Improving the Execution Time of Global Communication Operations

Matthias Kühnemann[*]
Fakultät für Informatik
TU Chemnitz
09107 Chemnitz, Germany
kumat@informatik.tu–chemnitz.de

Thomas Rauber
Fakultät für Mathematik u. Physik
Universität Bayreuth
95445 Bayreuth, Germany
rauber@uni–bayreuth.de

Gudula Rünger
Fakultät für Informatik
TU Chemnitz
09107 Chemnitz, Germany
ruenger@informatik.tu–chemnitz.de

ABSTRACT

Many parallel applications from scientific computing use MPI global communication operations to collect or distribute data. Since the execution times of these communication operations increase with the number of participating processors, scalability problems might occur. In this article, we show for different MPI implementations how the execution time of global communication operations can be significantly improved by a restructuring based on orthogonal processor structures. As platform, we consider a dual Xeon cluster, a Beowulf cluster and a Cray T3E with different MPI implementations. We show that the execution time of operations like *MPI_Bcast()* or *MPI_Allgather()* can be reduced by 40% and 70% on the dual Xeon cluster and the Beowulf cluster. But also on a Cray T3E a significant improvement can be obtained by a careful selection of the processor groups. We demonstrate that the optimized communication operations can be used to reduce the execution time of data parallel implementations of complex application programs without any other reordering of the computation and communication structure.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications—*Concurrent, distributed, and parallel languages*; D.3.1 [**Programming Techniques**]: Concurrent Programming—*distributed programming, parallel programming*

General Terms

Algorithm, Measurement, Performance, Experimentation

Keywords

orthogonal processor groups, global communication operations, parallel programs, MPI

[*]Supported by DFG (Deutsche Forschungsgemeinschaft).

1. INTRODUCTION

Parallel machines with distributed address space are widely used for the implementation of applications from scientific computing, since they provide good performance for a reasonable price and portable message-passing programs can be written using message-passing standards like MPI or PVM. For most applications, like grid-based computations, a data-parallel execution usually leads to good performance. But for target machines with a large number of processors, data parallel implementations may lead to scalability problems, in particular when collective communication operations are used for exchanging data. Often the scalability can be improved by re-formulating the program as a mixed task and data parallel implementation. This can be done by partitioning the computations into multiprocessor tasks and by assigning the tasks to disjoint processor groups for execution such that one task is executed by the processors of one group in a data parallel way, but different independent tasks are executed concurrently by disjoint processor groups. The advantage of a group-based execution is caused by the communication overhead of collective communication operations whose execution time shows a logarithmic or linear dependence on the number of participating processors, depending on the communication operation and the target machine.

Another approach to reduce the communication overhead is the use of orthogonal processor groups which are based on an arrangement of the set of processors as a virtual two- or higher-dimensional grid and a fixed number of decompositions into disjoint processor subsets representing hyperplanes [15]. To use orthogonal processor groups the application has to be re-formulated such that it consists of tasks that are arranged in a two- or higher-dimensional task grid that is mapped onto the processor grid. The execution of the program is organized in phases. Each phase is executed on a different partitioning of the processor set and performs communication in the corresponding processor groups only. For many applications, this may reduce the communication overhead considerably but requires a specific potential of parallelism within the application and a complete rearrangement of the resulting parallel program.

In this article we consider a different approach to reduce the programming overhead. Instead of rearranging the entire program to a different communication structure, we use the communication structure given in the data parallel program. But for each collective communication operation, we rearrange the structure of the operation such that it internally uses an orthogonal arrangement of the processor set. Using this approach a significant reduction in the execution time can be observed for different target platforms and different MPI implementations. The most significant improvement results for *MPI_Allgather()* operations. This is especially important as these operations are often used in scientific computing. Examples are iterative methods where *MPI_Allgather()* operations are used to collect data from different processors and to make this data available to each processor for the next time step. The advantage of the approach is that the application does not have to provide a specific potential of parallelism and that all programs using collective communication can take advantage of the improved communication. Also no rearrangement of the program is necessary.

The internal rearrangement of the collective communication operations is done on top of MPI on the application programmers level. So, the optimization can be used on a large range of machines providing MPI. As target platforms we consider a Cray T3E-1200, a Beowulf cluster with 528 processors connected by a fast Ethernet interconnection network and a dual Xeon cluster with a SCI network.

The rest of the paper is organized as follows. Section 2 describes how collective communication operations can be re-arranged such that they consist of different steps, each performed on a subset of the entire set of processors. Section 3 applies the improved operations in the context of larger application programs and shows the resulting improvements. Section 4 discusses related work, Section 5 concludes the paper.

2. ORTHOGONAL STRUCTURES FOR REALIZING COMMUNICATION OPERATIONS

Many implementations of the MPI standard are available, including highly-tuned versions for proprietary massively-parallel processors (MPPs), such as the Cray T3E, as well as mostly hardware-independent implementations such as MPICH [5] and LAM-MPI [14], which have been ported to run on a vast array of machine types.

In this section we describe how collective communication operations can be realized in phases based on an orthogonal partitioning of the processor set. The resulting orthogonal realizations can be used for arbitrary communication libraries that provide the collective communication operations. We demonstrate this for MPI considering the following collective communication operations: a *single-broadcast* operation (*MPI_Bcast()*), a *gather* operation (*MPI_Gather()*), a *scatter* operation (*MPI_Scatter()*), a *single-accumulation* operation (*MPI_Reduce()*), a *multi-accumulation* operation (*MPI_Allreduce()*) and a *multi-broadcast* operation (*MPI_Allgather()*).

2.1 Realization using a two-dimensional processor grid

We assume that the set of processors is arranged as a two-dimensional grid with a total number of $p = p_1 \cdot p_2$ processors. The grid consists of p_1 row groups $R_1, ..., R_{p_1}$ and p_2 column groups $C_1, ..., C_{p_2}$ with $|R_q| = p_2$ for $1 \leq q \leq p_1$ and $|C_r| = p_1$ for $1 \leq r \leq p_2$. The row groups provide a partitioning into disjoint processor sets. The disjoint processor sets resulting from column groups are orthogonal to the row groups. Using these two partitioning, the communication operations can be implemented in two phases, each working on a different partitioning of the processor grid. Based on the processor grid and the two partitioning induced, group and communicator handles are defined for the concurrent communication in the row and column groups. Based on the 2D grid arrangement, each processor belongs to both a row group and a column group. Figure 1 illustrates a set of 6 processors $P_0, P_1, ..., P_5$ arranged as $p_1 \times p_2 = 3 \times 2$ grid. The overhead for the processor arrangement itself is very small; only two functions to create the groups are required and the arrangement only has to be performed once for an entire application program.

2.1.1 *Single-Broadcast*

In a single-broadcast operation, a root processor sends a block of data to all processors in the communicator domain. Using the 2D processor grid as communication domain, the root processor first broadcasts the block of data within its column group C_1 (leader group). Then each of the receiving processors acts as a root in its corresponding row group and broadcasts the data within this group (concurrent group) concurrently to the other broadcast operations. Figure 2 illustrates the resulting two communication phases for the processor grid from Figure 1 with processor P_0 as root of the broadcast operation. Processors P_0, P_2 and P_4 form the leader group.

2.1.2 *Gather*

For a gather operation, each processor contributes a block of data; the root processor collects the blocks in rank order. For an orthogonal realization, the data blocks are first collected within the row group by concurrent group based gather operations such that the data blocks are collected by the processor belonging to that column group (leader group) to which the root of the global gather operation also belongs to. In a second step, a gather operation is performed within the leader group only and collects all data blocks at the root processor specified for the global gather operation. If b is the size of the original message, each processor in the leader group contributes a data block of size $b \cdot p_2$ for the second communication step. The order of the messages collected at the root processor is preserved.

Figure 3 (upwards) illustrates the two phases for the processor grid from Figure 1 where processor P_i contributes data block A_i.

2.1.3 *Scatter*

A scatter operation is the dual operation to a gather operation, so a scatter operation can be realized by reversing the order of the two phases used for a gather operation: first, the messages are scattered in the leader group, such that each processor in the leader group obtains all messages for processors in the row group to which it belongs to; then the messages are scattered in the row groups by concurrent group-based scatter operation.

2.1.4 *Single-Accumulation*

For a single-accumulation operation, each processor contributes a buffer with n elements; the root processor accumulates the values of the buffer with a specific reduction operation, like MPI_SUM. For an orthogonal realization, the

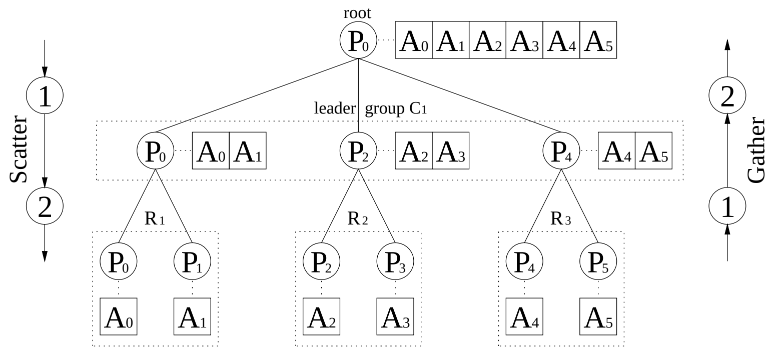


Figure 3: Illustration of an orthogonal realization of a *gather* operation (upward) and a *scatter* operation (downward) with 6 processors and root processor P_0 for 3 concurrent groups R_1, R_2 and R_3 of 2 processors each. In step (1) for the *gather* operation, processors P_0, P_2, P_4 concurrently collect messages from its row groups. In step (2), the leader group collects the messages built up the previous step.

buffers are first reduced within the row group by concurrent group-based reduce operations such that the buffer are accumulated in that column group to which the root of the global reduce operation belongs to. In a second step, a reduce operation is performed within the leader group, thus accumulating all values of the buffer at the specific root processor. The number of elements are always the same, which means that all messages have the same size in both phases.

2.1.5 Multi-Accumulation

For a multi-accumulation operation, each processor contributes a buffer with n elements and the operation makes the result buffer available for each processor. Using a 2D processor grid, the operation can be implemented in two steps: first, a group-based multi-accumulation operation is executed concurrently within the row groups, thus making the result buffer available to each processor of every row group. Second, concurrent group-based multi-accumulation operation are performed to reduce this buffer within the column groups. The messages have the same size in both phases.

2.1.6 Multi-Broadcast

For a multi-broadcast operation, each processor contributes a data block of size b and the operation makes all data blocks available in rank order for each processor. Using a 2D processor grid, the operation can be implemented in two steps: first, group-based multi-broadcast operations are executed concurrently within the row groups, thus making each block available for each processor within column groups, see Figure 4 for an illustration. Second, concurrent group-based multi-broadcast operations are performed to distribute the data blocks within the column groups; for this operation, each processor, contributes messages of size $b \cdot p_2$. Again, the original rank order of data blocks is preserved.

2.2 MPI Performance results in isolation

To investigate the performance of the implementation described in Section 2.1, we consider the communication on three different platforms with a distributed address space, a Cray T3E-1200, a small dual Xeon cluster and a large Beowulf-Cluster (CLiC). The T3E uses a three-dimensional torus network. The six communication links of each node are able to simultaneously support hardware transfer rates of 600 MB/s. The Beowulf Cluster CLiC ('**C**hemnitzer **L**inux **C**luster') is built up of 528 Pentium III processors clocked at 800 MHz. The processors are connected by two different networks, a communication network and a service network. Both are based on the fast-Ethernet-standard, i.e. the processing elements (PEs) can swap 100 MBit per second. The service network (Cisco Catalyst) allows external access to the cluster. The communication network (Extreme Black Diamond) is used for inter-process communication between the PEs. On the CLiC, LAM MPI 6.3 b2 and MPICH 1.2.4 were used for the experiments.

The Xeon cluster is built up of 16 nodes and each node consists of two Xeon processors clocked at 2 GHz. The nodes are connected by three different networks, a service network and two communication networks. The service network and one communication network are based on the fast-Ethernet-standard and the functionality is similar to the two interconnection networks of the CLiC. Additionally, a high performance interconnection network based on Dolphin SCI interface cards is available. The SCI network is connected as 2-dimensional torus topology and can be used by the ScaMPI (SCALI MPI) [4] library. The fast-Ethernet based networks are connected by a switch and can be used by two portable MPI libraries, LAM MPI 6.3 b2 and MPICH 1.2.4.

In the following, we present runtime tests for 48 and 96 processors on the T3E and CLiC and for 16 and 32 processors on the dual Xeon cluster. For other processor numbers, similar results are obtained. For 48 processors 8 different two-dimensional layouts ($2 * 24$, $3 * 16$, ..., $24 * 2$) and for 96 processors 10 different layouts ($2 * 48$, $3 * 32$, ..., $48 * 2$) are possible. For the runtime tests, we have used message sizes between 10 KBytes and 500 KBytes. This denotes the block of data contributed (e.g. *MPI_Gather()*) or obtained (e.g. *MPI_Scatter()*) by each participating processor. The following figures show the minimum, average and maximum performance improvements achieved by the orthogonal implementation described in Section 2.1 compared with the original MPI implementation over the entire interval of message sizes.

2.2.1 Orthogonal realization using LAM-MPI on the CLiC

On the CLiC, the orthogonal implementations based on the LAM-MPI library lead to the highest performance improvements for most collective communication operations. The orthogonal realizations of a *MPI_Bcast()*,

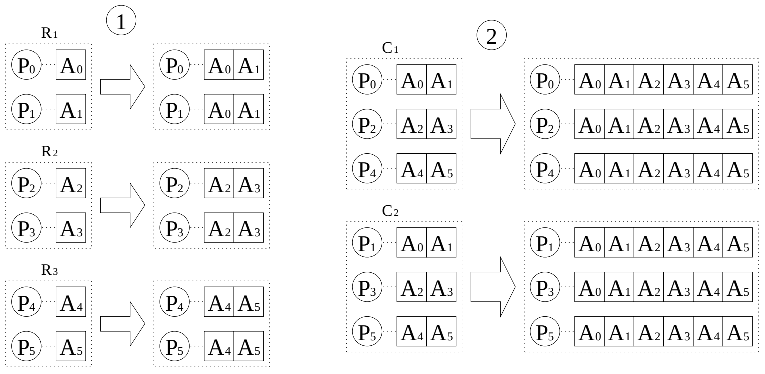


Figure 4: Illustration of an orthogonal implementation of *multi-broadcast* operation with 6 processors and root processor P_0. The operation may be realized by 3 concurrent groups R_1, R_2 and R_3 of 2 processors each and 2 orthogonal groups C_1 and C_2 of 3 processors each. Step (1) shows concurrent multi-broadcast operations on row groups and step (2) shows concurrent multi-broadcast operations on column groups.

MPI_Allgather() and *MPI_Allreduce()* operation show the most significant performance improvements. All partitions show a considerable improvement, but the largest improvements can be obtained when using a layout for which the number of row and column groups are about the same. The *MPI_Bcast()* operation shows significant average improvements of more than 20% for 48 and 40% for 96 processors, respectively, using balanced grid layouts, see Figure 5 (top). The orthogonal implementation of a *MPI_Allreduce()* operation shows average improvements of more than 20% for 48 and 30% for 96 processors, respectively, again using balanced group sizes, see Figure 6 (top). The execution time of the *MPI_Allgather()* operation can be dramatically improved by an orthogonal realization, see Figure 6 (bottom). For some of the group partitioning, improvements of over 60% for 48 and 70% for 96 processors, respectively, can be obtained. The difference between the minimum and maximum performance enhancements are extremely small, which means that this method leads to a reliable improvement for all message sizes.

The main reason for the significant performance improvements of these three collective communication operations achieved by orthogonal realization is the specific implementation of the *MPI_Bcast()* operation in LAM-MPI. The algorithm of the *MPI_Bcast()* operation to distribute the block of data does not exploit the star network topology of the CLiC, since the algorithm uses a structure describing a tree topology. In general, the orthogonal realization leads to a better utilization of the network, caused by a more balanced communication pattern. Both the *MPI_Allgather()* and the *MPI_Allreduce()* operation use a *MPI_Bcast()* operation to distribute the block of data to all participating processors. More detailed, the *MPI_Allreduce()* operation is composed of a *MPI_Reduce()* and a *MPI_Bcast()* operation. First the root processor reduces the blocks of data from all members of the processor group and broadcasts the result buffer to all processors participating in the communication operation. The message size is constant in both phases. The improvements correspond to the performance enhancements of the *MPI_Bcast()* operation, since the preceding *MPI_Reduce()* operation with orthogonal structure leads to a small performance degradation. The *MPI_Allgather()* operation is composed of a *MPI_Gather()* and a *MPI_Bcast()* operation in LAM-MPI. First the root processor collects blocks of data from all members of the processor group and broadcasts the entire message to all processors participating in the communication operation. The root processor broadcasts a considerably larger message of size $b \cdot p$, when b denotes the original message size and p is the number of participating processors. The significant improvements are again caused by the execution of the *MPI_Bcast()* operation corresponding to the larger message size.
The orthogonal implementation of the *MPI_Gather()* operation shows a small, but persistent average performance improvement for all grid layouts of more than 1% for 48 and 2% for 96 processors, respectively, see Figure 5 (bottom). There are only small variations of the improvements obtained for different layouts, but using the same number of row and column groups again leads to the best average performance.

An average performance degradation can be observed for the *MPI_Scatter()* and the *MPI_Reduce()* operation. Only for specific message sizes, a small performance improvement can be obtained, not shown in a figure.

2.2.2 Orthogonal realization using MPICH on the CLiC

The performance improvements on the CLiC based on the MPICH library are not as significant as with LAM-MPI, but also with MPICH persistent enhancements by an orthogonal realization can be obtained for some collective communication operations.
The orthogonal implementations of the *MPI_Gather()* and *MPI_Scatter()* operations lead to small, but persistent performance enhancements. For the *MPI_Gather()* and *MPI_Scatter()* operations more than 1% for 48 and 2% for 96 processors, respectively, can be obtained using balanced grid layouts. Depending on the message size up to 5% performance enhancements can be obtained with an optimal grid layout in the best case (no figure shown).

The orthogonal realization of the *MPI_Allgather()* operation leads to dramatical performance improvements for message sizes in the range of 32 KBytes and 128 KBytes, see Figure 7 (top); for larger message sizes up to 500 KBytes slight performance degradation between 1 % and 2 % can be observed. The main reason for the large differences in the improvements depending on the message size are the different protocols used for short and long messages. Both protocols are realized using non-blocking *MPI_Isend()* and *MPI_Irecv()* operations. For messages up to 128 KBytes, an

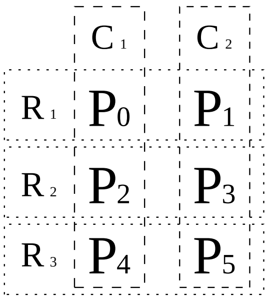


Figure 1: A set of 6 processors arranged as a two-dimensional grid with $p_1 = 3$ row groups and $p_2 = 2$ column groups in row-oriented mapping.

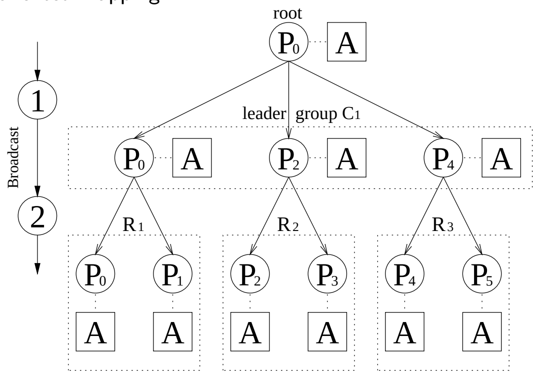
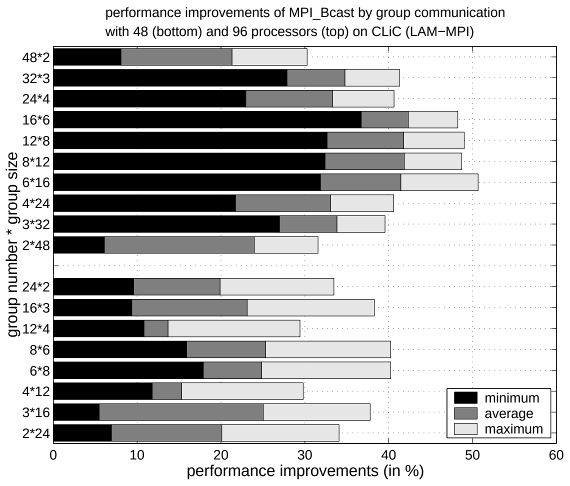


Figure 2: Illustration of an orthogonal realization of an *single-broadcast* operation with 6 processors and root processor P_0 realized by 3 concurrent groups of 2 processors each. In step (1), processor P_0 sends the message A within its column group C_1; this is the leader group. In step (2), each member of the leader group sends the message within its row group.

eager protocol is used where the receiving processor stores messages that arrive before the corresponding *MPI_Irecv()* operation has been activated in a system buffer that is dynamically allocated each time that such a message arrives. Issuing the *MPI_Irecv()* operation leads to copying the message from the system buffer to the user buffer. For messages that are larger than 128 KBytes, a *rendezvous* protocol is used that is based on request messages send by the destination processor to the source processor as soon as a receive operation has been issued, so that the message can be directly copied into the user buffer. The reason for the large improvements shown in Figure 7 (top) is caused by the fact that the asynchronous standard realization of the *MPI_Allgather()* operation leads to an allocation of a temporary buffer and a succeeding copy operation for a large number of processors whereas the orthogonal group-based realization uses the *rendezvous* protocol in the second communication phase according to an increased message size $b * p_2$.

2.2.3 Orthogonal realization using LAM-MPI on the dual Xeon cluster

The Xeon Cluster consists of 16 nodes with two processors per node. The processors participating in a communication operation are assigned to the nodes in a cyclic order



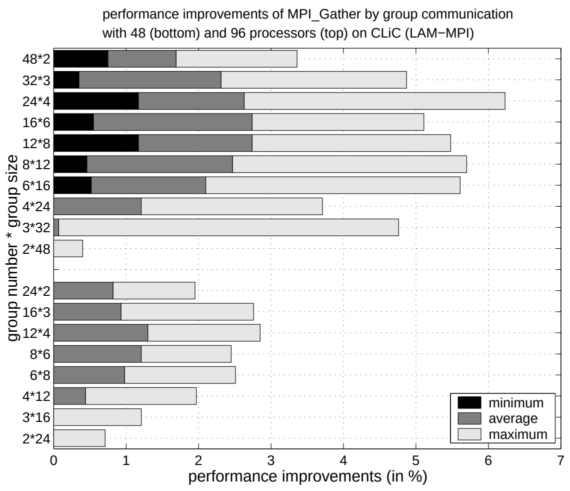
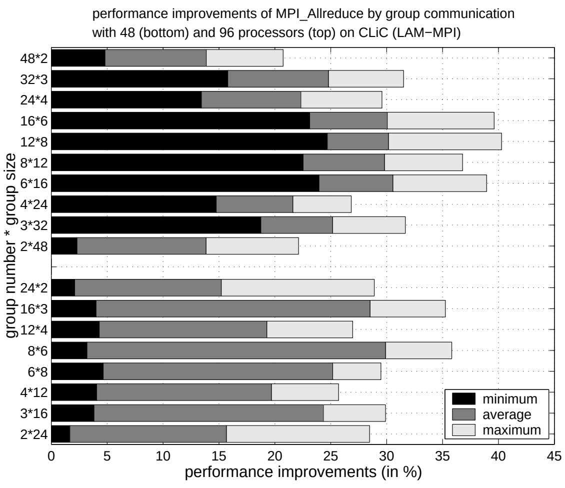


Figure 5: Performance improvements by group-based realization of *MPI_Bcast()* (top) and *MPI_Gather()* (bottom) with 48 and 96 processors on the CLiC (LAM-MPI).

to achieve a reasonable utilization of both interconnection networks. For runtime tests with 16 processors all 16 nodes are involved, i.e., processor i uses one physical processor of node i for $0 \leq i \leq 15$. When 32 processors participate in the communication operation, node i provides the processors i and $i + 16$ for $0 \leq i \leq 15$.

The performance results of the different communication operations are similar to the performance enhancements using LAM-MPI on the CLiC. The main reason is that both platforms use a star network topology, the same interconnection network (fast-Ethernet) and the same realization of communication operation based on the LAM-MPI library. Figure 8 shows as example that for an *MPI_Bcast()* (top) and an *MPI_Allgather()* (bottom) operation similar performance improvement as on the CLiC can be obtained. Because of the specific processor arrangement of the cluster the performance improvements of the various two-dimensional group layouts differ from the performance results on the CLiC, such that a balanced grid layout does not necessarily lead to the best average performance improvement. Concerning performance improvements and grid layouts similar performance improvements on the CLiC can be observed for the remaining collective MPI communication operations.



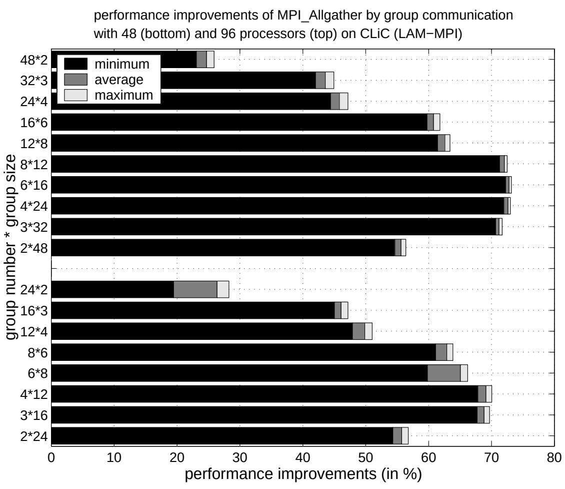
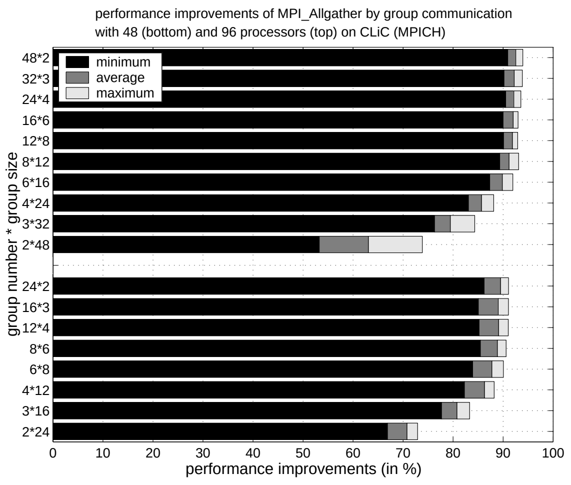


Figure 6: Performance improvements of *MPI_Allreduce()* (top) and *MPI_Allgather()* (bottom) by group communication with 48 and 96 processors on the CLiC (LAM-MPI).



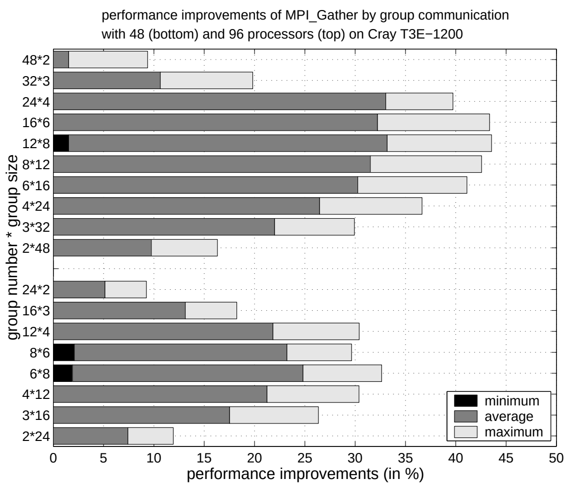


Figure 7: Performance improvements by group-based realization of *MPI_Allgather()* for message sizes in the range of 32 KBytes and 128 KBytes on the CLiC (MPICH) (top) and *MPI_Gather()* for messages sizes between 10 KBytes and 500 KBytes on the Cray T3E-1200 (bottom).

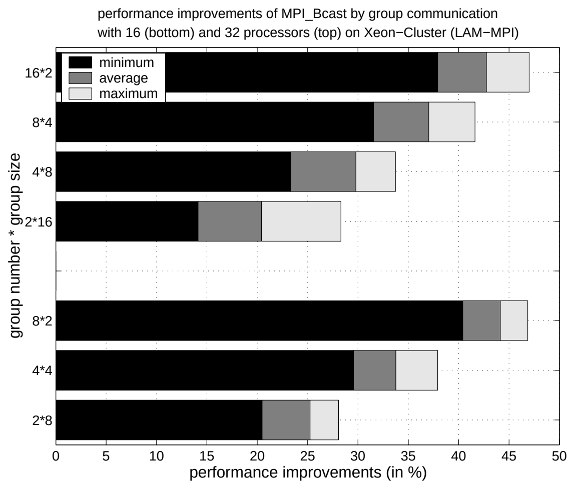
2.2.4 Orthogonal realization using ScaMPI on the dual Xeon cluster

In general, collective communication operations using the two-dimensional SCI torus are significantly faster than operations using an Ethernet network. Depending on the specific communication operation the SCI interface is by a factor of 100 faster than the Ethernet network. Several collective communication operations using ScaMPI on SCI still show performance improvements obtained by orthogonal group realization, see Figure 9 for an *MPI_Gather()* (top) and *MPI_Allgather()* (bottom) operation for smaller message sizes. For *MPI_Scatter()* similar performance results like for *MPI_Gather()* can be observed. For *MPI_Bcast()* and the accumulation operations slight performance degradations can be observed. The assignment of processors participating in the communication operation to the cluster nodes is done as described in Section 2.2.3.

2.2.5 Orthogonal realization on the Cray T3E-1200

For *MPI_Bcast()* and *MPI_Allgather()* operations, good performance improvements up to 20% can be obtained on the T3E using suitable grid layouts for messages in the range of 10 KBytes and 500 KBytes. The execution times of the orthogonal realizations are quite sensible to the grid layout and the specific message size, i.e., other grid layouts lead to smaller improvements or may even lead to performance degradation. Moreover, there is a large variation of the performance improvements with the message size. This is especially true for large messages where messages of similar size may lead to a significant difference in the performance improvement obtained. This leads to the large differences between the minimum and maximum improvement. In contrast to this smaller message sizes in the range of 10 KBytes and 100 KBytes lead to persistent average performance improvements for both operations, see Figure 10 for the *MPI_Bcast()* (top) and *MPI_Allgather()* (bottom) operations.

For the *MPI_Gather()* operation a significant performance improvement of more than 20% for 48 and 30% for 96 processors, respectively, can be obtained, see Figure 7 (bottom). For small message sizes, a slight performance degradation can sometimes be observed; for this reason, there is no minimum improvement shown for most of the layouts in the figure. For message sizes between 128



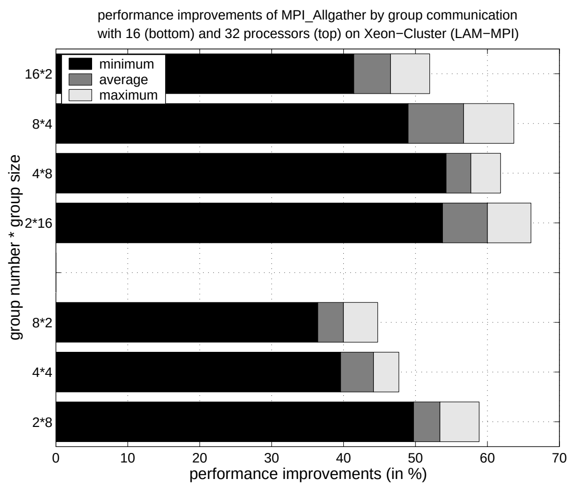
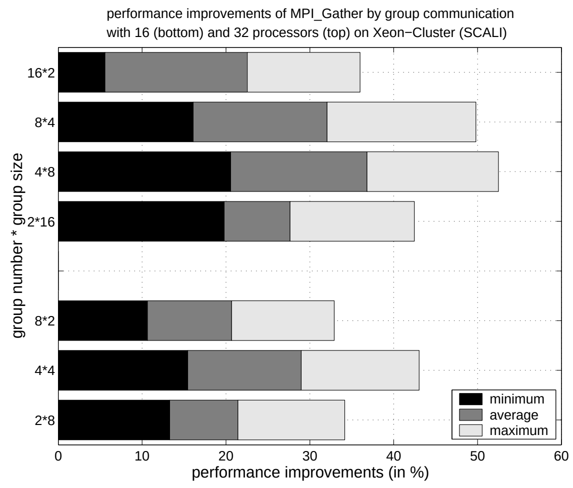


Figure 8: Performance improvements by group-based realization of *MPI_Bcast()* (top) and *MPI_Allgather()* (bottom) with 16 and 32 processors on the dual Xeon cluster (LAM-MPI).



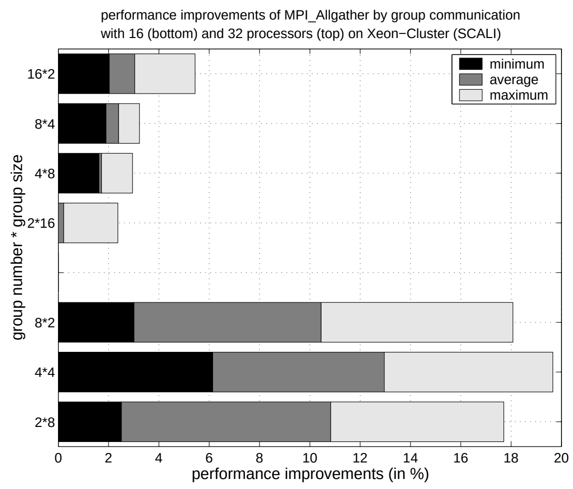


Figure 9: Performance improvements by group-based realization of *MPI_Gather()* (top) for message sizes in the range of 560 Byte and 64 KByte and *MPI_Allgather()* (bottom) for message sizes between 100 KByte and 500 KByte on the Xeon cluster (ScaMPI).

KBytes and 500 KBytes, the improvements obtained are nearly constant. The runtimes for *MPI_Gather()* operations increase more than linearly with the number p of processors. This effect is caused by the fact that the root processor becomes a bottleneck when gathering larger messages. This bottleneck is avoided when using orthogonal group communication.

2.3 Hierarchical processor group organization

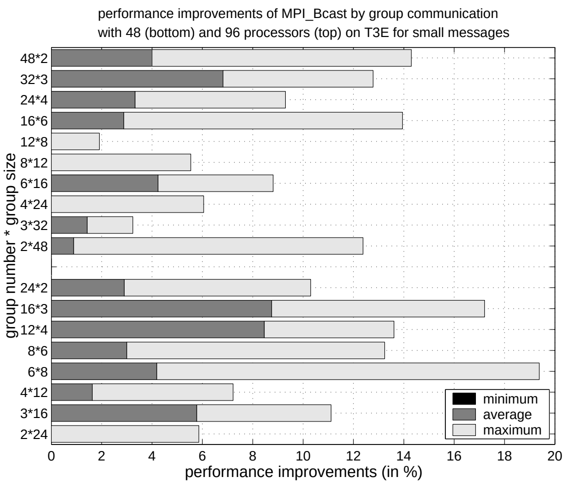
The orthogonal realization of communication operations presented in Section 2.1 can be recursively applied to the internal communication organization of the leader group or the concurrent groups, so that the communication in the leader group or the concurrent groups can be performed by again applying an orthogonal structuring of the group. Each hierarchical decomposition of a processor group leads to a new communication phase. For a fixed number of processors, the hierarchical decomposition can be selected such that the best performance improvement results.

For up to 96 processors, up to three hierarchical decompositions are useful and we present runtime tests for 96 processors on the CLiC based on the LAM-MPI library and on the Cray T3E. In particular, if the original leader group contains 16 or more processors, the group is decomposed again and the communication is performed in three instead of two phases.

Compared to the two-phase realization, a hierarchical realization of the *MPI_Bcast()* and *MPI_Gather()* operation leads to additional and persistent performance improvements.

2.3.1 *Hierarchical realization using LAM-MPI on the CLiC*

On the CLiC and T3E a sufficiently large number of processors is available to arrange different grid layouts for three communication phases. Comparing a two-dimensional with a three-dimensional realization for the *MPI_Bcast()* operation, an *additional* performance improvement of up to 15% can be obtained for the CLiC based on LAM-MPI. Figure 11 (top) shows the overall performance improvements obtained by a 3-dimensional realization. The hierarchical realization for the *MPI_Gather()* operation shows no additional performance improvements compared to the two-dimensional re-



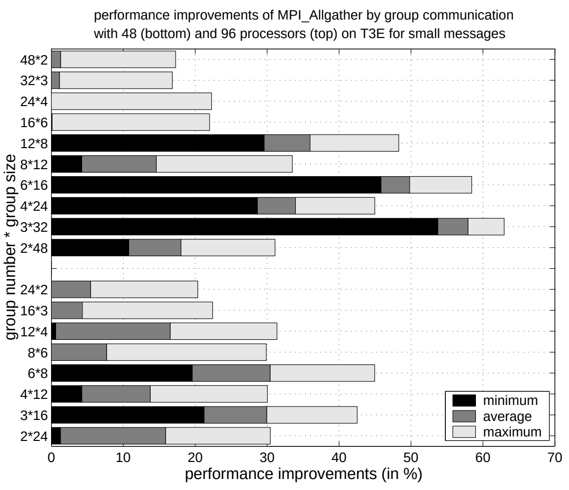
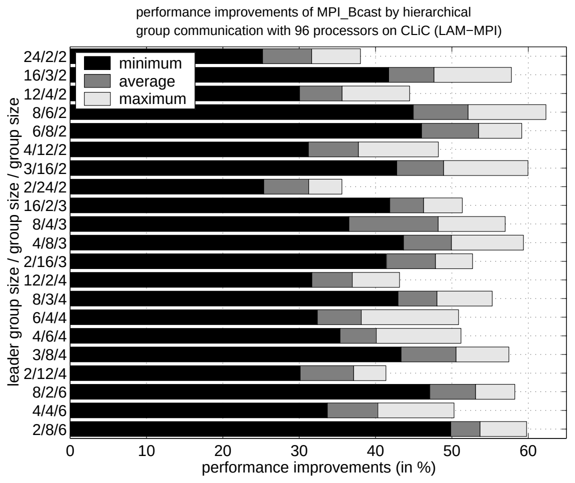


Figure 10: Performance improvements by group communication of *MPI_Bcast()* (top) and *MPI_Allgather()* (bottom) for message sizes in the range of 10 KBytes and 100 KBytes on the Cray T3E-1200.



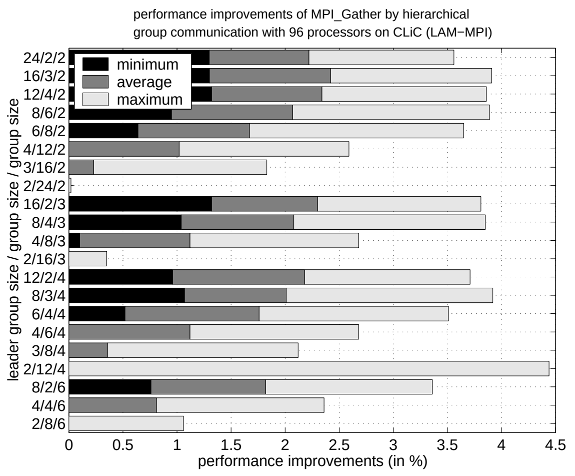


Figure 11: Overall performance improvements by hierarchical group communication of *MPI_Bcast()* (top) and *MPI_Gather()* (bottom) on the CLiC (LAM-MPI).

alization, see Figure 11 (bottom). The resulting differences between the minimum and maximum performance improvements are larger over the entire interval of message sizes.

2.3.2 Hierarchical realization on the Cray T3E-1200

For the *MPI_Bcast()* operation *all* group partitioning show an average performance improvement, compared to the runtime tests with two communication phases for message sizes up to 500 KBytes on the T3E, see Figure 12 (top). For suitable grid layouts average improvements of more than 20% can be obtained. The hierarchical realization with three communication phases for the *MPI_Gather()* operation leads to additional performance improvements, see Figure 12 (bottom). The improvements vary depending on the group partitioning. For some of the group partitionings, additional improvements of over 60 % can be obtained, which means that the hierarchical variant is more than twice as fast as the variant with two communication phases.

2.3.3 Hierarchical realization on the Xeon cluster

Figure 13 shows performance enhancements for four MPI communication operations obtained by a hierarchical orthogonal grid layout with three communication phases. Since 32 processors are available three different group layouts ($2 \times 8 \times 4$, $4 \times 4 \times 2$, $8 \times 2 \times 2$) are chosen for the Xeon cluster. Figure 13 shows the additional performance improvements for the orthogonal realization of *MPI_Bcast()*, *MPI_Allgather()* using LAM-MPI (top) and *MPI_Gather()*, *MPI_Scatter()* using ScaMPI (bottom). The orthogonal realizations using ScaMPI are obtained for smaller message sizes in the range of 560 Byte and 64 KByte, see also Section 2.2.4.

2.4 Grid Selection

For a given machine and a given MPI implementation, a suitable layout of the processor grid can be selected by performing measurements with different grid layouts and different message sizes for each of the collective communication operations to be optimized. The process of obtaining and analyzing the measurements can be automated such that for each communication operation, a suitable layout is determined that leads to smaller execution times. This process has to be done only once for each target machine and each MPI implementation and the selected layout can then be used for all communication operations in all application

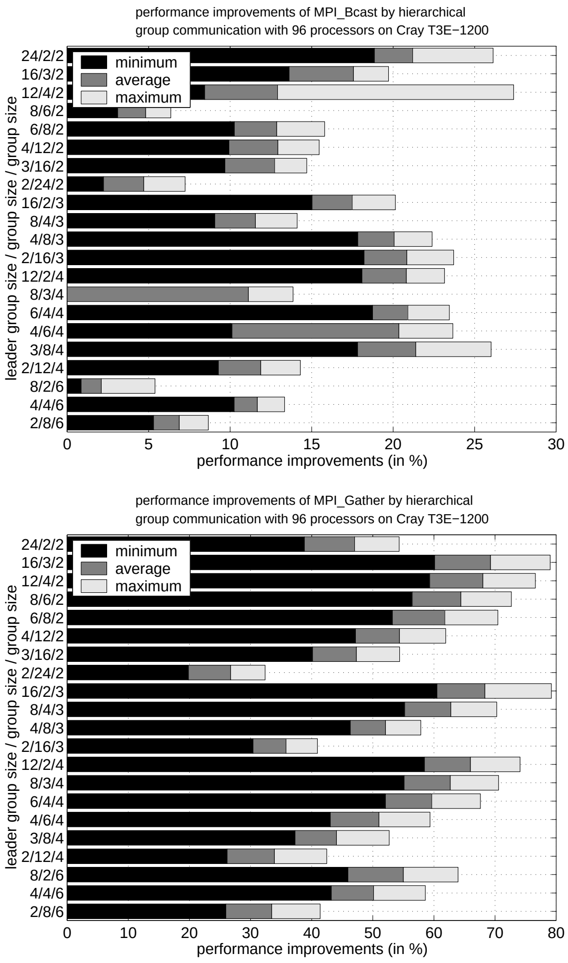


Figure 12: Overall performance improvement by hierarchical group communication of *MPI_Bcast()* (top) and *MPI_Gather()* (bottom) on the Cray T3E-1200.

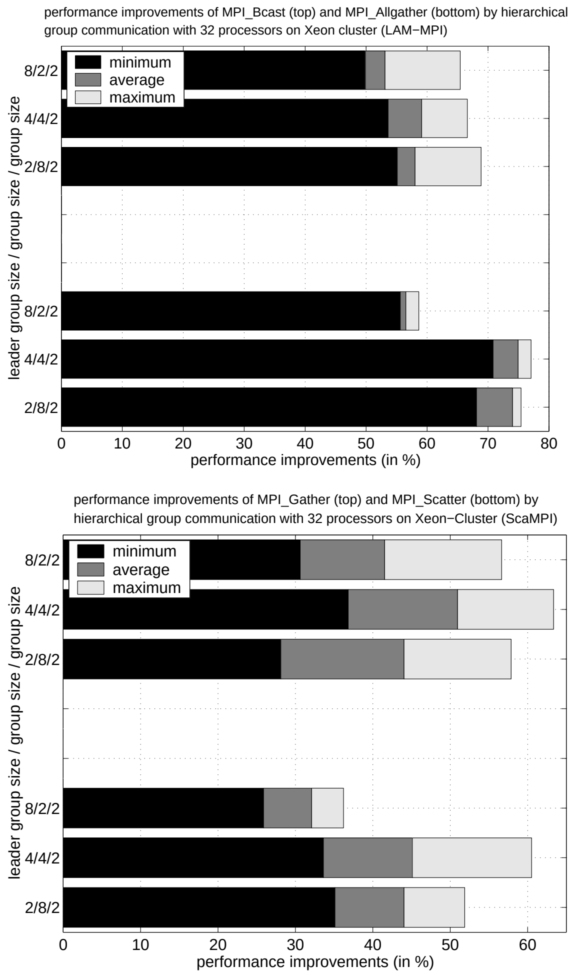


Figure 13: Overall performance improvements by hierarchical group communication of *MPI_Bcast()*, *MPI_Allgather()* using LAM-MPI (top) and *MPI_Gather()*, *MPI_Scatter()* using ScaMPI (bottom) on the Xeon cluster. The improvements for orthogonal realizations using ScaMPI (bottom) are obtained for message sizes between 560 Byte and 64 KByte.

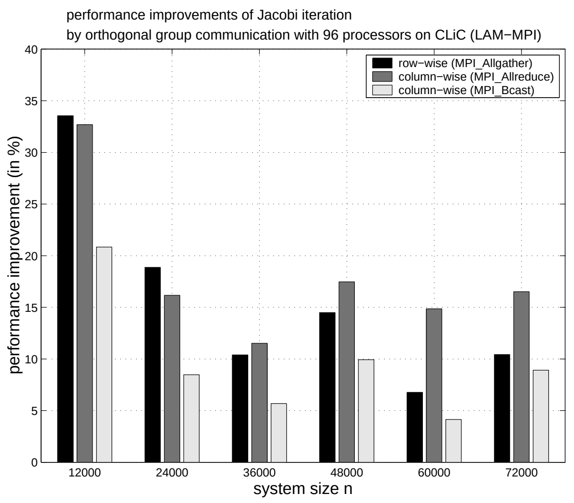
programs. In general, different optimal layouts may result for different communication operations, but our experiments with LAM-MPI, MPICH and Cray-MPI show that using the same number of row and column groups usually leads to good and persistent improvements compared to the standard implementation of the MPI operations.
Based on the measured execution times of the communication operation, it is also possible to identify intervals of message sizes such that a different grid layout is selected for a different interval, depending on the expected performance improvement. The measured data also shows whether for a specific communication operation and for a specific message size, no performance improvement is expected by an orthogonal realization. In this case, the original MPI implementation should be used.

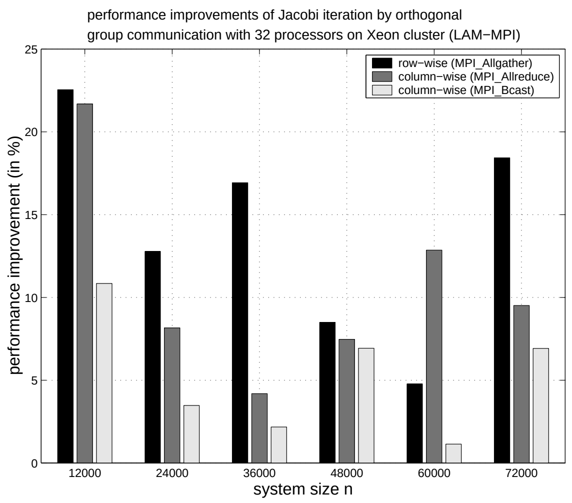
As shown in Section 2.2 the performance improvements obtained by an orthogonal realization depend on different features, like architectural characteristics of the parallel target platform or the internal implementation of the global communication operation. This means that there is no warranty for performance improvements by using orthogonal realization of global communication operation. But the runtime experiments show that an orthogonal realization may lead to a significant performance improvement of the communication behavior, especially for portable message passing libraries, which have been developed with a view to run on a wide range of machine types.

The result of the analysis step is a realization of the collective communication operations that uses orthogonal realization with a suitable layout whenever it is expected that this leads to a performance improvement compared to the given MPI implementation. In this way also a hierarchical layout can be obtained and analyzed. Furthermore, as described in Section 2.3, a hierarchical grid layout can be selected such that the best performance improvement results.

3. APPLICATIONS AND RUNTIME TESTS

To investigate the efficiency improvement of the approach for entire application programs, we consider parallel implementations of the Jacobi iteration with and without optimized communication and complex application programs, the parallel Adams methods to show the performance improvements by concurrent group communication.





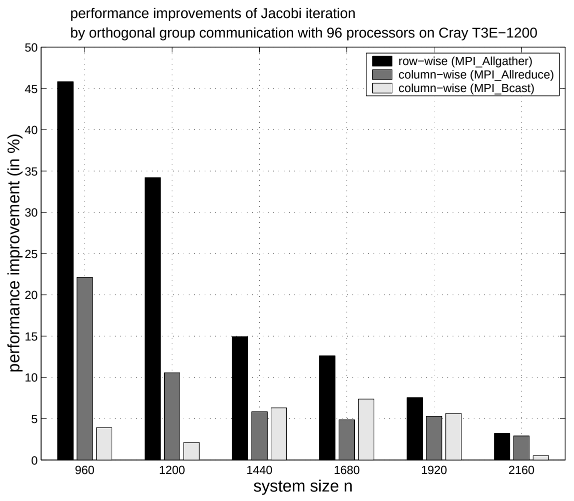


Figure 14: Performance improvements of Jacobi iteration by orthogonal group communication using balanced grid layouts on the CLiC (LAM-MPI) (top) and for smaller system sizes on the T3E (bottom).

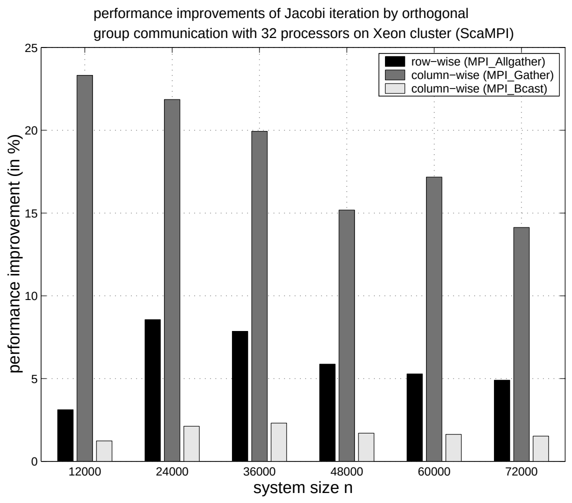


Figure 15: Performance improvements of the Jacobi iteration by orthogonal group communication on the Xeon cluster with LAM-MPI (top) and with ScaMPI (bottom).

3.1 Jacobi iteration

For the Jacobi iteration, three different implementations were investigated to distribute the iteration vector at the end of each iteration step. One possibility is to use a row-wise distribution of the iteration matrix and an *MPI_Allgather()*. The other two versions result from a column-wise distribution; one uses an *MPI_Allreduce()*, the other uses an *MPI_Reduce()* followed by an *MPI_Bcast()*. Figure 14 shows the performance improvements obtained by a 2D orthogonal structure for the CLiC with LAM-MPI (top) and the T3E (bottom). Figure 15 shows the improvements for the dual Xeon cluster using LAM-MPI (top) and ScaMPI (bottom). For the parallel realization using ScaMPI a specific implementation with *MPI_Gather()* and *MPI_Scatter()* instead of *MPI_Allreduce()* is shown in Figure 15 (bottom). The improvements are obtained for a large range of system sizes using balanced grid layouts.

3.2 Parallel Adams methods PAB und PABM

The parallel Adams methods are variants of general linear methods for solving ordinary differential equations (ODEs) $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$ proposed in [17]. General linear methods compute several stage values $\mathbf{y}_{\kappa,i}$ in each time step κ which correspond to numerical approximations of $\mathbf{y}_{\kappa,i} = \mathbf{y}(t_\kappa + a_i h)$ with abscissa vector (a_i), $i = 1, ..., k$, and stepsize $h = t_\kappa - t_{\kappa+1}$. The stage values of one time step are combined in the vector $\mathbf{Y}_\kappa = (\mathbf{y}_{\kappa,1}, ..., \mathbf{y}_{\kappa,k})$; for an ODE system of size n, this vector has size $n \cdot k$. The computation in each step is given by:

$$\mathbf{Y}_{\kappa+1} = (\mathbf{R}\otimes\mathbf{I})\mathbf{Y}_\kappa + h(\mathbf{S}\otimes\mathbf{I})\mathbf{F}(\mathbf{Y}_\kappa) + h(\mathbf{T}\otimes\mathbf{I})\mathbf{F}(\mathbf{Y}_{\kappa+1}), \quad (1)$$

for $\kappa = 1, 2, \ldots$. The resulting methods have the advantage that the computations of the parallel stages within each time step are completely independent from each other. Strong data dependencies only occur at the end of each time step. In a data parallel implementation of the PAB method, the stage values are computed one after another with all processors available. For a task parallel implementation, the stage vectors are computed by independent sets of pro-

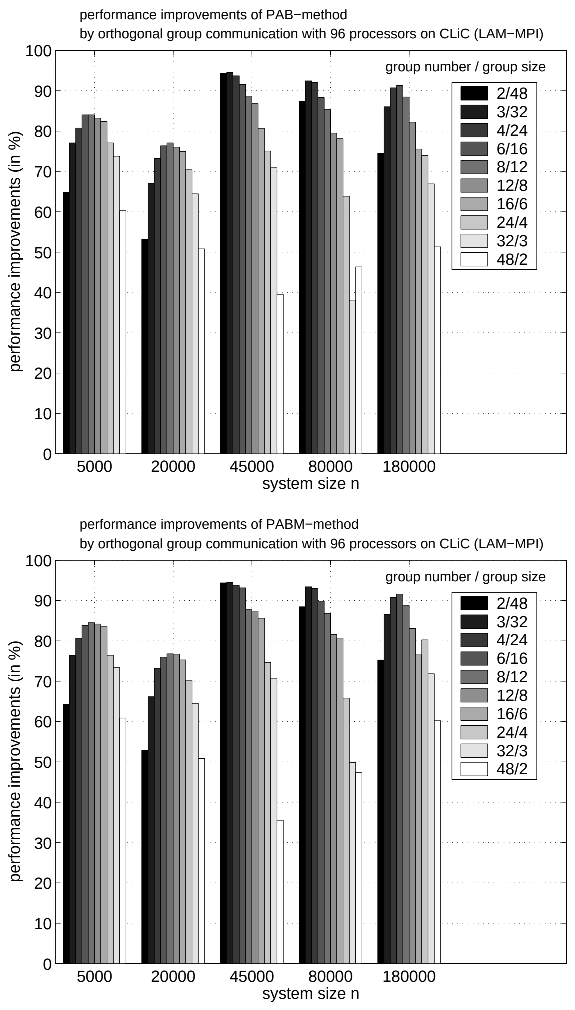


Figure 16: Performance improvements of *data parallel* PAB-method (top) and PABM-method (bottom) by orthogonal group communication on the CLiC (LAM-MPI).

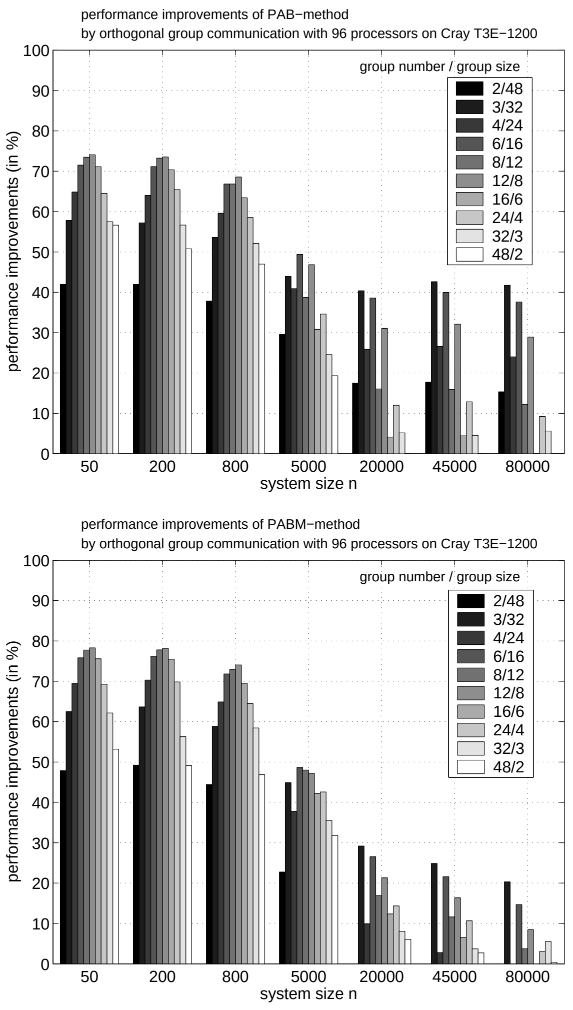


Figure 17: Performance improvements of *data parallel* PAB-method (top) and PABM-method (bottom) by orthogonal group communication for small system sizes with 96 processors on the Cray T3E-1200.

cessors that work concurrently to each other.

We consider an explicit PAB version and an implicit PABM version of the methods. The implicit methods use fix-point iteration with the PAB method as predictor. As application, an ODE system has been used that result from the spatial discretization of a reaction diffusion equation. This is a *sparse* ODE system, i.e., the evaluation time of one component of **f** is constant. We consider a data parallel, a orthogonal data parallel, a task parallel and a orthogonal task parallel implementation for varying processor numbers and different system sizes. The orthogonal realization of the data parallel implementation can be obtained as described in Section 2.2. The original MPI communication operation (*MPI_Allgatherv()*) is simply replaced by a corresponding group-based operation. The program structure of the original data parallel implementation is not rearranged. The task parallel versions require a complete reorganization of the code, but result in a small additional improvement compared to the orthogonal realization of the data parallel version.

Figures 16 and 17 show the performance improvements obtained for the PAB and PABM methods by a 2D orthogonal realization of the communication operations on the CLiC (LAM-MPI) and the T3E, respectively. The figures show the improvements of the data parallel implementation with orthogonal structure compared to the original data parallel variant for different system sizes. Figure 17 shows the PAB and PABM methods for small system sizes on the Cray T3E, confirming the performance improvements of orthogonal communication operations for small message sizes in isolation on this platform.

4. RELATED WORK

Related work comes from different research directions, including programming models and software support for scientific computing, parallel languages and libraries, and mixed task and data parallelism [2, 16]. Many environments for scientific computing are extensions to the HPF data parallel language. A good overview can be found in [6]. The HPF-2 standard supports the execution of tasks defined as computations on subranges of arrays on processor groups also defined as parts of the processor grid. Another example is HPJava which adopts the data distribution concepts of HPF but uses a high level SPMD programming model with a fixed number of logical control threads and includes collective communication operations encapsulated in a communication library. A language description is given in [18]. The concept of processor groups is supported in the sense that global data distributed over one process group can be defined and that the program execution control can choose one of the process groups to be active.

LPARX is a parallel programming system for the development of dynamic, nonuniform scientific computations supporting block-irregular data distributions [11]. KeLP extends LPARX to support the development of efficient programs for hierarchical parallel computers such as clusters of SMPs [1, 6]. In comparison to our approach, LPARX and KeLP are more directed towards the realization of irregular grid computations whereas our approach is based on regular grids using different partitions of the same set of processors. KeLP has been extended to KeLP-HPF which uses an SPMD program to coordinate multiple HPF tasks and, thus, combines regular data parallel computations in HPF with a coordination layer for irregular block-structured features on one grid [13]. An API for adaptive mesh algorithms based on LPARX is presented in [12].

MagPIE is a library of collective communication operations optimized for wide area systems [10, 9]. The underlying algorithms are designed so that a minimum amount of data is transferred over slow links of a multilayer interconnect [3]. But because of the heterogeneity of the target system, no orthogonal structures are used for the communication. MPICH-G2 [8] creates multilevel topology-aware trees for collective communication operations for heterogeneous systems and computational grids [7], but again no orthogonal structures are exploited.

5. CONCLUSION

In this paper, we have shown that the execution time of MPI collective communication operations can be significantly reduced by a restructuring of the communication operation based on a hierarchical decomposition into phases such that each phase realizes a part of the communication operation. As platforms, we have used a Cray T3E and a Beowulf cluster. On both platforms, large performance improvements have been observed for optimized communication operations in isolation and also for entire application programs using those communication operations. The orthogonal realization of collective communication operations can be used to reduce scalability problems in data parallel implementations by replacing the communication operations. This avoids a restructuring the entire communication and computation structure of the application, thus reducing the programming effort for parallel machines with a large number of processors significantly.

Acknowledgment

We thank the NIC Jülich for providing access to a T3E.

6. REFERENCES

[1] S.B. Baden and S.J. Fink. A Programming Methodology for Dual-Tier Multicomputers. *IEEE Transactions on Software Engineering*, 26(3):212–226, 2000.

[2] H. Bal and M. Haines. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, 6(3):74–84, July-August 1998.

[3] H. Bal, A. Plaat, M. Bakker, P. Dozy, and R. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. *Proc. of IPPS/SPDP 98*, pages 784–790, 1998.

[4] Scali / ScaMPI commercial MPI on SCI implemetation. http://www.scali.com/.

[5] LAM/MPI Parallel Computing. http://www.lam-mpi.org/.

[6] S.J. Fink. *A Programming Model for Block-Structured Scientific Calculations on SMP Clusters.* PhD thesis, University of California, San Diego, 1998.

[7] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. *Proc. of IPDPS 2000*, pages 377–386, 2000.

[8] N. T. Karonis, B. R. Toonen, and I. T. Foster. MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributing Computing*, 63(5):551–563, 2003.

[9] T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. *Proc. of IPDPS*, pages 492–499, 1999.

[10] T. Kielmann, R. Hofman, H. E. Bal, A. Plaat, and R. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. *ACM SIGPLAN Notices*, 34(8):131–140, 1999.

[11] S.R. Kohn and S.B. Baden. Irregular Coarse-Grain Data Parallelism under LPARX. *Scientific Programming*, 5:185–201, 1995.

[12] S.R. Kohn and S.B. Baden. Parallel Software Abstractions for Structured Adaptive Mesh Methods. *Journal of Parallel and Distributed Computing*, 61(6):713–736, 2001.

[13] J. Merlin, S.Baden, St. Fink, and B. Chapman. Multiple data parallelsim with HPF and KeLP. *J. Future Generation Computer Science*, 15(3):393–405, 1999.

[14] MPICH-A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich.

[15] T. Rauber, R. Reilein, and G. Rünger. ORT – A Communication Library for Orthogonal Processor Groups. In *Proc. of the ACM/IEEE SC 2001*. IEEE Press, 2001.

[16] D. Skillicorn and D. Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, 1998.

[17] P.J. van der Houwen and E. Mesina. Parallel adams methods. *J. of Comp. and App. Mathematics*, 101:153–165, 1999.

[18] G. Zhang, B. Carpenter, G.Fox, X. Li, and Y. Wen. A high level SPMD programming model: HPspmd and its Java language binding. Technical report, NPAC at Syracuse University, 1998.