PAPER

Evaluation of Performance Prediction Method for Master/Slave Parallel Programs

Yasuharu MIZUTANI[†a)], *Student Member*, Fumihiko INO[†], *and* Kenichi HAGIHARA[†], *Members*

SUMMARY This paper describes the design and implementation of a testbed for predicting master/slave (M/S) programs written using Message Passing Interface (MPI) programs. The testbed, named M/S Emulator (MSE), aims at assisting developers in evaluating the performance of M/S programs and dynamic load-balancing strategies on clusters of PCs. In order to realize this, MSE predicts the communication time by using a realistic parallel computational model, an extension of the LogGPS model. This extended model improves the prediction accuracy on a large number of processors, because it captures the master's bottleneck: the overhead required for retrieving arrival messages from the slaves. Current MSE also employs a best effort emulation method for predicting the calculation time. In our experiments, MSE demonstrated an accurate prediction on clusters, especially on a larger number of nodes. Therefore, we believe that our extended model enables us to analyze the scalability of the M/S program performance.

key words: *performance prediction, master/slave paradigm, load balancing, message passing, parallel computational model*

1. Introduction

With the rapid advances in cluster and grid computing [1], [2], high performance computing systems are increasing their heterogeneity of processors and interconnects. One adaptive programming paradigm for these heterogeneous systems is the master/slave (M/S) paradigm, where a single master manages task assignment to slaves and gathers computed results from the slaves. The M/S paradigm allows us to develop high performance programs by providing a dynamic load-balancing mechanism. For example, this paradigm can effectively parallelize some major computing strategies such as alpha-beta search, divide-and-conquer, and branch-and-bound algorithms [3]. Although M/S programs are effective on heterogeneous systems, they loose this effectiveness if the master is responsible for excessive slaves. That is, the programs significantly decrease their performance due to the resource contention at the master. Therefore, performance prediction is useful to investigate the optimal number of slaves as well as to develop sophisticated M/S paradigm.

The goal of this work is to develop a performance prediction system for the performance study of M/S programs on clusters. Our target programs include not only M/S programs where processing workload can be dynamically determined but also sophisticated M/S programs where the master can be dynamically generated, migrated, and structured in a hierarchical manner [4]–[7].

To achieve this goal, we have developed a testbed named Master/Slave Emulator (MSE) for Message Passing Interface (MPI) programs [8] by incorporating the following three design aspects: (D1) a low-overhead prediction by using a realistic parallel computational model; (D2) a performance saturation point modeling by adapting the used model; and (D3) reproduction of dynamic behavior. For design aspect (D1), we have selected the LogGPS model [9], which abstracts the communication by MPI. To analyze the performance saturation of M/S programs, we have adapted LogGPS to the M/S paradigm by determining the master's overhead required for retrieving arrival messages from the slaves. Finally, design aspect (D3) allows us to evaluate performance of novel dynamic load-balancing strategies since MSE reproduces the original behavior of programs by executing all program code. We are currently using emulation approach for (D3).

The rest of this paper is organized as follows. Section 2 presents some related work and summarizes problems to predict M/S program performance. Section 3 presents design aspects for the accurate prediction of M/S program performance while Sect. 4 gives the details of MSE based on the design. Section 5 presents some experimental results on a 64-node cluster. Finally, Sect. 6 concludes this paper.

2. Related Work

To predict parallel program performance, a number of tools have been proposed in the past.

MicroGrid [10] aims at providing a virtual grid infrastructure for the study of complex dynamic behavior of grid applications. To simulate network behavior, it uses a detailed network simulation including TCP congestion control. However, it requires an unpredictable large amount of computational effort [10] due to concentration of messages at the master. Therefore, for M/S programs, where messages gather at the master, detailed simulation approaches [10], [11] perturb the behavior for task distribution of the master, so that can drop the prediction accuracy. Thus, low-overhead approaches are suitable for M/S programs, especially on clusters with high-speed interconnects, where the master can pass messages frequently.

One low-overhead approach is to use a realistic parallel computational model such as the LogP [12] family of models [9], [13]–[15]. LogP abstracts the communication of

Manuscript received June 20, 2003.

Manuscript revised October 20, 2003.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan.

a) E-mail: mizutani@ist.osaka-u.ac.jp

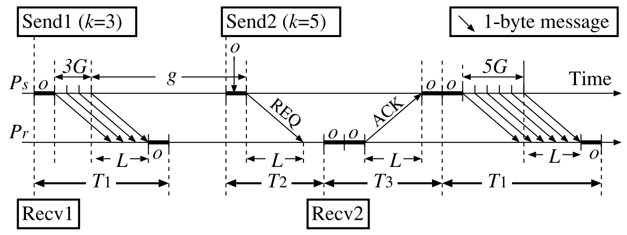


Fig. 1 An example of messages under the LogGPS model ($S = 4$).

messages by using four parameters:

- L: the latency, incurred in sending a message from its source processor to its target processor.
- o: the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message.
- g: the gap between messages, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor.
- P: the number of processors.

LogGP [13] incorporates long messages into LogP by adding the following parameter:

- G: the gap per byte for log messages, defined as the time per byte for a long message.

This model has been validated with a Gaussian elimination program [16] and a wavefront application [17] within 7% error. Furthermore, in order to capture the communication by MPI, LogGPS [9] abstracts messages in synchronous mode and in multiple packets by adding two parameters to LogGP.

- S: the threshold for message length above which synchronous messages are transmitted.
- s: the threshold for message length above which messages are sent in multiple packets.

Figure 1 shows an example of two messages, Send1 and Send2, under the LogGPS model, where $S = 4$. When $k \leq S$, P_s sends an asynchronous message (Send1) with the communication cost T_1. When $k > S$, P_s has to synchronize to P_r (Send2). In addition to cost T_1, the communication cost is defined by summing up the time to establish synchronization, cost T_2 and T_3.

On the other hand, LoPC [14] and LoGPC [15] address contention issue by adding a parameter (C), which represents the cost of contention. LoPC models processor contention [15] while LoGPC models processor and network contention.

Thus, many works have validated the accuracy of the LogP family and demonstrated accurate performance predictions for parallel programs. However, few works except CLUE [18] have validated these models with M/S programs. CLUE predicts the performance of an M/S program by executing the program and modeling communication like LogP. Although CLUE shows accurate predictions on a 5-node SMP cluster, it leaves unclear whether its prediction is accurate enough to investigate the optimal number of slaves as well as to analyze the behavior of sophisticated M/S programs.

Another approach for predicting the performance of M/S programs is to use a theoretical analysis specific to the M/S paradigm. Although this approach is helpful in determining the number of slaves, it requires strict assumptions like that all tasks assigned by the master must be of same-size [19] and that workload must be a representative distribution such as an exponential distribution [20], [21].

3. Design Aspects for Predicting Master/Slave Program Performance

This section presents design aspects for predicting M/S program performance and developing a system for the study of sophisticated M/S programs.

3.1 Low-Overhead Prediction for Accurate Performance Prediction

In M/S programs, the master's performance can determine their overall performance. Therefore, to realize an accurate prediction of M/S program performance, we have to predict the master's behavior in precise. From the discussions in Sect. 2, we have decided to use a realistic parallel computational model such as LogGPS. We have selected LogGPS because it presented an accurate prediction for data parallel programs written by MPI [9].

Note here that LogGPS captures no network contention. Although other contention-based models such as LoPC [14] and LoGPC [15] can be used, we have selected contention-less LogGPS since it has shown to be accurate on our cluster, as presented later in Sect. 5. Furthermore, LogGPS has two advantages to LoPC and LoGPC when network contention has little effect on the overall performance. First, LogGPS is simpler than LoPC and LoGPC, so that causes less perturbation on the master. Second, the LogGPS parameters depend only on the target hardware while the LoPC/LoGPC parameter, C, depends on both the target hardware and software. For example, LoPC and LoGPC require application-specific parameters such as the message injection rate and the fraction of messages determined for every pair of processors. Deriving these parameter values is complicated for some sophisticated programs [4]–[7] since such programs can dynamically generate and migrate the master.

3.2 Saturation Point Modeling for Scalability Analysis

There exist two performance issues on the scalability analysis for M/S programs.

(I1) Detecting the optimal number of masters for given processors.
(I2) Detecting the optimal number of processors for given set of tasks.

For issue (I1), we can successfully detect the optimal

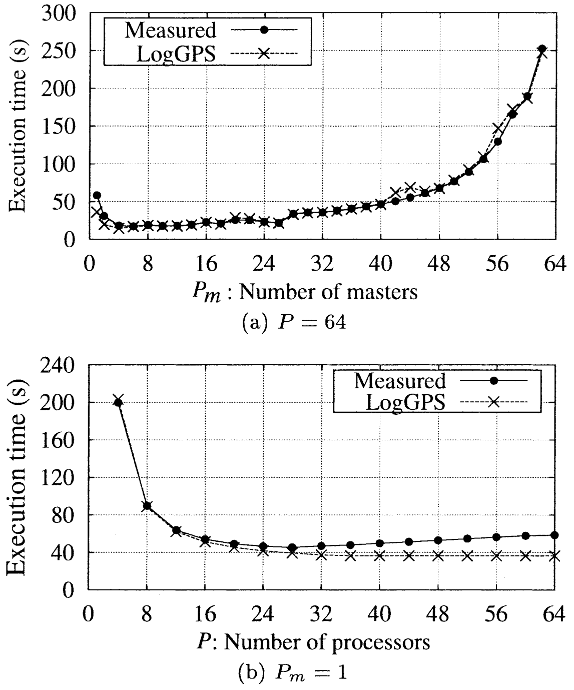


Fig. 2 Execution time for parallel mandelbrot set explorer based on M/S paradigm. P_m of P processors work as the master while remaining $P - P_m$ processors work as the slaves.

Table 1 RTT_P: Round trip time for 1-byte message on Fast Ethernet with P processors.

MPI implementation	RTT_P (μs)			σ^*: Increasing rate (%)
	$P = 2$	$P = 16$	$P = 64$	
MPICH [22]	144.0	152.1	185.8	29.0
LAM [23]	125.8	147.1	194.8	54.8
MPICH-SCore [24]	95.4	97.6	99.2	3.98

$* \sigma = (RTT_{64}/RTT_2 - 1) \cdot 100$

number of masters by using the LogP family models as shown in Fig. 2 (a). For issue (I2), we have to use the parameter values derived from the same number of processors as the target hardware. Otherwise, the LogP family fail to produce the performance curve as shown in Fig. 2 (b). This failure is a critical problem when we analyze the scalability of M/S program performance. In the following, we discuss why we have to derive it like this and how we have addressed this problem.

Table 1 shows the round trip time (RTT) measured on a Fast Ethernet network by using three MPI implementations. Note here that $P - 2$ of P processors stay idle while the remaining two processors exchange a 1-byte message as shown Fig. 3. Although two common processors transmit messages for all P, RTT increases with P in Table 1. This increase can strongly effect on the scalability analysis for M/S programs since their performance can be determined by the RTT between the master and the slaves. For example, performance prediction for 64 processors can result in 29.0% error on MPICH [22], if we use the parameter values derived from $P = 2$. Therefore, performance prediction for P processors requires the parameter values derived from the same P.

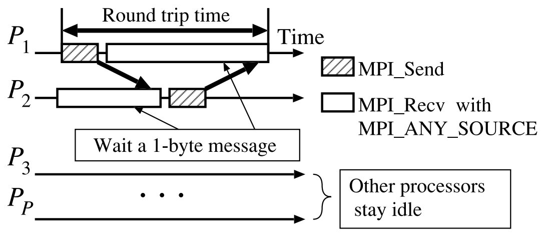


Fig. 3 Measurement of RTT.

However, smaller P is desirable for the scalability analysis. To estimate the RTT on a large P from a small P, we have adapted LogGPS by representing the overhead, $o = o' + Ok$, as a linear function of P: $o = o'_a + o'_b P + Ok$, where k is the message length; o' and O are LogGPS constants [9]; and o'_a and o'_b are additional constants derived from two values of o' measured on a pair of small Ps (described in Sect. 5).

Our linear representation for the overhead is appropriate from the following reason. The increase of RTT has no relation to network contention since the message length and the communication pattern are fixed for all P during the RTT measurement. It is due to the increase of the overhead required for retrieving arrival messages. For example, MPICH's `MPI_Recv` calls a `select` system call and Linux/FreeBSD's `select` retrieves the arrival-state sockets by linear search. Since every processor has $P - 1$ sockets, o increases with P, and thereby RTT increases with P.

3.3 Reproduction of Dynamic Behavior for Performance Study

To provide an accurate prediction for sophisticated M/S programs, we have to capture the dynamic behavior of programs influenced by their load-balancing strategies, because these programs dynamically change the behavior for good load balancing [4]–[7].

We currently use an emulation approach to capture this. That is, current MSE executes the target program without omitting any portion of program code. Although a few performance prediction systems omit some code by compiler analysis [25] for the programs whose behavior is statically determined, we avoid this omission since sophisticated M/S programs contain a code for load balancing, and distinguishing the code automatically from given programs brings hard issues. For example, such programs perform task queueing [5] as well as exchange workload information in addition to the original calculation and communication. Therefore, we have decided to use a run-time emulation approach to capture the dynamic behavior of the target programs.

The emulation approach also contributes to the prediction accuracy especially on heterogeneous systems. Since our target programs dynamically determine task distribution, static approaches [19] and post-mortem approaches [9], [13], [16] can fail to simulate the precise task assignment on the target system. This failure can lead to inaccurate prediction on heterogeneous systems, because the master can assign a heavy task to a slow slave through a low bandwidth network. Thus, the emulation approach is the simplest method for predicting the precise task assignment on the target system.

4. Master/Slave Emulator (MSE) for MPI Programs

In this section, we present the implementation of MSE. Figure 4 shows the process of performance prediction with MSE. To predict performance, MSE requires two inputs: (1) an executable binary file, generated by the compilation of the target program and the linkage with the MSE library, and (2) a configuration file for the target hardware, including the LogGPS parameters for interconnects and the relative speed parameters for processors. Thus, MSE requires no program modification.

During parallel execution, MSE emulates the execution on the target hardware. Figure 5 illustrates an example of emulation process, where processor P_s transmits a 1-byte message to P_r by `MPI_Send`. By using the cost definition in [9], P_s first estimates the completion time of the `MPI_Send`, T_s, and the arrival time of the message, T_a. After this, P_s concatenates the value of T_a to the original message, transmits them to P_r, and then keeps idle until time T_s. On the other hand, P_r splits the value of T_a from the arrival message and estimates the completion time of the matching `MPI_Recv`, T_r, by using the values of T_a and T'_r [9], where T'_r denotes the time when P_r calls the `MPI_Recv`. On times T_s and T_r, P_s and P_r return from the `MPI_Send` and the `MPI_Recv`, respectively.

As shown in Fig. 5, current MSE requires emulation overhead X, or the costs for estimating the completion time and transmitting the arrival time. Therefore, MSE requires higher speed network than target network to encapsulate X

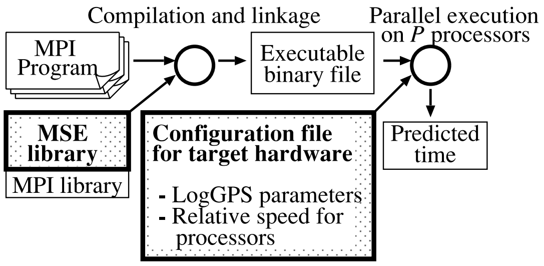


Fig. 4 Performance prediction with MSE.

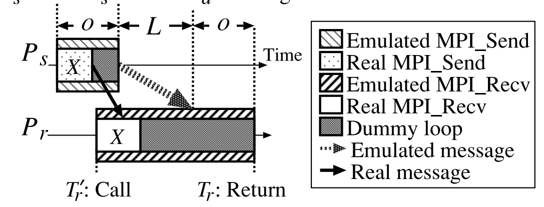


Fig. 5 1-byte message passing emulation.

in $T_s - T'_s$ for sending and $T_r - T'_r$ for receiving. Here, $T_s - T'_s$ and $T_r - T'_r$ represent the overhead of emulated `MPI_Send` and `MPI_Recv`, respectively. When X exceeds $T_s - T'_s$ or $T_r - T'_r$, the following two issues occur.

(1) Consecutive events are delayed. This delay causes difference of the timing of events occurrence between emulated and measured execution. Then, MSE must adjust predicted execution time in order to cancel the difference.
(2) This delay differs among processors, because the number of messages differs among processors due to the dynamic task assignment in M/S program. To keep the correct timing of message sending and arriving, MSE must synchronize the delay among processors.

Although the network speed restriction is a problem for performance prediction system, we think that we can resolve this problem by applying two techniques used in CLUE [18] to MSE. One is a virtual time mechanism and the other is a distributed discrete event simulation. The virtual time mechanism resolves the issue (1), because the mechanism manages predicted execution time apart from the actual time. The distributed discrete event simulation resolves the issue (2), because the simulation synchronizes virtual time among all processors.

5. Experimentation

To validate MSE on its prediction accuracy and emulation overhead, we applied it to two M/S programs: a parallel mandelbrot set explorer (MASE) for fractal visualization and a range of motion simulator (ROMS) for total hip replacement surgery [26], both solve a set of independent tasks whose workload is dynamically determined. A task of MASE corresponds to test whether a point on the complex plane is included in the mandelbrot set while that of ROMS corresponds to detect whether a three-dimensional rotation causes a collision between the femur and the artificial joints. We implemented them in three variations: single (SI), multiple (ML), and dynamic (DY) master implementations (described later in Sect. 5.3).

We used a 64-node cluster with Pentium III 1 GHz processors for experiments. Each node in the cluster connects to Myrinet [27] and Fast Ethernet switches, yielding full-duplex bandwidth of 2 Gb/s and 100 Mb/s, respectively. In the experiments, we emulated MPICH programs on Fast Ethernet by executing MPICH-SCore [24] programs on Myrinet with the same P processors.

Table 2 shows the LogGPS parameter values for MPICH on Fast Ethernet, derived from 8 processors. Here, we only show the values for asynchronous messages since MASE transmitted no synchronous message. Besides, we disregarded g since g is encapsulated in o on current machines [9], [15]. To derive o'_a and o'_b, we first derived two LogGPS constants, o' on $P = 2$ and $P = 8$, in accordance with [9], then solved a pair of equations in o'_a and o'_b variables as follows:

Table 2 LogGPS parameter values for MPICH on Fast Ethernet, derived from eight processors. P and k represents the number of processors and the length for messages, respectively.

	L (μs)	G (μs)	o for `MPI_Send` (μs)	o for `MPI_Recv` (μs)
MSE	50.0	0.0268	12.1 + 0.182P + 0.0708k	12.1 + 0.182P + 0.0722k
LogGPS			13.0 + 0.0706k	13.0 + 0.0723k

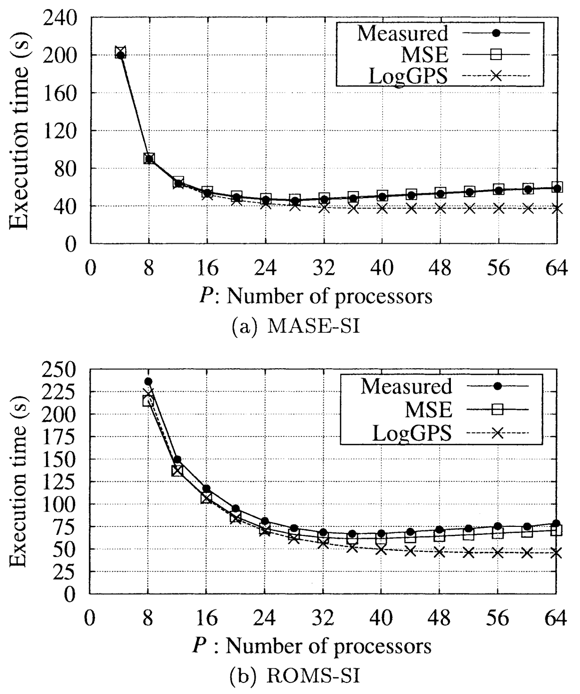


Fig. 6 Measured and predicted execution time for single master implementation (SI).

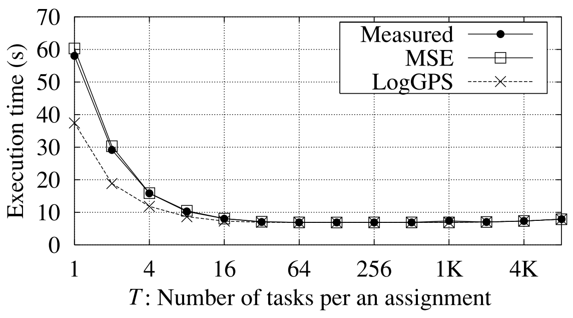


Fig. 7 Measured and predicted execution time for different task granularity (MASE-SI).

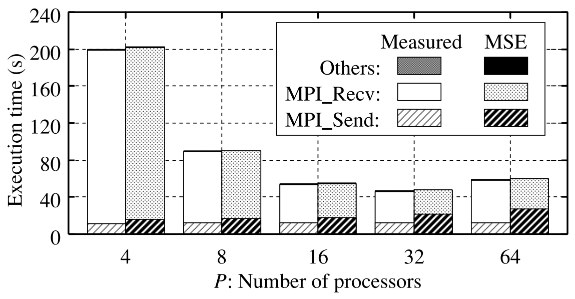


Fig. 8 Breakdown analysis of master's execution time (MASE-SI).

$$\begin{cases} o'_a + o'_b \cdot 2 &= 12.48 \quad (\mu\text{s}), \\ o'_a + o'_b \cdot 8 &= 13.57 \quad (\mu\text{s}). \end{cases}$$

As shown in Table 2, the coefficient of P has significant influence on o. For example, for $k = 1$, the value of o doubles when that of P increases from 2 to 71 processors.

5.1 Validating Prediction Accuracy

To validate the saturation point modeling of MSE, we first predicted the performance with the minimum task granularity, where the master became a performance bottleneck. Figure 6 shows the measured and the predicted execution time for the SI implementation. MSE shows an accurate prediction within 3% and 10% errors for MASE and ROMS, respectively. It also shows the performance saturation point in precise, so that the optimal P obtained by MSE agrees with the measured results: $P = 28$ and $P = 36$ for MASE and ROMS, respectively. On the other hand, LogGPS fails to show the performance curve, so that drops its accuracy as P increases. This failure can be explained as follows. Since MASE performs 1,048,576 round trip communications between the master and the slaves, the master takes at least $2o \cdot 1{,}048{,}576\,\mu$s. Substituting $P = 8$ and $P = 64$ to o (see Table 2) and subtracting them gives $1{,}048{,}576 \cdot 2 \cdot 0.182 \cdot (64 - 8)/10^6 = 21.4$ s, or the contribution of our linear overhead when $P = 64$. This value is close to 21.2 s, or the difference between the measured and the LogGPS time when $P = 64$. Thus, our linear overhead representation is necessary for the scalability analysis for M/S programs since it can significantly effect on the performance on large P.

We next analyze MASE in detail. Figure 7 shows the measured and the predicted results on $P = 64$ with different task granularity, T, where the master repeats the send and the receipt of 8 and $4T + 8$ byte messages, respectively. In Fig. 7, although LogGPS has shown to be inaccurate for $T = 1$, it becomes accurate as T increases. The above analysis on $T = 1$ explains this improvement. In MASE, the master transmits $1{,}048{,}576/T$ messages, so that the difference between MSE and LogGPS becomes $21.4/T$ s. Therefore, LogGPS raises its accuracy with the increase of T.

Figure 8 shows the breakdown of the master's execution time. We can see that MSE estimates the total time in precise for all P, however, with the increase of P, it estimates longer time for `MPI_Send` and shorter time for `MPI_Recv`. This discrepancy is due to our simplified model, which assumes that both the send and the receipt overheads have the same o'. That is, to simplify our model, we have made the assumption, but the actual send overhead contains no cost proportional to P. Our assumption works fine for

M/S programs where assigned tasks and their results are unredirected. However, for M/S programs where the master migrates before receiving the results of assigned tasks, the total number of the master's MPI_Send differs to that of its MPI_Recv, so that we have to avoid this simplification. One solution for this is to distinguish o'_a and o'_b between the send and the receipt overheads.

5.2 Evaluating Emulation Overhead

We now evaluate the emulation overhead of MSE. Figure 9 shows the account rate of the emulation overhead on the master, $R = X/(T_s - T'_s)$ for MPI_Send and $R = X/(T_r - T'_r)$ for MPI_Recv (see Fig. 5). When $R < 1$, the emulation overhead is low enough for an accurate prediction since the predicted time encapsulates the overhead. In Fig. 9, the value of R ranges from 0.20 to 0.94. Therefore, the emulation overhead is low enough to provide accurate predictions.

R of MPI_Recv (for $4T + 8$ byte messages) increases as P decreases in Fig. 9 (a) and comes close to 1.0 as T increases in Fig. 9 (b). The reason for this is the increase of the synchronization time of MPI_Recv, W, that both X and $T_r - T'_r$ contain. During this time W, the master waits for the slaves to request a task. For example, by probing the arrival messages in MPI_Recv, we obtained $W = 3.38\,\mu$s for $T = 1$ and $P = 64$ while $W = 12.9$ ms for $T = 4$K and $P = 64$ per one MPI_Recv on the average. Thus, R increases when the master waits for the slaves' requests. However, R does not go over 1.0 when the overhead of Myrinet is smaller than that of Fast Ethernet. By using the LogGPS parameters on $P = 64$, the account rate without the synchronization time, $R' = (X - W)/(T_r - T'_r - W) = (2.9 + 0.00767k)/(12.1 + 0.182 \cdot 64 + 0.0722k)$, approaches to 0.11 as k increases, where $X - W$ and $T_r - T'_r - W$ correspond to the overhead for MPI_Recv on Myrinet and Fast Ethernet, respectively.

On the other hand, although the master repeatedly sends the same 8-byte messages for all P and T, Fig 9 (b) shows an irregular behavior for MPI_Send where $512 \leq T \leq 2$K and $P = 64$. This is due to the flow control of MPICH-SCore since the increase of X causes that of R. In this case, the master receives all unexpected $4T+8$ byte messages until it successfully allocates the send buffer for the 8-byte message.

5.3 Validating Dynamic Behavior Emulation

We applied MSE to two sophisticated variations of MASE. One is the ML implementation, in which each of P_m masters has P/P_m responsible slaves, where $2 \leq P_m \leq P/2$. The other is the DY implementation, in which the number of masters can change during program execution. In DY, every master can split its responsible tasks and slaves in two groups and assign one group to a responsible slave. The assigned slave then becomes the master of the half slave and works as a secondary master until it completes the assigned tasks. The splitting and merging of the master are triggered by the statistics of the slaves' wait time from the request for a task until its assignment. To validate the emulation accuracy of the dynamic behavior of DY, we selected MASE, because the master becomes busy and tends to generate new masters compared to ROMS.

Figure 10 (a) shows the results for ML on $P = 64$. In Fig. 10 (a), although the measured time appears as an irreg-

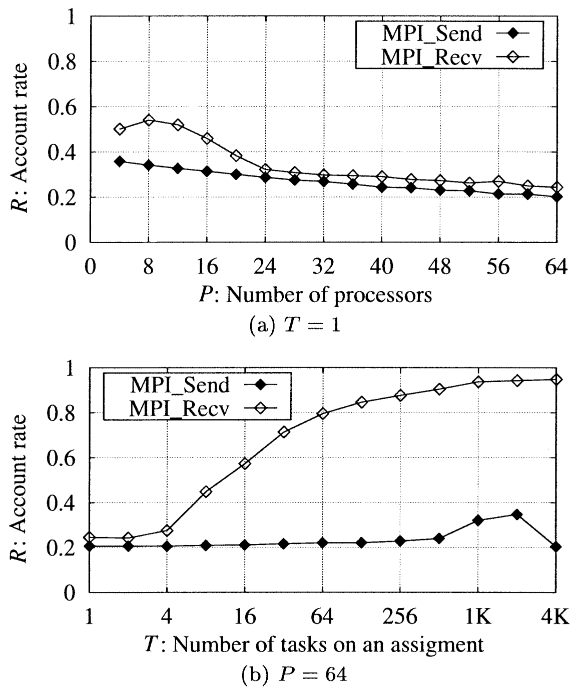


Fig. 9 Account rate of emulation overhead on the master for MASE-SI.

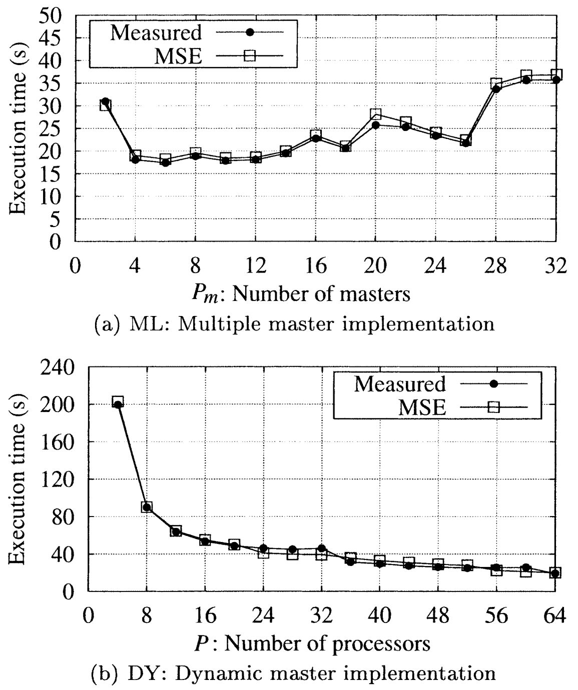


Fig. 10 Measured and predicted execution time for MASE ($T = 1$).

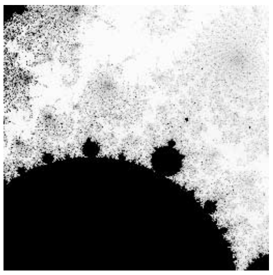


Fig. 11 Mandelbrot image expolored by MASE.

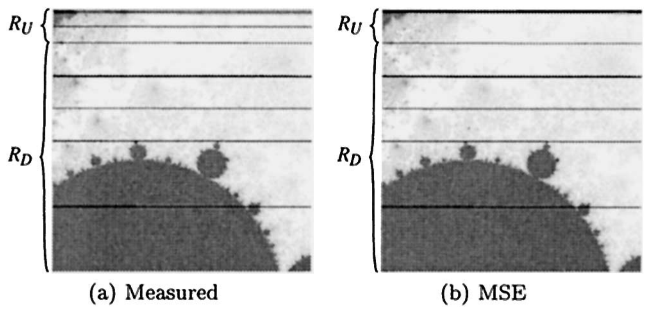


Fig. 12 Comparison of task distribution on MASE-DY.

ular graph, MSE presents an accurate prediction within 9% error. This good accuracy comes from our emulation approach since it realizes almost the same task distributions as the original. Actually, the measured time increases when $P = 16$ due to the splitting of tasks with imbalanced workloads.

Next, Fig. 10 (b) shows the results for DY, in which processors behave in a more complicated way. MSE also shows a similar performance to the measured results, so that it enables us to conclude that ML gives the best performance of below 20 s if we can detect the optimal number of P_m. Thus, MSE have demonstrated the irregular performance of M/S programs, which is not easy for static and post-mortem approaches.

Finally, we compare measured and emulated task distribution with 64 processors on MASE-DY. Figure 11 shows a mandelbrot image explored by MASE-DY, and Fig. 12 shows a task distribution map, which illustrates which master takes responsibility for which part on the mandelbrot image. In Fig. 12, a region bordered by horizontal black line corresponds to a set of tasks assigned to a master.

Since this program manages the splitting of masters according to the slaves' wait time, significantly affected by the master's communication overhead, the task distribution map is also affected by the accuracy of prediction. This accuracy is defined over the distribution map in terms of two points: (1) the horizontal position of lines, representing when a new master is splitted from which master; (2) the number of lines, representing the number of masters.

Figure 12 (a) and Fig. 12 (b) show a similar distribution in region R_D, however, fail to show it in region R_U. In region R_D, the difference of horizontal position was within a few pixels (although the size of the mandelbrot image was 1024 × 1024 pixels), and there was no difference on the number of lines. On the other hand, region R_U lacks the line due to no occurrence of the master splitting. However, this lack can be acceptable because this splitting was occurred at the lowest 4-th level, and the other splitting at the higher level, which determine the performance, is reproduced as shown in R_D. Thus, the predicted time was accurate as shown in Fig. 10 (b), and thereby we think that this distribution is not exactly as the real one, but similar enough to predict the execution time.

Consequently, MSE accurately predicts the execution time of sophisticated M/S programs as well as emulates the behavior of the programs, and MSE is useful to evaluate sophisticated M/S programs.

6. Conclusion

In this paper, we have shown the following three contributions (C1), (C2) and (C3) for accurate prediction of M/S program performance.

(C1) We presented three design aspects: (D1) low-overhead prediction by using a realistic parallel computational model, (D2) a performance saturation point modeling by adapting the used model, and (D3) the reproduction of dynamic behavior.

(C2) We demonstrated benefits of (D1) and (D2). The experimental results showed an accurate prediction within 10% error for a 64-node cluster. Therefore, our linear representation of master's overhead is important for scalability analysis of M/S program performance for investigating the optimal number of slaves.

(C3) We demonstrated the benefits of (D3) by predicting the performance of sophisticated programs. However, current MSE requires higher speed network than the target network.

Our prediction method is also applicable to other message passing programs, because our model is an extension of the LogP model.

Future work includes the development of more intelligent implementation for realizing (D3) toward predicting M/S program performance on high speed network by using slow speed network.

Acknowledgement

The authors are grateful to the anonymous reviewers for their valuable comments. This work was partly supported by JSPS Research for the Future Program JSPS-RFTF99I00903, JSPS Grant-in-Aid for Young Scientists (B)(15700030), for Scientific Research (C)(2)(14580374), and Network Development Laboratories, NEC.

References

[1] R. Buyya, ed., High Performance Cluster Computing, Prentice Hall PTR, Englewood Cliffs, NJ, 1999.

[2] I. Foster and C. Kesselman, eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, San Mateo, CA, July 1998.

[3] A. Grama, A. Gupta, G. Karpis, and V. Kumar, eds., Introduction to Parallel Computing, second ed., Addison-Wesley, Reading, MA, 2003.

[4] P. Czarnul, K. Tomko, and H. Krawczyk, "Dynamic partitioning of the divide-and-conquer scheme with migration in PVM environment," Proc. 8th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'01), pp.174–182, Sept. 2001.

[5] R.V. van Nieuwpoort, T. Kielmann, and H.E. Bal, "Efficient load balancing for wide-area divide-and-conquer applications," Proc. 8th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'01), pp.34–43, June 2001.

[6] K. Ooyama, Y. Mizutani, N. Fujimoto, and K. Hagihara, "Implementation of a parallel recursive program based on dynamic division of processor groups," Trans. IPSJ Programming, vol.43, no.SIG01, pp.107–117, Jan. 2001.

[7] K. Aida and W. Natsume, "Distributed computing with hierachical mater-worker paradigm for parallel branch and bound algorighm," Proc. 3rd IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid'03), pp.156–163, May 2003.

[8] Message Passing Interface Forum, "MPI: A message-passing interface standard," Int'l J. Supercomputer Applications and High Performance Computing, vol.8, no.3/4, pp.159–416, 1994.

[9] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: A parallel computational model for synchronization analysis," Proc. 8th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'01), pp.133–142, June 2001.

[10] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The MicroGrid: A scientific tool for modeling computational grids," Scientific Programming, vol.8, no.3, pp.127–141, 2000.

[11] J. Rexford, W. Feng, J. Dolter, and K.G. Shin, "PP-MESS-SIM: A flexible and extensible simulator for evaluating multicomputer networks," IEEE Trans. Parallel Distrib. Syst., vol.8, no.1, pp.25–40, Jan. 1997.

[12] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," Proc. 4th ACM SIGPLAN Symp. Principles Practice of Parallel Programming (PPoPP'93), pp.1–12, May 1993.

[13] A. Alexandrov, M.F. Ionescu, K.E. Schauser, and C. Scheiman, "LogGP: Incorporating long messages into the LogP model for parallel computation," J. Parallel and Distributed Computing, vol.44, no.1, pp.71–79, July 1997.

[14] M.I. Frank, A. Agarwal, and M.K. Vernon, "LoPC: Modeling contention in parallel algorithms," Proc. 6th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'97), pp.276–287, June 1997.

[15] C.A. Moritz and M.I. Frank, "LoGPC: Modeling network contention in message-passing programs," IEEE Trans. Parallel Distrib. Syst., vol.12, no.4, pp.404–415, April 2001.

[16] R. Rugina and K.E. Schauser, "Predicting the running times of parallel programs by simulation," Proc. 12th Int'l Parallel Processing Symp. (IPPS'98), pp.654–660, April 1998.

[17] V.S. Adve, R. Bagrodia, J.C. Browne, E. Deelman, A. Dube, E.N. Houstis, J.R. Rice, R. Sakellariou, D.J. Sundaram-Stukel, P.J. Teller, and M.K. Vernon, "POEMS: End-to-end performance design of large parallel adaptive computational systems," IEEE Trans. Softw. Eng., vol.26, no.11, pp.1027–1048, Nov. 2000.

[18] D.F. Kvasnicka, H. Hlavacs, and C.W. Ueberhuber, "Simulating parallel program performance with CLUE," Proc. 2001 Int'l Symp. Performance Evaluation of Computer and Telecommuication Systems (SPECTS'01), pp.140–149, July 2001.

[19] O. Beaumont, A. Legrand, and Y. Robert, "The master-slave paradigm with heterogeneous processors," Proc. 3rd IEEE Int'l Conf. Cluster Computing (CLUSTER'01), pp.419–426, Oct. 2001.

[20] C. Anglano, "Predicting parallel applications performance on non-dedicated cluster platforms," Proc. 12th ACM Int'l Conf. Supercomputing (ICS'98), pp.172–179, July 1998.

[21] A.G. Greenberg and P.E. Wright, "Design and analysis of master/slave multiprocessors," IEEE Trans. Comput., vol.40, no.8, pp.963–976, Aug. 1991.

[22] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," Parallel Computing, vol.22, no.6, pp.789–828, http://www.mcs.anl.gov/mpi/mpich/, 1996.

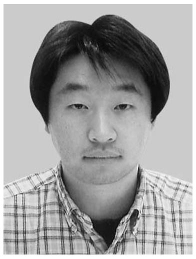
[23] G. Burns, R. Daoud, and J. Vaigl, "LAM: An open cluster environment for MPI," Proc. Supercomputing Symp. (SS'94), pp.379–386, http://www.lam-mpi.org/, June 1994.

[24] F. O'Carroll, H. Tezuka, A. Hori, and Y. Ishikawa, "The design and implementation of zero copy MPI using commodity hardware with a high performance network," Proc. 12th ACM Int'l Conf. Supercomputing (ICS'98), pp.243–250, http://www.pccluster.org/, July 1998.

[25] V.S. Adve, R. Bagrodia, E. Deelman, and R. Sakellariou, "Compiler-optimized simulation of large-scale applications on high performance architectures," J. Parallel and Distributed Computing, vol.62, no.3, pp.393–426, March 2002.

[26] Y. Kawasaki, F. Ino, Y. Mizutani, N. Fujimoto, T. Sasama, Y. Sato, S. Tamura, and K. Hagihara, "A high performance computing system for medical imaging in the remote operating room," Proc. 10th Int'l Conf. High Performance Computing (HiPC 2003), pp.162–173, Dec. 2003.

[27] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local-area network," IEEE Micro, vol.15, no.1, pp.29–36, http://www.myri.com/, Feb. 1995.



Yasuharu Mizutani received the M.E. degree in information and computer sciences from Osaka University in 2001. From 2001 to 2002, he was a Software Engineer at Mitsubishi Electric, Japan. He is currently working toward the Ph.D. degree at the Department of Computer Science, Graduate School of Information Science and Technology, Osaka University. His research interests include parallel programming language, software development tools, and performance evaluation.

Fumihiko Ino received the B.E. and M.E. degrees in information and computer sciences from Osaka University in 1998 and 2000, respectively. Since 2002, he has been an Assistant Professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include parallel and distributed systems, software development tools, and performance evaluation.

Kenichi Hagihara received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University in 1974, 1976, and 1979, respectively. From 1994 to 2002, he was a Professor in the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University. Since 2002, he has been a Professor in the Department of Computer Science, Graduate School of Information Science and Technology, Osaka University. From 1992 to 1993, he was a Visiting Researcher at the University of Maryland. His research interests include the fundamentals and practical application of parallel processing.