Optimal Bucket Algorithms for Large MPI Collectives on Torus Interconnects

Nikhil Jain
IBM Research - India
New Delhi, India
nikhil.jain@in.ibm.com

Yogish Sabharwal
IBM Research - India
New Delhi, India
ysabharwal@in.ibm.com

ABSTRACT

Collectives are an important and frequently used component of MPI. Bucket algorithms, also known as "large vector" algorithms, were introduced in the early 90's and have since evolved as a well known paradigm for large MPI collectives. Many modern day supercomputers such as the IBM Blue Gene and Cray XT are based on torus interconnects that offer a highly scalable interconnection architecture for distributed memory systems. While near optimal algorithms have been developed for torus interconnects in other paradigms, such as spanning trees, bucket algorithms have not been optimally extended to these networks. In this paper, we study the basic "divide, distribute and gather" MPI collectives for bucket algorithms – Allgather, Reduce-scatter and Allreduce – for large messages on torus interconnects.

We present bucket-based algorithms for these collectives on bidirectional links. We show that these algorithms are optimal in terms of bandwidth and computation for symmetric torus networks (i.e. when all the dimensions are equal), matching the theoretical lower bounds For an asymmetric torus, our algorithms are asymptotically optimal and converge to the lower bound for large dimension sizes. We also argue that our bucket algorithms are more scalable on multicores in comparison to spanning tree algorithms. Previous studies of bucket algorithms on torus interconnects have focused on unidirectional links and have been unable to obtain tight lower bounds and optimal algorithms. We close this gap by providing stronger lower bounds and showing that our bidirectional algorithms can easily be adapted to the unidirectional case, matching our lower bounds in terms of bandwidth and computational complexity.

We implement our algorithms on the IBM Blue Gene/P Supercomputer, which has quad-core nodes connected in a 3-dimensional torus, using the low level communication interface. We demonstrate that our algorithms perform within 7-30% of the lower bounds for different MPI collectives. We demonstrate good scaling using multicores. We also demonstrate a factor of 3 to 17 speedup for various collectives in comparison to the latest optimized MPI implementation.

Categories and Subject Descriptors

D.m [**Software**]: Miscellaneous

General Terms

Algorithm, Performance

Keywords

MPI, Collective, Communication, Torus Network

1. INTRODUCTION

MPI Collectives are an important and frequently used component of MPI. As opposed to point-to-point communication, collective communication routines are a collection of MPI message passing routines that perform one-to-many, many-to-one and many-to-many communications amongst the processors. Basic collectives include synchronization collectives such as *Barrier*, data movement collectives, such as *Broadcast*, *Scatter*, *Gather*, *Allgather*, *All-to-all* and computation collectives, such as *Reduce*, *Reduce-scatter* and *Allreduce.*

The performance of MPI collectives is often critical in determining the performance of parallel scientific applications. A five-year profiling study of applications running in production mode on the Cray T3E 900 at the University of Stuttgart revealed that more than 40% of the time spent in MPI functions was spent in the two functions Allreduce and Reduce [13].

Simple yet powerful techniques have been proposed for collective communications (See [2, 3, 6, 8, 9, 10, 11, 12, 14, 15] and references therein). Bucket algorithms, also known as "large vector" algorithms, were introduced in the early 90's by van de Geijn et al. [2]. They have since evolved as a well known paradigm for large MPI collectives. These algorithms are based on the "divide, distribute and gather" paradigm for the Reduce-scatter and Allgather collectives. These combined with Gather and Scatter form the basic building blocks for the implementation of other collectives.

In this paper, we focus on the many-to-many collectives: *Reduce-scatter*, *Allgather* and *Allreduce* for commutative operations. The *Reduce-scatter* and *Allgather* collectives are the fundamental "divide, distribute and gather" collectives of the bucket algorithms paradigm. The *Allreduce* collective is derived by combining both these collectives under this framework.

Initial Data			Collective	Final Data		
Node 1	Node 2	Node 3		Node 1	Node 2	Node 3
a,b,c	d,e,f	g,h,i	Reduce-scatter	a+d+g	b+e+h	c+f+i
a	b	c	Allgather	a,b,c	a,b,c	a,b,c
a,b,c	d,e,f	g,h,i	Allreduce	a+d+g, b+e+h, c+f+i	a+d+g, b+e+h, c+f+i	a+d+g b+e+h c+f+i

Figure 1: Many-to-many MPI collectives

The operation of these collectives are illustrated in Figure 1 for a 3-node system. In the Reduce-Scatter collective, each node starts with an initial vector. The corresponding elements of all nodes are reduced and these reduced elements are then distributed across the nodes. In the Allgather collective, every node receives all elements from all other nodes. The Allreduce collective is similar to the Reduce-scatter collective except that all reduced elements are collected at all nodes instead of being distributed. Observe that the Allreduce collective operation is equivalent to performing a Reduce-scatter collective followed by an Allgather collective operation.

Bucket algorithms have been adapted and optimized for various communication networks, including torus interconnects. Torus interconnects are attractive interconnection architectures for distributed memory supercomputers. They are more scalable in comparison to competing architectures such as the hypercube and therefore many modern day supercomputers such as the IBM Blue Gene and Cray XT are based on these interconnects. Chan et al. [4] developed new models for the theoretical study of MPI collectives on these interconnects, wherein a node can communicate with multiple nodes simultaneously using the different links of the interconnect. They showed that the new models can lead to dramatic decreases in the perceived lower bounds of collective communications since a node can communicate with multiple nodes simultaneously. They also extend the bucket algorithms for torus interconnects. However, their theoretical bounds are not tight – there exists a considerable gap between their proposed lower bounds and algorithms.

In this paper, we tighten existing lower bounds for the simultaneous communication model proposed by Chan et al [4] and propose new algorithms under their model. However, their model only allows for communication in one direction on any link at a given point in time. For this reason, we refer to their model as the unidirectional model. We extend their model to support bidirectional communication on the links. We present lower bounds and algorithms for the bidirectional model as well. Figure 2 lists the lower bounds for torus interconnects under the two models. It also lists the complexity of our results and the previously best known bucket algorithms for these models. Since we are dealing with long vector algorithms, we concentrate on the bandwidth and processing bounds. For the unidirectional model, our bounds strengthen the previously known bounds by a factor of 2. For both the unidirectional and the bidirectional models, we show that our algorithms are asymptotically optimal for asymmetric torus networks, converging to the lower bound for large dimension sizes. For symmetric torus, our algorithms match the lower bound proposed exactly (not asymptotically) showing that for this case, our lower bounds are tight and our algorithms are optimal. We argue that our bucket algorithms are more scalable on multicores in comparison to spanning tree algorithms. We implement our algorithms on Blue Gene/P and demonstrate that they perform within 7-30% of the lower bounds for different MPI collectives. Our algorithms show significant speedups of factor of 3 to 17 over the currently optimized MPI implementation on Blue Gene/P. We also demonstrate good scaling using multicores.

The rest of this paper is organized as follows. In Section 2, we discuss related work with respect to the models of communication, bucket algorithms and spanning tree algorithms on torus interconnects. In Section 3, we present lower bounds for the unidirectional and bidirectional model. In Section 4, we present algorithms for both these models and also present optimizations that overlap computation and communication and use multicores. In Section 5, we present our experimental results on the Blue Gene/P Supercomputer. We summarize and conclude in Section 6.

2. RELATED WORK

In this Section we discuss related work on MPI Collectives. We start by discussing the communication models used for evaluating the performance of the collectives. We then describe the previously known bucket algorithms for "large vector" MPI collectives. We then describe the spanning-tree based approach for MPI collectives and discuss how they differ from the bucket algorithms.

2.1 Communication models

We first describe the *simultaneous communication model* proposed by Chan et al. [4] for completeness. Since all models we consider involve simultaneous communication, we refer to their model as the *unidirectional communication model* to avoid any confusion. We follow this up by changes proposed in this model to accomodate bidirectionality.

Unidirectional communication model: This model makes the following assumptions. It is targeted for distributed memory parallel architectures with p computational nodes, indexed from 0 to $p-1$. Each node has one computational processor. All processors are identical. Any node can send directly to any other node where a communication network provides automatic routing. The underlying interconnection network is an N-dimensional torus. Each node is directly connected to each of its $2N$ nearest neighbors where two are on opposing sides of a dimensional axis. The network is homogeneous, i.e., all links have the same bandwidth. At any given time, a single node can send or

Lower Bounds:

Collective	unidirectional model				bidirectional model	
	Known results		Our results		Our results	
	bandwidth	compute	bandwidth	compute	bandwidth	compute
Allgather	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	–	$\frac{p-1}{p}\cdot\frac{n\beta}{N}$	–	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	–
Reduce-scatter	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{p-1}{p}\cdot n\gamma$	$\frac{p-1}{p}\cdot\frac{n\beta}{N}$	$\frac{p-1}{p}\cdot n\gamma$	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{p-1}{p}\cdot n\gamma$
Allreduce	$2\cdot\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{p-1}{p}\cdot n\gamma$	$2\cdot\frac{p-1}{p}\cdot\frac{n\beta}{N}$	$\frac{p-1}{p}\cdot n\gamma$	$2\cdot\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{p-1}{p}\cdot n\gamma$

Algorithms:

Collective	Known results		Our results		
	bandwidth	compute	bandwidth		compute
			symmetric	asymmetric torus	
unidirectional model:					
Allgather	$\frac{d_\ell-1}{d_\ell}\cdot n\beta$	–	$\frac{p-1}{p}\cdot\frac{n\beta}{N}$	$\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	–
Reduce-scatter	$\frac{d_\ell-1}{d_\ell}\cdot n\beta$	$\frac{p-1}{p}\cdot n\gamma$	$\frac{p-1}{p}\cdot\frac{n\beta}{N}$	$\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	$\frac{p-1}{p}\cdot n\gamma$
Allreduce	$2\cdot\frac{d_\ell-1}{d_\ell}\cdot n\beta$	$\frac{p-1}{p}\cdot n\gamma$	$2\cdot\frac{p-1}{p}\cdot\frac{n\beta}{N}$	$2\cdot\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	$\frac{p-1}{p}\cdot n\gamma$
Bidirectional model:					
Allgather	–	–	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{2N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	–
Reduce-scatter	–	–	$\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{2N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	$\frac{p-1}{p}\cdot n\gamma$
Allreduce	–	–	$2\cdot\frac{p-1}{p}\cdot\frac{n\beta}{2N}$	$2\cdot\frac{d_\ell-1}{d_\ell}\cdot\frac{n\beta}{2N}\cdot\frac{d_s^N-1}{d_s^N}\cdot\frac{d_s}{d_s-1}$	$\frac{p-1}{p}\cdot n\gamma$

Figure 2: Comparison of algorithms and lower bounds with known results (with respect to bandwidth cost and compute cost)
p=number of processors, n=vector size, N=number of torus dimensions, β=bandwidth cost (transmission cost per element), γ=computation cost per arithmetic operation and d_ℓ (d_s resp.)=length of the longest (shortest) dimension of the torus.

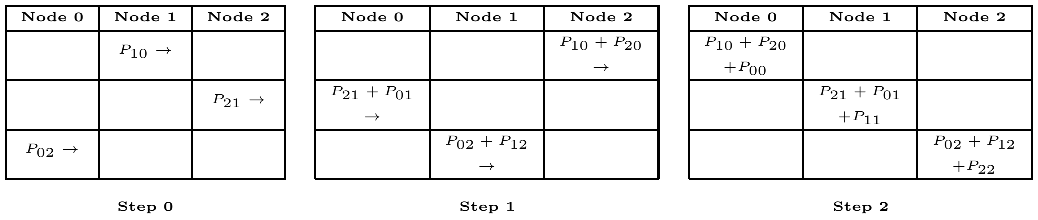
receive messages from $2N$ other nodes. A single node can do so simultaneously only if each of the messages are sent or received on each of its $2N$ different links. Note that a node is allowed to simultaneously participate in sends on some links and receives on other links.

In order to analyze the performance, the following notation is defined. Let α denote the startup cost incurred on sending a message and let β denote the bandwidth cost, i.e., the per data element transmission time on a link. Further, let γ denote the computation cost of performing an arithmetic operation, for instance the reduction of two elements. Typically the performance cost is broken down into three costs – the startup cost (terms involving α), the bandwidth cost (terms involving β) and the computation cost (terms involving γ), with the communication cost being the sum of the startup cost and bandwidth cost. In the absence of network conflicts, the communication cost of sending an n-byte message between two nodes is modeled by $\alpha+\beta n$. The path between two communicating nodes, determined by the topology and the routing algorithm, is completely occupied. Therefore, if some link in the communication path is occupied by two or more communicating pairs, a network conflict occurs. This extra cost is modeled with $\alpha + kn\beta$ where k is the maximum over all links of the number of conflicts on the links.

Bidirectional communication model: In this model, we modify the assumption for communication between two nodes as follows. We assume that there are two links between neighbouring nodes, one in each direction. At any given time, a node can send one message and simultaneously receive one message on the links connected to a neighbor. Therefore it can simultaneously be participating in $4N$ communications, consisting of $2N$ sends and $2N$ receives. All other assumptions remain the same.

2.2 Bucket Algorithms

We now describe the bucket algorithm [2, 3, 10] for the Reduce-scatter collective. In this algorithm, the nodes can be viewed as arranged in a one dimensional ring overlaid on the underlying network such that there is no contention of physical links in the embedding. Communication in this algorithm is performed along p logical chains. Each of these chains has a unique root node. Thus every node is a root for one of these chains. Each chain is of length p and extends from the root node towards the left all the way around to the node lying to the right of the root. Therefore each logical chain is the ring with one edge deleted from it – the one between the root and the node to its right. Each logical chain is responsible for n/p amount of data. These p logical chains are used to reduce the data in parallel. There are $p-1$ steps in this algorithm. Every node sends n/p data at each step. The node to the right of the root initiates the communication for that logical chain. At each step, the data is forwarded to the node on the right, which then performs the necessary reduction. At the end of $p-1$ steps the reduced data for the n/p data items is collected on the root of the



Node 0
Node 1
Node 2
$P_{10} \rightarrow$
$P_{21} \rightarrow$
$P_{02} \rightarrow$
Step 0
$P_{10} + P_{20} \rightarrow$
$P_{21} + P_{01} \rightarrow$
$P_{02} + P_{12} \rightarrow$
Step 1
$P_{10} + P_{20} + P_{00}$
$P_{21} + P_{01} + P_{11}$
$P_{02} + P_{12} + P_{22}$
Step 2

Figure 3: Reduce-Scatter on a one dimensional torus in the unidirectional model. Here P_{ij} denotes the data that initially resides on node i and is to be reduced on node j.

logical chain. The communication for all logical chains is performed in parallel. This keeps all links busy. Therefore at any step, a node is receiving data for one logical chain from the node on the left and simultaneously sending data for another logical chain to the node on the right. At the end of $p-1$ steps, each node has n/p of the reduced data. The algorithm is illustrated for a vector of length 3 on a 3 node system in Figure 3.

The cost of this approach is given by

$$T_{RS-Bkt-1D} \;=\; (p-1)\alpha \;+\; \frac{p-1}{p}n\beta \;+\; \frac{p-1}{p}n\gamma \quad (1)$$

The cost for the Allgather and Allreduce collectives can be derived similarly.

2.3 Spanning-tree Algorithms

In spanning tree algorithms, the main idea is to overlay a spanning tree (connecting all nodes) over the interconnection network. Data is then pipelined along the spanning tree in order to execute the MPI collective. In this framework, *Gather*, *Scatter*, *Reduce* and *Broadcast* form the basic collectives with other collectives derived by combining operations of these collectives. For operations such as *Scatter* and *Broadcast*, data is passed from the root towards the leaves along the spanning tree, whereas for *Gather* and *Reduce*, data is passed from the leaves towards the root along the spanning tree. It is easy to see that it is advantageous to overlay a spanning tree that is contention free, i.e., no two spanning tree edges are overlaid on the same link. Faraj et.al [1] show how to carefully overlay 6 spanning trees over a 3-dimensional torus without contention. This allows for the data to be divided into 6 parts and be pipelined along the spanning trees in parallel. Using this method, it can be shown that they theoretically achieve near-optimal bandwidth cost bounds of $n\beta/2N$ for Allgather and Reduce-scatter and $n\beta/N$ for the Allreduce collective.

We note some important differences between the spanning tree and bucket-based algorithms. (1) Spanning tree algorithms primarily pipeline the data along a tree/chain whereas bucket based algorithms divide the data along a ring. One way to visualize the bucket based algorithms is that they represent p spanning trees executing in parallel with each of the p nodes serving as the root for one of the trees. (2) Given the rooted nature of the spanning trees, the basic building blocks under this framework are the *Reduce* and *Broadcast* collectives along with the *Gather* and *Scatter* collectives. Given the divide-and-distribute nature of the bucket-based algorithms, the basic building blocks under this framework are the *Reduce-Scatter* and *Allgather* collectives along with the *Gather* and *Scatter* collectives.

3. LOWER BOUNDS

In this section we give proof sketches to derive the lower bounds under the unidirectional and bidirectional communication models as mentioned in Figure 2. The computation bounds are trivial. We only argue for the bandwidth cost bounds here.

Unidirectional model: Our proofs are based on studying the communication graphs for communication corresponding to a single element vectors. For a given communication, a *communication graph* is defined to be the graph $G = (V, E)$, where V, the set of vertices is the set of nodes $\{1, \ldots, p\}$ and there is a directed edge $(u, v) \in E$ directed from node u to v iff u sends some data to v.

We now present our lower bound for the Reduce-scatter collective.

THEOREM 3.1. *In the unidirectional communication model, the Reduce-scatter collective has a bandwidth cost of at least* $\frac{(p-1)}{p} \cdot \frac{n\beta}{N}$.

PROOF. A Reduce-scatter can be viewed as multiple Reduce operations with different roots.

Let us consider the reduce operation performed on a single element. It is easy to see that for the reduction to take place successfully, the communication graph must contain a spanning tree rooted at the node where the result is reduced, with all edges in this spanning tree directed towards the root. Hence there must be at least $p-1$ data elements that must be communicated.

Therefore, for the Reduce-scatter operation on a vector of size n, essentially n reduce operations are performed. Therefore, by the arguments above, $(p-1)n$ elements must be communicated. However, under this model, at any instance of time there can be one element being communicated on any link and there are pN links in all. Hence the communication cost must be at least $\frac{(p-1)}{p} \cdot \frac{n\beta}{N}$. $\square$

For the Allgather operation, the arguments are similar.

THEOREM 3.2. *In the unidirectional communication model, the Allgather collective has a bandwidth cost of at least* $\frac{(p-1)}{p} \cdot \frac{n\beta}{N}$.

PROOF. In this collective, every element is scattered from a source node to all other nodes. The underlying communication graph for the scattering of every element must contain a spanning tree with all edges directed away from the root. Hence, as above, $(p-1)n$ elements must be communicated, resulting in a bandwidth cost bound of $\frac{(p-1)}{p} \cdot \frac{n\beta}{N}$. $\square$

The arguments for the Allreduce collective are more involved.

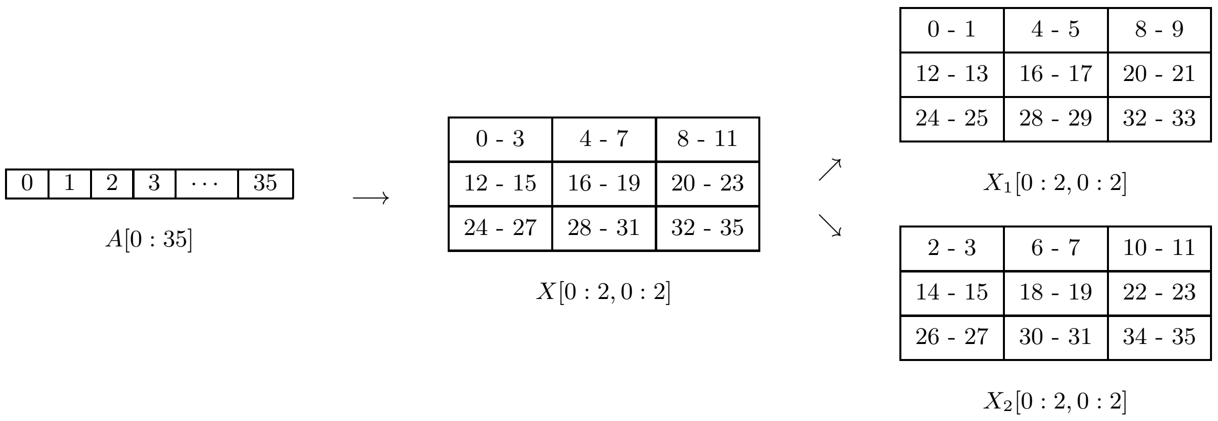


Figure 4: Node distribution-respecting division of elements

Theorem 3.3. *In the unidirectional communication model, the Allreduce collective has a bandwidth cost of at least* $2 \cdot \frac{(p-1)}{p} \cdot \frac{n\beta}{N}$.

Proof. We again consider a single element vector. We show that at least $2(p-1)$ elements must be communicated for an Allreduce of the single element vector to take place. The bound then follows as before by noting that there are only Np links over which this communication must happen.

It is always possible to obtain a complete ordering of the messages in a distributed system such that if the messages are processed in this order sequentially, we obtain the same result [7]. Let $m_1, m_2, \ldots, m_k$ be these messages in such an order. We are required to show that $k \geq 2(p-1)$. Let m_r be the first message in this ordering, on the completion of which, some node has the Allreduce result of the element vector. By following arguments similar to those provided in previous theorems, the communication up to m_r must define a spanning tree over the nodes and hence $r \geq p-1$. Note that at this point, no other node has the Allreduced result. Hence, every other node must receive at least one message thereafter in order to obtain the complete Allreduce result. Therefore $k - r \geq p - 1$. From the above two inequalities, we get that $k \geq 2(p-1)$. This proves the theorem. □

Bidirectional model: All arguments presented above simply extend for the bidirectional model by observing that the number of links in the *bidirectional communication model* are double that in the *unidirectional communication model*. These lead to the lower bounds presented in Figure 2.

Remark: We remark here that the proof arguments presented in [3] directly extend to give us the desired bounds for the bidirectional model. However, the unidirectional model requires more sophisticated arguments in comparison to the bidirectional model. While the bidirectional model is of greater practical interest, our study of the unidirectional model closes open questions regarding the optimality of bucket algorithms for torus interconnects under previously studied models.

4. MULTICOLOR BUCKET ALGORITHM FOR TORUS INTERCONNECTS

In this section we propose efficient bucket algorithms for these three collectives: Reduce-scatter, Allgather and Allreduce on torus interconnects. Throughout this section, we will explain and derive the cost models for the Reduce-scatter collective. The algorithm for the Allgather collective is the same as that for the Reduce-scatter executed in the reverse order but without performing any computations. The cost for the Allgather collective can therefore be obtained by simply removing the computation term from the cost (i.e., substituting $\gamma = 0$). As Allreduce collective can be performed by performing Reduce-scatter followed by an Allgather, it's cost can be obtained by adding the cost of these two collectives.

The Algorithm: For an N-dimensional torus, we divide the data into N (roughly equal) parts, in such a way that performing a Reduce-scatter on each of these parts results in a Reduce-scatter of the entire data. We think of these parts as being of different colors, numbered 1 to N. Let $X[i_1, i_2, \ldots, i_N]$ denote the data elements that are to be collected on node $(i_1, i_2, \ldots, i_N)$ in the N-dimensional torus on completion of the Reduce-Scatter operation.

Initially, each node has data corresponding to all elements, i.e., $\cup_{i_1,i_2,\ldots,i_N} X[i_1, i_2, \ldots, i_N] = X[0{:}d_1, 0{:}d_2, \ldots, 0{:}d_N]$. This data is carefully divided into N colors such that performing Reduce-Scatter on each of the colors and combining the results is equivalent to performing the Reduce-Scatter on the original vector. This can be accomplished by dividing each of $X[i_1, i_2, \ldots, i_N]$ into N parts. Let these N parts be denoted by $X_j[i_1, i_2, \ldots, i_N]$ for $0 \leq j < N$. Thus, the initial vector on every node corresponding to color j is $\cup_{i_1,i_2,\ldots,i_N} X_j[i_1, i_2, \ldots, i_N] = X_j[0{:}d_1, 0{:}d_2, \ldots, 0{:}d_N]$. We refer to such division of data as "node distribution-respecting." Figure 4 illustrates how node distribution-respecting data division is performed on a $d_1 \times d_2$ torus where $d_1 = 3$ and $d_2 = 3$. In this example, the data for a 36 element vector,A, is divided into 2 colors. After the Reduce-Scatter is completed, the distribution of the elements of vector A is described by the 2 dimensional matrix $X[0{:}2,0{:}2]$ (See Figure 4). The $(i, j)^{th}$ entry of X corresponds to the data that is reduced on node $d_2 \cdot i + j$. For example, the elements $0, 1, 2$ and 3 are to be reduced on node 0, the elements $4, 5, 6$ and 7 are to be reduced on node 1 and so on. The elements of this matrix are now split into two sets given by the matrices $X_1[0{:}2,0{:}2]$ and $X_2[0{:}2,0{:}2]$ (See Figure 4). According to X_1, the elements $0, 1$ are to be reduced on node 0, the elements $4, 5$ are to be reduced on node 1 and so on. Similarly, according to X_2, the elements $2, 3$ are to be reduced on

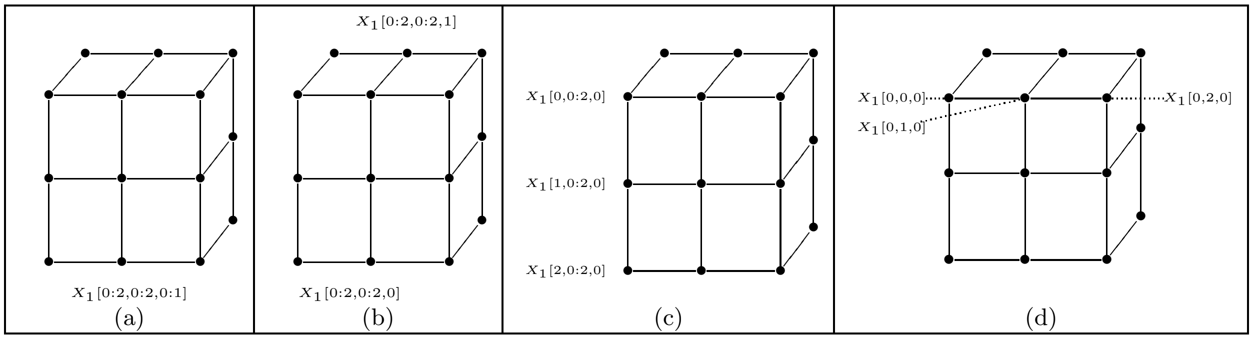


Figure 5: For a single color on a 3x3x2 Torus

node 0, the elements 6, 7 are to be reduced on node 1 and so on. Thus, the vector A is divided into two parts (colors) on every node, $A_1 = \{0, 2, \ldots, 34\}$ and $A_2 = \{1, 3, \ldots, 35\}$ corresponding to X_1 and X_2. This division of the data is "node distribution-respecting." The Reduce-scatter on these two individual vectors results in elements being reduced to the nodes as described by X_1 and X_2. Combining these resultant elements on each node is equivalent to Reduce-Scatter of the original vector.

The Reduce-Scatter for each color is performed in N phases, wherein, in each phase the bucket algorithms are run along the nodes lying along one of the dimensions. This is illustrated in Figure 5 for color=1 on a $d_1 \times d_2 \times d_3$ torus where $d_1 = 3$, $d_2 = 3$ and $d_3 = 2$. As shown in Figure 5(a), initially, every node has data corresponding to all indices, i.e., X_1[0:2,0:2,0:1]. In the first phase, Reduce-scatters are performed amongst the nodes along a third dimension (front and back) using the bucket algorithm. As shown in Figure 5(b), this results in the front face nodes having the data corresponding to the elements X_1[0:2,0:2,0] and the back face nodes having the data corresponding to the elements X_1[0:2,0:2,1]. In the second phase the bucket algorithm is run along the columns. Figure 5(c) shows the data distribution on the front face nodes after this phase. The nodes of the first, second and third row Reduce-scatter the data X_1[0:2,0:2,0] to obtain data corresponding to the elements X_1[0,0:2,0], X_1[1,0:2,0] and X_1[2,0:2,0] respectively. The data is distributed similarly for the nodes on the back face of the torus. In the third and final phase, the bucket algorithm is run along the rows. Figure 5(d) shows the data distribution on the front face first row nodes after this phase. The first, second and third nodes along the row Reduce-scatter the data X_1[0,0:2,0] to obtain data corresponding to the elements X_1[0,0,0], X_1[0,1,0] and X_1[0,2,0] respectively. The data is distributed similarly for the nodes on the back face of the torus.

Therefore for each color, we require to perform the Reduce-Scatter operation using the ring based bucket algorithm described in Section 2.2 along each of the dimensions once. Therefore, there are N phases of the algorithm, wherein, in each phase the ring based Bucket Reduce-Scatter is performed along one of the N dimensions. In N phases all N dimensions are covered. We can perform the communication for the N colors parallely in such a manner that in any phase no two colors use the links lying along the same dimension. Let $I_N =< 1, 2, \ldots, N >$ denote the sequence of the first N positive integers. Further, let $\pi_1, \pi_2, \ldots, \pi_N$ be N permutations of I_N. We call these permutations non-conflicting if $\pi_j(i) \neq \pi_k(i)$ for all $0 \leq i, j, k < N$ and $j \neq k$ where $\pi_\alpha(i)$ denotes the i^{th} element of the sequence π_α. For example it is easy to see that the permutations $\pi_1, \pi_2, \ldots, \pi_N$ defined by $\pi_j(i) = (i + j) \mod N$ for all $0 \leq i, j < N$ are non-conflicting. Given N non-conflicting permutations of I_N, we can run the N-dimensional bucket algorithm such that data of color j is Reduce-scattered along rings lying along dimension $\pi_j(i)$ in the i^{th} phase. This ensures that the communication for different colors do not contend for the same links in any given stage and that the links of all dimensions are fully utilized in a balanced manner.

In order to utilize the bidirectional links, we note that each node is connected to $2N$ bidirectional links, lying along the N dimensions. Moreover, only one direction of the links are required in order to form a ring of the nodes lying along a single dimension. Therefore a node can participate in $2N$ rings simultaneously, with pairs of these rings using the same links but in opposite directions. Therefore, when the Reduce-Scatter is performed for some color in some phase along some dimension, the data is further divided into 2 sub-parts and a Reduce-Scatter is performed for both these sub-parts along the same dimension but using different directions.

The Reduce-Scatter algorithm is described in Algorithm 1. The *for* loop in Step 3 corresponds to the N phases of the algorithm. The pseudocode is presented in a sequential manner for simplicity – in practice, all bucket algorithms within the *for* loop at step 4 would be executed simultaneously as communication across all the $2N$ links can be handled simultaneously.

For an asymmetric torus, the bandwidth cost is determined by the dimension that handles most of the data. Let the minimum and maximum dimensions be d_s and d_ℓ respectively. For the first phase, the data is divided into $2N$ equal parts. The bucket algorithm is invoked for each of these parts separately. Therefore, a Reduce-Scatter is invoked with data size $n/2N$ in this phase for every dimension. Note that the data size reduces by a factor of d_i after the Reduce-Scatter is performed along the i^{th} dimension. Hence the data size in the second phase is at most $n/(2N \cdot d_s)$. In general, after i phases the data size with which the Reduce-Scatter bucket algorithm is invoked is at most $n/(2N \cdot d_s^i)$.

Therefore, for any dimension d_i, the total bandwidth cost across the N phases is

Algorithm 1 N-dimension Torus Bucket Algorithm For ReduceScatter

```
1: Divide the data at each node logically into $N$ parts such that it is "node distribution-respecting."
2: Fix any $N$ non-conflicting permutations of $I_N$, say $\pi_1, \pi_2, \ldots, \pi_N$.
3: for $i$ = 1 to $N$ /* phases */ do
4:     for $c$ = 1 to $N$ /* colors */ do
5:         Let $q = \pi_c(i)$
6:         for each of $\frac{p}{d_q}$ rings formed by sets of $d_q$ nodes lying along the $q^{th}$ dimension do
7:             Run Bucket Algorithm for half the data of color $c$ along the ring in clockwise direction
8:             Run Bucket Algorithm for the other half of the data in anti-clockwise direction
9:         end for
10:    end for
11: end for
```

$$\begin{aligned} T_{bandwidth} &\leq \frac{d_\ell - 1}{d_\ell} \cdot \frac{n\beta}{2N} \cdot \left(1 + \sum_{k=1}^{N-1} \frac{1}{d_s^k}\right) \\ &= \frac{d_\ell - 1}{d_\ell} \cdot \frac{n\beta}{2N} \cdot \frac{d_s^N - 1}{d_s^N} \cdot \frac{d_s}{d_s - 1} \end{aligned}$$

We make a few observations regarding this bandwidth cost bound:

- Since $d_s \geq 2$, $T_{bandwidth} \leq 2 \cdot \frac{p-1}{p} \cdot \frac{n\beta}{2N}$ implying our algorithm is asymptotically tight.
- As $d_s \to \infty$, so do d_ℓ and p. We observe that $T_{bandwidth} \to \frac{n\beta}{2N}$ and the lower bound $\frac{p-1}{p} \cdot \frac{n\beta}{2N} \to \frac{n\beta}{2N}$ as well. Therefore, our algorithm cost converges to the lower bound.

The bound for the Reduce-scatter collective is therefore:

$$\begin{aligned} T_{RS-BiDir-Asymm} &= 2N\sum_{i=1}^{N}(d_i - 1)\alpha + \frac{p-1}{p}n\gamma + \\ &\quad \frac{d_\ell - 1}{d_\ell} \cdot \frac{n\beta}{2N} \cdot \frac{d_s^N - 1}{d_s^N} \cdot \frac{d_s}{d_s - 1} \quad (2) \end{aligned}$$

If we substitute $d_s = d_\ell = p^{1/N}$ in $T_{bandwidth}$, we obtain the cost for a symmetric torus of size $d \times \cdots \times d$ torus, where $d = p^{1/N}$

$$\begin{aligned} T_{RS-BiDir-Symm} &= 2N^2(d-1)\alpha + \frac{p-1}{p}n\gamma + \\ &\quad \frac{p-1}{p} \cdot \frac{n\beta}{2N} \quad (3) \end{aligned}$$

For the unidirectional case, we similarly obtain the following bounds:

$$\begin{aligned} T_{RS-UniDir-Asymm} &= N\sum_{i=1}^{N}(d_i - 1)\alpha + \frac{p-1}{p}n\gamma + \\ &\quad \frac{d_\ell - 1}{d_\ell} \cdot \frac{n\beta}{N} \cdot \frac{d_s^N - 1}{d_s^N} \cdot \frac{d_s}{d_s - 1} \quad (4) \\ T_{RS-Unidir-Symm} &= N^2(d-1)\alpha + \frac{p-1}{p}n\gamma + \\ &\quad \frac{p-1}{p} \cdot \frac{n\beta}{N} \quad (5) \end{aligned}$$

The bounds for Allgather and Allreduce can be obtained similarly.

Overlapping computation and communication: For collectives involving computation such as Reduce-Scatter and Allreduce, the amount of data being processed is proportional to the amount of data being communicated. This presents an excellent opportunity for overlapping computation with communication. In order to achieve this, we divide the bucket data for a color into smaller parts and transmit these parts in a pipelined manner. This allows the receiving node to perform computations on the part that has been received while the next part is being received. The number of parts has to be carefully chosen as additional startup costs are incurred for every additional part. The maximum of the bandwidth and computation bounds is a trivial lower bound (not tight) for this case.

Parallelization with multi-cores: On systems with multi-core nodes, the startup cost and processing costs can be divided over the cores. There are several ways of dividing this work.

One way is to divide the work on the basis of color since there are no data dependencies amongst the data for different colors. Using N cores, we can handle each color on a separate core. In Algorithm 1, this corresponds to the iterations of Step 4 being executed on separate cores as they are independent. We can further use $2N$ cores, where each core handles one of the directions for a fixed color. In Algorithm 1, this corresponds to Steps 7 & 8 also being executed in parallel on separate cores. Note that the efficiency of such a division of work crucially depends on the distribution of the computations with respect to the colors on the nodes. For instance, for the spanning tree algorithms [1], even though the total computation performed on every node is almost the same, the distribution of the computations is not uniform for the different colors. It can be seen that there are nodes that have to perform 3 computations for one color and none for some other color (these are nodes that essentially have 3 children in one spanning tree and are leaves in another). This can limit the perormance gains with color bsed division of work. Our N-dimensional bucket algorithm in contrast leads to uniform distribution of workload across the nodes as well as across the cores for the different colors.

There are several other approaches for parallelizing the computation using multicores that can be applied to the bucket algorithms as well as the spanning tree algorithms. However, there are certain issues in these approaches that make them unsuitable in comparison to the color based division of work. One approach is to perform computations on separate cores once the data is received on some core. However, this involves synchronization overheads for the cores to communicate regarding availability of data/results for computations. As algorithms typically work in pipeline mode where data is broken down into smaller packets, these overheads can be pretty substantial for small packet sizes. Another approach is to break down the data into smaller parts and have different cores work on the different parts. This is like having more colors. However, on using more than $2N$ colors, multiple cores/colors have to share the same links for communication. Although the bandwidth requirement is still the same, contention of link usage amongst the cores can lead to degradation in performance. For example, if two packets of one color are passed over the link before the packet of another color, this can lead to idling of the core handling the other color at the destination node.

5. EXPERIMENTAL RESULTS

In this Section, we demonstrate the improvements obtained using our algorithm when implemented on the Blue Gene/P, which has a 3-dimensional torus interconnect.

5.1 Blue Gene/P Overview

The Blue Gene/P [5] is IBM's next-generation massively-parallel supercomputer, which has evolved from the Blue Gene/L architecture. Each node in a Blue Gene/P system consists of four 850 MHz PowerPC 450 processor cores. The nodes are interconnected through five networks. The most important one of these is the torus network, which handles the bulk of the communication data from an application and offers the highest bandwidth in the system. Each node supports 850 MBps bidirectional links to each of its nearest neighbors for a total of 5.1GB/s bidirectional bandwidth per node. Direct memory access (DMA) based communication operations are provided in order to minimize message handling overhead on the processing cores during send and receive operations. The messaging is based on variable size packets, which are multiples of 32 bytes in size with a maximum packet size of 256 bytes. The messages carry a hardware header and software header of 8 bytes each, resulting in a total header of 16 bytes. Thus there can be a maximum of 240 bytes of payload per packet. There is an additional 14 byte overhead per packet due to acknowledgments.

5.2 MPI Collectives performance

Experimental Setup: We implemented our multi-color bucket algorithms for the collectives on the Blue Gene/P Supercomputer using the lower level communication APIs. These algorithms incorporate overlapping of computation and communication. Two versions of our algorithms have been implemented – single-core and multi-core. In the single-core version, only one core is used on each node. In the multi-core version, we used 3 cores on each node, with each core handling the data along both directions for one color. We also compared the performance of our algorithms with the latest IBM optimized MPI library, which incorporates the optimized spanning tree algorithms [1]. We note here that these algorithms are single-core and do not use the lower level interface APIs[1]. The MPI results presented use default options and may be marginally improved using specialized environment variables and compiler options. We conducted the experiments for different configurations – system sizes and data sizes. The performance results were obtained by taking the mean of the performance of ten runs for each of these configurations. We also determined the lower-bounds for each configuration. Since our optimizations overlap computation with communication, this lower bound is obtained by considering only the bandwidth bottleneck (see Figure 2) in the bidirectional communication model.

[1]It is non trivial in system's software to take advantage of multi-core as communication threads need to be started via inter-process-interrupts to enable those cores to accelerate communication. Hence, the MPI stack just uses a single core. While in an application library, its possible for the application to give control of the threads to the library to enable multicore optimizations

Results : The performance of the experiments is reported in Figure 6 using log-log plots. The performance is reported in units of time taken to complete the collective operation.

In Figures 6 (a)-(c), we report the performance for the Reduce-scatter, Allgather and Allreduce collectives on 4096 Blue Gene/P nodes for varying data sizes. For Reduce-scatter (Figure 6 (a)), our multi-core algorithm is consistently within 30% of the bandwidth cost bound for most data sizes. The single-core algorithm, on the other hand, is computation bound. In comparison to the MPI library implementation, the multi-core algorithm is consistently better by a factor of 12 to 17 and the single-core is better by a factor of 6. For Allgather (Figure 6 (b)), the performance of our multi-core algorithms is just within 7% of the lower bound. The performance of the single-core algorithms is also very similar to the multi-core algorithms since this collective does not involve any computations and is therefore bandwidth cost bound even on a single core. Our algorithm are consistently better by a factor of 3 in comparison to the MPI library implementation. For Allreduce (Figure 6 (c)), our multi-core algorithm is consistently within 20-30% of the bandwidth cost bound. The single-core algorithm is again computation-bound. In comparison to the MPI library implementation, the multi-core algorithm is consistently better by a factor of 3 and the single-core is better by a factor of 1.5.

Some of the gains observed, in comparison of our single-core version with the MPI library, can be attributed to our use of lower level APIs that avoids MPI stack overheads. From the Allreduce results, it can be estimated that this results in factor of 1.5 gain in performance. For the other collectives, this performance differs by more than a factor of 1.5 – this may possibly be attributed to the specifics of the MPI library implementation.

The gains observed in comparison of our single-core and multi-core versions can be attributed to the use of multi-cores. However, it can be seen these gains are not proportional to the number of cores that are used. This is because the multi-core version is communication bound whereas the single-core version is computation bound. In order to study the multi-core scalabilty of our algorithms, we performed another experiment in which we introduced an artificial delay so as to make even the multi-core version computation bound. As shown in Figure 6 (d), in this experiment, the multi-core version performs a factor of 3 better than the single-core version for most data size, which is commensurate with the number of cores used. This indicates that the computation load is uniform across the cores showing that the computation load on every node is same for every color, as discussed in Section 4. This also demonstrates the multi-core scalability of the multi-color bucket algorithms obtained by color-based division of work. Figure 6 (e) & (f) reports the scaling performance of our Allgather and Allreduce algorithms for different node count. The vector size is taken to be $32 \cdot p$ MB, where p is the number of nodes. These results indicate that our algorithms scale very well. The consistent proximity to the lower bound implies that not much better performance can be achieved.

6. CONCLUSIONS

We proposed new bucket algorithms for the Reduce-scatter, Allgather and Allreduce collectives for large vectors on torus interconnects. Our algorithms are tight for the symmetric torus and asymptotically optimal for asymmetric torus. We also tightened existing lower bounds under the unidirectional comunication model and showed that our algorithms are optimal under the unidirectional as well. There are several directions for future work. It would be interesting to perform a theoretical study of the algorithms and lower bounds for other collectives under these models. Another important open problem is to develop algorithms that match the theoretical lower bounds or to improve the lower bounds for asymmetric torus networks.

7. ACKNOWLEDGMENTS

We would like to thank Venkatesan T. Chakaravarthy for useful discussions related to the lower bounds that lead to simpler proofs. We would like to thank Rahul Garg and Sameer Kumar for useful discussions related to MPI and the paper in general. Finally, we would like to thank James Sexton, James Stasak, Pascal Vezolle and Guy Robinson for discussions related to applications that use these collectives.

8. REFERENCES

[1] A. Faraj, S. Kumar, B. Smith, A. Mamidala, J. Gunnels, and P. Heidelberger. MPI collective communications on the Blue Gene/P Supercomputer: algorithms and optimizations. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, pages 489–490, New York, NY, USA, 2009. ACM.

[2] B. L. Payne, M. Barnett, R. Littlefield, D. G. Payne, and R. V. D. Geijn. Global combine on mesh architectures with wormhole routing. In *Proc. of 7 th Int. Parallel Proc. Symp*, 1993.

[3] E. Chan, M. Heimlich, A. Purkayastha, and R. A. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[4] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur. Collective communication on architectures that support simultaneous communication over multiple links. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 2–11, New York, NY, USA, 2006. ACM.

[5] I. journal of Research and D. staff. Overview of the IBM Blue Gene/P project. *IBM J. Res. Dev.*, 52(1/2):199–220, 2008.

[6] J. Watts, R. Van, and D. Geijn. A pipelined broadcast for multidimensional meshes. Parallel Processing Letters, 1995.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[8] M. Barnett, D. G. Payne, R. A. van de Geijn, and J. Watts. Broadcasting on meshes with wormhole routing. *J. Parallel Distrib. Comput.*, 35(2):111–122, 1996.

[9] M. Barnett, R. J. Littlefield, D. G. Payne, and R. A. van de Geijn. Global combine algorithms for $2-d$ meshes with wormhole routing. *jpdc*, 24(2):191–201, 1995.

[10] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. Geijn, and J. Watts. Interprocessor collective communication library (intercom). In *Proceedings of the Scalable High Performance Computing Conference*, pages 357–364. IEEE Computer Society Press, 1994.

[11] P. Mitra, D. Payne, L. Shuler, R. van de Geijn, and J. Watts. Fast collective communication libraries, please. Technical report, Austin, TX, USA, 1995.

[12] R. A. van de Geijn. On global combine operations. *J. Parallel Distrib. Comput.*, 22(2):324–328, 1994.

[13] R. Rabenseifner. Automatic MPI counter profiling of all users: First results on a Cray T3E 900-512. Message Passing Interface DeveloperŠs and UserŠs Conference 1999 (MPIDC Š99), 1999.

[14] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *toc*, 38(9):1249–1268, 1989.

[15] Y. Saad and M. H. Schultz. Data communication in hypercubes. *J. Parallel Distrib. Comput.*, 6(1):115–135, 1989.

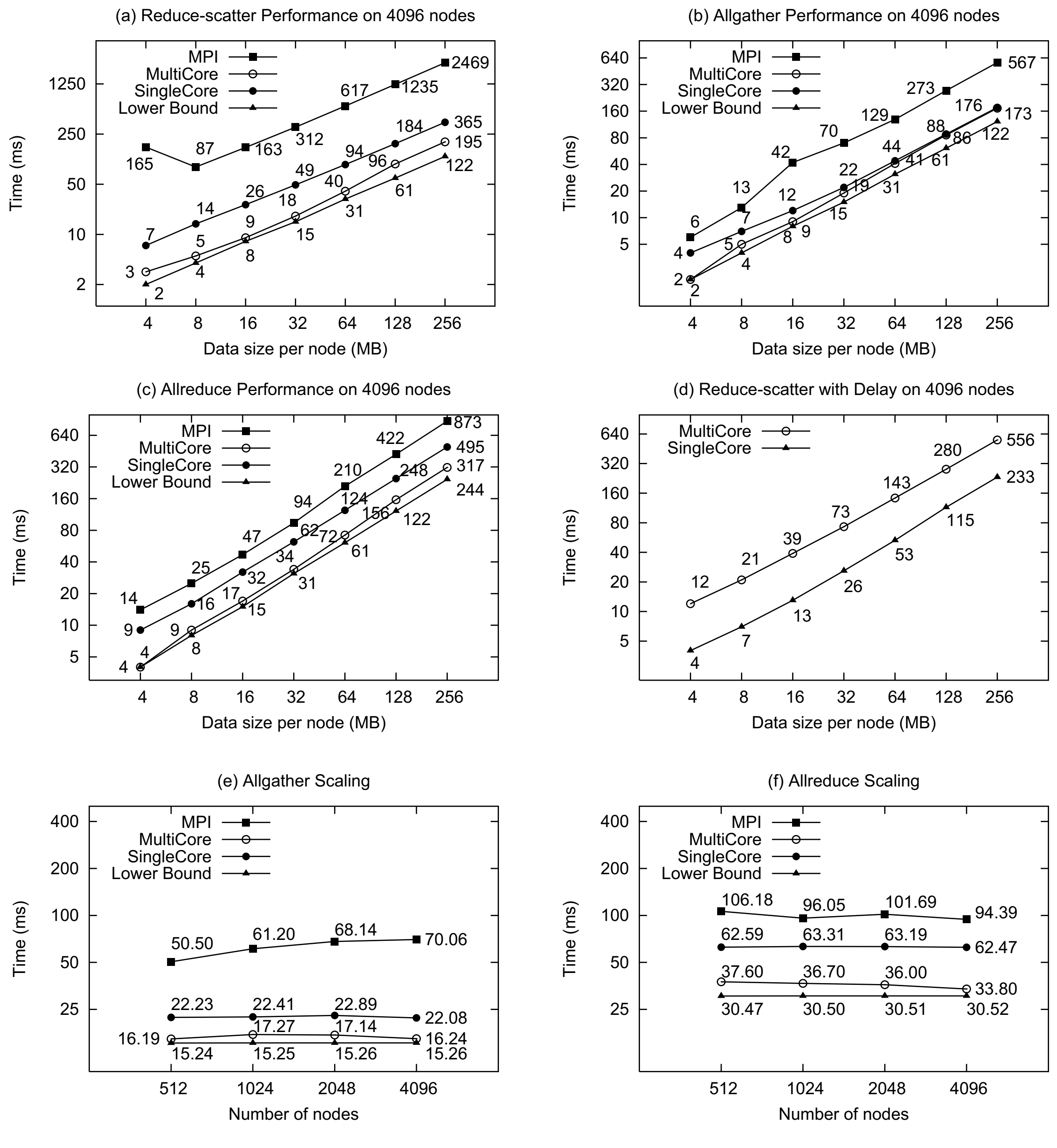


Figure 6: MPI Collectives Performance