Performance Modeling of Two-phase Realizations of Communication Operations

MATTHIAS KÜHNEMANN
Technische Universität Chemnitz
kumat@informatik.tu–chemnitz.de

THOMAS RAUBER
Universität Bayreuth
rauber@uni–bayreuth.de

GUDULA RÜNGER
Technische Universität Chemnitz
ruenger@informatik.tu–chemnitz.de

Abstract

Performance analysis and modeling of collective communication operations plays an important role in the performance modeling of message passing programs. For standard collective MPI operations the performance modeling and predictions show good results for a large range of distributed memory machines and clusters. In this paper, we look at specific realizations of collective MPI operations on top of MPI and investigate whether such optimizations can be modeled using the given modeling of the standard MPI operations. In particular, we consider two-phase realizations of communication operations. We present runtime functions for the modeling and verify that these runtime functions can predict the execution time both for communication operations in isolation and for communication operations in the context of an application program. The performance model presented can be used for an a priori estimation of different implementation variants of large application programs.

1 Introduction

For the realization of parallel programs, there are usually many different design decisions to make, but it is often difficult to estimate the effect of each of these decisions. Design decisions include the selection of an appropriate data distribution for a distributed address space, a suitable assignment of the operation to be performed, and also task scheduling for larger application programs. To get an efficient implementation, it is usually necessary to implement the most promising decisions and to compare the resulting execution time on a specific hardware platform. This is a time-consuming process and many application programmers are reluctant to invest the effort. Moreover, it is often the case that the design decisions depend on specific characteristics of the hardware platform, and porting the application program to another platform would require using a different decision to get the best performance.

To reduce the implementation effort, performance models can be applied. Using a performance model, the application programmer can get a fast a priori estimation of the effects of different implementation decisions and can thus compare the effect of different decisions on the resulting performance on a specific hardware platform before an implementation. The programmer can also test the effects of the design decisions for different hardware platforms and can thus select the decisions such that the resulting performance is portable to a different platform. This could mean that for the case that the program is intended to run on two different platforms A and B, the design decisions are taken such that the resulting implementation is efficient for both A and B, but when considering A or B in isolation a different design decision might lead to a better performance.

There are many approaches for performance modeling, including statistical and queuing methods as well as experimental approaches like simulation, benchmarking and trace-driven experiments, see the section on related work. In this article, we consider an analytical modeling with runtime functions that are structured according to the computation and communication operations. Analytical models are suitable for data parallel programs [8] as well as for mixed task and data parallel programs [7], and they can also be used for large and complicated application programs [6].

In this paper, we consider the use of runtime functions for the modeling of MPI collective communication operations with a specific internal realization that is based on an orthogonal structuring of the processors in a two-dimensional grid. The communication operations are realized in two communication phases performed on selected subsets of the processors. The advantage of such two-phase realizations lies in the fact that, depending on the specific target platforms, a much faster execution time can be obtained compared to the usual implementation that considers the set of processors without any specific structure. This can be shown for such different platforms as the Cray T3E with a high-speed 3D torus network or a Beowulf cluster with a comparably slow interconnection network. Being able to model the execution time of two-phase communication operations, it is possible for the implementation of a parallel program to a priori select the collective communication operation with the appropriate functionality as well as the best implementation variant (orthogonal vs. non-orthogonal).

The runtime function for a specific communication operation for both the orthogonal and the non-orthogonal case depends on several hardware parameters like the number

of processors as well as on specific network parameters that are influenced by the performance characteristics of the network like latency and bandwidth. The parameter values are also affected by the specific MPI implementation. Moreover, there is a dependence on the size of the message to be transmitted. For the orthogonal implementation, there is an additional dependence on the processor grid used for the realization and the organization into phases. The runtime functions for communication operations are the basis for the runtime functions of complete programs. These are assembled according to the communication structure of the application and also contain subfunctions to model the execution time of the computations performed. The subfunctions depend on parameters describing the performance of the node processors and their memory hierarchy which affects the time to perform memory accesses. We derive runtime functions for orthogonal implementations of MPI collective communication operation for a Cray T3E and a Beowulf cluster.

The rest of the paper is structured as follows. Section 2 describes the two-phase implementation of collective MPI communication operations. Section 3 presents the runtime functions. Section 4 considers the runtime modeling of a complete application programs. Section 5 discusses related work and Section 6 concludes.

2 Two-phase MPI implementation

For an two-phase implementation, the p processors available are arranged in a two-dimensional processor grid of size $p = p_1 \cdot p_2$, i.e., the processors are arranged as p_1 row groups with p_2 processors each and p_2 column groups with p_1 processors each. Based on the orthogonal arrangement of the processors, the collective communication operations are restructured and are performed in two phases. In each phase standard MPI operations are performed on one or several sub-groups. The combined effect of those MPI-operations performed in both phases is identical to the effect of the original MPI-operation being reimplemented. Depending on the collective communication operation to be realized, the first phase works on one row group or all row groups such that each of the row groups performs a communication operation and all row groups work concurrently to each other. Similarly, the second phase is executed on one column group or on all column groups concurrently.

Figure 1 illustrates the two-phase implementation of an MPI_Allgather() operation for 6 processors arranged as 3×2 processor grid. Each of the processors P_i contributes a block of data A_i, $i = 0, \ldots, 5$. In the first phase, an MPI_Allgather() operation is performed in each of the row groups, thus making the data block contributed by each processor available in the row group resulting in a larger data block at each processor. In the second phase, each processor participates in an MPI_Allgather() operation in its column group, thus making each of the larger data blocks available to all processors.

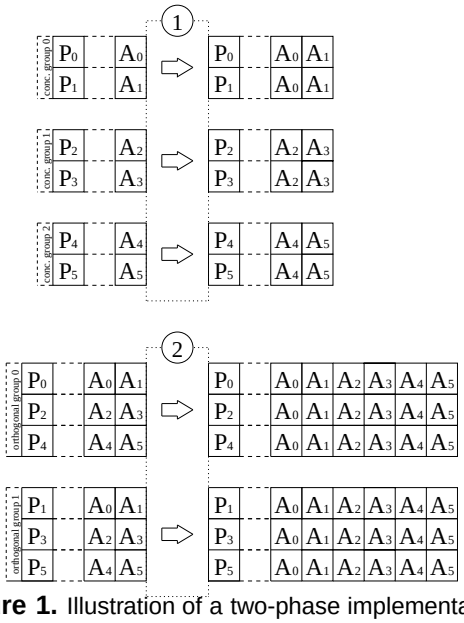

Figure 1. Illustration of a two-phase implementation of an MPI_Allgather() operation with 6 processors.

3 Performance modeling

In this section, we consider the performance modeling of the two-phase realization of collective communication operations using runtime functions. Runtime functions have been successfully used to model the execution time of communication operation for various communication libraries [4, 8]. The execution of an MPI_Bcast() operation, for example, on the CLiC (Chemnitzer Linux Cluster) using LAM MPI 6.3 b2 can be described by the runtime function

$$t_{sb}(p, b) = (0.0383 + 0.474 \cdot 10^{-6} \cdot log_2(p)) \cdot b.$$

For the performance modeling of two-phase implementations of the collective communication operations we adopt the approach to model the execution time of each phase of the orthogonal implementation in isolation. For each phase we use the runtime functions of the monolithic standard MPI communication operations from Table 1. The value b denotes the message size in bytes and p is the number

operation	runtime function
MPI_Bcast	$t_{sb\_lin}(p, b) = (\tau + t_c \cdot p) \cdot b$
	$t_{sb\_log}(p, b) = (\tau + t_c \cdot log_2(p)) \cdot b$
MPI_Gather	$t_{sc}(p, b) = \tau_1 + (\tau_2 + t_c \cdot p) \cdot b$
MPI_Scatter	$t_{ga}(p, b) = \tau_1 + (\tau_2 + t_c \cdot p) \cdot b$
MPI_Reduce	$t_{acc\_lin}(p, b) = (\tau + t_c \cdot p) \cdot b$
	$t_{acc\_log}(p, b) = (\tau + t_c \cdot \lfloor log_2(p-1) \rfloor) \cdot b$
MPI_Allreduce	$t_{macc}(p, b) = t_{acc}(p, b) + t_{sb}(p, b)$
MPI_Allgather	$t_{mb}(p, b) = t_{ga}(p, b) + t_{sb}(p, p \cdot b)$

Table 1. Runtime functions for collective communication operations on the CLiC.

coefficients for broadcast and accumulation on CLiC				
operation	formula	p	*coefficients*	
			$\tau[\mu s]$	$t_c[\mu s]$
MPI_Bcast	$t_{sb\_lin}(p,b)$	≤ 4	-0.085	0.092
	$t_{sb\_log}(p,b)$	> 4	0.038s	0.474
MPI_Reduce	$t_{acc\_lin}(p,b)$	≤ 4	-0.103	0.105
	$t_{acc\_log}(p,b)$	> 4	0.141	0.101

Table 2. Coefficients of the runtime function for (*MPI_Bcast()*) and (*MPI_Reduce()*) on the CLiC.

of processors participating in the communication operation. The coefficients τ_1 and t_c, respectively, can be considered as startup time and byte-transfer time and are determined by curve fitting with the least-squares method using measured values. For the measurements, message sizes between 10 KBytes and 500 KBytes have been used.

For an accurate modeling it is important to verify, whether concurrent group communication operations can interfere with each other with the effect that the transmission time is delayed, especially when larger message sizes are transmitted concurrently. If this is not the case, the formulas from Table 1 can be used for each of the two phases of the orthogonal implementation. In some of the formulas the startup time is very small and can be ignored. For a broadcast operation, for example, the communication time is modeled by adding the runtime function for the broadcast in the row group (using the formula from Table 1 with $p = p_1$) and the runtime function for the concurrent broadcast in the column groups (using the formula from Table 1 with $p = p_2$). The accurate predictions for the row and column groups show that this approach can be used for all collective MPI communication operations. This means that the concurrent subgroups do not influence each other. In the following, we use the notion *leader group(s)* for the row group(s), since the members of the row groups often act as leaders for the succeeding communication in the column group(s). Similarly, we use the notion *concurrent group(s)* for the column group(s), as the column groups often work in parallel. In the following, we consider the modeling of two-phase realizations of some important MPI operations.

MPI_Bcast For the single-broadcast operations, LAM-MPI uses two different internal realizations, one for $p \leq 4$ and one for $p > 4$. If up to 4 processors participate in the broadcast operation, a formula is used that depends linearly on p. For more than 4 processors a formula with a logarithmic dependence on p is used, because the broadcast transmissions are based on broadcast trees with logarithmic depth. The corresponding coefficients are given in Table 2.

Figure 2 (top) shows the deviations between measured and predicted execution times for single-broadcast on the CLiC for an two-phase realization. The figure shows the deviations for the leader group (*LG*) used in the first phase and the communication times for the concurrent groups (*CG*) used in the second phase; the bar *total* shows the total deviation of both communication phases. The figure shows minimum, maximum and average deviations between measured

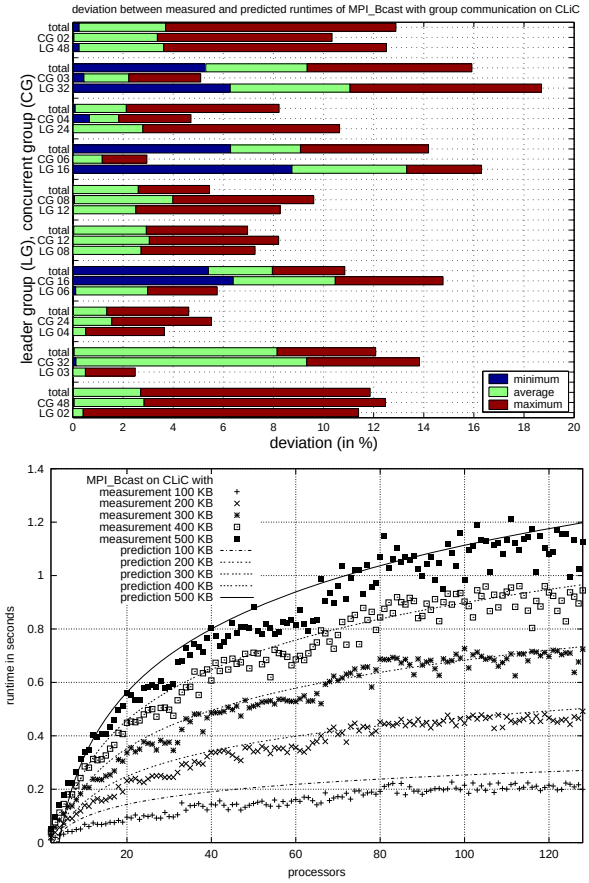

Figure 2. Deviations between measured and predicted runtimes for 96 processors (top) and modeling (bottom) of MPI_Bcast() on the CLiC.

and predicted runtime over the entire interval of message sizes. The predictions are quite accurate, but not absolutely precise for some groups, because the depth of the broadcast tree remains constant for a specified interval of processor sizes. This means that the communication time increases in stages and the runtime formulas do not model these stages. Figure 2 (bottom) shows the measured and predicted runtime of the single-broadcast operation as a function of the number of processors for fixed message sizes.

MPI_Gather The coefficients of the runtime functions for the CLiC are shown in Table 3. Using these coefficients, the predictions fit the measured runtimes quite accurately, see Figure 3 (top). The approximations are quite accurate for the entire interval of message sizes. The average devi-

coefficients for gather/scatter on CLiC			
operation	$\tau_1(V)[s]$	$\tau_2(V)[\mu s]$	$t_c(V)[\mu s]$
MPI_Gather	0.009	-0.0825	0.0929
MPI_Scatter	0.0	-0.0730	0.0897

Table 3. Coefficients for runtime function of MPI_Gather and MPI_Scatter on the CLiC.

runtime functions of gather/scatter operations on Cray T3E-1200			
MPI_Gather			
No.	p	$n[kbyte]$	runtime function
1	002 - 128	≤ 8448	$T_1(p,b) = (\tau_2 + t_c \cdot p) \cdot b$
2	017 - 032	> 8448	$T_2(p,b) = \tau_1 + (\tau_2 + t_c \cdot p) \cdot b$
3	033 - 128	> 8448	$T_3(p,b) = \tau_1 + (\tau_2 + t_c \cdot p) \cdot b$
MPI_Scatter			$T(p,b) = (\tau_2 + t_c \cdot p) \cdot b$

Table 4. Runtime functions of MPI_Gather and MPI_Scatter for a Cray T3E.

ations between measured and predicted runtimes lie clearly below 3 % in most cases.

On the T3E, the runtimes of MPI_Gather operations increase more than linearly with the number p of processors. This effect might be caused by the fact that the root processor is a bottleneck when gathering larger messages. To capture the sharp increases of the runtimes we use different runtime functions for different message sizes. Each increase can be captured by a specific formula, see Table 4. The use of a specific formula depends on the root message size, which is the size of the message that the root processor is gathering from all members of the processor group. Above 8448 KBytes a different runtime formula is used. Note that for the first formula no startup-time is necessary. The values of the coefficients are shown in Table 5.

Figure 3 (bottom) shows the deviations between measured and predicted runtimes on the T3E. The approximations are quite accurate for the entire interval of message sizes. The average deviations of 16 different processor groups are clearly below 3 %.

MPI_Scatter The predictions for the Scatter operations use the formula from Table 4. The values of the coefficients are shown in Table 5. The predictions fit the measured runtimes very accurately on both systems. The average deviations of most processor groups are below 2 % over the entire interval of all message sizes on the CLiC, see also Figure 4 (top). For instance the deviations of the leader group lies below 1 % for nearly all group sizes. The deviations between measurements and predictions of the MPI_Scatter operation on the Cray T3E-1200 are shown in Figure 4 (bottom).

MPI_Reduce The modeling of MPI_Reduce() operations is performed with the formula from Table 1. LAM-MPI uses a different internal realization for $p \leq 4$ than for $p > 4$. The specific values for the coefficients are shown in Table 2 for both cases. The communication time of a reduce operation increases in stages, because the number of time steps to accumulate an array depends on the depth of the reduce tree;

operation	No.	$\tau_1(V)[s]$	$\tau_2(V)[\mu s]$	$t_c(V)[\mu s]$
MPI_Gather	1	-	-0.00134	0.00308
	2	-0.020	0.0157	0.00505
	3	-0.036	0.0265	0.00617
MPI_Scatter	-	-	0.0002453	0.00297

Table 5. Coefficients for runtime function of MPI_Gather and MPI_Scatter for a Cray T3E-1200.

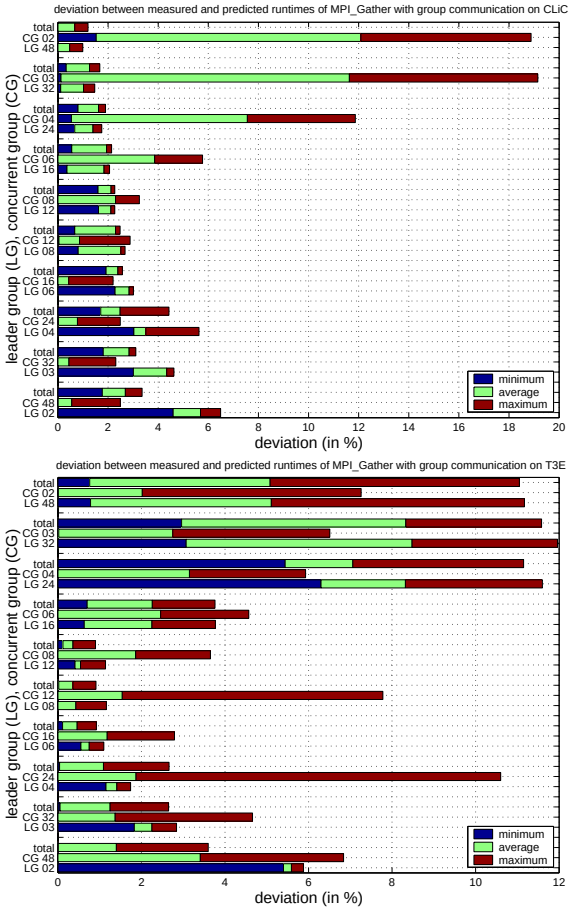

Figure 3. Deviations between measured and predicted runtimes for MPI_Gather on the CLiC (top) and Cray T3E-1200 (bottom) for 96 processors.

a detailed analysis shows that the number of processors can be partitioned into intervals such that for all processor numbers within an interval, reduce trees with the same depth are used. The predictions are very accurate and the average deviations between measured and predicted runtimes lies clearly below 3 % for most cases.

MPI_Allreduce In LAM-MPI the MPI_Allreduce() operation is composed of an MPI_Reduce()- and an MPI_Bcast() operation. First the root processor reduces the block of data from all members of the processor group und broadcasts the reduced array to all processors participating in the communication operation. The size of the array is constant in both phases. Figure 5 shows measured and predicted runtimes with fixed message sizes.

MPI_Allgather In LAM-MPI the MPI_Allgather() operation is composed of an MPI_Gather()- and an MPI_Bcast() operation. At first the root processor gathers a block of data from each member of the processor group und broadcasts the entire message to all processors participating in the communication operation. The entire message has size $p \cdot b$, when b denotes the original message size and p the number of involved processors. Figure 5 shows measured and pre-

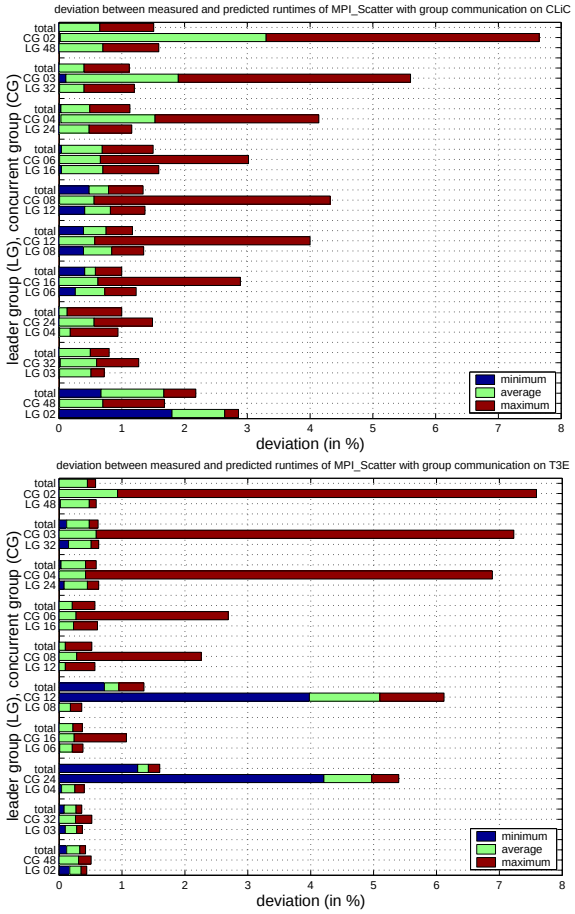

Figure 4. Deviations between measured and predicted runtimes for MPI_Scatter on the CLiC (top) and Cray T3E-1200 (bottom) for 96 processors.

dicted runtimes with fixed message sizes. The predictions fit the measured runtimes quite accurately. The deviations between measured and predicted runtimes lies below 5 % in most cases.

4 Application and runtime tests

In this section, we consider parallel implementations with two-phase communication of the well-known Jacobi iteration. In each iteration every processor performs $\lceil \frac{n}{p} \rceil \times n$ multiplications and about the same number of additions. In the row-wise distribution each processor computes $\lceil \frac{n}{p} \rceil$ scalar products yielding $\lceil \frac{n}{p} \rceil$ components of the new iteration vector. To provide the entire vector to each processor for the next step a multi-broadcast operation is performed. The execution time of the row-wise Jacobi iteration with p processors and a matrix size n (system size) can be modeled by the formula

$$T_{row}(p,n) = 2 \cdot \frac{n^2}{p} \cdot t_{op} + t_{mb}(p, \frac{n}{p}) \qquad (1)$$

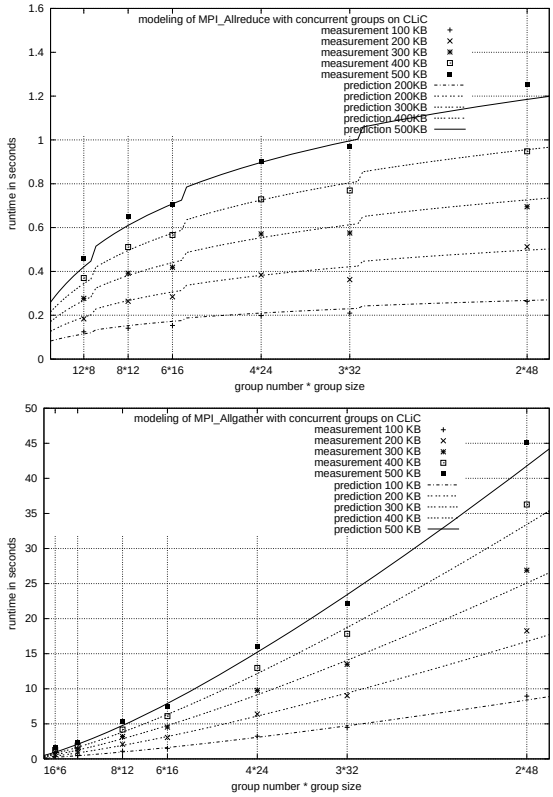

Figure 5. Measured and predicted runtimes for concurrent groups of MPI_Allreduce() (top) and MPI_Allgather() (bottom) on the CLiC.

where t_{op} denotes the time for the execution of an arithmetic operation. In the column-wise distribution each processor computes a new vector d of size n. Adding up all those vectors requires a single-accumulation operation (*MPI_Reduce*) and a single-broadcast operation results in the runtime

$$T_{col}(p,n) = 2 \cdot \frac{n}{p} \cdot (n-1) \cdot t_{op} + t_{acc}(p,n) + t_{sb}(p,n). \quad (2)$$

Figure 6 shows the performance measurements and predictions obtained by a 2D orthogonal processor structure for the CLiC. The figure shows that the predictions fit the measurements quite good.

5 Related Work

Runtime predictions using parameterized formulas for communication operation were proposed by [5] and, independently, by [4, 11] for phase-structured programs. [5] derives formulas for the Thinking Machines CM-2 whereas [4, 11] mainly consider the IBM SP-2 using the native MPL (message-passing library) runtime system. These approaches only consider the performance prediction of the communication operations in isolation, not in the context of application programs.

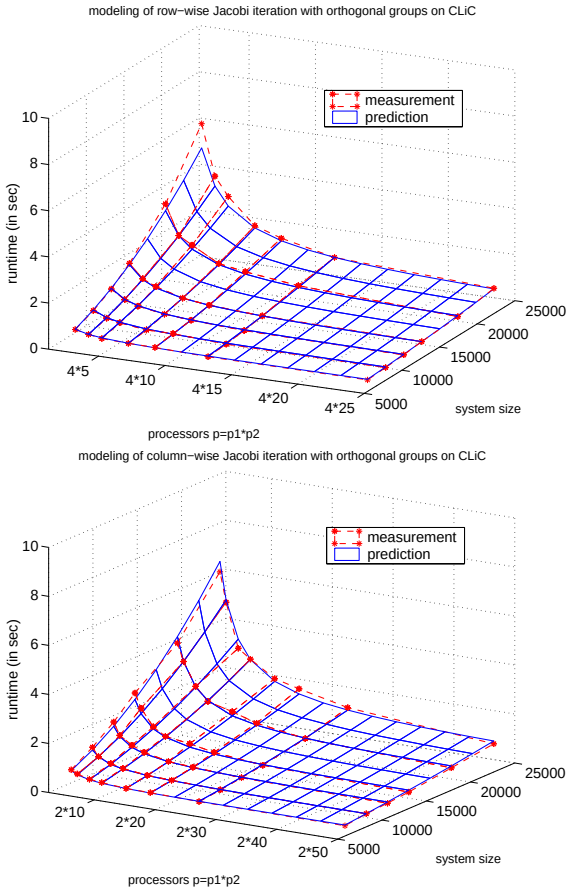

Figure 6. Modeling of row-wise (top) and column-wise (bottom) two-phase realization of Jacobi iteration on CLiC.

A predictive analytical model based on runtime functions that encompasses the performance and scaling characteristics of important ASCI applications is described in [6]. A multi-dimensional hydrodynamics code with adaptive mesh refinement is used as example. In contrast to the focus of this paper, non-orthogonal communication operations are used and the application is mainly data parallel. A predictive performance model of another ASCI application is described in [3]. A performance study for a weather prediction code has been presented in [1]. The paper is not so much concerned with predicting the performance, but with tuning the application for different platforms.

Besides runtime functions, there are many other approaches to model the execution time of communication operations or application programs. These include queuing methods [10], methods based on machine signatures and application profiles [9], or experimental evaluations. Moreover, several tools have been developed to support the programmer to develop efficient parallel programs. An example is the P^3T tool that has been implemented for predicting the performance of data parallel programs [2].

6 Conclusion

Orthogonal realizations of collective communication operations can significantly reduce the execution time, but the effects in specific situations are sometimes difficult to capture. In this paper, we have shown that the execution time of two-phase realization can be modeled by runtime functions. This gives the programmer a predictive tool to work with two-phase realizations also in the context of large application programs. Using the runtime functions, the application programmer can get an a priori estimation of the execution time of the two-phase realization and can thus develop the design of an efficient parallel implementation without extensive runtime experiments.

References

[1] J. Drake, S. Hammond, R. James, and P. Worley. Performance Tuning and Evaluation of a Parallel Community Climate Model. In *Proc. of IEEE/ACM SC99*, 1999.

[2] T. Fahringer. Automatic Estimation of Communication Costs for Data Parallel Programs. *Journal of Parallel and Distributed Computing*, 39(1):46–65, 1996.

[3] 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), 2000.

[4] K. Hwang, Z. Xu, and M. Arakawa. Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):522–536, 1996.

[5] S. Johnsson. Performance Modeling of Distributed Memory Architecture. *Journal of Parallel and Distributed Computing*, 12:300–312, 1991.

[6] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings. Predictive Performance and Scalability Modeling of a Large-Scale Application. In *Proc. of IEEE/ACM SC2001*, 2001.

[7] M. Kühnemann, T. Rauber, and G. Rünger. Performance Modelling for Task-Parallel Programs. In *Communication Networks and Distributed Systems Modeling and Simulation (CNDS'02)*, pages 148–154, 2002.

[8] T. Rauber and G. Rünger. PVM and MPI Communication Operations on the IBM SP2: Modeling and Comparison. In *Proc. 11th Symp. on High Performance Computing Systems (HPCS'97)*, 1997.

[9] A. Snavely, N. Wolter, and L. Carrington. Modeling Application Performance by Convolving Machine Signatures with Application Profiles. In *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[10] H. Takagi. *Queueing Analysis: A Foundation of Performance Evaluation.* North Holland, 1991.

[11] Z. Xu and K. Hwang. Early Prediction of MPP Performance: SP2, T3D and Paragon Experiences. *Parallel Computing*, 22:917–942, 1996.