Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance

Nicholas T. Karonis
High–Performance Computing Laboratory
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115
karonis@niu.edu

Bronis R. de Supinski
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
bronis@llnl.gov

Ian Foster
Argonne National Laboratory
Argonne, IL 60439
The University of Chicago
Chicago, IL 60637
foster@mcs.anl.gov

William Gropp
Ewing Lusk
John Bresnahan
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{*gropp,lusk,bresnaha*}*@mcs.anl.gov*

Abstract

The efficient implementation of collective communication operations has received much attention. Initial efforts modeled network communication and produced "optimal" trees based on those models. However, the models used by these initial efforts assumed equal point-to-point latencies between any two processes. This assumption is violated in heterogeneous systems such as clusters of SMPs and wide-area "computational grids", and as a result, collective operations that utilize the trees generated by these models perform suboptimally. In response, more recent work has focused on creating topology-aware *trees for collective operations that minimize communication across slower channels (e.g., a wide-area network). While these efforts have significant communication benefits, they all limit their view of the network to only two layers. We present a strategy based upon a* multilayer *view of the network. By creating* multilevel topology trees *we take advantage of communication cost differences at every level in the network. We used this strategy to implement topology-aware versions of several MPI collective operations in MPICH-G, the Globus-enabled version of the popular MPICH implementation of the MPI standard. Using information about topology discovered by Globus, we construct these topology-aware trees automatically during execution, thus freeing the MPI application programmer from having to write special files or functions to describe the topology to the MPICH library. We present results demonstrating the advantages of our multilevel approach by comparing it to the default (topology-unaware) implementation provided by MPICH and a topology-aware two-layer implementation.*

1. Introduction

The problem of building "optimal" trees for collective operations has received much attention over recent years. The telephone model, which assumes that send and receive times are equal, implies that the optimal broadcast algorithm uses a binomial tree. Under models that expand the telephone model to account for message latency, such as the postal [1] or LogP [5] models, the communication topology of an optimal broadcast algorithm becomes a generalized Fibonacci tree. All these approaches construct optimal trees for collective operations by first modeling the communication characteristics of a network with a set of parameters and then building the optimal trees based on parameter values and their model.

Underlying all of this work is the assumption that the communication times between all process pairs in

the computation are equal. While this is a reasonable approximation when the entire computation is performed on a single machine, it is not reasonable when the computation is executed on a cluster of symmetric multiprocessors (SMPs) in a local-area network or worse, in a *computational grid* [10, 6] environment, in which multiple parallel computers are connected by local area, campus-area, or even wide-area networks. Rapid improvements in network performance have engendered considerable interest in parallel computing in the latter context, as evidenced by such experiments and initiatives as the I-WAY [8], National Technology Grid [17], and Information Power Grid [14].

Under these circumstances the trees produced by the conventional models perform suboptimally. In such heterogeneous environments, communication costs at different levels in the hierarchy can differ by an order of magnitude or more. In these situations, *topology-aware* algorithms can achieve dramatic improvements in performance. For example, in the case of N processors distributed into two clusters, a traditional reduction algorithm may generate O(log N) intercluster messages, while a topology-aware algorithm generates only 1, for a cost saving of a factor of O(log N) if intercluster message costs dominate.

Previous work [13, 15] has demonstrated that topology-aware collective operations can indeed reduce communication costs by reducing the amount of communication performed over slow channels. However, this work limited the depth of network stratification to only two levels: other processors are either near or far. In this paper, we present new algorithms that allow collective operations to exploit knowledge concerning the structure of a multilevel network, in which the neighbors are processors that are categorized according to their expected point-to-point communication characteristics.

In order to permit experimental studies, we have implemented our algorithms for three of the collective operations supported by the Message Passing Interface (MPI) standard: `MPI_Bcast`, `MPI_Reduce`, and `MPI_Barrier`. We use the MPICH-G [7, 9] version of the popular MPICH implementation [12] of the MPI standard [16], which uses services provided by the Globus toolkit to support execution in heterogeneous and distributed environments. This use of MPICH-G enables experimentation within realistic wide-area environments that would not otherwise be easily accessible. In addition, we are able to use information provided by the Globus system to determine a particular computer system's topology automatically.

In the sections that follow, we briefly discuss recent topology-aware efforts. Next, we describe our multilevel topology approach. We present experimental results that illustrate the benefits of our multilevel approach by comparing it to the topology-*un*aware implementation currently distributed with MPICH and to MagPIe [15], one of the topology-aware two-level implementations of collective operations. We conclude with a discussion of future work.

2. Recent Work

Recent efforts have focused on creating "optimal" trees for collective operations where point-to-point communications are not necessarily equal between any two processes. Husbands and Hoe present MPI-StarT [13], an MPI implementation for a cluster of SMPs interconnected by a high-performance interconnect. They reported significant improvements after modifying the MPICH broadcast algorithm, which uses binomial trees. Their modifications use information that describes their cluster topology by minimizing intercluster communication during collective operations. MagPIe [15] is another MPI system designed to construct collective operation trees in heterogeneous communication environments. MagPIe recognizes a two-layer communication network that distinguishes between local- and wide-area communication. By minimizing wide-area communication, much in the same way MPI-StarT minimizes intercluster communication, MagPIe has seen significant improvements in all the MPI collective operations.

Both efforts have produced impressive results and clearly demonstrate that there are significant advantages to implementing collective operations in a topology-aware manner. However, both limit their view of the network to only two layers; MPI-StarT distinguishes between intra- and intercluster communication within the same local-area, and MagPIe distinguishes between local- and wide-area communication. There are opportunities for further optimization by using trees that stratify the network deeper than two layers.

3. Multilevel Topology Aware Approach

Figure 1 depicts an MPI application involving 32 processes distributed over three machines located at Lawrence Livermore National Laboratory (LLNL) and Argonne National Laboratory (ANL). At LLNL we depict 16 processes on the IBM SP equipped with 4-way SMP nodes, 4 processes per SMP node. At ANL we depict 8 processes on the SGI Origin2000 and another 8 processes on the IBM SP. The slowest communication is between sites, which use TCP over the wide-area

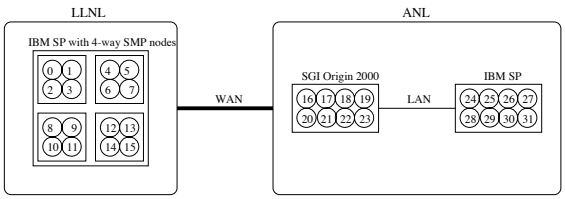

Figure 1. An example of a grid computation involving 16 processes on one IBM SP at LLNL and another 16 processes distributed evenly across an IBM SP and an SGI Origin2000 at ANL.

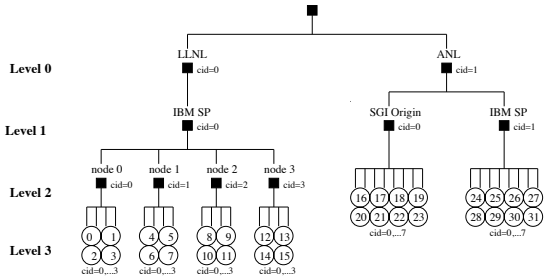

Figure 2. A multilevel view of a grid computation annotated with cluster ids at each level.

network, with faster communication between the Origin2000 and the IBM SP, which use TCP over their local-area network, and the fastest communication, of course, within each machine. To take full advantage of the communication cost differences at all levels, we depict this configuration with the *multilevel topology tree* in Figure 2.

Each level of the tree represents a different communication channel: TCP over a wide-area network at level 0, TCP over a local-area network at level 1, and so forth, with the slowest channel at the top. Processes are grouped into subtrees or *clusters* based on their ability to communicate with each other relative to a particular level. For example, processes 0–15 all have a common ancestor at level 0 because they are all at the same site. We represent the tree by assigning to each process a *depth* and *cluster identifier*, or *cluster id*, at each level. The cluster ids for each process appear in Figure 2. For example, at levels 0 and 1, processes 0–15 all have a cluster id 0; and at level 2 processes 0–4 have a cluster id 0 while processes 4–7 have cluster id 1. The depth of processes 0–15 is four, while the depth of the remaining processes is only three. This reflects the fact

```
cluster[0] = copy of user level communicator;
For (i = 1; i < MPID_Depth; i++)
  MPI_Comm_split (cluster[i-1],
                  MPID_Clusterid(i),
                  0, &cluster[i]);

cluster_rank = 0;
For (i = MPID_Depth - 1; i > 0; i--)
  MPI_Comm_split (cluster[i-1], cluster_rank,
                  0, &master[i]);
  if (cluster_rank != 0)
    /* this task not a level i master */
    MPI_Comm_free (&master[i]);
  MPI_Comm_rank (cluster[i], &cluster_rank);
```

Figure 3. Pseudo-code for the hidden communicator creation algorithm.

that the IBM SP has two methods of communication: internode communication over the IBM switch, and the faster intranode communication over shared memory. Using information about network topology discovered by Nexus [11], the communication component within Globus, the depths and cluster ids are *determined automatically* during execution by MPICH-G. Users do not need to modify their application, write special functions, or describe the topology in special files.

We augmented the MPICH Abstract Device Interface (ADI) API to include two new functions (`MPID_Depth` and `MPID_Clusterid`) that query the MPICH abstract device for a process's depth and cluster id. MPICH, by relying on its underlying abstract device to determine this information, can implement topology-aware collective operations in a *device-independent* manner. We then modified the MPICH implementations of three of the collective operations; broadcast, reduction, and barrier. Below, we describe our modifications for the `MPI_Bcast` function in detail.

3.1. Topology-aware Broadcast

Our approach to providing topology-aware collective communications is a natural extension of MPICH's current implementation. Each time a new MPI communicator is created, the current release of MPICH creates a hidden communicator that is used for all collective communication; we use the depth and cluster ids to create a set of hidden communicators that capture the communication topology of the system. These communicators allow our collective communication routines to minimize use of the slowest levels of the communication hierarchy.

Conceptually, our set of hidden communicators is partitioned into two sets: cluster communicators and master communicators. The cluster communicators capture the clusters to which each task belongs at each level of the system. We pick one task to represent each cluster during collective communications for intercluster communication at that level; in other words, the task acts as master of the cluster. The master communicators are composed of these master tasks.

Figure 3 shows an abstract version of the algorithm that creates the hidden communicators for a new user-level communicator. First, the algorithm makes a copy of the user level communicator that serves as the level zero cluster communicator; this communicator is identical to the hidden communicator used by the current version of MPICH. Then, a loop over `MPI_Comm_split` operations creates the remaining cluster communicators, each one dividing a cluster of the previous level into its constituent clusters. Finally, a second set of `MPI_Comm_split` operations gathers a representative from each of the constituent clusters into the master communicator for that level.

Note that we discard the master communicator if the task is not rank zero in the constituent cluster. This strategy results in a single master per constituent cluster. We use the rank zero member of each of the constituent clusters as its master because every constituent cluster must have at least one member, while there is no guarantee that the discarded communicators have a representative of each constituent cluster. We discuss how we use the master communicators shortly.

The abstract version of our algorithm is not very efficient. Each call to `MPI_Comm_split` requires two collective communications: one to allocate contexts for the new communicator and one to distribute the colors (which are all zero in our calls) and keys to every task. In practice, all of the required contexts can be allocated at the same time as the context for the user-level communicators. Further, the keys required to create the cluster communicators are the cluster ids applicable to each task. We must gather these because we assume a task cannot deduce the cluster ids of a remote node. However, these can be gathered once, during `MPI_Init`. Finally, each task can independently deduce the keys used to create the master communicators from the full mapping of the communication topology that results from gathering all of the cluster ids.

Our implementation performs each collective communication over the master communicators successively. This approach makes optimizing the collective communication at each level an orthogonal issue. As a result, we can easily plug in an algorithm that exploits specific capabilities of a given level, such as writing the message just once into a shared memory buffer or sending the message over a broadcast medium such as ethernet [4].

```
MPI_Bcast(buffer, count, datatype, 0, comm)
  For (i = 1; i < MPID_Depth; i++)
    MPI_Comm_rank (cluster[i], &cluster_rank);
    if (cluster_rank == 0)
      MPIR_LevelBcast (buffer, count, datatype,
                       0, master[i], i);
```

Figure 4. The multilevel broadcast algorithm for rank 0 root.

```
MPI_Bcast (buffer, count, datatype, root, comm)
  MPI_Comm_rank (comm, &rank);
  For (i = 1; i < MPID_Depth; i++)
    MPI_Comm_rank (cluster[i], &cluster_rank);
    if ((cluster_rank == 0
           && !(root is_member cluster[i]))
        || (rank == root))
      if (root is_member cluster[i-1])
        master_root = Rank_of_master (root, i);
        if (root is_member master[i])
          curr_master = master[i];
        else
          Substitute_root_for_master (root,
                                      master_root,
                                        master[i],
                                     &curr_master);
      else
        master_root = 0;
        curr_master = master[i];
      MPIR_LevelBcast (buffer, count, datatype,
                  master_root, curr_master, i);
```

Figure 5. The multilevel broadcast algorithm for arbitrary root.

We now present details of our broadcast implementation; implementations of other collective communications are similar. Figure 4 shows the algorithm that we use if task zero is the root of the broadcast. In this case, we can simply use the broadcast algorithm appropriate for the communication method of each of the master communicators. We use the same algorithm with a small modification if task zero is not the root. In this case, no change is necessary at any level for which the `cluster_rank` of the root is zero. Similarly, no change is necessary for a task at level i if the root is not in its level $i-1$ cluster. If the root is in the same level $i-1$ cluster and is a level i master, we use it as the root of the broadcast over master[i]; if the root

```
For (each message size M)
  MPI_Barrier(MPI_COMM_WORLD)
  if (MPI_COMM_WORLD rank == 0)
    t0 = get_time()
  For (r = 0; r < Nprocs; r ++)
    MPI_Bcast(root=r to MPI_COMM_WORLD
              message size M)
    ack_barrier()
  if (MPI_COMM_WORLD rank == 0)
    t1 = get_time()
    report message size M, time t1-t0
```

Figure 6. The broadcast timing application.

is not its level i master, we first substitute the root for its master. Figure 5 shows our complete broadcast algorithm.

4. Experimental Results

To demonstrate the advantages of our multilevel approach, we examine its effects on `MPI_Bcast`. The MPICH implementation of `MPI_Bcast` is based on binomial trees, hence in a distributed heterogeneous environment like a computational grid its performance is acutely sensitive to the distribution of the processes and the root of the broadcast. For example, in an application using $P = 2^k$ processes distributed evenly across $C = 2^i, 0 \leq i \leq k$ clusters, a broadcast implemented using a binomial tree propagates the message down its longest path using at least $log_2 C$ intercluster messages and $log_2 \frac{P}{C}$ intracluster messages. In contrast, under certain intercluster network performance conditions described by Bar-Noy and Kipnis in their postal model, our multilevel method could be used to send 1 intercluster message and $log_2 \frac{P}{C}$ intracluster messages. Assuming an intercluster latency l_s sec and bandwidth b_s Kb/sec; and an intracluster latency l_f sec and bandwidth b_f Kb/sec, broadcasting a message of N Kb using the binomial tree conservatively takes $O((logC)(l_s + \frac{N}{b_s}) + (log\frac{P}{C})(l_f + \frac{N}{b_f}))$ while broadcasting the same message using our multilevel method takes only $O((l_s + \frac{N}{b_s}) + (log\frac{P}{C})(l_f + \frac{N}{b_f}))$.

We wrote a small MPI application (depicted in Figure 6) that times the broadcasts of messages of increasing size. In an attempt to represent a broadcast with an arbitrary root, we timed how long it would take to broadcast each message of size M as each process in `MPI_COMM_WORLD` took its turn as the root. Also, in order to eliminate any potential pipelining that might occur between consecutive broadcasts, we inserted a barrier (`ack_barrier()`) after each broadcast in which all

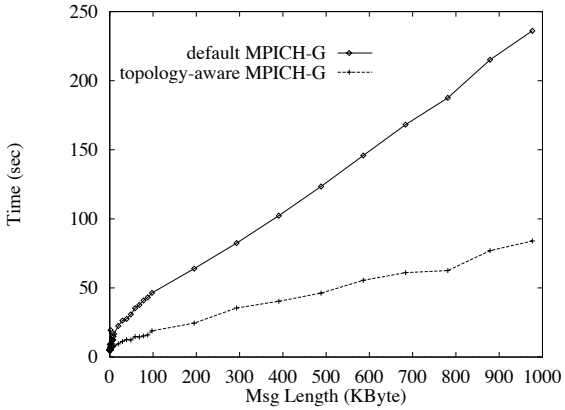

Figure 7. Original MPICH broadcast vs. topology-aware MPICH broadcast across a *wide-area network* running 16 processes on the IBM SP at SDSC, 16 processes on the IBM SP at ANL, and 16 processes on the SGI Origin2000 at ANL.

processes other than rank 0 `MPI_Send` an ACK message to process 0 and then wait to `MPI_Recv` a GO message. Process 0, after `MPI_Recv`'ing the ACK message from all the other processes, `MPI_Send`'s a GO message to each of the other processes, one at a time. We chose to write our own barrier rather than calling `MPI_Barrier` because we have reimplemented `MPI_Barrier` to reflect multilevel topology and we wanted these tests to reflect the differences only in the broadcast implementations.

We conducted three experiments, each time running the MPICH-G application depicted in Figure 6 on various combinations of three computers; the IBM SP at the San Diego Supercomputer Center (SDSC-SP) and the IBM SP (ANL-SP) and SGI Origin200 (ANL-O2K) at Argonne National Laboratory.

In our first experiment we measured the benefits of minimizing wide-area communication. We ran the application twice, first using MPICH-G 1.1.2 and then using our topology-aware version of MPICH-G 1.1.2, each time using 16 processes on each of the three computers. The performance of each implementation is depicted in Figure 7.

In our second experiment we measured the benefits of minimizing local-area communication. Again, we ran the application twice, this time using 16 processes on each of two machines located at ANL (ANL-SP and ANL-O2K). The performance of each implementation is depicted in Figure 8.

In our third and final experiment we measured the combined benefits of minimizing local- and wide-area

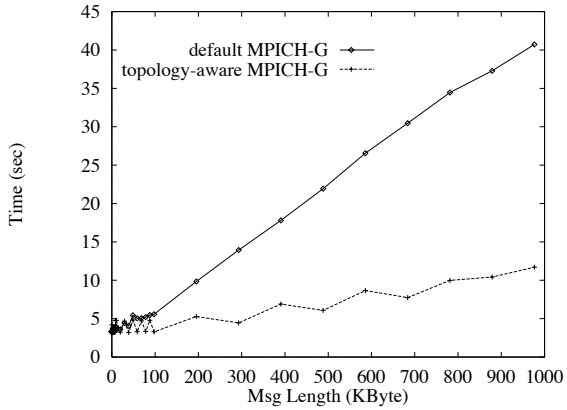

Figure 8. Original MPICH broadcast vs. topology-aware MPICH broadcast across a *local-area network* running 16 processes on the IBM SP and 16 processes on the SGI Origin2000 at ANL.

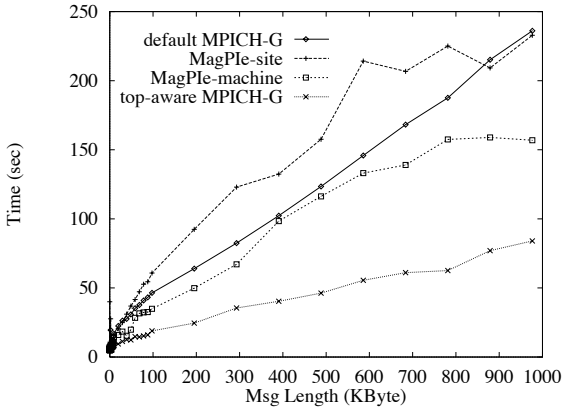

Figure 9. Original MPICH broadcast vs. topology-aware MPICH broadcast vs. MagPIe broadcast across *local- and wide-area networks* running 16 processes on the IBM SP at SDSC and 16 processes on each the IBM SP and SGI Origin2000 at ANL.

communication. We again compare our multilevel topology approach to the binomial tree provided by MPICH and include comparisons to the 2-level approach provided by MagPIe. In this experiment we ran the application four times, each time using 16 processes on each of the three computers. These results are depicted in Figure 9. The curves labeled "MagPIe-machine" and "MagPIe-site" represent two runs using MagPIe version 2.0.1 (as of July 1999), each time with a different cluster definition. Described briefly, MagPIe implements its 2-level approach by asking the application programmer to supply functions that define the 2-level cluster topology. In our first MagPIe run ("MagPIe-machine") we defined three clusters, one for each computer, of 16 processes each. In our second MagPIe run ("MagPIe-site") we defined two clusters; an ANL cluster comprised of the two ANL machines having 32 processes and an SDSC cluster comprised of the SDSC-SP having only 16 processes.

These results demonstrate the advantages of a multilevel view of the network. Figure 7 shows enhanced performance when wide-area communication is minimized, and Figure 8 shows that once the message has reached a particular site, there are even more benefits in minimizing local-area communication. Figure 9 reinforces these two independent notions by showing there are significant benefits to the multilevel approach when compared to a simple binomial tree and even when compared to a 2-level approach as implemented by MagPIe. A multilevel view of the network allows an application to avoid slower channels *at each level.* In our experiments, the broadcast is optimized by sending one message across the wide-area (level 0), then one message across the local-area (level 2), and then many messages within each computer (level 3).

5. Future Work

Our initial work in augmenting the MPICH ADI API and the Globus device has allowed us to modify three of the MPI collective operations (`MPI_Bcast`, `MPI_Reduce`, and `MPI_Barrier`) in MPICH. Encouraged by our initial results, we plan to ugrade the remainder of the MPICH collective operations in a similar manner.

Our general strategy implements a collective operation by first stratifying the network into multiple levels and then minimizing the communication across the slowest channels. However, in doing so we may encounter a tree that has multiple siblings at a particular level, for example many sites connected across the wide-area network or many machines at a particular site. When this happens, we implement the collective operation at that level using the default MPICH algorithm. For example, when performing an `MPI_Bcast` in a computation involving many sites, the first phase of our strategy would require a broadcast from the root node to a representative node at each site. In this situation, the default MPICH binomial tree would be used to perform that broadcast. Unfortunately,

a binomial tree is not always the best choice. In the postal model Bar-Noy and Kipnis show that the shape of a collective operation tree depends heavily on the point-to-point communication characteristics of the send/receive primitives upon which it is implemented. Their model incorporates a latency parameter $\lambda \geq 1$. They show that for low latencies, for example, communication within a single machine, the optimal broadcast tree is a binomial tree, but for higher latencies, for example, communication across a wide-area network, the optimal broadcast tree becomes flatter. We intend to investigate ways to select better, if not optimal, collective operation trees by choosing those that respect the different communication characteristics at each level of our multilevel view.

We believe the topology information detected by MPICH devices can be valuable to some MPI applications. Two examples of MPI computational grid computations are Cactus [3], an astrophysics code for solving general relativity problems, and OVERFLOW [2], a computational fluid dynamics code. We plan to investigate solutions, perhaps through attributes associated with MPI communicators, that make the topology information available at the MPI application layer.

6. Summary

As grid computations become increasingly prevalent, the need for topology-aware collective operations also increases. Through the creation and use of *hidden communicators* we have a version of MPICH-G that implements three collective operations in a topology-aware manner and have shown, at least for `MPI_Bcast`, that when compared to the binomial tree provided by MPICH and the 2-level approach provided by MagPIe there are significant advantages to executing collective operations using a multilevel view of the network. By modifying MPICH such that topology information is provided by the abstract device layer and the collective operations are implemented at the MPICH layer based on that information, we have provided a means by which any MPICH device may take advantage of the new topology-aware MPICH algorithms by simply providing two new functions.

Using information about topology discovered by Globus, MPICH-G users will find that the topology information is *automatically detected*, thus enabling their MPI applications to enjoy the benefits of these optimized collective operations without requiring code modifications, special functions, or special files.

Finally, this work has provided a foundation that will eventually allow MPICH-G applications to discover and use the topology information detected at the MPICH device layer.

Acknowledgments

We thank Joseph Link at Northern Illinois University for his help in running our experiments. We thank also Stuart Martin at Argonne National Laboratory for developing the Multi-site Advanced Reservation System (MARS) and the staffs at Argonne and the San Diego Supercomputer Center for their cooperation in using MARS. We found MARS to be a valuable, if not essential, tool in co-scheduling our experiments.

This work was supported in part by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the U.S. Department of Energy, under contract DE-FC02-99ER25398; by NASA's Information Power Grid program; and under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48; Release No. UCRL-JC-134070.

References

[1] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, June 1992.

[2] S. Barnard, R. Biswas, S. Saini, R. V. der Wijngaart, M. Yarrow, L. Zechter, I. Foster, and O. Larsson. Large-scale distributed computational fluid dynamics on the information power grid using globus. In *Frontiers '99: The 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 60–67, Feb. 1999.

[3] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, and P. Walker. Numerical relativity in a distributed environment. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, Apr. 1999.

[4] J. Bruck, D. Dolev, C. T. Ho, M. C. Rozu, and R. Strong. Efficient message passing interface (MPI) for parallel computing on clusters of workstations. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 64–73, June 1995.

[5] D. E. Culler, R. Karp, D. A. Patterson, A. S. K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel compuation. In *Proceedings of the 4th SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pages 1–12, May 1993.

[6] I. Foster and C. Kesselman, eds. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. A wide-area implementation of the Message Passing Interface. *Parallel Computing*, 24(12):1735–1749, 1998.

[8] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY metacomputing experiment. *Concurrency: Practice & Experience*, 10(7):567–581, 1998.

[9] I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of Supercomputing '98*, Nov. 1998.

[10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[11] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[12] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22:789–828, 1996.

[13] P. Husbands and J. C. Hoe. MPI-StarT: Delivering network performance to numerical applications. In *Proceedings of Supercomputing '98*, Nov. 1998.

[14] W. E. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. IEEE Computer Society Press, 1999.

[15] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MAGPIE: MPI's collective communcation operations for clustered wide area systems. In *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1999.

[16] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994.

[17] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett. From the I-WAY to the National Technology Grid. *Communications of the ACM*, 40(11):50–61, 1997.