# A Message Scheduling Scheme for All-to-all Personalized Communication on Ethernet Switched Clusters [*]

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

**Abstract**

We develop a message scheduling scheme for efficiently realizing all–to–all personalized communication (AAPC) on Ethernet switched clusters with one or more switches. To avoid network contention and achieve high performance, the message scheduling scheme partitions AAPC into phases such that (1) there is no network contention within each phase; and (2) the number of phases is minimum. Thus, realizing AAPC with the contention-free phases computed by the message scheduling algorithm can potentially achieve the minimum communication completion time. In practice, phased AAPC schemes must introduce synchronizations to separate messages in different phases. We investigate various synchronization mechanisms and various methods for incorporating synchronizations into the AAPC phases. Experimental results show that the message scheduling based AAPC implementations with proper synchronization consistently achieve high performance on clusters with many different network topologies when the message size is large.

**Keywords**: All-to-all personalized communications, Ethernet, scheduling.

## 1 Introduction

All–to–all personalized communication (AAPC) is one of the most common communication patterns in high performance computing. In AAPC, each machine in a system sends a different message of the same size to every other machine. The Message Passing Interface (MPI) routine that realizes AAPC is *MPI_Alltoall* [15]. 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

[*]This work is partially supported by NSF grants ANI-0106706, CCR-0208892, CCF-0342540, and CCF-0541096.

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 for efficiently realizing AAPC on Ethernet switched clusters with one or more switches. Similar to other AAPC scheduling schemes [6], our scheme partitions AAPC into contention-free phases and fully utilizes the bandwidth in the bottleneck links in all phases. Hence, realizing AAPC with the contention-free phases can potentially achieve the minimum communication completion time. In practice, phased AAPC schemes must introduce synchronizations to separate communications in different phases. We investigate various synchronization mechanisms and various methods for incorporating synchronizations into the AAPC phases, and discuss the variations of AAPC implementations that are based on the AAPC phases computed by the message scheduling algorithm. For each of the variations, we develop an automatic routine generator that takes the topology information as input and produces a customized *MPI_Alltoall* routine. We evaluate the automatically generated routines and compare them with the original *MPI_Alltoall* routine in LAM/MPI [10] and the recently improved MPICH [23]. The results show that the message scheduling based AAPC implementations with proper synchronization consistently achieve high performance on clusters with many different network topologies when the message size is large.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the network model. Section 4 details the proposed scheduling scheme. Section 5 discusses issues and variations of the message scheduling based implementations. 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 parallel machines, including hypercube [7, 24], mesh [1, 18, 17, 22], torus [6, 11], k-ary n-cube [24], and fat tree [3, 16]. Heuristic algorithms were developed for AAPC on irregular topologies [5, 14]. A framework for AAPC that is realized with indirect communications was reported in [8]. Efficient AAPC scheduling schemes for clusters connected by a single switch was proposed in [19]. Some of the algorithms in [19] are incorporated in the recent improvement of the MPICH library [23]. Contention-aware AAPC schemes for hierarchical networks were studied in [20]. Many techniques for optimizing other communication operations using contention-free communications on switch-based clusters were also developed (see for example [12, 13]). 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. Many advanced communication systems [9, 25] can take advantage of the algorithms developed in this paper.

## 3 Network Model

We consider homogeneous Ethernet switched clusters, where both nodes and links in the system are homogeneous. Links operate in the duplex mode that allows each machine to 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 [21]. As a result, the physical topology is always a **tree**. There is 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 directed edges $E$ corresponding to unidirectional channels. Since all edges are directed, we will use the terms *edge* and *directed edge* interchangeably. Let $S$ be the set of switches and $M$ be the set of machines. $V = S \cup M$. Let $u, v \in V$, a directed **edge** $(u, v) \in E$ if and only if there is a direct 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. A machine $u \in M$ can only be a leaf node and a switch $s \in S$ can only be an internal node. Figure 1 shows an example cluster.

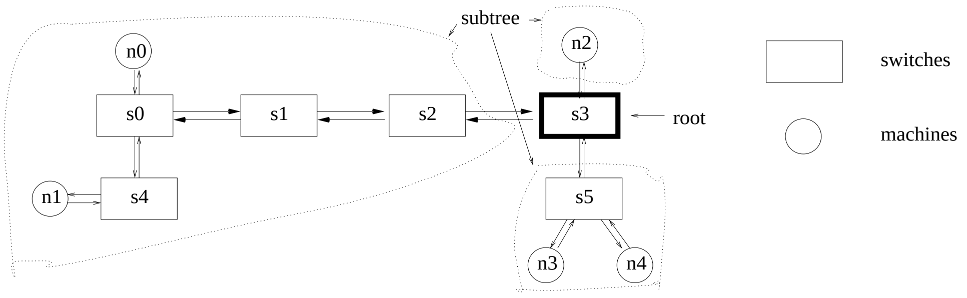


Figure 1: An example Ethernet Switched Cluster

The terminologies used in this paper are defined next. A *message*, $u \rightarrow v$, is a communication to be transmitted from node $u$ to node $v$. 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*$(n0, n1) = \{(n0, s0), (s0, s4), (s4, n1)\}$. Two messages, $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$, are said to have contention if they share a common directed edge, that is, there exists a directed 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 $\{u \rightarrow v | u \neq v \wedge u \in M \wedge v \in M\}$. 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. The *load of a pattern* is equal to the load of a bottleneck edge. Since the topology is a tree, for the AAPC pattern, edges $(u, v)$ and $(v, u)$ always have the same load. Since we only consider AAPC in this paper, we will use the terms "the load of an edge $(u, v)$" and "the load of a link $(u, v)$" interchangeably. A bottleneck edge on a graph refers to a bottleneck edge for the

AAPC pattern unless specified otherwise. For a set $S$, $|S|$ denotes the size of the set. The message size in the AAPC pattern is denoted as $msize$. 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. Assume that removing link $(u, v)$ (edges $(u, v)$ and $(v, u)$) from $G$ results in two connected components $G_u = (S_u \cup M_u, E_u)$ 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 of data to be transferred across $(u, v)$ 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

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

In general networks, the peak aggregate throughput may not be achievable. However, this physical limit can be approached through message scheduling for the tree topology.

## 4 AAPC Message Scheduling

In the following, we will present an algorithm that computes phases for AAPC. The phases conform to the following constraints, which are sufficient to guarantee optimality: (1) no contention within each phase; and (2) the total number of phases is equal to the load of AAPC on a given topology. In theory, when phases that satisfy these constraints are carried out without inter-phase interferences, the peak aggregate throughput is achieved. In practice, synchronizations must be used to separate the communications in different phases. We will focus on computing the contention-free AAPC phases in this section. Practical issues including different synchronization mechanisms and different ways to incorporate synchronizations into the AAPC phases will be discussed in the next section.

The scheduling algorithm has three major steps. In the first step, the algorithm identifies a *root* of the system. For a graph $G = (S \cup M, E)$, a *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. Note that a subtree of the root is a connected component after the root is removed from G. Once the root is identified, messages in AAPC are classified in two levels: *local messages* that are within a subtree, and *global messages* that are between subtrees. In the second step, the algorithm allocates phases for global messages. Finally, the third step assigns a phase to each of the local and global messages.

## 4.1 Identifying a 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 $G$ into two connected components, $G_u = (S_u \cup M_u, E_u)$ and $G_v = (S_v \cup M_v, E_v)$. The load of link $L$ is thus, $|M_u| \times |M_v| = (|M| - |M_v|) \times |M_v|$. Let us 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 an arbitrary $n$ 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 $G$ when $|M| \geq 3$.

**Lemma 1**: Each subtree of the root contains at most $\frac{|M|}{2}$ machines.

*Proof:* Using the process described above, 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 connected components after link $(u_n, u_{n-1})$ is removed from $G$. We have $|M_{u_n}| \geq |M_{u_{n-1}}|$, which implies $|M_{u_{n-1}}| \leq \frac{|M|}{2}$. The load on the bottleneck link $(u_n, u_{n-1})$ is $|M_{u_n}| \times |M_{u_{n-1}}|$. Let node $w$ be any 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 $(u_n, w)$ is greater than the load on link $(u_n, u_{n-1})$ $(|M_w| \times (|M| - |M_w|) > |M_{u_{n-1}}| \times (|M| - |M_{u_{n-1}}|))$, which contradicts 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. $\square$

In Figure 1, links $(s0, s1)$, $(s1, s2)$, and $(s2, s3)$ are bottleneck links. Let us assume that $(s1, s2)$ is initially selected to start the process. Removing $(s1, s2)$ yields two connected components: $G_{s1} = (S_{s1} \cup M_{s1}, E_{s1})$ and $G_{s2} = (S_{s2} \cup M_{s2}, E_{s2})$. Since $3 = |M_{s2}| > |M_{s1}| = 2$ and $s2$ has one branch to $s3$ in $G_{s2}$, the process will consider bottleneck link $(s2, s3)$. Removing this link results in two connected components $G_{s3} = (S_{s3} \cup M_{s3}, E_{s3})$ and $G_{s2} = (S_{s2} \cup M_{s2}, E_{s2})$. Since $|M_{s3}| > |M_{s2}|$ and $s3$ has two branches in $G_{s3}$, one to machine $n2$ and the other one to $s5$, switch $s3$ is identified as the root.

In the rest of the paper, we will assume that the root connects 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 system. Without loss of generality, let us assume that $|M_0| \geq |M_1| \geq ... \geq |M_{k-1}|$. Thus, the load of AAPC is $|M_0| \times (|M_1| + |M_2| + ... + |M_{k-1}|) = |M_0| \times (|M| - |M_0|)$.

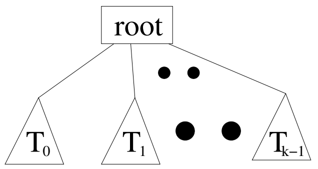


Figure 2: A two level view of the system

## 4.2 Global Message Scheduling

Global messages are messages between machines in different subtrees. We will use the notation $T_i \rightarrow T_j$ to represent the set of messages from machines in subtree $T_i$ to machines in subtree $T_j$. In global message scheduling, all messages in $T_i \rightarrow T_j$ are grouped together and allocated in consecutive phases. Since each message in $T_i \rightarrow T_j$ uses the edge from $T_i$ to the root, to avoid contention, each message in $T_i \rightarrow T_j$ must occupy a different phase. Since there are a total of $|M_i| \times |M_j|$ messages in $T_i \rightarrow T_j$, the global message scheduling scheme allocates $|M_i| \times |M_j|$ continuous phases for $T_i \rightarrow T_j$. The phases are allocated as follows.

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

Figure 3 shows the scheduling of global messages for the example in Figure 1. In this figure, $T_0$ contains two machines $n0$ and $n1$; $T_1$ contains two machines $n3$ and $n4$; and $T_2$ contains one machine $n2$. $|M_0| = 2$, $|M_1| = 2$, and $|M_2| = 1$. Messages in $T_1 \rightarrow T_2$ start at $|M_1| \times \sum_{n=2}^{1} |M_n| = 0$. Messages in $T_0 \rightarrow T_2$ start at $|M_0| \times \sum_{n=1}^{1} |M_n| = |M_0| \times |M_1| = 4$. Messages in $T_2 \rightarrow T_0$ start at $|M_0| \times (|M| - |M_0|) - |M_0| \times \sum_{n=1}^{2} |M_n| = 0$.

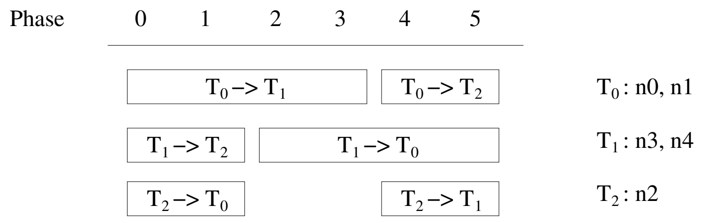


Figure 3: Global message scheduling for the example in Figure 1

**Lemma 2**: Using the global message scheduling scheme described above, the resulting phases have the following two properties: (1) the number of phases allocated is $|M_0| \times (|M| - |M_0|)$; and (2) in each phase, each subtree is allocated to send at most one global message and receive at most one global message.

*Proof*: The first property can be verified by examining phases allocated to all $T_i \rightarrow T_j$, $i \neq j$. For the second property, it can be shown that, for any subtree $T_i$, (1) phases allocated to $T_i \rightarrow T_j$, $j \neq i$, do not overlap; and (2) phases allocated to $T_j \rightarrow T_i$, $j \neq i$, do not overlap. We will leave the details to the reader. Since each message in $T_i \rightarrow T_j$ occupies a different phase, there can be at most one global message sent from $T_i$ and one global message sent to $T_i$ in each phase. $\square$

## 4.3 Global and Local Message Assignment

The global and local message assignment decides the phase for each message. The following lemma, which is the foundation of our assignment scheme, states that in a tree topology, a message sent to a node does not have contention with a message sent from the node regardless of the source of the message to the node and the destination of the message from the node.

**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 the assumption that $G$ is a tree. □

### 4.3.1 Handling global messages

**Lemma 4**: There is no contention among global messages.

*Proof:* From Lemma 2, there is at most one global message sent from and to each subtree in any phase. The global message that is sent from the subtree will go through the root first (before reaching another subtree) and the global message that is sent to the subtree must also go through the root. From Lemma 3, these two messages will not have contention within the subtree and its link to the root. Since this conclusion applies to all subtrees in all phases, there is no contention among the global messages. □

Lemma 4 indicates that as long as global messages in $T_i \rightarrow T_j$ are assigned to the phases allocated to $T_i \rightarrow T_j$, there will be no contention among the global messages. Let the machines in subtree $T_i$ be $m_{i,0}$, $m_{i,1}$, ..., $m_{i,(|M_i|-1)}$. To realize the global messages in $T_i \rightarrow T_j$, $0 \leq i \neq j < k$, each message $m_{i,i_1} \rightarrow m_{j,j_1}$, $0 \leq i_1 < |M_i|$ and $0 \leq j_1 < |M_j|$, must happen in the $|M_i| \times |M_j|$ phases that are allocated to $T_i \rightarrow T_j$. Our assignment algorithm uses two different methods to realize inter-subtree global communications. The first scheme is what we refer to as a *broadcast* scheme. In this scheme, the $|M_i| \times |M_j|$ phases are partitioned into $|M_i|$ rounds with each round having $|M_j|$ phases. In each different round, a different machine in $T_i$ sends one message to each of the machines 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 the $k$th round realize the broadcast from node $m_{i,k}$ to all nodes in $T_j$, which may result in the following pattern:

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

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 $|M_i| = a \times D$, $|M_j| = b \times D$. Table 1 shows an example of the rotate pattern when $|M_i| = 6$ and $|M_j| = 4$. In this case, $a = 3$, $b = 2$, and $D = 2$. In this scheme, the pattern for receivers is a repetition of $M_i$ times of a fixed sequence that enumerates all machines in $T_j$. In the example in Table 1, the fixed receiver sequence is $m_{j,0}, m_{j,1}, m_{j,2}, m_{j,3}$, which results in the receiver pattern of the following:

| phase | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ... |
|---|---|---|---|---|---|---|---|---|---|
| receiver | $m_{j,0}$ | $m_{j,1}$ | $m_{j,2}$ | $m_{j,3}$ | $m_{j,0}$ | $m_{j,1}$ | $m_{j,2}$ | $m_{j,3}$ | ... |

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, In Table 1, the base sequence for the senders is $m_{i,0}, m_{i,1}, m_{i,2}, m_{i,3}, m_{i,4}, m_{i,5}$. In the scheduling, the base sequence and the "rotated" base sequence are used. Let the base sequence be $m_{i,0}, m_{i,1}, ...m_{i,|M_i|-1}$. The base sequence can be rotated 1 time, which produces the sequence $m_{i,1}, ...m_{i,|M_i|-1}, m_{i,0}$. Sequence $m_{i,2}, ...m_{i,|M_i|-1}, m_{i,0}, m_{i,1}$ is the result of rotating the base sequence 2 times. The result from rotating the base sequence an arbitrary number of times can be defined similarly. The senders are scheduled as 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 $|M_i| \times |M_j|$ phases. Basically, at phases whose numbers are multiples of $a \times b \times D$, rotations are performed to find

Table 1: Rotate pattern for realizing $T_i \rightarrow T_j$ when $|M_i| = 6$ and $|M_j| = 4$

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

a new sequence. In Table 1, the base sequence is repeated $b = 2$ times. After that, a rotated sequence for the senders $m_{i,1}, m_{i,2}, m_{i,3}, m_{i,4}, m_{i,5}, m_{i,0}$ is repeated 2 times. It can be verified that all messages in $T_i \rightarrow T_j$ are realized in the rotate scheme. The following two lemmas, derived from the definitions, state the related properties of these two patterns.

**Lemma 5**: In the broadcast pattern that realizes $T_i \rightarrow T_j$, each sender $m_{i,n}$, $0 \leq n < |M_i|$, occupies $|M_j|$ continuous phases. $\square$

**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. $\square$

### 4.3.2 Handling local messages

Consider subtree $T_i$, the total number of local messages in $T_i$ is $|M_i| \times (|M_i| - 1)$, which is less than $|M_0| \times (|M| - |M_0|)$ since $|M_i| \leq \frac{|M|}{2}$ (Lemma 1). Thus, for each subtree, it is sufficient to schedule one local message in each phase. Let $u \rightarrow v$ be a local message in $T_i$. From Lemma 3, there are four cases when this local message can be assigned without contention (with global messages) in a phase. The cases are summarized in Table 2. Note that by assigning at most one local message in each subtree in a phase, there is no possibility of contention between local messages and the algorithm does not have to consider the specific topologies of the subtrees. The challenge in the local and global message assignment is that the global messages must be assigned in such a way that each of the local messages can have a case in Table 2.

### 4.3.3 The assignment algorithm

The detailed global and local message assignment 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 $m_{j,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_j|}$ is the receiver. 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 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 in Step 5, this alignment is needed to correctly schedule local messages. Using the rotate pattern 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 where each round has $|M_0|$ phases starting from phase 0. Thus, phases 0 to $|M_0| - 1$ belong to round 0, phases $|M_0|$ to $2 \times |M_0| - 1$ belong to round 1, and so on. 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 (for sending and receiving global messages) 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. For example, in a phase in round 0, if $m_{0,0}$ is the sender (decided in step 1), then $m_{0,1}$ will be the receiver in this phase. After the receiver

Table 2: Four cases for scheduling a local message $u \rightarrow v$ in $T_i$ without causing contention

| | |
|---|---|
| Case (1): | Node $v$ is the sender of a global message and node $u$ is the receiver of a global message. |
| Case (2): | Node $v$ is the sender of a global message and there is no receiving node of a global message in $T_i$. |
| Case (3): | Node $u$ is the receiver of a global message and there is no sending node of a global message. |
| Case (4): | There is no sending node and no receiving node of global messages in $T_i$. |

```
**Input**: Results from global message scheduling that identify which phases are used to
        realize $T_i \rightarrow T_j$ for all $0 \leq i \neq j < k$
**Output**: (1) the phase to realize each global message
            $m_{i,i_1} \rightarrow m_{j,j_1}$, $0 \leq i_1 < |M_i|$, $0 \leq j_1 < |M_j|$, $0 \leq i \neq j < k$.
        (2) the phase to realize each local message $m_{i,i_1} \rightarrow m_{i,i_2}$, $0 \leq i_1 \neq i_2 < |M_i|$, $0 \leq i < k$.

Step 1: Assign phases to messages in $T_0 \rightarrow T_j$, $1 \leq j < k$.
    1.a: For each $T_0 \rightarrow T_j$, the receivers in $T_j$ are assigned as follows:
        at phase $p$ in the phases for $T_0 \rightarrow T_j$, machine $m_{j,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_j|}$ 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 $T_0 \rightarrow T_j$. */
    1.b: For each $T_0 \rightarrow T_j$, the senders in $T_0$ are assigned according to the rotate pattern with
            the base sequence $m_{0,0}, m_{0,1}, ..., m_{0,|M_0|-1}$.

Step 2: Assign phases to messages in $T_i \rightarrow T_0$, $1 \leq i < k$.
    2.a: Assign the receivers in $T_i \rightarrow T_0$:
        /*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 $T_i \rightarrow T_0$ is computed based on the sender pattern in $T_0 \rightarrow T_j$ 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 $m_{i,0}$, $m_{i,1}$, ..., $m_{i,|M_i|-1}$.

Step 3: Schedule local messages in $T_0$ in phase 0 to phase $|M_0| \times (|M_0| - 1)$.
    message $m_{0,i} \rightarrow m_{0,j}$, $0 \leq i \neq j < |M_0|$, is scheduled at the phase where $m_{0,i}$ is the receiver
    of a global message and $m_{0,j}$ is the sender of a global message.

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

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

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

Figure 4: The global and local message assignment algorithm

pattern is decided, the senders of $T_i \rightarrow T_0$ are determined using the broadcast scheme with the sender order $m_{i,0}$, $m_{i,1}$, ..., $m_{i,|M_i|-1}$.

Step 3 embeds local messages in $T_0$ in the first $|M_0| \times (|M_0| - 1)$ phases. Note that $|M_0| \times (|M_0| - 1) \leq |M_0| \times (|M| - |M_0|)$ since $|M_0| \leq \frac{|M|}{2}$. Since the global messages for nodes in $T_0$ are scheduled according to Table 3, for each $m_{0,n} \rightarrow m_{0,m}$, $0 \leq n \neq m < |M_0|$, there exists a phase in the first $|M_0| \times (|M_0| - 1)$ phases such that $m_{0,n}$ is scheduled to receive a global message while $m_{0,m}$ is scheduled to send a global message. Thus, all local messages in $T_0$, $m_{0,n} \rightarrow m_{0,m}$, $0 \leq n \neq m < |M_0|$, can be scheduled in the the first $|M_0| \times (|M_0| - 1)$ 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 $m_{j,0}$, $m_{j,1}$, ..., $m_{j,|M_j|-1}$ and senders following the order $m_{i,0}$, $m_{i,1}$, ..., $m_{i,|M_i|-1}$. Hence, messages in $T_i \rightarrow T_j$, $i > j$ and $j \neq 0$ are assigned as

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

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 $|M_i| \times |M_{i-1}|$ phases for messages in $T_i \rightarrow T_{i-1}$, which is more than the $|M_i| \times (|M_i| - 1)$ 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

Table 3: Mapping between senders and the receivers in Step 2

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

the first phase for $T_0 \rightarrow T_n$ (since $|M_n| \times |M_{n-1}| \leq |M_0| \times |M_n|$) 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 $m_{i,i_2} \rightarrow m_{i,i_1}$ is scheduled. From Step 4, the receiver in $T_j \rightarrow T_i$, $j > i$, is organized such that, at phase $p$, $m_{i,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_i|}$ is the receiver. From Step 1, receivers in $T_0 \rightarrow T_i$ are also aligned such that at phase $p$, $m_{i,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_i|}$ is the receiver. Hence, in the phases for $T_i \rightarrow T_{i-1}$, either $m_{i,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_i|}$ 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 $m_{i,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_i|}$ at phase $p$. In other words, at phase $p$, $m_{i,(p-|M_0|\times(|M|-|M_0|))\ mod\ |M_i|}$ 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 $m_{i,i_1}$ is sending in $|M_{i-1}|$ continuous phases. Since the receiving pattern covers every node, $m_{i,i_2} \in T_i$, in every $|M_i|$ continuous phases and $|M_{i-1}| \geq |M_i|$, there exists at least one phase where $m_{i,i_1}$ is sending a global message and $m_{i,i_2}$ is the designated receiver of a global message. Local message $m_{i,i_2} \rightarrow m_{i,i_1}$ 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 the 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*: It is obvious that all global and local messages are assigned to phases that are allocated to the global messages. From Lemma 2, all messages are realized in $|M_0| \times (|M| - |M_0|)$ phases. For each subtree, the algorithm assigns, in one phase, at most one global message sent from the subtree, one global message sent to the subtree, and one local message. It can be verified that one of the four cases in Table 2 applies for the assignment of the local message. From Lemma 3, there is no contention between local and global messages. Since there is no contention among global messages (Lemma 4), there is no contention within each 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 $m_{0,0} = n0$, $m_{0,1} = n1$, $m_{1,0} = n3$, $m_{1,1} = n4$, and $m_{2,0} = n2$. From the algorithm, we first determine the receiver pattern in $T_0 \rightarrow T_1$ and $T_0 \rightarrow T_2$. For messages in $T_0 \rightarrow T_1$, $m_{1,(p-6) \ mod \ 2}$ is the receiver at phase $p$, which means the receiver pattern from phase 0 to phase 3 is $m_{1,0}$, $m_{1,1}$, $m_{1,0}$, and $m_{1,1}$. After that, the rotate pattern is used to realize all messages in $T_0 \rightarrow T_1$. The results are shown in the second column in the table. 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 two phases). Since the sender pattern in the first round is $m_{0,0}$ and $m_{0,1}$, according to Table 3, the receiver pattern should be $m_{0,1}$ and $m_{0,0}$. 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 $2 \times 1 = 2$ phases according to the assignment of the sender and receiver of global messages in each phase. 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 $T_1 \rightarrow T_0$ (phase 2 to phase 5). Counting phases from the last phase (phase 5), the algorithm ensures that each machine in $T_1$ appears as the designated receiver in every $|M_1| = 2$ consecutive phases and that each machine in $T_1$ sends a global message in $|M_0| = 2$ consecutive phases. This arrangement allows all local messages to be assigned without causing contention. Finally, in Step 6, we use the broadcast pattern for messages in $T_1 \rightarrow T_2$.

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

| | global messages | | | local messages | | |
|---|---|---|---|---|---|---|
| phase | $T_0 \rightarrow \{T_1, T_2\}$ | $T_1 \rightarrow \{T_2, T_0\}$ | $T_2 \rightarrow \{T_0, T_1\}$ | $T_0$ | $T_1$ | $T_2$ |
| 0 | $m_{0,0} \rightarrow m_{1,0}$ | $m_{1,0} \rightarrow m_{2,0}$ | $m_{2,0} \rightarrow m_{0,1}$ | $m_{0,1} \rightarrow m_{0,0}$ | | |
| 1 | $m_{0,1} \rightarrow m_{1,1}$ | $m_{1,1} \rightarrow m_{2,0}$ | $m_{2,0} \rightarrow m_{0,0}$ | $m_{0,0} \rightarrow m_{0,1}$ | | |
| 2 | $m_{0,1} \rightarrow m_{1,0}$ | $m_{1,0} \rightarrow m_{0,0}$ | | | | |
| 3 | $m_{0,0} \rightarrow m_{1,1}$ | $m_{1,0} \rightarrow m_{0,1}$ | | | $m_{1,1} \rightarrow m_{1,0}$ | |
| 4 | $m_{0,0} \rightarrow m_{2,0}$ | $m_{1,1} \rightarrow m_{0,1}$ | $m_{2,0} \rightarrow m_{1,0}$ | | $m_{1,0} \rightarrow m_{1,1}$ | |
| 5 | $m_{0,1} \rightarrow m_{2,0}$ | $m_{1,1} \rightarrow m_{0,0}$ | $m_{2,0} \rightarrow m_{1,1}$ | | | |

## 5 Message scheduling based AAPC implementations

One naive method to achieve contention-free AAPC is by separating the contention-free phases computed by the message scheduling algorithm using barrier synchronizations. In theory, this implementation achieves contention-free communication for AAPC. In practice, there are two major limitations in this implementation. First, the barrier synchronizations would incur substantial synchronization overheads unless special hardware for the barrier operation such as the Purdue PAPERS [2] is available. Second, using barriers to separate all phases may be overly conservative in allowing the data to be injected into the network. Most network systems have some mechanisms such as buffering to resolve contention. Allowing the network system to resolve a limited degree of contention usually results in a better utilization of network resources than resolving contention at the user layer with barriers. Hence, it may be more efficient to use the contention-free phases to *limit contention* instead of to totally eliminate contention. To address the first limitation, other synchronization mechanisms with less overheads such as the pair-wise synchronization can be used to replace the barriers. To address the second limitation, the separation of the communications in different phases may only be partially enforced (or not enforced) instead of being fully enforced. These issues give rise to many variations in how the contention-free AAPC phases can be used to realize AAPC efficiently. Note that synchronization messages can also cause contention. However, we ignore such contention since synchronization messages are small and such contention can usually be resolved by the network system effectively.

We will discuss the variations of message scheduling based AAPC schemes that we use to evaluate the proposed message scheduling algorithm. We will classify a scheme as *fully synchronized* when a synchronization mechanism is used to separate each pair of messages (in different phases) that have contention, *partially synchronized* when a synchronization mechanism is only used to limit the potential network contention, or *not synchronized* when no synchronization mechanism is employed. The implementations that we consider include schemes with no synchronizations, fully and partially synchronized schemes with pair-wise synchronizations, and fully and partially synchronized schemes with barrier synchronizations. Next, we will describe the implementations.

**Implementations with no synchronizations**

The simplest scheme is to use the contention-free phases to order the send and receive operations without introducing any synchronization mechanism. Ordering the messages according to the contention-free phases may reduce the network contention in comparison to other arbitrary ordering of the messages. We will call this scheme the *no-sync.* scheme.

For systems with multiple switches, a machine may be idle in some phases. These idle machines may move messages from one phase to an earlier phase in the *no-sync.* scheme, which destroys the contention-free phase structure. Dummy messages can be added so that all machines are busy in all phases, which may improve the chance for maintaining the contention-free phase structure. Ideally, the dummy communications can happen between any two idle machines in a phase. However, allowing dummy communications between an arbitrary pair of machines significantly increases the complexity for scheduling the dummy messages. In our implementation, we take a simple approach that limits the dummy communications to be within one switch. Specifically, for each idle machine in a phase, the scheme tries to find another machine in the same switch that does not receive or does not send. If such a machine exists, a dummy communication between the two machines is created. If such a machine does not exist, a dummy self-communication (send to self) is inserted in the phase for the idle machine. We will call this scheme the *dummy* scheme.

**Implementations with pair-wise synchronizations**

With pair-wise synchronizations, the contention-free communications can be maintained by ensuring that two messages that have contention are carried out at different times. There are two ways to perform the pair-wise synchronizations: *sender-based* and *receiver-based.* In the sender-based synchronization, to separate messages $a \rightarrow b$ in phase $p$ and $c \rightarrow d$ in phase $q$,

$p < q$, the synchronization message $a \rightarrow c$ is sent after $a$ sends $a \rightarrow b$, and $c$ sends $c \rightarrow d$ only after it receives the synchronization message. In the receiver-based synchronization, the synchronization message $b \rightarrow c$ is sent after $b$ finishes receiving $a \rightarrow b$, and $c$ sends $c \rightarrow d$ only after it receives the synchronization message. The sender-based scheme is more aggressive in that the synchronization message may be sent before $a \rightarrow b$ completes. Thus, some data in $a \rightarrow b$ may reside in the network when $c \rightarrow d$ starts. The receiver-based scheme may be over-conservative in that the synchronization message is sent only after the data in $a \rightarrow b$ are copied into the application space in $b$.

We compute the required synchronizations for the fully synchronized scheme as follows. For every communication in a phase, we check if a synchronization is needed for every other communication at later phases and build a dependence graph, which is a directed acyclic graph. After deciding all synchronization messages for all communications, we compute and remove redundant synchronizations in the dependence graph. The redundant synchronizations are the ones that can be derived from other 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. Let $|M|$ and $|S|$ be the numbers of machines and switches respectively. The dependence graph contains $O(|M|^2)$ nodes. The complexity to build the graph is $O(|M|^4|S|^2)$ and the complexity to remove redundant synchronizations is $O(|M|^6)$. Since these computations are performed off-line, such complexity is manageable. 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.

In a partially synchronized scheme, the AAPC phases are partitioned into blocks of phases. The number of phases in a block, $bs$, is a parameter. Block 0 contains phases 0 to $bs - 1$, block 1 contains phases $bs$ to $2 \times bs - 1$, and so on. The partially synchronized schemes use synchronizations to separate messages in different blocks instead of phases. The order of communications within one block is not enforced. The required synchronizations in a partially synchronized scheme are computed by first computing the required synchronizations for the fully synchronized scheme and then removing the synchronizations within each block.

In summary, there are four types of implementations with pair-wise synchronizations. We will name them as follows: *sender all* for the fully synchronized scheme with sender-based synchronizations; *sender partial (bs)* for the partially synchronized scheme with sender-based synchronizations and the parameter *bs* (the number of phases in a block); *receiver all* for the fully synchronized scheme with receiver-based synchronizations; and *receiver partial (bs)* for the partially synchronized scheme with receiver-based synchronizations.

**Implementations with barrier synchronizations**

The fully barrier synchronized AAPC scheme is the one with a barrier between each pair of phases. In the partially barrier synchronized scheme, the AAPC phases are partitioned into blocks of phases. The number of phases in a block, *bs*, is a parameter. A barrier is added between each pair of blocks (one barrier every *bs* phases). There are three variations of partially barrier synchronized schemes: no synchronization within each block, sender-based pair-wise synchronization within each block, and receiver-based pair-wise synchronization within each block. We name these implementations with barriers as follows: *barrier all* for the fully synchronized scheme; *barrier partial & none (bs)* for the partially synchronized schemes with no synchronizations within each block; *barrier partial & sender (bs)* for the partially synchronized schemes with *sender all* within each block; *barrier partial & receiver (bs)* for the partially synchronized scheme with *receiver all* within each block.

## 6 Experiments

For each of the AAPC variations described in the previous section, we develop a routine generator that takes the topology information as input and automatically produces a customized *MPI_Alltoall* routine that employs the particular scheme for the given topology. The automatically generated routines run on MPICH 2-1.0.1 point-to-point primitives. We also use an automatic tuning system [4] to select from all of the message scheduling based schemes the best ones to form a tuned routine for each topology. Practically, the performance of the tuned routines represents the best performance that can be obtained from our message scheduling based implementations. Table 5 gives the names and brief descriptions of the schemes used in the evaluation. Note that although the tuning system can theoretically be used to carry out all the experiments, we only use it to generate the tuned routines. All experiments are performed by manually executing the algorithms.

Table 5: Message scheduling based AAPC schemes used in the evaluation

| Name (parameter) | description |
|---|---|
| *No-sync.* | no synchronization |
| *Dummy* | no synchronization with dummy communications for idle machines |
| *Sender all* | fully synchronized with sender-based pair-wise synchronizations |
| *Sender partial (bs)* | partially synchronized with sender-based pair-wise synchronizations |
| *Receiver all* | fully synchronized with receiver-based pair-wise synchronizations |
| *Receiver partial (bs)* | partially synchronized with receiver-based pair-wise synchronizations |
| *Barrier all* | fully synchronized with barrier synchronizations |
| *Barrier partial & none (bs)* | partially synchronized with barrier synchronizations, no synchronization within each block |
| *Barrier partial & sender (bs)* | partially synchronized with barrier synchronizations, *sender all* within each block of phases |
| *Barrier partial & receiver (bs)* | partially synchronized with barrier synchronizations, *receiver all* within each block of phases |
| *Tuned scheduling based* | the best implementation selected from all of the schemes above |

The message scheduling based schemes are compared with the original *MPI_Alltoall* routine in LAM/MPI 7.1.1 [10] and a recent improved MPICH 2-1.0.1 [23]. LAM/MPI 7.1.1 and MPICH 2-1.0.1 are compiled with the default setting. Both LAM/MPI and MPICH *MPI_Alltoall* routines are based on point-to-point primitives. Since LAM/MPI and MPICH have different point-to-point implementations, we also port the LAM/MPI algorithm to MPICH and report the performance of the ported routine, which will be referred to as LAM-MPICH. Hence, in the evaluation, message scheduling based implementations are compared with each other and with native LAM/MPI 7.1.1, native MPICH 2-1.0.1, and LAM-MPICH.

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 100Mbps Ethernet switches.

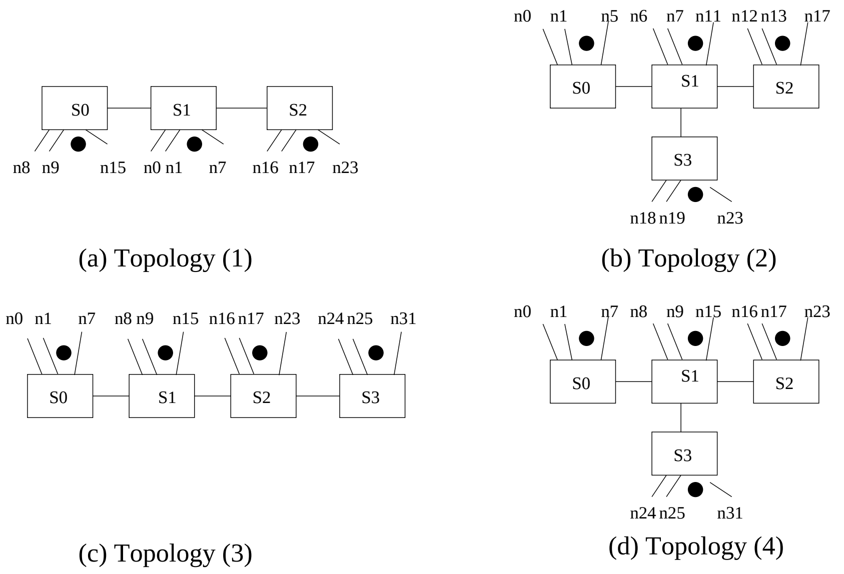
```
for (i=0; i< WARMUP_ITER; i++) MPI_Alltoall(...);
MPI_Barrier(...);
start = MPI_Wtime();
for (count = 0; count < ITER_NUM; count ++) { MPI_Alltoall(...); MPI_Barrier(...); }
elapsed_time = MPI_Wtime() - start;
```

Figure 5: Code segment for measuring the performance of *MPI_Alltoall*.

The code segment used in the performance measurement is shown in Figure 5. A barrier operation is performed after each all-to-all operation to ensure that the communications in different invocations do not affect each other. Since we only consider AAPC with reasonably large messages, the overhead introduced by the barrier operations is insignificant. The results reported are the averages of 50 iterations of *MPI_Alltoall* ($ITER\_NUM = 50$) when $msize \leq 256KB$ and 20 iterations when $msize > 256KB$.

The topologies used in the studied are shown in Figure 6, two 24-node clusters in Figure 6 (a) and Figure 6 (b) and two 32-node clusters in Figure 6 (c) and Figure 6 (d). We will refer to these topologies as topologies (1), (2), (3), and (4). The aggregate throughput, which is defined as $\frac{|M| \times (|M|-1) \times msize}{communication\ time}$, is used as the performance metric and is reported in all experiments.

Figures 7 compares the tuned scheduling based implementation with MPICH, LAM, and LAM-MPICH for topologies (1), (2), (3) and (4). In the figures, we also show the theoretical peak aggregate throughput as a reference. The peak aggregate throughput is obtained using the formula in Section 3, assuming a link speed of 100Mbps with no additional overheads. The algorithm in LAM/MPI does not perform any scheduling while the improved MPICH performs a limited form of scheduling. Both do not achieve high performance on all topologies since the network contention issue is not fully addressed in the implementations. On the


S0
S1
S2
n8 n9 n15 n0 n1 n7 n16 n17 n23
n0 n1 n5 n6 n7 n11 n12 n13 n17
S0
S1
S2
S3
n18 n19 n23
n0 n1 n7 n8 n9 n15 n16 n17 n23 n24 n25 n31
S0
S1
S2
S3
n0 n1 n7 n8 n9 n15 n16 n17 n23
S0
S1
S2
S3
n24 n25 n31

(a) Topology (1)

(b) Topology (2)

(c) Topology (3)

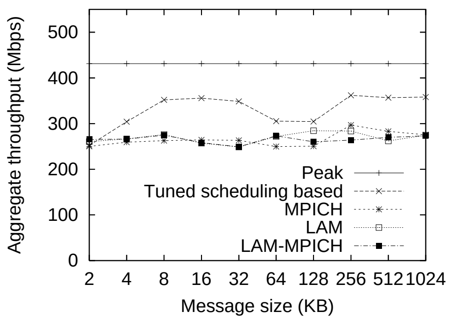
(d) Topology (4)

Figure 6: Topologies used in the evaluation

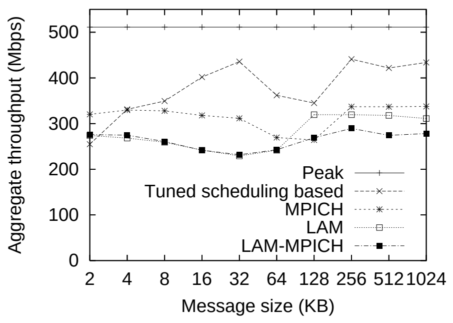
contrary, by introducing proper synchronization into the contention-free AAPC phases, the tuned scheduling based routine consistently achieve (sometimes significantly) higher performance than MPICH, LAM, and LAM-MPICH in the four topologies when the message size is larger than $4KB$. This demonstrates the strength of the message scheduling scheme.

Next, we will investigate different synchronization mechanisms and different methods to incorporate synchronizations into the contention-free phases in scheduling based AAPC implementations. The trends in the experimental results for the four topologies are somewhat similar. Thus, for each experiment, we will only report the results for two topologies.

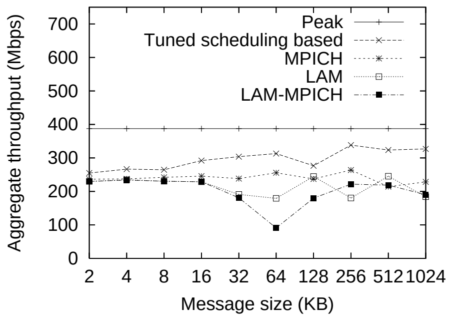
Figure 8 compares the receiver-based pair-wise synchronization with the sender-based pair-wise synchronization. When the message size is small, *receiver all* offers better performance. When the message size is large, the sender-based scheme gives better results. With the sender-based pair-wise synchronization, the AAPC scheme injects data into the network aggressively: a message $m_e$ in one phase may not be fully executed (the message may still be in the network system) before the next message $m_l$ that may have contention with $m_e$ starts. Hence, the sender-based scheme allows a limited form of network contention. On the other hand, using the receiver-based pair-wise synchronization, a message $m_l$ that may have contention



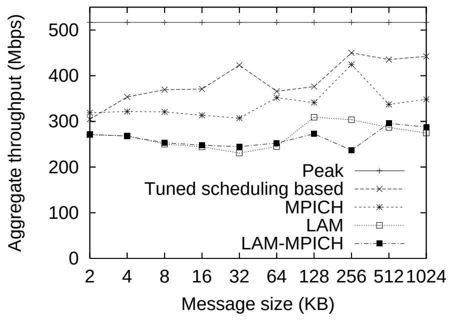
(a) Results for Topology (1)



(b) Results for Topology (2)



(c) Results for Topology (3)

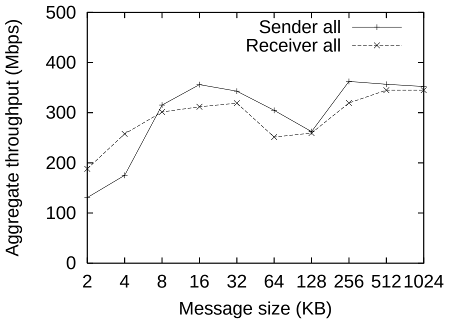


(d) Results for Topology (4)

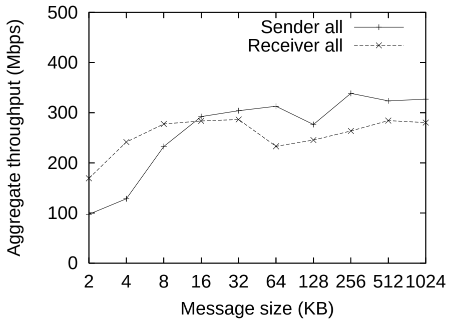
Figure 7: The performance of different AAPC implementations

with an earlier message $m_e$ can start only after the message in $m_e$ is received. The results indicate that the limited contention in the sender-based scheme can be resolved by the network system and the sender-based synchronization scheme offers better overall performance when the message size is reasonably large. Since the scheduling based implementations are designed for AAPC with reasonably large messages, we will use the send-based scheme for pair-wise synchronization in the rest of the evaluation.

Figure 9 compares the performance of message scheduling based AAPC schemes with different synchronization mechanisms, including *no-sync.*, *dummy*, *sender all*, and *barrier all.* The aggregate throughput achieved by *no-sync.* and *dummy* is much lower than that achieved by the fully synchronized schemes. Also, adding dummy communications to the idle machines seems to improve the performance over the *no-sync.* scheme in some situations (e.g. topology



(a) Results for Topology (1)



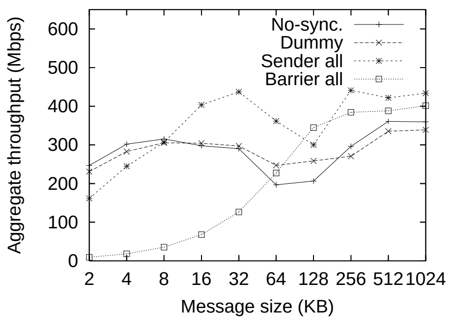
(b) Results for Topology (3)

Figure 8: Sender-based synchronization versus receiver-based synchronization

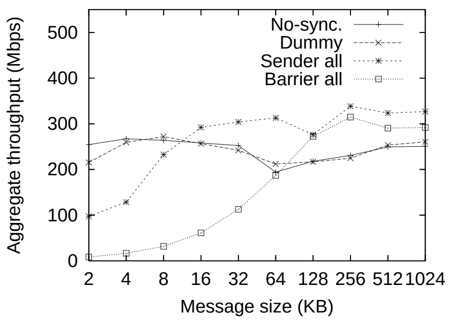
(2) with $msize = 64KB$) and to degrade the performance in some other situations. Due to the complexity of AAPC, it is unclear whether adding dummy communications is effective in maintaining the phase structure. The fully synchronized scheme with barriers incurs very large overheads when the message size is small. Even when the message size is large, *barrier all* still performs slightly worse than *sender all* in most cases. The 128KB case in Figure 9 (a) where *barrier all* out-performs *sender all* is an exception. It is difficult to decide the reason for this case: there are too many factors that can contribute to the performance. Yet, the trend clearly shows that the pair-wise synchronization is more efficient than the barrier synchronization in the implementation of the phased all-to-all communication algorithm.

Figure 10 compares the performance of partially synchronized schemes with sender-based pair-wise synchronizations, including *sender partial (2)*, *sender partial (8)*, and *sender partial (16)* with that of *no-sync.* and *sender all*. The trend in the figures is that as the message size increases, more synchronizations are needed to achieve high performance. The fully synchronized scheme performs the best when the message size is large ($\geq 32KB$). However, the partially synchronized schemes are more efficient for medium sized messages ($2KB$ to $16KB$) than both *no-sync.* and *sender all*.

Figure 11 shows the performance of different schemes with barrier synchronizations. When the message size is large, *Barrier partial & none (4)* performs similar to the *no-sync.* scheme.

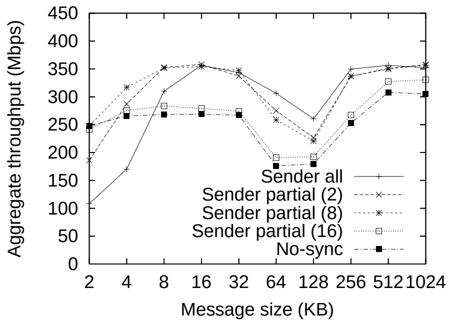


(a) Results for topology (2)

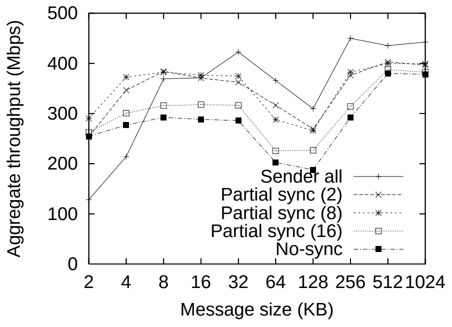


(b) Results for topology (3)

Figure 9: Message scheduling based schemes with different synchronization mechanisms

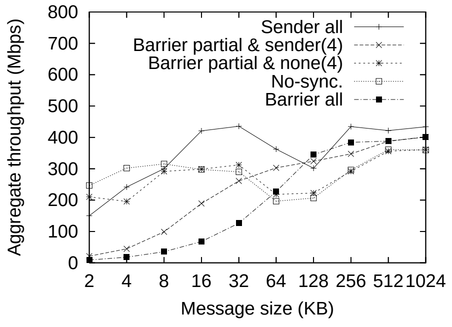


(a) Results for topology (1)

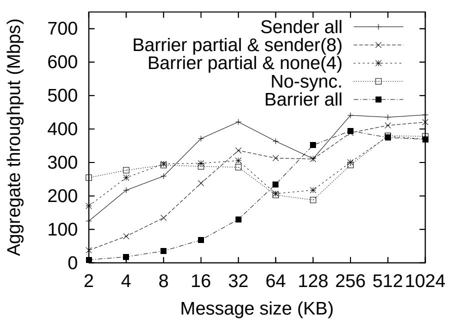


(b) Results for topology (4)

Figure 10: Partially synchronized schemes with sender-based pair-wise synchronizations

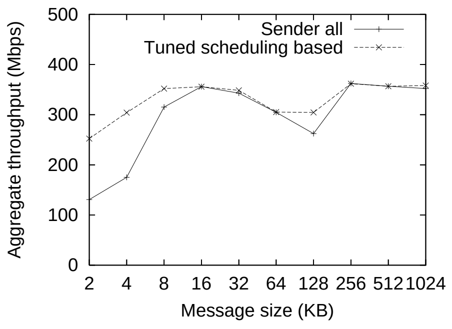


(a) Results for topology (2)

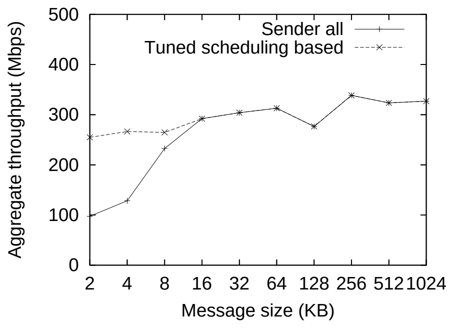


(b) Results for topology (4)

Figure 11: Schemes with barrier synchronizations

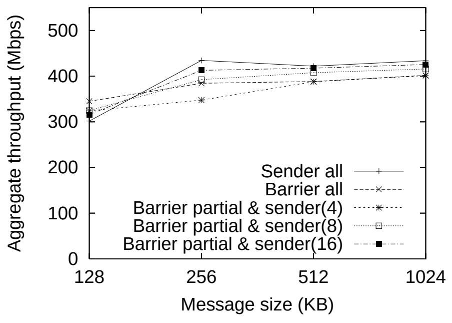


(a) Results for topology (1)

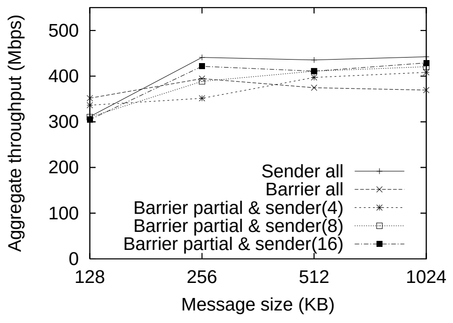


(b) Results for topology (3)

Figure 12: Performance of *Sender all* and *tuned scheduling based*



(a) Results for topology (2)



(b) Results for topology (4)

Figure 13: Performance of different fully synchronized schemes for large messages

When the message size is small, *Barrier partial & none (4)* incurs significant overheads. These results indicate that partially synchronized schemes with no synchronizations within each block are not effective. In all experiments, the hybrid barrier and sender-based pair-wise synchronizations never perform better than both *barrier all* and *sender all*, which indicates that such a combination may not be effective. The *sender all* scheme consistently achieves high performance when the message size is reasonably large. Figure 12 compares the performance of *sender all* and *tuned scheduling based*. The performance of *sender all* is very close to *tuned scheduling based* when the message size is larger than $16KB$.

Figure 13 shows the performance of different synchronization schemes for large messages. As discussed earlier, for large messages, fully synchronized schemes are more effective than partially synchronized schemes. Figure 13 shows the results for *sender all*, *barrier all*, *barrier partial & sender (4)*, *barrier partial & sender (8)*, and *barrier partial & sender (16)*. As can be seen from the figure, when the message size is larger than 512KB, the relative performance of these fully synchronized schemes is quite stable. Ordering the synchronization schemes based on the performance from high to low yields: *sender all*, *barrier partial & sender (16)*, *barrier partial & sender (8)*, and *barrier partial & sender (4)*, and *barrier all*. These results indicate that the sender-based pair-wise synchronization is sufficient even for large messages in the implementation. The heavy weight MPI barrier introduces more overheads without tangible benefits in realizing the phased all-to-all communication.

## 7 Conclusion

In this paper, we introduce a message scheduling algorithm for AAPC on Ethernet switched clusters that computes contention-free AAPC phases, and investigate practical issues in message scheduling based AAPC implementations, including various synchronization mechanisms and various methods for incorporating synchronizations into the contention-free phases. We demonstrate that the message scheduling based AAPC implementations with proper synchronization consistently achieve high performance on clusters with many different network topologies when the message size is sufficiently large. The performance may be further improved with hardware support for efficient barrier operation such as Purdue PAPERS [2].

## References

[1] S. Bokhari, "Multiphase Complete Exchange: a Theoretical Analysis." *IEEE Trans. on Computers*, 45(2):220-229, Feb. 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, Jan. 1995.

[3] V. V. Dimakopoulos and N.J. Dimopoulos, "Communications in Binary Fat Trees." *International Conference on Parallel and Distributed Computing Systems*, pages 383-388, Sept. 1995.

[4] A. Faraj and X. Yuan. "Automatic Generation and Tuning of MPI Collective Communication Routines." The *19th ACM International Conference on Supercomputing* (ICS'05), pages 393-402, Cambridge, MA, June 20-22, 2005.

[5] E. Gabrielyan and R. D. Hersch, "Network Topology Aware Scheduling of Collective Communications." The *10th International Conference on Telecommunications*, pages 1051-1058, Feb. 2003.

[6] 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.

[7] S. L. Johnsson and C. T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes." *IEEE Transactions on Computers*, 38(9):1249-1268, Sept. 1989.

[8] L. V. Kale, S. Kumar, K. Varadarajan, "A Framework for Collective Personalized Communication." *International Parallel and Distributed Processing Symposium* (CDROM), April, 2003.

[9] A. Karwande, X. Yuan, and D. K. Lowenthal, "An MPI Prototype for Compiled Communication on Ethernet Switched Clusters." *Journal of Parallel and Distributed Computing*, 65(10):1123-1133, October 2005.

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

[11] 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.

[12] R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan, "Optimal Contention-free Unicast-Based Multicasting in Switch-Based Networks of Workstations." The *11th International Parallel Processing Symposium*, pages 358-364, 1998.

[13] C. Lin, "Efficient Contention-free Broadcast in Heterogeneous Network of Workstations with Multiple Send and Receive Costs." The *8th IEEE International Symposium on Computers and Communications*, pages 1277-1284, 2003.

[14] W. Liu, C. Wang, and K. Prasanna, "Portable and Scalable Algorithms for Irregular All–to–all Communication." *Journal of Parallel and Distributed Computing*, 62(10):1493-1526, Oct. 2002

[15] 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.

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

[17] 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.

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

[19] A. Tam and C. Wang, "Efficient Scheduling of Complete Exchange on Clusters." The *ISCA 13th International Conference on Parallel and Distributed Computing Systems*, pages 111-116, August 2000.

[20] A. Tam, and C. Wang, "Contention-Aware Communication Schedule for High-Speed Communication." *Cluster Computing Journal*, 6(4):339-353, Oct. 2003.

[21] Andrew Tanenbaum, "Computer Networks." 4th Edition, Prentice Hall, 2004.

[22] R. Thakur and A. Choudhary, "All-to-all Communication on Meshes with Wormhole Routing." The *8th International Parallel Processing Symposium (IPPS)*, pages 561-565, 1994.

[23] 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.

[24] E. A. Varvarigos and D. P. Bertsekas, "Communication Algorithms for Isotropic Tasks in Hypercubes and Wraparound Meshes." *Parallel Computing*, 18(11):1233-1257, Nov. 1992.

[25] X. Yuan, R. Melhem and R. Gupta, "Algorithms for Supporting Compiled Communication." *IEEE Transactions on Parallel and Distributed Systems*, 14(2):107-118, February 2003.