LETTER

A Novel Sequential Tree Algorithm Based on Scoreboard for MPI Broadcast Communication

Won-young CHUNG[†a)], ***Member***, **Jae-won PARK**[†], **Seung-Woo LEE**[††], **Won Woo RO**[†b)], ***and*** **Yong-surk LEE**[†c)], ***Nonmembers***

SUMMARY The message passing interface (MPI) broadcast communication commonly causes a severe performance bottleneck in multicore system that uses distributed memory. Thus, in this paper, we propose a novel algorithm and hardware structure for the MPI broadcast communication to reduce the bottleneck situation. The transmission order is set based on the state of each processing node that comprises the multicore system, so the novel algorithm minimizes the performance degradation caused by conflict. The proposed scoreboard MPI unit is evaluated by modeling it with SystemC and implemented using VerilogHDL. The size of the proposed scoreboard MPI unit occupies less than 1.03% of the whole chip, and it yields a highly improved performance up to 75.48% as its maximum with 16 processing nodes. Hence, with respect to low-cost design and scalability, this scoreboard MPI unit is particularly useful towards increasing overall performance of the embedded MPSoC.

key words: *MPI, broadcast communication, multicore, distributed memory, MPSoC*

1. Introduction

Message passing is the most common programming model for distributed memory systems and is often used in parallel High Performance Computing (HPC), such as cluster-based systems [1]. Although message passing is generally used for HPC clusters, there is growing interest in using message passing for embedded applications [2]. The Message Passing Interface (MPI) standard specifies the Application Programming Interface (API) for a message passing library. Presently, it is the de facto standard for message passing.

The MPI standard includes a wide variety of point-to-point and collective communication functions. The collective operations currently defined by MPI offer a high-level interface to the user. They insulate the user from implementation details and provide with MPI implementers the freedom to optimize their implementation for specific architectures. That is, although collective algorithms do not provide unique functionality per se, collective operations allow for significant advantages in programmability, safety, and performance [3].

Given these advantages, MPI parallel programming frequently draws on collective operations. A profiling study revealed that some applications spend more than eighty percent of transfer time in collective operations. Among the set of collective operations, MPI_Bcast and MPI_Allreduce are two important examples [4]. Open source MPI implementations of the collective operations generally assume that the target is a collection of compute nodes connected by a central switch (e.g. Ethernet and InfiniBand). These implementations generally have poor performance as the machine is scaled to several thousands of nodes. It is therefore essential for MPI implementations to provide high-performance collective operations.

Recently, many researchers have focused on 'optimal implementation' that provides a set of available collective algorithms according to the given system environment. The ultimate goal of such endeavors has been to improve collective operation. It is critical to note that the optimal implementation of a collective operation for a given system depends on many factors, including physical topology of the system, number of processes involved, message size, communicator size, and the location of the root node [5]. In the case of the MPI library cell, a collective operation is converted to point-to-point operation routine by a collective algorithm that is selected based on those factors.

Moreover, among all the collective communication primitives, the MPI_Bcast operation is relatively simple but of great importance. The MPI_Bcast routine is a one-to-all form of communication that copies data in the memory of the root process into the other processes in the same communicator. Many algorithms have been suggested as means to perform broadcast operation. Optimal implementation for MPI_Bcast is also a process that applies broadcast algorithms according to key factors such as message length and the number of processing nodes.

Despite the existence of many studies on optimal implementation, there have been no studies exploring the state of processing nodes. If the synchronous communication of other nodes is detected, the sequence of transmission is set by the state of processing nodes so that it reduces broadcasting time. As a result, this study proposes a novel scoreboard broadcast algorithm that applies a sequential tree while referring to the state of processing nodes. Additionally, we design a scoreboard MPI hardware unit that applies the proposed scoreboard algorithm. To then design the scoreboard MPI hardware unit, a standard mode MPI hardware unit [6] from the previous study is modified.

Manuscript received May 26, 2011.
Manuscript revised July 23, 2011.

[†]The authors are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea.
[††]The author is with the Future Network Research Division, ETRI, Daejeon, Korea.
a) E-mail: wychung@mpu.yonsei.ac.kr
b) E-mail: wro@yonsei.ac.kr
c) E-mail: yonglee@yonsei.ac.kr

DOI: 10.1587/transinf.E94.D.2523

2. Related Works

One of the studies on effective MPI broadcast operation focuses specifically on broadcast algorithms. Beginning with the sequential tree algorithm, in which the root node sends an individual message to all participating nodes, many broadcast algorithms are suggested. These include, for example, a chain tree, binomial tree, binary tree, minimum spanning tree (MST), distance MST, hybrid (by Van de Gejin) broadcast, and modified Van de Gejin broadcast. By including one of these algorithms that improves performance and is easy to implement in MPI library cell, broadcast operation is converted to a bundle of point-to-point operations.

Secondly, a study on how to include some of these algorithms in the MPI library cell and select algorithms based on the information of the MPI function is in progress [7]. The MPI_Bcast routine is as follows.

*MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm.)*

In other words, message size can be assessed through "buffer", "count", and "datatype", the location of the root node through "root", the communicator size, and the number of processes through "comm.". As a result, the algorithm can be selected using this information.

Thirdly, a study on accelerating point-to-point communication by designing hardware that processes MPI communication remains in progress [1]. If the hardware for MPI communication is used, the throughput is increased to a greater extent than when the software is used, and the Clock Per Instruction (CPI) of processor is increased because a hardware accelerator processes the MPI communication. Due to these advantages, a standard mode MPI hardware unit was proposed and designed in our previous study [6].

Despite this plethora of studies, there have been no studies considering the state of the processing node. Figure 1 highlights why the state of the processing node is important during the broadcast. In Fig. 1, processing node A is broadcasting sequentially to B, C, and D. Each processing node has one input and one output data port, which is the general structure. Let us now assume that the transmit order is 1, 2, and 3 according to the algorithm. In this case, if communication between B and D is ongoing, transmission 1 is delayed until the communication is completed. If processing node D sends larger size data, then transmission 1 is delayed even further. Given that transmission 1 has not completed, transmission 2 and 3 cannot proceed. If the data is transmitted in the order of 3, 1, 2 or 3, 2, 1, the broadcasting time can be shortened. Likewise, if the root node can detect whether the other leaf nodes are communicating, the order of transmission can be decided as 3, 1, 2, or 3, 2, 1. In this study, to check the status of each processing node, a scoreboard register that records the status is added to the previously designed MPI unit [6].

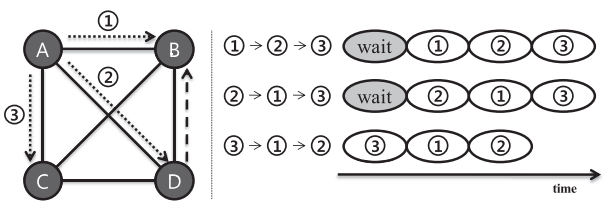

Fig. 1 The importance of the processing node state.

3. Proposed Algorithm & Architecture

Given the aforementioned factors, this study proposes a novel sequential tree algorithm based on the scoreboard. Figure 2 clarifies the method used to implement the broadcast algorithm. In Fig. 2 (a), which is a general design method, the MPI program is compiled and linked with the MPI library cell at the user level. Through this flow, the MPI_Bcast operation is converted into a bundle of point-to-point operations by a relevant broadcast algorithm. Figure 2 (b) indicates the design method that this study proposes, and a collective operation apart from broadcast operation is converted into a bundle of point-to-point communications by way of a modified MPI library at the user level. In the event of a broadcast operation, a modified MPI library cell generates an MPI_Bcast_Root message and MPI_Bcast_Leaf message. The MPI_Bcast_Root message is then sent to the root node at the transaction level and the MPI_Bcast_Leaf message is sent to leaf nodes - received nodes - at the same level. As the content of the scoreboard register is unknown at the user level, the broadcast algorithm should be applied at the transaction level.

Subsequently, the order of transmission is decided by reading the scoreboard register at the transaction level. Figure 2 (c) highlights the following procedure after MPI unit receives MPI_Bcast_Root message. In this case, the scoreboard unit reads the scoreboard information and records the information in the scoreboard register of the MPI unit. The Scoreboard algorithm unit then decides the order of transmission according to the scoreboard register value. Figure 2 (d) shows the following procedure after MPI unit receives MPI_Bcast_Leaf. In this case, if the other two leaf nodes are performing point-to-point communication, meaning the two nodes are in the 'busy' state, the two nodes wait

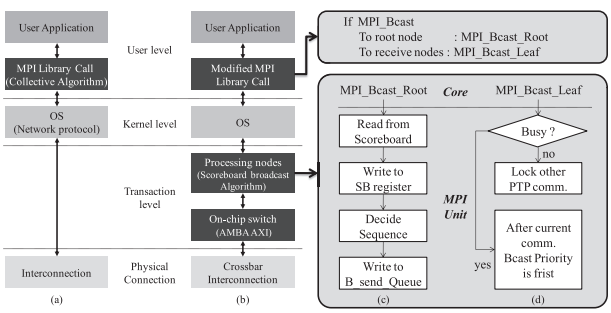

Fig. 2 Broadcast implementation & proposed algorithm.

to receive the data from the root node following the communication. If no communication is ongoing, meaning all nodes are in the 'wait' state, they wait to receive data from the root node.

Furthermore, Fig. 3 (a) represents the scoreboard MPI unit architecture with 4 processing nodes. When the MPI unit in each node is performing point-to-point communication, it transmits its state to the scoreboard. If the MPI unit is communicating, it records 1 for a 'busy' state, and if there is no communication, it records 0 for a 'wait' state in its own bit. The MPI unit of the root node, which received the MPI_Bcast_Root message, reads the scoreboard and records the information in the scoreboard register. As the number of processing node increases, the size of the scoreboard register increases linearly. The scoreboard algorithm unit then decides the order of transmission according to the sequential tree algorithm based on the scoreboard. Figure 3 (b) offers a block diagram of the processing node. In this block diagram, the scoreboard unit is added to the processing unit from the previous study [6]. At the same time, the local bus is added to the communication using the scoreboard.

Figure 4 depicts the process flow in the scoreboard MPI unit with clock base when MPI_Bcast_Root message is sent from the processor. If MPI_Bcast_Root message is transmitted in processor wrapper that is in MPI unit, scoreboard port wrapper sends SB read request signals to the scoreboard. In the next clock cycle, the scoreboard unit writes the information read from the scoreboard into the scoreboard register. At the same time, the scoreboard unit uses scoreboard information and selects processing node to be sent preferentially. Based on the selected processing node, the scoreboard unit generates a control message and records the message in the queue that is in the control message sender. In the third clock cycle, the control message is sent to receive node through AXI bus. In every cycle, the scoreboard unit generates control message and store the message in the queue. By processing the information in the queue sequentially, broadcasting is processed.

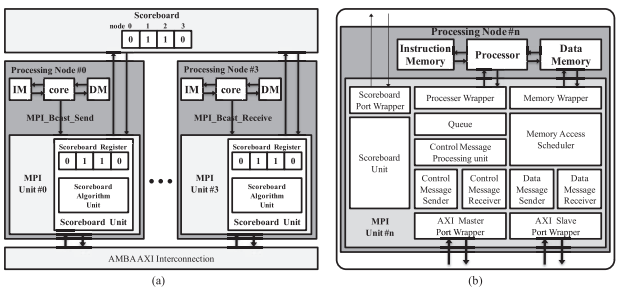

Fig. 3 Proposed scoreboard MPI unit architecture with 4 nodes.

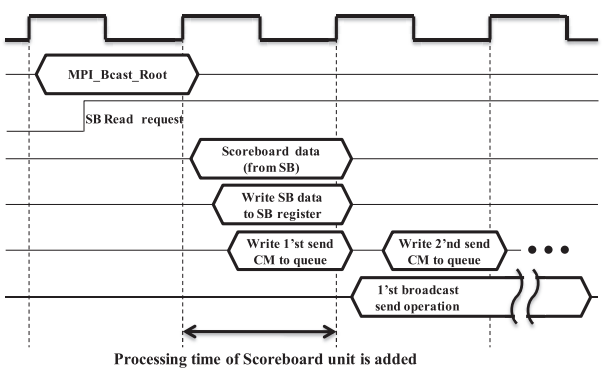

Fig. 4 Process flow in scoreboard MPI unit with clock base.

4. Simulation Results

To measure the broadcast communication performance of the scoreboard MPI unit, we compared it with the previous standard mode MPI unit [6]. Each unit is then designed as a Bus Functional Model (BFM) based on SystemC. We designed the BFM with consideration given to the delay time of each block, and it generates communication traffic in specific simulation environments. The following sections address the simulation results in greater detail.

4.1 Performance Evaluation of Proposed Model

In this section, we compared the proposed scoreboard MPI unit with the standard mode MPI unit from the previous study. This comparison was completed with regard to the results of the simulation in broadcast communications. The result of the simulation demonstrates the speed-up ratio of the proposed model when root node 0 broadcasts data to each leaf node.

Figure 5 depicts how node 0 broadcasts data to node 1, 2, and 3 when the total number of processing nodes is 4. We simulated the proposed model by increasing the amount of the broadcasting data from 4 bytes to 4k bytes. In Figs. 5 (a) and 5 (b), the other synchronous point-to-point communication between leaf nodes was fixed as 512 bytes and 2k bytes respectively.

In case 1, the broadcast register is '0000', representing no point-to-point communication. In case 2, the broadcast register is '0011', meaning there is another communication between node 2 and 3. In case 1 and 2, the transmission order in both models is 0 to 1, 0 to 2, and 0 to 3, though the performance of the proposed model is declined with the small amount of data. This is because the first data transmission is done without any conflict in both models. However, in the case of the proposed model, processing time that decides the transmission order through the scoreboard unit is added so that it takes more time to complete the broadcast. It is important to note that with small amount of data, the total transmission time is short, so there is slight performance degradation. In case 3, the broadcast register is '0101' or '0110', and the results of the simulation in both cases are nearly the same so the result of '0110' is used. In this case, the transmission order of the previous model was 0 to 1, 0 to 2, and 0 to 3, and the order of the proposed model is 0 to 3, 0 to 1, and 0 to 2. In the previous model, when the root node 0 transmits data to the leaf node 1, a conflict occurs as node 1 communicates with node 2 con-

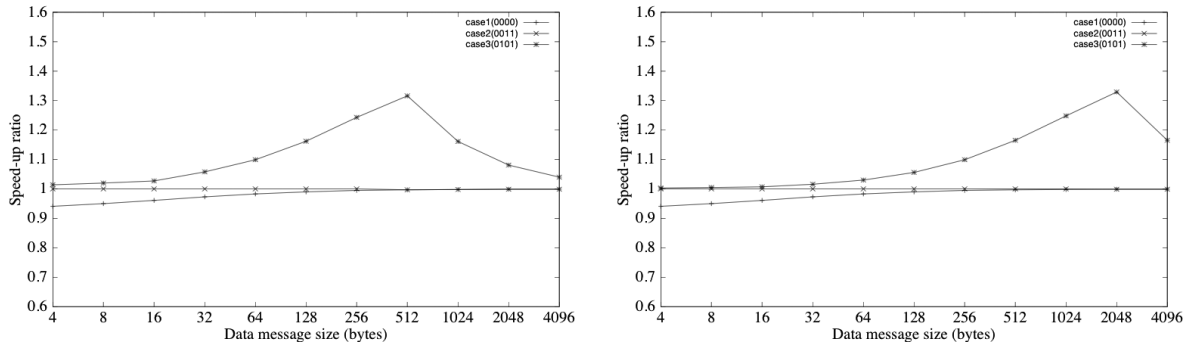

Fig. 5 Simulation results with 4 nodes (left (a): 512 B, right (b): 2 kB).

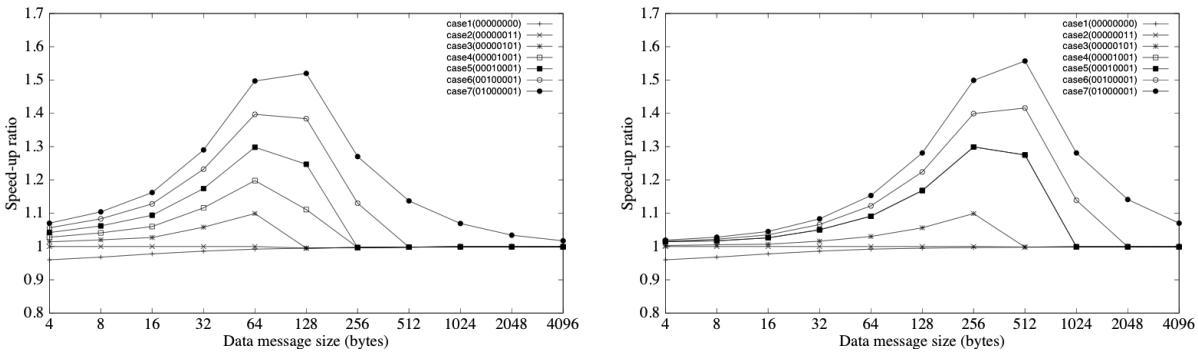

Fig. 6 Simulation results with 8 nodes (left (a): 512 B, right (b): 2 kB).

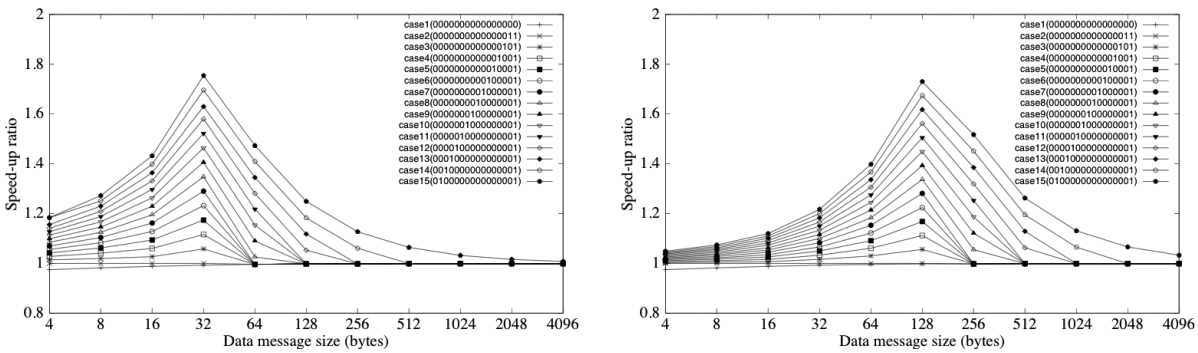

Fig. 7 Simulation results with 16 nodes (left (a): 512 B, right (b): 2 kB).

currently. Hence, the first transmission should wait until the transmission between node 1 and node 2 is completed. In the proposed model, however, there is no conflict because root node transmits data to node 3 first, which shortens the broadcasting time. For example, with the 2k bytes of broadcast data, it shows performance increase up to 32.9% (the previous model: 10300 ns, the proposed model: 7750 ns) as shown in Fig. 5 (b).

Figure 6 shows the result of the simulation with 8 nodes in the same experiment condition, while Fig. 7 shows the result of the simulation with 16 nodes. In the end, performance had increased up to 55.79% (the previous model: 7120 ns, the proposed model: 4570 ns) with 8 nodes, and increased up to 75.48% (the previous model: 1360 ns, the proposed model: 775 ns) with 16 nodes. The summary of the simulation overall is as follows:

1. If there is no communications between leaf nodes when the root node sends data, the processing time that root node uses to check the scoreboard unit is increased. This degrades the performance compared with the previous model. The degradation of the performance is obvious when the data size is small, but the total transmission requires a short amount of time with the smaller-sized data, so the impact on the whole system overall is slight.
2. The expected performance improvement is wide as the number of the processing node increases. Especially, in the event that the first node to communicate is 'busy', the two communication modes conflict with each other.

This allows for the performance to improve further.

4.2 Implementation and Verification

After verifying the performance through the SystemC simulator, we designed the scoreboard MPI unit using VerilogHDL. To this end, we synthesized a synopsys design compiler, and we used MagnaChip 0.18 μm for the synthesis library. To verify the proposed MPI unit, we designed the multiprocessor based on RISC MIPS DLX architecture. There are 2 processor nodes and a AXI bus on a chip. Each processor nodes is consisted of one RISC core, one instruction memory, one data memory and one MPI unit.

The memory occupies most of the areas, the MPI unit as well as the processor core occupy fewer areas. The number of equivalent NAND gates of the proposed scoreboard MPI unit is 24865.48, and it occupies less than 1.03% of the whole chip. It is approximately 1.01% larger than the previous MPI unit. In addition, when the number of processing nodes is 4, 8, and 16, the size of the scoreboard unit is 257.79, 495.96, and 987.96 respectively.

If the proposed scoreboard MPI unit is used, though it needs one more cycle (= 5 ns), to decide the order of transmission in the scoreboard unit, the total system performance increases. In addition, as hardware resource occupies less than 1.03% of the whole chip, cost is negligible.

5. Conclusion

In this paper, we proposed a novel algorithm and hardware structure for message passing interface (MPI) broadcast communication. The transmission order is set based on the state of each processing node that comprises the multicore system, so the novel algorithm minimizes the performance degradation caused by conflict. The proposed scoreboard MPI unit was evaluated by modeling it with SystemC, and implemented using VerilogHDL. According to the simulation result, the proposed algorithm shows highly improved performance as the number of nodes increases, and the performance increased up to 75.48% with 16 nodes. Furthermore, the proposed scoreboard MPI unit occupies less than 1.03% of the whole chip, and the size of the scoreboard unit occupies 0.005% of the scoreboard MPI unit. The proposed scoreboard MPI unit can also be used by adjusting the size of the scoreboard according to the number of the whole processing nodes. As a result, the scoreboard MPI unit is particularly useful from the perspective of scalability. Hence, with respect to low-cost design and scalability, this scoreboard MPI hardware unit is useful in significantly boosting the overall performance of the embedded MPSoC.

References

[1] D.L. Ly, M. Saldana, and P. Chow, "The challenges of using an embedded MPI for hardware-based processing nodes," Field-Programmable Technology (FPT) 2009, pp.120–127, Sydney, NSW, Dec. 2009.

[2] P. Mahr, C. Lorchner, H. Ishebabi, and C. Bobda, "SoC-MPI: A flexible message passing library for mutliprocessor systems-on-chips," International Conference on Reconfigurable Computing and FPGAs, pp.187–192, Dec. 2008.

[3] T. Hoefler, A. Lumsdaine, and W. Rehm, "Implementation and performance analysis of non-blocking collective operations for MPI," Supercomputing, SC '07, pp.1–10, 2007.

[4] R. Rabenseifner, "Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512," Proc. Message Passing Interface Developer's and User's Conference 1999 (MPIDC99), pp.77–85, 1999.

[5] J. Pjesivac-Grbovic, T. Angskum, G. Bosilca, G.E. Fagg, E. Gabriel, and J.J. Dongarra, "Performance analysis of MPI collective operations," Cluster Computing, vol.10, Issue 2, pp.127–143, June 2007.

[6] W. Chung, H. Jeong, W.W. Ro, and Y. Lee, "A low-cost standard mode MPI hardware unit for embedded MPSoC," IEICE Trans. Inf. & Syst., vol.E94-D, no.7, pp.1497–1501, July 2011.

[7] G. Almasi, C.J. Archer, C.C. Erway, P. Heidelberger, X. Martorell, J.E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of MPI collective communication on BlueGene/L systems," ICS '05, Boston, MA, USA, June 2005.