# Node-Aware Improvements to Allreduce

Amanda Bienz
*Department of Computer Science*
*University of Illinois*
*at Urbana-Champaign*
Urbana, Illinois
bienz2@illinois.edu

Luke N. Olson
*Department of Computer Science*
*University of Illinois*
*at Urbana-Champaign*
Urbana, Illinois
lukeo@illinois.edu

William D. Gropp
*Department of Computer Science*
*University of Illinois*
*at Urbana-Champaign*
Urbana, Illinois
wgropp@illinois.edu

***Abstract*—The `MPI_Allreduce` collective operation is a core kernel of many parallel codebases, particularly for reductions over a single value per process. The commonly used allreduce recursive-doubling algorithm obtains the lower bound message count, yielding optimality for small reduction sizes based on node-agnostic performance models. However, this algorithm yields duplicate messages between sets of nodes. Node-aware optimizations in MPICH remove duplicate messages through use of a single master process per node, yielding a large number of inactive processes at each inter-node step. In this paper, we present an algorithm that uses the multiple processes available per node to reduce the maximum number of inter-node messages communicated by a single process, improving the performance of allreduce operations, particularly for small message sizes.**

***Index Terms*—Parallel, Parallel algorithms, Interprocessor communications**

## I. Introduction

The advance of parallel computers towards exascale motivates the need for increasingly scalable algorithms. Emerging architectures provide increased process counts, yielding the potential to run increasingly large and complex applications, such as those relying on linear system solvers or neural networks. As applications are scaled to a larger number of processes, MPI communication becomes a dominant factor of the overall cost.

The `MPI_Allreduce` [1] is a fundamental component of a wide range of parallel applications, such as norm calculations in iterative methods, inner products in Krylov subspace methods, and gradient mean calculation in deep neural networks. The allreduce operation consists of performing a reduction operation over values from all processes, such as a summing values or determining the maximum. Therefore, the cost of the allreduce increases with process count, as displayed in Figure 1, motivating the need for improved performance and scalability on emerging architectures.

In this paper, we present an allreduce algorithm based on node-awareness, which exchanges inter-node communication for less costly intra-node messages as well as increased computational requirements. This algorithm reduces the number of inter-node messages from $\log_2(n)$ to $\log_{\texttt{ppn}}(n)$, where $n$ is the number of nodes involved and `ppn` is the number of processes per node, yielding significant speedups over standard allreduce methods for small message sizes.

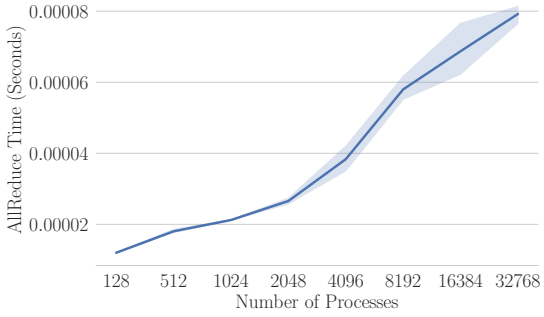


Fig. 1. Time required for `MPI_Allreduce` to reduce a single double-precision floating-point value across a variety of process counts on Blue Waters [2], [3]. The shaded area shows the variation between five separate runs.

The remainder of this paper is organized as follows. Section 2 describes common allreduce algorithms along with optimizations, including the node-aware allreduce algorithm that is implemented in MPICH [4]. In Section 3, we present a node-aware allreduce algorithm that reduces the number and size of inter-node messages. Performance models for the various allreduce algorithms are analyzed in Section 4, and performance results are displayed in Section 5. Finally, Section 6 contains concluding remarks.

## II. Background

The `MPI_Allreduce` operates upon $s$ sets of $p$ values into $s$ resulting values through operations such as summations or calculating the maximum value. These values are initially distributed evenly across $p$ processes and results are returned to all processes. A reduction requires $(p-1)\cdot s$ floating-point operations if the full reduction is performed on a single process. Therefore, splitting across $p$ processes yields a lower bound of $\frac{(p-1)\cdot s}{p}$ floating-point operations. Furthermore, as data is distributed across all processes, a minimum of $\log_2(p)$ messages must be communicated. Finally, the minimal data

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.

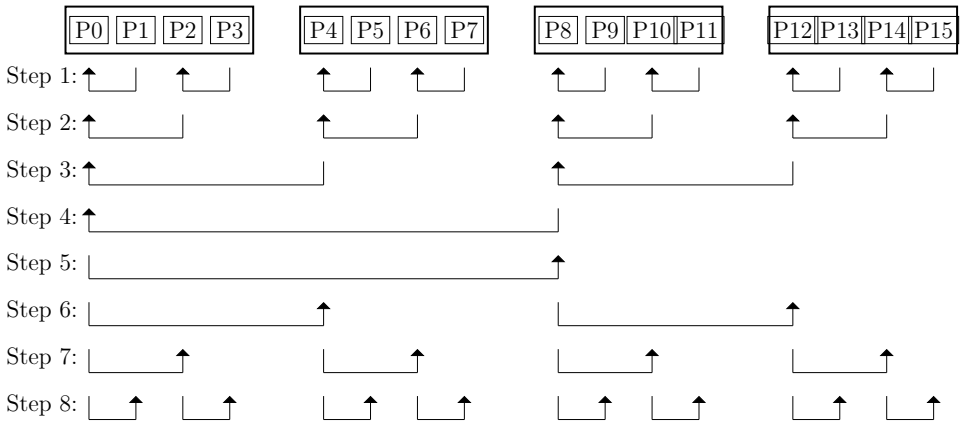


Fig. 2. Data movement for a tree allreduce over 16 processes, with data first reduced to a process $P0$ before being broadcast to other processes.

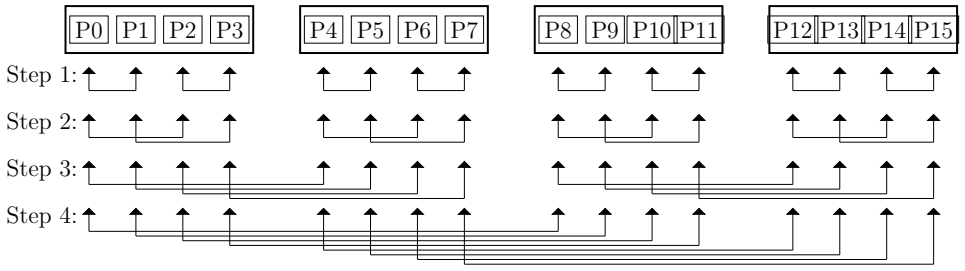


Fig. 3. Communication pattern for a recursive-doubling allreduce with 4 nodes, each containing 4 processes. Data is exchanged at each step and all processes are active in the reduction.

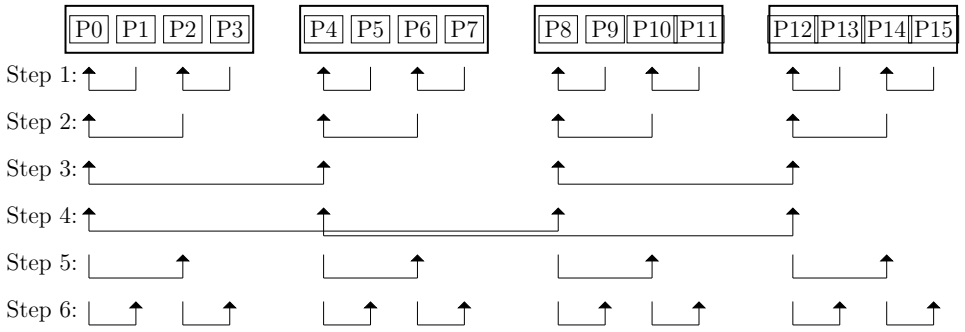


Fig. 4. Data movement for the MPICH SMP allreduce algorithm over 16 processes partitioned across 4 nodes. The data is reduced to a master process per node in steps 1 and 2, before being reduced among master processes through recursive doubling in steps 3 and 4. Finally, the data is broadcast from the master process to all idle processes per node.

transfer size is $\frac{2\cdot(p-1)\cdot s}{p}$ as $\frac{(p-1)\cdot s}{p}$ values must be both sent and received [5], [6].

There are a large number of existing allreduce algorithms with various levels of optimality dependent on message size $s$ and process count $p$. A straightforward algorithm, displayed in Figure 2, first reduces the data onto a master process before broadcasting to all other processes.

Assuming tree broadcasts and reductions, this algorithm requires $2 \cdot \log_2(p)$ messages and $2 \cdot \log_2(p) \cdot s$ values to be transported. Furthermore, the tree algorithm requires $\log_2(p) \cdot s$ floating-point operations. This algorithm is sub-optimal, with communication requirements significantly larger than ideal [7]. Furthermore, the tree algorithm yields large load imbalances with large numbers of inactive processes.

Recursive-doubling or the butterfly allreduce, exemplified in Figure 3, improves upon the tree algorithm by utilizing all processes, with sets of processes exchanging data at each step. This algorithm reduces the number and size of messages to $\log_2(p)$ and $\log_2(p) \cdot s$, respectively, while retaining computation requirements equivalent to the tree algorithm. Recursive-doubling achieves the lower bound for message count, yielding a near-optimal algorithm, based on the postal model, for small messages and power of 2 process counts [7], [8]. This algorithm can be altered to work efficiently for non-power of two process counts [5], [9].

Alternative algorithms optimize bandwidth and local com-

putation requirements for larger message sizes. Assuming $p$ is relatively small, data can be split into $p$ portions and communicated to all other processes in a pipeline [10]. Portioning and pipelining the data achieves the lower bound cost associated with data transport. However, each process sends $2 \cdot (p-1)$ messages, yielding reduced scalability for large process counts. Rabenseifner's algorithm [5], [8] improves upon pipelining by implementing a reduce-scatter, or a reduction with results scattered among the processes, followed by an allgather of these results. While remaining optimal in data transport, this algorithm requires only $2\log_2(p)$ messages.

### A. Node-Awareness

Emerging architectures often consist of a large number of symmetric multiprocessing (SMP) nodes, each containing many processes. Intra-node processes share memory, allowing for data to be quickly transported between processes on a node. Inter-node data transport requires data to be split into packets, injected into the network, and transported across network links to the node of destination. Therefore, inter-node communication is significantly more expensive than intra-node. Node-agnostic performance models, such as the postal model, fail to accurately capture the costs associated with inter-node communication. This model can be improved by splitting communication into intra- and inter-node as well as adding injection bandwidth limits [11], [12].

Standard allreduce methods, such as recursive-doubling and Rabenseifner's algorithm, reduce among processes in a node-agnostic fashion. Therefore, multiple messages and duplicate data are often exchanged between a set of nodes. This is exemplified in steps 3 and 4 of Figure 3, during which every process of one node is exchanging data with every process of another node, even though all processes per node hold identical values.

Duplicate inter-node communication can be removed through a node-aware SMP allreduce, displayed in Figure 4, which reduces all intra-node data to a master process, performs a standard allreduce among master processes, and then broadcasts results locally [13]. While the SMP approach requires the same number and size of inter-node messages as recursive-doubling, only a single process communicates from each node, eliminating injection bandwidth limits. However, this approach yields a large number of inactive processes and load imbalance among processes on each node.

### B. Related Work

Node-aware optimizations have been added to other collective algorithms [13], [14], with intra-node shared memory optimizations [15], [16]. Similarly, node-awareness yields improvement to unstructured MPI communication, such as that which occurs during sparse matrix-vector multiplication [17]. Furthermore, the order on which processes are mapped to each node can have a large effect on collective performance [18], [19].

Collective communication can be further improved through topology-awareness, reducing network contention and limiting message distance [20]–[24]. Collectives over large amounts of data can be further optimized for specific topologies [25], [26]. Furthermore, as optimal algorithms depend on both message sizes as well as architectural topology, autotuners can determine the best algorithm for various scenarios [27], [28]. Finally, collective algorithms can be optimized for accelerated topologies, such as those containing Xeon Phi's [29] and GPU's [30]–[33].

## III. Node-Aware Parallel Allreduce (NAPAllreduce)

The `MPI_Allreduce` is commonly used for reductions on a small number of values per process, such as calculating an inner product of two vectors or a norm. When reducing over a small set of values, the cost of the associated allreduce is dominated by the maximum number of messages communicated by any process. Furthermore, the cost of each message is dependent on the relative locations of the sending and receiving processes. Figure 5 displays the modeled cost of sending a single message containing of various sizes on Blue Waters, a Cray supercomputer at the National Center for Supercomputing Applications [2], [3]. The costs were calculated with the max-rate model using parameters measured through ping-pong tests [11], [12]. Intra-socket messages, transported through cache, are significantly cheaper than inter-socket messages, which are transferred through shared memory. Furthermore, inter-node communication is notably more expensive than intra-node.

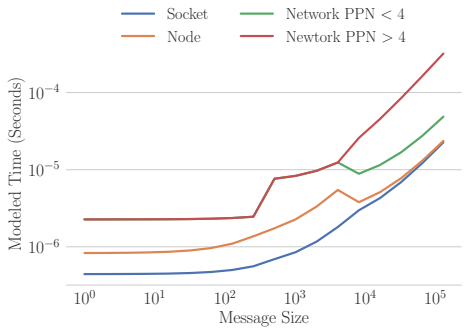


Fig. 5. Modeled cost of communicating a single message between two processes on Blue Waters. The costs are split into intra-socket ("Socket"), intra-node ("Node") and inter-node ("Network"). Inter-node communication costs are further split by `ppn` due to injection bandwidth limits. These costs are calculated with the max-rate model.

Recursive-doubling requires each process on a node to communicate duplicate data at every inter-node step, yielding $\log_2(n)$ inter-node messages per process, where $n$ is the number of nodes. The existing node-aware SMP algorithm improves the cost of relatively large reductions by removing duplicate messages between nodes, improving bandwidth costs. However, the majority of processes remain idle as a single process per node performs all inter-node allreduce operations, requiring each master process to communication $\log_2(n)$ inter-node messages. As a result, the maximum number of inter-node messages sent by a single process remains

**Algorithm 1:** NAP: `allreduce_NAP`

**Input:** data {Data to be reduced}
count {Size of data}
datatype {MPI Datatype}
MPI_Op {Reduction operation}
comm, rank, num_procs {MPI Communicator for Allreduce}
local_comm, local_rank, ppn {Intra-node communicator}

**Output:** reduced_data {Reduction of data over all processes in comm}

```
MPI_Allreduce(data, reduced_data, count, datatype, MPI_Op, local_comm)
```

prev_pos = 0
subgroup_size = ppn
group_size = subgroup_size · ppn
**for** $i$ = *0 to* $\log_{ppn}(\frac{num\_procs}{ppn})$ **do**
  group_start = $\lfloor \frac{rank}{group\_size} \rfloor \cdot$ group_size
  subgroup = $\frac{rank \cdot group\_start}{subgroup\_size}$
  proc = group_start + prev_pos + local_rank · subgroup_size + subgroup
  **if** $rank < proc$
    MPI_Send(data, count, datatype, dest, tag, comm)
    MPI_Recv(reduced_data, count, datatype, dest, tag, comm, recv_status)
  **else**
    MPI_Recv(reduced_data, count, datatype, dest, tag, comm, recv_status)
    MPI_Send(data, count, datatype, dest, tag, comm)
  `MPI_Op(reduced_data)`
  prev_pos = prev_pos + subgroup · subgroup_size
  subgroup_size = group_size
  group_size = group_size · ppn
  `MPI_Allreduce(data, reduced_data, count, datatype, MPI_Op, local_comm)`

equivalent to recursive-doubling. The remainder of this section introduces a node-aware allreduce algorithm, optimized for small reduction sizes, minimizing the maximum number of inter-node messages communicated by any process.

The SMP algorithm can be altered to use all ppn processes per node, splitting the required inter-node messages across all processes per node. The node-aware parallel (NAP) method, exemplified in Figure 6, consists of performing an intra-node allreduce so that all ppn processes hold a node's current reduction. Each process local to a node exchanges data with a specific node, before reducing results locally, as displayed in Step 3 of Figure 6. Therefore, the data from ppn nodes is reduced after a single step of inter-node communication, reducing the maximum number of inter-node messages to $\log_{\texttt{ppn}}(n)$. For example, a reduction over 16 nodes with 16 processes per node requires only a single inter-node step. Similarly, a NAP allreduce among 4096 nodes, with 16 processes each, requires only three inter-node steps.

The NAP allreduce algorithm is described in detail in Algorithm 1. At each inter-node step of this method, the number of nodes holding duplicate partial results increases by a power of ppn. Initially, only processes on a single node hold equivalent reduction results. However, at the beginning of the second inter-node step, all processes in a subgroup of ppn nodes hold equivalent data. In general, at the start of the $i^{th}$ step, processes in each subgroup of $\texttt{ppn}^{i-1}$ nodes hold the same partial results. Furthermore, a reduction is performed among groups of size $\texttt{ppn}^i$ at step $i$. These groups and subgroups are exemplified in Figure 7, in which the second inter-node step of a NAP allreduce with 4 processes per node is displayed. In the first step, each subgroup contains a single node and data is reduced over a row of nodes. During the second step, each row of nodes forms a subgroup, with these subgroups outlined in color, and data is reduced among all 4 subgroups.

Assuming SMP-style rank ordering, a process $q$ on node $m$ has local rank $r$, such that $q = n \cdot \texttt{ppn} + r$. For example, process $P9$ in Figure 6 is located on node 2 and has local rank 1. During each step of inter-node communication, process $q$ with local rank $r$ in subgroup $m$ communicates with process $u$ with local rank $m$ in subgroup $r$. Therefore, process $P9$ from Figure 6 exchanges data with $P6$, which is located on node 1 and has a local rank of 2. Note that any process with local rank equal to subgroup sits idly.

During later steps of communication, there are multiple processes with local rank $r$ in subgroup $m$. Therefore, the node position, or the index of a rank's node within the subgroup, must remain constant. In the second step of communication,

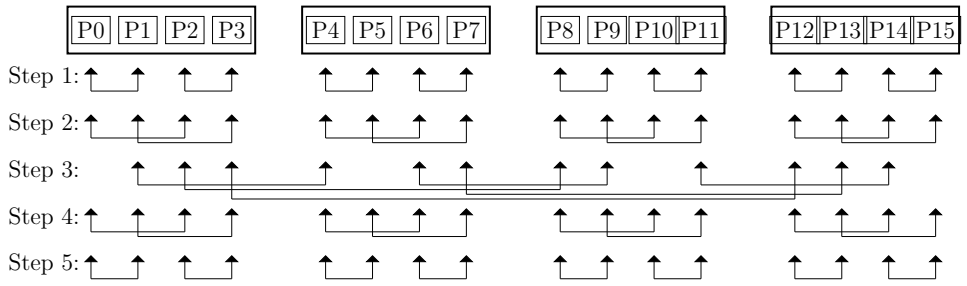


Fig. 6. Communication pattern for the NAP allreduce method. An intra-node allreduce is displayed in steps 1 and 2, while the single inter-node step is displayed in step 3. Steps 4 and 5 consist of the final intra-node allreduce. Note, one process per node sits idle during inter-node communication.

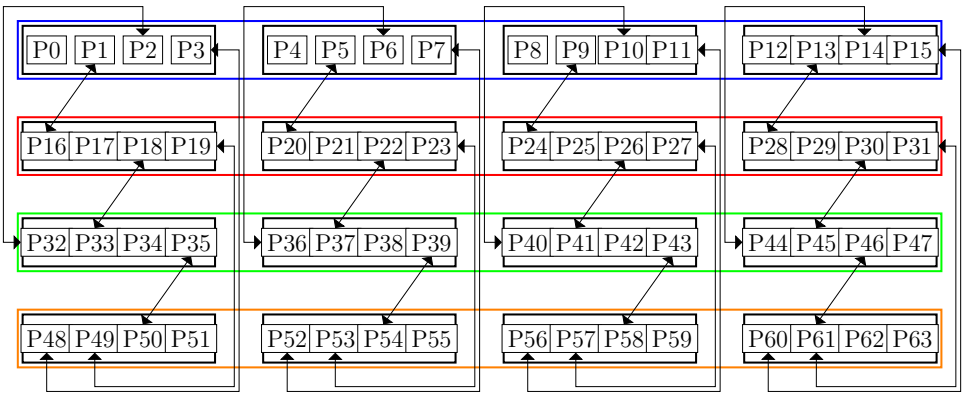


Fig. 7. The second inter-node step of a NAP allreduce over 16 nodes with 4 processes per node. Each row of nodes forms a subgroup, with all processes in a row containing equivalent data at the start of the step.

displayed in Figure 7, process $P9$ with local rank 1 in subgroup 0 has node position 2 as it lies on the third node in subgroup 0. Therefore, $P9$ exchanges data with $P24$ as this process has local rank 0 in subgroup 1 and also has node position 2.

### A. Non-Power of ppn Processes

The NAP allreduce algorithm reduces values among $p$ processes with only $\log_{\texttt{ppn}}(n)$ steps of inter-node communication. However, this algorithm requires that the number of processes is a power of ppn, limiting process counts for which this algorithm is viable. Assuming the number of nodes evenly divides ppn, the final step of inter-node communication can be reduced to involve only the necessary number of processes per node. Figure 8 displays the final step of a NAP allreduce with 12 nodes and 4 processes per node. All processes with local rank 3 sit idly during the final step of inter-node communication as there are no available nodes with which to communicate. However, the idle ranks recover the final result during the following intra-node allreduce.

The NAP allreduce can also be extended to node counts that are not divisible by ppn. In this case, subgroup sizes will not be equivalent during the final step of inter-node communication, as displayed in Figure 9. Subgroups with extra nodes will have no corresponding process with which to reduce data, meaning some nodes will not achieve the full reduction. However, as one process per node is idle during each step of inter-node communication, specifically the process with local rank equivalent to subgroup, each node has the potential to communicate with an extra node at each step. Therefore, the processes on extra nodes that have no corresponding process with which to exchange will instead send data to the idle process. Note this process does not need to receive data from the corresponding subgroup. As an example, process $P14$ receives data from $P34$ during the final step of inter-node communication, as a corresponding node in subgroup 2 does not exist.

## IV. Node-Aware Performance Modeling

Standard allreduce algorithms, such as recursive-doubling, minimize communication costs based on the standard postal model

$$T = \alpha t + \beta s + \gamma c, \tag{1}$$

where $\alpha$ is the per-message start-up cost, $\beta$ is the per-byte transport cost, $\gamma$ is the flop rate, and $t$, $s$, and $c$ are the number of messages, bytes, and floating-point operations, respectively. However, the cost of communication varies greatly with intra-node communication requiring significantly less cost than inter-node. Therefore, the performance model can better capture cost by splitting up intra- and inter-node costs [11], yielding

$$T = \alpha_\ell t_\ell + \beta_\ell s_\ell + \alpha t + \beta s + \gamma c \tag{2}$$

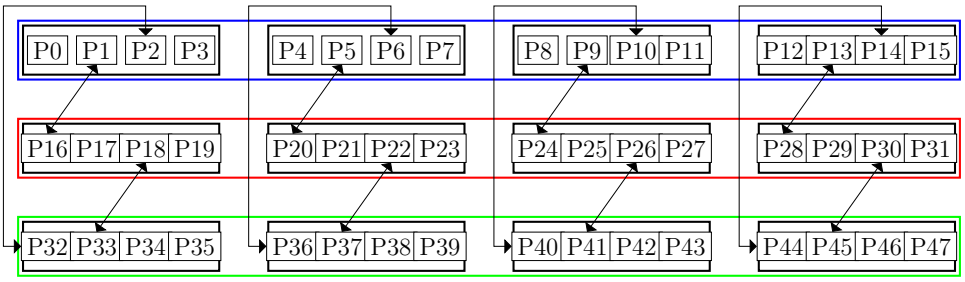


Fig. 8. Communication pattern for a NAP allreduce with a non power of ppn process count. As the number of nodes is divisible by ppn, the first step proceeds as normal, while the second step reduces over 3 subgroups. An extra process per node remains inactive for only the second step.

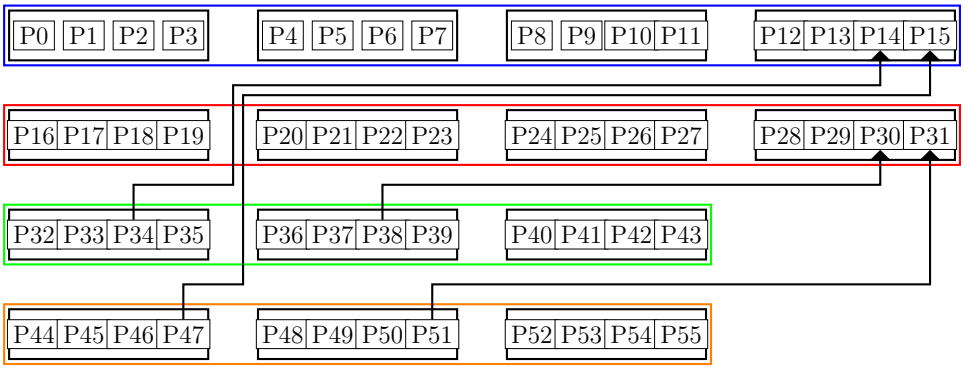


Fig. 9. The communication pattern for a NAP allreduce with a number of nodes that does not divide ppn. The initial step reduces over groups of nearly equal size. The second inter-node step reduces as normal if the corresponding process exists, and otherwise receives data from the corresponding idle process.

where $\alpha_\ell$, $\beta_\ell$, $t_\ell$, and $s_\ell$ all represent intra-node communication while the remaining variables model inter-node communication and local computation. Finally, inter-node bandwidth is greatly dependent on the number of processes communicating per node as injection bandwidth limits slow transport of large messages. Therefore, this model is further improved by incorporating the max-rate model [12]

$$T = \alpha_\ell t_\ell + \beta_\ell s_\ell + \alpha t + \frac{\texttt{ppn} \cdot s}{\min\left(R_N, \texttt{ppn} \cdot R_b\right)} + \gamma c \quad (3)$$

where $R_b$ in inter-process bandwidth, or the inverse of $\beta$, and $R_N$ is injection bandwidth. Note, this reduces to Equation 2 when inter-process bandwidth is achieved.

Performance costs of the various allreduce algorithms for small messages can be analyzed through the improved performance model in Equation 3. The performance model cost of an allreduce of size $s$ over $p$ processes with recursive-doubling is displayed in Equation 4.

$$\begin{aligned}&(\alpha_\ell + \beta_\ell s) \cdot (\log_2(\texttt{ppn})) \\ &+ \left(\alpha + \frac{\texttt{ppn} \cdot s}{\min(R_N, \texttt{ppn} \cdot R_b)}\right) \cdot (\log_2(n)) + \gamma s \cdot (\log_2(p))\end{aligned} \quad (4)$$

Recursive-doubling requires $\log_2(n)$ inter-node messages of size $s$, with injection bandwidth limiting performance for large values of $s$.

The SMP allreduce improves upon this cost model, with the associated performance model cost of the SMP algorithm displayed in Equation 5.

$$\begin{aligned}&(\alpha_\ell + \beta_\ell s) \cdot (\log_2(\texttt{ppn})) \\ &\quad + \left(\alpha + \frac{s}{R_b}\right) \cdot (\log_2(n)) + \gamma s \cdot (\log_2(p))\end{aligned} \quad (5)$$

While the SMP method yields equivalent inter-node communication requirements to recursive-doubling, inter-node messages of all sizes achieve inter-process bandwidth as only a single process per node performs inter-node communication at any time. The SMP approach does require slightly more intra-node communication than recursive-doubling due to the reduction and broadcast local to each node.

Finally, the NAP allreduce algorithm minimize inter-node communication requirements, exchanging inter-node messages for additional intra-node communication and local computation, as displayed in Equation 6.

$$\begin{aligned}&(\alpha_\ell + \beta_\ell s) \cdot (\log_2(p)) \\ &\quad + \left(\alpha + \frac{\texttt{ppn} \cdot s}{\min(R_N, \texttt{ppn} \cdot R_b)}\right) \cdot \left(\log_{\texttt{ppn}}(n)\right) \\ &\quad + \gamma s \cdot \left(\log_2(p) + \log_{\texttt{ppn}}(n)\right)\end{aligned} \quad (6)$$

The number of inter-node communication steps is reduced from $\log_2(n)$ to $\log_{\texttt{ppn}}(n)$. However, intra-node communication steps increase greatly from $\log_2(\texttt{ppn})$ to $\log_2(p)$ and and additional $\log_{\texttt{ppn}}(n)$ steps of local computation are required.

Furthermore, injection bandwidth will limit the rate at which bytes are transported for large messages as many processes per node are active in intra-node communication at each step. Therefore, the NAP allreduce is ideal for small reduction sizes across a large number of processes, where extra computation and bandwidth injection limits are not a factor.

Figure 10 shows the performance model costs for the recursive-doubling (RD), SMP, and NAP allreduce methods when reducing a single value across various process counts. The model parameters were measured for Blue Waters with ping-pong tests and the STREAM benchmark [34], [35]. The

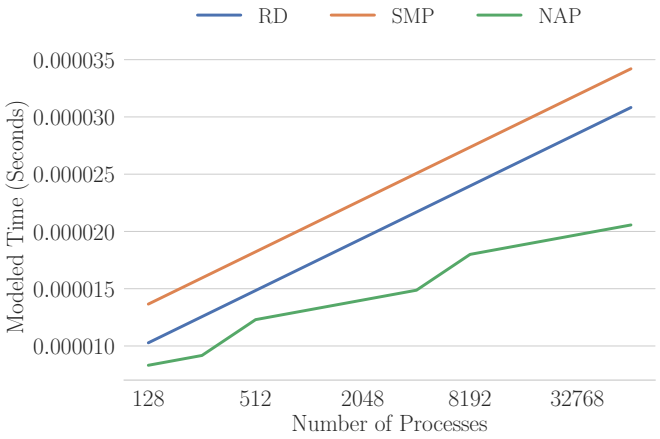


Fig. 10. The modeled allreduce cost for reducing a single value across various process counts with the recursive-doubling (RD), SMP, and NAP methods.

performance models indicate that the NAP allreduce outperforms the other methods for small message sizes, particularly as process count increases. Furthermore, Figure 11 displays the performance model costs for performing an allreduce with each method using $32,768$ processes, indicating the NAP allreduce outperforms recursive-doubling and SMP methods for small message sizes, while the SMP allreduce outperforms the recursive-doubling and NAP methods for large message sizes.

## V. Results

The recursive-doubling, SMP, and NAP allreduce algorithms were implemented on top of CrayMPI, utilizing the `MPI_Send` and `MPI_Recv` methods for each exchange of data. Due to the associated overhead, results are presented for these implementations rather than comparing with recursive-doubling and SMP implementations that exist in MPICH. All tests were performed on Blue Waters with 16 processes per node. Furthermore, each timing was calculated by performing thousands of allreduce operations to reduce error from timer precision, and each of these tests was performed 5 times on different partitions of Blue Waters. Each plot contains lines displaying the average results over the 5 separate runs and outlines show the variation in timings over these 5 tests.

Figure 12 displays the cost of the recursive-doubling (RD), SMP, and NAP methods for reducing a single value on each

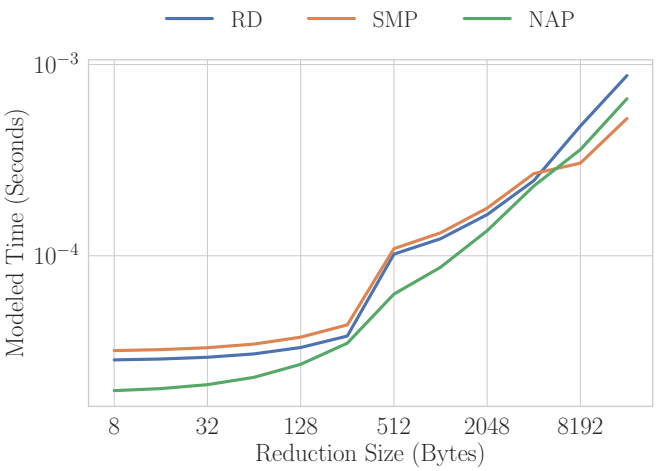


Fig. 11. The modeled cost of performing an allreduce of various reduction sizes with each method on 32 768 processes.

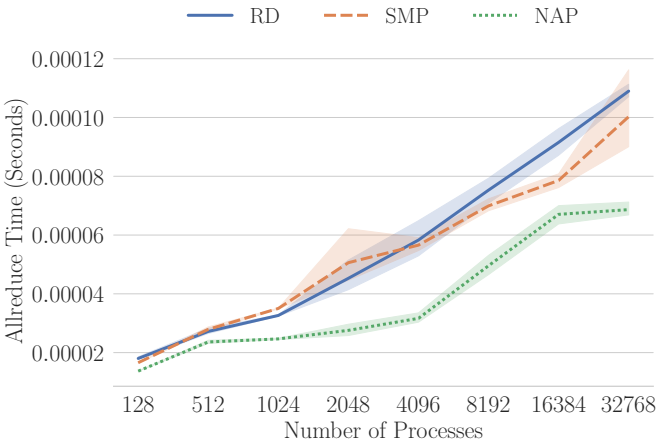


Fig. 12. The measured cost of performing an allreduce of a single value with the recursive-doubling (RD), SMP, and NAP methods.

process for various process counts. Furthermore, Figure 13 shows the associated speedups obtained with the NAP method. The NAP allreduce algorithm obtains notable speedups over the other methods, particularly at process counts that are a power of `ppn`.

Figures 14 and 15 show the costs and speedups, respectively, for performing the various allreduce methods on 32 768 processes for a variety of reduction sizes. The NAP method yields significant speedups over the recursive-doubling and SMP methods for smaller message sizes. However, the SMP approach outperforms the NAP method for reduction sizes over 2048 bytes, similar to expected performance based on the models in Figure 10.

## VI. Conclusions and Future Work

The NAP allreduce method yields notable improvements over standard recursive-doubling and existing node-aware SMP methods, in both performance models and measured costs for small message sizes, of up to 2048 bytes. The NAP

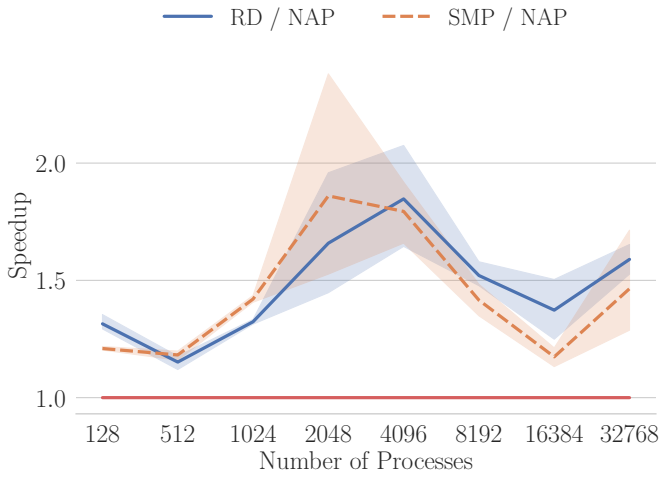


Fig. 13. Speedup acquired from the NAP allreduce over the recursive-doubling and SMP methods when reducing a single value.

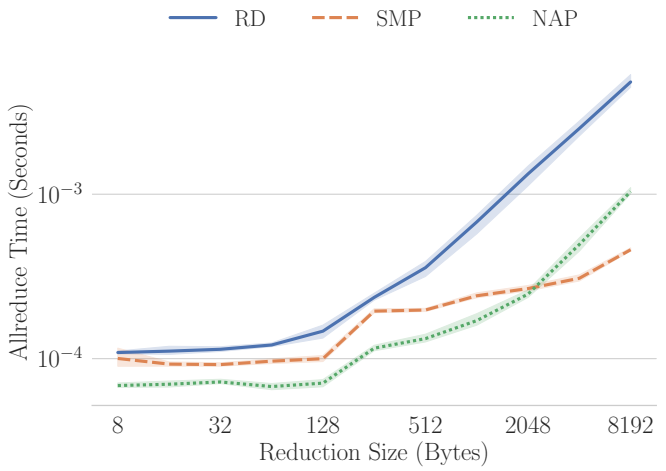


Fig. 14. The cost of reducing various numbers of values over 32 768 processes with the recursive-doubling, SMP, and NAP allreduce methods.

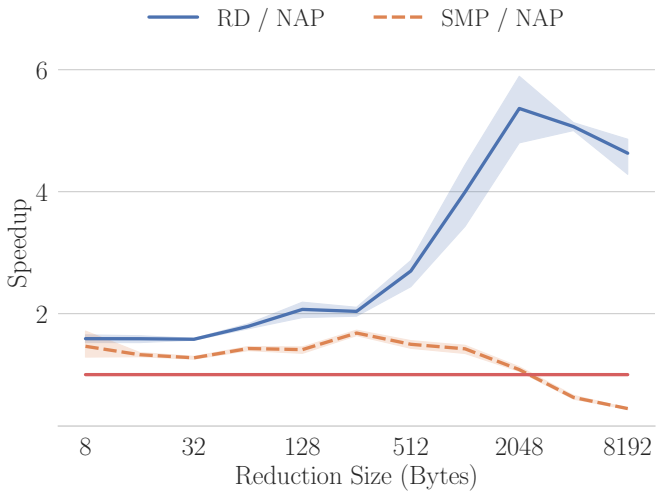


Fig. 15. The speedup in the NAP allreduce algorithm over the recursive-doubling and SMP methods for reducing various numbers of values on 32 768 processes. The NAP allreduce yields improved performance up to a reduction size of 2048 bytes.

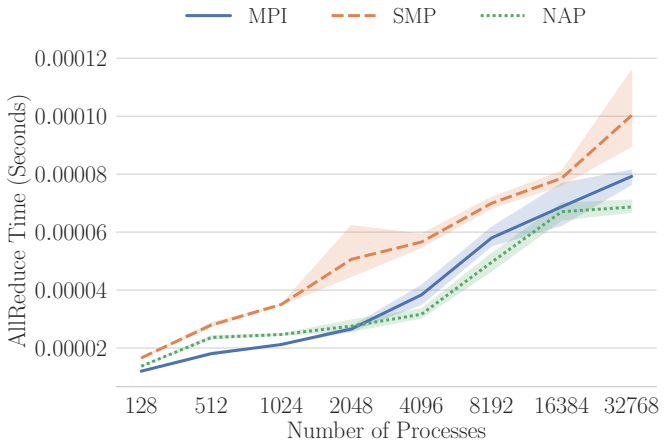


Fig. 16. The cost of reducing various numbers of values over 32 768 processes with the SMP algorithm as implemented in MPICH (labeled MPI), SMP, and NAP allreduce methods.

algorithm relies on power of ppn process counts, but natural extensions allow for all other process counts. However, non power of ppn process counts require the same number of inter-node communication steps as the succeeding power of ppn. Therefore NAP allreduce speedups are most significant at power of ppn process counts.

This paper is focused on the cost of the recursive-doubling, SMP, and NAP allreduce methods when implemented on top of MPICH, calling MPI_Send and MPI_Recv for each step of communication. However, there is significant overhead associated with these calls, in comparison to direct implementation of these methods in MPICH. Figures 16 and 17 display the cost of performing the SMP and NAP allreduce methods on top of MPI, compared to the SMP method as implemented in MPICH, measured by calling the MPI_Allreduce routine. The overhead associated with implementing on top of MPICH can be seen as the difference between the MPI and SMP costs. While the node-aware allreduce yields slight improvements over MPICH's SMP approach, speedups are minimal due to the additional overhead. Therefore, this method should be implemented as a part of MPICH to achieve optimal performance.

Similar node-aware approaches can be extended to other collective algorithms. Natural extensions exist to the MPI_Allgather, in which a similar recursive-doubling algorithm performs well for small gather sizes. Furthermore, the node-agnostic ring algorithm again has multiple processes communicating duplicate data between nodes, which could be improved upon. Using the max-rate model as a guide, node-aware extensions could be applied to larger MPI_Allreduce methods, optimizing the reduce-scatter and allgather approach to avoid injection bandwidth limits while utilizing as many

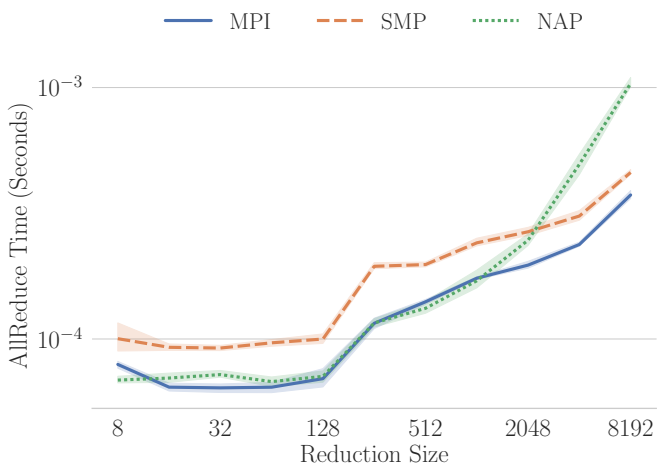


Fig. 17. The cost of reducing various numbers of values over 32 768 processes with the SMP algorithm as implemented in MPICH (labeled MPI), SMP, and NAP allreduce methods.

processes per node as possible.

Finally, locality-aware collective algorithms can be extended to other parts of the architecture, such as reducing inter-socket communication in exchange for increased intra-socket message counts. Similary, these algorithms can be optimized for heterogeneous architectures, in which many layers of memory and communication exist.

## References


[1] "MPI: A message-passing interface standard," Knoxville, TN, USA, Tech. Rep., 1994.

[2] NCSA, "Blue Waters," https://bluewaters.ncsa.illinois.edu/, 2012.

[3] B. Bode, M. Butler, T. Dunning, T. Hoefler, W. Kramer, W. Gropp, and W. Hwu, "The Blue Waters super-system for super-science," in *Contemporary High Performance Computing: From Petascale Toward Exascale*, 1st ed., ser. CRC Computational Science Series, J. S. Vetter, Ed. Boca Raton: Taylor and Francis, 2013, vol. 1, pp. 339–366. [Online]. Available: http://j.mp/RrBdPZ

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789 – 828, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167819196000245

[5] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005. [Online]. Available: http://dx.doi.org/10.1177/1094342005051521

[6] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: Theory, practice, and experience: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1002/cpe.v19:13

[7] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 65:1–65:43, Aug. 2019. [Online]. Available: http://doi.acm.org/10.1145/3320060

[8] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science - ICCS 2004*, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–9.

[9] M. Ruefenacht, M. Bull, and S. Booth, "Generalisation of recursive doubling for allreduce," in *Proceedings of the 23rd European MPI Users' Group Meeting*, ser. EuroMPI 2016. New York, NY, USA: ACM, 2016, pp. 23–31. [Online]. Available: http://doi.acm.org/10.1145/2966884.2966913

[10] J. Worringen, "Pipelining and overlapping for MPI collective operations," in *28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN '03. Proceedings.*, Oct 2003, pp. 548–557.

[11] A. Bienz, W. D. Gropp, and L. N. Olson, "Improving performance models for irregular point-to-point communication," in *Proceedings of the 25th European MPI Users' Group Meeting, Barcelona, Spain, September 23-26, 2018*, 2018, pp. 7:1–7:8. [Online]. Available: https://doi.org/10.1145/3236367.3236368

[12] W. Gropp, L. N. Olson, and P. Samfass, "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*, ser. EuroMPI 2016. New York, NY, USA: ACM, 2016, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/2966884.2966919

[13] H. Zhu, D. Goodell, W. Gropp, and R. Thakur, "Hierarchical collectives in MPICH2," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, M. Ropo, J. Westerholm, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–326.

[14] J. L. Träff and A. Rougier, "MPI collectives and datatypes for hierarchical all-to-all communication," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: ACM, 2014, pp. 27:27–27:32. [Online]. Available: http://doi.acm.org/10.1145/2642769.2642770

[15] S. Jain, R. Kaleem, M. G. Balmana, A. Langer, D. Durnov, A. Sannikov, and M. Garzaran, "Framework for scalable intra-node collective operations using shared memory," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 29:1–29:12. [Online]. Available: https://doi.org/10.1109/SC.2018.00032

[16] R. L. Graham and G. Shipman, "MPI support for multi-core architectures: Optimized shared memory collectives," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, A. Lastovetsky, T. Kechadi, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 130–140.

[17] A. Bienz, W. D. Gropp, and L. N. Olson, "Node aware sparse matrix-vector multiplication," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 166 – 178, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731519302321

[18] T. Ma, T. Herault, G. Bosilca, and J. J. Dongarra, "Process distance-aware adaptive MPI collective communications," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 196–204.

[19] J. Zhang, J. Zhai, W. Chen, and W. Zheng, "Process mapping for MPI collective communications," in *Euro-Par 2009 Parallel Processing*, H. Sips, D. Epema, and H.-X. Lin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 81–92.

[20] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci, M. Schulz, and C. H. Still, "Mapping applications with collectives over sub-communicators on torus networks," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 97:1–97:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389128

[21] P. Sack and W. Gropp, "Faster topology-aware collective algorithms through non-minimal communication," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '12. New York, NY, USA: ACM, 2012, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/2145816.2145823

[22] T. Ma, G. Bosilca, A. Bouteiller, and J. J. Dongarra, "Kernel-assisted and topology-aware MPI collective communications on multicore/many-core platforms," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 1000 – 1010, 2013, best Papers: International Parallel and Distributed Processing Symposium (IPDPS) 2010, 2011 and 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731513000166

[23] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, "Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, April 2010, pp. 1–8.

[24] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *Proceedings 14th*

*International Parallel and Distributed Processing Symposium. IPDPS 2000*, May 2000, pp. 377–384.

[25] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117 – 124, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731508001767

[26] P. Patarasuk and X. Yuan, "Bandwidth efficient all-reduce operation on tree topologies," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–8.

[27] A. Faraj and X. Yuan, "Automatic generation and tuning of MPI collective communication routines," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ser. ICS '05. New York, NY, USA: ACM, 2005, pp. 393–402. [Online]. Available: http://doi.acm.org/10.1145/1088149.1088202

[28] A. Faraj, X. Yuan, and D. Lowenthal, "STAR-MPI: Self tuned adaptive routines for MPI collective operations," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/1183401.1183431

[29] K. Kandalla, A. Venkatesh, K. Hamidouche, S. Potluri, D. Bureddy, and D. K. Panda, "Designing optimized MPI broadcast and allreduce for many integrated core (MIC) infiniband clusters," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, Aug 2013, pp. 63–70.

[30] H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, and D. K. Panda, "MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters," *Computer Science - Research and Development*, vol. 26, no. 3, p. 257, Apr 2011. [Online]. Available: https://doi.org/10.1007/s00450-011-0171-3

[31] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters with NVIDIA GPUs," in *2013 42nd International Conference on Parallel Processing*, Oct 2013, pp. 80–89.

[32] L. Oden, B. Klenk, and H. Frning, "Energy-efficient collective reduce and allreduce operations on distributed GPUs," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2014, pp. 483–492.

[33] I. Faraji and A. Afsahi, "GPU-aware intranode MPI_Allreduce," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: ACM, 2014, pp. 45:45–45:50. [Online]. Available: http://doi.acm.org/10.1145/2642769.2642773

[34] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[35] J. D. McCalpin, "STREAM: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/. [Online]. Available: http://www.cs.virginia.edu/stream/