Collective communication on the Schooner supercomputer

Sam Bird
University of Oklahoma
Norman, OK, USA

Abstract—**Several algorithms may be used to implement a given collective communication operation, with differing performance based on the size of the data being sent, communication hardware on the physical computers, and network topologies. Selecting the right algorithm is particularly important in high-performance computing, where the distributed nature of the computation makes message passing both an essential component and potential bottleneck.**

Open MPI's built-in collective functions dynamically select which algorithm to use on the basis of the size of the message, when benchmarked against existing clusters. However, this approach cannot guarantee that Open MPI's benchmarks are applicable to any given compute cluster. I benchmark Open MPI's algorithms against those optimised for short and long vector length messages. I find that, while Open MPI's built-in selected algorithm is frequently a good choice for any given collective and vector length, different collective implementations outperform the default under certain parameters on the Schooner supercomputer.

I. Introduction

Collective communication operations are foundational to the practice of high-performance computing, particularly in distributed memory computer architectures common to supercomputers like Schooner at the University of Oklahoma. A common implementation of collective communication operations is the Open MPI library, which implements the Message Passing Interface. In their 2007 paper, Chan et al [1] provide a comprehensive, systematic review of seven common collective operations: broadcast, reduce, scatter, gather, allgather, reduce-scatter, and allreduce.

These collectives underpin the message-passing model of distributed computation, and their implementation is nontrivial. Each collective operation executed will incur some amount of latency and use some amount of bandwidth, and the amount of these used varies not only with the amount of data being sent and number of destinations to which it is sent, but also with the implementation of the collective operation being used to send it.

In an attempt to minimise both latency and bandwidth, the Open MPI library uses a strategy – *fixed decision* mode – to determine which algorithm is the best to use for a given collective operation, given the data size [2]. However, the decision parameters for the algorithm selection cannot account for every compute cluster or network topology that might be encountered.

Chan et al provide two algorithms for each collective communication operation, optimised for short and long data vector lengths respectively. I benchmarked these algorithms, as well as Open MPI's selected algorithm, across varying data sizes for each collective.

II. Background

Chan provides a comparison of lower bounds of each collective operation, as well a discussion of the difference in effective cost when sending short vectors of data in messages compared to long vectors. They also briefly discuss different topologies that may be used to conceptualise a network of distributed processors. While these are useful discussions, they run up against the realities of real-world execution on a supercomputer. First, messages sent between different physical nodes are sent through switches, meaning that the physical network is neither linear, nor a torus, nor fully connected, though with the speed of the switches we may assume that it is fully connected. Second, while implementations such as Open MPI could analyse the size of the data being sent in a collective operation and strategically choose an algorithm to use, in practice the question of what the best algorithm to use for a given operation will vary from system to system, and indeed within each system as resource availability changes.

The question of selecting an optimal strategy for collective operations is not a novel one: since Chan's paper renewed interest in rigorous examination of collective algorithms, numerous articles have proposed different communication models for improved collective performance, with variations both in the success of those models, but crucially also with various benchmarks for success. Rico-Gallego et al (2019) [3] undertook a meta-analysis of those models, noting that most attempt to achieve two objectives: accurate prediction of communication time and to serve as a reliable guide for implementation of those models. Cai and Liu et al (2021) [4] demonstrate a method for synthesising optimal models given the parameters of network topology and the criterion to optimise; that is to say, latency-optimal or bandwidth-optimal communication.

To better understand the performance of different collective models, I implemented the seven collective communication operations in three ways: the short and long vector algorithms described by Chan, and the built-in functions provided by Open MPI.

III. Theory

A. Broadcast, scatter, gather, etc.

Chan models the cost of each collective in terms of its latency, bandwidth, and computation cost. They use the following terminology:

- n is the length of the data – that is to say, we are sending a vector of n elements.
- p is the number of nodes in our network.
- α is the cost of setting up a message, and β is the cost of sending a single data item. Consequently, we say that the cost of sending a message is $\alpha + n\beta$.
- γ is the cost required to perform an arithmetic operation. For example, in a reduction, this is the cost to add one item to the sum or average being computed.

Chan establishes lower bounds on each collective operation, based on the assumption that the network is fully connected, bidirectional (that is, messages travelling opposite directions in the network will not block each other), and that each node may send to and receive from at most one other node during a given timestep. I use the same assumptions for my analysis throughout this report.

Collective operation	**Lower performance bound**
Broadcast	$\lceil \log p \rceil \alpha + n\beta$
Reduce	$\lceil \log p \rceil \alpha + n\beta + \frac{p-1}{p} n\gamma$
Scatter	$\lceil \log p \rceil \alpha + \frac{p-1}{p} n\beta$
Gather	$\lceil \log p \rceil \alpha + \frac{p-1}{p} n\beta$
Allgather	$\lceil \log p \rceil \alpha + \frac{p-1}{p} n\beta$
Reduce-scatter	$\lceil \log p \rceil \alpha + \frac{p-1}{p} n(\beta + \gamma)$
Allreduce	$\lceil \log p \rceil \alpha + \frac{p-1}{p} n(2\beta + \gamma)$

TABLE I
Lower bound of each collective operation (Chan 2019).

B. Rotation

Chan does not discuss a rotation collective operation, where the data is initially segmented across all nodes (similar to the state of the network before a gather operation), and each node rotates the data it holds. For example, the data in node 1 is sent to node 0, the data in node 2 is sent to node 1, and so on.

Before.

Node 0	Node 1	Node 2	Node 3
x_0			
	x_1		
		x_2	
			x_3

After.

Node 0	Node 1	Node 2	Node 3
			x_0
x_1			
	x_2		
		x_3	

The lower bound of the bandwidth cost of this operation is $\frac{n}{p}\beta$. n items are distributed across p nodes, so if all items are distributed evenly across the nodes then every node will send n/p items. Of course, each node must receive that number of items, but that bandwidth cost is accounted for by the sending node. Each item has a cost of β to send, which leaves us with this bandwidth lower bound.

The lower bound of the latency of this operation is a constant α, as every node will need to send exactly one message to rotate the data. As there is no additional computation, this yields a total lower performance bound of $\alpha + \frac{n}{p}\beta$ for this operation.

IV. Mechanism

A. Algorithm design

The implementation of each collective operation is built of one or more algorithms described by Chan, which I will call primitives. These may be divided into three classes of algorithm: simple, minimum spanning tree, and bucket.

The simple algorithms naïvely send or receive messages from the root node, looping through the list of peers to send or receive a message. These algorithms are used in the long vector versions of the scatter and gather algorithms. An example of the simple scatter algorithm is given below:

```
smpl_scatter(x, our_id, root_id, num_nodes)

if our_id == root_id:
    for i in 0 to num_nodes:
        if i != our_id:
            MPI_Send(x, i)
else:
    MPI_Recv(x, root_id)
```

Chan observes that these algorithms have a cost of $(p-1)\alpha + \frac{p-1}{p} n\beta$, worse than the minimum spanning tree algorithm, but finds that it may be superior in practice because the β term may be smaller than expected due to the cost overlapping with other messages. For vectors of length $\leq 2^{27}$, I find this to be the case for scatter operations, but not gather (see sections V-C and V-D).

The minimum spanning tree algorithms partition the set of nodes into two subsets, and the root node selects a destination node in the middle of each subset. That destination node then itself becomes a root node, partitions its subset, and selects new destination nodes. This process repeats recursively until all nodes have sent or received the data. Figure 4.1.2 illustrates the sending of data across a minimum spanning tree.

MST algorithms, while used in both short and long vector implementations of several collectives, tend to be used in the short vector versions because they achieve a latency lower bound of $\lceil \log p \rceil \alpha$, while the simple algorithms incur $(p-1)\alpha$ and bucket algorithms incur $p\alpha$ latency costs.

Bucket algorithms conceive of the network as a ring and pass data around the ring. This is particularly effective in an operation like allgather, where the data is initially scattered across every node. Each node can send its subvector to one neighbour and receive from the other, then send along the newly received data on the next timestep.

These algorithms tend to form the basis for the long vector algorithms because of the reduced bandwidth cost. An

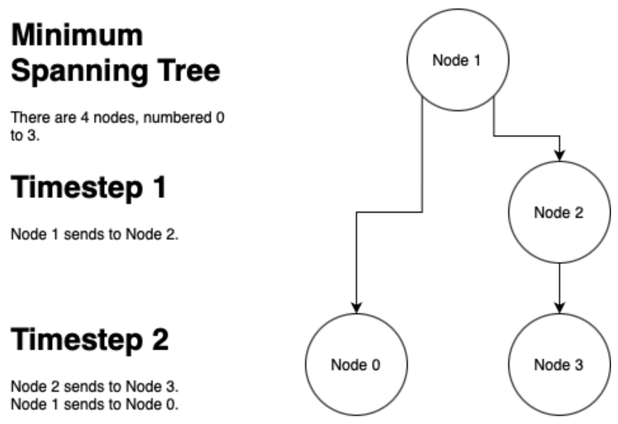

Fig. 4.1.1. Minimum spanning tree commmunication.

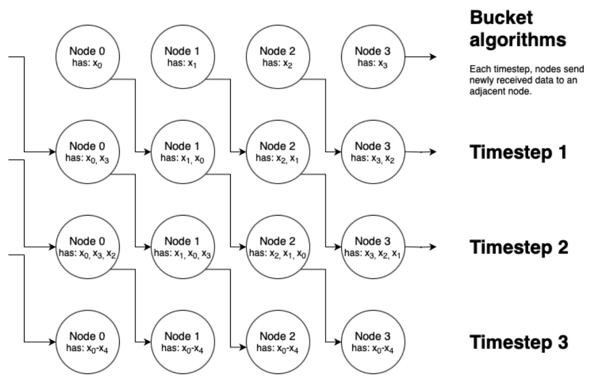

Fig. 4.1.2. Bucket commmunication. Observe the data is shared in $p - 1$ timesteps.

MST algorithm, sending data from a root node, will incur a $\lceil \log p \rceil n\beta$ cost, while a bucket algorithm will only incur a cost of $\frac{p-1}{p} n\beta$, at the expense of sending more messages.

Chan develops the short and long vector versions of each collective operation from these primitives. The construction of each collective is described in table II.

B. Collective timing

For each collective communication operation, I benchmarked Chan's algorithms for short and long vectors, as well as the Open MPI built-in collective (which I refer to as *short*, *long*, and *builtin*, respectively) on OU's Schooner supercomputer. I timed the execution of each algorithm 30 times on 8 different lengths of vectors being sent, ranging from 2^{20} elements to 2^{27} elements. Each run had 32 virtual processors, but in order to examine the effect of different physical topologies, the runs were done over 2, 4, and 8 physical compute nodes.

I created a new run for each combination of collective operation, algorithm type (short, long, or builtin), and number of physical nodes (2, 4, or 8). Each combination was run 30 times, and each run iterated over the data sizes to benchmark.

Collective operation	Short vector algorithm	Long vector algorithm
Broadcast	MST Broadcast	MST Scatter, Bucket Allgather
Reduce	MST Reduce	Bucket Reduce-scatter, MST Gather
Scatter	MST Scatter	Simple Scatter
Gather	MST Gather	Simple Gather
Allgather	MST Gather, MST Broadcast	Bucket Allgather
Reduce-scatter	MST Reduce, MST Scatter	Bucket Reduce-scatter
Allreduce	MST Reduce, MST Broadcast	Bucket Reduce-scatter, Bucket Allgather

TABLE II
ALGORITHMS USED IN THE IMPLEMENTATION OF COLLECTIVE OPERATIONS.

The timing for each data size within each run was collected using Open MPI's `MPI_Wtime` function. Only the time spent executing the collective operation was measured; the time to allocate or free memory between operations was not counted.

```
for (data_size in 2^20 to 2^27) {
    input_arr = (float*) calloc(data_size,
                                sizeof(float));
    output_arr = (float*) calloc(data_size,
                                 sizeof(float));
    setup_collective(input_arr, data_size,
                          my_rank, num_procs);
    MPI_Barrier(MPI_COMM_WORLD);
    double tic = MPI_Wtime();
    int opr = OPERATION(input_arr, output_arr,
                        data_size, my_rank,
                        num_procs);
    double toc = MPI_Wtime();
    timings[i] = toc - tic;
}
```

V. ANALYSIS

For each collective communication operation, I compare the performance of each algorithm on 2, 4, and 8 physical nodes, though each run had 32 processes. The vector size is the number of elements used in each operation, where each element is a C `float`. On the Schooner supercomputer, floats are 4 bytes large. The time elapsed for each operation is shown, measured in seconds.

A. Broadcast

The broadcast operation distributes data from one source node to all other nodes. The short vector algorithm is a primitive, the minimum spanning tree broadcast, while the long vector algorithm is a minimum spanning tree scatter followed by a bucket allgather.

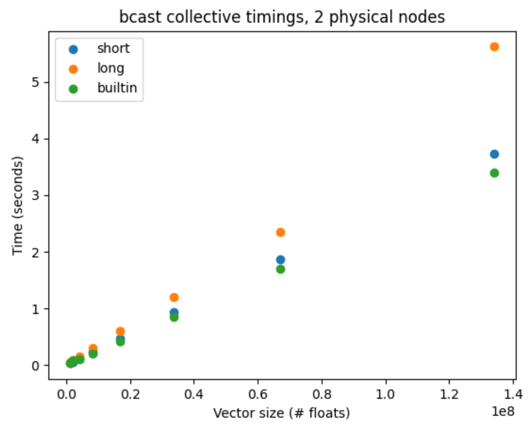

Fig. 5.1.1. Broadcast collective, 2 compute nodes.

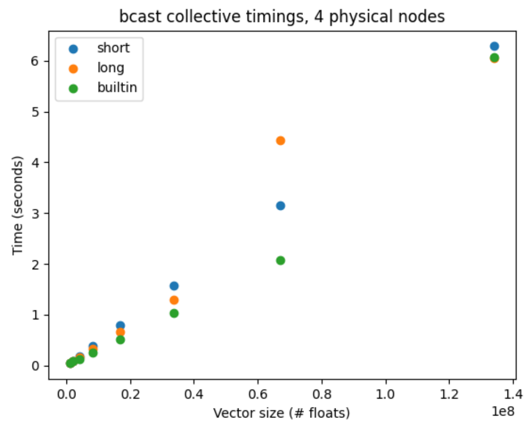

Fig. 5.1.2. Broadcast collective, 4 compute nodes.

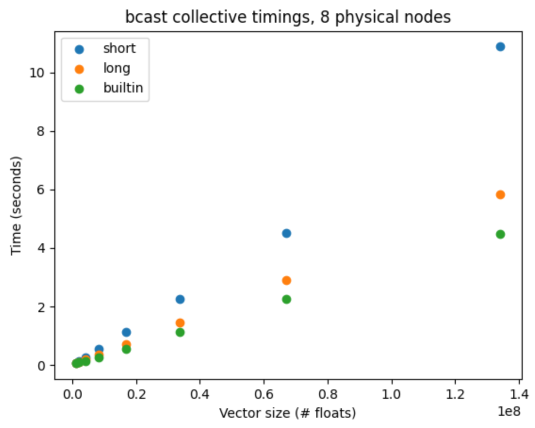

Fig. 5.1.3. Broadcast collective, 8 compute nodes.

Generally speaking, the builtin algorithm is the best performer for broadcast. On 2 physical nodes, the long vector algorithm performs worst, while on 8 physical nodes the short vector algorithm is the worst performer.

B. Reduce

The reduce operation collects data spread across the network to the root node and applies some operation to it, such as finding the sum or average of numerical data. In these experiments, the data was simply summed. Like broadcast, the short vector algorithm is a primitive – minimum spanning tree reduce. The long vector algorithm is a bucket reduce-scatter followed by a minimum spanning tree gather.

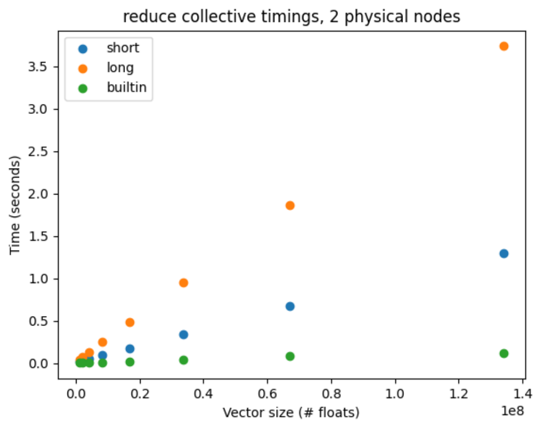

Fig. 5.2.1. Reduce collective, 2 compute nodes.

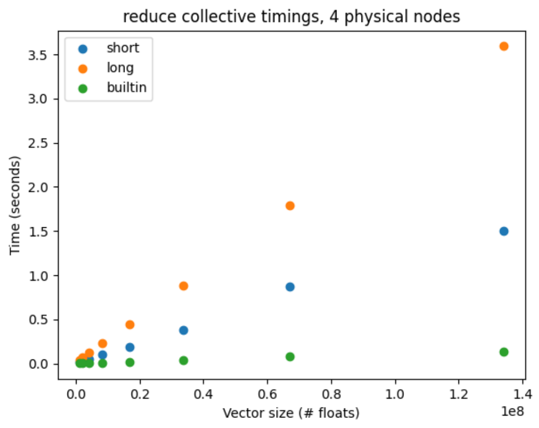

Fig. 5.2.2. Reduce collective, 4 compute nodes.

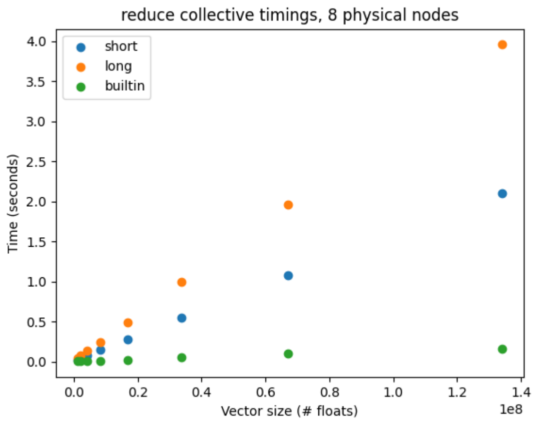

Fig. 5.2.3. Reduce collective, 8 compute nodes.

The builtin operation is the clear winner across every physical topology. The short vector algorithm consistently outperforms the long vector algorithm. Although the primitives used in the short and long vector algorithms are reused in several other collective operations, in no other case does the Open MPI builtin algorithm so dramatically outperform both the short and long vector algorithms.

C. Scatter

The scatter operation distributes one subset of a vector held at the root process to each other process in the network. The short vector algorithm is the minimum spanning tree scatter algorithm, while the long vector algorithm is the 'simple' scatter algorithm.

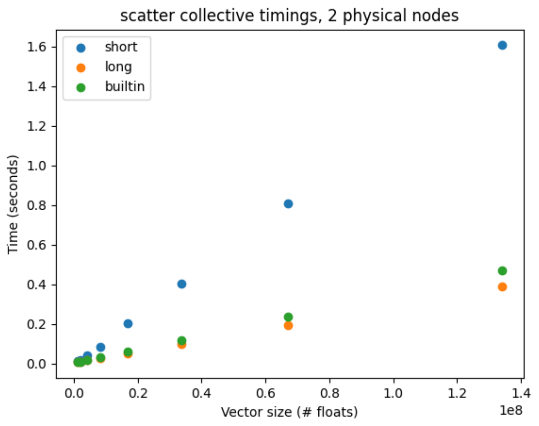

Fig. 5.3.1. Scatter collective, 2 compute nodes.

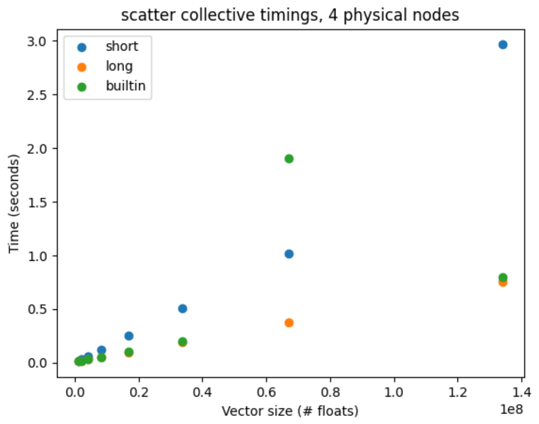

Fig. 5.3.2. Scatter collective, 4 compute nodes.

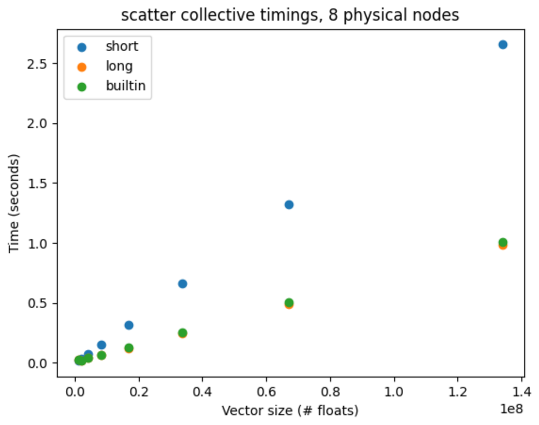

Fig. 5.3.3. Scatter collective, 8 compute nodes.

In this case, I replicate Chan's finding that the simple scatter algorithm outperforms the minimum spanning tree algorithm in practice, despite having a worse lower performance bound. This is true independent of the number of physical nodes used. With the exception of one outlier in the 4 physical nodes trial, the long vector algorithm had roughly equal performance to the builtin algorithm.

D. Gather

Gather is the scatter operation's inverse. Given sub-vectors of data scattered across processes in a network, the gather operation collects the data in the root process. The short vector algorithm is the minimum spanning tree algorithm; the long vector algorithm is the 'simple' gather algorithm.

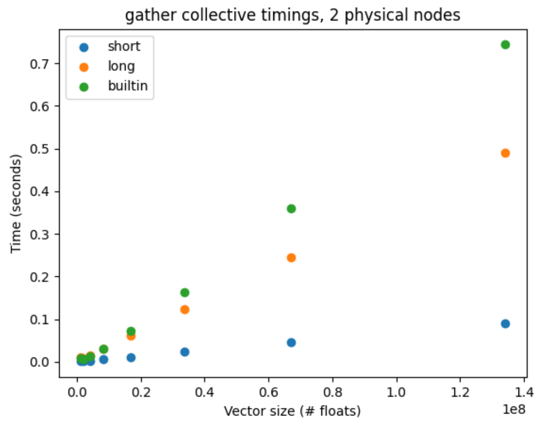

Fig. 5.4.1. Gather collective, 2 compute nodes.

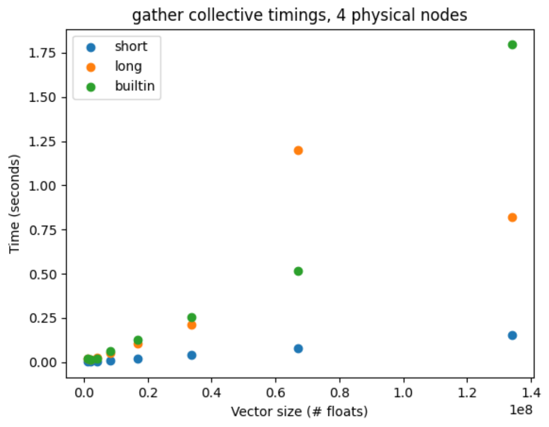

Fig. 5.4.2. Gather collective, 4 compute nodes.

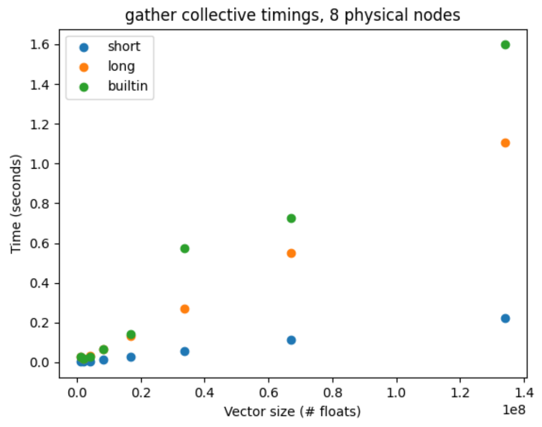

Fig. 5.4.3. Gather collective, 8 compute nodes.

Unlike the scatter operation, I did not observe the simple algorithm for gather outperforming the minimum spanning tree approach. In fact, the MST algorithm outperformed the simple algorithm and Open MPI's builtin gather operation significantly, regardless of the number of physical nodes used.

E. Allgather

Similarly to the gather operation, allgather collects data that is held in processes across the network. Unlike gather, however, allgather collects the data to every process in the network, rather than just one root node. The short vector algorithm is a minimum spanning tree gather followed by an MST broadcast, while the long vector algorithm uses a bucket allgather approach.

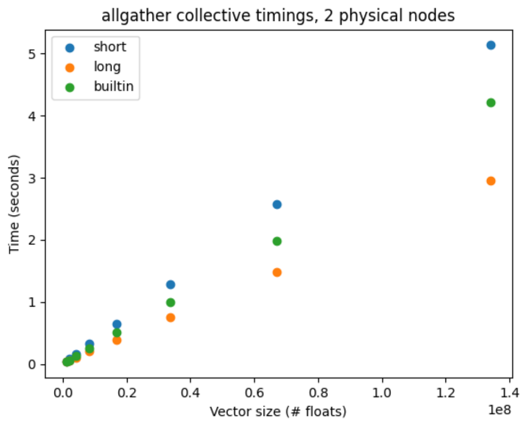

Fig. 5.5.1. Allgather collective, 2 compute nodes.

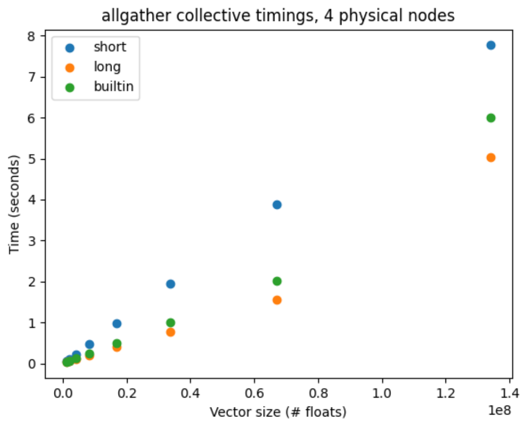

Fig. 5.5.2. Allgather collective, 4 compute nodes.

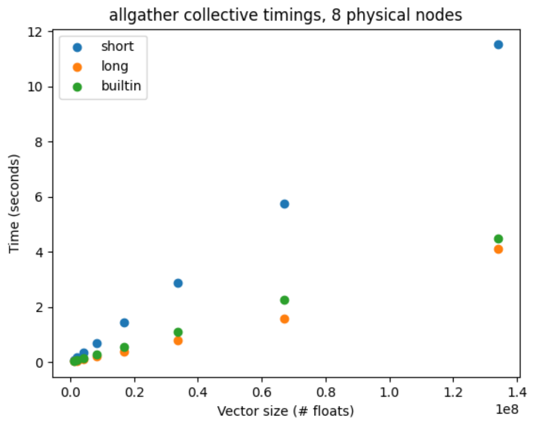

Fig. 5.5.3. Allgather collective, 8 compute nodes.

Again, the long vector algorithm was the best performer of the three. This is most pronounced across two physical nodes, while the builtin algorithm is more competitive when using four or eight. The performance penalty for using the short vector algorithm became more pronounced as the number of physical nodes increased. This is significant because it suggests that the latency penalty incurred by sending more messages using the bucket algorithm is not particularly important, even across different physical compute nodes.

F. Reduce-scatter

The reduce-scatter operation, as the name suggests, performs a reduction on the data and scatters the results across every node. This is a reduction followed by a scatter (as opposed to reduction followed by a broadcast, which is an allreduce). The short vector algorithm is a minimum spanning tree reduce followed by an MST scatter, while the long vector algorithm is a bucket reduce-scatter.

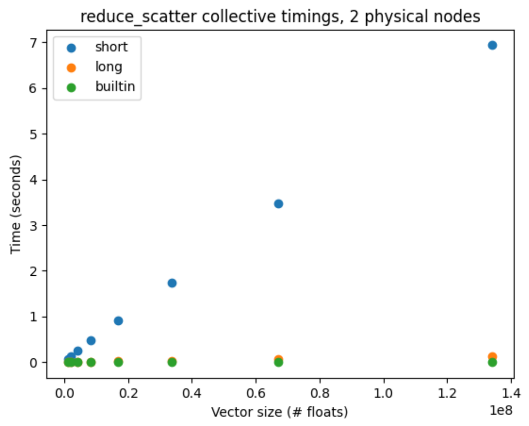

Fig. 5.6.1. Reduce-scatter collective, 2 compute nodes.

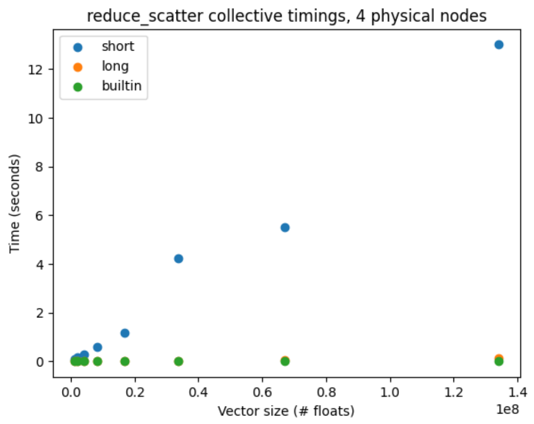

Fig. 5.6.2. Reduce-scatter collective, 4 compute nodes.

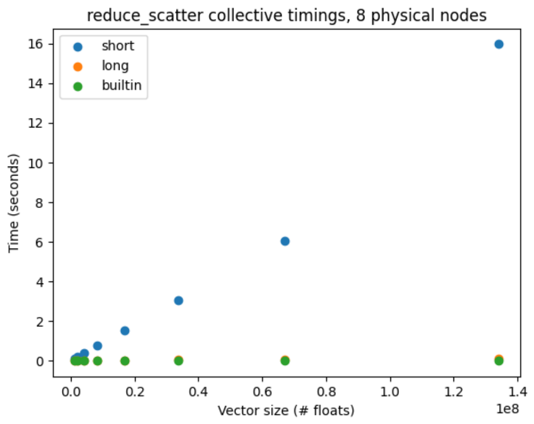

Fig. 5.6.3. Reduce-scatter collective, 8 compute nodes.

The experiments gave almost identical results for the builtin and long algorithms, to such an extent that we can be reasonably certain that Open MPI uses a bucket algorithm for its reduce-scatter operation. The short vector algorithm was markedly worse across all physical nodes.

G. Allreduce

The allreduce operation performs a reduction and broadcasts the full result to every node in the network. The short vector algorithm is a minimum spanning tree reduce followed by an MST broadcast, and the long vector algorithm is a bucket reduce-scatter followed by a bucket allgather.

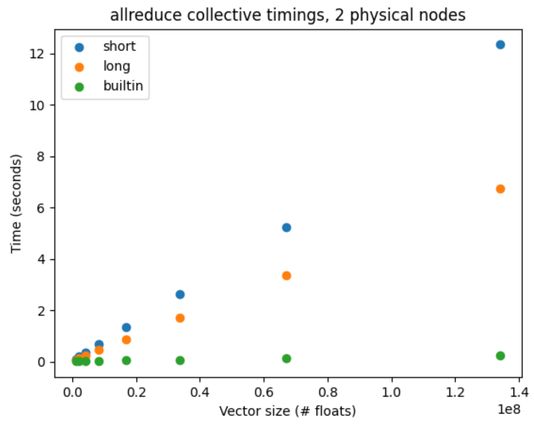

Fig. 5.7.1. Allreduce collective, 2 compute nodes.

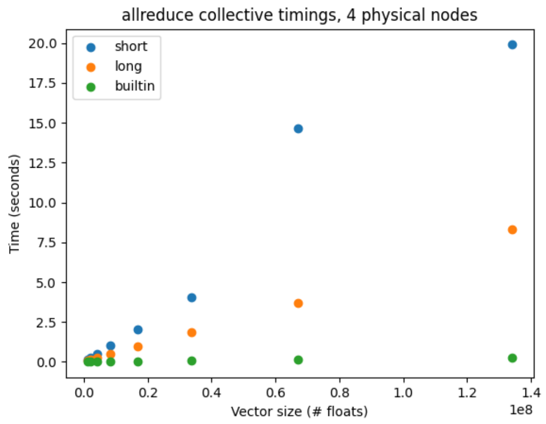

Fig. 5.7.2. Allreduce collective, 4 compute nodes.

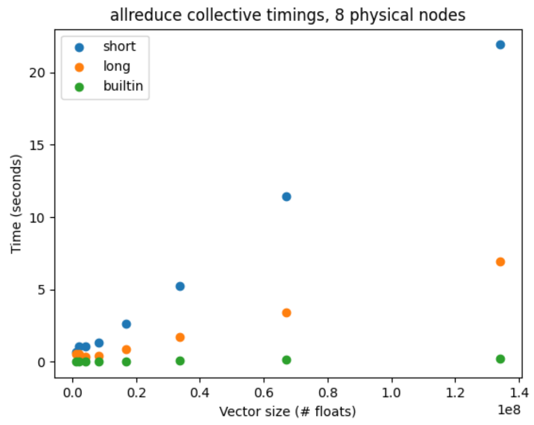

Fig. 5.7.3. Allreduce collective, 8 compute nodes.

The builtin algorithm performed best here, followed by the long vector algorithm. With the exception of broadcast, the primitives that make up the short and long vector algorithms (MST reduce, MST broadcast, bucket reduce-scatter, bucket allgather) match or outperform the builtin algorithm for each operation they are designed for, suggesting that constructing an allreduce operation from other collective operations is not an optimal approach.

H. Discussion

For the scatter, gather, allgather, and reduce-scatter collectives, one of the algorithms Chan describes outperforms the Open MPI builtin operation. The builtin algorithm outperforms the others for the broadcast, reduce, and allreduce collectives. A summary of the best performing algorithm class for each collective is given in table III.

Collective operation	Best algorithm class
Broadcast	builtin
Reduce	builtin
Scatter	long
Gather	short
Allgather	long
Reduce-scatter	long/builtin
Allreduce	builtin

TABLE III
BEST PERFORMING CLASS OF ALGORITHM, BY COLLECTIVE.

The results are reflective of the challenges in building a one-size-fits-most algorithm for collective communication operations. No class of algorithms consistently performed best, regardless of the number of physical compute nodes used.

Despite using the same set of data sizes across every experiment, there seemed to be no obvious answer to whether or not the vectors were ’short’ or ’long’ based on algorithm performance. These are, of course, relative terms – the smallest length I benchmarked, 2^20 or about one million elements, is not particularly short in many contexts. What length of vector is considered short or long is clearly variable across different collective operations.

There was no clear ’crossover point’ where any class of algorithm ceased to be the best for a given operation, nor did the builtin algorithm ever appear to switch to a different implementation, despite this being the purpose of Open MPI’s fixed decision mode. Further benchmarks are needed against different data sizes to determine at which data size such a point lies.

VI. Summary

Collective communication operations are integral to the high-performance computing applications that occur on OU’s Schooner supercomputer. Chan et al describe the lower performance bounds of several of these operations, as well as several methods for implementing them. I describe the lower performance bound of an additional collective communication operation, the rotation.

Chan’s algorithms use three different approaches to the problem of communicating data: the simple message-passing approach, the minimum spanning tree, and the bucket algorithm. These algorithms, as well as different combinations of these algorithms, are intended to reduce the time needed to complete a given collective communication operation when sending short and long vectors.

I find that on the Schooner supercomputer, no algorithm class consistently outperformed the others. Open MPI's builtin methods were the best performers for the broadcast, reduce, and allreduce operations. Chan's long vector algorithms were best for the scatter, allgather, and reduce-scatter operations, while their short vector algorithm was best for the gather operation.

I did not determine any data size at which one class of algorithm ceases to be the best for a given operation and another class begins to outperform it. Further work is needed to find the vector lengths for which each algorithm class performs best and construct a decision tree that is optimal for Schooner, as Open MPI's fixed-decision strategy attempts to do for arbitrary distributed systems.

Acknowledgements

The computing for this project was performed at the OU Supercomputing Center for Education & Research (OSCER) at the University of Oklahoma (OU).

I am grateful to my instructor, Dr Richard Veras, for his knowledge and guidance on this project.

References

[1] E Chan, M Heimlich, A Purkayastha, and R van de Geijn. *Collective communication: theory, practice, and experience*. Concurrency Comput-tat.: Pract. Exper. 2007; 19:1749–1783.

[2] "11.10. Tuning Collectives — Open MPI 5.0.x documentation," *Open-mpi.org*, 2024. https://docs.open-mpi.org/en/v5.0.x/tuning-apps/coll-tuned.html (accessed 14 August 2024).

[3] J Rico-Gallego, J Díaz-Martín, R Manumachu, A Lastovetsky. *A Survey of Communication Performance Models for High-Performance Computing*. ACM Computing Surveys 2019; 51(6), 126:1-36.

[4] Z Cai, Z Liu, S Maleki, M Musuvathi, T Mytkowicz, J Nelson, O Saarikivi. *Synthesizing Optimal Collective Algorithms*. PPoPP '21, February 27–March 3, 2021, Virtual Event, Republic of Korea.