Interprocessor Collective Communication Library (InterCom) *

Mike Barnett
Department of Computer Science
University of Idaho
Moscow, Idaho 83844-1010

Satya Gupta David G. Payne
Supercomputer Systems Division
Intel Corporation
15201 N.W. Greenbrier Pkwy
Beaverton, Oregon 97006

Lance Shuler Robert van de Geijn
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712–1188

Jerrell Watts
Center for High Performance Computing
The University of Texas
Austin, Texas 78758–4497

Abstract

In this paper, we outline a unified approach for building a library of collective communication operations that performs well on a cross-section of problems encountered in real applications. The target architecture is a two-dimensional mesh with worm-hole routing, but the techniques also apply to higher dimensional meshes and hypercubes. We stress a general approach, addressing the need for implementations that perform well for various sized vectors and grid dimensions, including non-power-of-two grids. This requires the development of general techniques for building hybrid algorithms. Finally, our approach also supports collective communication within a group of nodes, which is required by many scalable algorithms. Results from the Intel Paragon system are included.

1 Introduction

The Interprocessor Collective Communication (InterCom) Project is a comprehensive study of techniques for a high performance implementation of commonly used collective communication algorithms. It is the emphasis on a high performance implementation that sets it aside from the MPI effort [6], which tries to *standardize* the *interface* to communication libraries. Indeed, we expect the fruits of our efforts to be incorporated into implementations of the MPI standard.

The following collective communication operations have been identified as being useful in many applications: *broadcast, scatter, gather, collect*, and *global combine.* Typical approaches to implementing such collective communication algorithms are limited to considering the case of short vectors, which are treated with one technique, or to considering the case of long vectors, for which very different techniques are appropriate. For a general purpose library, it is crucial that an implementation performs well for *all* vector lengths.

In our previous papers on collective communication, we studied individual communication operations and their implementation, including possible hybrid approaches [1, 2, 3, 5]. It is through this progression of studies that we have discovered that all the aforementioned collective communication operations can be built from similar primitives. It is this observation that has led us to propose a unified approach to hybrid design.

Finally, there is a strong applications need for communication operations which support collective communication within a subset of the nodes comprising the application. Such a subset of nodes could be a row, a column, a block of nodes, or simply a list of nodes. Our approach builds on and supports such *group* collective communication.

2 Target Architectures

Our current implementation assumes a two-dimensional physical mesh of processing nodes, with bidirectional links between nodes, and worm-hole (cut-through) routing. Furthermore, we assume that it is possible to model the time required for sending a message of length n bytes between any two nodes by

$$\alpha + n\beta$$

where α is the latency for sending a message, and β is the communication time per item, in the absence of network conflicts. A processor can both send and receive at the same time. But it can only send to, or receive from, one other node at a given time. When two messages traverse the same physical link on the communication interconnect, we assume they share the bandwidth of that link.

3 Target Collective Communication

We explain the target collective communication operations in the setting where all processors are involved in the communication. There are p processors, labeled $\mathbf{P}_0, \ldots, \mathbf{P}_{p-1}$. We use x to represent a vector containing n data items; it is partitioned into subvectors,

$$x = \begin{pmatrix} x_0 \\ \vdots \\ x_{p-1} \end{pmatrix}$$

where x_i is of length n_i. Similarly, vector $y^{(j)}$, $j = 0, \ldots, p-1$, contains n items, and is partitioned conformal with x. The operation $\oplus$ represents an associative and commutative combine operation such as an element-wise summation or element-wise product of vectors.

The target collective communication operations are given in Table 1.

4 Building Blocks

The building blocks for our library consist of the following implementations of the operations *for linear array networks with worm-hole routing*:

Short vector primitives. These are implementations of the operations in Table 1 that (attempt to) minimize the number of startups incurred. For example, the broadcast uses a minimum spanning tree (MST) implementation that incurs $\lceil \log_2(p) \rceil$ startups.

Long vector primitives. A few of the communication operations, which we refer to as long vector primitives, can be used as building blocks for efficient implementations of all the collective communication operations for long vectors. For long vectors, we emphasize minimizing the penalty per item of data, eliminating network conflicts, and simplicity. The long vector primitives and their implementations are:

MST scatter. The scatter is implemented by recursively halving the vector and network, thereby avoiding all network conflicts.

MST gather. The scatter operation in reverse.

Bucket collect. For this implementation of the collect, the linear array is viewed as a ring. Buckets are passed between the nodes that move the subvectors to be collected, leaving the result on all nodes, as illustrated in Fig. 1. Note that no network conflicts occur.

Bucket distributed global combine. Similar to the bucket collect, executed in reverse, where the buckets are used to accumulate contributions. This is illustrated in Fig 2.

The implementations of the short and long vector primitives can be shown to be optimal on linear arrays.

Using the long vector primitives, we can build efficient long vector implementations for the other communications:

Broadcast. MST scatter followed by bucket collect. This can be shown to be asymptotically within a factor of two of optimal.

Combine-to-one. Distributed bucket combine followed by a MST gather. This can be shown to be asymptotically within a factor of two of optimal.

Combine-to-all. Distributed bucket combine followed by a bucket collect. This can be shown to be asymptotically optimal.

5 Hybrid algorithms

We illustrate how hybrid algorithms can be formed by first considering a logical two-dimensional mesh of nodes. All of the hybrids are implemented using one of the following templates:

Template 1.

1. Perform the operation within rows.

Operation	Before	After
Broadcast	x at $\mathbf{P}_k$, k given	x at all $\mathbf{P}_j$
Scatter	x at $\mathbf{P}_k$, k given	x_j at $\mathbf{P}_j$
Gather	x_j at $\mathbf{P}_j$	x at $\mathbf{P}_k$, k given
Collect	x_j at $\mathbf{P}_j$	x at $\mathbf{P}_j$
Combine-to-one	$y^{(j)}$ at $\mathbf{P}_j$	$\oplus_{i=0}^{p-1} y^{(i)}$ at $\mathbf{P}_k$, k given
Combine-to-all	$y^{(j)}$ at $\mathbf{P}_j$	$\oplus_{i=0}^{p-1} y^{(i)}$ at $\mathbf{P}_j$
Distributed Combine	$y^{(j)}$ at $\mathbf{P}_j$	$\oplus_{i=0}^{p-1} y_j^{(i)}$ at $\mathbf{P}_j$

Table 1: Summary of target collective communication operations.

Figure 1: Bucket collect on four processors.

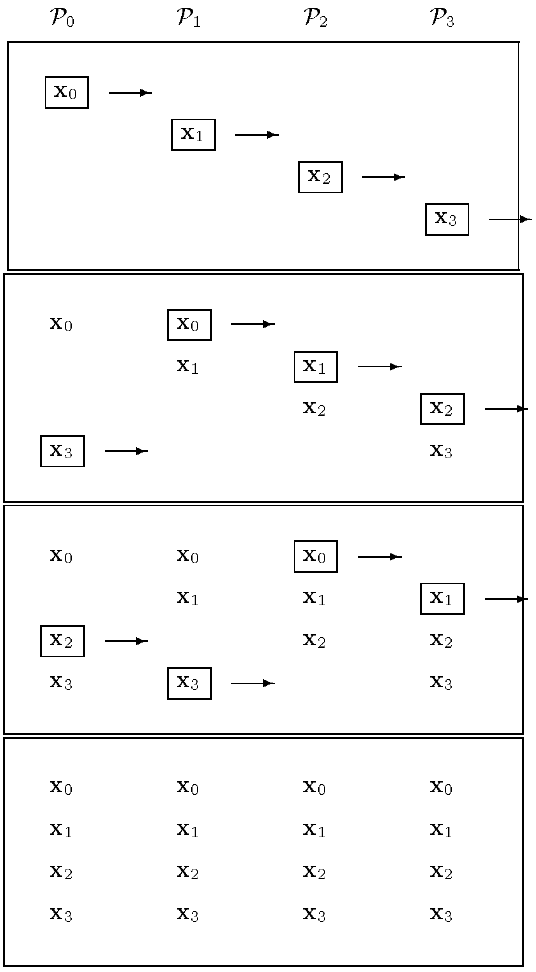
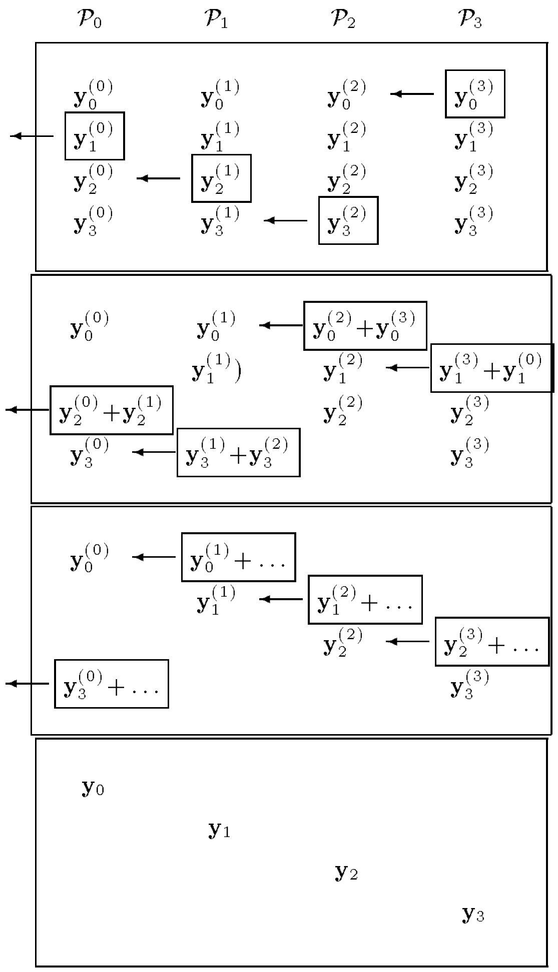


Figure 2: Bucket distributed global combine on four processors.



2. Perform the same operation within columns using the results from 1.

Examples:

- The **scatter** can be implemented as a scatter within the column containing the root, followed by independent scatters within each row. The new root for the latter operation will be that node in the row which lies in the same column as the original root.
- The **collect** can be implemented as a collect within rows followed by a collect of the results within columns.

Template 2.

1. Perform the first stage of an operation within rows.
2. Perform the operation within columns using the results from 1.
3. Perform the second stage of an operation within rows using the results of 2.

Examples:

- The **broadcast** can be implemented as a scatter within rows, followed by a broadcast of the results within columns, followed by a collect within rows.
- A **combine-to-all** can be implemented as a distributed bucket combine within rows, followed by a combine-to-all within columns, followed by a bucket collect within rows.

For the operation within columns, either a short or long vector strategy can be employed, depending on whether the first step reduces the vector length to the point where latency dominates.

A simple extension of the above approach leads to general hybrid algorithms for multidimensional meshes. If the nodes are viewed as a higher dimensional mesh than the physical mesh, different strategies can be employed in the different dimensions of this logical mesh. Strategies that successively reduce the vector lengths during initial stages will reduce overall time due to bandwidth. Eventually, vectors become small enough and strategies that reduce latency (short vector strategies) become more appropriate.

Physical linear arrays in the network like rows and columns are now viewed as logical multidimensional arrays based on the factorization of the row and/or column dimension. This means that during later stages network conflicts will start occurring. The effects of this can be reduced by an appropriate choice, depending on the vector length, on the number of dimensions, on the order of the dimensions, and on the strategy for each dimension. For example, alternating physical directions during stages that utilize a strategy that shortens the vector lengths greatly reduces the effects of network conflicts during later stages.

This approach to hybrids is illustrated for the broadcast in Fig. 3 for 12 nodes when they are logically viewed as a $2 \times 2 \times 3$ mesh. During the first stage, a scatter is employed in the first dimension, and no network conflicts occur. During the second stage, again a scatter occurs, this time in the second dimension. In this second step, network conflicts may occur, depending on details of the physical network. Finally, the vector is assumed to be short enough that a minimum spanning tree broadcast within the final dimension is appropriate. Network conflicts may occur, but the vectors are now 1/4 of the original length. The process is completed by a collect corresponding to the second scatter followed by a collect corresponding to the first scatter. *We emphasize that this example is meant only to illustrate the mechanism.*

In practice, our implementations use highly accurate models of the expected execution time of the different choices, in combination with a cheap but effective heuristic, to make the specific choice of hybrid strategy, based on the vector length, the number of processors, the history of the hybrid, and a few parameters that describe the communication network.

It should be noted that this approach has a heavy dependence on the integer factorization of the dimensions of the physical mesh. As a result, if one or both of these dimensions are prime, or have only a few large integer factors, the hybrid algorithms will not be very effective.

6 Experimental Results

In this section, we present experimental results from a complete implementation of the library for the case of collective communication within all nodes. The experiments were performed on a 512-node Intel Paragon, running under OSF release R1.1. We present data from two different physical mesh partitions: the case of a 16×32 mesh, which has the considerable advantage of power-of-two dimensions, and a 15×30 mesh, which deviates significantly from a power-of-two mesh. To highlight the benefits of the hybrid algorithms, we present data in Table 2 for three vector lengths that shows the time of the different algorithms for short, medium, and long vectors. The results for a wide range of vector lengths are presented in Figure 4. These results indicate that hybrid algorithms perform

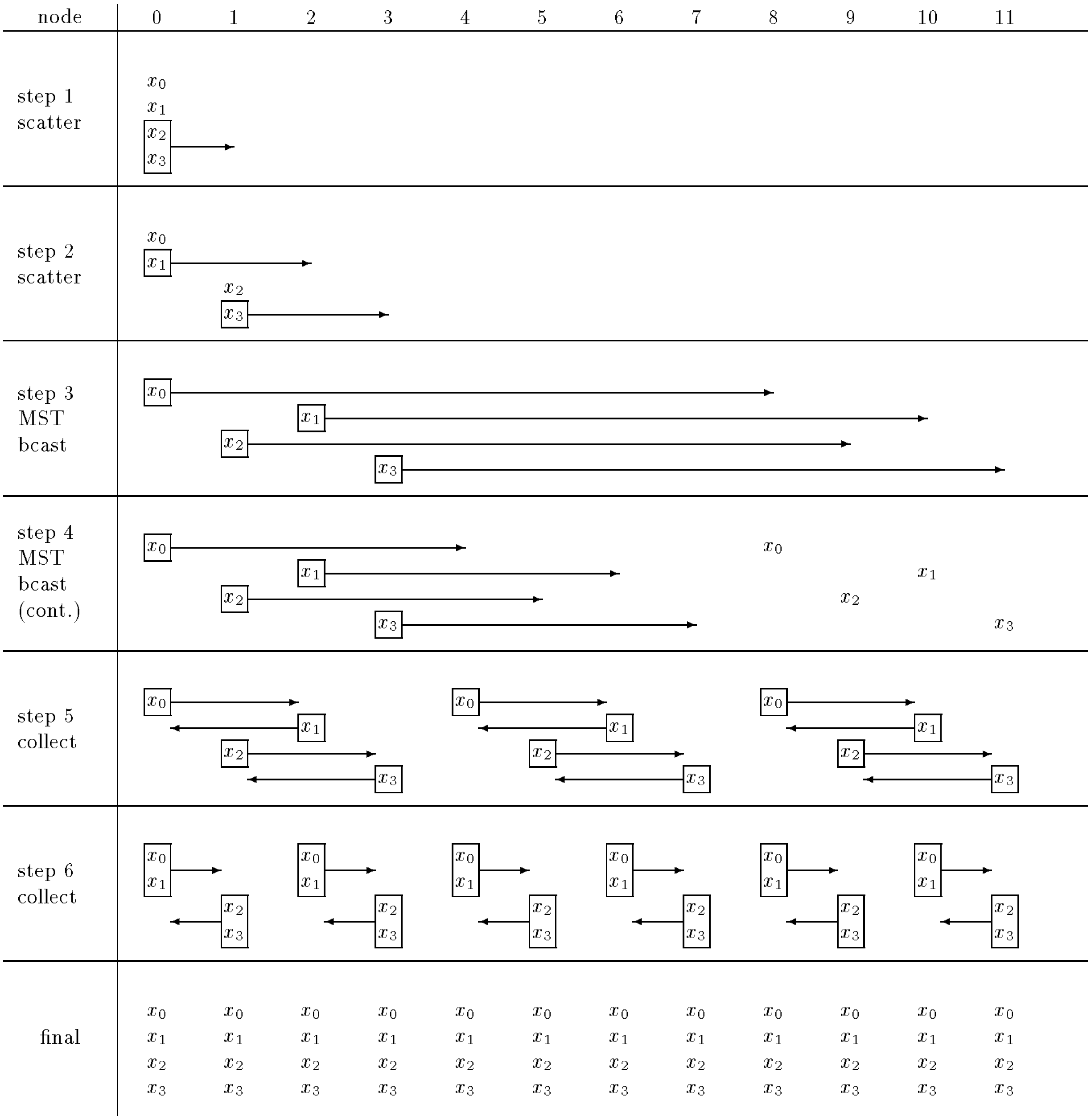


Figure 3: Broadcast hybrid: In Step 1 and 2, scatters within subgroups of two nodes are performed. Next, separate MST broadcasts within subgroups of three nodes are performed in Step 3 and 4. Finally, simultaneous collects within subgroups of two complete the broadcast. Except for Step 1 and 6, limited network conflicts occur. The strategy benefits from the fact that network conflict is least when the vectors sent are long. Notice that this is one of many possible strategies: A MST broadcast involving all 12 nodes; Scatter involving four nodes, MST broadcast in groups of three, collect in groups of four; etc.

better than short and long vector algorithms across the range of vector lengths. In a very few cases the short or long vector algorithm performs slightly better than the hybrid algorithm. We would like to mention that these results are based on the preliminary heuristics for determining hybrid algorithms, which are still being tuned.

		size in bytes		
Operation	Method	256	262144	1048576
Broadcast	short	0.00128	0.12401	0.48549
	long	0.02939	0.05426	0.11965
	hybrid	0.00139	0.03932	0.09957
Collect	short	0.00338	0.07789	0.29512
	long	0.02672	0.04358	0.08838
	hybrid	0.00355	0.03074	0.07469
Global Sum-to-all	short	0.00676	0.15933	0.60273
	long	0.05547	0.09729	0.19747
	hybrid	0.00696	0.07251	0.17360
Scatter	short	0.00119	0.03927	0.14828
	long	0.00269	0.01161	0.03431
	hybrid	0.00125	0.01139	0.03506

Table 2: Time (in sec.) for the different operations and algorithms for short, intermediate, and long vector lengths. All results are for a 16×32 mesh of nodes.

7 Other algorithms

It should be noted that for some of the communications, optimal algorithms for long vectors exist that in theory outperform our approach. For example, on hypercubes Ho and Johnsson's EDST broadcast [4] will outperform our scatter/collect broadcast by a factor of two for long vectors. However, it is our experience that such pipelined algorithms are generally difficult to implement and are extremely architecture dependent. Indeed, such theoretically superior algorithms are often outperformed by simpler algorithms (e.g. scatter-collect algorithm) when implemented on real systems. This has been our experience on the Touchstone Delta and the Intel Paragon, making these approaches inappropriate for general library development.

8 Current Status and Future Work

We have implemented a complete library for the Intel Paragon, based on the described techniques. This library exhibits considerably better performance than any other collective communication library for the Paragon we have seen, and has expanded functionality. As of this writing, the library only allows collective communications that involve all nodes.

To port the library between platforms, or, more importantly, to tune it for new operating system releases, it suffices to enter a few parameters that describe the latency, bandwidth, and computation characteristics of the system. Indeed, we ported the library from the original version, which was designed for the Touchstone Delta, to the Paragon by changing only these parameters.

For completeness, we also have a version tuned for the Touchstone Delta mesh computer, and a version for the iPSC/860 that has the same functionality, but uses algorithms more appropriate for hypercubes (including the EDST broadcast). A version tuned for the SUNMOS operating system developed at Sandia National Laboratory is also under development.

There is a strong applications need for library routines that support collective communication within a subset of the nodes comprising the application – collective communication within *groups*. Such a group could be a row, a column, a block of nodes, or simply a list of nodes. Our approach builds on and supports group collective communication. We expect to release an updated version of the library that allows for group collective communication in the near future.

9 Conclusion

This project provides the building blocks for very efficient implementations of collective communication libraries like MPI. The only requirement is that reasonable algorithms exist for short vectors and for the building blocks used to derive algorithms for long vectors. Beyond this, we make few restrictions on the underlying architecture. For example, we do not require a power-of-two number of nodes. Finally, the scope of this project goes beyond the derivation of hybrid algorithms in that eventually our collective communications will also operate within groups.

Acknowledgements

This research was sponsored in part by Intel Supercomputer Systems Division, the Intel Research Council and the University of Texas Center for High Performance Computing.

This research was performed in part using the Intel Paragon System and the Intel Touchstone Delta

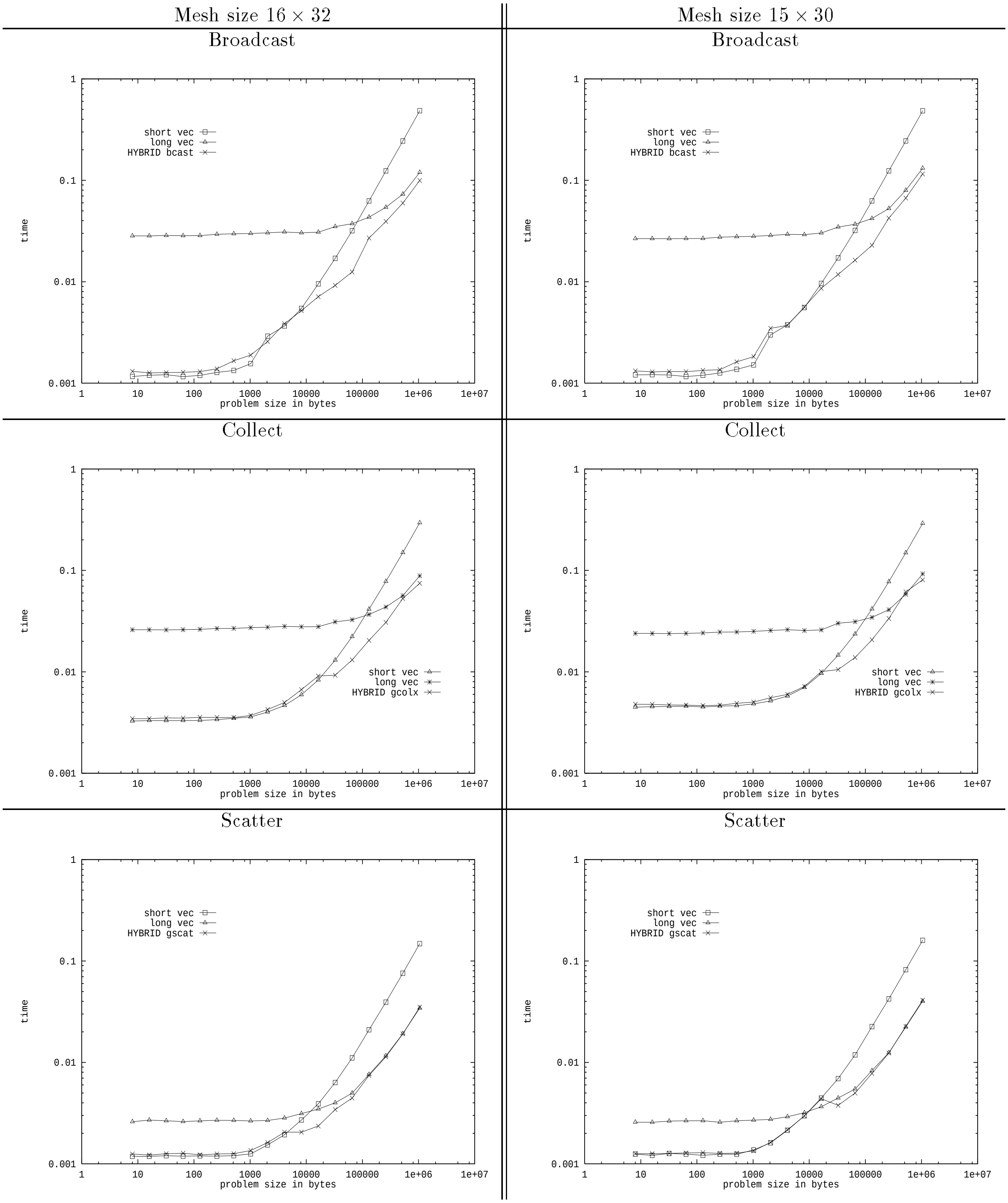


Figure 4: Performance of the various algorithms on the Intel Paragon.

System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Intel Supercomputer Systems Division and the California Institute of Technology.

Obtaining Documentation

A number of the mentioned papers that inspired this work are available via anonymous `ftp` from `cs.utexas.edu` directory `pub/rvdg`. Manuals and other information regarding the InterCom library can be found there as well. To obtain the InterCom library for Intel Paragon systems, contact `shuler@cs.utexas.edu`. For the iPSC/860 implementation, contact `jwatts@chpc.utexas.edu`.

The body of papers related to the implementation of communication algorithms on various architectures is vast. We limit ourselves to listing our own papers that led us to the presented approach, and a few key papers of others. Further references can be found in the bibliographies of those papers.

References

[1] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn. Efficient Communication Primitives on Mesh Architectures with Hardware Routing, Sixth SIAM Conf. on Par. Proc. for Sci. Comp., Norfolk, Virginia, March 22-24, 1993.

[2] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn, Global Combine on Mesh Architectures with Wormhole Routing, *7th International Parallel Processing Symposium*, pages 156–162, IEEE Computer Society Press, Newport Beach, CA, April 13-16, 1993.

[3] M. Barnett, D.G. Payne, R. van de Geijn, and J. Watts, Broadcasting on Meshes with Worm-Hole Routing, *Journal of Parallel and Distributed Computing*, submitted.

[4] C.-T. Ho and S. L. Johnsson, Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes, In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648, IEEE, 1986.

[5] R. A. van de Geijn. Efficient Global Combine Operations. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 291–294. IEEE Computer Society Press, 1991.

[6] D. W. Walker, The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers. to appear in *Parallel Computing*, April 1994. Upto Date information about the MPI standard is available from `netlib`, directory `mpi`.