Adaptive Selection of Communication Methods to Optimize Collective MPI Operations

Olaf Hartmann[a], Matthias Kühnemann[a], Thomas Rauber[b], Gudula Rünger[a]

[a]Department of Computer Science, Chemnitz University of Technology, 09107 Chemnitz, Germany

[b]Department of Mathematics and Physics, Bayreuth University, 95445 Bayreuth, Germany

Abstract Many parallel applications from scientific computing use collective MPI communication operations to distribute or collect data. The execution time of collective MPI communication operations can be significantly reduced by a restructuring based on orthogonal processor structures or by using specific point-to-point algorithms based on virtual communication topologies. The performance improvement depends strongly on numerous factors, like the collective MPI communication operation, the specific group layout, the message size, the specific MPI library, and the architecture parameters of the parallel target platform. In this paper we describe an adaptive approach to determine and select a specific processor group layout or communication algorithm for the realization of collective communication operations with the objective of minimizing the communication overhead. In the case that a communication method is faster than the original implementation of the collective MPI communication operation, the specific communication method is applied to perform the communication operation.

1. Introduction

The execution of data or task parallel implementations of program applications can lead to scalability problems, especially for target platforms with a large number of processors [2]. To reduce the communication time of collective MPI communication operations different approaches can be considered [5,6]. One of them is the use of orthogonal processor groups performing two or more communication phases. An orthogonal processor group is obtained by arranging subsets of processors as two- or multi-dimensional grid. In [3] we have shown how the execution time of collective MPI communication operations can be significantly reduced by orthogonal processor groups for different target platforms and different MPI implementations. Performance improvements of 40% up to 70% can be obtained for LAM-MPI communication operations, like a `MPI_Bcast()` or `MPI_Allgather()`, on a Cray T3E-1200 and a Beowulf cluster.

Since only a specific collective MPI communication operation is replaced by two or more communication phases of orthogonal processor groups, the use of this approach is completely independent of the computation or communication structure of the parallel program application. This means that all applications using MPI operations for exchanging data can benefit from the improved communication method. The overhead caused by generating the processor groups is negligible and can easily be done by using a specific communicator handle for the corresponding row or column group.

However, it is not known in advance whether a performance gain can be achieved by an orthogonal processor layout and which of the numerous processor layouts may lead to an optimal performance gain. The reason is that the performance improvement depends on numerous factors, like the specific collective MPI communication operation, the message size, the total number of processors participating in the operation, the specific MPI library, and the architecture characteristics of the target machine [8,7]. In this paper we introduce an adaptive approach that determines and automatically selects the processor layout depending on the message size, such that the best performance improvement for a corresponding collective communication operation is achieved. In addition, we consider specific realizations of collective communication using point-to-point operations. The fastest realization is then used in the parallel application program. The adaptive approach can be applied for each collective MPI communication operation and each parallel target machine providing MPI.

2. Design of Orthogonal Groups and Communication Algorithms

For a collective MPI communication operation on $p = p_1 \cdot p_2$ processors, an orthogonal group layout of processors can be arranged as row groups with p_1 processors and column groups with p_2 processors. The disjoint processor sets resulting from column groups are orthogonal to the row groups. Using the orthogonal processor group a collective MPI communication operation is performed in two phases. To perform more than two communication phases the rearrangement can be applied recursively to the row groups or the column groups, respectively. In [3] the generation and use of orthogonal processor groups is described in more detail.

Another possibility to reduce the communication overhead of a collective communication operation is an implementation based on a series of non-blocking point-to-point MPI communication operations. In this paper we consider a selection of the most promising algorithms. The algorithms are based on specific virtual communication topologies, where the exact algorithm depends on the specific collective MPI communication operation. These communication methods are integrated into a program library which provides a large variety of different realizations for each collective communication operation. Based on this library, a modeling tool can determine and select an optimal communication method depending on the message size for a given number of processors on a specific execution platform. We consider a star, a hypercube, a binomial tree and a ring as virtual communication topology to implement a corresponding collective MPI operation.

The user can perform the benchmark and the execution of the application program separately. This reduces the overhead caused by the performance benchmark when a large number of program starts of application programs is performed. The adaptive approach can easily be used by including a specific header file and linking the C program library to the program application.

3. Implementation of the adaptive approach

The basic idea of the approach is to execute a specific benchmark program that contains different implementations for each collective MPI operation with the objective of determining the fastest communication method for a specific interval of message sizes. As communication methods the vendor-specific collective MPI communication operation, orthogonal realizations with all possible two- and three-dimensional group layouts and two different communication algorithms based on virtual topologies are considered. Table 1 shows which communication topology is used for which collective MPI operations. For each collective MPI operation two algorithms P2P_A and P2P_B are implemented using non-blocking point-to-point (P2P) MPI communication operations. Benchmark programs of these algorithms are implemented as standalone C program library and can be used for each collective communication operation on each parallel target platform providing MPI. Based on the benchmark results, the fastest communication method is determined in an evaluation phase which has to be executed once for each combination of a target platform and an MPI implementation.

For each collective MPI communication operation the information gathered in the evaluation phase is stored in an information table, such that a specific interval of message sizes is mapped to the fastest communication method. This mapping can be used to select the fastest communication method in the execution phase in which the application program containing one or more collective MPI operations is executed.

MPI operation	P2P_A	P2P_B
`MPI_Bcast()`	binomial tree	binomial tree (scatter) + hypercube (allgather)
`Gather()`	star	binomial tree
`Scatter()`	star	binomial tree
`Reduce()`	star	binomial tree
`Allreduce()`	binomial tree (reduce) + binomial tree (bcast)	hypercube
`Allgather()`	ring	hypercube

Table 1: Two communication algorithms `P2P_A` and `P2P_B` are implemented to perform a collective MPI communication operation. The algorithms use non-blocking point-to-point MPI operation.

4. Parallel Target Platforms and Experimental Results

This section gives an overview of the hardware characteristics of the parallel target platforms that are used to investigate and evaluate the performance behavior of collective MPI operations with and without optimized communication methods. Section 4.2 gives a summary of the performance results on these platforms achieved by optimized communication method in isolation.

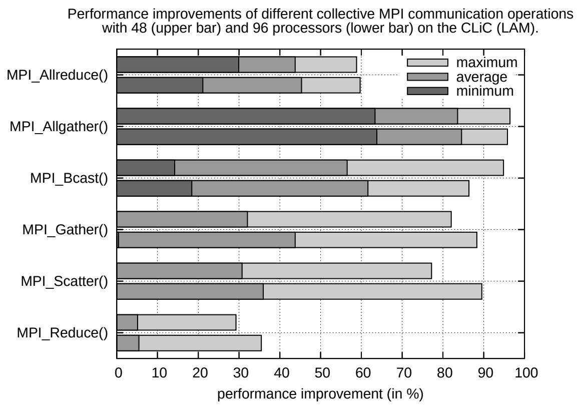
4.1. Target Platforms

The runtime experiments are performed on a Beowulf cluster (CLiC), a dual Xeon cluster, a Cray T3E and a IBM Regatta p690+ cluster, which have the following characteristics.

- The Beowulf Cluster `CLiC` (Chemnitzer `Linux` Cluster') is built up of 528 Pentium III processors clocked at 800 MHz. The processors are connected by a fast-Ethernet network which can be used by LAM MPI 6.5.6 and MPICH 1.2.5.2.
- The `Dual Xeon Cluster` is built up of 16 nodes consisting of two Xeon processors clocked at 2 GHz each. The nodes are connected by a fast-Ethernet network and a high performance interconnection network that uses Dolphin SCI interface cards. The SCI network is connected as two-dimensional torus and can be used by the ScaMPI (SCALI MPI) library [1]. The fast-Ethernet based network is connected by a switch and can be used by two portable MPI libraries, LAM MPI 6.5.6 and MPICH 1.2.5.2.
- The `Cray T3E-1200` uses a bidirectional three-dimensional torus network to connect the nodes each containing a DEC Alpha 21164 processor with 600 MHz. The six communication links of each node are able to simultaneously support hardware transfer rates of 600 MB/s.
- The `JUMP` (`JU`elich `M`ulti `P`rocessor) is a IBM Regatta p690+ cluster, that is built up of 41 SMP nodes; each node consists of 32 Power4+ processors clocked at 1.7 GHz. The nodes are interconnected by a High Performance Switch (HPS). As Message-Passing library a proprietary implementation of MPI is used for all runtime experiments, that is part of the Parallel Environment v4.1 developed by IBM.

4.2. Experimental results in isolation

A specific set of different orthogonal group layouts depends on the total number of available processors of a parallel target platform. In general, we arrange the processor groups into all possible two-dimensional or three-dimensional grid layouts based on the total number of processors participating in the communication operation. On the T3E and the CLiC, 96 processors have been used for the experimental evaluation leading to 10 different two-dimensional group layouts ($2 \times 48, 3 \times 32, 4 \times 24, 6 \times 16, 8 \times 12, 12 \times 8, 16 \times 6, 24 \times 4, 32 \times 3, 48 \times 2$). Additional performance tests with a total number of 48 processors allow 8 different two-dimensional group layouts



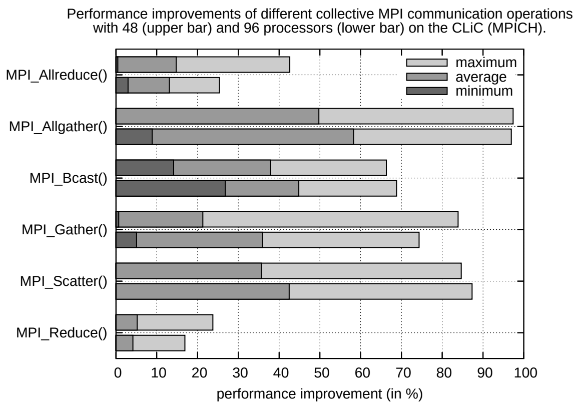


Figure 1. Summary of the performance improvements achieved by different communication methods for collective MPI operations. The diagrams show the improvements obtained with LAM-MPI (left) and with MPICH (right) on the Beowulf cluster CLiC. The bars represent the minimum, average, and maximum performance improvements over all message sizes for a total number of 48 processor (upper bar) and 96 processors (lower bar) for each MPI operation.

$(2 \times 24, 3 \times 16, 4 \times 12, 6 \times 8, 8 \times 6, 12 \times 4, 16 \times 3, 24 \times 2)$ on both platforms. On the JUMP, runtime tests with 64 processors are considered; 32 processors are available on the dual Xeon cluster. The message sizes for the runtime tests are between 1 KByte and 1024 KByte.

4.2.1. Beowulf cluster CLiC

The standard implementations of MPICH and LAM-MPI have the drawback that an unsuitable use of communication protocols may lead to significant performance degradations on several execution platforms. This can be observed, e.g., for the original `MPI_Gather()`, `MPI_Scatter()` and `MPI_Allgather()` operation on the CLiC. Different communication protocols, like the eager and the rendezvous protocol, are used to optimize the data throughput for smaller and larger message sizes. The performance degradations are caused by an unsuited selection of the specific message size at which the system switches to a different communication protocol. This disadvantage can be compensated using orthogonal processor groups. The reason is that an orthogonal realization use the significantly faster rendezvous protocol for larger messages in the second communication phase, in comparison to the slower eager protocol for smaller messages used by the original MPI operation.

We have measured the performance improvements achieved by the optimized communication methods for all message sizes considered. Figure 1 shows the minimum, average, and maximum performance improvements obtained. The figure depicts the improvements for a total number of 48 and 96 processors on the CLiC using LAM-MPI (left) and MPICH (right). Since the adaptive approach determines the most efficient communication method for separate intervals of message sizes, the figures give a good survey of the expected reduction of the communication overhead in the context of parallel program applications on the CLiC. Finally we notice that for an `MPI_Bcast()` and an `MPI_Allgather()` operation, a point-to-point based algorithm usually leads to the best performance enhancements in contrast to the `MPI_Gather()`, `MPI_Scatter()` and `MPI_Reduce()` operation, where an orthogonal realization often lead to the best improvements.

4.2.2. Dual Xeon Cluster

The network interconnection based on the Fast-Ethernet standard can be used by the LAM-MPI and the MPICH implementation. Since the same portable MPI-library and network system are in-

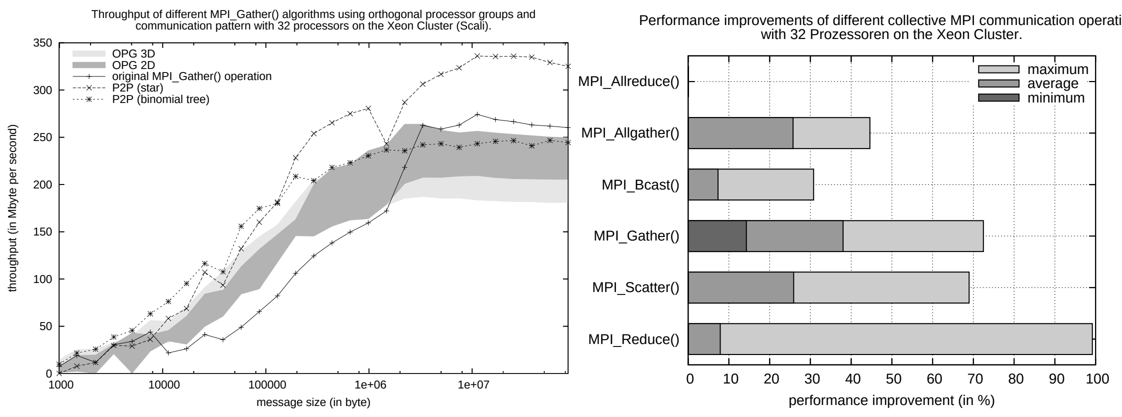


Figure 2. The left figure shows the throughput of different `MPI_Gather()` communication methods with 32 processors on the dual Xeon cluster using the SCI network. The grey areas depict the throughput achieved by orthogonal processor groups representing the fastest and the slowest group layout with two communication phases (orthogonal processor groups `OPG 2D`) and three communication phases (orthogonal processor groups `OPG 3D`). The right figure shows a summary of the performance improvements achieved by optimized communication methods for all collective MPI communication operations on the dual Xeon cluster with ScaMPI/SCI for 32 processors.

vestigated for the same collective MPI communication operations, similar experimental results are obtained as on the CLiC. The performance improvements achieved by the optimized communication methods are also significant for most of the collective MPI operations corresponding to the smaller total number of 32 processors.

In spite of the fact that the collective communication operations in ScaMPI are optimized for the SCI network architecture, a clear performance improvements can be obtained for all collective MPI operations using different communication methods, see Figure 2 (right) for all collective MPI operations. Figure 2 (left) shows the throughout of different implementations of the `MPI_Gather()` operation with ScaMPI/SCI in detail.

4.2.3. Cray T3E-1200

An optimized proprietary message-passing library is provided on the Cray T3E-1200 to use collective MPI communication operations. But nevertheless the orthogonal realizations show significant performance improvements for various collective operations on this platform. Except for the single-accumulation operation `MPI_Reduce()` and the multi-accumulation operation `MPI_Allreduce()` consistent improvements are obtained for all other collective MPI operations. As examples, Figure 3 shows the data throughput for different implementations to perform a single-broadcast operation (left) and a scatter operation (right). Both point-to-point communication algorithms lead to a performance gain for the `MPI_Bcast()` operation. Figure 4 (left) shows a summary of the performance improvements for all collective MPI operations on the T3E.

4.2.4. JUMP Cluster

Similar to the T3E, a proprietary message-passing library developed by IBM is provided as part of the Parallel Environment V 4.1 to perform a collective MPI communication operation on the JUMP. But also for this implementation all collective MPI communication operations show performance improvements in the average using orthogonal realizations and point-to-point algorithms over a wide range of message sizes for a total number of 32 and 64 processors, see Figure 4 (right).

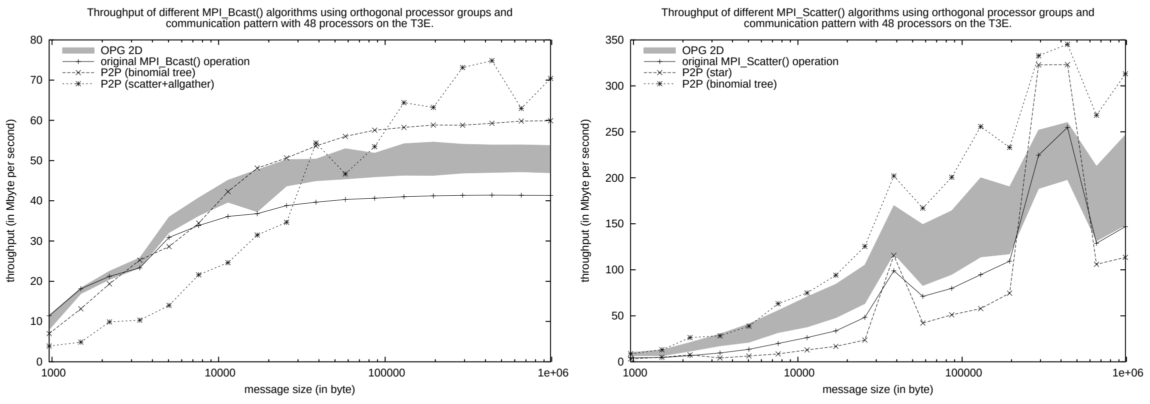


Figure 3. Throughput of different `MPI_Bcast()` (left) and `MPI_Scatter()` communication methods (right) with 48 processors on the Cray T3E-1200.

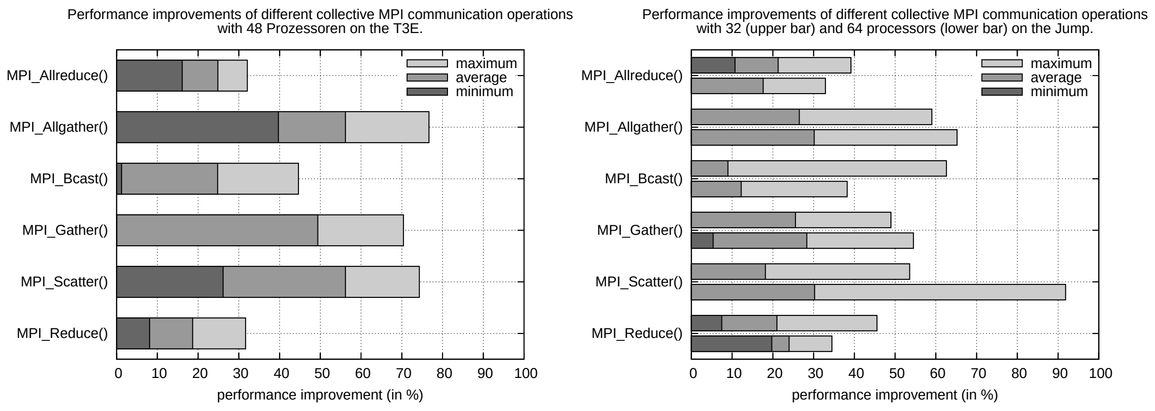


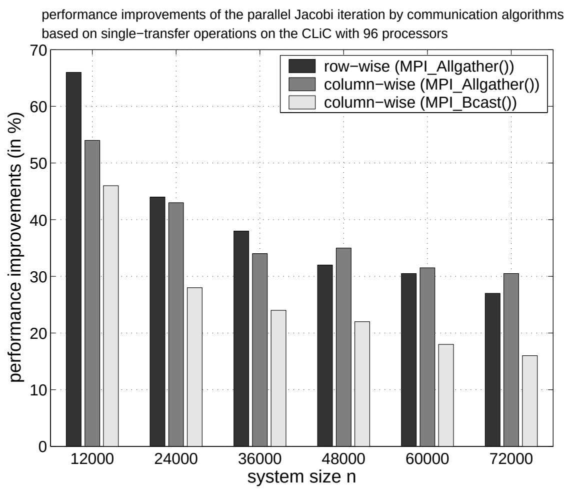
Figure 4. Summary of the performance improvements achieved by optimized communication methods for collective MPI communication operations. The figures show the minimum, average and maximum improvements over all message sizes obtained on the Cray T3E-1200 (left) for a total number of 48 processors and on the IBM Regatta cluster (right) for a total number of 32 processors (upper bar) and 64 processors (lower bar).

5. Runtime Tests of Parallel Program Applications

We consider the optimized communication methods in the context of complex program applications in order to verify the performance improvements achieved in isolation. For this purpose we apply the adaptive approach to reduce the communication overhead of a Jacobi iteration and a solution method of ordinary differential equations on the target platforms described in Section 4.1. Section 5.1 presents the runtimes of different program implementations of the Jacobi iteration and Section 5.2 considers the parallel Adams methods PAB and PABM.

5.1. Parallel Jacobi Iteration

We consider three different ways to implement the Jacobi iteration in a data parallel way based on a row-wise and a column-wise distribution of the matrix A. For both distributions the computational work for computing the new entries of the next iteration vector $x^{(k)}$ is the same and is equally allocated to the processors. For systems of size n each processor performs $\lceil \frac{n}{p} \rceil \times n$ multiplications



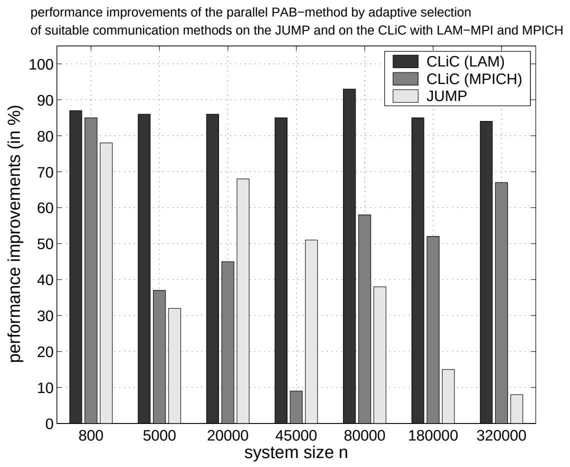


Figure 5. The left diagram shows runtime improvements for different implementations of the parallel Jacobi iteration achieved by optimized communication methods for different system sizes on the CLiC (LAM-MPI) for 96 processors. The right figure shows performance improvements of the parallel PAB Method achieved by an adaptive selection of suitable communication methods of the `MPI_Allgatherv()` operation on the CLiC (LAM and MPICH) for 96 processors. Additional performance improvements are depicted for identical system sizes on the JUMP for 64 processors.

and about the same number of additions in each iteration. But because each processor computes different parts and each processor needs the entire new iteration vector $x^{(k)}$ in the next iteration step, different communication operations are required for the implementations.

The row-wise realization uses a `MPI_Allgather()` operation to distribute the intermediate result of vector $x^{(k)}$ to all processors participating in the computation. For the column-wise realization, either a `MPI_Allreduce()` and `MPI_Allgather()` operation or a `MPI_Reduce()` and `MPI_Bcast()` operation can be used to distribute the intermediate result of vector x. Figure 5 (left) shows the performance improvements of different implementations of the parallel Jacobi iteration using the adaptive approach to select a suitable communication method on the CLiC (LAM-MPI) for 96 processors. In most cases, the adaptive approach selects point-to-point algorithms for performing the collective communication operations, since these algorithms lead the best performance improvements. On the target platforms considered, the point-to-point algorithms are slightly faster than the orthogonal realizations for the most system sizes.

5.2. Parallel Adams-Bashford Methods

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 [4]. The name was chosen due to a similarity of the stage equations with classical Adams formulas. 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 $n \cdot K$.

A `MPI_Allgatherv()` operation is used to distribute the intermediate result. Figure 5 (right) shows the average performance improvements that are obtained by an adaptive selection of communication methods to perform the `MPI_Allgatherv()` operation on the CLiC and the JUMP.

On the CLiC the runtime results are investigated for both portable MPI implementations LAM and MPICH for a total number of 96 processors and different system sizes. Using LAM-MPI a performance gain of up to 90% can be observed. Based on the MPICH implementation, also a significant reduction of the communication overhead is obtained using optimized communication methods. Furthermore, the good performance results could be confirmed also for a propietary MPI implementation on the JUMP cluster for 64 processors, see also Figure 5 (right). Again, the point-to-point realizations are selected by the adaptive approach, since these are about 10% faster than the orthogonal realizations for the system sizes considered.

6. Conclusions

In this paper we have considered an adaptive approach to select suitable orthogonal group layouts and point-to-point communication algorithms with the objective of reducing the execution time of collective MPI communication operations. An adaptive selection is crucial in reducing the communication overhead, since the resulting performance improvements depend on numerous factors. The resulting overhead in the evaluation phase of the adaptive approach is small compared to the possible performance gain. Especially for implementations of collective operations of portable MPI libraries, like LAM or MPICH, significant improvements of the communication time are obtained, since these operations are not optimized for the underlying hardware architecture of the target platforms. But also for optimized collective communication operations of proprietary MPI libraries significant performance improvements are achieved by an adaptive approach. The experimental results show that the adaptive selection of different communication methods depending on the message size leads to average performance improvements for most of the collective MPI operations, since at least one of the optimized communication methods is almost always faster than the vendor implementation.

References

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

[2] A. Hoisie, O. Lubeck, and H. Wasserman. Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications. *International Journal of High Performance Computing Applications*, 14(4):330-346, Sage Publications, 2000.

[3] M. Kühnemann, T. Rauber, and G. Rünger. Optimizing MPI Collective Communication by Orthogonal Structures. In *Cluster Computing - The Journ. of Networks, Software Tools and Applications, Special Issue on Cluster Computing in Science and Engineering*, 8(4), Kluwer, 2005.

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

[5] E.W. Chan, M.F. Heimlich, A. Purkayastha, R.A. van de Geijn. On Optimizing Collective Communication. In *Proc. of Int. Conference on Cluster Computing*, IEEE, Sep. 2004.

[6] G.D. Benson, Cho-Wai Chu, Q. Huang, S.G. Caglar. A comparison of MPICH Allgather Algorithms on Switched Networks. In *Lecture Notes in Computer Science 2840*, Springer, Sep. 2003.

[7] S.S. Vadhiyar, G.E. Fagg, J. Dongarra. Automatically tuned collective communications. In *Proc. of SC99: High Performance Networking and Computing*, ACM/IEEE, Nov. 1999.

[8] S.S. Vadhiyar, G.E. Fagg, J. Dongarra. Performance Modeling of Self Adapting Collective Communications for MPI. *Los Alamos Computer Science Institute Symposium*, 2001.