A Two-Phase Scheduling Algorithm for Efficient Collective Communications of MPICH-G2

Junghee Lee[1,2], Dongsoo Han[1,*]

[1] Information and Communications University
119 Munjiro, Yuseong-Gu, Daejeon, Korea
{lake@icu.ac.kr,dshan@icu.ac.kr}

[2] Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-Gu, Daejeon, Korea
{lake@etri.re.kr}

Abstract. In this paper, we propose a packet-level parallel data transfer and a Two-Phase Scheduling(TPS) algorithm for collective communication primitives in MPICH-G2. The algorithms are characterized by two unique features: 1) a concurrent data transfer of packets from a source node to multiple destination nodes and 2) a scheduling of enhancing the performance of collective communications by early identification of bottleneck incurring nodes. The proposed technique is implemented and the performance improvement is measured. According to the performance evaluation, the proposed method has achieved about 20% performance improvement against conventional block data transfer methods when a binomial tree is used for the communication in LAN. In TPS algorithm, the distribution of messages to bottleneck incurring nodes is delayed to minimize the affection of the node to the total performance. Using TPS algorithm on WAN, significant performance improvement has also been achieved for various data sizes and number of nodes.

1 Introduction

Grids environment provides an enormous number of storage and computing resources connected to heterogeneous wide-area networks (WANs) or local-area networks (LANs). In Grids, computing resources constituting the GRID may have various capabilities and computing powers. In order to develop a communication schedule algorithm that enables effective access to such heterogeneous resources, both network bandwidth and latency should be considered as primary design factors. When we consider that the status of network frequently changes in WAN environment, the network status change should be considered also in improving the communication performance in the WAN environment [1].

Numerous researches have been done for the efficient scheduling of communications among computing resources. Heuristic algorithms such as FEF[1], ECEF[1], TTCC[2],

* Corresponding author

and HLOT[3] were proposed for the fast construction of communication trees. However, these algorithms use only the latency factor for the construction of communication trees. Moreover, in the current MPICH-G2, a receiver node sends data of multiple packets to other nodes, only after they receive all packets from a sender node. In other words, in case of multi-chained data transmission, A→B→C, node B does not start sending data to node C, until it completes receiving the entire data from node A.

In this paper, we suggest two ideas for the frequently used collective communication primitive, MPI_Bcast of MPICH -G2, in order to overcome above two problems. First, we propose a packet-level parallel data transfer mechanism for LAN and WAN environments and a two-phase scheduling algorithm for WAN environment. In the packet-level parallel data transfer technique, each node sends individual packet to destination nodes on receiving a packet from a source node, and it simultaneously distributes the data to multiple destination nodes. The objective of this method is to improve the performance of the current data transfer method of the collective communication primitive of MPICH-G2 in Grids environment.

Second, we propose a two-phase scheduling algorithm(TPS) which uses transmission time for a tree construction metric. The algorithm improves the total performance by placing nodes, which are prone to bottleneck, to leaf nodes in a communication tree. The objective of this algorithm is to avoid a bottleneck caused by nodes with long transmission time and then to achieve performance improvement of the collective communication primitive.

In this paper, the performances of our methods are measured and the results are compared with the conventional methods. According to the performance evaluation for nodes connected to LAN, the packet-level parallel data transfer method has achieved about 20 % performance improvement against conventional data transfer methods. A binomial tree is used for the scheduling of the communication. According to the simulation of communications for nodes connected to WAN, the scheduling algorithm which uses both packet-level parallel data transfer and TPS algorithm has achieved overall performance improvement for various data sizes and number of nodes, against algorithms such as ECEF, HLOT, and flat tree.

This paper is organized as follows. In section 2 we describe the current status of MPICH-G2 and related work. Our proposed methods, packet-level parallel data transfer are explained in section 3 and our two-phase scheduling algorithm is explained in section 4, respectively. Then, we show the experimental results in section 5, and finally, we draw conclusion and describe future work in section 6.

2 Related Work

2.1 A grid-enabled MPI, MPICH-G2

Firstly, the current version of collective primitives of MPICH-G2 uses two-layered network topology-aware scheduling to reduce communication time, which just divides communication nodes into two divisions: nodes connected to LAN and nodes connected to WAN[4]. However, since it doesn't consider detailed network information

upon constructing the communication tree, it may cause a relatively long communication time in WAN environment, which is characterized as a changeable network situations and long latency. Therefore, the schedule based on accurate network information has more opportunity in improving the performance of communication of MPICH-G2. Second, MPICH-G2 entrusts the control of data transmission to TCP/IP stack of the operating system, i.e., the current primitives of MPICH-G2 send data to the buffer of TCP/IP stack and wait completion of operations. MPICH-G2 doesn't intervene in the transfer and wait until the sending is finished.

As a solution to fast data transmission of MPICH-G2, GridFTP[4] is provided. GridFTP in MPICH-G2 provides interfaces of opening multiple sockets between two endpoints, partitioning a large message into small packets, sending those packets in parallel using multiple sockets, and, lastly, re-assembling the large message. However, it is used not for collective operations but for collaborative environments. The facility provides a means of handling only two endpoints that have large blocks of data to send/receive, and it demands high-latency and high-bandwidth for efficient communications since the two endpoints transfer enormous data through multiple sockets. Moreover, some codes must be instrumented into MPI programs for the facility to be used in communicating programs. For example, user should set an attribute, assign two endpoints, and set the parameters such as the number of sockets and TCP buffer size.

2.2 Heuristic Algorithms

Many heuristic algorithms have been designed for collective operations of MPI(Message-Passing Interface): FEF(Fastest Edge First)[1], ECEF(Earliest Edge First)[1], TTCC(Two-Tree Collective Communication)[2], etc. FEF selects a node with the smallest communication cost from a root, and ECEF chooses the node with the minimum sum of communication cost and ready time of its sender[1]. TTCC transfers data with two communication trees made by ECEF algorithm[1]. HLOT is for WAN with comparatively very large latency, and after comparing its weight of edge with one of flat tree it decides if it uses a selected edge, or not. These algorithms intend to improve performance by using the schedule that considers network information. However, SPOC and FNF aren't well suited for Grid environments, and FEF and ECEF don't consider bandwidth or message size. When a parameter of algorithm is latency, the algorithms are suited for small messages. For a long message, bandwidth is also an important factor of data transmission. Moreover, these heuristic algorithms contain overheads for scheduling such as the creation cost of a tree, memory cost, managing cost for network information, etc. For example, TTCC can increase network loads since it creates two communication trees and sends along the two paths. It also assumes that TCP/IP can select one node out of two nodes, concurrently sending data. These heuristic algorithms are limited to scheduling only with network information. They don't consider the change of basic communication method.

3 Packet-level Parallel Data Transfer

In this section, we propose packet-level data transfer and one-to-many communication using network information in MPICH-G2. This method, which uses these two ways, is named packet-level parallel data transfer.

3.1 Packet-level Data Transfer

The current implementation of MPI_Bcast in MPICH-G2 is as follows: Each node doesn't start data transfer until receiving the entire data. We call this kind of data transfer blocking data transfer. To improve the performance of the current MPI_Bcast primitive, we propose a packet-level data transfer technique and it works as follows: In this technique, a node starts to send data to the next node immediately after receiving a packet unit from the source node. As a consequence, a node may send packets many times to the destination node while it receives the entire data from a source node. The technique is also used in cut-through routing[5].

Fig. 1 contrasts packet-level data transfer to blocking data transfer technique. Data is partitioned into 3 packets, and the path is simply represented with a linear tree with 4 nodes. The progress of data transfer is illustrated in Fig. 1. The packet-level data transfer technique finishes the communication within Time 4, whereas blocking method completes the communication within Time 6. In the packet-level data transfer, remarkable performance enhancement will be gained when the height of tree is high and the size of data is large. In this transfer, the size of packet should be carefully decided because prevailing of small-sized packets may cause network congestion. To prevent the network congestion, we set data size to be sent at one time to MTU(maximum transfer unit) of IP layer. Then, it is possible to reduce the completion time for the broadcast operation of MPICH-G2 without incurring any significant overhead.

	packet-level data transfer			blocking data transfer		
	Node 0	Node 1	Node 2	Node 0	Node 1	Node 2
Time 0	xyz			xyz		
Time 1		x			x	
Time 2		xy	x		xy	
Time 3		xyz	xy		xyz	
Time 4			xyz			x
Time 5						xy
Time 6						xyz

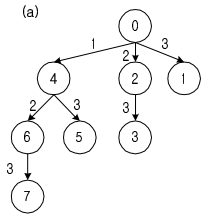
Fig. 1. Comparison packet-level data transfer with blocking data transfer

3.2 Parallel Data Transfer

If the capacity of the sender to transmit data could be sufficient enough, we can improve the performance of transmission with one-to-many data transfer. One-to-many

data transfer means that one sender can simultaneously transfer data to multiple receivers. With this method, the increase of receivers can reduce available bandwidth of sender. Therefore, the optimal number of receivers should be determined considering both bandwidth and latency.

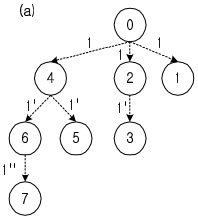
For instance, we suppose that data consist of 4 packets(w, x, y and z), each node has sufficient available bandwidth and a binomial tree with 8 nodes is used as a communication tree. The process of blocking data transfer is described in Fig. 2, and one of packet-level parallel data transfer illustrated in Fig. 3. Each node in Fig. 2 begins to transfer data only after the entire data, w, x, y and z are received from a source node. However, each one in Fig. 2 starts sending to destination nodes immediately after receiving packets, and transfers a packet simultaneously. At the Time 0, node 0 in Fig. 2 sends data only to node 4, and can forward data after Time 4, whereas, node 0 of Fig. 3 can send concurrently to node 4, 2, and 1, and node 4 can forward a packet immediately after receiving a packet from node 0. The data is completed in Time 6 as shown in Fig. 3. However, a broadcast operation in time depends on the depth and the width of the communication tree. When only the packet-level data transfer is applied, the tree with the large width will show worse performance than that with smaller width. However, when the available bandwidth is enough large, the better performance can be achieved using large width than small one of communication tree.



(b)

Time \ Node	0	1: 0->4	4->6	6->7	4->5	2: 0->2	2->3	3: 0->1
Time 0	wxyz							
Time 1		w						
Time 2		wx						
Time 3		wxy						
Time 4		wxyz						
Time 5			w			w		
Time 6			wx			wx		
Time 7			wxy			wxy		
Time 8			wxyz			wxyz		
Time 9				w	w		w	w
Time 10				wx	wx		wx	wx
Time 11				wxy	wxy		wxy	wxy
Time 12				wxyz	wxyz		wxyz	wxyz

Fig. 2. Example of blocking data transfer: (a) communication tree (b) time table



(b)

Time \ Node	0	4	6	7	5	2	3	1
Time 0	wxyz							
Time 1		w				w		w
Time 2		wx	w		w	wx	w	wx
Time 3		wxy	wx	w	wx	wxy	wx	wxy
Time 4		wxyz	wxy	wx	wxy	wxyz	wxy	wxyz
Time 5			wxyz	wxy	wxyz		wxyz	
Time 6				wxyz				

Fig. 3. Example of packet-level data transfer: (a) communication tree (b) time table

4 Proposed Algorithm

Algorithms of collective communications of MPI typically construct trees for the generation of a scheduling. However, finding an optimal tree for such an algorithm is known to be NP-hard problem. Many heuristic algorithms are proposed for collective communications of MPI. For example, flat tree is well suited for wide area networks, and binomial tree is almost optimal in local area networks [6]. Thus current implementation of MPI_Bcast in MPICH-G2 uses these two trees. Since LAN generally guarantees high speed and binomial tree works well in LAN, we focus on WAN environment in which it has relatively high latency and the status of network frequently changes. In this section, we develop a TPS tree algorithm for such WAN environment.

As noted earlier, our algorithm considers both latency and bandwidth between two nodes. We use completion time between two nodes as a target metric to be optimized. Completion time between two nodes is calculated with the following well-known equation, $completiontime_{i,j} = latency_{i,j} + \frac{messagesize}{bandwidth_{i,j}}$, and completion time of MPI_Bcast becomes the maximum time in the sums of completion time from root to every leaf node along a path. Nodes with long transmission are prone to incur bottlenecks of entire communication. If these nodes are placed in the middle of a tree, descendant nodes of these nodes will suffer from long communication delay. TPS tree algorithm identifies nodes with long transmission time, and tries to place such nodes to the leaves of a communication tree.

```
Two-Phase Scheduling Algorithm

Input:
V : set of nodes joining communication
B : set of nodes with long completion time
A : set of senders
root : root node

Output:
E := set of result edge

A := {root}
B := {}

Tree construction steps:
for i in V
      sort communication time from j to i where j in V and i <> j
B := k nodes with worst minimum time except root
//The first phase
while A <> V-B
      find j to which minimum edge from x where x in A, j in V-B-A, and x<>j
      add edge(x,j) to E
      add x to A
      x := j
//The second phase
for i in B
      find j minimizing weight of edge(j,i) + sum of weight from root to j in V-B
      add edge(j,i) to E
```

Fig. 4. Two-Phase Scheduling Algorithm

Fig. 4 shows TPS tree algorithm. Numbers on the edge of the tree denotes completion time. The algorithm sorts the values of *edge(j,i)* for individual node *i* in *V*, where node *i* has not receive a message yet and node *j* has the message. Once all the values of *edge(j,i)* are computed and sorted, node *i* can figure out which node can send the message to *i* in shortest completion time. Then, the algorithm selects number of k nodes with longest completion time. These nodes might incur bottlenecks. There are several ways to decide the number k, but we do not delve into details in this paper. Our method chooses nodes whose completion time is above the average of minimum completion time. At first, sender set A contains only a root node. Tree construction starts from the root. The algorithm finds fastest message arriving node *j* from node *x* in A. Where node *j* is neither in A nor in B, and B contains number of *k* nodes. The chosen node *j* is added to sender set A and will receive data from its sender node. After finishing the first tree construction phase except *k* nodes, the insertion of *k* nodes is conducted at the second phase. TPS attaches the rest nodes in B to the tree so that the constructed tree achieves minimum completion time from root to the nodes in B. Through above two phases approach, we can prevent nodes with long completion time being placed in the middle of the tree. That is because our algorithm is named TPS algorithm. Time complexity of TPS is $O(N^2 logN)$, where *N* is the number of nodes.

	A	B	C	D	E	F	G	H
A	0	400	800	420	500	500	800	150
B	480	0	200	450	700	540	700	370
C	700	250	0	500	870	850	700	600
D	400	450	500	0	500	950	1300	780
E	500	670	800	500	0	500	700	520
F	580	550	850	800	500	0	280	480
G	920	700	700	1260	720	300	0	200
H	300	520	600	800	550	320	200	0

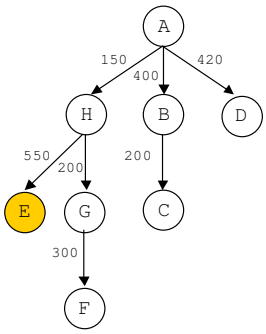


Fig. 5 Example of TPS

Fig. 5 shows an example of applying TPS algorithm to a tree. The completion time is used for the metric of tree construction. The completion time of every pair of nodes in the tree is computed and registered in a matrix. The values in red circle denote a pair with shortest completion time in each column. Then TPS finds the largest element among the values in red circle. In this example, node E has largest minimum completion time, 500. Note that, node D needs not to be considered in this case because it is already contained in the tree. In other words, node A and node D is already connected with each other when we consider node E. If we set *k* to 1, TPS constructs a tree with N- *k* nodes, i.e., 7 nodes using ECEF method. Then H, G, F, B, C, and D are picked in

sequence, and lastly, TPS decides a node where it has to attach node E. Obviously, completion time of A-H-E is a minimum among other choices, so TPS attach node E to node H. As a result, the completion time of this broadcast becomes 700 ms. We contrast our tree algorithm to other algorithms in terms of environment, metric, and time complexity in Table 1.

Table 1. Comparison of tree algorithms

Tree	Environment	Metric	Time complexity
SPOC	LAN	Message initiation cost	*O(NlogN)*
FNF	LAN	Message initiation cost	$O(N^2)$
FEF	LAN/WAN	Communication time	$O(N^2 logN)$
ECEF	LAN/WAN	Ready $time_i$ + Communication $time_{i,j}$	$O(N^2 logN)$
Look-ahead	LAN/WAN	Ready $time_i$ + Communication $time_{i,j}$ + Look-ahead $value_i$	$O(N^3)$
TTCC	LAN/WAN	Ready $time_i$ + Communication $time_{i,j}$	$O(N^2 logN)$
HLOT	LAN/WAN	$Latency_{i,j}$	$O(N^2 logN)$
TPS	WAN	Completion $time_{ij}$	$O(N^2 logN)$

Now, we analyze our algorithm via LogGP model[7]. LogGP model is suited for both short and long messages, whereas LogP model[8] is suited for a short message. LogGP model uses five parameters: latency, overhead, gap, gap per byte for long messages, and the number of processors. Since gap per byte for long messages, G, is defined as the time per byte for a long message, it can be expressed by using bandwidth, $G = 1/bandwidth$. Then completion time of sending message with length k from process i to process j can be calculated as $o_s + \frac{k-1}{bandwidth} + L_{ij} + o_r$, where o_s is overhead of receiver, and L_{ij} is latency from i to j.

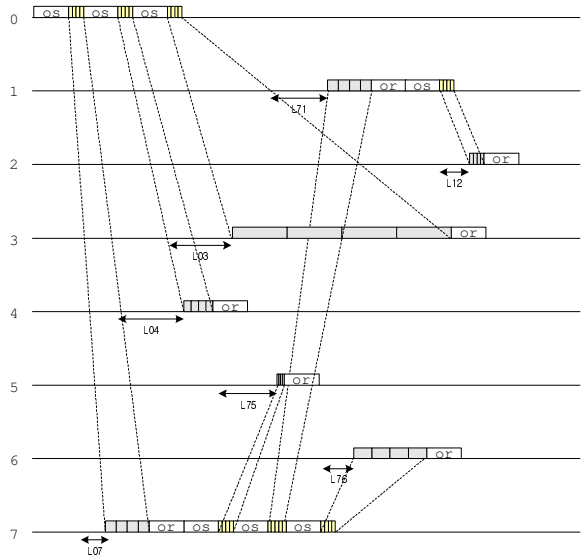


Fig. 6. Time diagram of Fig. 5

The completion time of broadcast is computed by equation, $completion\_time_B = Max\{completion\_time_l\}$, where l is the l-th leaf node of tree, and $completion\_time_l = \sum_{i=0}^{depth_l - 1} completion\_time_{st}$, where $depth_i$ denotes the depth of i-th leaf node, and $edge(s,t)$ is a part of the path from root to i-th leaf node of tree. $Completion\_time_{st}$ is computed by equation, $completion\_time_{st} = o_s + \frac{k-1}{bw_{st}} + L_{st} + o_r$, where bw_{st} is bandwidth from node s to t.

Fig. 6 depicts a timing diagram of Fig. 5 using LogGP model. The values of o_s and o_r are set with arbitrary numbers. Though the results may be changed when these values are changed, the pattern or appearances of the diagram will remain the same.

5 Evaluation

In this section, we present achieved performance enhancement on LAN through measurement, and expected performance enhancement on WAN through simulation. As explained earlier, communication performance on WAN is a dominant factor in deciding the performance of collective communications on WAN and LAN. In LAN, we use a binomial tree, which is known to be good for LAN. In WAN, TPS algorithm is used for a tree construction and we have tested the performance improvement

through simulation. In this section, experiment environment, method of time measurement, and analysis result of the test are also explained.

5.1 Performance Measurement for WAN

Simulation Scenario. We apply our proposed tree algorithm on WAN. To evaluate the performance of the proposed algorithm, we use ns-2 simulator[9], which is a widely used network simulation tool. For the comparative study, we implemented TPS, ECEF[1], HLOT[3], and flat[6] algorithms. 220 nodes of transit-stub topology were generated using GT-ITM[10]. The delay and bandwidth of the network were randomly assigned. The scale of delay spans from 10 ms to 1000 ms, and the bandwidth spans from 10 Kbps to 10 Mbps, respectively. Two types of background traffic were used: CBR and FTP. Both of the traffics have randomly selected individual starting and ending time of exchanging messages for traffic generation. Consequently, the communication among grid nodes suffers from network congestion and burst of traffics while the background traffic is active. The detailed simulation steps are as follows:

1. Specified number of Grid nodes are randomly selected
2. Delays and bandwidths of the selected grid nodes are obtained. The obtained metrics are used for the construction of a tree.
3. Construct broadcast trees for collective communication, and perform a broadcast under the same topology and traffic condition as step 1.

The simulation process was repeated 15 times and average transmission time was computed.

Simulation Results.

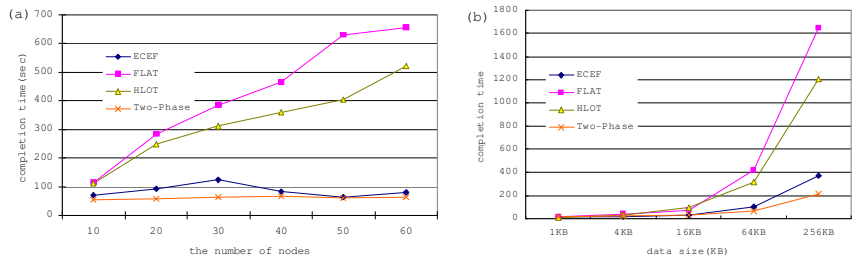


Fig. 7. Simulation results: (a) according to the number of nodes (b) according to data size

Fig. 7(a) shows the result of simulation. The completion time depending on the change of the number of selected nodes is illustrated in the graphs. The number of randomly selected nodes was varied from 10 to 60, hopping by 10 nodes. In every case, flat tree produced the worst results, and TPS was revealed to bring the best results. Fig. 7(b) shows the result of simulation depending on the change of data size. The size of data, 1 KB, 4 KB, 16 KB, and 256 KB, were used. For all data sizes, TPS outperformed

other algorithms. As the size of data is getting bigger, the performance gap is much more conspicuous.

5.2 Performance Measurement for LAN

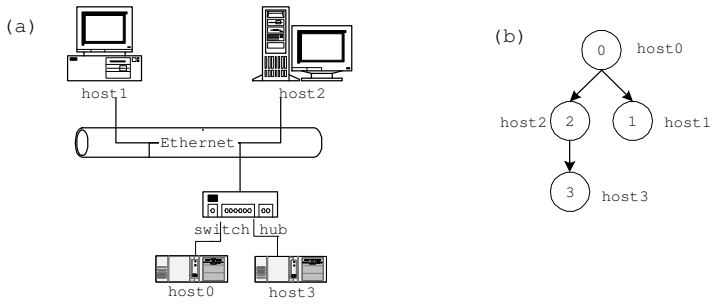


Fig. 8. (a) Test environment (b) binomial tree in LAN

Since a binomial tree is known to be well suited for LAN, and network circumstances of LAN is more stable in terms of speed and variances than WAN environment, we used a binomial tree like current MPICH-G2. Here, we examine the effect of packet-level data transfer. First of all, we evaluated the effectiveness of using packet-level data transfer with a simple socket programming. The performance comparison between conventional block transfer method and the proposed method is conducted using multiplexing I/O. Fig. 8 depicts the testing environment. Node 0, 1, 2, and 3 connected to LAN are the components of the binomial tree.

To compare the performance between conventional block data transfer and our packet-level parallel data transfer, we measured the total elapsed time for broadcast. The completion time was measured as follows. Each node sends a completion message to the root when it receives the entire data, and the end time is determined by the message arrived last at the root node. Each packet contains both a header field and a data field. The header field contains information such as header size, total data size, current data size, etc. The time attribute holds the information of end time of processing. Note that, the time required in sending an end message is negligible if the total data size is large enough. The size of a packet that can be transmitted to the network at a time is limited to MTU. Finally, to guarantee that only one packet is delivered at a time, the next packet is transmitted only after the transmission of the previous packet is completely ended. MTU was set to 1024 bytes and data size was varied from 100KB to 1000KB with 100KB intervals. The scale of time unit was *ms*. Fig. 9 shows the results of experiment using a binomial tree. In Fig. 9(a), packet-level parallel data transfer is revealed to show better performance than blocking data transfer for all levels of data size. With the involvement of 4 nodes, about 18.9% performance improvement is gained. In the Fig. 9, when the data size became large, i.e., when the

necessary packet number was increased, the performance gain was evident. Since there is additional overhead of packet-level parallel data transfer, we can conclude that the benefits of using packet-level data transfer method pay the cost of it. The extents of performance improvement is higher on LAN environment than on WAN, because the available bandwidth on LAN is more stable than on WAN. As revealed in the result of the measurement, if the data size grows, more significant performance enhancement was achieved. Furthermore, when the available bandwidth grows, the completion time is expected to be shortened.

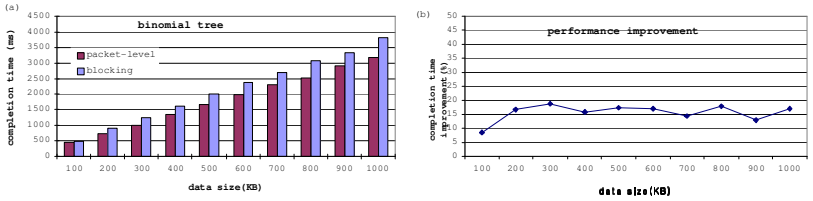


Fig. 9. Binomial tree with 4 nodes: (a) Completion time (b) ratio of completion of (a)

6. Conclusion and Future work

In this paper, we proposed a method to improve the performance of collective communication primitives in MPICH-G2, which is an interface of MPI. We devised TPS algorithm as a tree algorithm and propose a packet-level parallel data transfer for collective communication of MPICH-G2.

In TPS, we use completion time as a metric of tree, and completion time is calculated with $latency + \frac{messagesize}{bandwidth}$. That is, we consider message size and bandwidth as well as latency. TPS is object to reduce completion time and to avoid bottleneck. It first selects k nodes with the largest weight. These k nodes with great possibility of bottleneck are put in leaf nodes. There are several ways to decide k. We use an average minimum completion time of each node as a way to select k nodes. We thought that nodes with minimum completion time accessing average time have high possibility to bottleneck. Running time of proposed algorithm takes $O(N^2 logN)$. The effect of the proposed method was theoretically analyzed and experimentally showed by implementing and testing the technique. According to the test, the proposed method showed a better performance than the current conventional version of collective operations in MPICH-G2.

In a packet-level parallel data transfer method, each node sends the packet to other multiple destination nodes in receiving packets from source node. In the experiment in the real network of LAN, the packet-level parallel data transfer demonstrated superior performance to the conventional entire data transfer. And, according to the simulation results of TPS, we can confirm a performance enhancement of TPS compared to

ECEF, HLOT, and flat tree. The performance enhancement of TPS is larger as the number of nodes is increased and the size of data is enlarged.

In future, we are planning to implement the technique into MPICH-G2 collective communication primitives and test the performance. Finally, the number of k nodes in TPS should be clarified with accompanied by theoretical analysis.

References

1. P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna, "Efficient collective communication in distributed heterogeneous systems," 19th IEEE International Confer-ence on Distributed Computing Systems, 1999.
2. Kwangho Cha, Dongsoo Han, and Chansu Yu, "Two-tree collective communication", Proceedings of the IASTED International Conference on Networks, Parallel and Distrib-uted Processing and Applications, pp.30-35, Oct. 2003, Japan
3. Kyunglang Park, Hwangjik Lee, Younjoo Lee, Ohyoung Kwon, et la., "An Efficient Collective Communication Method for Gridi Scale Networks," ICCS 2003, LNCS 2660, pp. 819-828, January 2003.
4. http://www.hpclab.niu.edu/mpi
5. Peter S. Pacheco, Parallel Programming with MPI, Morgan Kautmann Publishers, Inc. 1997.
6. Thilo Kielman, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang, "MAGPIE: MPI's Collective Communications Operations for Clustered Wide Area Systems", Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 131-140, 1999
7. N. Karonis, M. Papka, J. Binns, J. Bresnahan, J. Insley, D. Jones, and J.Link, "High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment," Future Generation of Computer Systems (FGCS), Vol. 19, No. 6, pp. 909-917, August 2003
8. Albert Alexandrove, Mihai F. Lonescu, Klaus E. Schauser, Chris Scheiman, "LogGP: Incorporating Long Messages into the LogP Model- One step close towards a realistic model for parallel computation", 7th Annual ACM Symposium on Parallel Algorithms and Architectures, CA, pp. 95-105, July 1995.
9. David Culler, Richard Karp, David Patterson, Abhijit Sahay, et la., "LogP: Towards a Realistic Model of Parallel Computation", Proceedings Symposium on Principles and Practice of Parallel Programming, CA, pp. 1-12, May 1993.
10. http://www.isi.edu/nsnam/ns/
11. Ellen W. Zegura, "GT-ITM: Georgia Tech Internetwork Topology Models", http://www.cc.gatech.edu/projects/gtitm.