar, D.N. Jayasimha, D.K. Panda, and P. Sadayappan. Complete exchange in 2D meshes. *Proceedings of the Scalable High Performance Computing Conference*, pages 406–413, May 1994.

[25] N.S. Sundar, D.N. Jayasimha, D.K. Panda, and P. Sadayappan. Hybrid direct/indirect algorithms for complete exchange in 2D meshes. Technical Report OSU-CISRC-12/95-TR52, Dept. of Computer and Information Science, The Ohio State University, 1995. `ftp://ftp.cis.ohio-state.edu/pub/tech-report/1995/TR52.ps.gz`.

[26] R. Take. A routing method for all-to-all burst on hypercube networks. *Proceedings of the 35th National Conference of Information Processing Society of Japan*, pages 151–152, 1987. (In Japanese).

[27] Rajeev Thakur and Alok Choudhary. All-to-all communication on meshes with wormhole routing. *Proceedings of the 8th International Parallel Processing Symposium*, pages 561–565, April 1994. `http://www.mcs.anl.gov/home/thakur/ipps94-all-to-all.ps`.

[28] Rajeev Thakur, Alok Choudhary, and Geoffrey Fox. Complete exchange on a wormhole routed mesh. Technical Report SCCS-505, Northeast Parallel Architectures Center, Syracuse University, July 1993. `http://www.npac.syr.edu/techreports/html/0500/abs-0505.html`.

[29] Rajeev Thakur, Ravi Ponnusamy, Alok Choudhary, and Geoffrey Fox. Complete exchange on the CM-5 and Touchstone Delta. *Journal of Supercomputing*, 8:305–328, 1995.

Related WWW URLs

- PVM : `http://www.epm.ornl.gov/pvm/pvm_home.html`
- Cray T3D : `http://www.cray.com/PUBLIC/product-info/mpp/CRAY_T3D.html`

A Appendix: Data Movement in Cyclic Exchange

There are issues of organizing data and copying data blocks that arise in any combining algorithm. The time to copy data blocks is an overhead that reduces performance. This appendix presents the details of the scheme used to overlap most of this data movement with communication.

For the sake of clarity, the basic version of Cyclic Exchange, described in section 3, is now called Level 1 Cyclic Exchange.

Data need to be organized appropriately so that, in each communication, the set of data blocks to be sent are contiguous in the data array. This necessitates two kinds of movement of data blocks: reorder between phases and data movement within each phase. In each phase, a processor receives data, some of which is meant for it and some for other processors. So, between phase i and $i-1$, the processor needs to gather the various data blocks needed for communication in phase $i-1$. This process is called reordering and is typical of combining algorithms.

The set of blocks within each node is organized as a linear array. In phase i of the algorithm, the four quadrants of the data array each contain data blocks for processors in the four quadrants of size $2^i \times 2^i$. If every processor organizes its data quadrants in a fixed sequence, some processors will be forced to communicate two quadrants that are not contiguous in the array, forcing a movement of these data blocks to a user buffer. Unless these data movements are overlapped with communication, this is an additional penalty paid by CE for exchanging two quadrants at a time. (Algorithms like Quadrant Exchange [9], which only exchange one quadrant at a time do not have this overhead.) CE achieves a complete overlap of data block movement with communication by a combination of proper arrangement of quadrants in different processors and the use of four types of exchange patterns (see Figure 10). This mechanism, described further below, leads to Level 2 Cyclic Exchange.

The sequence of data quadrants within the array is different for white and black nodes. For white nodes, data quadrants are organized in the following order: upper left (quadrant numbered 0), lower left (quadrant 1) , upper right (quadrant 2), and lower right (quadrant 3). For black nodes, the order is: upper left (quadrant 0), upper right (quadrant 1), lower left (quadrant 2) and lower right (quadrant 3). Since whiteness and blackness of nodes remains invariant in all phases except the last ($i = 0$), two kinds of reorder schemes are needed: White-to-White and Black-to-Black. In the last phase, all nodes are Whites; so black nodes will perform a Black-to-White reorder at the end of phase $i = 1$. It is assumed that every processor has its data in the appropriate format when the complete exchange begins. If they don't, the time of reordering before the start of the algorithm can be reduced by a scheme which is described further below.

The algorithm also uses another data buffer in user space, of size $\frac{3}{4}.2^{2p}m$ (3 quadrants). Consider, for example, the communications of a white node in the upper left quadrant of the submesh (which varies from phase to phase). It needs to send the data blocks in data quadrants UR and LR (numbered 2 and 3)

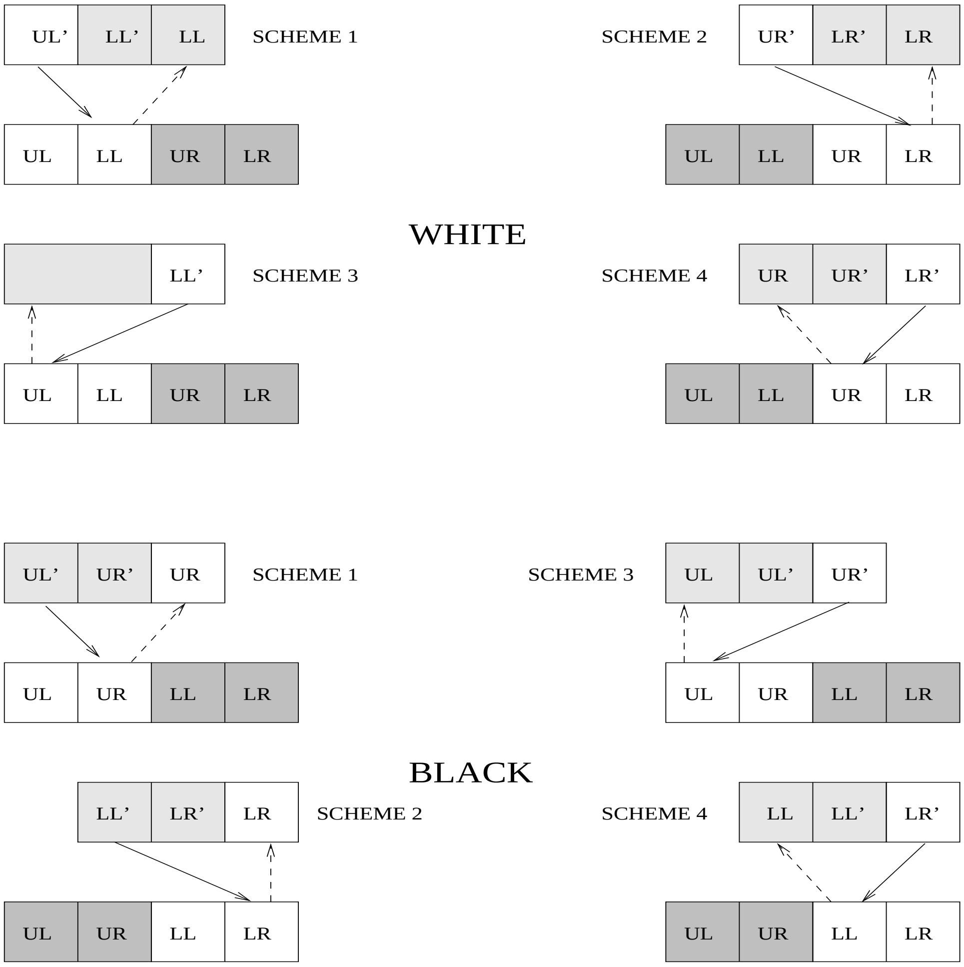

Figure 10: Movement of Data by Processors in the Four Quadrants of the Submesh.

```
{ SPMD program executed by the processor in
  row r and column c. }
algorithm CyclicExchange(r, c).
begin
    for i ← p − 1 downto 0 do { Phase i }
            HorizProc ← (r, c ± 2^i)
            VertProc ← (r ± 2^i, c)
            if ( r mod 2 = c mod 2 ) or ( i = 0 ) {White Node}
                    Partner1 ← HorizProc; Partner2 ← VertProc
            else {Black Node}
                    Partner2 ← HorizProc; Partner1 ← VertProc
            endif
            Step1: Send 2^(2p−1) data blocks (2 quadrants) to Partner1
                    Post a receive from Partner1, with user buffer as destination
                    Gather a data quadrant into user buffer
                    Gather a quadrant into the data array,
                                        from the old copy of the data array
                    Complete the receive from Partner1
            Step2: Exchange 2^(2p−1) data blocks with Partner2
                    Send 2^(2p−1) data blocks (2 quadrants) to Partner2
                    Post a receive from Partner2, with user buffer as destination
                    Gather a data quadrant into user buffer
                    Gather a quadrantinto the data array,
                                        from the old copy of the data array
                    Complete the receive from Partner2
            if ( i > 0 ) perform interphase data reordering of 2^(2p−1)m bytes
    endfor
end
```

Figure 11: The Cyclic Exchange Algorithm for Complete Exchange.

in its first (horizontal) communication (see Figure 10. It receives the data quadrants UL' and LL' from its horizontal counterpart. If these were placed in the data array directly, the data quadrants containing LL and LL' are not contiguous, necessitating data movement for the next (vertical) communication. So, CE places the incoming data quadrants in a user buffer. While the first (horizontal) communication is progressing, the data quadrant LL is moved to the user buffer (see Figure 10) so that it is contiguous with LL'. The vertical send is now initiated from the user buffer, with the received data being placed in the two data quadrants that were used in the previous (horizontal) communication. Concurrent with the vertical communication, the data quadrant UL' in the user buffer is moved to the space occupied by LL earlier in the data array.

To reduce the cost of reordering, our implementation of Cyclic Exchange uses two copies of the data array, and toggles between them in successive phases. This allows each processor to reorder only that half of the array which is needed for the first communication in the next phase. The other half is reordered as the first communication of the next phase progresses. This cuts the non-overlapped component of reordering in half. This addition, which we shall call Level 3, implies that the dashed arrows in Figure 10 really refer to placement of the necessary data blocks in the user buffer from the *old copy* of the data array. Level 3 is the complete version and its algorithm is listed in Figure 11. Note that this scheme can also be used to reduce the initial reordering cost if the data is not in the appropriate format when the algorithm begins.

Thus, the space complexity of Cyclic Exchange is $11\ 2^{2p-2}m$. To see that this is a modest requirement, consider that for a 32×32 mesh ($p = 5$) with $m = 1024$ bytes, the storage required is less than 3 MB, whereas most modern systems provide 16 MB or more per node.

B Appendix: Implementation issues in Cray T3D

Certain issues had to be handled in implementing the hybridization scheme on the T3D. The performance had to be measured for at least an 8×8 mesh since subtle issues involving contention do not arise in smaller meshes. Yet, since this installation had a topology of $8 \times 4 \times 4$ nodes, it is not possible to get a physically square 8×8 partition. This necessitates an embedding of the logically square mesh onto the torus structure. Such an embedding should ideally preserve the contention pattern that would be seen on a physically square mesh, but this is very difficult to do with Direct Exchange (DE) since its contention pattern is hard to characterize, especially in the absence of barrier synchronization between steps. Since Cyclic Exchange (CE) has a predictable pattern of contention, an embedding was chosen so as to minimize the deviation from the pattern of contention that would be seen on a square mesh for Cyclic Exchange. This embedding is done as follows. Number the nodes in the logical square mesh in row-major order and those in the physical machine also in row-major (xyz) order. The mapping from logical node identifiers (i_l) to the physical node numbers (i_p) is:

$$i_l \longrightarrow i_p = \left\{ \begin{array}{cl} i_l & \text{if } i_l \text{ in } 0 \ldots 31 \\ i_l + 32 & \text{else} \end{array} \right\}$$

This embedding has the property that horizontal communications in CE will have the same contention pattern as with a physically square mesh. The vertical communications of CE in the initial phase will, however, face no contention unlike the pattern in the square mesh. The pattern of contention in subsequent phases will be exactly identical to that in a square mesh. Also, CE will never use the wrap-around meshes of the torus, although the same cannot necessarily be expected of DE. It is to be noted that the embedding is necessary only for the 8×8 mesh; it is possible to acquire partitions that are physically 2×2 or 4×4.