MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems

Thilo Kielmann Rutger F. H. Hofman Henri E. Bal Aske Plaat Raoul A. F. Bhoedjang

Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

kielmann@cs.vu.nl rutger@cs.vu.nl bal@cs.vu.nl aske@cs.vu.nl raoul@cs.vu.nl

http://www.cs.vu.nl/albatross/

Abstract

Writing parallel applications for computational grids is a challenging task. To achieve good performance, algorithms designed for local area networks must be adapted to the differences in link speeds. An important class of algorithms are collective operations, such as broadcast and reduce. We have developed MAGPIE, a library of collective communication operations optimized for wide area systems. MAGPIE's algorithms send the minimal amount of data over the slow wide area links, and only incur a single wide area latency. Using our system, existing MPI applications can be run unmodified on geographically distributed systems. On moderate cluster sizes, using a wide area latency of 10 milliseconds and a bandwidth of 1 MByte/s, MAGPIE executes operations up to 10 times faster than MPICH, a widely used MPI implementation; application kernels improve by up to a factor of 4. Due to the structure of our algorithms, MAGPIE's advantage increases for higher wide area latencies.

1 Introduction

The emergence of computational grids [16] makes it feasible to run parallel programs on large-scale, geographically distributed computer systems. Early performance experiments with such systems show promising results for several parallel applications [3, 13, 32]. Unfortunately, writing parallel applications for computational grids is a challenging task. In particular, programmers may have to restructure their programs to deal with differences in processor and network speeds in order to minimize traffic over slow wide area links. This requirement complicates the programming of grid applications.

In this paper, we investigate a possible solution. The idea is to create a library of MPI's *collective communication* primitives that is optimized for wide area systems. The MPI message passing library [29] is widely used for high performance scientific programs. In addition to the basic send and receive primitives, it defines fourteen different collective operations (such as barrier, broadcast, reduce, and gather) which can be optimized for wide area systems. In this way, the programmer can use a familiar programming model, and the details of the wide area system are hidden completely.

We have implemented this idea in a new MPI library, called MAGPIE.[1] This library is based on MPICH, a widely used MPI implementation, but the collective communication primitives use new algorithms, optimized for wide area systems. The basic assumption is that the wide area system is hierarchically structured. We expect a computational grid to consist of many parallel computers connected by wide area networks. We use the term *cluster* to denote a single parallel machine (which can be a network of workstations, an MPP, or an SMP). All nodes within a cluster are connected by a high-speed interconnect that is orders of magnitude faster than the wide area interconnect between clusters.

Collective communication algorithms are usually designed for local area networks, which have a low latency. Wide area systems however, have a high latency (and a lower bandwidth) and the performance of collective communication operations is dominated by the traffic over the wide area links. Thus, our algorithms are designed to reduce traffic over the slow links, resulting in a markedly different communication structure.

Our algorithms are *wide area optimal* in that an operation incurs only a single wide area latency, and every data item is sent at most once across each wide area link. The performance improvements of MAGPIE over MPICH are substantial, and depend on cluster topology, latency, and bandwidth. Speedups of a factor of 3 or 4 are typical, even for a wide area latency of 10 milliseconds. With 8 clusters, MAGPIE's AllGather runs 8 times faster than MPICH's algorithm. For larger latencies the advantage will increase, because MAGPIE's algorithms are optimized for long latency interconnects.

With MAGPIE, MPI programs can use collective communication operations efficiently and transparently on a hierarchical interconnect (such as a metacomputer). Not a single line of application code has to be changed to use the MAGPIE algorithms. We present algorithms for all 14 collective operations of MPI that are wide area optimal: the algorithms transfer the minimal amount of data over the slow links of a multilayer interconnect, and incur only a single wide area latency. (For operations that use reduction, wide area optimality requires associative operators.)

Our measurements have been performed on a system with dedicated wide area links between all clusters with a constant latency and bandwidth. On interconnects with varying speeds, such as the Internet, the wide area optimality of our algorithms guarantees that the latency that is incurred is the longest wide area latency of the interconnect.

In the rest of the paper we describe the design of our collective communication algorithms (Section 2) and the MAGPIE library that implements these algorithms (Section 3). We present measurements and discuss the performance of several MPI applications (Section 4). Section 5 discusses related work. Section 6 concludes.

[1] A magpie is a black-and-white bird that flies over wide areas to collect things.

2 Algorithm Design

We define the completion time of a collective operation as the moment at which all processors have received all messages that belong to that operation. All our collective communication algorithms aim to minimize this completion time. (That is, we focus on operation latency rather than on operation throughput.) Intuitively, the performance of collective communication operations on a wide area system is dominated by the time spent on the wide area links; local communication plays only a minor role. In a collective operation, all processors have to communicate with each other, so the completion time certainly cannot be less than the wide area latency. In the design of our wide area algorithms, we have used the following two conditions:

1. Every sender-receiver path used by an algorithm contains at most one wide area link.
2. No data item travels multiple times to the same cluster.

Condition (1) ensures that the wide area latency contributes at most once to an operation's completion time. Condition (2) prevents us from wasting precious wide area bandwidth. We call algorithms that satisfy both conditions *wide area optimal*. In Section 3 we will show that we can construct wide area optimal algorithms for all of MPI's collective operations except the global reduction operation with non-associative operators. Moreover, we will demonstrate that our implementations of wide area optimal algorithms consistently outperform implementations that are not wide area optimal.

Bernaschi et al. provide the key insight for constructing communication graphs that are wide area optimal [5]. Their communication graphs are based on the performance characteristics of the point-to-point communication primitives that are used to implement a collective operation. Specifically, they parameterize algorithms with the difference between the completion time t_s of a message send and the completion time t_r of the matching receive. We assume that messages are sent asynchronously, so a message send completes when the message has been injected into the network. Note that t_s depends on message size; t_r additionally depends on network bandwidth and latency.

Bernaschi et al. show that when the difference $t_r - t_s$ is small, the optimal broadcast tree is a binomial tree. If the difference is large, a one-level flat tree is optimal. In between both extremes exists a whole spectrum of tree shapes. The extremes are illustrated in Figure 1, which shows a flat tree at the top and a binomial tree at the bottom. Node *P*0 acts as the broadcasting sender. The other nodes are annotated with the time at which they receive their messages. We show two receive times, one for $t_r = 11$ and one for $t_r = 1000$, respectively. In both cases $t_s = 10$. With $t_r = 1000$ and $t_s = 10$ (a large difference) the one-level tree gives the best completion time (1060 versus 3000 time units), while for $t_r = 11$ and $t_s = 10$ (a small difference) the binomial tree performs best (33 versus 71 time units).

With this insight, it is clear that the different performance characteristics of local area and wide area links dictate different communication graphs for local area and wide area traffic. We use two types of graphs: an *intra cluster* graph connects all processors within a single cluster and an *inter cluster* graph connects the different clusters. To interface both graph types, we designate a *coordinator node* within each cluster.

On the wide area links, latency dominates bandwidth for small and moderate message sizes, so $t_r \gg t_s$. Consequently, the flat tree is the best interconnection structure between clusters. Notice that the flat tree also satisfies condition (1). We therefore use the flat tree as our inter cluster graph in all our collective communication algorithms. Within clusters, on the local area links, we expect $t_r - t_s$ to be relatively small, so binomial trees are almost optimal for intra

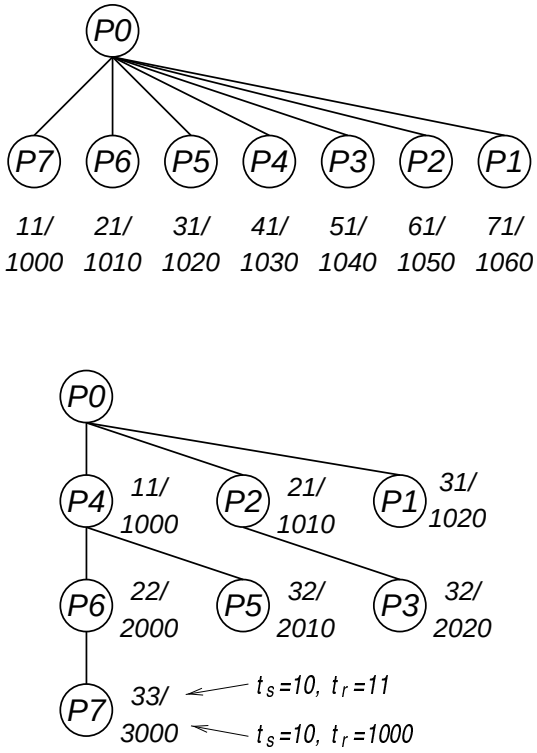


Figure 1: Optimal Broadcast Trees for $t_r \gg t_s$ (top) and $t_r \approx t_s$ (bottom)

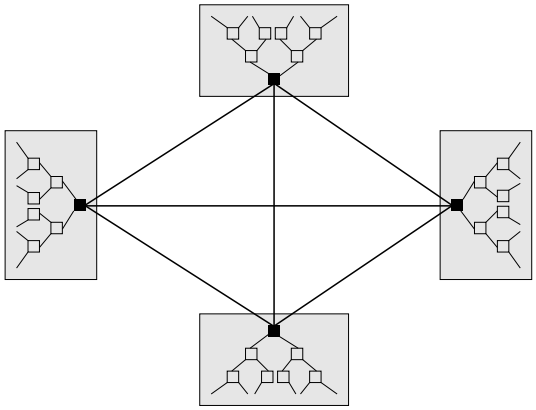
cluster graphs. (Since the performance difference is minimal, we use binary trees for ease of implementation.)

Condition (2) specifies that a data item should not be re-sent to the same cluster. How this translates to an algorithm depends on the semantics of the actual operation. It cannot be summarized in a general algorithm, but will be discussed in Section 3.

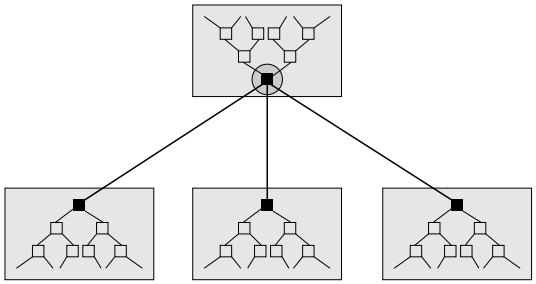
As noted above, the exact switchover point from the one-level tree to a deeper structure depends on the actual values for latency, bandwidth, message size, and the number of clusters. Computing optimal graph shapes therefore requires run time instrumentation (see, for example, [27]). In Section 3 we will show that MAGPIE's current combination of one-level trees and binary trees fits the wide area case well enough to regularly outperform MPICH's algorithms.

We are now ready to outline the general structure of our algorithms. We distinguish between two kinds of algorithms. In the *asymmetrical* algorithms one dedicated process, called the *root*, either acts as sender (in one-to-many algorithms such as a broadcast) or as receiver (in many-to-one algorithms such as a reduction). In the *symmetrical* algorithms all processes are equal peers; they all send and receive. In all asymmetrical algorithms the root also acts as the coordinator of its cluster. In all other clusters and in the symmetrical algorithms the coordinator is chosen arbitrarily. Figure 2 shows our communication graphs for symmetrical and asymmetrical operations; in the latter the specially marked node denotes the root process. In the asymmetrical algorithms, a single one-level tree connects the root cluster to the remaining clusters. In the symmetrical algorithms, each cluster has its own one-level tree that connects it to the remaining clusters.

Asymmetrical algorithms perform two macro steps. In a one-to-many operation, the root first sends its data to the coordinators



(a) symmetrical operation



(b) asymmetrical operation

Figure 2: Wide Area Optimal Communication Graphs

of the remote clusters (using the one-level tree), and second, the coordinators locally send to their nodes (using the binary trees). In the many-to-one case, the nodes first send to their coordinator and then the coordinators send to the root. Symmetrical algorithms perform three macro steps. First, nodes send to their coordinators. Second, the coordinators perform an all-to-all exchange, and third, coordinators send to their nodes. In the following, we use this basic structure for implementing wide area optimal algorithms for MPI's collective operations.

Our system is designed for a two layer (LAN/WAN) hierarchy. Some systems have additional layers (for example, a wide area grid of local area clusters of SMPs). Extra physical layers do not imply a need for extra layers in MAGPIE's structure. MAGPIE's algorithms bridge an order of magnitude difference between the top and the rest of the hierarchy. As long as this difference is large, traffic reductions further down the hierarchy are negligible, and two-layer algorithms suffice.

3 Wide Area Collective Communication Algorithms

MAGPIE implements the collective communication operations of MPI 1.1 [29], on top of MPICH 1.1, a widely used public MPI implementation [19]. MAGPIE's collective algorithms are optimized for a hierarchical interconnect. The algorithms build upon MPICH's send and receive primitives. Our experimentation system consists of four Myrinet [7] clusters of 200 MHz/128 MByte Pentium Pros, connected by a 6 Mbit/s ATM network. The machines run BSD/OS 3.0. The clusters are located at four universities in The Netherlands. The local Myrinet network is a 2D torus, the wide area ATM network is fully connected. The system, called DAS, is more fully described in [32] (and on *http://www.cs.vu.nl/das/*).

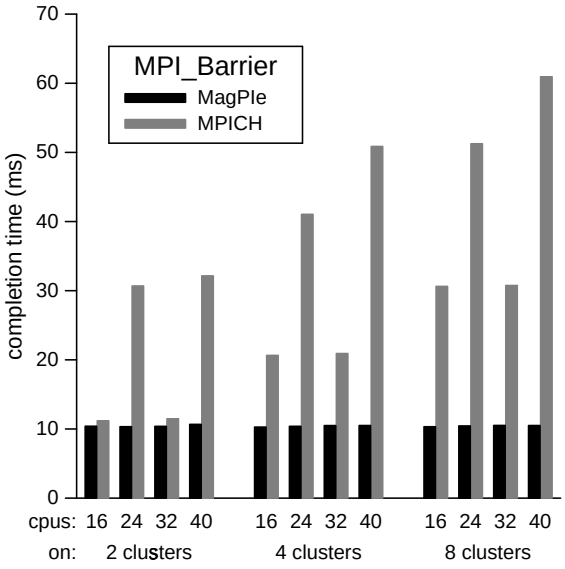


Figure 3: MPI_Barrier

MPICH has been ported to the Panda communication substrate [2] by implementing a Panda-specific MPICH device. Panda gives access to IP and Myrinet. It also informs MAGPIE how many clusters are part of the actual configuration and which process is located in which cluster. We use the LFC [6] Myrinet control program. Our MPICH port has a local sender completion time of $t_s = 8$ μs and a receiver completion time of $t_r = 20$ μs for empty messages, both measured as in [21]. The maximum bandwidth is 66 MByte/s.

One of the clusters has 128 processors, and has been set up to allow easy experimentation with different inter cluster topologies, wide area latencies, and wide area bandwidths, by adding delay loops to the networking subsystem. The instrumentation is transparent to the application. All measurements in this section are performed on this local experimentation system. (Section 4 also contains measurements on the real wide area system.) For the rest of this section, the wide area latency is set to 10 ms and the bandwidth is set to 1.0 MByte/s. (All latencies in this paper are one-way.) On most metacomputers, wide area latency will be significantly higher, and, since MAGPIE's algorithms have been optimized for long latency, the advantage of MAGPIE over MPICH will be even higher. We will now discuss the individual collective operations. We start with the barrier.

3.1 Barrier

MPI_Barrier synchronizes a group of processes; no data is exchanged in this operation. Therefore, the second of the two wide area optimality conditions is trivially satisfied, and we only need to look at the single-latency condition. Since this is a symmetrical operation, MAGPIE's algorithm consists of three steps: a local gather of empty messages to the coordinators (using a binary tree), a wide area all-to-all exchange (using a flat structure), and a local broadcast (using a binary tree). Since there is only a single wide area exchange, this algorithm adheres to the single-latency condition.

MPICH's algorithm arranges the P processes in a hypercube and takes up to $\lfloor \log_2 P \rfloor + 2$ times the wide area latency, depending on whether the hypercube layout matches the cluster topology.

The completion times of both algorithms (MPICH's and MAGPIE's) are shown in Figure 3. Results are shown for 2, 4, and 8 clus-

ters, with a total number of 16, 24, 32, and 40 processors, equally distributed over the clusters. The run times for MAGPIE are shown in black while MPICH's times are shown in grey. (We compare against version 1.1 of MPICH.) MAGPIE's barrier terminates after the wide area latency of 10 ms while MPICH's algorithm chains multiple wide area sends.

3.2 Broadcast

In MPI_Bcast, a so-called root process sends a data vector to all other processes. This operation is the simplest case of an asymmetrical, one-to-many, algorithm. In MAGPIE, it consists of two steps: the root sends to the coordinator nodes which in turn broadcast inside their clusters. This algorithm adheres to both wide area optimality conditions: it needs only a single wide area latency and sends the data vector once to each cluster.

MPICH's broadcast operation arranges processors in a binomial tree, which performs badly when multiple clusters are used. Wide area messages may be chained and, even worse, data may be sent multiple times to the same cluster, depending on the exact cluster topology.

Figure 4 shows execution times of both broadcast algorithms. The left graph shows performance for messages of a single byte. It shows the impact of wide area latency on completion time. MAGPIE's broadcast needs exactly one wide area latency, while MPICH chains several wide area sends, depending on the number of clusters and processors. The right diagram shows 64 KByte messages, for which bandwidth restrictions of wide area links become important. Again, MAGPIE's completion time is virtually independent of the number of clusters and processors. MPICH sends the data vector multiple times to each cluster (especially when P is not a power of 2), and is therefore more sensitive to wide area bandwidth. Note that MPICH's completion time decreases when clusters are added, because adding clusters adds wide area links, and increases the bisection bandwidth.

3.3 Personalized Broadcast Operations

In addition to MPI_Bcast, MPI also has *personalized* broadcast operations: MPI_Scatter, MPI_Gather, and MPI_Alltoall. In MPI_Allgather, all processes collect the same data vector from each other.

The MPI standard also defines so-called vector versions of these operations, for example, MPI_AllGatherv. With respect to wide area optimality, their implementations and their runtime performance are identical to their non-vectorized counterparts.

MPICH implements the simplest possible scatter algorithm in which the root process linearly and directly sends the pieces of data to the respective nodes. Interestingly, this algorithm is wide area optimal: all processes form a global, one-level flat tree, conforming to the single-latency requirement while no data is sent unnecessarily. MAGPIE uses this algorithm, too. Related observations hold for the gather and for the personalized all-to-all exchange (no graphs are shown).

With MPI_Allgather, a coordinator-based algorithm can improve the total completion time, because all processes receive the same data, as with MPI_Bcast. In MAGPIE's algorithm, the coordinators first gather data locally into a subvector of their local cluster. They then gather the complete data vector by exchanging their partial vectors with each other, and finally broadcast the vector locally. This algorithm is wide area optimal.

MPICH's algorithm arranges the processors in a logical ring in which in $P-1$ communication rounds each processor simultaneously receives data from its predecessor and sends to its successor node, always sending the data item it just received. This logical ring shows excellent pipelining behavior, but also chains wide area latencies. Figure 5 shows the result. With messages of 1 byte, the logical ring needs as many wide area steps as there are clusters. Furthermore, the full data vector is sent between each pair of clusters, which is redundant, and reduces MPICH's performance for large messages (see the 64 KByte part of Figure 5).

3.4 Global Reduction Operations

MPI has a group of algorithms that perform a global reduction operation (such as sum, max, logical and, etc.) on data vectors provided by all processors. However, not all reduction operations are associative, which causes problems. For example, due to possible rounding errors or overflow/underflow conditions, computing a global sum is not associative; finding a global maximum, on the other hand, is. Since MPI provides no means to indicate whether an operation is *strictly* associative (for all possible data sets), implementations must ensure that the reduction operation is always executed in the same order. MPICH achieves this by arranging the processes in a binomial tree that imposes a fixed execution order. The problem is that such an algorithm is not wide area optimal, because of chaining of wide area latencies and repeated transmission of data (analogous to MPICH's broadcast algorithm). To enable further optimizations, this fixed order has to be relaxed. Therefore, MAGPIE allows the user to assert strict associativity, either by a run time flag for a whole application run, or inside the source code by calls to a special function for specific operations only.

MAGPIE's associative algorithm first reduces the cluster-local data on the coordinator nodes. In a final wide area step, these partial results are combined. The algorithm is wide area optimal, because it adheres to the single-latency condition, and also sends the minimal amount of data between clusters. The comparison of this algorithm with MPICH's tree algorithm is presented in Figure 6 for the case of 64KB data vectors.

For non-associative operators, MAGPIE implements two additional algorithms: one for short messages, and one for long messages. The first algorithm gathers all data at the root, which then applies the operation in the prescribed order, irrespective of the network topology. This satisfies only the single-latency condition, but sends $s \cdot P$ bytes of data to the root process, where s is the size of the data vector. This algorithm is faster than MPICH's tree algorithm for short data vectors (see the case of 1-byte data vectors in Figure 6). For long messages, the algorithm is not optimal: MPICH's binomial tree algorithm sends less data, and is used instead. Currently, a fixed threshold of 512 bytes is used to choose between both algorithms.

MPI defines three more reduction operations: a reduce-to-all, a reduce-scatter, (which scatters the result data vector to all processes), and a scan (a parallel prefix computation.) MAGPIE implements these three operations using the same choice of three algorithms each, as with the reduce operation. For short messages, MAGPIE implements algorithms adhering to the single latency condition by re-using MPI_Allgather and replicating the reduction computation to all nodes, respectively. For long messages, optimized algorithms (based on coordinator nodes) exploit user-asserted associativity. If neither of these algorithm versions is applicable, MPICH's algorithms are used. Again, MAGPIE selects at runtime which of the three respective algorithms to use. A detailed description of MAGPIE's reduction operations can be found in [24].

3.5 Dynamically Created Communicator Objects

MAGPIE fully supports the use of dynamically created communicators (other than MPI_COMM_WORLD). Our algorithms assume, however, that process ranks reflect cluster topology. (Cluster 0 contains processes $0 \ldots c_0$, cluster 1 contains processes $c_0+1 \ldots c_1$ etc.) Our underlying implementation can easily enforce this with MPI's default communicator MPI_COMM_WORLD. Dynamically

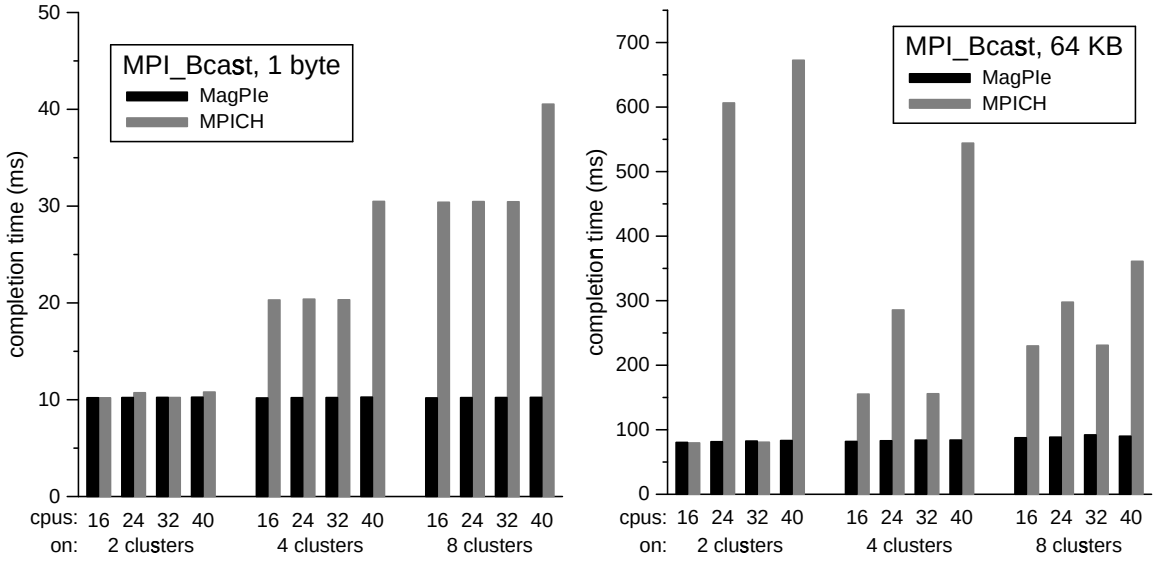


Figure 4: MPI_Bcast

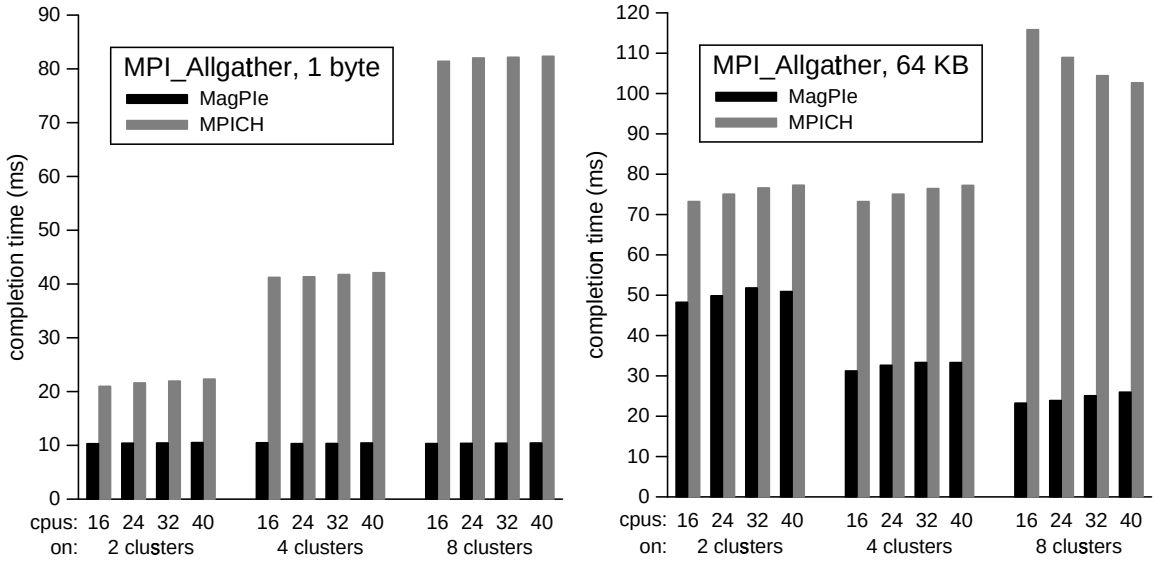


Figure 5: MPI_Allgather

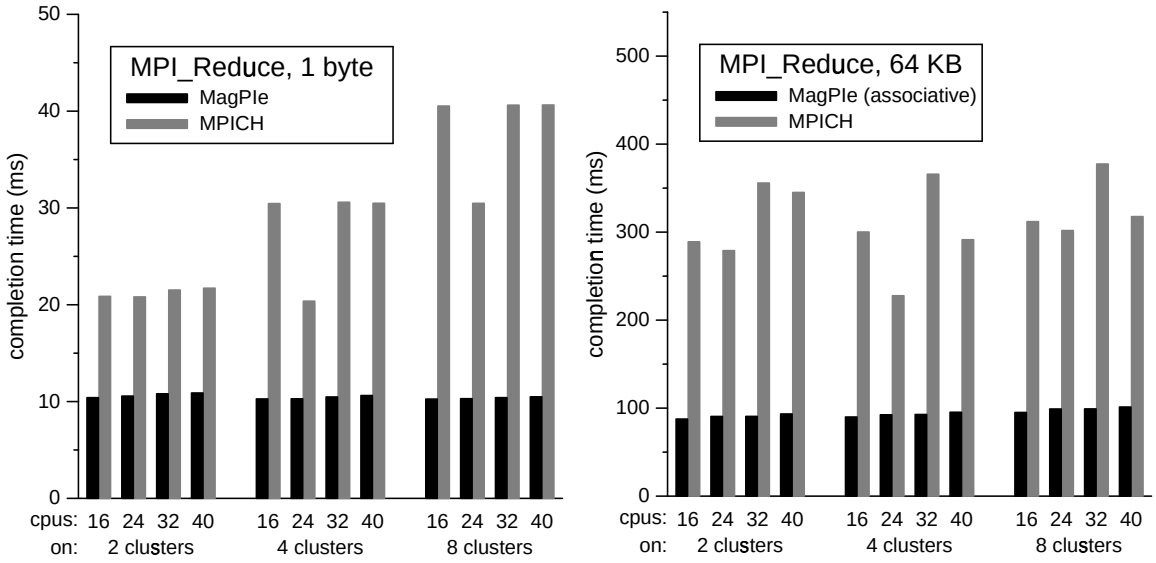


Figure 6: MPI_Reduce

created communicators, however, may break this assumption. We call this a *reordered* communicator.

MAGPIE keeps track of the cluster topology of dynamically created communicators and sends messages only to those clusters that contain processes of the actual communicator. For the reduction operations, reordered communicators impose an additional restriction. Here, the associative algorithms may only be applied when the reduction operator is also commutative. For MPI_Scan, a reordered communicator totally disables the associative algorithm, because commutativity does not help here.

3.6 Summary

MAGPIE implements the complete set of collective operations according to the MPI standard. Table 1 compares MAGPIE's and MPICH's algorithms for all 14 operations. For the global reduction operations, we distinguish between short and long messages, and strict associativity, as asserted by a user.

For each algorithm, we note whether it is wide area optimal, we name the shape of the communication graph of the wide area part, and we mention other collective operations used as building blocks when applicable. Wide area optimality of the reduction algorithms for short messages is marked as "near" (they are sub-optimal by a negligible amount of time). Six of the algorithms in MPICH are wide area optimal, while MAGPIE has (nearly) wide area optimal algorithms for all operations. The exceptions are global reductions with long messages for non-associative operators. Optimal graph shapes for this case depend on latency, bandwidth, and message size; they are not computed in the current system.

For communication graph shapes, "flat" denotes a flat-tree algorithm, "bin" denotes a global binomial tree, and ";" is sequential composition. The hypercube and ring shapes refer to MPICH's algorithms as discussed above, while the chain shape of the scan algorithm means that each processor sends to its direct successor.

For scatter and gather the flat tree is a natural choice. It is also wide area optimal; both MPICH and MAGPIE use it. For broadcast, MPICH uses the binomial tree, and MAGPIE improves on this by using the flat tree between cluster coordinators. Many operations are based on broadcast (for example, AllGather), and benefit accordingly. For reduce, MPICH uses the binomial tree as well, while MAGPIE uses the flat tree (except for large messages). Additionally, when the reduction operator is strictly associative the cluster coordinators perform these operations, which significantly reduces the amount of data sent over the wide area links.

4 Application Performance

To evaluate the impact of MAGPIE's optimized collective operations on application performance we use the following programs: ASP, QR, and PARMETIS. Table 2 lists execution statistics of the applications for a run of 40 processors divided over 8 clusters, yielding 5 processors per cluster.

ASP solves the all-pairs-shortest-path problem with a parallel version of the Floyd-Warshall algorithm. The rows of the 4000×4000 distance matrix are block-striped across all processors [32]. The program invokes MPI_Bcast 4000 times, with a message size of 16 KByte, and allows these broadcasts to be pipelined. Figure 7 shows run times for MAGPIE and MPICH for 32, 40, and 64 processors, on 1, 2, 4 and 8 clusters. As with the previous experiments, wide area latency is 10 ms, and bandwidth is 1 MByte/s. With 64 processors on 8 clusters, MAGPIE achieves a speedup of 56.6; MPICH achieves a speedup of 34. As was mentioned in Section 3.2, MPICH uses a broadcast tree that works especially badly for numbers of processors that are not a power of 2, which explains the large difference for 40 processors. As can be seen in Table 2, MAGPIE sends fewer messages and less data across wide area links. MAGPIE also chains fewer wide area latencies, on average only one per collective operation, compared to four needed by MPICH.

QR is a parallel implementation of QR factorization with a cyclic column distribution. We use a pivoting version, which has less parallelism than the non-pivoting variant. Pivoting also introduces some load imbalance. In our test runs, this was never more than 2%, so it was never necessary to enter a load balancing phase. The input is a 8192×8192 matrix. QR performs 8192 calls to MPI_Bcast with an average message size of 32 KByte for the broadcast of Householder vectors, and 8192 calls to MPI_Allreduce with messages of size 28 bytes for the pivot phase. QR does not allow

		MAGPIE			MPICH		
Operation		Optim.	Shape	Implementation	Optim.	Shape	Implementation
MPI_Barrier		yes	flat		no	hypercube	
MPI_Bcast		yes	flat		no	bin	
MPI_Scatter/MPI_Scatterv		yes	flat		yes	flat	
MPI_Gather/MPI_Gatherv		yes	flat		yes	flat	
MPI_Allgather/MPI_Allgatherv		yes	flat		no	ring	
MPI_Alltoall/MPI_Alltoallv		yes	flat		yes	flat	
MPI_Reduce	short msg.	near	flat	MPI_Gather	no	bin	
	long msg.	no	bin		no	bin	
	associative	yes	flat		no	bin	
MPI_Allreduce	short msg.	near	flat	MPI_Allgather	no	bin ; bin	MPI_Reduce ; MPI_Bcast
	long msg.	no	bin ; bin	MPI_Reduce ; MPI_Bcast	no	bin ; bin	MPI_Reduce ; MPI_Bcast
	associative	yes	flat		no	bin ; bin	MPI_Reduce ; MPI_Bcast
MPI_Reduce_scatter	short msg.	near	flat	MPI_Allgather	no	bin ; flat	MPI_Reduce ; MPI_Scatterv
	long msg.	no	bin ; flat	MPI_Reduce ; MPI_Scatterv	no	bin ; flat	MPI_Reduce ; MPI_Scatterv
	associative	yes	flat		no	bin ; flat	MPI_Reduce ; MPI_Scatterv
MPI_Scan	short msg.	near	flat	MPI_Allgather	no	chain	
	long msg.	no	bin		no	chain	
	associative	yes	flat		no	chain	

Table 1: Comparison of Algorithms

application	ASP		QR		ParMeTiS	
problem size	4000 × 4000		8192 × 8192		1000000	
operations	Broadcast		Bcast/AllReduce		Bcast/AllReduce/Barrier	
	MAGPIE	MPICH	MAGPIE	MPICH	MAGPIE	MPICH
run time (s)	151	194	1799	2776	6.3	13.3
wide area msgs	462114	1359784	2394161	5993113	19419	21373
wide area data (MB)	432.49	1272.74	1903.24	5357.52	4.15	5.07
# of wide area latencies (total)	4000	16000	16384	98304	307	1715
# of wide area latencies (avg)	1	4	1	6	1.24	6.94

Table 2: Applications on 40 Processors, 8 Clusters

	ASP		QR	
Processors	MAGPIE	MPICH	MAGPIE	MPICH
32	372	447	325	490
40	375	1026	325	908

Table 3: Wide Area System Run Times (seconds)

communication to be pipelined, and multicluster runs are slower than single cluster runs, as shown in Figure 8. MAGPIE outperforms MPICH by up to a factor of two. Relative to a single processor, the speedup on a single cluster of 64 processors is 50.2, both for MAGPIE and MPICH. When the 64 processors are divided over 8 wide area clusters, MAGPIE achieves a speedup of 28.8, and MPICH 13.9. Again, Table 2 shows that MAGPIE sends fewer messages and less data across wide area links. It also needs only one wide area latency per collective operation while MPICH needs six on average.

The third application is an algorithm from the PARMETIS graph partitioning library [23, 31]. We use its major component, the PartKway algorithm, with a graph of 1,000,000 nodes and 5,342,746 edges. The graph is initially randomly distributed across the processors. PartKway calculates a partitioning which is balanced over nodes and offers a near-minimal cut through edges. The parallelization uses a mixture of MPI's collective operations, dominated by calls to MPI_Allreduce, MPI_Barrier, and MPI_Bcast. Again, MAGPIE outperforms MPICH by up to a factor of two. Here, MAGPIE needs on average 1.24 wide area latencies per collective operation (compared to 6.94 for MPICH). This is due to non-strictly associative reduction of large data vectors.

Finally, we ran ASP and QR also on the real wide area DAS

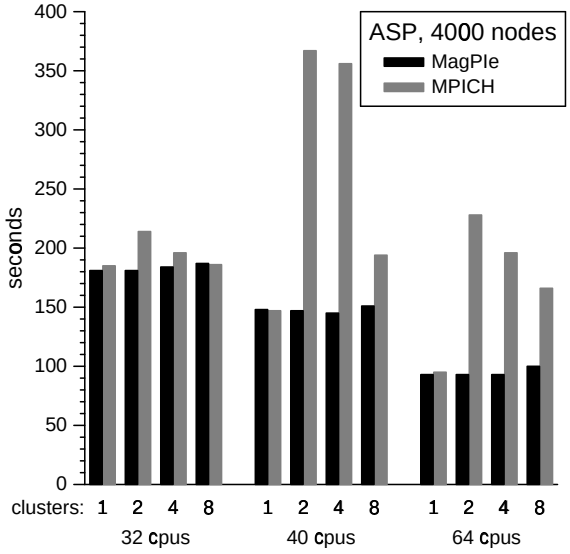


Figure 7: Application Runtimes for ASP

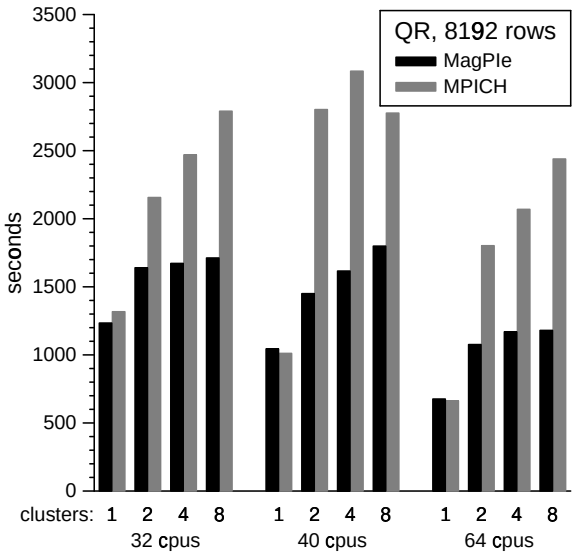


Figure 8: Application Runtimes for QR

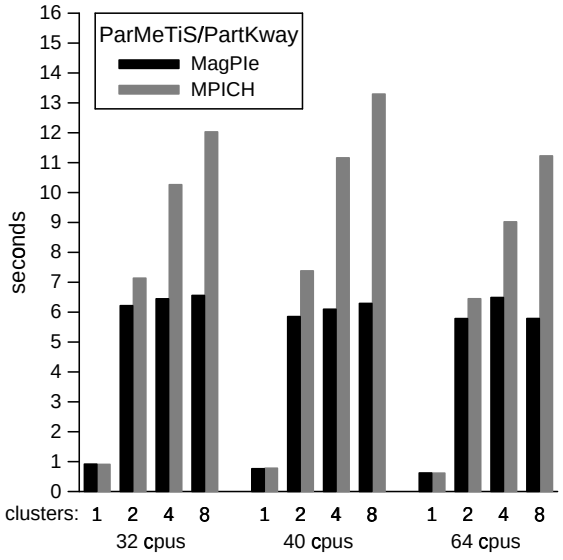


Figure 9: Application Runtimes for PARMETIS, PartKway Algorithm

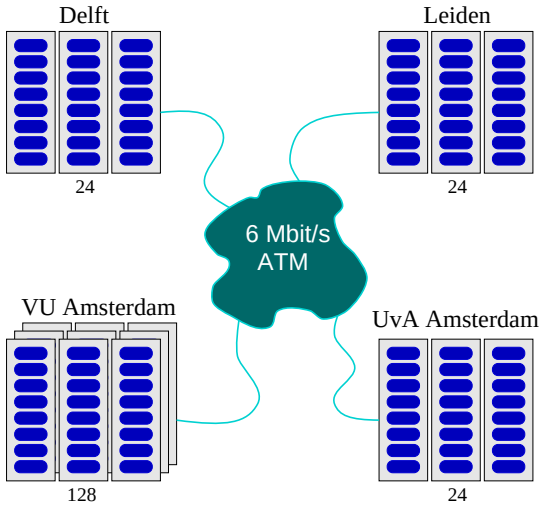


Figure 10: The Distributed ASCI Supercomputer (DAS)

system (as shown in Figure 10), in which latency varies between 1 ms and 3 ms, and MPI applications achieve a bandwidth of 575 KByte/s. We used 4 clusters, with 32 and 40 processors in total. For QR we used matrices of size 3072×3072, due to memory restrictions. (The problem size of ParMeTiS that fits in memory runs too fast to produce reliable results.) Table 3 shows the run times for the experiment. Again, MAGPIE is faster than MPICH, and MPICH's broadcast and reduce are highly sensitive to bandwidth when the number of processors is not a power of two.

5 Related Work

As machines scale up in size, interconnects are becoming increasingly hierarchical. In NUMA machines, remote memory access times differ by a factor of about 3–5, depending on how many hops a request must travel [26, 35]. On clusters of SMPs, the difference is larger, about one order of magnitude, depending on whether a reference can be satisfied within the local machine or whether it has to travel over the LAN [9, 25, 28, 34, 36]. However, on computational grids and metacomputers, bandwidth and latency differences in the interconnect can easily exceed *three* orders of magnitude [3, 32, 37]. Coping with such a large non-uniformity in the interconnect can significantly complicate application development. In previous work we experimented with optimizing the performance of traditional (single-level network) non-MPI programs for a hierarchical interconnect, by changing the communication structure [3, 32]. Among the communication patterns that could be optimized successfully were broadcast and reduce. MAGPIE now offers this way of optimization transparently to MPI programs.

Several metacomputing projects are currently building the infrastructure on top of which MAGPIE may utilize distributed computing capacity. The most prominent systems are Globus [15] and Legion [18].

PLUS [33] aims at interoperability of heterogeneous parallel applications based on separate communication daemons. It currently supports PVM, MPI, and PARIX. PVMPI [10] and its successor MPI_Connect [11] integrate multiple MPI applications based on PVM and SNIPE [12]. Both provide point-to-point communication between applications. MPI_Connect also provides a shared communicator MPI_COMM_WORLD. None of these systems optimizes collective communication. Foster et al. [13, 14] describe

a wide area version of MPI built upon Globus components. This work focuses on heterogeneity and interoperability issues. Our system also runs transparently on a LAN and a WAN, and, in addition, optimizes collective operations. PACX-MPI [17] implements a subset of the MPI standard, coupling multiple clusters into a shared MPI universe. The system also implements cluster-aware algorithms for a few collective operations without an elaborated design philosophy. MagPIe implements the complete set of MPI's collective operations with in-depth treatment of wide-area optimality and associativity of the reduction operations.

Husbands et al. [20] report significantly improved performance on a cluster of SMPs with a handcrafted two-level implementation of MPI_Bcast. Banikazemi et al. [4] investigate optimal communication structures for multicast operations on heterogeneous networks of workstations, focusing on processor speed. Our focus is network speed in wide area meta computers. Lowekamp et al. [27] describe a system that automatically analyzes characteristics of heterogeneous networks to develop optimized communication patterns. This work could be used with MAGPIE to determine the optimal communication shape at run time.

The LogP and LogGP models of parallel computation are useful for the analysis of optimal collective operations [1, 5, 8, 22]. They provide the theoretical underpinning for the design of our algorithms. Park et al. present an algorithm for constructing optimal broadcast trees for single-level networks based on terms similar to LogP [30].

6 Conclusion

Current programming environments offer little support for hierarchical interconnects. MAGPIE implements MPI's collective operations, taking the hierarchical structure of the network topology into account. Separate algorithms for the wide area level and the local area level are used, with cluster coordinators acting as intermediates.

Traditionally, for low-latency message passing systems where completion times of sender and receiver are close to each other ($t_r \approx t_s$), a binomial tree is the optimal one-to-many communication strategy, with receivers forwarding messages. Wide area links have a longer latency and lower bandwidth ($t_r \gg t_s$), and the one-level flat tree is typically the optimal one-to-many strategy. With many-to-many operations, flat trees are optimal shapes for both short and long latencies.

The measurements of the individual operations (using preset bandwidth and latency) show improvements over MPICH that vary between a factor of 2 and 8, depending on the number of clusters and the message length. In addition, application measurements have been performed with ASP, QR factorization, and the PARMETIS library. With a wide area latency of 10 ms, a bandwidth of 1 MByte/s, and 64 processors divided over 8 clusters, MAGPIE consistently outperforms MPICH by about a factor of 2. Measurements on the real wide area system confirm these results.

MPICH does not take the interconnection topology into account, and therefore sends data items multiple times over wide area links for certain topologies. Moreover, latency chaining occurs in most of its algorithms. In contrast, MAGPIE takes the cluster topology into account, and avoids these problems. Our measurements are performed with relatively modest wide area latencies. For higher wide area latencies, MAGPIE's advantage will increase. We believe that MAGPIE will be a good basis for further research on grid-aware MPI implementations [14].

Writing correct and efficient parallel programs is hard. Having to take non-uniformity of the interconnect into account makes it even harder. MPI's collective operations provide a convenient abstraction that can be implemented efficiently for a non-uniform interconnect. For problems that can be expressed in terms of the operations (ASP and QR, for example), the MAGPIE library offers transparent optimization, and completely hides non-uniformity of the interconnect.

The system is available as a plug-in to MPICH from:

http://www.cs.vu.nl/albatross/

Acknowledgements

This work is supported in part by a SION grant from the Dutch research council NWO, and by a USF grant from the Vrije Universiteit. Ceriel Jacobs and the anonymous referees provided valuable feedback on an earlier draft of this paper. We thank Kees Verstoep and John Romein for keeping the DAS in good shape, and Cees de Laat (University of Utrecht) for getting the wide area links of the DAS up and running.

References

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model — One Step Closer Towards a Realistic Model for Parallel Computation. In *Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, Santa Barbara, CA, July 1995.

[2] H. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Rühl, and F. Kaashoek. Performance Evaluation of the Orca Shared Object System. *ACM Transactions on Computer Systems*, 16(1):1–40, 1998.

[3] H. Bal, A. Plaat, M. Bakker, P. Dozy, and R. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In *IPPS-98 International Parallel Processing Symposium*, pages 784–790, Apr. 1998.

[4] M. Banikazemi, V. Moorthy, and D. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. In *International Conference on Parallel Processing*, pages 460–467, Minneapolis, MN, Aug. 1998.

[5] M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory. *Concurrency: Practice and Experience*, 10(5):359–386, April 1998.

[6] R. Bhoedjang, T. Rühl, and H. Bal. User-Level Network Interface Protocols. *IEEE Computer*, 31(11):53–60, 1998.

[7] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 1–12, San Diego, CA, May 1993.

[9] A. Erlichson, N. Nuckolls, G. Chesson, and J. Hennessy. SoftFLASH: Analyzing the Performance of Clustered Distributed Virtual Shared Memory. In *Proc. 7th Intern. Conf. on Arch. Support for Prog. Lang. and Oper Systems*, pages 210–220, Oct. 1996.

[10] G. E. Fagg, J. J. Dongarra, and A. Geist. Heterogeneous MPI Application Interoperation and Process Management under PVMPI. In *Proc. 4th European PVM/MPI Users' Group Meeting*, number 1332 in LNCS, pages 91–98, Cracow, Poland, 1997.

[11] G. E. Fagg, K. S. London, and J. J. Dongarra. MPI_Connect: Managing Heterogeneous MPI Applications Interoperation and Process Control. In *Proc. 5th European PVM/MPI Users' Group Meeting*, number 1497 in LNCS, pages 93–96, Liverpool, UK, 1998.

[12] G. E. Fagg, K. Moore, J. J. Dongarra, and A. Geist. Scalable Network Information Processing Environment (SNIPE). In *SC'97*, Nov. 1997. Online at http://www.supercomp.org/sc97/proceedings/.

[13] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. Wide-Area Implementation of the Message Passing Interface. *Parallel Computing*, 24(12-13):1735–1749, 1998.

[14] I. Foster and N. Karonis. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In *SC'98*, Orlando, FL, Nov. 1998. Online at http://www.supercomp.org/sc98/proceedings/.

[15] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

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

[17] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed Computing in a Heterogeneous Computing Environment. In *Proc. 5th European PVM/MPI Users' Group Meeting*, number 1497 in LNCS, pages 180–187, Liverpool, UK, 1998.

[18] A. Grimshaw and W. A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Comm. ACM*, 40(1):39–45, Jan. 1997.

[19] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[20] P. Husbands and J. C. Hoe. MPI-StarT: Delivering Network Performance to Numerical Applications. In *SC'98*, Nov. 1998. Online at http://www.supercomp.org/sc98/proceedings/.

[21] G. Iannello, M. Lauria, and S. Mercolino. Cross–Platform Analysis of Fast Messages for Myrinet. In *Proc. Workshop CANPC'98*, number 1362 in Lecture Notes in Computer Science, pages 217–231, Las Vegas, Nevada, January 1998. Springer.

[22] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser. Optimal Broadcast and Summation in the LogP model. In *Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 142–153, Velen, Germany, June 1993.

[23] G. Karypis and V. Kumar. A Coarse-Grained Parallel Formulation of a Multilevel *k*-way Graph Partitioning Algorithm. In *Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[24] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MPI's Reduction Operations in Clustered Wide Area Systems. In *Proc. MPIDC'99, Message Passing Interface Developer's and User's Conference*, Atlanta, GA, March 1999.

[25] L. Kontothanassis, G. Hunt, R. Stets, N. Hardavellas, M. Cierniak, S. Parthasarathy, W. Meira, S. Dwarkadas, and M. Scott. VM-Based Shared Memory on Low-Latency, Remote-Memory-Access Networks. In *ISCA-24, Proc. 24th Annual International Symposium on Computer Architecture*, pages 157–169, June 1997.

[26] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *24th Ann. Int. Symp. on Computer Architecture*, pages 241–251, June 1997.

[27] B. Lowekamp and A. Beguelin. ECO: Efficient Collective Operations for Communication on Heterogeneous Networks. In *International Parallel Processing Symposium*, pages 399–405, Honolulu, HI, 1996.

[28] S. Lumetta, A. Mainwaring, and D. Culler. Multi-Protocol Active Messages on a Cluster of SMP's. In *SC'97*, Nov. 1997. Online at http://www.supercomp.org/sc97/proceedings/.

[29] Message Passing Interface Forum. MPI: A Message Passing Interface Standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[30] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni. Construction of Optimal Multicast Trees Based on the Parameterized Communication Model. In *Proc. Int. Conference on Parallel Processing (ICPP)*, volume I, pages 180–187, 1996.

[31] The PARMETIS Graph Partitioning Library. Online at http://www-users.cs.umn.edu/~karypis/metis/parmetis/main.shtml, 1997.

[32] A. Plaat, H. Bal, and R. Hofman. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. In *High Performance Computer Architecture HPCA-5*, pages 244–253, Orlando, FL, Jan. 1999.

[33] A. Reinefeld, J. Gehring, and M. Brune. Communicating Across Parallel Message-Passing Environments. *Journal of Systems Architecture*, 44:261–272, 1998.

[34] D. J. Scales, K. Gharachorloo, and A. Aggarwal. Fine-Grain Software Distributed Shared Memory on SMP clusters. In *HPCA-4 High-Performance Computer Architecture*, pages 125–137, Feb. 1998.

[35] V. Soundararajan, M. Heinrich, B. Verghese, K. Gharachorloo, A. Gupta, and J. Hennessy. Flexible Use of Memory for Replication/Migration in Cache-Coherent DSM Multiprocessors. In *ISCA-98, 25th International Symposium on Computer Architecture*, pages 342–355, June 1998.

[36] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network. In *Proc. 16th ACM Symp. on Oper. Systems Princ.*, pages 170–183, Oct. 1997.

[37] R. Wolski. Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. In *6th High-Performance Distributed Computing*, Aug. 1997. The network weather service is at http://nws.npaci.edu/.