Efficient Runtime Algorithm Selection of Collective Communication with Topology-Based Performance Models

Takeshi Nanri[1,2] **and Motoyoshi Kurokawa**[3]
[1]Research Institute for Information Technology, Kyushu University, Fukuoka, Japan
[2]CREST, JST, Japan
[3]Advanced Center for Computing and Communication, RIKEN, Saitama, Japan

Abstract— *In communication libraries for recent supercomputers, decisions of appropriate implementations are becoming difficult by increasing size and complexity of them. Especially, because of the wide variety of the rank allocation and the improvability of the collisions, traditional static method cannot choose the appropriate technology for the given situation. As a dynamic technique for choosing implementation technologies, this paper introduces a method that selects a suitable algorithm of collective communications at runtime. At the first invocation of a collective communication, this method predicts the performance of each algorithm from the information about the rank allocation, the network topology and the routing policy. Then it discards the algorithms that are predicted much slower than others. After that, it examines each algorithm at each invocation of the collective communication to find the fastest one empirically. Results of some preliminary experiments showed the efficiency of the proposed method.*

Keywords: Collective Communication, Performance Model, Runtime Optimization

1. Introduction

In high-performance computing, communication libraries are playing an important role. To fulfill the strict requirement on performance with limited resources, they change the implementation technologies of some functions according to the situation. For example, most of the libraries prepare at least two protocols, Eager or Rendezvous, for transferring messages, and chooses one of them with a consideration of the tradeoff between the performance and the memory usage. Similar techniques are used for choosing algorithms for collective communications or deciding sizes of segments in pipelined communications. Previously, these choices of implementations have been done statically, in which they are selected according to the thresholds provided before execution. In most cases, benchmarks on some possible combinations of parameters, such as the number of ranks and the size of messages, are used to decide these thresholds [8], [9].

However, as the sizes and the complexities of computer systems for high-performance computing are increased significantly, such static strategy has become insufficient to enable efficient usage of the systems. One of the reasons is simply because the parameters to be considered has become large in number and wide in range. In addition to that, especially for communication libraries, usage of cost-efficient topologies of interconnect networks, such as Fat Tree, Mesh or Torus, has increased the difficulty on appropriate choice. On these topologies, there are some additional issues that affect the performance significantly, such as the collisions among independent messages, and the distance between the sender and the receiver. This means, even when the number of ranks and the size of the message are the same, best implementation technology can be different according to the situations only known at runtime, such as relative locations of ranks. Therefore, demands for the techniques to choose suitable implementations at runtime are increasing [1], [10].

In this paper, as one of them, a method is proposed for choosing the appropriate algorithm of collective communications at runtime. Collective communications, such as broadcasts, reductions and all-to-all exchanges, are popularly used in parallel programs of computational sciences to achieve productivity and performance portability. There have been many algorithms introduced for each of these collective communications. The method proposed in this paper is a combination of two approaches, performance prediction and performance measurement. At first, it gathers the information about the allocations of ranks, and applies them to the performance models of the available algorithms along with the information about the topology and the routing policy of the system to predict the time for completing the communication with them. By comparing the results, algorithms that are predicted to be significantly slower than others are discarded from the candidates. Then, each of the remained algorithms is examined one at each call of the collective communication to gather the empirical performance data. After all algorithms are examined, the fastest algorithm is chosen to be used for the rest of the calls. In this paper, this method is applied for Alltoall communication on a system with Fat Tree topology. The effect of the method is examined with some experiments.

2. Alltoall communication

As previously mentioned, collective communications are patterns of communications among a group of ranks to

copy, reduce or exchange data. This paper applies our method of runtime algorithm selection method to one of those communications called Alltoall, which is frequently used in Fast Fourier Transform or in matrix transposing. This communication involves every rank interchanging data among all the ranks at the end of the operation. Since each rank needs to send different data for each receiver, this is one of the most network-intensive collective communications used in parallel programs.

There have been many algorithms of Alltoall introduced to achieve better performance. For example, Simple Spread algorithm basically posts all receives and all sends, starts the communications, and waits for all communications to finish. Each rank proceeds with the communication sequentially and the order of the communications for node i is as follows: $i \rightarrow i+1$, $i \rightarrow i+2$, ..., $i \rightarrow (i+p-1) mod p$. This parallel algorithm is achieved using the non-blocking communication, such as MPI_Isend and MPI_Irecv in MPI(Message Passing Interfase). This means that the rank returns immediately after a function call. Thus, consecutive communications can overlap. However, it may cause severe collisions on a rank at which many messages have arrived at the same time.

To avoid such collisions, there are some algorithms that divide the entire communications in phases. For each phase, the targets of communications are carefully chosen so that each rank sends data to different rank. Ring algorithm, for example, uses the phase number as the distance of the target, while the Pairwise algorithm uses exclusive-or on the rank number and the phase number to decide the ranks to exchange data. These algorithms can be further classified by the method how they synchronize the phases. In one implementation, barrier synchronization is called before each phase, while there is another implementation that calls a barrier only once at the beginning of the algorithm. Another approach uses a light barrier in which senders wait for acknowledgement messages from receivers before sending data, like rendezvous protocol.

On the other hand, Bruck algorithm packs data for different receivers in one message to reduce the number of phases. At first, it rotates the data blocks on the rank i by a distance of i blocks both at the beginning and at the end of the algorithm rank. Then it sends messages in $\log(p)$ steps. In each step k, each rank sends to $(myrank + 2k)$ and receives from $(myrank - 2k)$, where $myrank$ represents the id of the rank. Since it requires redundant message transfer, this algorithm is mainly used for small message sizes.

These algorithms are only a part of the introduced ones for Alltoall communication. The relative speed of each algorithm depends not only on the traditional parameters such as the size of the message and the number of ranks, but also other factors such as the topology of the system and the location of ranks. The relative speed among algorithms changes according to not only traditional parameters such as the size of the message and the number of ranks, but also the topology of the system and the location of ranks. Therefore, static strategy cannot find the appropriate algorithm for given situation.

3. Effect of the Rank Allocation on the Performance of Collective Communication Algorithms

This section examines the effect of the rank allocation on the relative performance of the algorithms of collective communications by an experiment.

As the environment of the experiment, RICC (RIKEN Integrated Cluster of Clusters) at RIKEN is used. Each compute node of RICC consists of two processors of Intel Xeon X5570 2.93GHz and 12GB of memory. Each processor has four cores, and the processes are mapped to the cores one by one.

Compute nodes are connected by InfiniBand with Fat-Tree topology. It has two spine switches, and both are connected to all leaf switches. Figure 1 shows the topology, and the routing policy of the interconnect network of RICC. Each leaf switch is connected to each spine switch with two links, which means, there are four links in total from leaf to spine. The total number of compute nodes is 1024. 20 nodes are connected to each of the 51 leaf switches, and four nodes are connected to one. Each compute node has a unique number. As shown in the figure, contiguous numbers are attached to the nodes in a leaf switch; compute nodes on the 0th leaf switch have numbers from 0 to 19, those on the 1st leaf switch have 20 to 39, and so on. On the other hand, four links to spine switches are numbered from 0 to 3. At the transfer of a message to the node in other leaf switch, the node number of the target is divided by four, and the remainder is used as the number of the link to be used for sending the message upwards from leaf to spine, as well as downwards from spine to leaf.

On such topology, the balance of the usage ratio of the links depends on the rank allocation. Therefore, in this experiment, 128 ranks are allocated in five patterns as shown below to see the effect of the unbalanced usage on the performance of the collective communication algorithms.

Pattern 0:

> In each of 32 leaf switches, four ranks are allocated on the compute nodes with the node number of the multiple of four from the one with the smaller

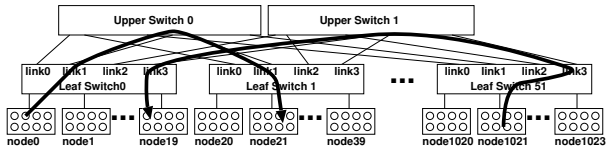

Figure 1: Topology and Routing Policy of the Network of RICC

number. That means in the 0th switch, four ranks are allocated on node 0, 4, 8 and 12, while in the 1st switch, another four ranks are allocated on node 20, 24, 28 and 32, and so on. With this pattern, at each leaf switch, four ranks share one link from, and to the spine switch.

Pattern 1:

In each of 32 leaf switches, four ranks are allocated on the compute nodes with the node number of the even number from the one with the smaller number. That means in the 0th switch, four ranks are allocated on node 0, 2, 4 and 8, while in the 1st switch, another four ranks are allocated on node 20, 22, 24 and 28, and so on. With this pattern, at each leaf switch, each two ranks share one link from, and to the spine switch. With this pattern, at each leaf switch, each two of four ranks share one link from, and to the spine switch.

Pattern 2:

In each of 32 leaf switches, four ranks are allocated on the compute nodes sequentially from the one with the smaller node number. That means in the 0th switch, four ranks are allocated on node 0, 1, 2 and 3, while in the 1st switch, another four ranks are allocated on node 20, 21, 22 and 23, and so on. With this pattern, each rank exclusively use one link from, and to the spine switch.

Pattern 3:

In each of 16 leaf switches, eight ranks are allocated on the compute nodes sequentially from the one with the smaller node number. With this pattern, at each leaf switch, each two of eight ranks share one link from, and to the spine switch.

Pattern 4:

In each of eight leaf switches, 16 ranks are allocated on the compute nodes sequentially from the one with the smaller node number. With this pattern, at each leaf switch, each four of 16 ranks share one link from, and to the spine switch.

Figure 2 and 2 show the elapsed time of each algorithm with the message size of 16KB and 1MB, respectively. Horizontal axis is the pattern number.

In the figure 2, the performance of the algorithm Bruck varies with the pattern. With pattern 0 and 4, the time with Bruck is more than two times longer than the time with the fastest algorithm. On the other hand, with pattern 2, the time with Bruck is almost the same with the fastest one. This is because of the difference of the message size transferred. When the number of ranks is P, Bruck repeats communication with message size 16KB * P/2 for $\log_2 P$ times, while other repeat 16KB message transfers for P-1 times. Therefore, change of the available bandwidth with the pattern has affected severer on Bruck than others. In addition to that, the reason for the difference of the performance

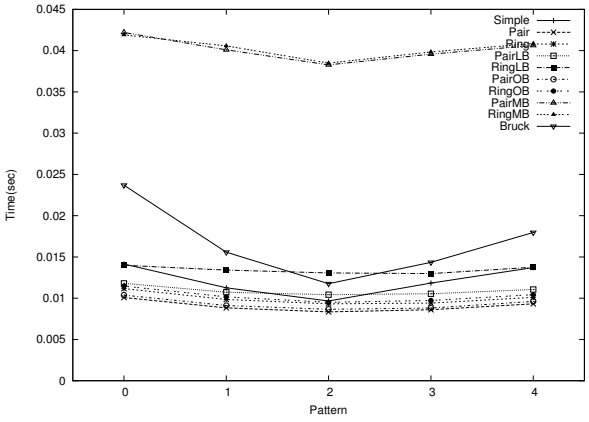

Figure 2: Elapsed time of each algorithm with each pattern(16KB)

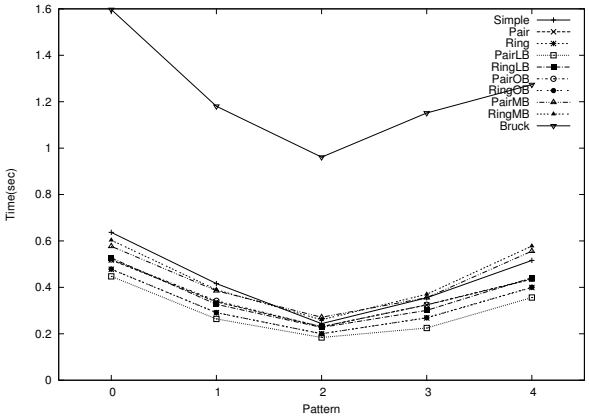

Figure 3: Elapsed time of each algorithm with each pattern(1MB)

between pattens 0 and 4, or patterns 1 and 3 is because the number of communications across leaf switches differs. In the figure 3, on the other hand, other algorithms also show the similar variation of performance with Bruck.

These results denote that the performance of algorithm changes significantly according to the relative locations of ranks. In addition to that, the effect of the rank allocation to the performance differs for each algorithm. Therefore, choice of the algorithm should be done with the consideration of the relative positions of ranks in the network topology.

4. Runtime Algorithm Selection of Collective Communication in Consideration of the Rank Allocation

4.1 Overview of the Method

The method proposed in this paper selects the algorithm according to the rank allocation as follows:

1) Before the execution of the program:
   Gather the information about the topology and the routing policy of the interconnect network of the system. Prepare performance models of each algorithm based on the information.
2) At the beggining of the execution:
   Acquire the information about the rank allocation.
3) At each call of a function of collective communication:
   If it is the first call of the function with the specified parameters, apply the rank allocation information to the performance models to estimate the time with each algorithm. Then, discard algorithms with the predicted time longer than the threshold from the candidate. This threshold is a relative value from the time of the algorithm that is predicted to be the fastest. The ratio of the time used in the experiments of this paper is 2.0.
   For each call, until all of the candidate algorithms are examined, use one algorithm for one call to complete the communication, and record the measured time.
   If sufficient number of measurements are done for each algorithm, choose the fastest one as the best algorithm, and use the best algorithm for the remaining calls.

In this method, the process for predicting performance of algorithms must be designed and implemented in consideration of the topology and the routing policy of the system. This paper introduces an example of the process of performance prediction for Alltoall communications on the Fat-Tree topology of RICC.

On the other hand, as the empirical technique for choosing the best one from the remaining algorithms, STAR-MPI [1] by Faraj, et al. is used. This method consists of two phases, Learning and Probing. In the Learning phase, one of the candidate algorithms is used for each call of the collective communication to measure the execution time with it. After repeated calls, when sufficient measurements are done on each algorithm, this phase is finished, and the fastest one is chosen to be used for the following the Probing phase. In this phase, the execution time of the chosen algorithm is measured for each call. If the time changed significantly from the previous measurement, it uses the second one.

4.2 Performance Prediction of Alltoall Algorithms on a Fat-Tree Topology

This section describes the performance prediction method for Alltoall algorithms on a Fat-Tree topology. In this method, the average bandwidth is estimated according to the topology and the allocations of ranks.

First of all, the estimation of the average bandwidth is done by considering the effect of collisions occur when each rank communicates with all the other ranks. A collision occurs when more than one communications share the same link with the same direction at the same time, The method proposed in this paper assumes that, when a collision occurs, the bandwidth of the link is shared evenly by each of the communications that caused it. Therefore, the average bandwidth of a link is estimated by dividing the base bandwidth of the link without collision, by the expected number of communications that use the link in the same direction.

In calculating this expected number on the link that connects leaf switches and spine switches, the way of calculation depends on the direction of communications. As for the upward direction, a link from a leaf switch to one of the spine switches is shared by communications that have the same value of the remainder after the division of the node number of the target by four. On the other hand, communications to the compute nodes in a leaf switch that share the same remainder of dividing the node number of the target by four share the same link from a spine switch in the downward direction. In this paper, the effects of the collisions in these two directions are assumed to be almost the same. Therefore, only the downward directions are considered to calculate the average bandwidth.

As the parameters for estimating the average bandwidth, N_{node} denotes the number of nodes, P_n is the number of ranks in a node, N_{LS} is the number of leaf switches, LS_i is the i-th leaf switch, UL_{ij} is the j-th link from LS_i to the spine switches, N_i is the number of nodes in LS_i, and NUL_{ij} is the number of nodes in LS_i with the remainder of dividing the node number by four is j. In addition to that, the base bandwidth of the communications in a node, in a leaf switch and across leaf switches is assumed to be the same value B.

The average bandwidth is estimated for each of the types of communications. For the communications between two ranks in a node, they assumed to be no collision, and the average bandwidth remains as B. On the other hand, for communications between two nodes in a leaf switch, one link from a node to the leaf switch is assumed to be shared by all ranks in the node. Therefore, the average bandwidth is calculated to be B/P_n.

For communications across leaf switches, as described above, the expected number of communications that share the same link is used to calculate the average bandwidth. For the link UL_{ij}, the expected number of ranks in LS_i that shares the link at the same time is calculated as follows:

$RCV_{ij} = 1 + (NUL_{ij} * P_n - 1) * (N_{node} - N_i) * P_n/(N_{node} * P_n - 1)$

The weighted average of these estimations of available bandwidth by using the ratio of the occurrence of each type of communication as the weight, is calculated in the following fomula.

$BRij = \frac{P_n - 1 + N_i - 1 + (N_{node} - N_i) * P_n / RCV_{ij}}{N_{node} * Pn - 1} * B$

BR_{ij} is calculated for each i, j with $0 \leq i < N_{LS}$, $0 \leq j < 4$, and the minimum value B_{min} is used as the average bandwidth of the system. This value is applied to the performance models of algorithms to predict the execution

time of collective communication with them. Table 1 shows the models of the algorithms of Alltoall used in this paper.

Table 1: Performance models of Alltoall

Algorithm	Model
Simple Spread	$(P-1)*L+(P-1)*M/B_{min}$
Ring	$(P-1)*(L+M/B_{min})$
Ring with One Barrier	$(P-1)*(L+M/B_{min})+(L*log_2P)$
Ring with MPI Barrier	$(P-1)*(L+M/B_{min})+(L*(P-1)*log_2P)$
Pair with Light Barier	$(P-1)*(L+M/B_{min})+L*(P-1)$
Pair	$(P-1)*(L+M/B_{min})$
Pair with One Barrier	$(P-1)*(L+M/B_{min})+(L*log_2P)$
Ring with MPI Barrier	$(P-1)*(L+M/B_{min})+(L*(P-1)*log_2P)$
Ring with Light Barier	$(P-1)*(L+M/B_{min})+L*(P-1)$
Bruck	$L*log_2P+P*M*log_2P/(2*B_{min})2$

These models use Hockney model as the basic model for point to point communication. In this model, time for each point to point communication is estimated as $L + M/B$ where M is the size of the message, and L and B are the latency and the bandwidth of the network respectively. This is one of the simplest algorithms. There have been some other models, such as LogP [2], LogGP [3] and PLogP (Parameterized Log-P) [4] that are proposed to achieve better correctness of the estimation. For example, PLogP represents the change of the bandwidth for different message sizes. Applying these models to the model of algorithms is remained as the future works.

In the implementation of Alltoall in this paper, L and B of the system are measured before executing the program. A benchmark program that repeats point-to-point communications is used for this measurement. In addition to that, the information about the topology, such as the number of links between leaf switch and spine switch, and the policy of routing are also collected before execution to construct the formula for calculating B_{min}.

On the other hand, the information of rank allocation is achieved at the beginning of the program. This information is applied to the formula to estimate the performance of each algorithm. From the results of this estimation, the algorithms that are predicted to be sufficiently slower than others are discarded from the candidates for the following Learning phase. The experiments in this paper used the threshold of time for discarding algorithms as the time two times longer than the fastest one. This threshold will be examined carefully with consideration of the accuracy of the prediction, in a future work.

5. Experiments

5.1 Overview

To examine the effect of the proposed method, experiments are done on a program called Alltoall Benchmark in the sample codes of STAR-MPI. This program is written in C with MPI, and repeats Alltoall communications for 200 times. The environment of the experiment is RICC. In the experiment, 10 jobs are submitted for each pair of a message size and a number of ranks.

On RICC, jobs are managed by a scheduler that is called Meta Job Scheduler. This scheduler prepares only one job pool, instead of job queues. The users specify the resource requirements at the submission of a job, such as the number of ranks, the size of memory and the estimated execution time. Then the scheduler uses these pieces of information to map the ranks of the job to the scheduling table of compute nodes. Nodes start execution of ranks according to the table. Since there is no job queue or node block, the scheduler can map ranks of jobs to the nodes without considering their locations. Therefore, with this policy, the number of idle nodes can be reduced, which can cause the throughput to be increased significantly.

This flexibleness on allocating ranks causes unstableness of the performance of communications. With this situation, existing static methods for choosing algorithms of collective communications cannot find the appropriate one. On the other hand, runtime selection techniques, such as the one proposed in this paper or STAR-MPI, are able to find suitable algorithms for given situations.

5.2 Results

Figure 4, 5 and 6 show the average times of alltoall communications with 16KB of message size on 1 rank $times$ 32 nodes, 4 ranks $times$ 32 nodes and 8 ranks $times$ 32 nodes, respectively. Horizontal axis is the job number. Each curve of the figures shows the average time with the method proposed in this paper (DYN GROUP), the method of runtime algorithm selection that measures all available algorithms (DYNþ£Őąþ£ŐąNOGROUP), and with the algorithms, Bruck (Bruck), Pairwise (Pair), Pairwise with Light-Barrier (PairLB), Pairwise with MPI-Barrier (PairMB), Pairwise with One-Barrier (PairOB), Ring (Ring), Ring with Light-Barrier (RingLB), Ring with MPI-Barrier (RingMB) and Ring with One-Barrier (RingOB).

As shown in the figures, performances of both DYN GROUP and DYN NOGROUP are close to the fastest algorith in most of the cases. These results indicates that both runtime selection methods could find the fastest algorithm. The overhead of each method is shown as the gap from the time of the fastest one. The major part of the overhead is the time for examining slow algorithms.

Figure 7 and 8 show the ratio of the time of DYN GROUP over DYN NOGROUP, for different number of ranks. In all

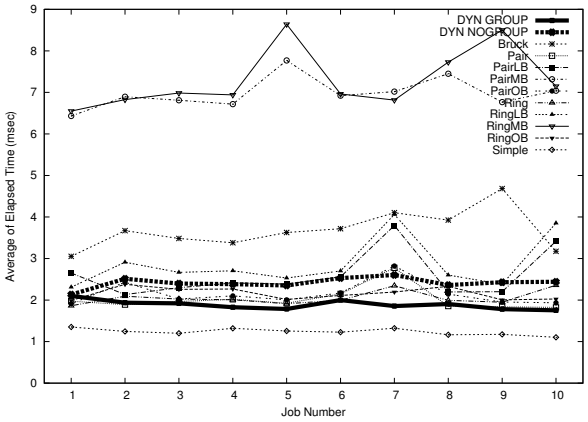

Figure 4: Average Time of Alltoall at each Job (16KB, 1 rank × 32 nodes)

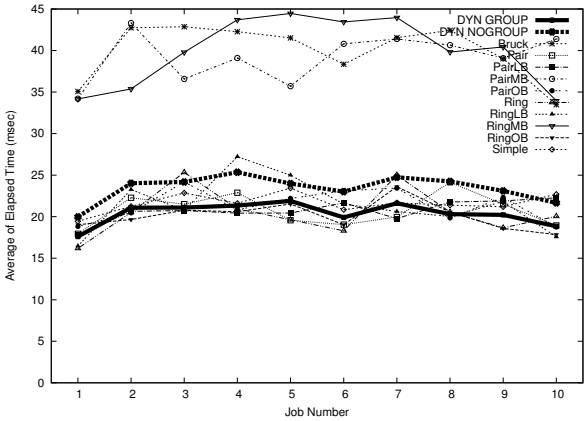

Figure 5: Average Time of Alltoall at each Job (16KB, 4 rank × 32 nodes)

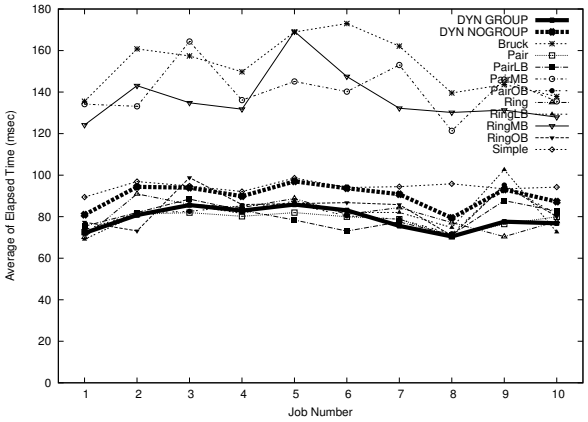

Figure 6: Average Time of Alltoall at each Job (16KB, 8 rank × 32 nodes)

the cases, DYN GROUP was faster than, or as fast as DYN NOGROUP. This is the effect of reducing the number of candidate algorithms at runtime. The reason of the significant effect on smaller message sizes is because most of the available algorithms are for middle or large sizes.

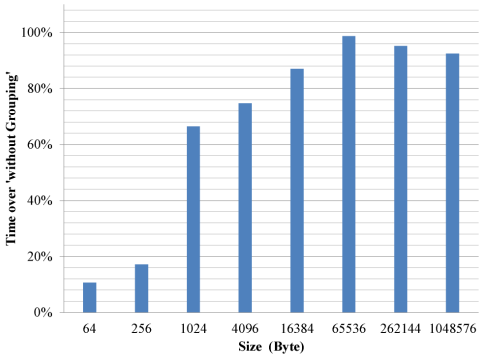

Figure 7: Ratio of the Time with and without Discarding Slow Algorithms (4 ranks × 32 nodes)

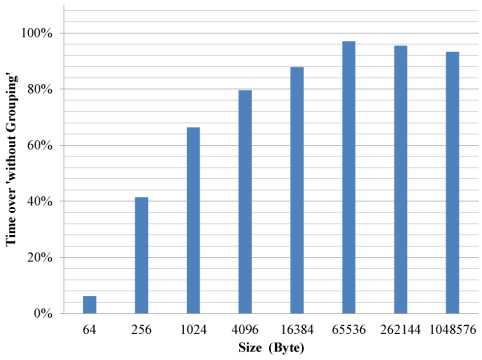

Figure 8: Ratio of the Time with and without Discarding Slow Algorithms (8 ranks × 32 nodes)

5.3 Accuracy of the performance prediction

This section examines the accuracy of the performance prediction introduced in this paper. As an example, Table 2 compares the predicted time with the average of the measured time of each algorithm in the case with 1 rank × 32 nodes.

The predicted time is less than a half of the measured time. Therefore, the prediction is not accurate enough. In the experiment, the proposed method has discarded two algorithms, Pair_mpi_barrier and Ring_mpi_barrier because they are predicted to be more than two times slower than the fastest one, Simple. However, the measured times shows that Bruck was also more than two times slower than Simple.

Therefore, the performance is expected to be gained if the prediction becomes more accurate. For example, us-

Table 2: Measured and Predicted Time of the Algorithms

Algorithm	Measured Time (ms)	Predicted Time (ms)	Remained or Discarded
Simple	1.352	0.606	remain
Pair	1.974	0.606	remain
Ring	1.865	0.606	remain
Pair_light_barrier	2.650	0.776	remain
Ring_light_barrier	2.310	0.776	remain
Pair_one_barrier	1.912	0.661	remain
Ring_one_barrier	1.989	0.661	remain
Pair_mpi_barrier	6.432	2.366	discard
Ring_mpi_barrier	6.551	2.366	discard
Bruck	3.053	1.150	remain

ing more detailed performance models, such as LogGP or Parametrized LogP, may solve this problem.

6. Related Works

Several algorithms for improving the performance of collective communications have been proposed for decades [5], [6]. More recently, some researchers have focused their efforts on taking advantage of existing algorithms in order to find techniques for selecting the most efficient algorithm for any given system/workload configuration. For example, Abdul Hamid, et al. have proposed an analysis of algorithm selection for optimizing collective communication with MPICH for Ethernet and Myrinet networks [8]. Instead of using the same thresholds for any MPICH installation on any parallel computer, they have investigated a way to find the optimum change-over points for different systems. Still their approach is a static optimization that could not be applicable on unstable situations by all means. Thus, Faraj, et al. have proposed STAR-MPI [1]. Their solution for tackling the best-performing algorithm selection problem uses dynamic profiling empirical data coupled with a static algorithm grouping method based on the performance results of different network platform systems. In addition, the authors also mentioned that the number of candidate algorithms can cause severe overhead, and performance models of those algorithms can be a clue to reduce their number. Nishtala, et al. have implemented a method of runtime algorithm selection on GASNet [10]. This method reduces the number of candidate algorithms by using performance models of them. However, since this method does not consider the effect of rank allocations, the estimation of the performance is not precise enough. In addition to that, this method measures the execution times of all candidate algorithms at the first call of the collective communication, overhead of the method is significant.

We have proposed an idea about combining both performance prediction and runtime measurement for choosing algorithms at runtime [11]. In this method, latencies and bandwidths are measured at the beginning of the program. Because the available number of measurement is limited to achieve low overhead, this method cannot consider the effect of rank allocation sufficiently.

7. Conclusions

A method for choosing an appropriate algorithm of a collective communication at runtime was proposed. This method used information about the network topology, the routing policy and the rank allocation to reduce the number of candidate algorithms examined at runtime. As an example of this method, Alltoall communication function was implemented for Fat-Tree topology of RICC. Experiments showed that the proposed method could find an appropriate algorithm with low overhead.

As the future works, functions for other collective communications and other topologies will be constructed.

Acknowledgements

Experiments in this paper are done on RICC(RIKEN Integrated Cluster of Clusters), at RIKEN, Japan.

References

[1] Faraj, A., Yuan, X. and Lowenthal, D.: STAR-MPI: Self Tuned Adaptive Routines for MPI Collective Communications, *Proceedings of International Conference on Supercomputing*, pp. 199–208 (2006).

[2] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R. and von Eiken, T.: LogP : Towards a Realistic Model of Parallel Computation, *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Pratice of Parallel Programming*, (1993).

[3] Alexandrov, A., Ionescu, M. F., Schauser, K. E. and Scheiman, C.: LogGP : Incorporating Long Messages into the LogP model - One Step Closer Towards a Realistic Model for Parallel Computation, *Proceedings of the 7th annual ACM symposium on Parallel Algorithms and Architectures*, pp. 95–105, (1995).

[4] Kielmann, T., Bal, H. and Verstoep, K.: Fast measurement of LogP parameters for message passing platforms, *International Parallel & Distributed Processing Symposium*, LNCS 1800, pp. 1176–1183, (2000).

[5] Barnet, M., Gupta, S., Payne, D., Shuler, L., van de Geijn, R. and Watts, J.: Interprocessor Collective Library, *Scalable High Performance Computing Conference*, pp. 357–364, (1994).

[6] Rabenseifner. R.: Optimization of Collective Reduction Operations, *International Converence on Computational Science*, LNCS 3036, pp. 1–9 (2004).

[7] Thakur, R., Rabenseifner, R. and Gropp, W.: Optimizing of Collective Communication Operations in MPICH, *Mathematics and Computer Science Division*, Argone National Laboratory, ANL/MCS-P1140-0304, (2004).

[8] Hamid, A. and Coddington, P.: Analysis of Algorithm Selection for Optimizing Collective Communication with MPICH for Ethernet and Myrinet Networks, *8th International Conference on Parallel and Distributed Computing*, Applications and Technologies, pp. 133–140 (2007).

[9] Sivac-Grbovi'c, J. P., Fagg, G. E., Angskun, T., Bosilca, G. and Dongarra, J.: MPI Collective Algorithm Selection and Quadtree Encoding, *In Proceedings of the 13th European PVM/MPI User's Group Meeting*, pp. 40–48 (2006).

[10] Nishtala, R.: Automatically Tuning Collective Communication for One-Sided Programming Models, PhDThesis, UC Berkeley, Berkeley (2009).

[11] Nanri, T. and Kurokawa, M.: Effect of Dynamic Algorithm Selection of Alltoall Communication on Environments with Unstable Network Speed, *International Conference on High Performance Computing and Simulation (HPCS), 2011*, pp. 693–698 (2011).