Full-Duplex Inter-Group All-to-All Broadcast Algorithms with Optimal Bandwidth

Qiao Kang
EECS Department, Northwestern University
qiao.kang@eecs.northwestern.edu

Jesper Larsson Träff
Faculty of Informatics, Vienna University of Technology
traff@par.tuwien.ac.at

Reda Al-Bahrani
EECS Department, Northwestern University
rav650@eecs.northwestern.edu

Ankit Agrawal
EECS Department, Northwestern University
ankitag@eecs.northwestern.edu

Alok Choudhary
EECS Department, Northwestern University
choudhar@eecs.northwestern.edu

Wei-keng Liao
EECS Department, Northwestern University
wkliao@eecs.northwestern.edu

ABSTRACT

MPI inter-group collective communication patterns can be viewed as bipartite graphs that divide processes into two disjoint groups in which messages are transferred between but not within the groups. Such communication patterns can serve as basic operations for scientific application workflows. In this paper, we present parallel algorithms for inter-group all-to-all broadcast (Allgather) communication with optimal bandwidth for any message size and process number under single-port communication constraints. We implement the algorithms using MPI point-to-point and intra-group collective communication functions and evaluate their performance on the Cori supercomputer at NERSC. Using message sizes ranging from 256B to 64MB, the experiments show a significant performance improvement achieved by our algorithm, which is up to 9.27 times faster than production MPI libraries that adopt the so called root-gathering algorithm.

KEYWORDS

Inter-group communication; All-to-all broadcast, Allgather

ACM Reference Format:
Qiao Kang, Jesper Larsson Träff, Reda Al-Bahrani, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao. 2018. Full-Duplex Inter-Group All-to-All Broadcast Algorithms with Optimal Bandwidth. In *25th European MPI Users' Group Meeting (EuroMPI '18), September 23–26, 2018, Barcelona, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3236367.3236374

1 INTRODUCTION

The MPI [12] inter-group collective communication patterns can be viewed as bipartite graphs that divide processes into two disjoint groups in which messages are transferred between but not within the groups. Different from the half-duplex case, where one group is the sender group and the other group is the receiver group, full-duplex inter-group collective communication means that both groups send and receive messages from each other. Since processes in one group only need to receive messages from the other group, these communication patterns can be flexibly used by many applications.

Scientific application workflows utilize inter-group collective communication. Such workflow systems divide components into groups. Processes in two different components exchange computed results via all-to-all broadcast (Allgather) and personalized all-to-all scatter (all-to-all). In [11], the authors propose a framework for parallel data transfer among workflow components on high-end supercomputers for a computationally intensive weather prediction system, SCALE-LETKF [7]. Hardware/Hybrid Accelerated Cosmology Code (HACC) [6] is a system that involves real-time communication and processing of Peta-byte size of data among components [14]. To improve system performance, researchers have proposed strategies for reducing inter-group communication cost among workflow components. For example, Zhang et al. [20] propose a distributed framework that maximizes on-chip data exchange that can reduce the communication cost caused by communication among job components. Docan et al. [4] design an Active Spacing approach that reduces data exchange by moving programs to staging areas. Although reducing data communication frequency and size can improve overall performance, inter-group communication among components is still unavoidable. Given such applications scenarios, it makes sense to investigate and to improve the inter-group collective communication operations provided by MPI.

In this paper, we focus on an important inter-group collective communication pattern, namely all-to-all broadcast. This function corresponds to MPI_Allgather with an inter-group communicator in the MPI standard. MPI_Allgather with an inter-group communicator is defined such that every process in each of the two disjoint groups of processes receives messages from all processes in the other group. MPICH [1] and OpenMPI [5], the most widely-used MPI implementations in the parallel processing community, adopt the "root

EuroMPI '18, September 23–26, 2018, Barcelona, Spain

ACM ISBN 978-1-4503-6492-8/18/09…$15.00
https://doi.org/10.1145/3236367.3236374

gathering" algorithm. The root gathering algorithm is discussed in [15] and [16]. The underlying principle of the root gathering algorithm is single-process accumulation per group followed by a pair-wise exchange of the gathered messages. Then, one-to-all broadcasts within the two groups achieve the objective of inter-group Allgather. Although the root gathering algorithm is easy to implement, it clearly does not achieve optimal bandwidth term lower bound under a single-port communication constraint.

We propose an optimal inter-group all-to-all communication algorithm in terms of bandwidth for any number of processes and any message size. We provide a detailed theoretical analysis to justify that the algorithm achieves the optimal bandwidth lower bound. We show that the optimal bandwidth can be achieved with a linear number of process connections. Unlike the root gathering algorithm, the proposed algorithm avoids the one-to-all broadcasts of large message size.

We implement the proposed algorithms for inter-group all-to-all broadcast using MPI point-to-point and intra-group collective communication functions. Experiments are conducted on Cori, a Cray XC40 supercomputer at the National Energy Research Scientific Computing Center (NERSC). Direct comparisons with the MPI native library function MPI_Allgather that exploits the root gathering algorithm are presented. Although the communication network topology on Cori is pseudo fully connected (dragon-fly), we prove the point by evaluating and comparing our algorithms against the MPI library installed on Cori. Using message sizes ranging from 64B to 128MB, the experiments show a significant performance improvement achieved by our algorithm, which is up to 9.27 times faster than the production MPI libraries.

2 BACKGROUND AND RELATED WORKS

The communication model used in this paper is based on the assumptions presented in [10], which are summarized below.

(1) **Parallel architecture**: An undirected and connected graph is used to represent a network, where processes are vertices and links are edges. All pair-wise processes are directly connected by a link.
(2) **Basic communication operations**: Message send and receive are the basic operations.
(3) **Bidirectional single-port communication constraint**: When a send or receive function at a process is called, this function is blocked until it returns. A process can receive and send a message at the same time.
(4) **Communication cost**: Let β be the transition time per word and α be communication startup latency. If there is a link between sender s and a receiver r, sending a message of size k words from s to r has communication transmission cost $\alpha + k\beta$. We refer the α term as the *latency term* and $k\beta$ as the *bandwidth term*.

2.1 Collective communication

Collective communications such as Allgather, all-to-all, and one-to-all broadcast in MPI can be divided into intra-group communication and inter-group communication. For example, MPICH [1] and OpenMPI [5] wrap collective communication functions by conditioning on whether the input communicator is an inter-group or an intra-group communicator.

Existing literature has thoroughly studied intra-group collective communication algorithms [3]. Johnson et al. [9] have implemented algorithms for all-to-all broadcast and total exchange under single-port communication constraint on hypercube topology. Bruck algorithm [2] gives similar results for fully connected topology with any number of processes. Thakur et al. [17] optimized intra-group Allgather using recursive doubling and the Bruck algorithm [2] for non-power number of processes. Träff [19] has proposed an algorithm for intra-group Allgatherv that can handle irregular data pattern on ring topology. There are numerous other results, also taking the hierarchy of modern systems into account.

Inter-group collective communication algorithm, on the other hand, needs further research. For inter-group all-to-all broadcast (MPI_Allgather), MPICH [1] and OpenMPI [5], the most widely-used MPI implementations in the HPC community, adopt the root gathering algorithm. The root gathering algorithm is discussed in [15] and [16]. We present details of the root gathering algorithm later in this section.

2.2 Problem Definition

We first introduce the mathematical notation used in this paper. Let $A = \{a_0, ..., a_{p-1}\}$ and $B = \{b_0, .., b_{q-1}\}$ be two disjoint groups of processes. Group A has size p and group B has size q. Initially, every $a_i \in A$ has a unique message $m_{A,i}$ of size k_A words and every $b_i \in B$ has a unique message $m_{B,i}$ of size k_B words. We denote $m_A = \{m_{A,i} | \forall 0 \leq i < p\}$ and $m_B = \{m_{B,i} | \forall 0 \leq i < q\}$. The goal of the all-to-all broadcast operation is to let every b_j $\forall 0 \leq j < q$ receive $m_A = m_{A,i}$ $\forall 0 \leq i \leq p-1$ and every a_j $\forall 0 \leq j < p$ receive $m_B = m_{B,i}$ $\forall 0 \leq i \leq q-1$.

2.3 Optimal Communication Cost

Under the single-port communication constraint, there is a largest lower bound on bandwidth for inter-group full-duplex Allgather algorithms. Given optimal bandwidth, there are lower bounds for the startup latency as well.

Consider a single node $a \in A$, which has to receive all messages $m_{B,0}, .., m_{B,(q-1)}$ in the end. By the single-port communication constraint, it receives messages with rate β. Suppose there is no idle time for the receiving channel, a has to receive qk_B words. Hence the lower bound on message bandwidth is $qk_B\beta$. The same argument can be applied to $b \in B$. The lower bound on the message bandwidth is $pk_A\beta$. Let OPT be the achievable minimum bandwidth for inter-group Allgather. OPT $\geq \max(qk_B, pk_A)\,\beta$ is an overall lower bound of bandwidth for any algorithm that performs inter-group all-to-all broadcast. We prove that this lower bound is the largest lower bound on the bandwidth by presenting a feasible algorithm that can achieve it.

Given optimal bandwidth term, it is possible to derive a lower bound for the latency term. For arbitrary $a_i \in A$,

if a_i receives c messages of the form $m_{A,j}$ for $0 \leq j < p$, the bandwidth is at least $(ck_A + qk_B)\beta > qk_B\beta$. Hence optimal bandwidth is not achieved if $k_Bq > k_Ap$. By single-port communication constraint, at most q messages can be sent across two groups in parallel. Therefore, $\frac{p}{q}\alpha$ is a lower bound for the startup latency when $k_Bq > k_Ap$, since p separate messages are transferred from group A to group B without aggregation. In addition, another lower bound for the startup latency for inter-group Allgather is the startup latency $\log(p+1)\alpha$ for one-to-all broadcast of $p+1$ processes, which is a sub-problem of inter-group all-to-all broadcast. Therefore, $\log(p)\alpha$ is also a lower bound for startup latency.

Simply applying intra-group Allgather does not achieve optimal communication cost. For example, an intra-group Allgather for $p+q$ processes with $\max(k_A, k_B)$ message size has bandwidth $(p+q-1)\max(k_A, k_B)\beta$, which is more than the optimal bandwidth lower bound.

2.4 Root Gathering Algorithm

The root gathering algorithm, adopted by both MPICH and OpenMPI, is discussed in [15] and [16]. The underlying principle is single-process accumulation per group followed by inter-group one-to-all broadcasts from the roots to the remote groups, achieving the objective of inter-group Allgather. The root gathering algorithm can be implemented using existing intra-collectives as building blocks conveniently, and requires no new communicators to be created which could be expensive. However, it does not achieve optimal communication cost lower bound under single-port communication constraint.

The root gathering algorithm has three stages. In the first stage, root processes a_0 and b_0 gather all messages from other processes within their groups with communication cost $\max(\log(p)\alpha + (p-1)k_A\beta, \log(q)\alpha + (q-1)k_B\beta)$. In the second stage, the root a_0 in group A broadcasts aggregated message m_A of size pk_A to all q processes in group B. In the third stage, the root b_0 in group B broadcasts aggregated message m_B of size qk_B to all p processes in group A. The implementations of one-to-all broadcast in MPICH and OpenMPI have communication cost $2\log(p)\alpha + 3pk\beta$ for message size k and p processes. In their implementations of the inter-group one-to-all broadcast, the root process of the local group sends messages to the root of the remote group. Then, the remote group performs intra-group one-to-all broadcast. In addition, stage 2 and stage 3 are executed sequentially. The inter-group broadcast from group A to group B with message size pk_A has communication cost $(2\log(p)+1)\alpha + 3pk_A\beta$. The inter-group broadcast from group B to group A with message size qk_B has communication cost $(2\log(q)+1)\alpha + 3qk_B\beta$. As a result, the root gathering algorithm implemented by MPICH and OpenMPI has communication cost at least $(2\log(p) + 3\log(q))\alpha + (\max(pk_A, qk_B) + 3qk_B + 3pk_A)\beta$ for large p and q. This communication cost is used as a benchmark for evaluation.

Recent studies for one-to-all broadcast can improve the performance of the root gathering algorithm. Sanders et al. [13] and Träff [18] have proposed algorithms for one-to-all broadcast with communication cost close to $\log(p)\alpha + k\beta$ for message size k and p processes. Thus, we assume broadcasting pk_A messages to q processes has theoretical communication cost $\log(q+1)\alpha + pk_A\beta$. Furthermore, if Stage 2 and Stage 3 can be performed partially in parallel, for example, an exchange of messages at roots followed by intra-group one-to-all broadcasts, the total communication cost is the sum of the root gathers in the first stage, the exchange of messages accumulated at the roots, and the intra-group one-to-all broadcasts. Thus, the total bandwidth of the root gathering algorithm can be $3\max(pk_A, qk_B)\beta$ and the total startup latency can be $2\log(q)\alpha$ for large p and q. To our best knowledge, this idea has not yet been implemented. Nevertheless, it still does not achieve optimal bandwidth.

3 DESIGN

We first present an algorithm that can achieve optimal bandwidth term for the case when q divides p for a certain condition. Then, we generalize the algorithm to handle all cases. The algorithm is shown as Algorithm 1. Figure 1 gives an example to illustrate the algorithm when $p = 6$ and $q = 2$. The six processes in group A are labeled in $a_0, a_1, \cdots a_5$ in circles. In group B, the two processes are labeled b_0 and b_1. Messages ready to be transferred at any given time step are labeled next to the process ranks. The arrows indicate the directions of message transfer. The messages being transferred are labeled next to the arrows.

THEOREM 3.1. *The bandwidth (b.w) term of Algorithm 1 is* $\left(\max(k_A, k_B)\frac{p}{q} + \max\left(\frac{p}{q}k_A, k_B\right)(q-1)\right)\beta$.

PROOF. The algorithm consists of two steps. Lines 2-8 correspond to the first step, which is depicted by subfigures (a), (b), and (c) of Figure 1. Processes in group A are divided into $\frac{p}{q}$ subgroups based on their ranks. Process i in subgroup 0 exchanges its message with process $i \bmod q$ in group B, followed by subgroup 1 and so on. This step completes in $\frac{p}{q}$ message exchanges. Communication cost of the first step is $\frac{p}{q}\alpha + \frac{p}{q}\max(k_A, k_B)\beta$. Lines 11-15 correspond to the second step, two concurrent intra-group Allgather operations running within group A and group B independently. Lines 11-13 is the concurrent intra-group Allgather of message size k_B over q processes in each subgroup. It has communication cost of $\log(q)\alpha + k_B(q-1)\beta$. Line 15 is an intra-group Allgather of message size $k_A\frac{p}{q}$ within group B and has communication cost of $\log(q)\alpha + k_A\left(p - \frac{p}{q}\right)\beta$. The communication cost of the second step is $\log(q)\alpha +$

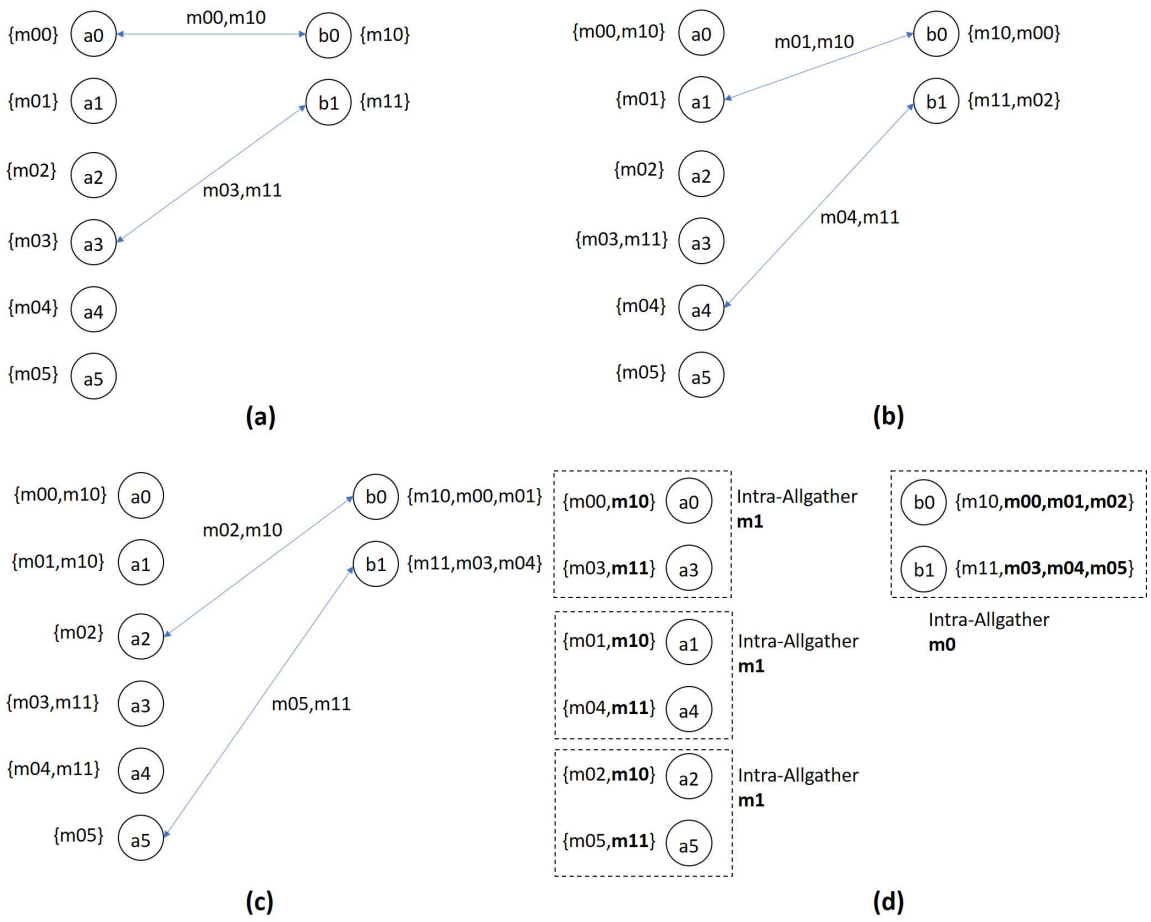

Figure 1: An execution of Algorithm 1 when $p = 6$ and $q = 2$. Messages are exchanged between two groups. Then, intra-group Allgather operations with process size q are run in parallel.

$$\max\left(k_B\left(q-1\right), k_A\left(p-\frac{p}{q}\right)\right)\beta.$$

$$\begin{aligned} b.w &= \left(\max\left(k_A, k_B\right)\frac{p}{q} + \max\left(\left(p-\frac{p}{q}\right)k_A, \left(q-1\right)k_B\right)\right)\beta \\ &= \left(\max\left(k_A, k_B\right)\frac{p}{q} + \max\left(\frac{p}{q}\left(q-1\right)k_A, \left(q-1\right)k_B\right)\right)\beta \\ &= \left(\max\left(k_A, k_B\right)\frac{p}{q} + \max\left(\frac{p}{q}k_A, k_B\right)\left(q-1\right)\right)\beta \end{aligned}$$

□

When $\frac{p}{q}k_A > k_B$, the overall bandwidth is dominated by the communication cost of group B receiving all messages. Hence the bandwidth is optimal. Otherwise the bandwidth is $\left(k_B\frac{p}{q} + k_B\left(q-1\right)\right)\beta$, so the bandwidth term does not converge to k_Bq as $\frac{p}{q} \to \infty$, which implies that the algorithm is not optimal.

We can further analyze the worst case bandwidth of Algorithm 1 compared with the optimal bandwidth. Suppose we have $k_Ap < k_Bq$. The bandwidth of Algorithm 1 is $\left(k_B\frac{p}{q} + k_B\left(q-1\right)\right)\beta$. The optimal bandwidth is $k_Bq\beta$. If k_B and p are arbitrarily large values while q and k_A are relatively small numbers with constraint $k_Ap < k_Bq$, the term $k_B\frac{p}{q}$ is much larger than qk_B. For example, let $q = 1$ and $k_A = 1$. Let p and k_B both be very large integers satisfying $p < k_B$. b_0 would send its message $m_{B,0}$ of size k_B to p processes sequentially. However, the receiving channels of $p-1$ processes in group A are idle at every step. Therefore, the worst case bandwidth term of Algorithm 1 is not bounded by any constant multiple of OPT. We refer to scenarios where $k_Ap < k_Bq$ as *message hazards*.

3.1 Full-duplex Communication with Message Fragmentation

We propose a message fragmentation method for messages transferred from group B to group A. We divide messages of size k_B into min $\left(k_B, \frac{p}{q}\right)$ segment blocks, so they can be transferred via different routes. The min function is used to handle the case when $k_B < \frac{p}{q}$, assuming the minimum message size to be transferred is a word size. Once a process has

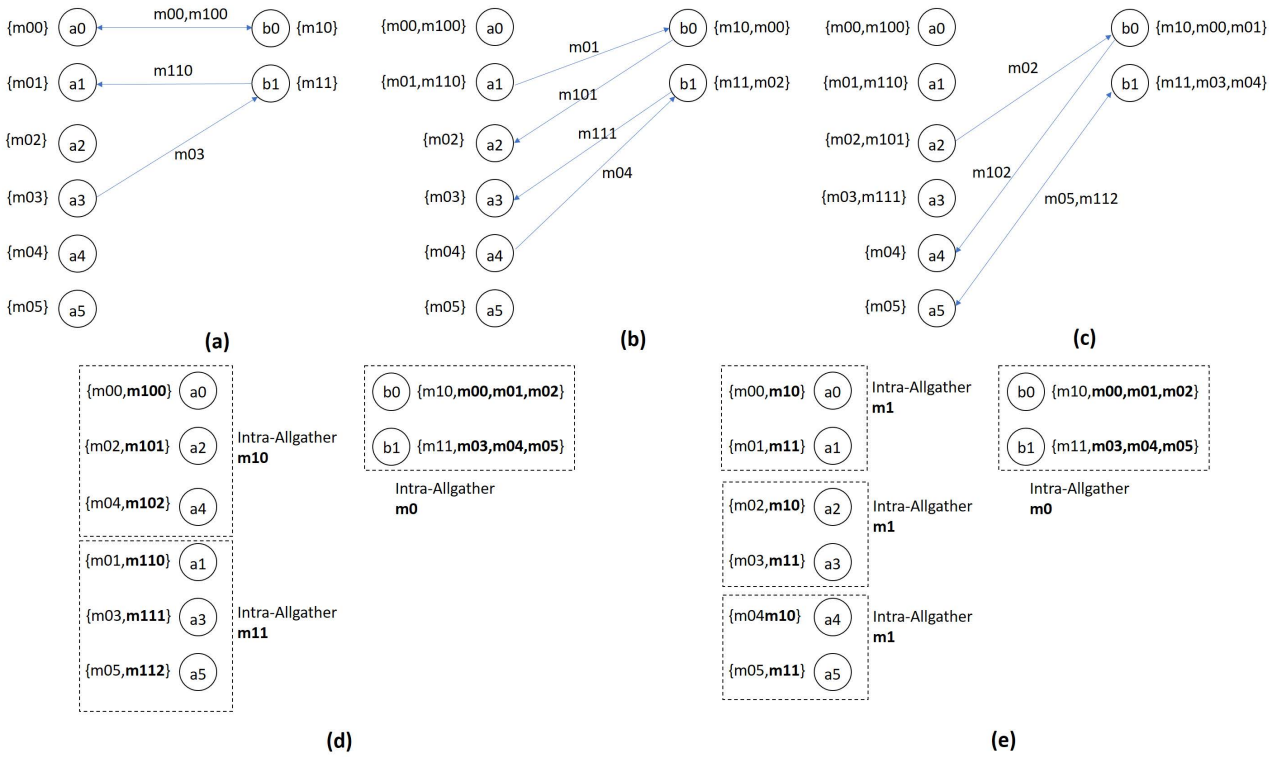

Figure 2: An execution of Algorithm 2 when $p = 6$ and $q = 2$.

Algorithm 1: Full-duplex Inter-group All-to-All Broadcast

```
# Exchange of messages between two groups in p/q steps
for i ∈ (0, .., p/q − 1) do
    # Concurrent loop j
    for j ∈ (0, .., q − 1) do
        a_{i+(p/q)j} send m_{A,i+(p/q)j} to b_j
        b_j send m_{B,j} to a_{i+(p/q)j}
    end
end
# Subgroups of processes in group A perform an
  intra-group Allgather for m_B
# Concurrent loop i
for i ∈ (0, .., p/q − 1) do
    Intra-Allgather a_{(p/q)j+i}, m_{B,j} ∀(0 ≤ j < q)
end
# Group B performs intra-group Allgather for m_A
Intra-Allgather b_j, m_{A,(p/q)j+i} ∀(0 ≤ j < q, 0 ≤ i < p/q)
```

received all segments, it can reconstruct them back into the original message. We use $m_{B,j,i}$, where $0 \leq i < \min\left(k_B, \frac{p}{q}\right)$, to denote the $(i+1)^{th}$ piece of the fragmented message j. Algorithm 2 is the data fragmentation version of the full-duplex inter-group all-to-all broadcast.

The improvement of Algorithm 2 over Algorithm 1 is that messages are transferred across groups much faster. In Lines 2-9 of Algorithm 2, the fragmented m_B is received by group A with bandwidth $k_B\beta$, which does not depend on p and q. Lines 2-9 of Algorithm 1, on the other hand, have bandwidth $\frac{p}{q}k_B\beta$ for all m_B to be received by group A.

In Algorithm 2, Lines 2-9 initiate $\frac{p}{q}$ operations, so the startup latency is $\frac{p}{q}\alpha$. Lines 12-20 and 22 run in parallel. For Lines 12-20, the number of steps is at most $\log\left(\frac{p}{q}\right) + \log(q) = \log(p)$. For Line 22, the number of steps is $\log(q)$. Therefore, the overall startup latency for Lines 12-22 is $\log(p)\,\alpha$. We have shown that $\frac{p}{q}\alpha$ and $\log(p)\,\alpha$ are both lower bounds for startup latency. Hence the proposed algorithm has startup latency at most two times that of the optimal startup latency.

THEOREM 3.2. *The bandwidth term of Algorithm 2 is bounded by* $(\max(pk_A, qk_B) + k_B)\,\beta = OPT + k_B\beta$. *(Optimality)*

PROOF. The total bandwidth of Lines 2-9 is $\max\left(k_A\frac{p}{q}, \frac{k_B}{\min\left(k_B, \frac{p}{q}\right)}\frac{p}{q}\right)\beta = \max\left(k_A\frac{p}{q}, k_B\right)\beta$. Lines 12-16 have a bandwidth of $\left(\min\left(\frac{p}{q}, k_B\right) - 1\right)\lceil\frac{k_B}{\frac{p}{q}}\rceil\beta$, since they are intra-group Allgather operations of $\lceil\frac{k_B}{\frac{p}{q}}\rceil$ message size over $\min\left(\frac{p}{q}, k_B\right)$ processes. When $k_B > \frac{p}{q}$, Lines 12-16 have a bandwidth of $\left(\frac{p}{q} - 1\right)\frac{k_B}{\frac{p}{q}}\beta$. Otherwise Lines 12-16 have a bandwidth of $(k_B - 1)\,\beta$. In both cases, the bandwidth

Algorithm 2: Full-duplex Inter-group All-to-All Broadcast with Message Fragmentation

1 # Exchange messages between two groups in $\frac{p}{q}$ steps.
2 **for** $i \in (0, .., \frac{p}{q} - 1)$ **do**
3 # Concurrent loop j
4 **for** $j \in (0, .., q - 1)$ **do**
5 $a_{\frac{p}{q}j+i}$ send $m_{A,\frac{p}{q}j+i}$ to b_j
6 # Messages from group B is fragmented.
7 b_j send $m_{B,j,i \bmod k_B}$ to a_{qi+j}
8 **end**
9 **end**
10 # Subgroups of processes in group A perform an intra-group Allgather to reconstruct fragmented messages
11 # Concurrent loop i, j
12 **for** $i \in (0, .., q - 1)$ **do**
13 **for** $j \in (0, .., \lfloor \frac{p}{qk_B} \rfloor)$ **do**
14 Intra-Allgather $a_{(x+k_Bj)q+i}, m_{B,i,x} \forall (0 \le x < min(\frac{p}{q}, k_B))$
15 **end**
16 **end**
17 # Subgroups of processes in group A perform an intra-group Allgather for m_B
18 **for** $i \in (0, .., \frac{p}{q} - 1)$ **do**
19 Intra-Allgather $a_{qi+j}, m_{B,j} \forall (0 \le j < q)$
20 **end**
21 # Group B performs intra-group Allgather for m_A
22 Intra-Allgather $b_j, m_{A,\frac{p}{q}j+i} \forall (0 \le j < q, 0 \le i < \frac{p}{q})$

is less than $k_B\beta$. Lines 18-20 have a total bandwidth of $(q-1)\,k_B\beta$, since they are intra-group Allgather functions of k_B message size over q processes. Line 22 has communication cost $\left(p - \frac{p}{q}\right) k_A\beta$, equivalent to line 15 of Algorithm 1. The intra-group Allgather at line 22 runs parallel to lines 12-20. It follows that the overall bandwidth can be bounded using Equation 1.

$$\begin{aligned} b.w &\le \left(\max\left(k_A\frac{p}{q}, k_B\right) + \max\left(k_A\left(p - \frac{p}{q}\right), k_Bq\right)\right)\beta \\ &\le \left(\max\left(k_A\frac{p}{q}, k_B\right) + \max\left(k_A\frac{p}{q}, k_B\right)(q-1) + k_B\right)\beta \\ &= \left(\max\left(pk_A, qk_B\right) + k_B\right)\beta \end{aligned} \tag{1}$$

□

Message completeness means that all processes in group A has messages m_B and all processes in group B has message m_A. We apply Lemma 3.3 to prove the message completeness of Algorithm 2. This lemma has been proven in [8].

LEMMA 3.3. *If $ax + y = c$ for $a, c \in \mathbb{Z}^+$, then there is a unique non-negative integer solution pair (x, y) for $y < a$.*

THEOREM 3.4. *Algorithm 2 achieves the objective of inter-group Allgather function (message completeness).*

PROOF. For processes in group B, let $b_c \in B$ be an arbitrary process for $0 \le c < q$. In Lines 2-9, b_c receives messages of the form $m_{A,\frac{p}{q}c+x}$ $\forall 0 \le x < \frac{p}{q}$. In Line 22, b_c participates in the Allgather function with messages $m_{A,\frac{p}{q}c+x}$ $\forall 0 \le x < \frac{p}{q}$. In the end, b_c has all messages of the form $m_{A,\frac{p}{q}y+x}$ $\forall 0 \le x < \frac{p}{q}, 0 \le y < q$, which is equivalent to $m_{A,z}$ $\forall 0 \le z < p$. Hence b_c is message complete for m_A. Since b_c is arbitrarily chosen, all processes in group B eventually receive message m_A.

It remains to show that all processes in group A receives all messages from group B. Let $a_c \in A$ be an arbitrary process for some $0 \le c < p$. In Lines 2-9, since $j < q$, there are unique solutions $i = i_c$ and $j = j_c$ such that $c = qi + j$ by Lemma 3.3. As a result, a_c receives only one message, which is $m_{B,j_c,i_c \bmod k_B}$.

In Lines 12-16, if $k_B \ge \frac{p}{q}$, the j loop has upper bound 0. Since $x < \frac{p}{q}$, $i = j_c$ and $x = i_c$ are the unique solutions for $c = qx + i$ by Lemma 3.3. Hence a_c only participates in an intra-group Allgather function once, with its own message m_{B,j_c,i_c}. Furthermore, a_c has already obtained m_{B,j_c,i_c} in Lines 2-9. At the end of the Allgather operations, a_c receives all messages in form of $m_{B,j_c,y}$ $\forall 0 \le y < \frac{p}{q}$, which means that m_{B,j_c} has been received by a_c. Otherwise we have $k_B < \frac{p}{q}$. For simplicity of the proof, we assume that k_B divides $\frac{p}{q}$. Since $i < q$, $i = j_c$ and $k_Bj + x = i_c$ are the unique solutions for $c = (x + k_Bj)q + i$ by Lemma 3.3. Moreover, there exist unique solutions $j = j'_c$ and $x = x_c$ for $x + k_Bj = i_c$ by Lemma 3.3, because $i_c < \frac{p}{q}$, $j < \frac{p}{qk_B}$, and $x < k_B < \frac{p}{q}$. As a result, a_c only participates in an intra-group Allgather operation once, with its own message m_{B,j_c,x_c}. Furthermore, a_c has already obtained message $m_{B,j_c,i_c \bmod k_B} = m_{B,j_c,x_c}$ in Lines 2-9. At the end of the intra-group Allgather function, it has all messages in the form of $m_{B,j_c,y}$ $\forall 0 \le y < k_B$, which means that m_{B,j_c} has been received by a_c.

In Lines 18-20, $a_c = a_{qi+j}$ participates in an intra-group Allgather operation only if $j = j_c$ and $i = i_c$ by Lemma 3.3, with message m_{B,j_c}. Moreover, it has obtained message m_{B,j_c}. At the end of the intra-group Allgather operation, it has all messages of form $m_{B,y}$ $\forall 0 \le y < q$. Consequently, a_c is message complete. Since c is chosen arbitrarily, all processes in group A receive all messages from group B in the end. □

Algorithm 1 is a special case of Algorithm 2 when we force messages from group B to be indivisible. In other words, $k_B = 1$. As a result, the proof of message completeness for Algorithm 2 implies correctness of Algorithm 1.

Although the underlying topology assumption is fully connected, the algorithms do not utilize all links. Optimal bandwidth requires far less number of links than the $(p + q)(p + q - 1)$ links in the fully connected case. The intra-group Allgather only requires a ring topology to reach optimal bandwidth. For Algorithm 2, Lines 2-9 require at most $2p$ number of links, since every process in group B sends messages to $\frac{p}{q}$ processes in group A and every process in group A sends messages to only one process in group B. Lines 12-16 require each of the q groups of $\frac{p}{q}$ processes connected in a ring,

so the minimum number of connections is $\left(\frac{p}{q}-1\right)q = p-q$. Lines 18-22 require each of the $\frac{p}{q}+1$ groups of q processes connected in a ring, the minimum number of connections is $(q-1)\left(\frac{p}{q}+1\right) = p+q-\frac{p}{q}-1$. There are repeated links in these estimates. Nevertheless, the minimum number of links to maintain optimal bandwidth is less than $4p$, which is a linear number of connections in terms of process number.

3.2 Implementation for Remainders

In the previous sections, we assume that integer divisions do not have remainders. However, p, q, k_B can have any values in real applications, so the implementation has to be able to handle the cases when $\frac{p}{q}$ and $\frac{p}{qk_B}$ are not integers. We present one possible implementation.

If q does not divide p, let $p' = (p+q-(p \bmod q))$ and $q' = q + \lceil \frac{p \bmod q}{\frac{p'}{q}-1} \rceil$. Hence q divides p'. p' and q' converge to p and q as $\frac{p}{q}$ becomes large. Moreover, $p' < 2p$ and $q' < 2q$. We call p' and q' the ceiling upper bounds. We propose an implementation method for Algorithm 2 with bandwidth bounded by the communication cost for p' and q' number of processes.

We replace p with p' in Algorithm 2. Any operations on the dummy ranks are ignored. For example, the send/receive operation is ignored and intra-group Allgather operations do not take those ranks into account. Clearly, processes in group B receive all messages.

The data fragmentation at Lines 12-16 of Algorithm 2 can be implemented as the following. If $k_B \geq \frac{p'}{q}$, messages $m_{B,i}$ $\forall (p \bmod q) \leq i < q$ are divided into $\frac{p'}{q}-1$ pieces of message size $\lceil \frac{k_B}{\frac{p'}{q}-1} \rceil$ by replacing the upper limit of x with $\min\left(\frac{p'}{q}-1, k_B\right)$. Line 7 is changed accordingly by ignoring the $i = \frac{p'}{q}-1$ case, since the $\frac{p'}{q}^{th}$ segment does not exist. The bandwidth is less than $k_B\beta$ for Lines 12-16. If $k_B < \frac{p'}{q}$ and k_B does not divide $\frac{p'}{q}$, $m_{B,i}$ is divided into k_B pieces of message size 1 $\forall 0 \leq i < q$. Loop j at Line 13 is from 0 to $\lfloor \frac{p'}{qk_B} \rfloor - 1$. The bandwidth at Line 14 is equivalent to an intra-group Allgather operation of k_B processes and message size 1, which is less than $k_B\beta$. In either case, processes $\left(\frac{p'}{q}-1\right)q+i$ $\forall 0 \leq i < (p \bmod q)$, referred them as the remainder processes, are not referenced by the i, j double loops.

Lines 18-20 are the intra-group Allgather operation for m_B in group A. The range of iterations for the i loop range is replaced with $0, .., \frac{p'}{q}-2$. Processes with ranks $r_j = \left(\frac{p'}{q}-1\right)q + j$ $\forall 0 \leq j < (p \bmod q)$, the remainder processes defined previously, are not referenced by the loop. We assign process with rank r_j to the intra-group Allgather operation with loop index $i = \lfloor j / \lceil \frac{p \bmod q}{\frac{p'}{q}-1} \rceil \rfloor$ using dummy messages. Hence every intra-group Allgather function in Line 19 of Algorithm 2 has at most $q + \lceil \frac{p \bmod q}{\frac{p'}{q}-1} \rceil$ processes. The bandwidth of an intra-group Allgather function with $q + \lceil \frac{p \bmod q}{\frac{p'}{q}-1} \rceil$ processes and message size k_B is $\left(q + \lceil \frac{p \bmod q}{\frac{p'}{q}-1} \rceil - 1\right) k_B\beta = \left(q'-1\right) k_B\beta$.

Consequently, the total bandwidth of the algorithm implementation is less than $\left(\max\left(p'k_A, q'k_B\right) + k_B\right)\beta$ by replacing p with p' and q with q' in Equation 1 of Theorem 3.2.

Process a_c receives message $m_{B,c \bmod q}$ before Line 18 for any $0 \leq c < p-(p \bmod q)$ by the message completeness proof in the previous section. It remains to show that the remainder processes are message complete. In Lines 18-20, every remainder process is assigned to one of $\frac{p}{q}$ groups to participate in the intra-group Allgather operations with an empty message. In addition, these intra-group Allgather operations without adding the remainder processes have q processes, each with distinct $m_{B,c}$ $\forall 0 \leq c < q$. Hence all processes receive m_B in the end.

Figure 3 shows an example when $p = 8$ and $q = 3$ for $k_B = 2$. The ceiling upper bounds are $p' = 9$ and $q' = 4$. $k_B < \frac{p'}{q}$. Hence $m_{B,i}$ is divided into $m_{B,i,0}$ and $m_{B,i,1}$ $\forall 0 \leq i < q$. Processes a_6 and a_7 are the remainder processes.

4 EXPERIMENTAL RESULTS

A number of experiments have been performed on Cori, a Cray XC40 supercomputer at the National Energy Research Scientific Computing Center (NERSC). We use the Cray MPI compiler(cray-mpich/7.6.0), which is based on MPICH. Message buffers are filled with random numbers. We measure the MPI_Allgather 8 times and take the average. For timing of each measurement, we pick the maximum time among all processes. We implement Algorithm 2 for the full-duplex inter-group Allgather using MPI. Direct comparisons with the root gathering algorithm are made by calling the MPI_Allgather with an inter-group communicator.

We use four parameter settings on 64 nodes and 256 nodes to present a comprehensive evaluation of Algorithm 2. The first setting is when $p = q$ and $k_A = k_B$. With this setting, two groups of processes are symmetric. The second setting is when $p = 3q$ and $k_A = k_B$. This setting evaluates imbalanced number of processes with the same message size. The third setting is when $p = 3q$ and $k_A = 4k_B$. This setting evaluates imbalanced number of processes with larger group sending larger messages. The final setting is when $p = 15q$ and $k_B = 256k_A$. This setting evaluates the imbalanced number of processes with the smaller group sending large messages, which corresponds to the message hazard scenario.

The four settings for parameters are experimented with three process configurations. For the first two configurations, one process is assigned to a single node on 64 nodes and 256 nodes. Hence the total number of processes is 64 and 256 respectively. These two setups emulate a homogeneous communication model such that data transfer between any two processes have the same communication cost. For hybrid

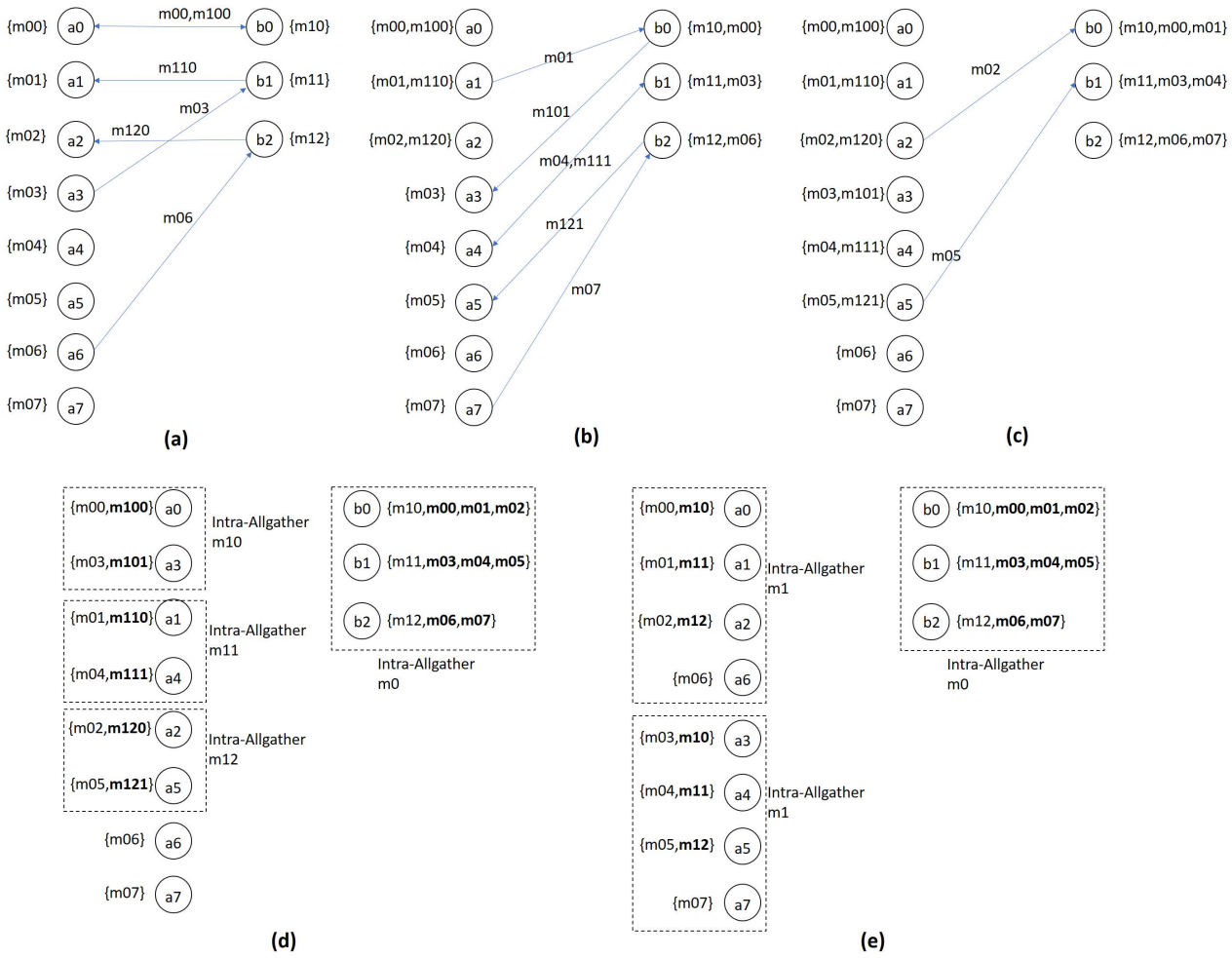

Figure 3: An execution of Algorithm 2 when $p = 8$ and $q = 3$ using the implementation strategy provided. The ceiling upper bounds are $p' = 9$ and $q' = 4$. $k_B = 2 < 3 = \frac{p'}{q}$.

MPI and OpenMP applications that use shared memory at the same node and process communications among nodes, this setup is reasonable. For the last setup, we assign 32 processes to every node of the 64 nodes. Thus, the total number of processes is 2048. This setup emulates real-world applications that fully utilize cores with MPI.

Figure 4a gives the timing results of the root gathering and Algorithm 2 when $p = 32$ and $q = 32$ with k_A and k_B from 64KB to 8MB on 64 nodes. Figure 4e gives the timing results when $p = 128$ and $q = 128$ with k_A and k_B from 64KB to 8MB on 256 nodes. Figure 4i shows the timing results when $p = 1024$ and $q = 1024$ with k_A and k_B from 8KB to 1MB on 64 nodes. We observe that the root gathering algorithm is slower since the root gathering algorithm has worse bandwidth than Algorithm 2.

Figure 4b gives the timing results of the root gathering and Algorithm 2 when $p = 48$ and $q = 16$ with k_A and k_B from 64KB to 8MB. Figure 4f shows the timing results when $p = 192$ and $q = 64$ with k_A and k_B from 64KB to 8MB. Figure 4j gives the timing results when $p = 1536$ and $q = 512$ with k_A and k_B from 8KB to 1MB. Similar to the $p = q$ pattern, the improvement of Algorithm 2 over the root gathering algorithm increases as the number of processes increase.

Figure 4c shows the timing results of the root gathering and Algorithm 2 when $p = 48$ and $q = 16$ with k_A from 8KB to 1MB and k_B from 32KB to 4MB. Figure 4g illustrates the timing results when $p = 192$ and $q = 64$ with k_A from 8KB to 1MB and k_B from 32KB to 4MB. Figure 4k gives the timing results when $p = 1536$ and $q = 512$ with k_A from 1KB to 128KB and k_B from 4KB to 512KB. This message setting differs from the previous one in the way that group A has 4 times larger message size k_A than message size k_B of group B. As the message size increases, observations on communication cost are the same as $k_A = k_B$.

Figure 4d shows the timing results of the root gathering and Algorithm 2 when $p = 60$ and $q = 4$ with k_A from 2KB to 256KB and k_B from 512KB to 64MB. Figure 4h gives the

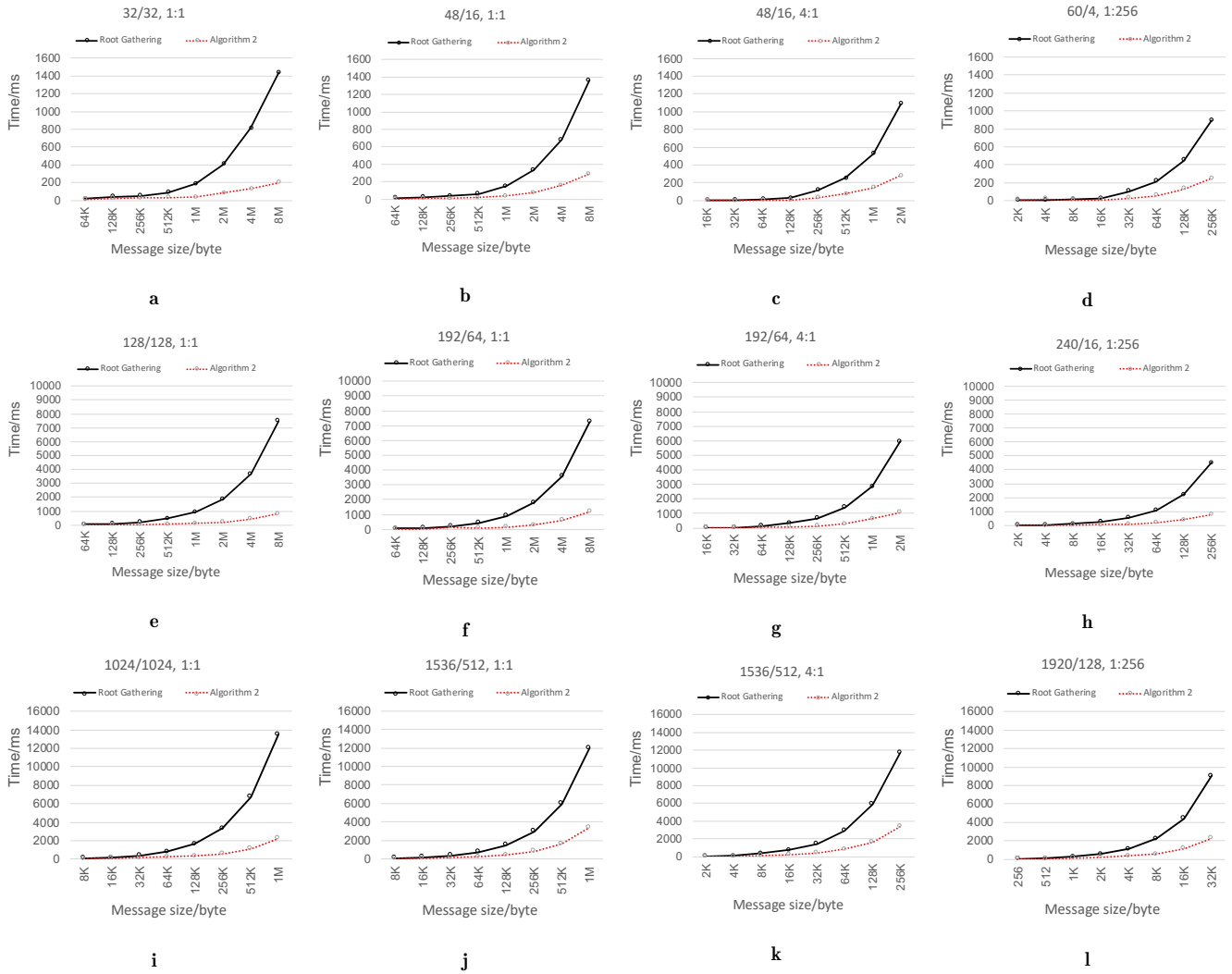

Figure 4: These figures illustrate the comparisons of Algorithm 2 and the root gathering algorithm. The title format is "p/q, $k_A : k_B$". p and q are the number of processes in group A and group B. The x-axis is message size m in terms of byte. The data type used is MPI_INT. The actual message size are m times their corresponding multipliers (k_A and k_B) in the title.

timing results when $p = 240$ and $q = 4$ with k_A from 2KB to 256KB and k_B from 512KB to 64MB. Figure 4l illustrates the timing results when $p = 1920$ and $q = 128$ with k_A from 256B to 32KB and k_B from 64KB to 8MB. This message pattern satisfies the message hazard condition $k_A p < k_B q$. Group B has smaller number of processes than group A, but it sends much larger message size k_B.

The bandwidth of the root gathering algorithm implemented by MPICH is $(\max(pk_A, qk_B) + 3pk_A + 3qk_B)\,\beta$. The bandwidth of Algorithm 2 is $\max(pk_A, qk_B)\,\beta$. Hence the expected communication cost improvement for large message sizes is Equation 2.

$$\frac{\max(pk_A, qk_B) + 3pk_A + 3qk_B}{\max(pk_A, qk_B)} \tag{2}$$

Table 1 shows the communication cost improvement of Algorithm 2 compared with the root gathering algorithm. In general, the 64 processes running on 64 nodes have results closely matching the expected improvement. The 256 processes running on 256 nodes have results showing larger improvements than the expected improvements. The 2048 processes running on 64 nodes have results showing smaller improvements than the expected improvements. The reason is that the communication cost between two any processes is not always the same in these two process settings. Hence the improvements do not follow the expected ratio when assuming the same communication cost between any two processes. Since Cori is pseudo-fully connected, two nodes selected arbitrarily can have different communication cost if the number of nodes used is large enough. Moreover, the communication cost between two processes on the same node is less than the communication cost between two processes on different nodes. In real applications, these two scenarios can happen. Nevertheless, Algorithm 2 is at least 3 times

Table 1: The following tables show the expected improvements computed by Equation 2 and actual improvements for different parameter settings and number of processes. Setting 1 refers $p = q$ and $k_A = k_B$. Setting 2 refers $p = 3q$ and $k_A = k_B$. Setting 3 refers to $p = 3q$ and $k_A = 4k_B$. Setting 4 refers to $p = 15q$ and $k_B = 256k_A$. The column names of the last three columns indicate number of nodes $\times$ number of processes per node.

Setting	Expected	64×1	256×1	64×32
1	7.00	7.25	9.27	5.95
2	5.00	4.70	6.13	3.55
3	4.25	3.89	5.52	3.45
4	4.18	3.58	5.64	3.95

faster than the root gathering algorithm. Hence the design is robust to heterogeneous communication costs.

For $pk_A > qk_B$, the communication cost of Algorithm 2 is independent of the message size k_B. For example, when $p = 3q$ and fixing k_A, the communication cost of $k_B = k_A$ and $k_B = \frac{1}{4}k_A$ are the same. However, the communication cost of the root gathering algorithm increases as k_B increases. Hence the improvement of communication cost becomes larger as k_B increases. Setting 2 and 3 in Table 1 support this conclusion. This result can also be observed in Figures 4b, 4f, 4j and Figures 4c, 4g, 4k.

5 CONCLUSION AND FUTURE WORK

We have proposed a full-duplex inter-group all-to-all broadcast algorithm with optimal bandwidth. The algorithm can be used to implement the MPI_Allgather function for MPI production libraries. MPI_Allgather is used by workflow systems that exchange messages among components.

We have formulated the algorithm and analysis with respect to communication cost and correctness. We have shown that the minimum number of connections used by the proposed algorithm for reaching the optimal bandwidth is linear with respect to the number of processes. Experimental results have shown that the algorithm is faster than the root gathering algorithm and in line with our expectations.

The current strategy for handling the remainder processes proposed in section 3.2 is designed for optimizing bandwidth when we assign one process per node. We will propose algorithms that can handle the remainder processes when multiple processes are allowed to run on the same node. Moreover,the algorithm can be extended for MPI_Allgatherv and probably new inter-group collective communications with group size larger than two.

6 ACKNOWLEDGMENT

This work is supported in part by the following grants: NSF awards CCF-1409601; DOE awards DE-SC0007456, DE-SC0014330; and NIST award 70NANB14H012.

REFERENCES

[1] 2017. MPICH3. http://www.mpich.org/downloads/.

[2] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. 1997. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems* 8, 11 (1997), 1143–1156.

[3] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert Van De Geijn. 2007. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 19, 13 (2007), 1749–1783.

[4] Ciprian Docan, Fan Zhang, Tong Jin, Hoang Bui, Qian Sun, Julian Cummings, Norbert Podhorszki, Scott Klasky, and Manish Parashar. 2015. Activespaces: Exploring dynamic code deployment for extreme scale data processing. *Concurrency and Computation: practice and Experience* 27, 14 (2015), 3724–3745.

[5] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. 2004. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, 97–104.

[6] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, et al. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (2016), 49–65.

[7] JP Hacker and WM Angevine. 2013. Ensemble data assimilation to characterize surface-layer errors in numerical weather prediction models. *Monthly Weather Review* 141, 6 (2013), 1804–1821.

[8] G. H. Hardy and E. M. Wright. 1979. *An Introduction to the Theory of Numbers* (5th ed.).

[9] S Lennart Johnsson and C-T Ho. 1989. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on computers* 38, 9 (1989), 1249–1268.

[10] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. 1994. *Introduction to parallel computing: design and analysis of algorithms*. Vol. 400. Benjamin/Cummings Redwood City.

[11] Jianwei Liao, Balazs Gerofi, Guo-Yuan Lien, Takemasa Miyoshi, Seiya Nishizawa, Hirofumi Tomita, Wei-Keng Liao, Alok Choudhary, and Yutaka Ishikawa. 2017. A flexible I/O arbitration framework for netCDF-based big data processing workflows on high-end supercomputers. *Concurrency and Computation: Practice and Experience* 29, 15 (2017), e4161.

[12] MPI Forum. 2015. *MPI: A Message-Passing Interface Standard. Version 3.1.* www.mpi-forum.org.

[13] Peter Sanders, Jochen Speck, and Jesper Larsson Träff. 2009. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.* 35, 12 (2009), 581–594.

[14] Christopher Sewell, Katrin Heitmann, Hal Finkel, George Zagaris, Suzanne T Parete-Koon, Patricia K Fasel, Adrian Pope, Nicholas Frontiere, Li-ta Lo, Bronson Messer, et al. 2015. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 50.

[15] Pedro Silva and João Gabriel Silva. 1999. Implementing MPI-2 extended collective operations. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 125–132.

[16] Anthony Skjellum, Nathan E Doss, and Kishore Viswanathan. 1994. *Inter-communicator extensions to MPI in the MPIX (MPI eXtension) Library*. Technical Report. Citeseer.

[17] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[18] Jesper Larsson Träff and Andreas Ripke. 2008. Optimal broadcast for fully connected processor-node networks. *J. Parallel and Distrib. Comput.* 68, 7 (2008), 887–901.

[19] Jesper Larsson Träff, Andreas Ripke, Christian Siebert, Pavan Balaji, Rajeev Thakur, and William Gropp. 2010. A pipelined algorithm for large, irregular all-gather problems. *International Journal of High Performance Computing Applications* 24, 1 (2010), 58–68.

[20] Fan Zhang, Ciprian Docan, Manish Parashar, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. 2012. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *26th International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 1352–1363.