Message Scheduling for All-to-all Personalized Communication on Ethernet Switched Clusters

Ahmad Faraj   Xin Yuan
Department of Computer Science, Florida State University
Tallahassee, FL 32306
{faraj, xyuan}@cs.fsu.edu

Abstract

We develop a message scheduling scheme that can theoretically achieve *maximum* throughput for all–to–all personalized communication (AAPC) on any given Ethernet switched cluster. Based on the scheduling scheme, we implement an automatic routine generator that takes the topology information as input and produces a customized *MPI_alltoall* routine, a routine in the Message Passing Interface (MPI) standard that realizes AAPC. Experimental results show that the automatically generated routine consistently out-performs other *MPI_alltoall* algorithms, including those in LAM/MPI and MPICH, on Ethernet switched clusters with different network topologies when the message size is sufficiently large. This demonstrates the superiority of the proposed AAPC algorithm in exploiting network bandwidths.

1 Introduction

All–to–all personalized communication (AAPC) is one of the most common communication patterns in high performance computing. In AAPC, each node in a system sends a different message of the same size to every other node. The Message Passing Interface routine that realizes AAPC is *MPI_Alltoall* [10]. AAPC appears in many high performance applications, including matrix transpose, multi-dimensional convolution, and data redistribution. Since AAPC is often used to rearrange the whole global array in an application, the message size in AAPC is usually large. Thus, it is crucial to have an AAPC implementation that can fully exploit the network bandwidth in the system.

Switched Ethernet is the most widely used local–area–network (LAN) technology. Many Ethernet–switched clusters of workstations are used to perform high performance computing. For such clusters to be effective, communications must be carried out as efficiently as possible. In this paper, we investigate efficient AAPC on Ethernet switched clusters.

We develop a message scheduling scheme that theoretically achieves the maximum throughput of AAPC on any given Ethernet switched cluster. Similar to other AAPC scheduling schemes [4], our scheme partitions AAPC into contention free phases. It achieves the maximum throughput by fully utilizing the bandwidth in the bottleneck links in all phases. Based on the scheduling scheme, we develop an automatic routine generator that takes the topology information as input and produces an *MPI_Alltoall* routine that is customized for the specific topology. We compare the automatically generated routine with the original routine in LAM/MPI [7] and a recently improved *MPI_Alltoall* implementation in MPICH [17]. The results show that the automatically generated routine consistently out-performs the existing algorithms when the message size is sufficiently large, which demonstrates the superiority of the proposed AAPC algorithm in exploiting network bandwidths.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the network model and defines the scheduling problem. Section 4 details the proposed scheduling scheme. Section 5 discusses implementation issues. Section 6 reports experimental results. Finally, the conclusions are presented in Section 7.

2 Related Work

AAPC has been extensively studied due to its importance. A large number of optimal message scheduling algorithms for different network topologies with different network models were developed. Many of the algorithms were designed for specific network topologies that are used in the parallel machines, including hypercube [5, 18], mesh [1, 13, 12, 16], torus [4, 8], k-ary n-cube [18], and fat tree [3, 11]. Heuristic algorithms were developed for AAPC on irregular topologies [9]. A framework for AAPC that is realized with indirect communications was reported in [6]. Efficient AAPC scheduling schemes for clusters connected by a single switch was proposed in [14]. Some of the algorithms in [14] are incorporated in the recent improvement of MPICH library [17]. We consider Ethernet switched clusters with one or more switches. AAPC on such clusters is a special communication pattern on a tree topology. To the best of our knowledge, message scheduling for such cases has not been developed.

3 Network Model and Problem Definition

In an Ethernet switched network, links operates in the duplex mode that supports simultaneous communications on both directions of the link with the full bandwidth. Thus, machines connected to an Ethernet switch can send and receive at the full link speed simultaneously. The switches may be connected in an arbitrary way. However, a spanning tree algorithm is used by the switches to determine forwarding paths that follow a tree structure [15]. As a result, regardless of how the switches are connected, the physical topology of the network is always a **tree** with switches being the internal nodes and machines being leaves. There is always a unique path between any two nodes in the network.

The network can be modeled as a directed graph $G = (V, E)$ with nodes V corresponding to switches and machines and edges E corresponding to unidirectional channels. Let S be the set of switches in the network and M be the set of machines in the network. $V = S \cup M$. Let $u, v \in V$, a directed **edge** $(u, v) \in E$ if and only if there is a link between node u and node v. We will call the physical connection between node u and node v **link** (u, v). Thus, link (u, v) corresponds to two directed edges (u, v) and (v, u) in the graph. Since the network topology is a tree, the graph is also a tree: there is a unique path between any two nodes. Node $u \in M$ can only be a leaf node. Figure 1 shows an example cluster. We assume that all links have the same bandwidth. In practice, switches may support ports with different speeds. For example, Dell PowerEdge 2324 provides 24 100Mbps ports and 2 1Gbps ports. The higher speed ports are usually used to connect switches. In this case, the higher speed inter-switch link will unlikely be the bottleneck in the communication and thus, two switches connected by such a link can be considered as a single switch in the model.

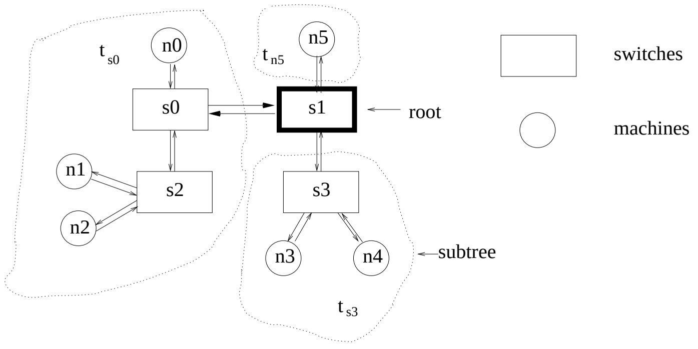


Figure 1: An example Ethernet Switched Cluster

The terminologies used in this paper are defined next. A *message*, $u \rightarrow v$, is a communication transmitted from node u to node v. Since the graph is a tree, the path for a message is fixed. The notion $path(u, v)$ denotes the set of directed edges in the unique path from node u to node v. For example, in Figure 1, $path(n0, n3) = \{(n0, s0), (s0, s1), (s1, s3), (s3, n3)\}$. A path may also be represented as a sequence of nodes. For example, $path(n0, n3)$ may be denoted as $n0 \rightarrow s0 \rightarrow s1 \rightarrow s3 \rightarrow n3$. Two messages, $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$, are said to have contention if they share a common edge, that is, there exists an edge (x, y) such that $(x, y) \in path(u_1, v_1)$ and $(x, y) \in path(u_2, v_2)$. A *pattern* is a set of messages. The *AAPC pattern* on a network $G = (S \cup M, E)$ is $\{u \rightarrow v | u \neq v \in M\}$. The message size in the AAPC pattern is denoted as *msize*. A *contention free pattern* is a pattern where no two messages in the pattern have contention. A *phase* is a contention free pattern. For a given pattern, the *load* on an edge is the number of times the edge is used in the pattern. The most loaded edge is called a *bottleneck* edge. Since the topology is a tree, edges (u, v) and (v, u) always have the same load. We will use the terms "the load of an edge (u, v)" and "the load of a link (u, v)" interchangeably. The *load of a pattern* is equal to the load of a bottleneck edge. Since we only consider the AAPC pattern in this paper, a bottleneck edge on a graph refers to a bottleneck edge when the AAPC pattern is realized. For a set S, $|S|$ denotes the size of the set. Since scheduling for AAPC when $|M| \leq 2$ is trivial, we will assume that $|M| \geq 3$.

Let edge (u, v) be one of the bottleneck edges for the AAPC pattern. Assume that removing link (u, v) (edges (u, v) and (v, u)) partitions graph $G = (S \cup M, E)$ into two sub-graphs, $G_u = (S_u \cup M_u, E_u)$ and $G_v = (S_v \cup M_v, E_v)$. G_u is the connected component including node u, and G_v is the connected component including node v. AAPC requires $|M_u| \times |M_v| \times msize$

bytes data to be transferred across the link (u, v) in both directions. Let B be the bandwidth on all links. The best case time to complete AAPC is $\frac{|M_u| \times |M_v| \times msize}{B}$. The aggregate throughput of AAPC is bounded by

$$Peak\ aggregate\ throughput \leq \frac{|M|(|M|-1)msize}{\frac{|M_u| \times |M_v| \times msize}{B}} = \frac{|M|(|M|-1)B}{M_u \times M_v}$$

In general networks, this peak aggregate throughput may not be achieved due to node and link congestion. However, as will be shown later, for the tree topology, this physical limit can be approached through message scheduling.

4 AAPC Message Scheduling

In the following, we will present an algorithm that constructs phases for AAPC. The phases conform to the following constraints, which are sufficient to guarantee optimality: (1) no contention within each phase; (2) every message in AAPC appears exactly once in the phases; and (3) the total number of phases is equal to the load of AAPC on a given topology. If phases that satisfy these constraints can be carried out without inter-phase interferences, the peak aggregate throughput is achieved.

Our scheduling algorithm has three components. The first component identifies the *root* of the system. For a graph $G = (S \cup M, E)$, the *root* is a switch that satisfies the following conditions: (1) it is connected to a bottleneck edge; and (2) the number of machines in each of the subtrees connecting to the root is less than or equal to $\frac{|M|}{2}$, half of all machines in the system. Once the root is identified, the algorithm schedules messages in two levels: *local messages* that are within a subtree, and *global messages* that are between subtrees. The second component performs global message scheduling that determines the phases when messages between two subtrees are carried out. Finally, the third component performs global and local message assignment, which decides the final scheduling of local and global messages.

4.1 Identifying the Root

Let the graph be $G = (S \cup M, E)$. The process to find a root in the network is as follows. Let link $L = (u, v)$ (edges (u, v) and (v, u)) be one of the bottleneck links. Link L partitions the graph into two subgraphs, $G_u = (S_u \cup M_u, E_u)$ and $G_v = (S_v \cup M_v, E_v)$. The load of link L is thus, $|M_u| \times |M_v| = (|M| - M_v) \times |M_v|$. Assume that $|M_u| \geq |M_v|$. If in G_u, node u has more than one branch containing machines, then node u is the root. Otherwise, node u should have exactly one branch that contains machines (obviously this branch may also have switches). Let the branch connect to node u through link (u_1, u). Clearly, link (u_1, u) is also a bottleneck link since all machines in G_u are in G_{u_1}. Thus, we can repeat the process for link (u_1, u). This process can be repeated n times and n bottleneck links (u_n, u_{n-1}), (u_{n-1}, u_{n-2}), ..., (u_1, u), are considered until the node u_n has more than one branch containing machines in G_{u_n}. Then, u_n is the root. Node u_n should have a nodal degree larger than 2 in G.

Lemma 1: Using the above process to find the root, each subtree of the root contains at most $\frac{|M|}{2}$ machines.

Proof: Using the above process, we identify a root u_n and the connected bottleneck link (u_n, u_{n-1}). Let $G_{u_n} = (S_{u_n} \cup M_{u_n}, E_{u_n})$ and $G_{u_{n-1}} = (S_{u_{n-1}} \cup M_{u_{n-1}}, E_{u_{n-1}})$ be the two subtrees after link (u_n, u_{n-1}) is removed from G. Follow the root identifying process, we have $|M_{u_n}| \geq |M_{u_{n-1}}|$ and $|M_{u_{n-1}}| \leq \frac{|M|}{2}$. Since $|M| > 2$, G_{u_n} contains at least 2 machines and $G_{u_{n-1}}$ contains at least 1 machine. The load on the bottle link (u_n, u_{n-1}) is $|M_{u_n}||M_{u_{n-1}}|$. Let node w be a node that connects to node u_n in G_{u_n} and $G_w = (S_w \cup M_w, E_w)$ be the corresponding subtree. We have $\frac{|M|}{2} \geq |M_{u_{n-1}}| \geq |M_w|$ [Note: if $|M_{u_{n-1}}| < |M_w|$, the load on link (u_n, w), $|M_w| \times (|M| - |M_w|) > |M_{u_{n-1}}| \times (|M| - |M_{u_{n-1}}|)$, is greater than the load on link (u_n, u_{n-1}), which contradicts to the fact that (u_n, u_{n-1}) is a bottleneck link]. Hence, each subtree of the root contains at most $\frac{|M|}{2}$ machines, and subtree $t_{u_{n-1}}$ is one of the subtrees that have the largest number of machines. $\square$

In Figure 1, the link connecting $s0$ to $s1$ is the bottleneck link. Both nodes $s0$ and $s1$ can be the root. Assume that $s1$ is selected as the root. It is connected with three subtrees t_{s0} that contains three machines $n0$, $n1$, and $n2$, t_{s3} that contains two machines n_3 and n_4, and t_{n5} that contains one machine $n5$.

4.2 Global Message Scheduling

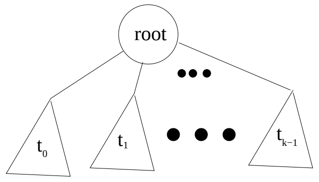


Figure 2: A two level view of the network

Let the root connect to k subtrees, t_0, t_1, ..., t_{k-1}, with $|M_0|, |M_1|, ..., |M_{k-1}|$ machines respectively. Figure 2 shows the two-level view of the network. Only global messages use the links between the subtrees and the root. Local messages only use links within a subtree. Without loss of generality, let us assume that $|M_0| \geq |M_1| \geq ... \geq |M_{k-1}|$. Thus, the load of AAPC is $|M_0| * (|M_1| + |M_2| + ... + |M_{k-1}|) = |M_0| \times (|M| - |M_0|)$ and we must schedule both local and global messages in $|M_0| \times (|M| - |M_0|)$ phases while maintaining contention-free phases. The scheduling is performed in two steps. First, phases are allocated for global messages in global message scheduling where messages from one subtree to another subtree are treated as groups. Second, individual global and local messages are assigned to particular phases. We will discuss global message scheduling in this subsection and describe global and local message assignment in the next subsection.

We will use the notation $t_i \rightarrow t_j$ to represent either a message from a machine in subtree t_i to a machine in subtree t_j or general messages from subtree t_i to subtree t_j. The global message scheduling decides phases for messages in $t_i \rightarrow t_j$. Let us first consider a simple case where $|M_0| = |M_1| = ... = |M_{k-1}| = 1$. In this case, there is $|M_i| \times |M_j| = 1$ message

Phase 0	Phase 1	...	Phase $k-3$	Phase $k-2$
$t_0 \rightarrow t_1$	$t_0 \rightarrow t_2$	...	$t_0 \rightarrow t_{k-2}$	$t_0 \rightarrow t_{k-1}$
$t_1 \rightarrow t_2$	$t_1 \rightarrow t_3$	...	$t_1 \rightarrow t_{k-1}$	$t_1 \rightarrow t_0$
...	...	...	...	...
$t_{k-2} \rightarrow t_{k-1}$	$t_{k-2} \rightarrow t_0$	...	$t_{k-2} \rightarrow t_{k-4}$	$t_{k-2} \rightarrow t_{k-3}$
$t_{k-1} \rightarrow t_0$	$t_{k-1} \rightarrow t_1$	...	$t_{k-1} \rightarrow t_{k-3}$	$t_{k-1} \rightarrow t_{k-2}$

Table 1: Phases from ring scheduling

in $t_i \rightarrow t_j$. A ring scheduling algorithm [17, 14] can be used to schedule the messages in $1 \times (k-1) = k-1$ phases. In ring scheduling, $t_i \rightarrow t_j$ is scheduled at phase $j-i-1$ if $j > i$ and phase $(k-1)-(i-j)$ if $i > j$. The ring scheduling produces $k-1$ phases shown in Table 1.

When scheduling messages with any number of machines in a subtree, we group all messages from one subtree to another into consecutive phases. The total number of messages from t_i to t_j is $|M_i||M_j|$. We extend ring scheduling to allocate phases for groups of messages. In the extended ring scheduling, for subtree t_i, the messages to other subtrees follow the same order as the ring scheduling. For example, for t_1, messages in $t_1 \rightarrow t_2$ happen before messages in $t_1 \rightarrow t_3$, messages in $t_1 \rightarrow t_3$ happen before messages in $t_1 \rightarrow t_4$, and so on. Specifically, the phases are allocated as follows. Note that messages in $t_i \rightarrow t_j$ occupy $|M_i||M_j|$ consecutive phases.

- When $j > i$, messages in $t_i \rightarrow t_j$ start at phase $|M_i| * (|M_{i+1}| + |M_{i+2}| + ... + |M_{j-1}|) = |M_i| \times \sum_{k=i+1}^{j-1} |M_k|$. Note that when $i+1 > j-1$, $\sum_{k=i+1}^{j-1} |M_k| = |M_{i+1}| + |M_{i+2}| + ... + |M_{j-1}| = 0$.
- When $i > j$, messages in $t_i \rightarrow t_j$ start at phase $|M_0| * (|M| - |M_0|) - (|M_i| + |M_{i-1}| + ... + |M_{j+1}|) * |M_j| = |M_0| * (|M| - |M_0|) -$ $(|M_j| \sum_{k=j+1}^{i} |M_k|)$.

Lemma 2: Using the extended ring scheduling described above, the resulting phases have the following two properties: (1) the number of phases is $|M_0| * (|M| - |M_0|)$; and (2) in each phase, global messages do not have contention on links connecting subtrees to the root.
Proof: When $j > i$, messages in $t_i \rightarrow t_j$ start at phase $|M_i| * (|M_{i+1}| + |M_{i+2}| + ... + |M_{j-1}|)$ and end at phase $|M_i| * (|M_{i+1}| + |M_{i+2}| + ... + |M_{j-1}| + |M_j|) - 1 < |M_0| * (|M_1| + ... + |M_{k-1}|) = |M_0| * (|M| - |M_0|)$.

When $i > j$, messages in $t_i \rightarrow t_j$ start at phase $|M_0| * (|M| - |M_0|) - (|M_i| + |M_{i-1}| + ... + |M_{j+1}|) * |M_j|$ and end at phase $|M_0| * (|M| - |M_0|) - (|M_i| + |M_{i-1}| + ... + |M_{j+1}|) * |M_j| + |M_i| * |M_j| - 1 < |M_0| * (|M| - |M_0|)$. Thus, the number of phases is less than or equal to $|M_0| * (|M| - |M_0|)$. Note the phase count starts at phase 0.

Messages in $t_0 \rightarrow t_{k-1}$ start at phase $|M_0| * (|M_1| + |M_2| + ... + |M_{k-2}|)$ and end at phase $|M_0| * (|M_1| + |M_2| + ... + |M_{k-2}|) + |M_0| * |M_{k-1}| - 1 = |M_0| * (|M| - |M_0|) - 1$. Thus, the number of phases is exactly $|M_0| * (|M| - |M_0|)$.

Examining the starting and ending phases for messages in $t_i \rightarrow t_j$, it can be shown that phases for $t_i \rightarrow t_j$, $j \neq i$, do not overlap and that phases for $t_j \rightarrow t_i$, $j \neq i$, do not overlap.

Thus, at each phase, at most one node in a subtree is sending and at most one node in a subtree is receiving. As a result, the two edges of the link connecting a subtree to the root will be used at most once in each phase. Hence, in each phase, global messages do not have contention on links connecting subtrees to the root. □

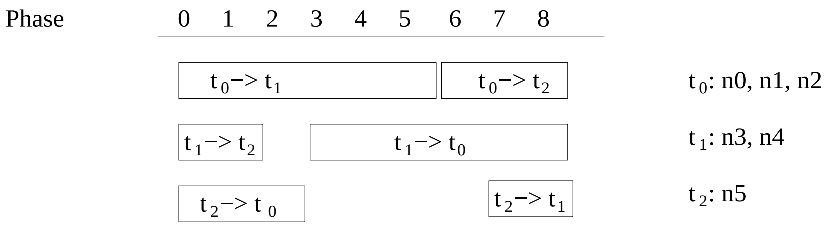


Figure 3: Global message scheduling for the example in Figure 1

Figure 3 shows the scheduling of global messages for the example shown in Figure 1. In this figure, $t_0 = t_{s0}$ contains three machines $n0$, $n1$, and $n2$; $t_1 = t_{s3}$ contains two machines $n3$ and $n4$; and $t_2 = t_{n5}$ contains one machine $n5$. Thus, $|M_0| = 3$, $|M_1| = 2$, and $|M_2| = 1$. Messages in $t_1 \rightarrow t_2$ start at $|M_1| \times \sum_{k=2}^{1} |M_k| = 0$. Messages in $t_0 \rightarrow t_2$ start at $|M_0| \times \sum_{k=1}^{1} |M_k| = |M_0| \times |M_1| = 6$. Messages in $t_2 \rightarrow t_0$ start at $|M_0| * (|M| - |M_0|) - |M_0| \times \sum_{k=1}^{2} |M_k| = 0$. The figure also shows that some subtrees are idle at some phases. For example, subtree t_1 does not have a sending machine in phase 2.

4.3 Global and Local Message Assignment

Let the root connect to k subtrees, t_0, t_1, ..., t_{k-1}, with $|M_0|$, $|M_1|$, ..., $|M_{k-1}|$ machines respectively. $|M_0| \geq |M_1| \geq ... \geq |M_{k-1}|$. As shown in the previous subsection, global messages are scheduled in $|M_0| \times (|M| - |M_0|)$ phases. Consider subtree t_i, the total number of local messages in t_i is $|M_i| \times (|M_i| - 1)$, which is less than the total number of phases. Thus, if in each phase, one local message in each subtree can be scheduled without contention with the global messages, all messages in AAPC can be scheduled in $|M_0| \times (|M| - |M_0|)$ phases. The contention free scheduling of global and local messages is based on the following lemma.
Lemma 3: Let $G = (S \cup M, E)$ be a tree and $x \neq y \neq z \in S \cup M$, $path(x, y) \cap path(y, z) = \phi$.
Proof: Assume that $path(x, y) \cap path(y, z) \neq \phi$. There exists an edge (u, v) that belongs to both $path(x, y)$ and $path(y, z)$. As a result, the composition of the partial path $path(y, u) \subseteq path(y, z)$ and $path(u, y) \subseteq path(x, y)$ forms a non-trivial loop: edge (u, v) is in the loop while edge (v, u) is not. This contradicts to the assumption that G is a tree. □
Lemma 4: Using the global message scheduling scheme, at each phase, the global messages do not have contention.
Proof: Let root connect to subtrees t_0, t_1, ..., t_{k-1} as shown in Figure 2. From Lemma 2, at each phase, there is no contention in the link connecting a subtree to the root. When there is only one global message in a subtree in a phase, there will be no contention in that phase in that subtree. Thus, the only case when global messages may have contention inside a subtree is when there are two global messages involving nodes in a subtree in a phase. In this case, one global message is sent to a node in the subtree and the other one is sent

from a node in the subtree. Let the two messages be $x \rightarrow o_1$ and $o_2 \rightarrow y$, where $x \in M_i$, $y \in M_i$, and o_1 and o_2 are in other subtrees. The sub-path for $x \rightarrow o_1$ inside t_i is equal to $path(x, root)$ and the sub-path for $o_2 \rightarrow y$ is equal to $path(root, y)$. From Lemma 3, these two paths do not have contention inside t_i. $\Box$

The contention free scheduling of local messages is also based on Lemma 3. As discussed earlier, the total number of local messages in a subtree is less than the total number of phases. Thus, it is sufficient to schedule one local message in each phase. Let $u \neq v \in t_i$. From Lemma 3, there are three cases when message $u \rightarrow v$ can be scheduled without contention (with global messages) in a phase: (1) node v is the sender of a global message and node u is the receiver of a global message; (2) node v is the sender of a global message and there is no receiving node of a global message in t_i; and (3) node u is the receiver of a global message is there is no sending node of a global message. The global and local message assignment algorithm assigns phases to all global messages in such a way that all local messages can be scheduled without contention. Note that by scheduling at most one local message in each subtree, our scheduling algorithm does not have to consider the specific topologies of the subtrees.

Let us now consider how the phases are assigned to global messages. Let us number the nodes in subtree t_i as $t_{i,0}$, $t_{i,1}$, ..., $t_{i,(|M_i|-1)}$. To realize inter-subtree communication $t_i \rightarrow t_j$, $0 \leq i \neq j < k$, each message $t_{i,i_1} \rightarrow t_{j,j_1}$, $0 \leq i_1 < |M_i|$ and $0 \leq j_1 < |M_j|$, must happen in the $|M_i| * |M_j|$ phases that are allocated to $t_i \rightarrow t_j$. Our assignment algorithm uses two different methods to realize inter-subtree communications. The first scheme is what we refer to as a *broadcast* scheme. In this scheme, the $|M_i| * |M_j|$ phases are partitioned into $|M_i|$ rounds with each round having $|M_j|$ phases. In each round, a different node in t_i sends one message to each of the nodes in t_j. This method has the flexibility in selecting the order of the senders in t_i in each round and the order of the receivers in t_j within each round. One example is to have kth round realize the broadcast from node $t_{i,k}$ to all nodes in t_j, which results in the following pattern:

$t_{i,0} \rightarrow t_{j,0}, ..., t_{i,0} \rightarrow t_{j,|M_j|-1}$, $t_{i,1} \rightarrow t_{j,0}, ...,$ $t_{i,1} \rightarrow t_{j,|M_j|-1}$, ..., $t_{i,|M_i|-1} \rightarrow t_{j,0}, ...,$ $t_{i,|M_i|-1} \rightarrow t_{j,|M_j|-1}$.

The second scheme is what we refer to as a *rotate* scheme. Let D be the greatest common divisor of $|M_i|$ and $|M_j|$. $D = gcd(|M_i|, |M_j|)$ and $|M_i| = a \times D$, $|M_j| = b \times D$. In this scheme, the pattern for receivers is a repetition of M_i times of a fixed sequence that enumerates all nodes in t_j. One example of the fixed sequence is $t_{j,0}, t_{j,1}, ...t_{j,|M_j|-1}$, which results in a receiver pattern of

$$t_{j,0}, t_{j,1}, ...t_{j,|M_j|-1}, t_{j,0}, t_{j,1}, ...t_{j,|M_j|-1}, ..., t_{j,0}, t_{j,1}, ...t_{j,|M_j|-1}.$$

Note that the rotate pattern does not restrict the fixed sequence that enumerates all nodes in t_j. Different from the broadcast scheme, in a rotate scheme, the sender pattern is also an enumeration of all nodes in t_i in every $|M_i|$ phases. There is a *base sequence* for the senders, which can be an arbitrary sequence that covers all nodes in t_i. For example, the base sequence can be $t_{i,0}, t_{i,1}, ...t_{i,|M_i|-1}$. In the scheduling, the base sequence and the "rotated" base sequence are used. Let the base sequence be $t_{i,0}, t_{i,1}, ...t_{i,|M_i|-1}$. The base sequence can be rotated 1 time, which produces the sequence $t_{i,1}, ...t_{i,|M_i|-1}, t_{i,0}$. Sequence $t_{i,2}, ...t_{i,|M_i|-1}, t_{i,0}, t_{i,1}$ is the result of rotating the base sequence 2 times. The result from rotating the base sequence n times can be defined similarly. The senders are scheduled as

phase	comm.	phase	comm	phase	comm	phase	comm
0	$t_{i,0} \rightarrow t_{j,0}$	6	$t_{i,0} \rightarrow t_{j,2}$	12	$t_{i,1} \rightarrow t_{j,0}$	18	$t_{i,1} \rightarrow t_{j,2}$
1	$t_{i,1} \rightarrow t_{j,1}$	7	$t_{i,1} \rightarrow t_{j,3}$	13	$t_{i,2} \rightarrow t_{j,1}$	19	$t_{i,2} \rightarrow t_{j,3}$
2	$t_{i,2} \rightarrow t_{j,2}$	8	$t_{i,2} \rightarrow t_{j,0}$	14	$t_{i,3} \rightarrow t_{j,2}$	20	$t_{i,3} \rightarrow t_{j,0}$
3	$t_{i,3} \rightarrow t_{j,3}$	9	$t_{i,3} \rightarrow t_{j,1}$	15	$t_{i,4} \rightarrow t_{j,3}$	21	$t_{i,4} \rightarrow t_{j,1}$
4	$t_{i,4} \rightarrow t_{j,0}$	10	$t_{i,4} \rightarrow t_{j,2}$	16	$t_{i,5} \rightarrow t_{j,0}$	22	$t_{i,5} \rightarrow t_{j,2}$
5	$t_{i,5} \rightarrow t_{j,1}$	11	$t_{i,5} \rightarrow t_{j,3}$	17	$t_{i,0} \rightarrow t_{j,1}$	23	$t_{i,0} \rightarrow t_{j,3}$

Table 2: Rotate pattern for realizing $t_i \rightarrow t_j$ when $|M_i| = 6$ and $|M_j| = 4$

follows. The base sequence is repeated b times for the first $a \times b \times D$ phases. At phase $a \times b \times D$, the scheme finds the smallest n such that after the base sequence is rotated n times, the message (sender and receiver pair) at phase $a \times b \times D$ does not happen before. The sequence resulting from rotating base sequence n times is then repeated b times. This process is repeated D times to create the sender pattern for all $|M_i||M_j|$ phases. Basically, at phases that are multiples of $a \times b \times D$ phases, rotations are performed to find a new sequence that is repeated b times. It can be shown that all messages in $t_i \rightarrow t_j$ are realized in the rotate scheme.

Table 2 shows an example when $|M_i| = 6$ and $|M_j| = 4$. In this case, $a = 3$, $b = 2$, and $D = 2$. The receivers repeat the pattern $t_{j,0}, t_{j,1}, t_{j,2}, t_{j,3}$. The base sequence for the senders is $t_{i,0}, t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}, t_{i,5}$. This sequence is repeated 2 times. At phase $2 * 3 * 2 = 12$, the senders follow a rotated sequence $t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}, t_{i,5}, t_{i,0}$ and repeat the pattern 2 times. It can be verified that all messages in $t_i \rightarrow t_j$ are realized.

The following two lemmas illustrate the properties of the broadcast pattern and the rotate pattern.

Lemma 5: In the broadcast pattern that realizes $t_i \rightarrow t_j$, each sender $t_{i,k}$ occupies $|M_j|$ continuous phases.

Proof: Straight-forward from the definition of the broadcast pattern. □.

Lemma 6: In the rotate pattern that realizes $t_i \rightarrow t_j$, counting from the first phase for messages in $t_i \rightarrow t_j$, each sender in t_i happens once in every $|M_i|$ phases and each receiver in t_j happens once in every $|M_j|$ phases.

Proof: Straight-forward from the definition of the rotate pattern. □.

Either the broadcast pattern or the rotate pattern can be used to realize messages in $t_i \rightarrow t_j$, $0 \leq i \neq j < k$. The challenge in the scheduling, however, is that we must be able to embed all local messages in the $|M_0| * (|M| - |M_0|)$ phases. The scheduling algorithm is shown in Figure 4. The algorithm consists of six steps. We will explain each step next.

In the first step, the messages from t_0 to all other subtrees t_j, $1 \leq j < k$ are scheduled. First, the receivers in $t_0 \rightarrow t_j$ are assigned such that at phase p, node $t_{j,(p-|M_0|(|M|-|M_0|))\ mod\ |M_j|}$ is the receiver. It can be easily verified that in the phases for $t_0 \rightarrow t_j$, a receiver sequence that covers all nodes in t_j is repeated $|M_0|$ times, which facilitates the rotate pattern to be used for all the messages in $t_0 \rightarrow t_j$. The reason that the receivers use that particular pattern is to align the receivers with the receivers in $t_i \rightarrow t_j$ when $i > j$. As will be shown Step 5, this alignment is needed to correctly schedule local messages. Using the rotate pattern

```
Input: Results from global message scheduling that identify which phases are used to
    realize $t_i \to t_j$ for all $0 \le i \ne j < k$
Output: (1) the phase to realize each global message
        $t_{i,i_1} \to t_{j,j_1}$, $0 \le i_1 < |M_i|$, $0 \le j_1 < |M_j|$, $0 \le i \ne j < k$.
    (2) the phase to realize each local message $t_{i,i_1} \to t_{i,i_2}$, $0 \le i_1 \ne i_2 < |M_i|$, $0 \le i < k$.

Step 1: Assign phases to messages in $t_0 \to t_j$, $1 \le j < k$.
    1.a: For each $t_0 \to t_j$, the receivers in $t_j$ are assigned as follows:
        at phase $p$ in the phases for $t_0 \to t_j$, machine $t_{j,(p-|M_0|(|M|-|M_0|))\ mod\ |M_j|}$ is the receiver.
        /* it can be verified that a sequence that enumerates the nodes in $t_j$ is repeated $|M_0|$ times
          in phases for $t_0 \to t_j$. */
    1.b: For each $t_0 \to t_j$, the senders in $t_0$ are assigned according to the rotate pattern with
          the base sequence $t_{0,0}, t_{0,1}, ..., t_{0,|M_0|-1}$.

Step 2: Assign phases to messages in $t_i \to t_0$, $1 \le i < k$.
    2.a: Assign the receivers in $t_i \to t_0$:
        /*Step 1.b organizes the senders in $t_0$ in such a way that every $|M_0|$ phases, all nodes in $t_0$
          appear as the sender once. We call $|M_0|$ phases a *round* */
        The receiver pattern in $t_i \to t_0$ is computed based on the sender pattern in $t_0 \to t_j$ according
        to the mapping shown in Table 3. Round $r$ has the same mapping as round $r\ mod\ |M_0|$.
        /* the mapping ensures that the local messages in $t_0$ can be scheduled */
    2.b: Assign the senders in $t_i$ using the broadcast pattern with order $t_{i,0}$, $t_{i,1}$, ..., $t_{i,|M_i|-1}$.

Step 3: Schedule local messages in $t_0$ in phase 0 to phase $|M_0|(|M_0|-1)$.
      message $t_{0,i} \to t_{0,j}$, $0 \le i \ne j < |M_0|$, is scheduled at the phase where $t_{0,i}$ is the receiver
      of a global message and $t_{0,j}$ is the sender of a global message.

Step 4: Assign phases to global messages in $t_i \to t_j$, $i > j$ and $j \ne 0$.
      Use the broadcast pattern with receivers repeating pattern $t_{j,0}$, $t_{j,1}$, ..., $t_{j,|M_j|-1}$ for each
      sender $t_{i,k}$ and senders following the order $t_{i,0}$, $t_{i,1}$, $t_{i,k}$, ..., $t_{i,|M_i|-1}$.

Step 5: Schedule local messages in $t_i$, $1 \le i < k$ in phases for $t_i \to t_{i-1}$.
        /* the last phase for $t_i \to t_{i-1}$ is the last phase $|M_0|(|M|-|M_0|)-1$.*/
        Steps 1 through 4 ensure that for each local message $t_{i,i1} \to t_{i,i2}$,
        there is a phase in the phases for $t_i \to t_{i-1}$ such that $t_{i,i2}$ is the sender
        of a global message and either $t_{i,i1}$ is a receiver of a global message or no node in $t_i$
        is receiving a global message. This step schedules $t_{i,i1} \to t_{i,i2}$ in this phase.

Step 6: Use either the broadcast pattern or the rotate pattern for messages in $t_i \to t_j$, $i < j$ and $i \ne 0$.
        /* scheduling of these global message would not affect the scheduling of local messages. */
```

Figure 4: The global and local message assignment algorithm

round 0		round 1		...	round $\|M_0\| - 2$		round $\|M_0\| - 1$		...
send	recv	send	recv	...	send	recv	send	recv	...
$t_{0,0}$	$t_{0,1}$	$t_{0,0}$	$t_{0,2}$	...	$t_{0,0}$	$t_{0,\|M_0\|-1}$	$t_{0,0}$	$t_{0,0}$	...
$t_{0,1}$	$t_{0,2}$	$t_{0,1}$	$t_{0,3}$	...	$t_{0,1}$	$t_{0,0}$	$t_{0,1}$	$t_{0,1}$	...
...	...	...	...	...	...	...	...	...	...
$t_{0,\|M_0\|-2}$	$t_{0,\|M_0\|-1}$	$t_{0,\|M_0\|-2}$	$t_{0,0}$	...	$t_{0,\|M_0\|-2}$	$t_{0,\|M_0\|-3}$	$t_{0,\|M_0\|-2}$	$t_{0,\|M_0\|-2}$	...
$t_{0,\|M_0\|-1}$	$t_{0,0}$	$t_{0,\|M_0\|-1}$	$t_{0,1}$	...	$t_{0,\|M_0\|-1}$	$t_{0,\|M_0\|-2}$	$t_{0,\|M_0\|-1}$	$t_{0,\|M_0\|-1}$	...

Table 3: Mapping between senders and the receivers in Step 2. Round r has the same mapping as round $r \ mod \ |M_0|$

ensures that each of the nodes in t_0 appears once as the sender in every $|M_0|$ phases counting from phase 0.

In the second step, messages in $t_i \rightarrow t_0$ are assigned. In this step, phases are partitioned into rounds with each round have $|M_0|$ phases. The primary objective of this step is to make sure that all local messages in t_0 can be scheduled. The objective is achieved by creating the pattern shown in Table 3, which is basically a rotate pattern for $t_0 \rightarrow t_0$. Since in step 1, each node in t_0 appears as a sender in every $|M_0|$ phases, the scheduling of receivers in $t_i \rightarrow t_0$ can directly follow the mapping in Table 3. Using this mapping, every node in t_0 appears as a receiver in every $|M_0|$ phases, which facilitates the use of a broadcast pattern to realize messages in $t_i \rightarrow t_0$, $i > 0$. After the receiver pattern is decided, the senders of $t_i \rightarrow t_0$ are determined using the broadcast scheme with the sender order $t_{i,0}$, $t_{i,1}$, ..., $t_{i,|M_i|-1}$.

Step 3 embeds local messages in t_0 in the first $|M_0| * (|M_0| - 1)$ phases. Note that $|M_0| * (|M_0| - 1) \leq |M_0| * (|M| - |M_0|)$. Since the global messages for nodes in t_0 are scheduled according to Table 3, for each $t_{0,n} \rightarrow t_{0,m}$, $0 \leq n \neq m < |M_0|$, there exists a phase in the first $|M_0|(|M_0| - 1)$ phases such that $t_{0,n}$ is scheduled to receive a global message while $t_{0,m}$ is scheduled to send a global message. Thus, all local messages in t_0, $t_{0,n} \rightarrow t_{0,m}$, $0 \leq n \neq m < |M_0|$, can be scheduled in the the first $|M_0|(|M_0| - 1)$ phases.

In Step 4, global messages in $t_i \rightarrow t_j$, $i > j$ and $j \neq 0$ are assigned. The broadcast pattern is used to assign global messages with receivers repeating the pattern $t_{j,0}$, $t_{j,1}$, ..., $t_{j,|M_j|-1}$ and senders following the order $t_{i,0}$, $t_{i,1}$, ..., $t_{i,|M_i|-1}$. Hence, messages in $t_i \rightarrow t_j$, $i > j$ and $j \neq 0$ are assigned as

$$t_{i,0} \rightarrow t_{j,0}, ..., t_{i,0} \rightarrow t_{j,|M_j|-1}, t_{i,1} \rightarrow t_{j,0}, ..., t_{i,1} \rightarrow t_{j,|M_j|-1}, t_{i,|M_i|-1} \rightarrow t_{j,0}, ..., t_{i,|M_i|-1} \rightarrow t_{j,|M_j|-1}.$$

In Step 5, we schedule local messages in subtrees other than t_0. Local messages in t_i, $1 \leq i < k$, are scheduled in the phases for $t_i \rightarrow t_{i-1}$. Note that $|M_{i-1}| \geq |M_i|$ and there are $|M_i||M_{i-1}|$ phases for messages in $t_i \rightarrow t_{i-1}$, which is more than the $|M_i|(|M_i| - 1)$ phases needed for local messages in t_i. There are some subtle issues in this step. First, all local messages are scheduled before assigning phases to global messages in $t_i \rightarrow t_j$, $1 \leq i < j$. The reason that global messages in $t_i \rightarrow t_j$, $1 \leq i < j$, do not affect the local message scheduling in subtree t_n, $1 \leq n < k$, is that all local messages are scheduled in phases after the first phase for $t_0 \rightarrow t_n$ (since $|M_n| * |M_{n-1}| \leq |M_0| * |M_n|$) while phases for $t_i \rightarrow t_j$, $1 \leq i < j$, are all before that phase. Second, let us examine how exactly a communication $t_{i,i_2} \rightarrow t_{i,i_1}$ is scheduled. From Step 4, the receiver in $t_j \rightarrow t_i$, $j > i$, is organized such that, at phase p, $t_{i,(p-|M_0|(|M|-|M_0|))\ mod\ |M_i|}$ is the receiver. From Step 1, receivers in $t_0 \rightarrow t_i$ are also aligned such that at phase p, $t_{i,(p-|M_0|(|M|-|M_0|))\ mod\ |M_i|}$ is the receiver. Hence, in the phases for $t_i \rightarrow t_{i-1}$, either $t_{i,(p-|M_0|(|M|-|M_0|))\ mod\ |M_i|}$ is a receiver of a global message or no node in t_i is receiving a global message. Thus, at all phases in $t_i \rightarrow t_{i-1}$, we can assume that the designated receiver is $t_{i,(p-|M_0|(|M|-|M_0|))\ mod\ |M_i|}$ at phase p. In other words, at phase p, $t_{i,(p-|M_0|(|M|-|M_0|))\ mod\ |M_i|}$ can be scheduled as the sender of a local message. Now, consider the sender pattern in $t_i \rightarrow t_{i-1}$. Since $t_i \rightarrow t_{i-1}$ is scheduled using the broadcast pattern, each t_{i,i_1} is sending in $|M_{i-1}|$ continuous phases. Since the receiving pattern covers every node, $t_{i,i_2} \in t_i$, in every $|M_i|$ continuous phases and $|M_{i-1}| \geq |M_i|$, there exists at least one phase where t_{i,i_1} is sending a global message and t_{i,i_2} is the designated receiver of a global message. Local message $t_{i,i_2} \rightarrow t_{i,i_1}$ is scheduled in this phase. Hence, all messages in t_i can be scheduled in phases for $t_i \rightarrow t_{i-1}$ without contention.

Finally, since all local messages are scheduled, we can use either the broadcast scheme or rotate scheme to realize messages in $t_i \rightarrow t_j$, $i < j$ and $i \neq 0$.

Theorem: The global and local message assignment algorithm in Figure 4 produces phases that satisfy the following conditions: (1) all messages in AAPC are realized in $|M_0| \times (|M| - |M_0|)$ phases; and (2) there is no contention within each phase.

Proof: From Lemma 2, we see that all global messages are scheduled in $|M_0| \times (|M| - |M_0|)$ phases. Step 3 in the algorithm indicates that local messages in t_0 are scheduled in $|M_0| \times (|M_0| - 1)$ phases. In Step 5, all local messages in t_i are scheduled in the phases allocated to communications in $t_i \rightarrow t_{i-1}$. Thus, all messages in AAPC are scheduled in $|M_0| \times (|M| - |M_0|)$ phases.

Lemma 4 shows that there is no contention among global messages in each phase. Since local messages in different subtrees cannot have contention and since in one phase, at most one local message in a subtree is scheduled, the contention can only happen between a global message and a local message inside a subtree. In the scheduling of local messages in t_0 (Step 3), a local message $t_{0,i} \rightarrow t_{0,j}$ is scheduled in the phase when $t_{0,i}$ is a receiver of a global message and $t_{0,j}$ is a sender of a global message. From Lemma 3, local message $t_{0,i} \rightarrow t_{0,j}$ does not have contention with the two global messages. Local messages in t_i, $1 \leq i < k$, is scheduled in Step 5 in phases allocated to communications in $t_i \rightarrow t_{i-1}$. A local message $t_{i,i_1} \rightarrow t_{i,i_2}$ is scheduled in a phase when t_{i,i_2} is a sender of a global message and either t_{i,i_1} is a receiver of a global message or no node in t_i is scheduled to receive a global message. From Lemma 3, local message $t_{i,i_1} \rightarrow t_{i,i_2}$ cannot have contention with global messages. Thus, there is no contention within a phase. □

Table 4 shows the result of the global and local message assignment for the example in Figure 1. In this table, we can assume $t_{0,0} = n0$, $t_{0,1} = n1$, $t_{0,2} = n2$, $t_{1,0} = n3$, $t_{1,1} = n4$, and $t_{2,0} = n5$. From the algorithm, we first determine the receiver pattern in $t_0 \rightarrow t_1$ and $t_0 \rightarrow t_2$. For messages in $t_0 \rightarrow t_1$, $t_{1,(p-9)\ mod\ 2}$ is the receiver at phase p, which means the receiver pattern from phase 0 to phase 5 are $t_{1,1}$, $t_{1,0}$, $t_{1,1}$, $t_{1,0}$, $t_{1,1}$, $t_{1,0}$. After that, the rotation pattern is used to realize all messages in $t_0 \rightarrow t_1$ and $t_0 \rightarrow t_2$. The results are shown in the second column in the figure. In the second step, messages in $t_1 \rightarrow t_0$ and $t_2 \rightarrow t_0$ are assigned. Messages in $t_2 \rightarrow t_0$ occupy the first round (first three phases). Since the sender pattern in the first round is $t_{0,0}$, $t_{0,1}$, and $t_{0,2}$, according to Table 3, the receiver

phase	global messages			local messages		
	$t_0 \rightarrow \{t_1, t_2\}$	$t_1 \rightarrow \{t_2, t_0\}$	$t_2 \rightarrow \{t_0, t_1\}$	t_0	t_1	t_2
0	$t_{0,0} \rightarrow t_{1,1}$	$t_{1,0} \rightarrow t_{2,0}$	$t_{2,0} \rightarrow t_{0,1}$	$t_{0,1} \rightarrow t_{0,0}$		
1	$t_{0,1} \rightarrow t_{1,0}$	$t_{1,1} \rightarrow t_{2,0}$	$t_{2,0} \rightarrow t_{0,2}$	$t_{0,2} \rightarrow t_{0,1}$		
2	$t_{0,2} \rightarrow t_{1,1}$		$t_{2,0} \rightarrow t_{0,0}$	$t_{0,0} \rightarrow t_{0,2}$		
3	$t_{0,0} \rightarrow t_{1,0}$	$t_{1,0} \rightarrow t_{0,2}$		$t_{0,2} \rightarrow t_{0,0}$		
4	$t_{0,1} \rightarrow t_{1,1}$	$t_{1,0} \rightarrow t_{0,0}$		$t_{0,0} \rightarrow t_{0,1}$	$t_{1,1} \rightarrow t_{1,0}$	
5	$t_{0,2} \rightarrow t_{1,0}$	$t_{1,0} \rightarrow t_{0,1}$		$t_{0,1} \rightarrow t_{0,2}$		
6	$t_{0,0} \rightarrow t_{2,0}$	$t_{1,1} \rightarrow t_{0,0}$				
7	$t_{0,1} \rightarrow t_{2,0}$	$t_{1,1} \rightarrow t_{0,1}$	$t_{2,0} \rightarrow t_{1,0}$		$t_{1,0} \rightarrow t_{1,1}$	
8	$t_{0,2} \rightarrow t_{2,0}$	$t_{1,1} \rightarrow t_{0,2}$	$t_{2,0} \rightarrow t_{1,1}$			

Table 4: Results of global and local message assignment for the cluster in Figure 1

pattern should be $t_{0,1}$, $t_{0,2}$, $t_{0,0}$ as shown in Table 4. The receivers for $t_1 \rightarrow t_0$ are assigned in a similar fashion. After that, the broadcast pattern is used to realize both $t_1 \rightarrow t_0$ and $t_2 \rightarrow t_0$. In Step 3, local messages in t_0 are assigned in the first $3 \times 2 = 6$ phases according to the assignment of the sender and receiver of global messages in each phase. For example, in phase 0, local message $t_{0,1} \rightarrow t_{0,0}$ is scheduled since node $t_{0,0}$ is a sender of a global message and $t_{0,1}$ is a receiver of a global message. Note that the mapping in Table 3 ensures that all local messages in t_0 can be scheduled. In Step 4, $t_2 \rightarrow t_1$ is scheduled with a broadcast pattern. In Step 5, local messages in t_1 and t_2 are scheduled. The local messages in t_1 are scheduled in phases for $t_1 \rightarrow t_0$, that is, from phase 3 to phase 8. The alignment of the receivers in $t_0 \rightarrow t_1$ and $t_2 \rightarrow t_1$ ensures that each machine in t_1 appears as the designated receiver in every $|M_1| = 2$ phases starting from the first phase for $t_0 \rightarrow t_1$. Notice that in phase 6, no node in t_1 is receiving a global message. However, the designated receiver is $t_{1,1}$ in this phase. In $t_1 \rightarrow t_0$, each node in t_1 is the sender for $|M_0| = 3$ consecutive phases and the receiver pattern in t_1 covers every node in every 2 phases. All local messages in t_1 can be scheduled. In this particular example, message $t_{1,0} \rightarrow t_{1,1}$ is scheduled at phase 7 where $t_{1,0}$ is a (designated) receiver of a global message and $t_{1,1}$ is a sender of a global message, and $t_{1,1} \rightarrow t_{1,0}$ is scheduled at phase 4. Finally, in Step 6, we use the broadcast pattern for messages in $t_1 \rightarrow t_2$.

5 Implementation Issues

We develop an automatic routine generator that takes the topology information as input and automatically produces an *MPI_Alltoall* routine that is customized to the specific topology. The routine is intended to be used when the message size is large. The software and some automatically generated routines are available at http://www.cs.fsu.edu/∼xyuan/CCMPI. We plan to make the source code public in the near future. Currently the system works with LAM/MPI [7] and the generated routine is built on top of MPI point-to-point communication routines. This section discusses some implementation issues.

To be optimal, the AAPC phases created by the message scheduling scheme must be separated to preserve the contention-free schedule. A simple way to achieve this is to add a barrier between each phase. Using barriers, however, would incur substantial synchronization overheads unless special hardware for the barrier operation such as the Purdue PAPERS [2] is available.

In our implementation, we use a pair-wise synchronization scheme. When two messages $a \rightarrow b$ in phase p and $c \rightarrow d$ in phase q have contention, $p < q$, the pair-wise synchronization makes sure that these two messages do not occur at the same time by introducing a synchronization message from node a to node c. The synchronization message is sent after message $a \rightarrow b$. Message $c \rightarrow d$ is performed after node c receives the synchronization message. Some synchronization messages may not be necessary as the ordering can be derived from other synchronization messages. Such synchronizations are referred to as *redundant synchronizations*. For example, assume that message $m1$ must synchronize with message $m2$ and with another message $m3$. If message $m2$ also needs to synchronize with message $m3$, then the synchronization from $m1$ to $m3$ can be removed.

Our implementation computes the required synchronizations as follows. For every communication at a phase, we check if a synchronization is needed for every other communication at later phases and build a dependence graph. After deciding all synchronizations messages for all communications, we compute and remove redundant synchronizations in the dependence graph. In code generation, synchronization messages are added for all the remaining edges in the dependence graph. This way, the AAPC algorithm maintains a contention-free schedule while minimizing the number of synchronization messages.

6 Experiments

We evaluate the scheduling scheme by comparing our automatically generated routine with the original routine in LAM/MPI [7] and a recent improved *MPI_Alltoall* implementation in MPICH [17]. LAM/MPI implements all-to-all by simply posting all nonblocking receives and sends and then waiting for all communications to finish. The improved MPICH implementation uses different techniques and adapts based on the message size and the number of nodes in the system. For messages larger than 250 Bytes, when the number of nodes is a power of two, MPICH uses a pairwise algorithm where node n sends and receives from node $n \oplus i$ at step i ($1 \leq i < N$, N is the number of nodes). When the number of nodes is not a power of two, a ring algorithm is used. In this case, at step i, node n sends to node $n+i$ and receives from node $n-i$. Both pairwise and ring algorithms finish AAPC in $N-1$ steps.

We use LAM/MPI 6.5.9 in the experiments. The MPICH implementation is slightly modified to work with LAM/MPI. The experiments are performed on a 32-node Ethernet switched cluster. The nodes of the cluster are Dell Dimension 2400 with a 2.8MHz P4 processor, 128MB of memory, and 40GHz of disk space. All machines run Linux (Fedora) with 2.6.5-1.358 kernel. The Ethernet card in each machine is Broadcom BCM 5705 with the driver from Broadcom. These machines are connected to Dell PowerEdge 2224 and Dell PowerEdge 2324 100Mbps Ethernet switches.

The topologies used in the experiments are shown in Figure 5. Figure 5 (a) is a 24-node cluster connected by a single 2324 switch. Figure 5 (b) and Figure 5 (c) are two 32

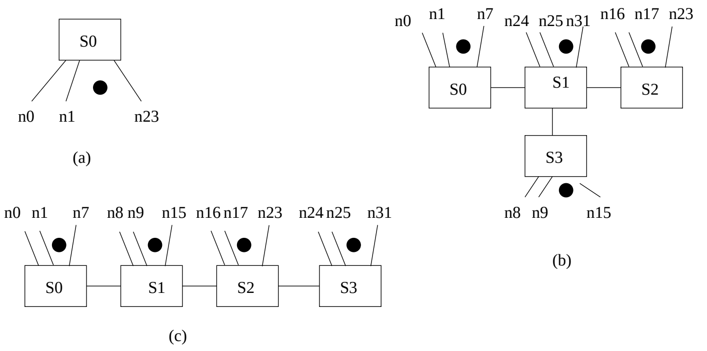
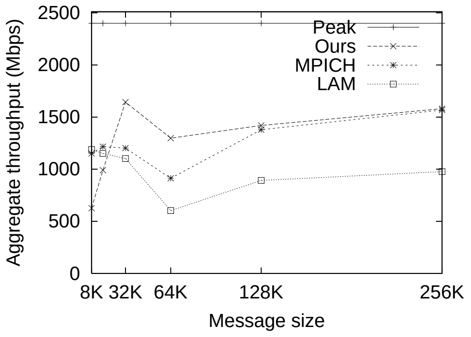


Figure 5: Topologies used in the experiments

msize	LAM	MPICH	Ours
8KB	29.7ms	30.7ms	56.5ms
16KB	61.4ms	58.1ms	71.4ms
32KB	128.2ms	117.6ms	86.0ms
64KB	468.8ms	309.7ms	217.7ms
128KB	633.7ms	410.0ms	398.0ms
256KB	1157ms	721ms	715ms

(a) Completion time



(b) Aggregate throughput

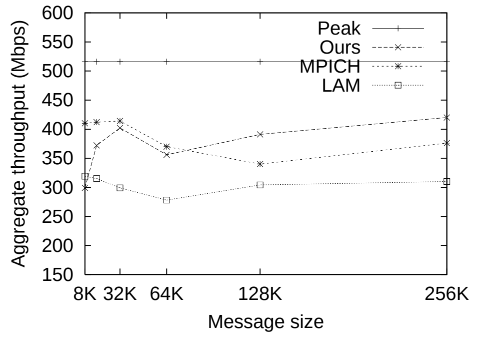
Figure 6: Results *MPI_Alltoall* on the topology in Figure 5 (a)

node clusters. In both cases, 8 nodes are connected to each of the 2224 switches. The results reported are the averages of three executions. In each execution, 10 iterations of *MPI_Alltoall* are measured and the average execution time for each invocation of the routine is recorded.

Figures 6, 7, and 8 show the results for topologies in Figure 5 (a), (b) and (c), respectively. We show the average AAPC completion time and the actual aggregate throughput of the AAPC. In all network configurations, due to the synchronization overheads, our automatically generated routine performs worse than LAM and the improved MPICH when the message size is small. However, when the message size is sufficiently large, our routine out-performs LAM and the improved MPICH. This demonstrates the superiority of our algorithm in exploiting network bandwidths. The algorithm in LAM/MPI does not perform any scheduling and results in severe network contention when the message size is large. The improved MPICH all-to-all algorithm performs a limited form of scheduling. It performs AAPC in phases but does not consider contention in the network links. In addition, the phased algorithm in MPICH does not have synchronizations between phases which may introduce a limited form of node contention since different nodes may finish a phase and start a new phase at different times. The limited form of node contention is shown in the experiment for the topology in Figure 5 (a) when the message sizes are $32KB$ and $64KB$. Note that the

msize	LAM	MPICH	Ours
8KB	199ms	155ms	212ms
16KB	403ms	308ms	341ms
32KB	848ms	613ms	632ms
64KB	1827ms	1374ms	1428ms
128KB	3338ms	2989ms	2595ms
256KB	6550ms	5405ms	4836ms

(a) Completion time

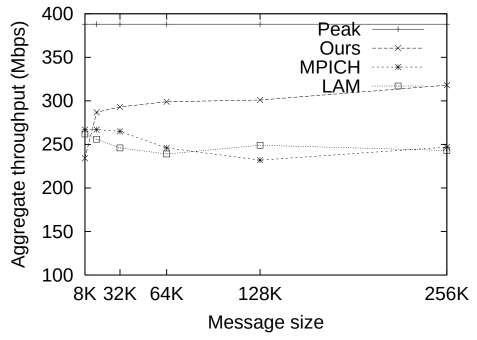


(b) Aggregate throughput

Figure 7: Results *MPI_Alltoall* on the topology in Figure 5 (b)

msize	LAM	MPICH	Ours
8KB	242ms	238ms	271ms
16KB	495ms	476ms	443ms
32KB	1034ms	958ms	868ms
64KB	2127ms	2061ms	1700ms
128KB	4080ms	4379ms	3372ms
256KB	8375ms	8210ms	6396ms

(a) Completion time



(b) Aggregate throughput

Figure 8: Results *MPI_Alltoall* on the topology in Figure 5 (c)

AAPC algorithm in the improved MPICH was designed specifically for this type of network [14]. Since the improved MPICH algorithm does not consider contention in network links, its performance depends heavily on the network topology when multiple switches are used. As shown in the results in Figure 8, for the topology in Figure 5 (c), the algorithm has a similar performance as the simple algorithm in LAM/MPI. Our automatically generated routine offers consistent better results when the message size is sufficient large on all topologies.

7 Conclusion

In this paper, we introduce a message scheduling algorithm for AAPC on Ethernet switched clusters. We demonstrate that our AAPC algorithm can utilize network bandwidths more effectively than existing AAPC implementations in LAM/MPI and MPICH. The proposed AAPC algorithm can be applied to other networks with a tree topology.

References

[1] S. Bokhari, “Multiphase Complete Exchange: a Theoretical Analysis,” *IEEE Trans. on Computers*, 45(2), 1996.

[2] H. G. Dietz, T. M. Chung, T. I. Mattox, and T. Muhammad, “Purdue’s Adapter for Parallel Execution and Rapid Synchronization: The TTL_PAPERS Design”, *Technical Report*, Purdue University School of Electrical Engineering, January 1995.

[3] V. V. Dimakopoulos and N.J. Dimopoulos, “Communications in Binary Fat Trees,” *ICDCS’95*, 1995.

[4] S. Hinrichs, C. Kosak, D.R. O’Hallaron, T. Stricker, and R. Take. An Architecture for Optimal All–to–All Personalized Communication. In *6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 310–319, June 1994.

[5] S. L. Johnsson and C. T. Ho, “Optimum Broadcasting and Personalized Communication in Hypercubes”, *IEEE Transactions on Computers*, Volumn 38, No. 9, pages 1249-1268, Sept. 1989.

[6] L. V. Kale, S. Kumar, K. Varadarajan, “A Framework for Collective Personalized Communication,” *International Parallel and Distributed Processing Symposium (IPDPS’03)*, April, 2003.

[7] LAM/MPI Parallel Computing. http://www.lam-mpi.org/.

[8] C. C. Lam, C. H. Huang, and P. Sadayappan, “Optimal Algorithms for All–to–All Personalized Communication on Rings and two dimensional Tori,” *Journal of Parallel and Distributed Computing*, 43(1):3-13, 1997.

[9] W. Liu, C. Wang, and K. Prasanna, “Portable and Scalable Algorithms for Irregular All–to–all Communication,” *16th ICDCS*, pages 428-435, 1996.

[10] The MPI Forum. *The MPI-2: Extensions to the Message Passing Interface*, July 1997. Available at http://www.mpi-forum.org/docs/mpi-20-html/ mpi2-report.html.

[11] R. Ponnusamy, R. Thakur, A. Chourdary, and G. Fox, “Scheduling Regular and Irregular Communication Patterns on the CM-5,” *Supercomputing*, pages 394-402, 1992.

[12] N.S. Sundar, D. N. Jayasimha, D. K. Panda, and P. Sadayappan, “Hybrid Algorithms for Complete Exchange in 2d Meshes,” *International Conference on Supercomputing*, pages 181–188, 1996.

[13] D.S. Scott, “Efficient All–to–All Communication Patterns in Hypercube and Mesh topologies,” *the Sixth Distributed Memory Computing Conference*, pages 398-403, 1991.

[14] A. Tam and C. Wang, “Efficient Scheduling of Complete Exchange on Clusters,” *the ISCA 13th International Conference on Parallel and Distributed Computing Systems*, August 2000.

[15] Andrew Tanenbaum, “Computer Networks”, 4th Edition, 2004.

[16] R. Thakur and A. Choudhary, “All-to-all Communication on Meshes with Wormhole Routing,” *8th International Parallel Processing Symposium (IPPS)*, 1994.

[17] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimizing of Collective Communication Operations in MPICH,” *ANL/MCS-P1140-0304*, Mathematics and Computer Science Division, Argonne National Laboratory, March 2004.

[18] E. A. Varvarigos and D. P. Bertsekas, “Communication Algorithms for Isotropic Tasks in Hypercubes and Wraparound Meshes,” *Parallel Computing*, Volumn 18, pages 1233-1257, 1992.