Scaling All-to-all Operations Across Emerging Many-Core Supercomputers

Shannon Kinkead
Sandia National Laboratories
University of New Mexico
Albuquerque, New Mexico, USA
skinkea@sandia.gov
sgkinkead@unm.edu

Jackson Wesley
University of New Mexico
Albuquerque, New Mexico, USA
jwesley1@unm.edu

Whit Schonbein
Sandia National Laboratories
Albuquerque, New Mexico, USA
wwschon@sandia.gov

David DeBonis
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
ddebonis@lanl.gov

Matthew G. F. Dosanjh
Sandia National Laboratories
Albuquerque, New Mexico, USA
mdosanj@sandia.gov

Amanda Bienz
University of New Mexico
Albuquerque, New Mexico, USA
bienz@unm.edu

ABSTRACT

Performant all-to-all collective operations in MPI are critical to fast Fourier transforms, transposition, and machine learning applications. There are many existing implementations for all-to-all exchanges on emerging systems, with the achieved performance dependent on many factors, including message size, process count, architecture, and parallel system partition. This paper presents novel all-to-all algorithms for emerging many-core systems. Further, the paper presents a performance analysis against existing algorithms and system MPI, with novel algorithms achieving up to 3x speedup over system MPI at 32 nodes of state-of-the-art Sapphire Rapids systems.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; **Parallel algorithms**; *Massively parallel algorithms*; *Concurrent algorithms*.

KEYWORDS

Alltoall, collectives, many-core, MPI, HPC, communication, locality-aware, hierarchical

1 INTRODUCTION

Emerging HPC systems continue to scale up through the inclusion of accelerators (e.g., GPUs) and increasing core counts in each node. While recent algorithmic focus has favored GPU-aware optimizations, the large number of available CPU cores per node presents unique challenges related to inter-process communication. For instance, Sapphire Rapids machines such as Lawrence Livermore National Laboratory's (LLNL) Dane and Sandia National Laboratories' (SNL) Amber contain over 100 cores per node split across 2 sockets, and further split across 4 NUMA domains within each. Similarly, LLNL's El Capitan leverages AMD MI300A chips, which contain 96 cores per node. These node architectures introduce new complexities, amplifying bottlenecks such as injection bandwidth limitations [8]. Further, the large number of processes at each various level of locality (node, socket, NUMA region, etc) creates disparities in cost between regions of communication. For example, Sapphire Rapids systems such as Dane and Amber have 14 cores per NUMA region, 56 per socket, and 112 per node, allowing for significant amounts of communication at all levels of the hierarchy, including intra-NUMA, inter-NUMA, inter-socket, and inter-node.

The widely used all-to-all collective creates bottlenecks on emerging systems negatively impacting many different parallel operations. These operations include fast Fourier transforms (FFTs), matrix transpositions, and deep learning frameworks. The all-to-all operation consists of each process exchanging a unique portion of its local buffer with every other process, requiring a minimum of $\log_p$ messages or $s \cdot p$ bytes to be exchanged during an all-to-all operation of s bytes among p processes. As a result, all-to-all operations have poor scalability, with significant costs associated both with exchange size and process count. There are a large number of existing all-to-all algorithms, including optimizations for small data sizes that minimize message count as well as those for large exchanges which minimize message volume. Further, there are a number of existing node-aware optimizations, which aim to reduce message counts and sizes injected into the network, in exchange for increased on-node exchanges. However, the performance of all-to-all operations remains a bottleneck in many applications.

This paper presents optimizations for all-to-all operations on emerging many-core systems through the following contributions:

- Implements existing node-aware all-to-all optimizations, comparing a variety of underlying exchanges within each;
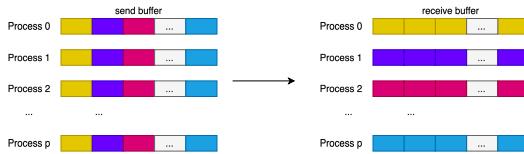
- Develops two novel all-to-all optimizations for many-core systems;
- Presents performance analysis and scaling studies of both existing and new optimizations across three state-of-the-art Sapphire Rapids supercomputers; and
- Achieves up to 3x speedup over system MPI when scaled to 32 nodes.

The remainder of this paper is structured as follows. Section 2 provides background information on the all-to-all operation, details existing approaches, and discusses related works. Two novel algorithmic approaches are described in Section 3, and Section 4 presents performance and scalability analysis of both new and existing all-to-all approaches on three Sapphire Rapids systems. Section 5 discusses the follow on work we are planning. Finally, Section 6 provides concluding remarks.

2 BACKGROUND

The all-to-all collective operation, as displayed in Figure 1, exchanges data among all processes within a communicator. Each process sends an equal-sized portion of data to every other process while concurrently receiving an equal portion originating on each other process, effectively transposing the original data across processes.


Figure 1: Alltoall between P processes

There are multiple commonly used all-to-all exchange algorithms. The Bruck algorithm [4] minimizes message counts through a tree-like exchange, requiring $\log_p$ steps to exchange data among p processes. This algorithm requires sending $\frac{s \cdot p}{2}$ bytes during each step, for a given data size s. While this algorithm excels for small data sizes, the data volume reduces its performance for larger values of s.

Large all-to-all exchanges minimize the amount of data communicated, with each rank communicating s bytes directly to each of the $p - 1$ other processes. There are multiple approaches for exchanging data, most commonly through either Pairwise Exchange, as detailed in Algorithm 1, or the non-blocking approach detailed in Algorithm 2.

Pairwise Exchange consists of $p - 1$ disjoint steps of communication. At step i, process p sends corresponding data to process $p + i$ and receives data from $p - i$, typically through blocking APIs such as `MPI_Sendrecv`. This approach limits network contention and queue search overheads with only a single exchange at any time. However, large synchronization overheads can occur with pairwise exchange, as at any step i, if process $p - i$ is not yet available, process p must wait idly.

Algorithm 1: Pairwise Exchange

> **Input:** p {process rank}
> n {process count}
> $s_{\text{size}}, s_{\text{type}}, s_{\text{buf}}$ {send size, type, and buffer}
> $r_{\text{size}}, r_{\text{type}}, r_{\text{buf}}$ {recv size, type, and buffer}
>
> **for** $i \leftarrow 0$ **to** n **do**
> $s_{\text{proc}} = p + i \mod n$
> $r_{\text{proc}} = p + n - i \mod n$
>   `MPI_Sendrecv`$(s_{\text{buf}}, s_{\text{size}}, s_{\text{type}}, s_{\text{proc}}, \ldots) r_{\text{buf}}, r_{\text{size}}, r_{\text{type}}, r_{\text{proc}}, \ldots)$

The non-blocking exchange consists of initializing all sends and receives with non-blocking communication, minimizing synchronization costs. However, this approach can yield significant overheads associated with queue search and network contention at large scales.

Algorithm 2: Non-blocking

> **Input:** p {process rank}
> n {process count}
> $s_{\text{size}}, s_{\text{type}}, s_{\text{buf}}$ {send size, type, and buffer}
> $r_{\text{size}}, r_{\text{type}}, r_{\text{buf}}$ {recv size, type, and buffer}
>
> **for** $i \leftarrow 0$ **to** n **do**
> $s_{\text{proc}} = p + i \mod n$
> $r_{\text{proc}} = p + n - i \mod n$
>   `MPI_Isend`$(s_{\text{buf}}, s_{\text{size}}, s_{\text{type}}, s_{\text{proc}}, \ldots)$
>   `MPI_Irecv`$(r_{\text{buf}}, r_{\text{size}}, r_{\text{type}}, r_{\text{proc}}, \ldots)$
> `MPI_Waitall`$(2 \times (n - 1), \ldots)$

All-to-all exchanges have been further optimized for recent generations of supercomputers, which have symmetric multiprocessing (SMP) nodes. Hierarchical collectives gather data among a leader per node before performing an all-to-all exchange among all leaders [21, 23]. Multi-leader collectives extend the hierarchical approach with multiple leaders per node. Node-aware aggregation alternatively aggregates all data to be communicated with another node, and then redistributes data within the node [20] (Note: A paper by the same name was published in CLUSTER, but only the extended ArXIV version contains the all-to-all algorithm). All of these existing optimizations are detailed further in Section 3.

2.1 Related Work

Additional all-to-all operations have been explored in a variety of contexts. A batched all-to-all [16] combines pairwise exchange and non-blocking approaches, communicating a batch of messages at each step, in an effort to balance the synchronization costs of pairwise exchange with the contention and queue search overheads of non-blocking. All-to-all exchanges have been further optimized for GPU-Aware communication, through optimizations to IPC communication [5], hierarchical optimizations [22], and compression [22]. Node-aware aggregation techniques have been widely explored

within all-to-all operations and well as its variable-sized counterpart, MPI_Alltoallv [12, 7]. Intra-node shared memory optimizations have also been explored throughout collective operations [9].

Locality-aware optimizations have been explored in many other contexts, including a variety of collectives [1, 3], collective IO [11], and irregular communication [2, 6, 18]. Topology-aware collectives have further been explored, extending optimizations to the network topology [19, 10, 17, 13, 14, 15].

This paper aims to build on the existing body of work on all-to-all collective optimizations through multiple contributions. First of all, we compare the existing node-aware, hierarchical, and multileader all-to-all implementations, including analyzing the underlying data exchanges. We extend the Locality-Aware collectives presented in [1, 3] to the all-to-all collective operation, as well as introduce the Multileader with Node-Aware all-to-all algorithm, and analyze the performance and scalability of these new operations in comparison to the existing all-to-all optimizations.

3 METHODS

All-to-all exchanges can be optimized on emerging architectures through node- and locality-awareness, differentiating between local and non-local exchanges. Each of these approaches, described in Algorithms 3 to 5, requires performing an all-to-all exchange on an MPI sub-communicator. Each of these all-to-all operations could use any underlying implementation, such as Bruck, pairwise exchange, or non-blocking.

3.1 Hierarchical

The hierarchical all-to-all, as described in Algorithm 3 and corresponding Figure 2, first gathers all data to a leader process, then

---

Algorithm 3: Hierarchical

Input: p {process rank}

n {process count}

$s_{\text{size}}, s_{\text{type}}, s_{\text{buf}}$ {send size, type, and buffer}

$r_{\text{size}}, r_{\text{type}}, r_{\text{buf}}$ {recv size, type, and buffer}

`local_comm` {All processes local to region}

ppn, l {Size and rank of local_comm}

`group_comm` {All processes with equal local rank}
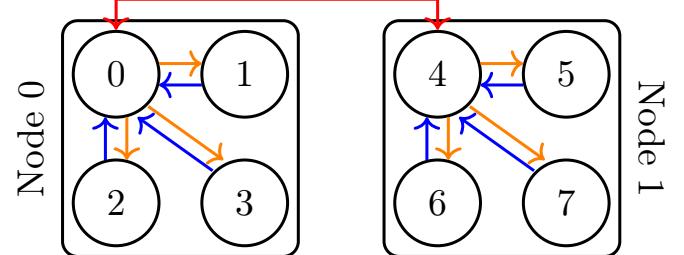
$s_{\text{buf}_{leader}} \leftarrow$ buffer of size $s_{\text{size}} \cdot n \cdot ppn$

$r_{\text{buf}_{leader}} \leftarrow$ buffer of size $r_{\text{size}} \cdot n \cdot ppn$

// Gather to leader

MPI_Gather($s_{\text{buf}}, s_{\text{size}} \cdot n, \ldots, s_{\text{buf}_{leader}}, \ldots,$ `local_comm`)

Repack Data

// Alltoall exchange between leaders

MPI_Alltoall($s_{\text{buf}_{leader}}, s_{\text{size}} \cdot ppn^2, \ldots, r_{\text{buf}_{leader}},$ $r_{\text{size}} \cdot ppn^2 \ldots,$ `group_comm`)

Repack Data

// Scatter from leader

MPI_Scatter($r_{\text{buf}_{leader}}, r_{\text{size}} \cdot n, \ldots, r_{\text{buf}}, \ldots,$ `local_comm`)

---

performs all-to-all exchanges among leaders, and finally scatters
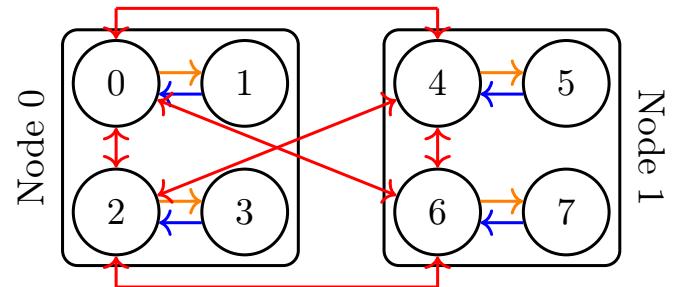
data from the leader to all non-leader processes. These steps are shown in color in Algorithm 3, where these colors correspond to the arrows in Figure 2. This standard hierarchical exchange has exactly


Figure 2: Hierarchical all-to-all. Blue arrows indicate the intra-node gather, red arrows show the internode all-to-all operation, and yellow arrows indicate the intra-node scatter operation

one leader per node, resulting in `local_comm` containing all processes in a node and `group_comm` containing the single leader per node. This algorithm reduces the number of inter-node messages by limiting the number of leaders, minimizing contention of the network, as well as per-node resources such as the NIC. However, with large numbers of active processes per node, the intra-node gather and scatter can result in significant overheads.

Multi-leader extensions to this algorithm consist of further partitioning the processes within each node so that multiple leaders are active within each all-to-all, as shown in Figure 3. In the case of


Figure 3: Multi-leader all-to-all, 2 leaders per node. Blue arrows indicate the intra-node gather, red nodes show the all-to-all operation between leaders, and yellow arrows show the intra-node scatter operation.

a multi-leader all-to-all, Algorithm 3 receives a `local_comm` communicator with the subset of processes per node corresponding to a single leader while `group_comm` contains all leaders. Multiple active leaders reduces the overheads of the gather and scatter operations. However inter-leader all-to-all operation requires an exchange among multiple leaders per node, increasing the amount of inter-node communication.

3.2 Node-Aware

While hierarchical and multi-leader collectives reduce the number of inter-node messages, each message increases in size, and many

---

Algorithm 4: Node-Aware

Input: p {process rank}

n {process count}

$s_{\mathsf{size}}, s_{\mathsf{type}}, s_{\mathsf{buf}}$ {send size, type, and buffer}

$r_{\mathsf{size}}, r_{\mathsf{type}}, r_{\mathsf{buf}}$ {recv size, type, and buffer}

`local_comm`   {All processes local to region}

ppn, l {Size and rank of local_comm}

`group_comm`   {All processes with equal local rank}

$tmp_{\mathsf{buf}} \leftarrow$ buffer of size s_{size}

// Inter-region Alltoall

MPI_Alltoall($s_{\mathsf{buf}}, s_{\mathsf{size}} \cdot ppn, \ldots, tmp_{\mathsf{buf}}, r_{\mathsf{size}} \cdot ppn\ldots,$ group_comm)
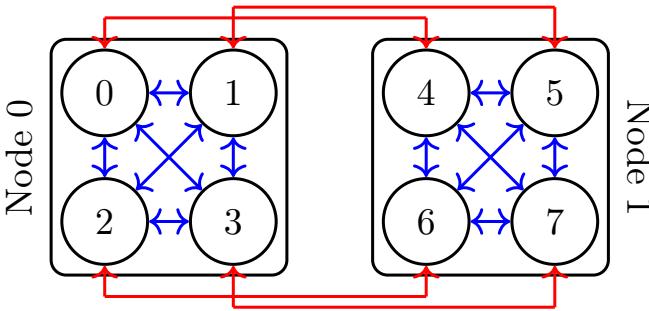
Repack Data

// Intra-region Alltoall

MPI_Alltoall($tmp_{\mathsf{buf}}, r_{\mathsf{size}} \cdot ppn, \ldots, r_{\mathsf{buf}}, r_{\mathsf{size}} \cdot ppn\ldots,$ local_comm)

---
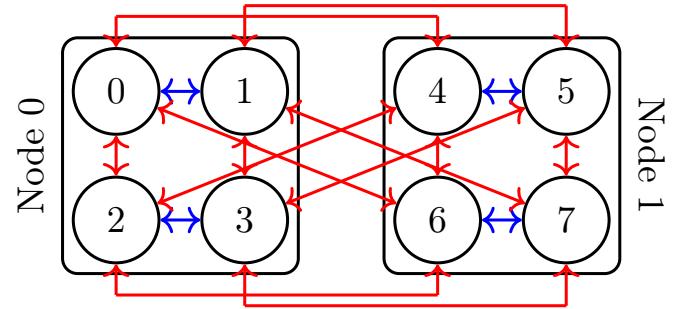
processes sit idle during the all-to-all exchange between leaders. Node-aware all-to-all exchanges, as detailed in Algorithm 4 and cor-


Figure 4: Node-aware all-to-all. Red Arrows indicate the internode all-to-all operation, while blue arrows indicate the intra-node all-to-all operation.

responding Figure 4, aggregate data among all processes per node, reducing the number of inter-node messages per process while evenly distributing data across all processes. This exchange consists first performing an `MPI_Alltoall` exchange on the `group_comm` communicator (red arrows), which consists of processes across all nodes that have equal local rank. After exchanging between nodes, all processes per node redistribute received data on `local_comm` (blue arrows), the set of all processes per node. The node-aware algorithm greatly reduces inter-node communication costs for large exchanges in exchange for an intra-node redistribution of data.

On emerging many-core systems, the intra-node redistribution across all processes per node can create moderate overheads. **This paper presents locality-aware aggregation, extending the standard node-aware approach with multiple groups of aggregation per node, as displayed in Figure 5.** Locality-aware aggregation is performed when `local_comm` within Algorithm 4 contains a subset of all processes per node, and `group_comm` contains a single process from each subset with corresponding local rank. This optimization reduces the cost of redistributing data lo-


Figure 5: Locality-aware all-to-all, 2 groups per node
Red arrows indicate the first round all-to-all, between regions, while blue arrows indicate the second all-to-all, within regions.

cally in exchange for a slight increase in steps of inter-node communication.

3.3 Combining Multi-leader with Node-Aware

While large bandwidth-bound exchanges are optimized when all processes communicate an equal portion of data, such as in locality-aware aggregation, small exchanges, which are bottlenecked by latency, benefit from the reductions in message count that are seen with hierarchical optimizations. However, multi-leader approaches increase inter-node message counts as each leader communicates with multiple leaders per node. As core counts increase on nodes, the number of leaders in the multi-leader approach increases correspondingly to offset the overheads of intra-node gathers and scatters. As a result, inter-node message counts are far from optimal.

Note: MPICH also includes an alltoall implementation described as node-aware multi-leaders based, which is documented as: 'Each rank on a node places the data for ranks sitting on other nodes into a shared memory buffer. Next each rank participates as a leader in inter-node Alltoall'. The authors are unaware of any publication that further details this algorithm, but based on this MPICH documentation, this implementation requires all ranks to participate in inter-node communication, similar to the standard node-aware approach presented in Algorithm 4.

This paper presents a novel combination of multi-leader and node-aware approaches, as detailed in Algorithm 5, in which the inter-node all-to-all exchange from Algorithm 3 is replaced with the node-aware operation from Algorithm 4. This multi-leader node-aware all-to-all, displayed in Figure 6 consists of first gathering to each leader (blue arrows) before exchanging data among corresponding leaders per node (red arrows). Data is then redistributed among the leaders within each node (brown arrows) before finally being scattered to the non-leader processes (yellow arrows). As a result, intra-node gather and scatter overheads are reduced with one leader per subset of each node's processes, while inter-node message counts remain low with each leader communicating only a single inter-node message to every node.

Algorithm 5: Multi-Leader + Node-Aware
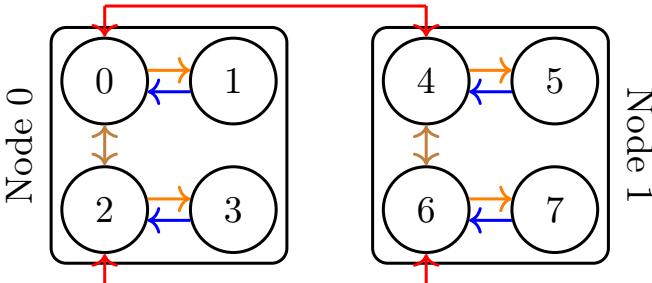
Input: p	{process rank}
n	{process count}
$s_{\text{size}}, s_{\text{type}}, s_{\text{buf}}$	{send size, type, and buffer}
$r_{\text{size}}, r_{\text{type}}, r_{\text{buf}}$	{recv size, type, and buffer}
node_comm	{All processes local to node}
ppn, l	{Size and rank of node_comm}
leader_comm	{All processes local to leader}
ppl	{Size of leader_comm}
group_comm	{All processes with equal local rank l}
n_{nodes}	{size of group_comm}
leader_group_comm	{All leaders local to the same node}

$s_{\text{buf}_{leader}} \leftarrow$ buffer of size $s_{\text{size}} \cdot n \cdot ppn$

$r_{\text{buf}_{leader}} \leftarrow$ buffer of size $r_{\text{size}} \cdot n \cdot ppn$

// Gather to leader
MPI_Gather($s_{\text{buf}}, s_{\text{size}} \cdot n, \ldots, s_{\text{buf}_{leader}}, \ldots,$ local_comm)

Repack Data

// Inter-region Alltoall
MPI_Alltoall($s_{\text{buf}_{leader}}, s_{\text{size}} \cdot ppn \cdot ppl, \ldots, r_{\text{buf}_{leader}},$
$r_{\text{size}} \cdot ppn \cdot ppl \ldots,$ group_comm)

Repack Data

// Intra-region Alltoall
MPI_Alltoall($r_{\text{buf}_{leader}}, r_{\text{size}} \cdot n_{nodes} \cdot ppl^2, \ldots, s_{\text{buf}_{leader}},$
$r_{\text{size}} \cdot n_{nodes} \cdot ppl^2, \ldots,$ leader_group_comm)

Repack Data

// Scatter from leader
MPI_Scatter($s_{\text{buf}_{leader}}, r_{\text{size}} \cdot n, \ldots, r_{\text{buf}}, \ldots,$ local_comm)


Figure 6: Multi-leader + Node-Aware all-to-all, 2 leaders per node. Blue errors indicate intra-node gather operations to the leader, red arrows indicate the inter-node all-to-all operation, brown arrows indicate the inter-region all-to-all region, and yellow arrows show the intra-node scatter operations.

4 RESULTS

To analyze the performance of all existing and new node-aware all-to-all optimizations, we tested the algorithms across three many-core systems: Dane (at LLNL), Amber (at SNL), and Toulomne (LLNL). Table 1 describes the architectures of these systems. Dane and Amber consist of Intel Sapphire Rapids processors, with 112 cores per node. System MPI on these machines is Intel MPI. Toulomne

uses AMD's MI-300A chips, which have 96 cores per node. Toulomne uses Cray-MPICH as the system MPI implementation. On each system, we used all cores available per node. All figures display the minimum of 3 runs for each data point.

Two experiments were performed on each system. The first experiment tested all-to-all algorithms using the maximum number of cores available per node (112 for Dane and Amber, 96 on Toulomne. All-to-all performance was then analyzed on all systems for data size exchanges from 4 to 4096 bytes per process, and scaled from 2 to 32 nodes. While the per-message scales were limited to 4096 bytes, at 32 nodes (3584 processes on Dane and Amber), each process must exchange a buffer of $14, 680, 064$ bytes. Additionally, the algorithms supporting multiple leaders (Multileader, Locality-Aware, and Multileader with Locality Awareness) were also tested with 4, 8, and 16 processes per leader. It is important to note, none of the multileader or locality-aware aggregation groups were explicitly mapped to regions of locality, such as NUMA domains. In most cases, group sizes force the groups to cross NUMA regions and/or sockets. Mapping processes to specific cores is beyond the scope of this paper. However, while it would decrease the portability of the algorithms, explicitly forcing this mapping is likely to improve locality-aware and multi-leader approaches beyond the results presented in the remainder of this section.

The second experiment analyzed the performance of intra-node vs inter-node communication in the hierarchical, multi-leader, node-aware, locality-aware, and multileader with node-awareness approaches. In this experiment, each of the algorithms above were analyzed for performance for data size exchanges from 4 to 4096 bytes, scaled from 2 to 32 nodes. Additional timings were measured for any internal gathers, scatters, and all-to-alls within each algorithm. Additionally, multi-leader, locality-aware, and multi-leader with node awareness were evaluated with 4, 8, and 16 processes per leader.

All node-aware optimizations were compared against the system MPI all-to-all implementation. Further, each of Algorithms 3 through 5 were tested with both pairwise exchange and non-blocking all-to-alls replacing each instance of MPI_Alltoall within the algorithm. **Except in the case of System MPI, solid lines indicate all instances of MPI_Alltoall calls using pairwise-exchange, while dashed lines indicate non-blocking underlying implementations.** While System MPI results are also plotted as a solid line, note that since both IntelMPI and CrayMPICH are proprietary, the underlying algorithm in all cases is unknown. The System MPI performance is included in all plots as a point of reference.

Due to space constraints, figures 7 through 16 analyze the performance of the various all-to-all algorithms only on Dane. The performance of the algorithms that are optimal at any message size is then presented for Amber and Tuolomne, in Figures 17 and 18, respectively.

Figure 7 compares the performance of hierarchical all-to-all operations with multi-leader (Algorithm 3) for exchanges of a variety of sizes across 32 nodes of Dane. For large data sizes, the performance increases corresponding to the number of leaders per node, with the standard hierarchical all-to-all (e.g. a single leader per node) performing poorly. Improvements are seen when increasing the number of leaders per node, decreasing the costs of the intra-node gather and scatter. For smaller data sizes, multi-leader

Name	CPU	Network	MPI	LibFabric Version
Dane	Intel Sapphire Rapids	Cornelis Networks Omni-Path	OpenMPI 4.1.2	2.2.0
Amber	Intel Sapphire Rapids	Cornelis Networks Omni-Path	OpenMPI 4.1.6	2.1.0
Tuolomne	AMD Instinct MI300A	Slingshot-11	HPE Cray MPICH 8.1.32	2.1

Table 1: System Architectures
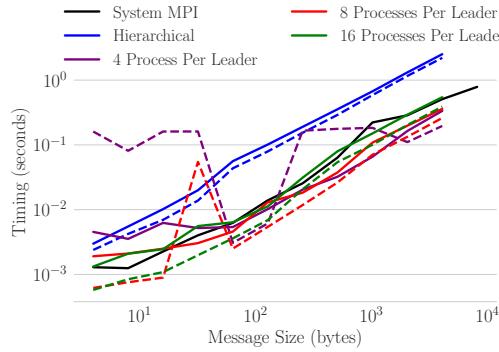
Figure 7: Hierarchical vs Multileader

algorithms outperform standard hierarchical, but fewer leaders are beneficial, with 4 leaders per node achieving best performance. Intra-node gathers and scatters have less overhead for these smaller data sizes. Further, large numbers of leaders increase inter-node message counts.
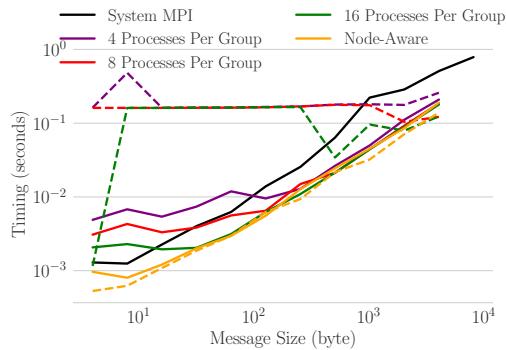
Figure 8: Node-Aware vs Locality-Aware

Figure 8 presents the cost of node- and locality-aware all-to-all exchanges (Algorithm 4) of various sizes across 32 nodes of Dane. While node-aware aggregation performs best for most data sizes, locality-aware aggregation with 4 to 20 groups outperforms node-aware aggregation at the largest tested data size. While this improvement is only seen at the largest tested data size, the improvement is consistently seen from 8 to 32 nodes as shown in Figure 12. Note: we plot the minimum of each benchmarking run. The nonblocking all-to-all shows a large amount of variability, resulting in the sharp decrease in execution time at 1024 bytes in figures 8 and 10. Other runs do not show this dip.
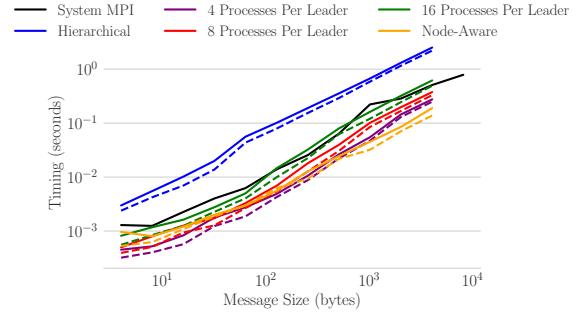
Figure 9: Multileader + Locality

Figure 9 shows the impact of the number of leaders per node on multi-leader node-aware all-to-alls (Algorithm 5). If a single leader was chosen, this algorithm reduces to the hierarchical approach. If all processes per node are their own leaders, this reduces to the node-aware algorithm. For small data sizes, this algorithm performs best with many but not all processes per node performing inter-node communication, limiting the number of leaders to around 20.
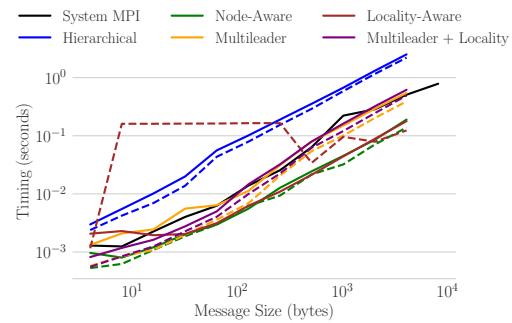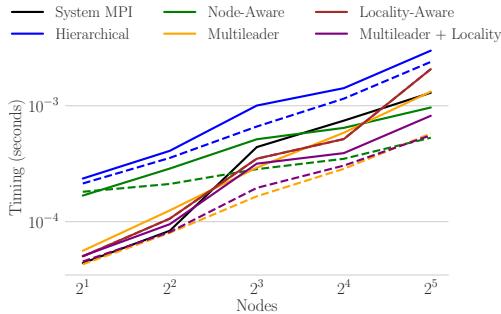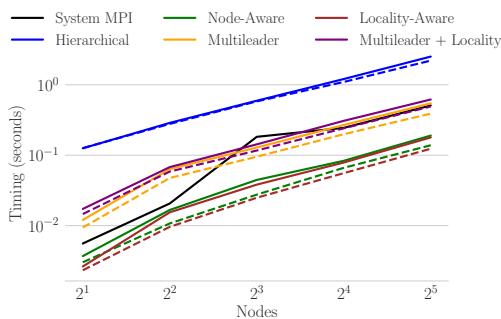

Figure 10: Various Sizes, 32 Nodes
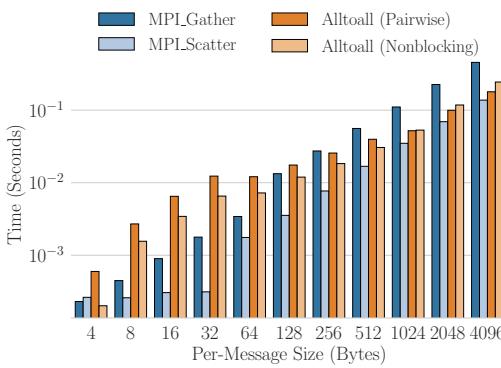
Figure 10 shows the performance of all algorithms for a variety of data sizes across 32 nodes of Dane. Based on the results presented in Figures 7 to 9, all multi-leader and locality-aware results use 28 leaders/groups per node (4 processes per leader/group). The multi-leader node-aware approach performs best for small data sizes, notable outperforming system MPI which is likely using the Bruck algorithm. The node-aware algorithm optimizes larger data sizes while the largest sizes are improved with locality-aware aggregation.

Figures 11 and 12 show the performance of all algorithms when scaled across various node counts, for a data size of 4 bytes and 4096 bytes, respectively.

Figure 11: Message Size: 4 bytes, Node Scaling

Figure 12: Message Size 4096 bytes, Node Scaling

4.1 Inter- and Intra-node communication

Due to space constraints, figures 13 through 16 only show all-to-all algorithms that leverage pairwise exchange for the internal all-to-all operation. Figure 13 shows the performance of the individual


Figure 13: Dane, various sizes, 32 nodes, Hierarchical Timing Breakdown

communication components of the hierarchical algorithms on 32 nodes on Dane. At smaller sizes (fewer than 256 byte messages), internode communication (the all-to-all operation between the single leader on each node) dominates communication. The nonblocking implementation outperforms the pairwise implementation until message sizes reach 2048 bytes. At larger message sizes (256 bytes

and higher), the gather operation (intra-node communication) begins to dominate the execution time.
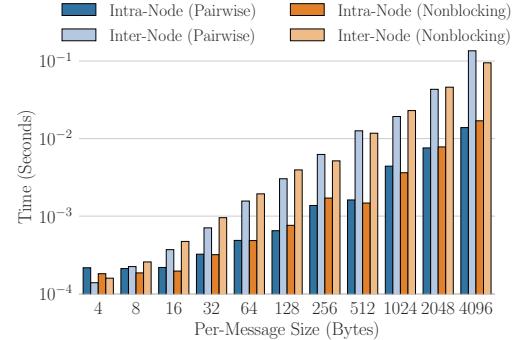
Figure 14: Dane, various sizes, 32 nodes, Node Aware Timing Breakdown

Figure 14 shows the performance of the inter-node and intra-node communication in the node-aware algorithms on 32 nodes on Dane. In both the pairwise and nonblocking cases, the inter-node communication again dominates the overall execution time. However, intra-node communication scales with internode communication.
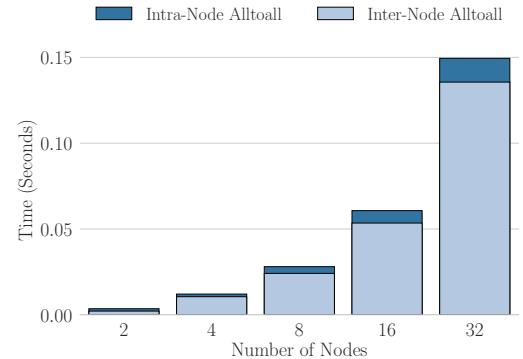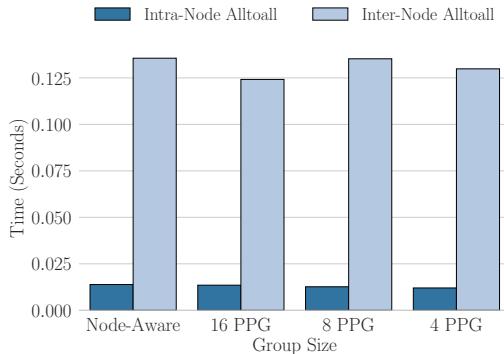

Figure 15: Dane, 1024 Integer Message Size, 2 to 32 nodes, Pairwise Node Aware
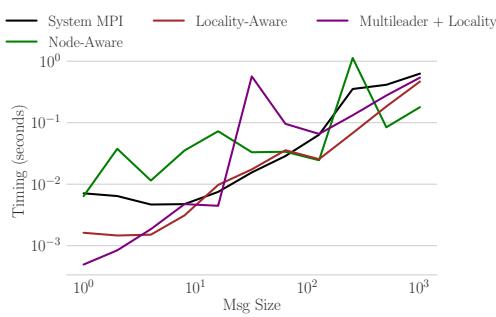
Figure 15 shows the performance of the individual components of the node-aware algorithm on Dane for a constant message size (4096 bytes), performed on node counts ranging from 2 to 32. Similar to the previous timing data, the inter-node communication dominates the overall latency, regardless of node count.

Figure 16 shows the impact of scaling the number of processes per group in the Locality-Aware algorithm, scaling from 1 process per group (the Node-Aware algorithm), to 16 processes per group. In each case, the inter-node communication dominates the exchange, while the intra-node communication shows little variability between processes per group. However, we do see slightly improved internode communication when there are 16 and 4 processes per group, which corresponds to 7 and 28 leaders, respectively, participating in the internode communication.
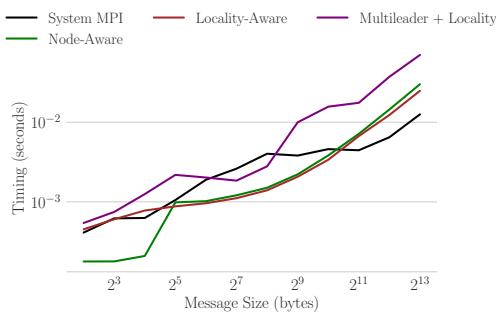
Figure 16: Dane, 1024 Integers, 32 Nodes, Various Processes per Leader, Locality Aware Timing Breakdown

4.2 Amber and Tuolomne

Figure 17: Amber, Various Sizes, 32 Nodes

Figure 17 shows the performance of the node-aware, locality-aware, and multi-leader node-aware approaches, in comparison to system MPI, on 32 nodes of Amber. Similar to previous results, the multi-leader node-aware approach performs best for small data sizes, while node-aware aggregation outperforms all other approaches for large data sizes.

Figure 18: Tuolomne, Various Sizes, 32 Nodes

Finally, figure 18 shows the performance of the all-to-all algorithms on 32 nodes of Tuolomne. In this case, the node-aware algorithm shows the best performance for small message sizes, though

system MPI performance is only slightly slower. The locality-aware and multi-leader node-aware show greater latency. However, at larger message sizes, system MPI shows the best performance.

4.3 Discussion

The results exploring group and locality size in the Locality-Aware, Multileader, and Multileader with Node-Aware algorithms show an interesting trend. Figure 16 indicates that both 16 processes per group and 4 processes per group outperform other group sizes and the Node-Aware algorithm (in which all processes on a node are part of a single group), which indicates that optimizing the group/locality size is not a single-modal function. With the increasing prevalence of many-core nodes, multi-socket nodes with multiple NUMA domains per CPU, we posit that this is likely due to the added complexity of intra-node communication.

5 FUTURE WORK

The algorithms presented in this paper present opportunities for further exploration. One such area is extending the algorithms to other collective operations. This paper has focused on all-to-all which is a critical collective. We plan to extend this work by applying this approach on both other HPC critical collectives (all-gather, broadcast, etc.) and AI critical collectives (allreduce, reduce-scatter, etc.). Additionally, using the insights from the data gathered here, we can develop a model to evaluate these impacts at capability-scale.

Future research includes the exploration the impact of NUMA domain on locality-aware algorithms, as well as the impact of using MPI Datatypes on the repacking cost inherent in these algorithms. Additionally, expanding this research to other systems, including those with GPUs, will be valuable in understanding how these algorithm choices can be applied more generally. Finally, we plan to explore how the optimal algorithm can be dynamically selected for a given computer, system MPI, process count, and data size.

6 CONCLUSIONS

In this paper, we compare several all-to-all algorithms, including two novel algorithms, on Sapphire Rapids and MI-300A. We noted that the multi-leader node-aware approach showed superior performance to the other approaches for small message sizes across a range of node counts, outperforming system MPI in many cases. Locality-aware aggregation shows promise, outperforming standard node-aware aggregation for the largest tested data sizes on Dane. However, to achieve the full benefit of this method, groups should likely be mapped to regions of locality such as NUMA domains. This paper also presented a study of the impact of group size and the number of leaders per node on multi-leader algorithms, showing generally improved performance with larger leader counts. Additionally, we demonstrated that inter-node communication dominates communication in hierarchical approaches for small message sizes, though intra-node communication dominates the data exchange for larger message sizes. However, in Node- and Locality-Aware exchanges, the inter-node communication dominates at all message sizes, regardless of node count.

We note that while leveraging non-blocking communication in the internal all-to-all operation often improves the speed at

data exchanges, especially at smaller message sizes, it also shows greater variability that can result in unpredictable performance degradation.

REFERENCES

[1] Amanda Bienz, Shreeman Gautam, and Amun Kharel. 2022. A Locality-Aware Bruck Allgather. In *Proceedings of the 29th European MPI Users' Group Meeting* (Chattanooga, TN, USA) *(EuroMPI/USA '22)*. Association for Computing Machinery, New York, NY, USA, 18–26. https://doi.org/10.1145/3555819.3555825

[2] Amanda Bienz, William D. Gropp, and Luke N. Olson. 2019. Node aware sparse matrix–vector multiplication. *J. Parallel and Distrib. Comput.* 130 (2019), 166–178. https://doi.org/10.1016/j.jpdc.2019.03.016

[3] Amanda Bienz, Luke Olson, and William Gropp. 2019. Node-Aware Improvements to Allreduce. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*. 19–28. https://doi.org/10.1109/ExaMPI49596.2019.00008

[4] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. 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. https://doi.org/10.1109/71.642949

[5] Chen-Chun Chen, Kawthar Shafie Khorassani, Quentin G Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K

Panda. 2022. Highly efficient alltoall and alltoallv communication algorithms for GPU systems. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 24–33.

[6] Gerald Collom, Rui Peng Li, and Amanda Bienz. 2023. Optimizing Irregular Communication with Neighborhood Collectives and Locality-Aware Parallelism. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (Denver, CO, USA) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 427–437. https://doi.org/10.1145/3624062.3624111

[7] Ke Fan, Steve Petruzza, Thomas Gilray, and Sidharth Kumar. 2024. Configurable Algorithms for All-to-All Collectives. In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*. 1–12. https://doi.org/10.23919/ISC.2024.10528936

[8] William Gropp, Luke N Olson, and Philipp Samfass. 2016. Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test. In *Proceedings of the 23rd European MPI Users' Group Meeting*. 41–50.

[9] Surabhi Jain, Rashid Kaleem, Marc Gamell Balmana, Akhil Langer, Dmitry Durnov, Alexander Sannikov, and Maria Garzaran. 2018. Framework for Scalable Intra-Node Collective Operations using Shared Memory. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 374–385. https://doi.org/10.1109/SC.2018.00032

[10] Krishna Kandalla, Hari Subramoni, Abhinav Vishnu, and Dhabaleswar K. Panda. 2010. Designing topology-aware collective communication algorithms for large scale InfiniBand clusters: Case studies with Scatter and Gather. In *2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 1–8. https://doi.org/10.1109/IPDPSW.2010.5470853

[11] Qiao Kang, Sunwoo Lee, Kaiyuan Hou, Robert Ross, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao. 2020. Improving MPI Collective I/O for High Volume Non-Contiguous Requests With Intra-Node Aggregation. *IEEE Transactions on Parallel and Distributed Systems* 31, 11 (2020), 2682–2695. https://doi.org/10.1109/TPDS.2020.3000458

[12] Kawthar Shafie Khorassani, Ching-Hsiang Chu, Quentin G. Anthony, Hari Subramoni, and Dhabaleswar K. Panda. 2021. Adaptive and Hierarchical Large Message All-to-all Communication Algorithms for Large-scale Dense GPU Systems. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 113–122. https://doi.org/10.1109/CCGrid51090.2021.00021

[13] Teng Ma, George Bosilca, Aurelien Bouteiller, and Jack Dongarra. 2012. HierKNEM: An Adaptive Framework for Kernel-Assisted and Topology-Aware Collective Communications on Many-core Clusters. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. 970–982. https://doi.org/10.1109/IPDPS.2012.91

[14] Teng Ma, George Bosilca, Aurelien Bouteiller, and Jack J Dongarra. 2013. Kernel-assisted and topology-aware MPI collective communications on multicore/many-core platforms. *J. Parallel and Distrib. Comput.* 73, 7 (2013), 1000–1010.

[15] Seyed H. Mirsadeghi and Ahmad Afsahi. 2016. Topology-Aware Rank Reordering for MPI Collectives. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1759–1768. https://doi.org/10.1109/IPDPSW.2016.139

[16] Evelyn Namugwanya, Amanda Bienz, Derek Schafer, and Anthony Skjellum. 2023. Collective-Optimized FFTs. *arXiv preprint arXiv:2306.16589* (2023).

[17] Paul Sack and William Gropp. 2012. Faster topology-aware collective algorithms through non-minimal communication. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New Orleans, Louisiana, USA) *(PPoPP '12)*. Association for Computing Machinery, New York, NY, USA, 45–54. https://doi.org/10.1145/2145816.2145823

[18] Trevor Steil, Tahsin Reza, Benjamin Priest, and Roger Pearce. 2023. Embracing Irregular Parallelism in HPC with YGM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 35, 13 pages. https://doi.org/10.1145/3581784.3607103

[19] Hari Subramoni, Krishna. Kandalla, Jithin Jose, Karen Tomko, Karl Schulz, Dmitry Pekurovsky, and Dhabaleswar K. Panda. 2014. Designing Topology-Aware Communication Schedules for Alltoall Operations in Large InfiniBand Clusters. In *2014 43rd International Conference on Parallel Processing*. 231–240. https://doi.org/10.1109/ICPP.2014.32

[20] Jesper Larsson Träff. 2019. Decomposing Collectives for Exploiting Multi-lane Communication. *arXiv preprint arXiv:1910.13373* (2019).

[21] Jesper Larsson Träff and Antoine Rougier. 2014. MPI collectives and datatypes for hierarchical all-to-all communication. In *Proceedings of the 21st European MPI Users' Group Meeting*. 27–32.

[22] Qinghua Zhou, Pouya Kousha, Quentin Anthony, Kawthar Shafie Khorassani, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda. 2022. Accelerating MPI all-to-all communication with online compression on modern GPU clusters. In *International Conference on High Performance Computing*. Springer, 3–25.

[23] Hao Zhu, David Goodell, William Gropp, and Rajeev Thakur. 2009. Hierarchical collectives in MPICH2. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10, 2009. Proceedings 16*. Springer, 325–326.