# Building a High-Performance Collective Communication Library

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

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

Lance Shuler
Parallel Computing Sciences Department, 1424
Sandia National Laboratory
Albuquerque, New Mexico 87185-1109

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

Jerrell Watts
Computer Science Department
California Institute of Technology
Pasadena, California 91125

## Abstract

*In this paper, we report on a project to develop 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 are more general. The approach differs from traditional library implementations in that we address the need for implementations that perform well for various sized vectors and grid dimensions, including non-power-of-two grids. We show how a general approach to hybrid algorithms yields performance across the entire range of vector lengths. Moreover, many scalable implementations of application libraries require collective communication within groups of nodes. Our approach yields the same kind of performance for group collective communication. Results from the Intel Paragon system are included. To obtain this library for Intel systems contact* `intercom@cs.utexas.edu`.

## 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 this emphasis on a high-performance *implementation* that sets it aside from the MPI effort [16], 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 an earlier paper [1], we have presented a general approach to building collective communication libraries. In this paper, we give a further description of this approach, with emphasis on an explanation of hybrids.

## 2 Target Architectures

Our current implementation assumes a two-dimensional physical mesh of processing nodes, with

1063-9535/94 $4.00

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 $\alpha$ is the latency for sending a message, and $\beta$ 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. In addition, we assume that the time for performing an arithmetic operation is denoted by $\gamma$.

## 3 Target Collective Communication

We explain the target collective communication operations in the setting where all processors are involved in the communication. Assume there are $p$ processors, labeled $\mathbf{P}_0, \ldots, \mathbf{P}_{p-1}$. Let $x$ represent a vector containing $n$ data items; $x$ 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. We assume in this paper that $n_i \approx n/p$.

The target collective communication operations are given in Table 1.

## 4 Building Blocks

In this section, we present building blocks for our library. All the building building blocks have the property that they

- are simple to implement,
- do not require power-of-two size partitions, and
- incur no network conficts.

The implementations of the short vector primitives can be shown to have optimal latency. The implementation of the long vector primitives can be shown to be asymptotically optimal on linear arrays as vector size increases.

We start by discussing the implementation of the building blocks in the setting of linear arrays, which due to worm-hole routing can be considered unidirectional rings. (For example, if all messages are sent to the right nearest neighbor, only the rightmost processor in the linear array sends to the left. Hence, there are no message conflicts.)

### 4.1 Short Vector Primitives

Algorithms for implementing collective communications for short vectors must minimize startup cost, i.e. the number of messages sent. On hypercubes, this can be easily accomplished by staging the algorithms as $\log p$ steps during which communication is performed in each hypercube dimension. For meshes, this idea can be utilized as well, provided some care is taken at each stage [5].

All our target short vector collective communication operations can be built from four primitives. These are **broadcast**, **combine-to-one**, **scatter**, and **gather**.

Consider the **broadcast**. For short vectors, this operation can be implemented on a linear array of nodes in the following way: Start by assuming a given root node has the message of length $n$. The broadcast can proceed by dividing the linear array in two (approximately) equal parts and choosing a receiving node in the part that does not contain the root. The broadcast proceeds recursively by treating each of the involved nodes as a new root for a broadcast within its own half of the previous array. It is easy to see that no network conflicts occur and the total time required is

$$\lceil \log p \rceil (\alpha + n\beta).$$

The **combine-to-one** can be implemented similarly by running the broadcast communications in reverse order and interleaving communication with the combine operation. This requires a total time of

$$\lceil \log p \rceil (\alpha + n\beta + n\gamma).$$

The **scatter** can be implemented like the broadcast, except at each stage only the data that eventually resides in the other part of the network is sent. If each node receives an equal share of the initial vector, the cost is approximately

$$\lceil \log p \rceil \alpha + \frac{p-1}{p} n\beta.$$

| **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.

The **gather** can be implemented as the scatter in reverse and incurs the same cost.

### 4.2 Long Vector Primitives

For long vectors, a strategy that minimizes overhead due to vector length, in addition to avoiding network conflicts, is necessary. It should be noted that the above mentioned **scatter** and **gather** operations have this property, and they also act as long vector primitives. In addition, we propose two more long vector primitives, the **bucket collect** and **bucket distributed combine**. These four primitives constitute the set from which all our target long vector collective communication operations can be built.

The **bucket collect** is a special implementation of the collect, which views the linear array as a ring. **Buckets** are passed between the nodes that move the subvectors to be collected, leaving the result on all nodes. Note that no network conflicts occur. Cost:

$$(p-1)\alpha + \frac{p-1}{p} n\beta.$$

The **bucket distributed global combine** is similar to the bucket collect, executed in reverse, where the buckets are used to accumulate contributions. Cost:

$$(p-1)\alpha + \frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma.$$

## 5 Using the building blocks

In this section, we describe how the short and long vector primitives can be used to generate short and long vector implementations for all collective communications.

### 5.1 Short vector algorithms

For short vectors, the broadcast, combine-to-one, scatter and gather primitives are, of course, implementations of the operations themselves. The other three collective communications can be generated using these primitives as follows:

**Collect:** Gather followed by broadcast. Cost:

$$2\lceil \log p \rceil \alpha + \left( \frac{p-1}{p} + \lceil \log p \rceil \right) n\beta.$$

**Distributed global combine:** Combine-to-one followed by scatter. Cost:

$$2\lceil \log p \rceil \alpha + \left( \frac{p-1}{p} + \lceil \log p \rceil \right) n\beta + \lceil \log p \rceil n\gamma.$$

**Global combine-to-all:** Combine-to-one followed by broadcast. Cost:

$$2\lceil \log p \rceil \alpha + 2\lceil \log p \rceil n\beta + \lceil \log p \rceil n\gamma.$$

For all these implementations, the startup cost is within a factor two of optimal. On both the Touchstone Delta and the Paragon, due to machine specific issues, the startup actually is optimal.

### 5.2 Long vector algorithms

For long vectors, the collect, distributed combine, scatter and gather primitives are once again the implementations themselves. The other three collective communications can be generated using these primitives as follows:

**Broadcast:** Scatter followed by collect. Cost:

$$(\lceil \log p \rceil + p - 1)\alpha + 2\frac{p-1}{p} n\beta.$$

**Combine-to-one:** Distributed combine followed by gather. Cost:

$$2(p-1)\alpha + 2\frac{p-1}{p}n\beta + \frac{p-1}{p}n\gamma.$$

**Global combine-to-all:** Distributed combine followed by collect. Cost:

$$2(p-1)\alpha + 2\frac{p-1}{p}n\beta + \frac{p-1}{p}n\gamma.$$

In later sections, we will talk about stage 1 and 2 of the long vector algorithms for each of the communications.

For the broadcast and combine-to-one, it can be argued that the $\beta$ term is asymptotically within a factor two of optimal, while for the combine-to-all it can be argued that the $\beta$ term is asymptotically optimal.

## 6 Hybrid algorithms

We illustrate the different possibilities for creating hybrid algorithms by considering the broadcast operation on a linear array of 12 nodes, with $\mathbf{P}_0$ as root. At the extremes, we can use the minimum spanning tree or the scatter/collect broadcasts. Other choices view the linear array logically as a higher dimensional mesh and within each dimension, a choice is made to do scatter/collect or minimum spanning tree broadcasts. Fig. 1 illustrates an example of this.

For larger numbers of nodes, a larger number of choices exist. This is illustrated in Table 2 and Fig. 2 for the case of 30 nodes. The first entry in Table 2 is described by the pair $(3 \times 10, SMC)$, which indicates a logical $3 \times 10$ grid, with the broadcast executed as a Scatter in the first dimension, a Minimum spanning tree broadcast in the second dimension, dimension, and finally a Collect in the first dimension. Similarly, the second entry in Table 2 is described by the pair $(2 \times 3 \times 10, SSMCC)$, which indicates a logical $2 \times 3 \times 10$ grid, with the broadcast executed as a Scatter in the first dimension, a Scatter in the second dimension, a Minimum spanning tree broadcast in the third dimension, a Collect in the second dimension, and finally a Collect in the first dimension. Other entries in the table have analogous interpretations.

In general, given a linear array of $p$ nodes which is logically viewed as a $d_1 \times \ldots \times d_k$ mesh, there are a large number of choices for the broadcast. (Notice that $k$ must also be chosen.) Again, Table 2 and Fig. 2 illustrate several possible hybrid solutions for $p = 30$.

| logical mesh | hybrid | time |
|---|---|---|
| $3 \times 10$ | SMC | $16\alpha + (240/30)n\beta$ |
| $2 \times 3 \times 5$ | SSMCC | $9\alpha + (160/30)n\beta$ |
| $1 \times 30$ | M | $5\alpha + (150/30)n\beta$ |
| $2 \times 15$ | SMC | $6\alpha + (150/30)n\beta$ |
| $6 \times 5$ | SMC | $11\alpha + (140/30)n\beta$ |
| $5 \times 6$ | SMC | $10\alpha + (138/30)n\beta$ |
| $5 \times 2 \times 3$ | SSMCC | $11\alpha + (138/30)n\beta$ |
| $5 \times 2 \times 3$ | SSSCCC | $13\alpha + (118/30)n\beta$ |
| $2 \times 3 \times 5$ | SSSCCC | $13\alpha + (118/30)n\beta$ |
| $15 \times 2$ | SMC | $19\alpha + (118/30)n\beta$ |
| $15 \times 2$ | SSCC | $20\alpha + (118/30)n\beta$ |
| $10 \times 3$ | SMC | $15\alpha + (114/30)n\beta$ |
| $5 \times 6$ | SSCC | $15\alpha + (98/30)n\beta$ |
| $6 \times 5$ | SSCC | $15\alpha + (98/30)n\beta$ |
| $3 \times 10$ | SSCC | $17\alpha + (94/30)n\beta$ |
| $10 \times 3$ | SSCC | $17\alpha + (94/30)n\beta$ |
| $2 \times 15$ | SSCC | $20\alpha + (86/30)n\beta$ |
| $1 \times 30$ | SC | $34\alpha + (58/30)n\beta$ |

Table 2: Some choices of hybrids and their expense when broadcasting on a linear array with 30 nodes[1]. The choices are listed in increasing order of the $\beta$ term. Which makes them progressively more appropriate for long vectors, at a cost of higher latency.

The cost for broadcasting for a given $p$, with strategy $(\mathrm{SS}\cdots\mathrm{SC}\cdots\mathrm{CC}, d_1 \times \ldots \times d_k)$ is given by

$$\sum_{i=1}^{k} \left[ \begin{array}{ll} (\lceil \log(d_i) \rceil + d_i - 1)\alpha & +2 \times \boldsymbol{\prod_{j=1}^{i-1} d_j} \\ & \times \frac{d_1-1}{d_1} \times \frac{n}{\prod_{j=1}^{i-1} d_j}\beta \end{array} \right]$$

$$= \sum_{i=1}^{k} \left[ (\lceil \log(d_i) \rceil + d_i - 1)\alpha + 2 \times \frac{d_1 - 1}{d_1} \times n\beta \right].$$

Similarly, the cost of a strategy given by $(\mathrm{SS}\cdots\mathrm{SMC}\cdots\mathrm{CC}, d_1 \times \ldots \times d_k)$ has cost

$$\sum_{i=1}^{k-1} \left[ (\lceil \log(d_i) \rceil + d_i - 1)\alpha + 2 \times \frac{d_1 - 1}{d_1} \times n\beta \right] + \lceil \log(d_k) \rceil \alpha + \lceil \log(d_k) \rceil n\beta.$$

Here the bold-face indicates factors included to compensate for network conflicts.

---

[1] An observant reader will notice that three of the examples in Table 2 have a cost which in fact are *worse* than the minimum spanning tree broadcast cost, $5\alpha + 5n\beta$. Those hybrid solutions are included in the table as we feel that they help illustrate the mechanism by which hybrids can be chosen.

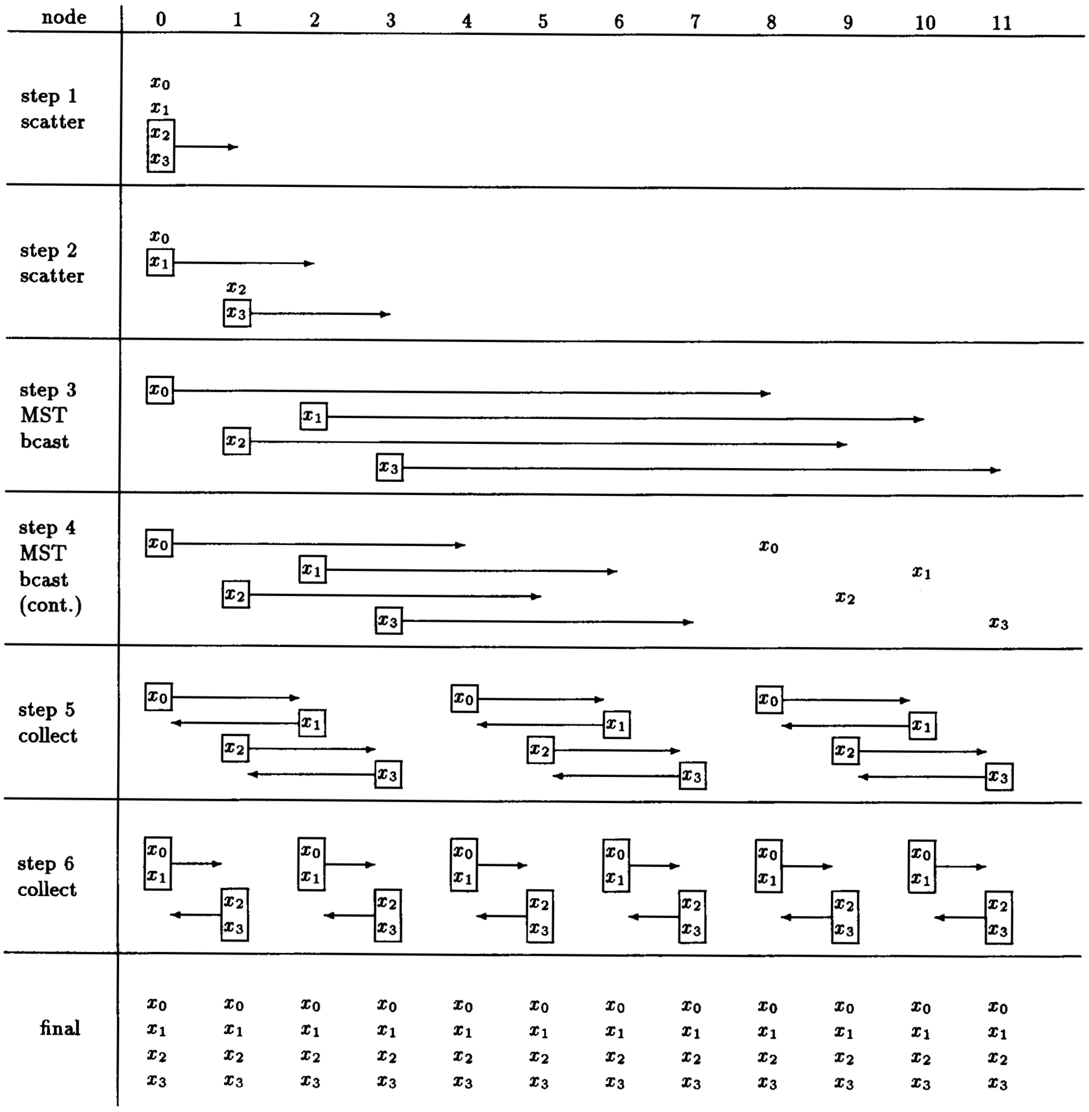

Figure 1: Broadcast hybrid: In Step 1 and 2, scatters within subgroups of two nodes are performed. Next, separate MST broadcasts within subgroups of three nodes are performed in Step 3 and 4. Finally, simultaneous collects within subgroups of two complete the broadcast. Except for Step 1 and 6, limited 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.

Broadcast on a Linear Array of 30 nodes

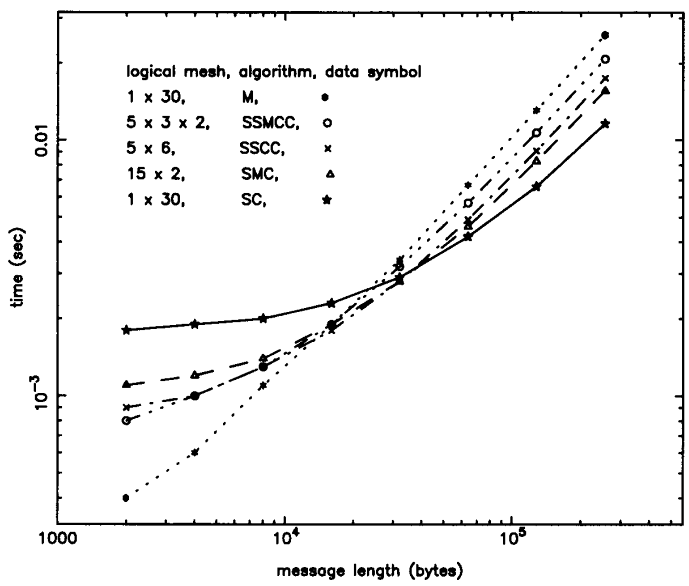

Figure 2: Predicted performance of some of the broadcast hybrids for a linear array of 30 nodes enumerated in Table 2, using machine parameters similar to those of the Paragon. While the benefits of these hybrids are marginal for 30 nodes, this figure provides a representative illustration of the benefits that can be expected from the use of hybrids.

We note that the different choices of hybrids for broadcast and other collective operations can be generated by the template given in Fig. 3. For example, in the cast of the broadcast hybrid, the short vector algorithm is the MST broadcast, and stage 1 and stage 2 of the long vector algorithms are scatter and bucket collect, respectively. Similarly, for a collect, the short vector algorithm uses a gather followed by a MST broadcast, stage 1 of the long vector algorithm is a void operation, and stage 2 is a bucket collect. 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 may not be as effective.

We have not had a chance to fully study the theoretical aspects of choosing the optimal hybrid. This is partially due to the added complications posed by reality, which makes it more desirable to design effective heuristics than to develop theoretically optimal methods which assume conditions that do not mirror reality. For example, it is clearly beneficial to choose long vector primitives early during a hybrid, since they reduce the length of the message, thereby reducing network conflicts during the later stages of the hybrid. This can indeed be proven to be optimal. It is less clear whether to have the earlier stages involve communication between nearby nodes, as we do in our example for the broadcast, or to have the later stages involve nearer nodes. Again, one can make the argument that while the vectors are long, the hybrid should choose the localized groups in an effort to reduce network conflicts. By the time the later stages of the hybrid are executed, the vector lengths are shorter and hence the effect of network conflict is less.

## 7 Applying techniques to the Paragon

### 7.1 Further issues

A number of issues complicate using the simple approach discussed so far on a real machine. We will use the Intel Paragon as an example. First of

```
Template
if $p = 1$ or $n$ small
    short vector algorithm
else
    view nodes as $r \times c$ logical mesh
    long vector alg. stage 1 within rows
    call this algorithm recursively for
        pieces within columns
    long vector alg. stage 2 within rows
fi
```

Figure 3: Template for generating hybrid algorithms on linear arrays

all, the Paragon is a mesh, not a linear array. It pays to take this into account when choosing logical meshes. In particular, long vector primitives can perform within physical rows and columns, which reduces the latency for bucket based primitives from $(p-1)\alpha$ to $(r+c-2)\alpha$, where $r$ and $c$ are the physical mesh dimensions. On meshes, the use of long vector primitives can be enhanced by alternating directions within the mesh [3]. Also, the model for communication is considerably more complex: details of how messages are sent greatly affects the parameters in the model, $\alpha$ and $\beta$. Furthermore, there is an excess of bandwidth on each link of the network compared to the bandwidth from a node to the network. As a result, each link can in effect accommodate more than one message simultaneously without penalty.

Incorporating the above observations, we have refined our techniques to the point where very good hybrids can be obtained as long as good short and long vector primitives are provided as well as an accurate model for their expense as a function of message length and number of interleaving subgroups. Further details for this go beyond the scope of this paper.

### 7.2 Experimental Results

In this section, we present representative 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 collective operations executed on two different physical mesh partitions: the case of a collect on a 16 × 32 mesh, which has the considerable advantage of power-of-two dimensions, and the case of a broadcast on a 15 × 30 mesh, which deviates significantly from a power-of-two mesh. The results are given in Fig. 4.

| Operation | length (bytes) | NX (sec) | InterCom (sec) | ratio |
|---|---|---|---|---|
| Broadcast | 8 | 0.0012 | 0.0013 | **0.92** |
| | 64 K | 0.031 | 0.012 | **2.58** |
| | 1M | 0.51 | 0.10 | **5.10** |
| Collect X (known lengths) | 8 | 0.27 | 0.0035 | **77.1** |
| | 64 K | 0.32 | 0.013 | **24.6** |
| | 1M | 0.94 | 0.075 | **12.5** |
| Global Sum | 8 | 0.0036 | 0.0041 | **0.88** |
| | 64 K | 0.17 | 0.024 | **7.10** |
| | 1M | 2.72 | 0.17 | **16.0** |

Table 3: Time (in sec.) for the representative collective communications. All results are for a $16 \times 32$ mesh of nodes.

To highlight the benefits of the hybrid algorithms, we present data in Table 3 for three vector lengths that shows the time of the different algorithms for short, medium, and long vectors. Notice that often better than an order of magnitude improvement is observed over the current implementations that are part of the NX operating system for the Intel Paragon. While the performance is in general considerably better than the NX collective communications calls, for short vectors, the iCC library, developed as part of the InterCom project, performs somewhat worse. This is due to the fact that the short vector primitives are implemented using recursive function calls, which carry a measurable overhead.

## 8 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 [7] 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. They are also more succeptible to timing irregulaties resulting from the more complex operating systems of current generation machines. Indeed, such theoretically superior algorithms are often out-

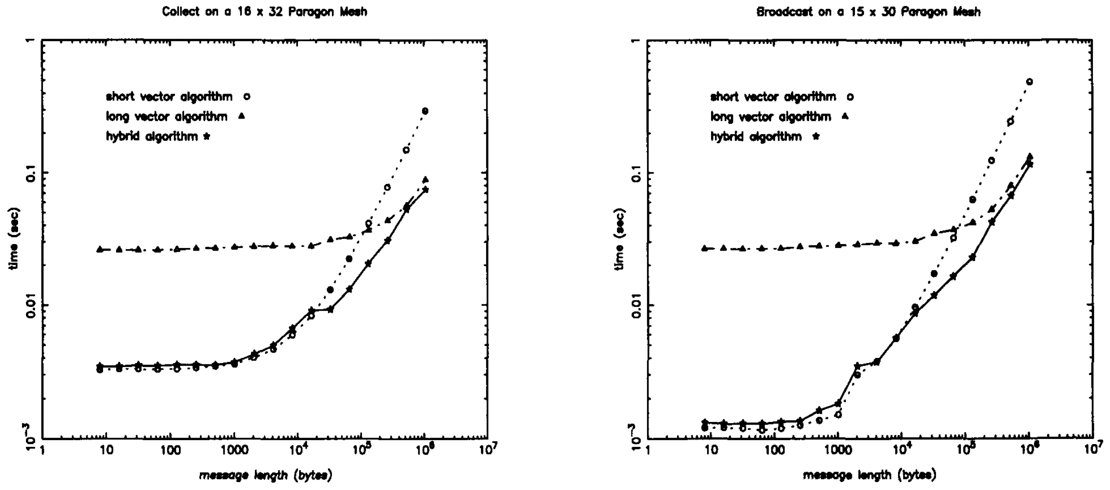

Figure 4: Performance of representative hybrid collective communication operations on the Intel Paragon. *left:* Collect on a 16 × 32 physical mesh. *right:* Broadcast on a 15 × 30 physical mesh.

performed 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 *theoretically* optimal approaches inappropriate for general library development.

## 9 Group Communication

Over the past decade, it has become increasingly obvious that many applications require parallel implementations formulated in terms of computation and communication within node *groups* (e.g. rows and columns of a *logical* mesh). Until recently, application programmers have been forced to write such operations themselves. Fortunately, this need was recognized by the MPI effort, which provides facilities for creating groups, as well as performing collective communications involving only nodes within the same group.

Notice that our hybrids themselves require communication within groups. For example, the broadcast hybrid requires group versions of the collect, scatter and minimum spanning broadcast. To perform a ring collect within a column, for example, we simply called the ring collect primitive with an array containing the ID's of the processors comprising a column in the mesh. The ring collect routine would treat those processors as a group of contiguous nodes numbered 0 to $r - 1$, using the group array to provide the logical-to-physical mapping. Thus, in the process of creating our library, we also created the mechanism necessary to support the group abstraction. As a result, it was relatively straightforward for us to provide a MPI-like interface to our collective communications, thereby extending our high-performance hybrid algorithms to group collective communication. Performance for group operations is maintained by extracting information about the *physical* layout of a user-specified group. In cases where a group comprises a physical rectangular submesh, the same row- and column-based techniques are used as in the whole-mesh operations. When a group is unstructured or its structure cannot be ascertained, it is treated as though it were a linear array.

## 10 Obtaining Documentation and Using the InterCom Library

To obtain the InterCom library for Intel systems, contact `intercom@cs.utexas.edu`. Manuals and other information regarding the InterCom library are available via anonymous `ftp` from `cs.utexas.edu` in the directory `pub/rvdg/iCC`. A number of papers that inspired the development of this library can be found in the directory `pub/rvdg`. To use the InterCom library, obtain the InterCom collective operations calling sequences from the manual, introduce them into your Fortran or C program, and simply link the InterCom library into your program. For example, to compile a Fortran program `main.f` for the Paragon and link the InterCom library to it, execute the following -

```
if77 -o main main.f iCC.<vers>.a -nx -lkmath
```

(where `<vers>` indicates the version number of the InterCom library which is to be linked). An analogous command is used for a C program. The InterCom library also contains a direct NX interface, which converts all NX collective opertions to InterCom collective operations (except the NX broadcast operation, `csend(-1)`, which must be changed explicitly to the InterCom operation `iCC_bcast()`). To link in the InterCom library using the NX interface, link in `NXtoiCC.<vers>.a` instead of `iCC.<vers>.a`.

## 11 Conclusion

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.

To port the library between platforms or 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, in addition to changing only the message send and receive calls to the native point-to-point communication library. Indeed, we ported the library from the original version, which was designed for the Touchstone Delta, to the Paragon by changing only these parameters, tuning it in a matter of hours.

In addition to the Paragon and Delta versions, we also have a version tuned for the 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 planned.

A version of the library that lacks the group interface was released in summer 1994. We expect to release an updated version of the library that allows for group collective communication in fall 1994.

## Acknowledgements

This research was performed in part using the Intel Paragon System and the Intel Touchstone Delta 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. Funding for this project was provided in part by the Intel Research Council, Intel Supercomputer Systems Division, and the University of Texas Center for High Performance Computing.

## References

[1] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. van de Geijn and J. Watts. Interprocessor Collective Communication Library (InterCom). *Proceedings of Scalable High Performance Computing Conference,* pg. 357–364, IEEE Computer Society Press, Knoxville, TN, May 23–24, 1994.

[2] M. Barnett, R. Littlefield, D.G. Payne and R. van de Geijn. Efficient Communication Primitives on Mesh Architectures with Hardware Routing. *Sixth SIAM Conference on Parallel Processing for Scientific Computing,* Norfolk, VA, Mar. 22–24, 1993.

[3] 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, Apr. 13–16, 1993.

[4] M. Barnett, D. Payne and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. University of Texas Department of Computer Science TR-91-38, Dec. 1991.

[5] 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. (Currently University of Texas Department of Computer Sciences TR-93-24.)

[6] J.-C. Bermond, P. Michallon and D. Trystram. Broadcasting in Wraparound Meshes with Parallel Monodirectional Links. *Parallel Computing,* 18(6):639–648, June 1992.

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

[8] S. L. Lillevik. The Touchstone 30 Gigaflop Delta Prototype *Sixth Distributed Memory Computing Conference Proceedings*, pg. 671–677, IEEE Computer Society Press, 1991.

[9] R. Littlefield. Characterizing and Tuning Communications Perfomance on the Touchstone Delta and iPSC/860. *Proceedings of the 1992 Intel User's Group Meeting,* Dallas, Texas, Oct. 4–7, 1992.

[10] L. M. Ni and P. K McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer,* 26(2):62–76, Feb. 1993.

[11] Y. Saad and M. H. Schultz. Data Communication in Parallel Architectures. *Parallel Computing,* 11(2):131–150, Aug. 1989.

[12] S. R. Seidel. Broadcasting on Linear Arrays and Meshes. Oak Ridge National Laboratory Technical Report ORNL/TM-12356, Mar. 1993.

[13] M. Simmen. Comments on Broadcast Algorithms for Two-Dimensional Grids *Parallel Computing,* 17(1):109–112, Apr. 1991.

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

[15] R. van de Geijn and J. Watts. A Pipelined Broadcast for Multidimensional Meshes. *Parallel Processing Letters*, to appear.

[16] D. W. Walker. The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers. *Parallel Computing*, Apr. 1994. (Up to date information about the MPI standard is available from `netlib`, directory `mpi`.)