MPI's Reduction Operations in Clustered Wide Area Systems

Thilo Kielmann Rutger F. H. Hofman
Henri E. Bal Aske Plaat Raoul A. F. Bhoedjang
Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
{kielmann,rutger,bal,aske,raoul}@cs.vu.nl
http://www.cs.vu.nl/albatross/

Abstract— **The emergence of meta computers and computational grids makes it feasible to run parallel programs on large-scale, geographically distributed computer systems. Writing parallel applications for such systems is a challenging task which may require changes to the communication structure of the applications. MPI's collective operations (such as broadcast and reduce) allow for some of these changes to be hidden from the applications programmer. We have developed MAGPIE, a library of collective communication operations optimized for wide area systems. MAGPIE's algorithms are designed to send the minimal amount of data over the slow wide area links, and to only incur a single wide area latency. This paper discusses MPI's collective reduction operations. Compared to systems that do not take the topology into account, such as MPICH, large performance improvements are possible. For larger messages, best performance is achieved when the reduction function is associative. On moderate cluster sizes, using a wide area latency of 10 millisecond and a bandwidth of 1 MByte/s, operations execute up to 8 times faster than MPICH; application kernels improve by up to a factor of 3. Due to the structure of our algorithms, the advantage increases for higher wide area latencies.**

I. INTRODUCTION

Several research projects pursue the idea of integrating computing resources at different locations into a single, powerful parallel system. Metacomputing projects like Globus [15] and Legion [17] build the software infrastructure that makes such an integration possible. An important problem, however, is how to write parallel programs that run efficiently on metacomputers (or computational grids [16]). The key difference with traditional parallel programs is that communication between distant computers can be orders of magnitude slower than that between the processors within a parallel machine. Wide-area networks typically have a latency and bandwidth that is a factor 100–1000 worse than that of local interconnects.

Earlier research showed that, for many applications, it is possible to overcome the slowness of wide-area links by optimizing programs at the application level [5], [13], [26]. Metacomputers typically consist of several *clusters* (parallel machines like MPPs or networks of workstations), connected by slow wide-area links. They typically have a hierarchical structure with slow and fast links. Programmers can take this hierarchical structure into account and minimize the amount of traffic over the slow wide-area links [26].

Such optimizations, however, can complicate metacomputer programming significantly. The goal of our work is to hide the hierarchical structure as much as possible from the programmer, by implementing the optimizations in a communication library. This works especially well for collective communication primitives, such as found in MPI. Current implementations of MPI (for example, MPICH [2]) are designed for "flat" systems and run inefficiently on wide-area, hierarchical systems. We have designed and implemented a new library, called MAGPIE, whose collective communication routines are optimized for hierarchical systems.

In an earlier paper, we described the general design of MAGPIE and the implementation and performance of some collective operations [20]. In this paper, we discuss in more detail the *reduction* operations: MPI_Reduce, MPI_Allreduce, MPI_Reduce_scatter, and MPI_Scan. We show how these operations can be implemented efficiently on a wide-area system. An optimization for associative reduction operations is to first do the reductions locally (within a cluster) and transfer only the partial results over the wide-area links. This optimization substantially reduces the amount of data that has to be sent over the wide-area network. For non-associative operations, MAGPIE uses the size of the parameters to determine the best strategy.

Performance measurements show that the optimizations typically win a factor of 3 to 4 compared to MPICH. Speedups of application kernels are also better for MAGPIE than for MPICH. This performance gain comes without having to change a single line of application code. The only extension we have made to MPI is to allow the programmer to specify whether a reduction operation should be regarded as associative.

In the rest of the paper we describe the design of MAGPIE's reduction algorithms (Section II) and the MAGPIE library that implements these algorithms (Section III). We present measurements and discuss the performance of a few MPI applications (Section IV). Section V discusses related work. Section VI concludes.

II. Algorithm Design

MagPIe implements wide area optimal algorithms for all of MPI's collective operations. We first outline the general structure of MagPIe's algorithms. Then, we discuss how associativity of the reduction operators influences the applicability of this structure to the reduction operations.

A. General Algorithm Structure

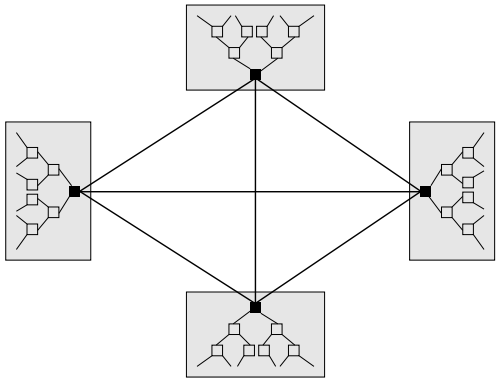
MagPIe's goal is to minimize completion time of a collective operation, which we define as the moment at which all processors have received all messages that belong to that operation. The performance of collective communication operations on a wide area system is dominated by the time spent on the wide area links; local communication plays a minor role. In a collective operation, all processors have to communicate with each other, so the completion time cannot be less than the wide area latency. (On interconnects with varying latencies the highest latency dominates completion time. To simplify our analysis, we assume that all wide-area links have the same bandwidth and 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. Data items only travel to those clusters that need them; no data item travels multiple times to the same cluster.

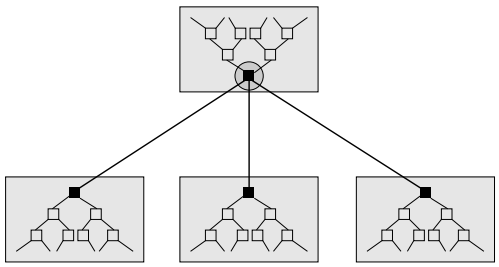
Condition (1) ensures that the wide area latency contributes at most once to an operation's completion time. Condition (2) prevents waste of precious wide area bandwidth. We call algorithms that adhere to both conditions *wide area optimal*. Reduction operations have a high potential for optimization, by computing partial reductions locally in each cluster. The applicability of this optimization depends on associativity and commutativity of the reduction operation. We will discuss this issue in detail below.

In previous work [20], we showed how MPI's collective operations for synchronization and data exchange can be implemented by wide area optimal algorithms. In Section III we will show how MPI's reduction operations can be implemented accordingly. We distinguish 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: a message send completes when the message has been injected into the network. Note that t_s only depends on message size; t_r additionally depends on network bandwidth and latency. These performance characteristics determine the shape of the optimal communication graph [7], [19], [25].

Local communication contributes a negligible amount to the overall completion time; for local communication we use graph shapes that best fit the needs of the operations, like binary or binomial trees. According to [7], binomial trees are optimal when $t_r - t_s$ is small. However, for wide area communication, where $t_r \gg t_s$, the optimal shape is a one-level flat tree [7]. Thus we have two communication graphs: the *intra-cluster* graph that connects the processors within a single cluster, and the *inter-cluster* graph that connects the different clusters. To interface both graphs, we designate a *coordinator node* for each cluster. Notice that the one-level tree satisfies condition (1). Condition (2) depends on the semantics of the actual operation, as will be discussed further in Section III.



(a) symmetric operation



(b) asymmetric operation

Fig. 1. Wide Area Optimal Communication Graphs

The optimal *inter-cluster* graph shape depends on the values for latency, bandwidth, message size, and the number of clusters. Computing optimal graph shapes therefore requires run time instrumentation (see, for example, [22]). MagPIe does not yet perform this analysis. In Section III we show that MagPIe's combination of one-level trees and binary/binomial trees fits the wide area case well enough to outperform MPICH's algorithms in our tests.

To outline the general structure of our algorithms, we distinguish two kinds of algorithms. In the *asymmetric* algorithms one dedicated process, called the *root*, either acts as sender (in one-to-many algorithms) or as receiver (in many-to-one algorithms such as a MPI_Reduce). In the *symmetric* algorithms (MPI_Allreduce, MPI_Reduce_scatter, and MPI_Scan) all processes are equal peers; they all send and receive. In asymmetric algorithms the root acts as the coordinator of its cluster. In other clusters the coordinator is chosen arbitrarily. In symmetric algorithms the coordinator is also

chosen arbitrarily. Figure 1 shows the communication graphs for symmetric and asymmetric operations; in the latter the root process is marked with a circle.

Asymmetric algorithms perform two steps. The nodes first send to their coordinator and then the coordinators send to the root. Symmetric algorithms perform three steps. First, nodes send to their coordinators. Second, the coordinators perform an all-to-all exchange, and third, coordinators send to their nodes. We use this basic structure for implementing wide area optimal algorithms for MPI's reduction operations.

B. Non-Associative Reduction Operators

We now turn to the group of reduction operators. MPI's reduction operations are parameterized by the actual operator that is applied to the data. MPI assumes all operators (such as sum, product, minimum, or maximum) to be associative; in addition, the programmer can mark them as commutative. Concerning execution order, the MPI standard states that "*any implementation can take advantage of associativity, or associativity and commutativity in order to change the order of evaluation. This may change the result of the reduction for operations that are not strictly associative and commutative, such as floating point addition*" [24]. Accordingly, application programmers must not rely on a specific execution order. For yielding reproducible results, the standard adds: "*It is strongly recommended that* MPI_Reduce *be implemented so that the same result be obtained whenever the function is applied on the same arguments, appearing in the same order.*" Hence, implementations are allowed to use optimized algorithms, but they always have to use the same execution order (whichever they choose), independent of process locality. For clustered wide area systems, this implies that an implementation must yield identical results, independent of the number and size of clusters.

Implementations can check at runtime whether a given reduction operator is declared to be commutative. For associativity no information is provided by MPI. Unfortunately, associativity is required for certain optimizations. For example, to add numbers stored at different processors, one can first add all numbers within each cluster and send only the partial sums (rather than all numbers) over the WAN. For this optimization to be applicable, the programmer must decide whether the add operation can be considered to be associative or not. (For example, if there is a possibility of overflow or loss of precision, it can not.)

Associativity cannot be derived automatically from a reduction operator. While some operators are known always to be associative (like minimum, maximum, and many bit-wise operations), integer or floating point arithmetic *may* be associative, depending on the input data. Only the application programmer knows whether a certain reduction is associative. For this reason, MAGPIE allows users to explicitly *assert* associativity, either by a runtime flag or by a collective operation that asserts or withdraws this property to or from a specific communicator object. In the following section, we present in detail how MPI's reduction operations benefit from optimizations when strict associativity can be assumed.

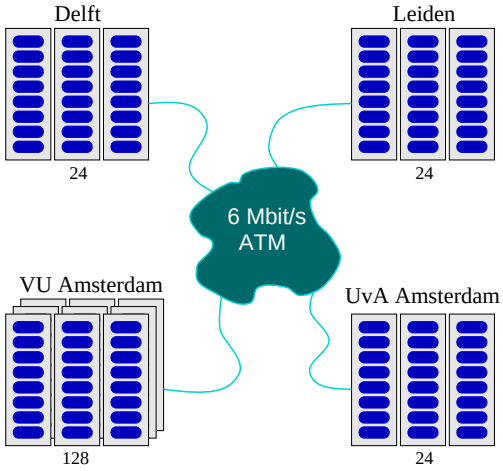


Fig. 2. The Distributed ASCI Supercomputer (DAS)

III. Wide Area Collective Communication Algorithms

MAGPIE implements the collective communication operations of MPI 1.1 [23], on top of MPICH 1.1, a widely used public MPI implementation [2]. 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 [10] clusters of 200 MHz/128 MByte Pentium Pros, connected by a 6 Mbit/s ATM network (as shown in Figure 2). 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 [26] (and on *http://www.cs.vu.nl/das/*). MPICH has been ported to the Panda communication substrate [4], which gives access to IP and Myrinet. We use the LFC [9] 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 according to [18]. 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. This instrumentation is transparent to the application. All measurements in this section were performed on the local experimentation system; Section IV also contains measurements on the real wide area system. For the re-

mainder of this paper, 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.

A. Basic Collective Operations

MAGPIE implements all 14 collective communication operations defined by version 1.1 of the MPI standard. The operations for synchronization and data exchange have been presented in [20]. Some of them are used as building blocks for the reduction operations. We briefly discuss how they are implemented by MPICH and by MAGPIE. We compare against version 1.1 of MPICH.

A.1 Broadcast

In MPI_Bcast, the root process sends a data vector to all other processes. In MAGPIE, 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. In our test cases (clusters of equal size), the latter effect is strongest when the number of processors is not a power of two.

A.2 Personalized Broadcast Operations

In addition to MPI_Bcast, MPI also has *personalized* broadcast operations: MPI_Scatter, MPI_Gather, MPI_Allgather, and MPI_Alltoall.

MPICH implements the simplest possible scatter algorithm in which the root process linearly and directly sends the pieces of data to the respective nodes. This algorithm is wide area optimal, and MAGPIE does not improve on it. The gather operation and the personalized all-to-all exchange are implemented similarly.

With MPI_Allgather, MPICH's algorithm arranges the P 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 chains wide area latencies and is therefore not wide area optimal. 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.

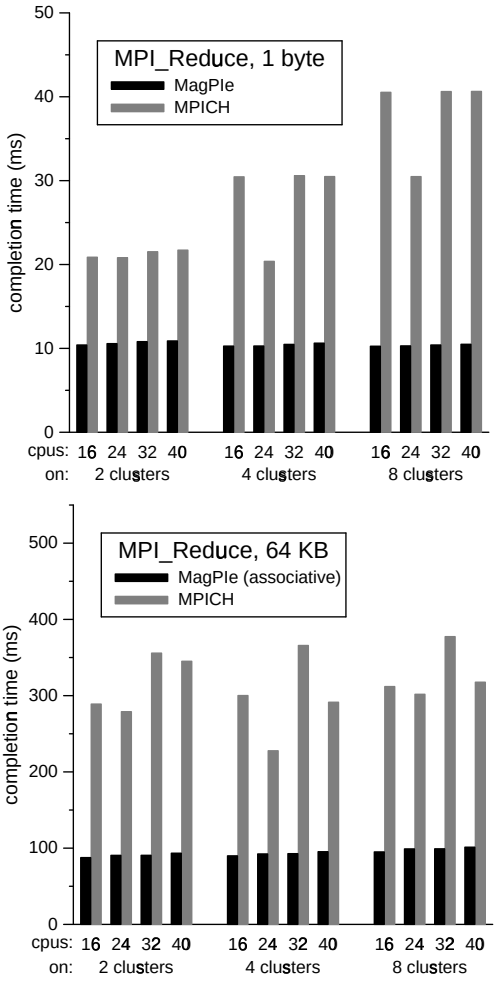


Fig. 3. MPI_Reduce

B. Reduction Operations

We will now discuss how MPI's reduction operations can be efficiently implemented for wide area systems.

B.1 MPI_Reduce

With MPI_Reduce, only the root process gets the result of the global reduction. MPICH ensures that the reduction operation is always executed in the same order by arranging the processes in a binomial tree. Such an algorithm is not wide area optimal because of chaining of wide area latencies and possibly repeated transmission of data, analogous to the problem with MPICH's broadcast algorithm.

MAGPIE employs three algorithms, one for associative reductions, and two different algorithms for short and long messages for non-associative operators.

MAGPIE's algorithm that exploits user-asserted associativity first reduces the cluster-local data on the coordinator nodes. In a final wide area step, these partial results are sent to the root process which in turn combines them to compute the overall result. This algorithm is wide area optimal; 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 3 for the case of 64KB data vectors. Results are shown for 2, 4, and 8 clusters, 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.

For non-associative operators, MAGPIE implements two more 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 (by re-using MPI_Gather), but sends $s \cdot P$ bytes of data to the root process. This algorithm is faster than MPICH's tree algorithm for short data vectors (see the case of 1-byte data vectors in Figure 3). Reductions of very short data vectors, even with single data items, are frequently used, for example, for finding pivot elements or for computing global sums.

For long messages, this 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 switch between both algorithms.

MAGPIE fully supports the use of dynamically created communicators (other than MPI_COMM_WORLD). Our algorithms assume, however, that process ranks reflect cluster topology. (Within cluster 0, processes $0 \ldots c_0$ are located, cluster 1 contains processes $c_0 + 1 \ldots c_1$ etc.) With MPI's default communicator object MPI_COMM_WORLD, our underlying implementation can easily enforce this. Dynamically created communicators, however, may break this assumption. In this case, the associative algorithm may only be applied when the reduction operator is also commutative.

B.2 MPI_Allreduce

MPI_Allreduce resembles MPI_Reduce, but delivers the reduction result to all processes. MPICH provides a so-called naive implementation by first reducing to the process with rank zero and subsequently broadcasting the result. This implementation always yields correct results, but is not wide area optimal. Sequential compositions of two collective operations need at least two times the wide area latency. Furthermore, both basic operations (as implemented by MPICH) are not wide area optimal themselves.

Again MAGPIE provides three algorithms. For associative operators, the processes in each cluster first reduce to their coordinator nodes. The coordinators then perform an all-to-all exchange of their partial results which they then combine and finally broadcast in their local clusters. This algorithm is wide area optimal. Again, in the case of reordered communicators, the algorithm can only be applied to commutative operators. Its completion time compared to MPICH's "naive" implementation is shown in Figure 4 (bottom) for the case of 64 KB messages.

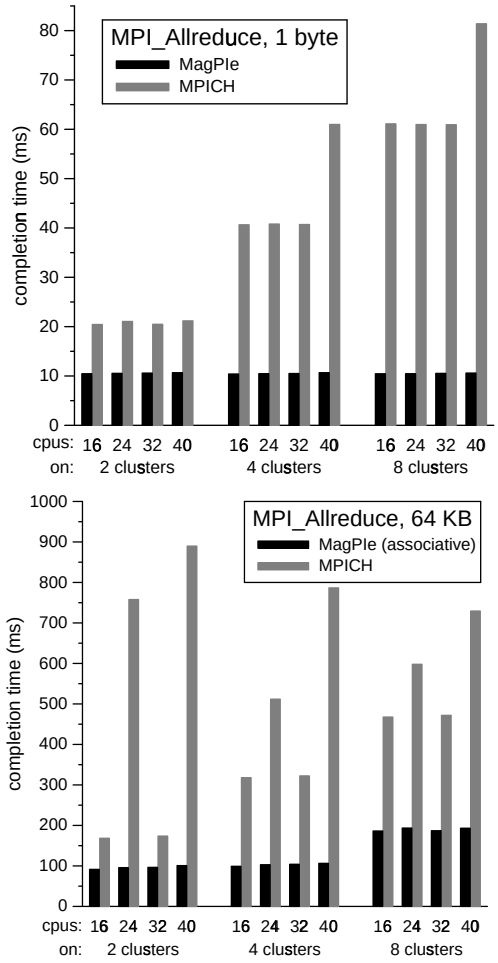


Fig. 4. MPI_Allreduce

For non-associative operators, MAGPIE has two algorithms. For short messages, MPI_Allgather is called which delivers all data items at all processes with a single wide area latency. Then, all processes compute the reduction result locally. The completion time compared to MPICH is shown in Figure 4 (top) for 1-byte messages. For long messages, MPICH's approach is used. Because MAGPIE implements a wide area optimal broadcast, it is still faster than MPICH.

B.3 MPI_Reduce_scatter

MPI_Reduce_scatter is similar to MPI_Allreduce. Here,

the resulting data vector is implicitly scattered among the processes such that each process gets a different part of the reduced data vector. MPICH implements a sequential composition, a reduce followed by a scatter operation. Analogous to MPI_Allreduce, this is not wide area optimal.

In MAGPIE's associative algorithm the coordinator nodes scatter their partial results to each other. In this way, every cluster gets only those parts of the vector that it needs. The minimal amount of data is sent over the wide area links, and only a single wide area latency is needed. The rest of the algorithm is identical to MPI_Allreduce. Using a communicator object with reordered process ranks, the algorithm is only applicable to commutative operators.

For non-associative operators, an MPI_Allgather based algorithm is used for short messages while MPICH's sequential composition is re-used for long messages. The runtime comparison can be found in Figure 5. (Here, 1-byte messages actually mean one byte per process because otherwise a scatter would not be useful.) With 64 KB messages, MAGPIE's runtimes improve with increasing numbers of clusters because more clusters imply shorter messages over the individual wide-area links.

B.4 MPI_Scan

The operation performed by MPI_Scan is also known as *parallel prefix computation.* Here, every process i gets the result of reducing the data vectors from the processes $0, 1, \ldots, i$. MPICH implements this by letting each process i receive the partial result from $i-1$, combine it with the local data, and send it to process $i+1$. This algorithm is not wide area optimal because it arranges all processes in a logical chain, and thus also chains wide area latencies. This can be seen in Figure 6 for the case of 1-byte messages. For 64 KB messages, the figure shows that this chaining also significantly adds to the completion time when increasing the number of processors with a fixed number of clusters.

An optimal scan algorithm for single cluster systems arranges the processes in a binomial tree [11]. MAGPIE implements this algorithm as a general optimization. Figure 6 (64 KB) shows the corresponding improvement. As binomial trees are used inside clusters, MAGPIE's runtime increases slower than MPICH's when the number of processes is increased.

For associative operators, MAGPIE implements a wide area optimal scan algorithm. The processes with the respectively highest ranks act as coordinators of their clusters. In a first step, all processes perform a scan locally in their cluster, yielding the respective partial results. In the wide area exchange phase, the coordinators send their partial result to all coordinators of clusters with processes of higher ranks. The coordinators in turn reduce all results from other clusters into a single partial result and broadcast it locally in their cluster. Finally, all processes reduce their partial result from the first step with the result from the other clusters. This algorithm is wide area optimal because it needs only a single wide area latency and sends a single data vector per cluster, only to those clusters that need it. In case of a communicator with reordered ranks, this algorithm can not be applied, because commutativity does not help for the scan operation. The completion time of this algorithm is shown in Figure 6 with 64 KB messages.

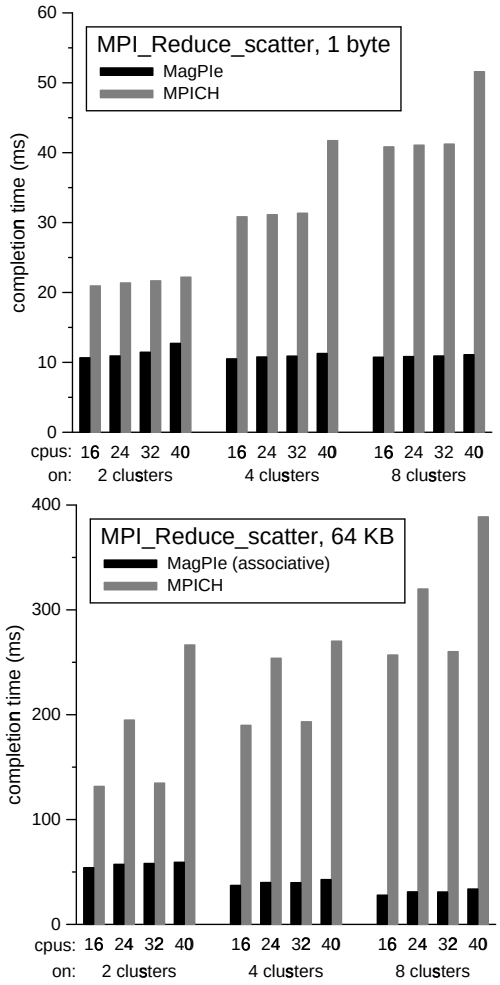


Fig. 5. MPI_Reduce_scatter

For non-associative operators, MAGPIE implements a short-message algorithm similar to MPI_Allreduce. It is based on MPI_Allgather where the final computation only reduces the data vectors up to the rank of the respective process. Figure 6 (1-byte messages) shows the behavior of this algorithm. For long messages, MAGPIE uses a global binomial tree. (No graph is shown for this case.)

C. Summary

MAGPIE implements the complete set of collective op-

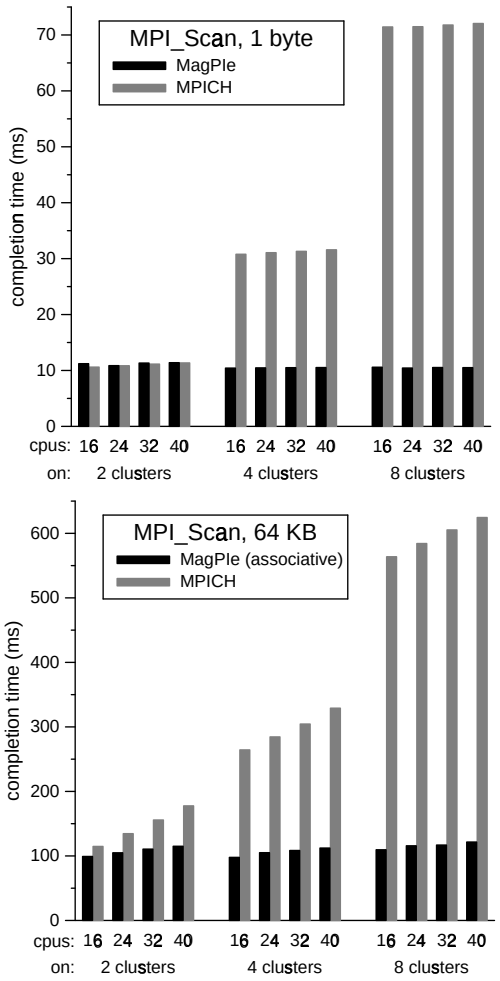


Fig. 6. MPI_Scan

erations according to the MPI standard. Table I compares MAGPIE's and MPICH's algorithms. For each algorithm, we summarize wide area optimality, communication graph shape of the wide area part, and mention other collective operations used as building blocks, when applicable. Wide area optimality of the algorithms for short messages is marked as "near"; they are sub-optimal by a negligible amount of time. For communication graph shapes, "flat" denotes a flat-tree algorithm, "bin" denotes a global binomial tree, and ";" is sequential composition.

Reduction operations with short data vectors are frequently used in parallel applications. For this case, MAGPIE implements (nearly) wide area optimal algorithms. For long data vectors, the application programmer has to assert the strict associativity of the reductions to use MAGPIE's wide area optimal algorithms. If strict associativity is not the case, then MAGPIE still improves completion times compared to MPICH by relying on its wide area optimal broadcast algorithm and by using the tree based scan operation.

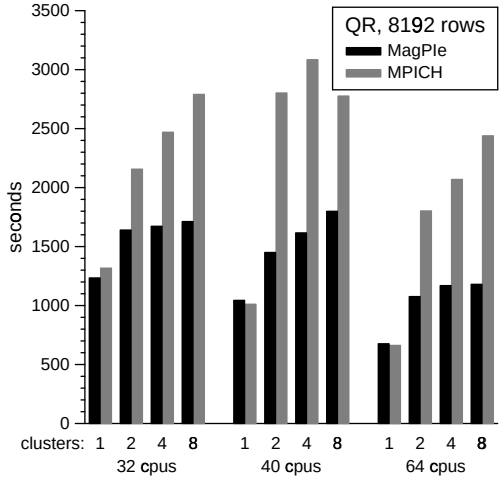


Fig. 7. Application Runtimes for QR

IV. APPLICATION PERFORMANCE

To evaluate the impact of MAGPIE's optimized reduction operations on application performance we use the following program kernels: QR, MMUL, and TRI. Table II lists execution statistics of the applications for runs with 40 processors evenly divided over 8 clusters.

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. Figure 7 shows that MAGPIE's run times outperform 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. Table II shows that MAGPIE sends fewer messages and less data across wide area links. Furthermore, MAGPIE chains less wide area latencies; its average latency count (per collective operation) is 1 while MPICH's is 6.

MMUL is a parallel algorithm for matrix multiplication $A \cdot B = C$. It has been specifically written to use the reduction operations. It implements straight-forward multiplication with time complexity $O(n^3)$. By using a data distribution where each process holds several rows of A and C as well as several columns of B, this algorithm can easily communicate using MPI_Reduce_scatter. Not only yields MPI_Reduce_scatter efficient communication, its combination of computation and data distribution further-

TABLE I

COMPARISON OF ALGORITHMS

Operation		MAGPIE Optim.	MAGPIE Shape	MAGPIE Implementation	MPICH Optim.	MPICH Shape	MPICH Implementation
Reduce	short msg.	near	flat	Gather	no	bin	
	long msg.	no	bin		no	bin	
	associative	yes	flat		no	bin	
Allreduce	short msg.	near	flat	Allgather	no	bin ; bin	Reduce ; Bcast
	long msg.	no	bin ; bin	Reduce ; Bcast	no	bin ; bin	Reduce ; Bcast
	associative	yes	flat		no	bin ; bin	Reduce ; Bcast
Reduce-Scatter	short msg.	near	flat	Allgather	no	bin ; flat	Reduce ; ScatterV
	long msg.	no	bin ; flat	Reduce ; ScatterV	no	bin ; flat	Reduce ; ScatterV
	associative	yes	flat		no	bin ; flat	Reduce ; ScatterV
Scan	short msg.	near	flat	Allgather	no	Chain	
	long msg.	no	bin		no	Chain	
	associative	yes	flat		no	Chain	

TABLE II

APPLICATIONS ON 40 PROCESSORS, 8 CLUSTERS

application	QR		MMUL		TRI	
problem size	8192 × 8192		2000 × 2000		3 × 1000000	
operations	Bcast/AllReduce		Reduce_scatter		Bcast/Scan	
sequential run time (s)	33584		3983		3390	
	MAGPIE	MPICH	MAGPIE	MPICH	MAGPIE	MPICH
parallel run time (s)	1799	2776	96	300	111	169
speedup	18.7	12.1	41.5	13.3	30.5	20.1
wide area msgs	2394161	5993113	336085	597155	29957	63275
wide area data (MB)	1903.24	5357.52	223.03	525.78	2.49	17.20
wide area latency (total)	16384	98304	4000	20000	2000	11000
wide area latency (avg)	1	6	1	5	1	5.5

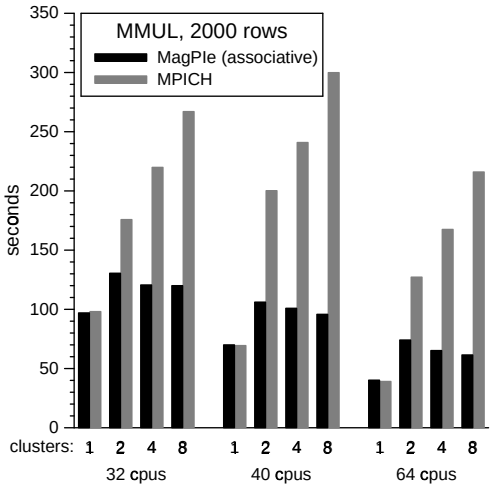


Fig. 8. Application Runtimes for MMUL

more allows all matrices to be fully distributed which significantly lowers the memory requirements of the parallel version.

For the runtimes shown in Figure 8, we enabled associative optimization. When the matrix elements are known to yield neither over nor underflow situations, this optimization is legal. If not, then even a result computed without this optimization could not be trusted, because still over or underflows may occur—although always the same.

With the matrices being of size 2000 × 2000, MPI_Reduce_scatter is called 4000 times with a message size of 16000 bytes. MAGPIE outperforms MPICH up to a factor of three. Due to large memory requirements and the related cache effects (the algorithm uses straightforward, un-blocked for loops), MMUL achieves superlinear speedups. Relative to a single processor, the speedup on a single cluster of 64 processors is 100, both for MAGPIE and MPICH. When the 64 processors are divided over 8 wide area clusters, MAGPIE achieves a speedup of 64.8, MPICH only 18.4. Again, Table II shows that MAGPIE sends fewer messages and less data across wide area links. It also chains fewer latencies.

The TRI kernel repeatedly invokes a solver for tridiagonal equation systems. Following [21], the solver treats the equations as a system of recurrences and uses MPI_Scan to solve them. The tridiagonal solver needs an *exclusive* scan (as defined by version 2 of the MPI standard). Because both MPICH and MAGPIE implement MPI version 1.1, only the inclusive MPI_Scan can be used. Some additional numerical instability had to be introduced to get the right functionality without additional communication.

In our measurements, the tridiagonal matrix had size 1000000. The solver calls 1000 times MPI_Bcast with a message size of 8 bytes, and 1000 times MPI_Scan with a message size of 48 bytes. MAGPIE outperforms MPICH

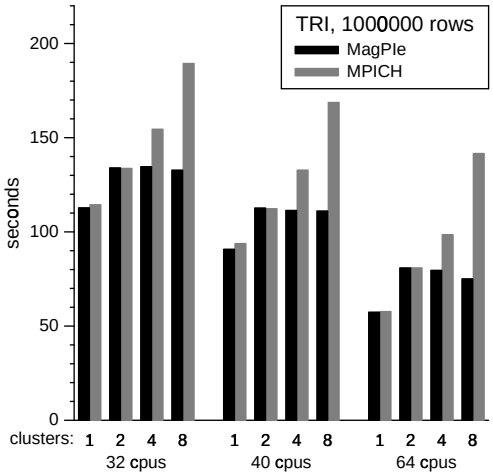


Fig. 9. Application Runtimes for TRI

TABLE III
WIDE AREA SYSTEM RUN TIMES

	32 processors		40 processors	
	MAGPIE	MPICH	MAGPIE	MPICH
QR	325	490	325	908
MMUL	24	61	24	81
TRI	123	138	101	116

up to a factor of two. Speedups for both MAGPIE and MPICH are 59 for 64 processors on a single cluster. With 64 processors and 8 clusters, MAGPIE achieves a speedup of 45.1, but MPICH only 23.9. In Table II, the numbers for wide area data include all protocol headers. Because TRI sends only very short messages, MPICH sends much more data over the wide-area links than MAGPIE.

Finally, we ran the three application kernels also on the wide area DAS system, 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. Due to memory restrictions on two of the clusters, we used matrices of size 3072×3072 for QR and of size 1000×1000 for MMUL. Table III shows the run times for the experiment. Again, MAGPIE is consistently faster than MPICH. MPICH's broadcast and reduce are highly sensitive to bandwidth when the number of processors is not a power of two. This can be observed with QR and MMUL which both send large messages. The TRI kernel is not affected by this problem because it only uses very short messages.

V. RELATED WORK

On computational grids and meta computers, bandwidth and latency differences in the interconnect can easily exceed three orders of magnitude [5], [13], [15], [17], [26], [27]. Coping with such a large non-uniformity in the interconnect can significantly complicate application development. We experimented with optimizing the performance of traditional (single-level network) non-MPI programs for a hierarchical interconnect, by changing the communication structure [5], [26]. Among the communication patterns that could be optimized successfully were broadcast and reduce. MAGPIE now offers this optimization transparently to MPI programs.

Foster et al. [13], [14], [15] describe a wide area version of MPI using the Nexus multi threaded run time system. 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. Banikazemi et al. [6] investigate optimal communication structures for multicast operations on heterogeneous networks of workstations, focusing on processor speed. Our focus is network speed in wide area metacomputers. Lowekamp et al. [22] 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], [12]. They provide the theoretical underpinning for the design of our algorithms. Using LogP, Karp et al. present optimal algorithms for the reduce and allreduce operations on homogeneous (single cluster) networks [19].

Bernaschi et al. extent this work on the reduce and allreduce operations to long data vectors (which are not covered by LogP) [7], and to the reduce-scatter operation [8]. Their work constructs optimally shaped reduction trees which are not wide area optimal, analogous to MPICH's global binomial tree. Since the authors assume reduction operators either to be associative or to be associative and commutative, these algorithms can only gradually improve MPICH's results. As shown above, MAGPIE performs better in this case. For non-strictly associative operators, Bernaschi et al.'s work cannot help either. A similar algorithm for the reduce-scatter operation, restricted to processor numbers being a power of 2, has been presented by Bae et al. [3].

VI. CONCLUSION

Current programming environments offer little support for hierarchical interconnects. No other implementation of MPI's collective operations that we are aware of uses algorithms optimized for such a hierarchical interconnect. MAGPIE does take the hierarchical structure of the network topology into account, using separate algorithms for the wide area level and the local area level, with cluster coordinators acting as intermediates.

MPI defines a group of reduction operations: Reduce, Reduce-Scatter, All-Reduce, and Scan. For associative operators MAGPIE uses wide area optimal algorithms.

For reduction operations with short data vectors, MAGPIE uses algorithms that are nearly wide area optimal. For non-associative operators, when execution order matters for correctness, MAGPIE's algorithms are no longer wide area optimal, although by using better broadcast and scan algorithms, performance is still improved over MPICH for most reduction operations.

Although the MPI standard recognizes the possibility of exploiting associativity for optimization, it does not specify a standard way to implement this feature.

For a wide area latency of 10 ms and a bandwidth of 1 MByte/s, the measurements of the individual operations show improvements over MPICH that vary between a factor of 3 to 8, depending on the operation, the number of clusters, and the message length. For the application kernels, QR, MMUL, and TRI, MAGPIE consistently outperformed MPICH by a factor of 2 or more.

The current version of MAGPIE assumes a static topology. In future work the best tree shape can be computed dynamically based on wide area latency and bandwidth as measured at run time, as well as message size.

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 heavily use the collective operations, the MAGPIE library offers transparent optimization, and completely hides non-uniformity.

The system is available as a plug-in for 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. 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] Argonne National Laboratory. MPICH implementation home page. http://www.mcs.anl.gov/Projects/mpi/mpich/, 1995.

[3] S. Bae, D. Kim, and S. Ranka. Vector Reduction and Prefix Computation on Coarse-Grained, Distributed-Memory Parallel Machines. In *IPPS-98, International Parallel Processing Symposium*, 1998.

[4] 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), 1998.

[5] 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.

[6] 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, 1998.

[7] M. Bernaschi and G. Iannello. Collective Communication operations: experimental results vs. theory. *Concurrency: Practice and Experience*, 10(5):359–386, April 1998.

[8] M. Bernaschi, G. Iannello, and M. Lauria. Efficient Implementation of Reduce-Scatter in MPI. Submitted for publication, 1998. Available from http://grid.grid.unina.it/~iannello/dapaa.htm.

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

[10] 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.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* M. I. T. Press, 1990.

[12] 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.

[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(11), 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.

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

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

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

[18] 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.

[19] 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.

[20] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. Submitted for publication, 1998. Available from http://www.cs.vu.nl/albatross/.

[21] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

[22] 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.

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

[24] Message Passing Interface Forum. MPI Standard document, Version 1.1. Available from http://www.mcs.anl.gov/Projects/mpi/standard.html, 1995.

[25] 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.

[26] 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*, Orlando, FL, Jan. 1999.

[27] 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/.