A High Performance Simulator System for a Multiprocessor System Based on a Multi-way Cluster

Arata Shinozaki[1], Masatoshi Shima[1], Minyi Guo[2], and Mitsunori Kubo[1]

[1] Future Creation Lab., Olympus Corp., Shinjuku-ku, Tokyo 163-0914, Japan
`{arata_shinozaki, masatoshi_shima, mi_kubo}@ot.olympus.co.jp`
[2] School of Computer Science and Eng., University of Aizu, Aizu-Wakamatsu,
Fukushima 965-8580, Japan
`minyi@u-aizu.ac.jp`

Abstract. In the ubiquitous era, it is necessary to research on the architectures of multiprocessor system with high performance and low power consumption. A processor simulator developed in high level language is useful because of its easily changeable system architecture which includes application specific instruction sets and functions. However, there is a problem in processing speed that both PCs and workstations provide insufficient performance for the simulation of a multiprocessor system. In this research, a simulator for a multiprocessor system based on the multi-way cluster was developed. In the developed simulator system, one processor model consists of an instruction set simulator (ISS) process and several inter-processor communication processes. In order to get the maximization of the simulation performance, each processor model is assigned to the specific CPU on the multi-way cluster. Also, each inter-processor communication process is implemented using MPI library, which can minimize the CPU resource usage in a communication waiting state. The evaluation results of the processing and communication performance using a distributed application program such as JPEG encoding show that each ISS process in the developed simulator system consumes approximately 100% CPU resources for keeping enough inter-processor communication performance. This result means that the performance increases in proportion to the number of integrated CPUs on the cluster.

1 Introduction

In the ubiquitous era, micro processors are widely used in many applications including not only PCs and PDAs but also home appliances and cars. For these applications, high performance parallel computing is required for multimedia codec processing, digital signal processing, and secure communication processing. Performance improvement by increasing the operating frequency of a processor is reaching the upper-bound because of leakage current and power consumption [1]. Both multi-core processor systems and multiprocessor systems [2, 3] are effective solution to improve performance without increasing operating frequency. However, they still consume a lot of power. To solve these problems, it becomes necessary to research on the architecture of heterogeneous multiprocessor system with application-specific instruction sets and functions, so that it optimizes the balance between high performance and low power consumption, and lowers the redundancy of processing.

In general, a simulator developed in a high level language is useful for the research on the architecture of a processor system, because the system architecture can be easily changeable [4-8] with additional application specific instruction sets and functions. However, there is a problem that both PCs and workstations provide insufficient performance for the simulation of a multiprocessor system.

In this research, a simulator system for a multiprocessor system was developed based on a multi-way cluster. It is expected to provide high parallelism and high performance with multiple CPUs on each node. This paper shows the usefulness of the developed simulator system according to the evaluation results on the processing capability and communication performance using a distributed processing application.

2   PE and PE Network

Fig. 2.1 shows a schematic diagram of a multiprocessor system. In the part (a), each CPU is a Processing Element (PE). The system bus connecting the CPUs is called a PE Network. From this point of view, a part of multiprocessor system can be simplified as PEs connected by a PE Network.
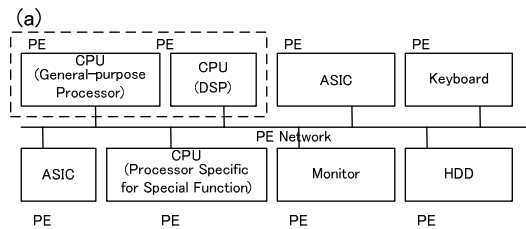
Fig. 2.1. Schematic Diagram of a Multiprocessor System

A PE can be also viewed as a processor, which contains a general purpose processor, a DSP, and a processor specific for special function, with application specific instruction sets and functions. Furthermore, a PE contains peripherals including a monitor, a keyboard, or a HDD. Similarly, PE Network can be viewed as a peripheral bus, a system bus, the Internet, or other networks. The following sections show that the simulator system can connect various PEs with common network interface and protocol. Then, they show the simulator system is useful for the simulation of a heterogeneous multiprocessor system.

3   Simulator System

3.1   Simulator System

Fig. 3.1 shows the simulator system architecture. The simulator system consists of a simulation engine, a control center (CC), and a GUI.
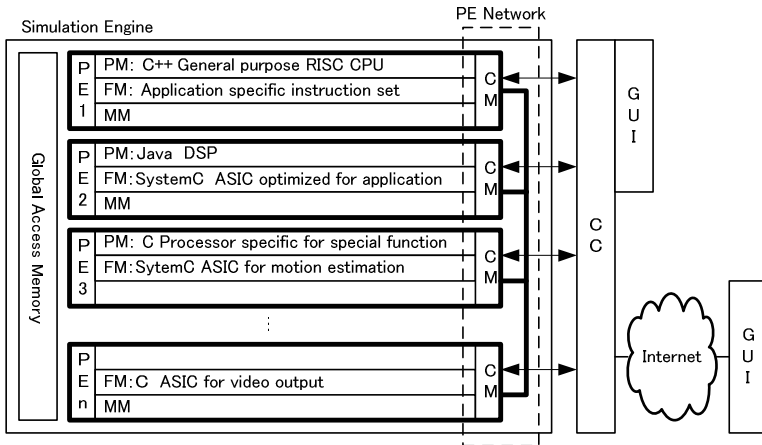
Fig. 3.1. Simulator System Architecture

The simulation engine consisted of (1) PEs, (2) PE Network, and (3) Global Access Memory. Every PE is constructed from (a) Processing Module (PM), (b) application-specific Function Module (FM), (c) Memory Module (MM), and (d) Communication Module (CM). A PM is a core information processing engine. An FM provides application specific instruction sets and functions.

For example, PE1 integrates a general purpose CPU as a PM extended its function with an application specific instruction set as an FM. PE2 accelerates a DSP with an ASIC optimized for an application as an FM. In PE3, a processor specific for special function cooperates with an ASIC for motion estimation, which requires high accuracy and massive data processing in video processing.

An MM is the registers and local memory of a PM and an FM. An MM also serves as a memory for inter-PE communication. MM stores information sent and received over the PE network, because the MM is accessible from CMs. A CM is the protocol-independent general model of PE Network. There is no special module which connects CMs. The connection information provided in each CM decides the structure of PE Network. Also, system information is sent and received through CMs. The Global Access Memory stores information shared between PEs such as large amount of video data.

The CC analyzes information of user's operations, sends control information to the simulation engine, receives simulation results from it, and controls the whole system. The CC also works as the gateway of the simulation engine, and operates the simulation engine as if it were a part of a large information processing system.

As the GUI is independent from a platform it runs, it can operate the simulation engine through the CC over various networks.

PMs and FMs can be implemented with the following various models focusing on different functions and abstraction levels: an untimed function model which simulates only function without time concept, an instruction set model which simulates only the behavior of instructions, a cycle-accurate model [9, 10] which defines the behavior in

each clock cycle, and a RTL model which is equivalent to the target hardware. These models are written in a language such as C, C++, Java, SystemC [11, 12], or their combination.

3.2 Simulator System Platform

Fig. 3.2 shows a simulator system platform based on a multi-way cluster which integrates three nodes with different number of CPUs.
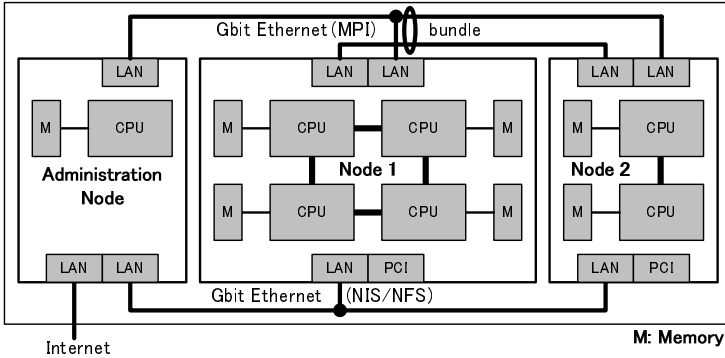

Fig. 3.2. Simulator System Platform

A one-way administration node executes the GUI and the CC, and controls the whole simulator system. Each of 6PEs is assigned to the specific CPU in a four-way node 1 and a two-way node 2 for high-speed simulation. Mainly, the node 1 processes main program and the node 2 executes pre-processing, post-processing, and external I/O processing.

The intra-node communication is implemented with high-speed system bus and memory bus exclusively used for the specific CPU. The simulator system platform implemented Opteron CPU [13] operated in 2.4GHz. One CPU can directly connect up to three adjacent CPUs through HyperTransport [14], which is bidirectional multi-channel system bus possible to transfer data at 6.4GBytes/sec. Each CPU has 4GB memory directly-accessible at 3.2GBytes/sec.

The inter-node communication is implemented with a Gigabit-Ethernet. The Ethernet cables can be logically bundled to avoid performance degradation caused by the limit of bandwidth. Inter-PE communication model is implemented using MPI [15] library which can program the timing of a request for communication and the sequence of communication handshake.

MMs and the Global Access Memory are implemented as shared memory on the OS. If the shared memory is created on other CPUs, a PM, an FM and a CM can access them in high speed through HyperTransport.

SuSE Linux 9.1 Professional (kernel 2.6) [16] facilitates process assignment to each CPU, process control, MPI programming, and the creation of shared memory under 64-bit environment.

3.3   Implementation of Simulator System

Each PM of 6 PEs is implemented with a proprietary MIPS R2000/3000 instruction set [17] simulator (ISS) in C++. An ISS can simulate each step of instruction behavior, and it is suitable for the high-speed verification of algorithm of applications. For the inter-PE communication, system call 1 instruction (syscall1) and system call 2 instruction (syscall2) were expanded from the system call instruction in the MIPS instruction set. Syscall1 requests to send data for the data receiving PE, and syscall2 notifies that the data receiving PE finished using data for the data sending PE.

MMs are implemented with 256KB local memory and registers. The registers consist of general-purpose registers, exception registers, and extended system control registers. The Global Access Memory is composed of 4MB internal bulk memory on the node 1, 4MB external memory, and also 4MB media memory on the node 2.

3.4   Details of Simulator System

3.4.1   Process Organization and CPU Assignment
Fig.3.3 shows process organization and CPU assignment for the communication between PE1 and PE2 / PE3. The PE is implemented with one PM process, i.e., MIPS ISS, and CM processes consisted of pairs of sending process and receiving process. These pairs are used for (1) inter-PE communication and (2) communication with the CC. All processes of a specific PE is assigned to a corresponding CPU. Therefore, all the processes in the PE are possible to use almost 100% of the specific CPU resources. All the processes are accessible to the registers and the local memory in the MM. Fig. 3.4 shows the local memory map.
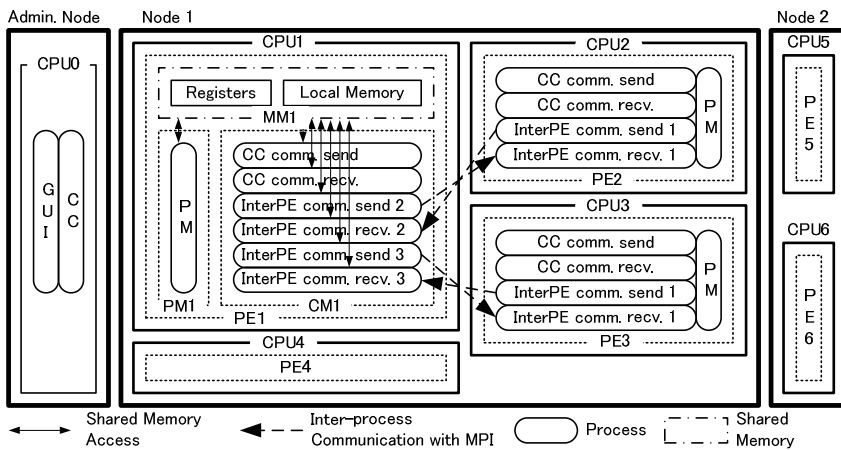

Fig. 3.3. Process Organization and CPU Assignment

3.4.2   Communication Module (CM)
A CM is implemented using MPI library as native code on the multi-way cluster, and works with a pair of a sending process and a receiving one. MPI enables to write the detailed sequence of the communication handshake as intended.
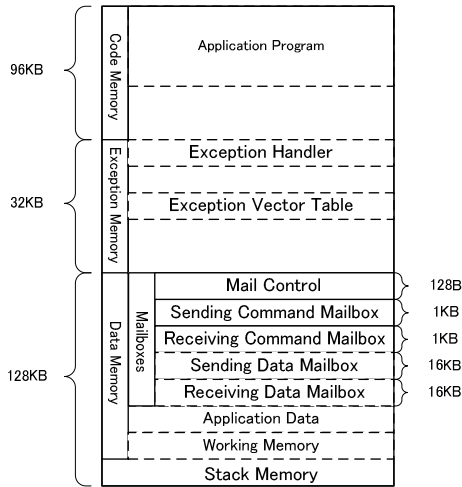
Fig. 3.4. Local Memory Map

When the simulator extends the network structure by increasing or decreasing the pair of the CM processes, these CM processes are required to minimize the influence on the PM process. Then, every sending process or receiving process minimizes its CPU resource usage, organized with a loop starting from MPI_Recv() function to block its execution and sleep in a waiting state. This enabled the PM process to use almost 100% of the specific CPU resources.

3.4.3 Mailbox

Necessary information for inter-PE communication includes (1) control information (i.e., commands) and (2) data stored in command mailbox and data mailbox respectively. They consist of sending mailbox and receiving mailbox constructed on the local memory.

The command mailbox uses 64-byte fixed area, and stores the following commands: (1) ID of data sending and receiving PE, (2) communication type, (3) address to store data in data sending and receiving PE, (4) data size and number of packets, and (5) repeat count of data communication. The command of the communication type stores the signal to:

   a.   request to send data issued from the data sending PE (REQ signal)
   b.   respond for the REQ signal issued from the data receiving PE (ACK signal)
   c.   notify that the data receiving PE finished using data(FIN signal)

The suitable size and the number of data mailbox can be defined according to the characteristics of the application.

The Mail Control was constructed at the top of mailbox area. It manages communication status using the following register and flags:

   A.   Sending Session Counter (SSC) to count the number of communication session

B. Sending Mailbox Full (SMF) to notify whether the sending mailbox is empty to the PM process

C. Receiving Mailbox Full (RMF) to notify whether the receiving data mailbox receives data to the PM process

3.4.4 Sequence of Communication

Fig. 3.5 shows the sequence of the inter-PE communication to send data from PE1 to PE2 with a single communication buffer. PM1 app. and PM2 app. express the description of application program running on PM1 and PM2 respectively. PM1 and PM2 express the behavior of PMs which application programs cannot detect, for example the control of a flags and mailboxes etc.

(flow)

(1) [PM1 app.] stores sending commands including the REQ signal and the address to indicate the top of sending data to the sending command mailbox.

(2) [PM1 app.] stores sending data to the sending data mailbox.

(3) [PM1 app.] executes syscall1.

(4) [PM1] sets SCC and SMF.

(5) [PM1] extracts the ID of PE2 from the sending command mailbox, then, sends the sending commands including the REQ signal to CM2

(6) [CM2] receives the commands including the REQ signal, and returns from a waiting state.

(7) [CM2] stores the sending commands including the ACK signal and the address to indicate the top of the receiving data to the sending command mailbox.

(8) [CM2] sends the sending commands including the ACK signal to CM1

(9) [CM1] receives the commands including the ACK signal, and returns from a waiting state.

(10) [CM1] sends the sending data in the sending data mailbox to CM2

(11) [CM1] clears SMF, and waits the next commands by MPI_Recv(). PM1 app. becomes able to store the sending commands and data for the next session to the sending mailboxes.

(12) [CM2] receives the data in the receiving data mailbox.

(13) [CM2] sets RMF, and enters a waiting state by MPI_Recv().

(14) [PM2 app.] detects RMF, and stores the received data to the working memory

(15) [PM2 app.] sets the sending commands including the FIN signal to the sending command mailbox.

(16) [PM2 app.] executes syscall2.

(17) [PM2] clears RMF.

(18) [PM2] extracts the ID of PE1 from the sending command mailbox, then, send the sending commands including the FIN signal to CM1

(19) [CM1] receives the commands including the FIN signal, and returns from a waiting state.

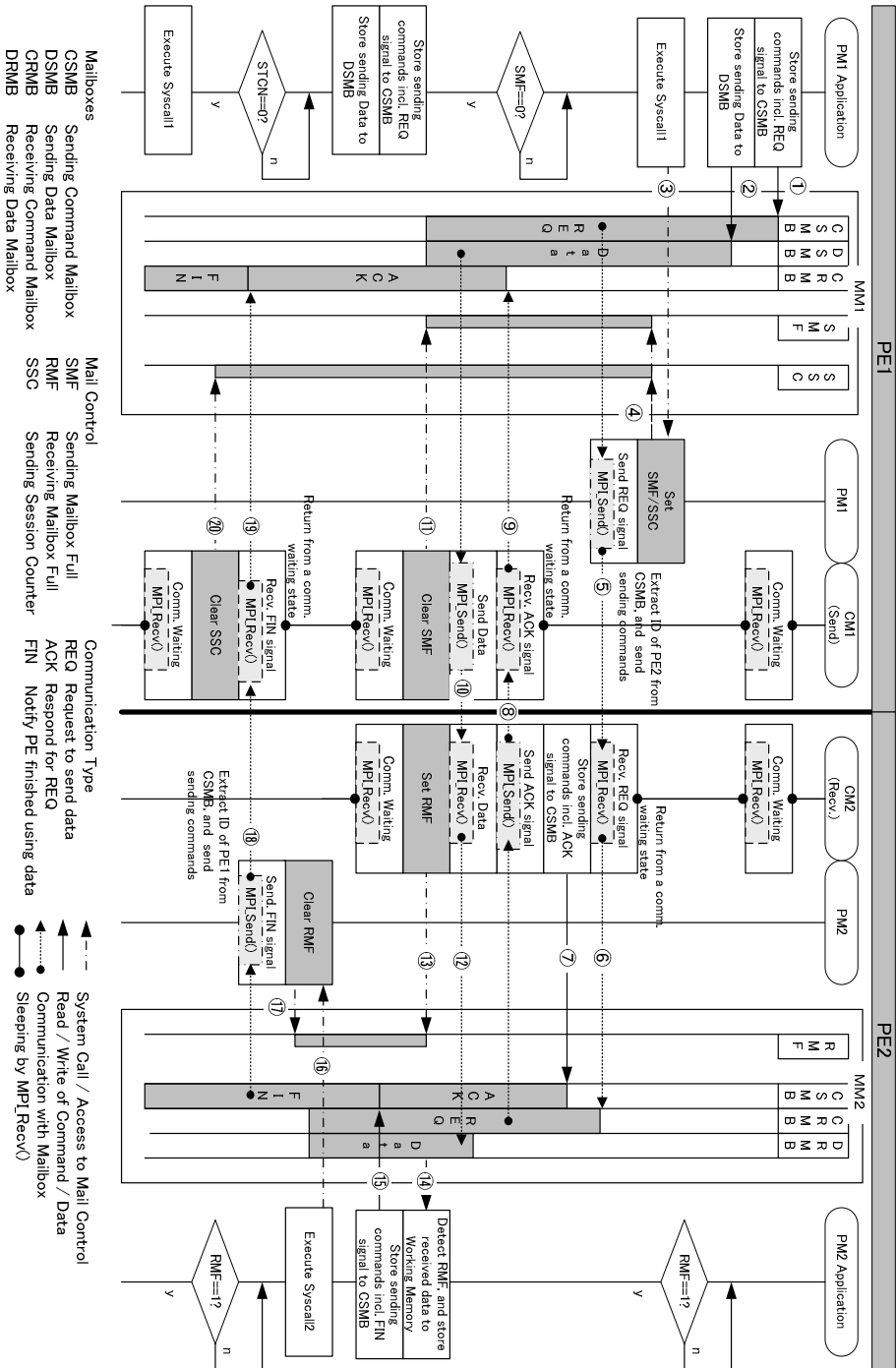(20) [CM1] clears SSC, and enters a waiting state by MPI_Recv(). After that, PM1 app. can execute syscall1.

Fig. 3.5. Sequence of Inter-PE Communication with a Single Buffer

4   Performance Evaluation

This section shows the basic processing performance of PE without CM processing and the basic communication performance of CM without PM processing. After that, we will evaluate the processing and communication performance of this simulator system and its usefulness, using an application -- JPEG encoding program.

4.1   Performance Evaluation of PM Processing

The following five application programs were selected for performance evaluation of PM processing: (1) transposition of 32x32-bit matrix, (2) two-dimensional DCT for 8x8-element matrix, (3) a test program for arithmetical and logical instructions (4) a test program for branch and jump instructions, and (5) a test program for memory access instructions. Each application program executed 1G instructions on each of 1 PE, 2 PEs, 4 PEs, or 6 PEs without inter-PE communication, measuring the processing time to evaluate the processing performance of ISSs as PM. Fig. 4.1 shows the average processing performance and CPU resource usage of PM.
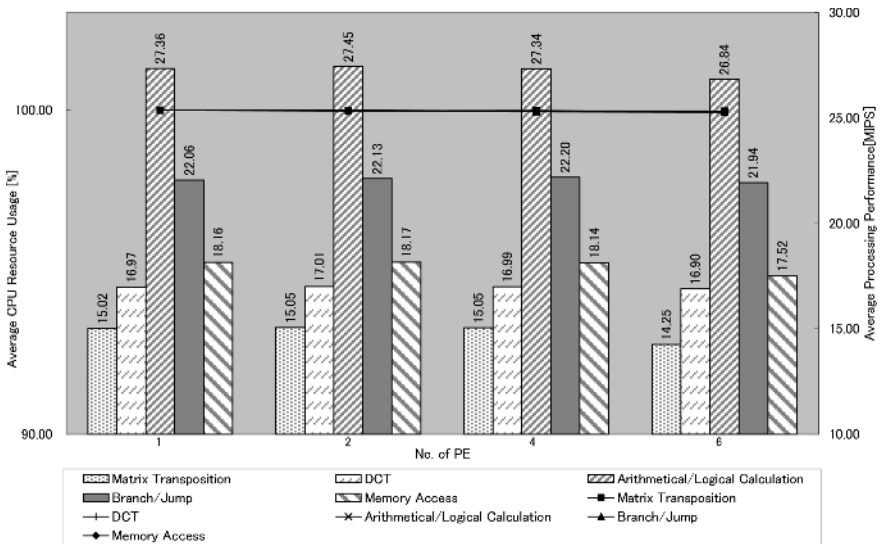

Fig. 4.1. Average Processing Performance and CPU Resource Usage of PM

This instruction simulator caches decoded instructions. Therefore, the processing performance of PM depends on the cache hit rate. In terms of matrix transposition, which shows the lowest performance, the average processing performance of PM was over 14MIPS in 6PE case. The performance is only 5% lower than that in 1 PE case. Other application programs show almost the same results. For each PE was assigned to the specific CPU, the CPU resource usage in all of application programs shows almost 100% as expected.

4.2 Performance Evaluation of CM Processing

On the intra-node and inter-node communication models shown in Fig. 4.2, (1) 64-byte packets same as a command mailbox and (2) 16K-byte packets same as a data mailbox were sent and received sequentially in 60 seconds. Each communication channel has no dependency. The average of inter-PE communication speed of all channels and the CPU resource usage were calculated. To measure the communication performance while all PMs consume almost 100% of CPU resources, the execution priority of each CM process was set to be low. Table 4.1 shows the results of the measurement.
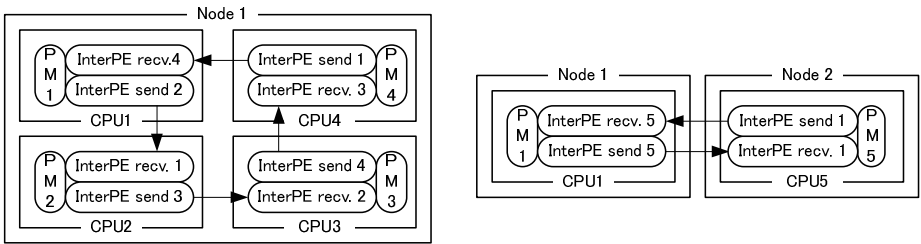

Fig. 4.2. Intra-node Comm. Model (left) / Inter-node Comm. Model (Right)

Table 4.1. Average Inter-PE Comm. Speed and CPU Resource Usage of PM and CM

Communication Model		Intra.		Inter.	
Size of Packet[Bytes]		64	16K	64	16K
Average CPU Resource Usage	PM[%]	94.58	98.87	73.31	79.58
	CM(send)[%]	4.31	0.38	6.21	10.05
	CM(recv.)[%]	1.11	0.74	20.49	10.37
Average Comm. Speed in All InterPE Comm. Channel	[Packets/sec]	1,222	476	63,991	4,805
	[Mbps]	0.60	59.52	31.26	601.07

Regarding the inter-node communication, the performance of CM kept 600Mbps worth of 4800 packets/sec with 64KByte data packets because of the buffering mechanism in the Ethernet board. Considering a MPI header, actual communication performance is higher than 600Mbps. It consumes almost all of communication bandwidth of Gigabit-Ethernet. If necessary, bundling the Ethernet cables is able to expand the bandwidth. For an application in which communication plays an important role, it is possible to increase communication speed by lowering the CPU resource usage of PM. The same can be said for the intra-node communication.

On the other hand, regarding the intra-node communication, communication speed is 1200 packets/sec with 16KB packets and 500 packets/sec with 64B packets using 4 channels, while PM consumes about 95% CPU resources. If much higher communication speed is required, it is possible to share the information without communication by constructing mailboxes on the Global Access Memory based on the characteristics of the multi-way cluster.

4.3 Performance Evaluation of Simulator System Using JPEG Encoding Application Program

The JPEG encoding application is divided into six sub-programs and executed on each PE as shown in Table 4.2. An input image is specified with the format of 24-bit-depth bitmap file, the size of VGA (640x480), the sampling factor of 4:2:2, and the quality of 75. Table 4.2 shows the processing performance of PM, the CPU resource usage of PM, the times and the processing rate defined below, the data size in inter-PE communication, and the performance of CM processing.

- PE operating time   = the elapsed time from the beginning of processing to the end of it on each PE
- PM operating time   = the time consumed as a user process out of PE operating time
- PM waiting time   = PM operating time * (No. of instructions executed in a communication waiting state / No. of all instructions)
- PM processing time   = PM operating time – PM waiting time
- CM communicating time = PE operating time – PM operating time
- PE processing rate   = (PM processing time + CM communicating time) / PE operating time

Table. 4.2. Simulator System Performance Using JPEG Encoding Application Program

Assigned Node	Node 2	Node 1				Node 2	
Assigned PE	PE5	PE1	PE2	PE3	PE4	PE6	Average
Function	Bitmap File Reading	RGB to YCrCb	Down Sampling	DCT/ Quantization	Huffman/R un-length	JPEG File Writing	
PM Processing Performance[MIPS]	15.29	14.03	14.58	14.08	14.06	15.68	14.62
PM CPU Resource Usage	99.32%	99.52%	99.95%	99.97%	99.90%	99.95%	99.77%
PE Operating Time[sec]	21.41	21.40	21.40	21.40	21.40	21.41	
PM Operating Time[sec]	21.24	21.20	21.31	21.31	21.28	21.32	
PM Processing Time[sec]	0.36	3.07	1.55	8.16	2.19	0.02	
PM Waiting Time[sec]	20.88	18.13	19.76	13.15	19.08	21.30	
CM Commnicating Time[sec]	0.17	0.20	0.09	0.09	0.13	0.09	
PE Processing Rate	2.50%	15.28%	7.65%	38.55%	10.84%	0.50%	12.55%
Size of Receiving Data[KBytes]	0	900	900	600	1200	～600	
Size of Sending Data[KBytes]	900	900	600	1200	～600	0	
CM Processing Performance [MBytes/sec]	5.05	8.75	16.10	19.75	～14.06	～6.73	

Each PM could achieve over 14MIPS performance with CM processing. CM could achieve over 5MBytes/sec communication performance using less than 1 % CPU resources. The performance of PM itself is equivalent to the performance in the case of single PM processing without CM processing shown in Fig. 4.1. The performance of CM is higher than that of CM processing without PM processing, which is shown in Table 4.1. However, this JPEG encoding application used only single communication buffer and could send the next data only after detecting the execution of data processing. As a result, 90% of PE operating time was consumed for a waiting state, and the simulation time exceeded 20 seconds for the encoding of one JPEG picture. In the next step, the communication waiting time will be reduced using multiple

communication buffer and overlapping PM processing and CM processing to improve the performance.

One of the performance improvement methods is averaging the processing times in all of PMs and optimizing the partitioning of functions assigned to each PE by detecting performance bottlenecks. This simulator system enables researchers to profile and detect the bottlenecks of the processing time and waiting time of each PE. For example, the result in Table 4.2 implies that PE3 is one of bottlenecks consuming the longest processing time, and partitioning them into DCT and quantization will be effective. Also, it implies PE6 consumes less than 1% PE processing rate, and combining it with PE4 will be effective. Thus, the simulator system is useful to optimize the partitioning of functions, and research new additional instruction sets and application specific functions. As a result, the simulator system is suitable to research the architecture of a heterogeneous multiprocessor system.

5   Conclusion

This research focused on a simulator system for a multiprocessor system based on the high performance multi-way cluster integrated multiple CPUs on each processing node. The implemented simulator system maximized its performance, assigning each PE to the specific CPU on the multi-way cluster and implementing CM with MPI which can minimize the CPU resource usage in a communication waiting state. The simulator system executed application at over 14MIPS on each PM, achieving communication performance at over 5MBytes/sec with the distributed processing of JPEG encoding using single communication buffer. Thus, this showed that the implemented simulator system is useful for the simulation of distributed application on a multiprocessor system.

This paper showed that the implemented simulator system enables to profile application processing time and waiting time of each PE, detect bottlenecks, optimize the partitioning of functions in available PE resources, and research additional new instruction sets and application specific functions for the architecture of a heterogeneous multiprocessor system suitable for the ubiquitous era.

In the future, extending the current simulator system, we will research and develop a full-scale simulator system suitable for a multi-stream application with new sets of PE including ISSs and cycle-accurate models with additional instruction sets, application specific untimed function models, and RTL models in a system description language such as SystemC. The simulator system will promote the research on the target multiprocessor system.

References

1. Matsuzawa, A., Issues of Current LSI Technology and the Future Technology Direction. IEICE Transactions vol. J87-C No.11, pages 802-809, 2004.
2. Pham, D., et al., The Design and Implementation of a First-Generation CELL Processor – A Multi-Core SoC. ICICDT 2005 pages 49 – 52, 2005

3. Intel PentiumD Processor, http://www.intel.com/products/processor/index.htm (March, 2006)

4. Imafuku, S., Ohno, K., and Nakashima, H. Reference filtering for distributed simulation of shared memory multi-processor. In Proc. 34th Annual Simulation Symposium, pages 219—226, May 2001.

5. Mukherjee, S., Reinhardt, S., Falsafi, B., Litzkow, M., Huss-Lederman, S., Hill, M., Larus, J., and Wood, D. Wisconsin Wind Tunnel II: A fast and portable parallel architecture simulator. In Proc. Workshop on Performance Analysis and Its Impact on Design, June 1997.

6. Rosenblum, M., Herrod, S., Witchel, E., and Gupta, A. Complete computer system simulation: The SimOS approach. IEEE Parallel & Distributed Technology, 3(4): 34—43, 1995.

7. Veenstra, J., and Fowler, R. Mint: A front end for efficient simulation of shared-memory multi-processor. In Proc. MASCOTS'94, pp. 201—207, 1994.

8. Cmelik, R., and Keppel, D. Shade: A fast instruction set simulator for execution profiling. In Proc. of 1994 ACM SIGMETTRICS Conference on Measurement and Modeling of computer systems, Philadelphia, 1996.

9. Shima, M., Shinozaki, A., Sato, T. Cycle-Accurate Processor Modeling Written in Java Lan-guage. IEICE CPSY2002-53, pages 13-18, 2002.

10. Shima, M., Shinozaki, A., Ohta, S., Ito, K. Cycle-Accurate System Modeling in Java. IEICE VLD2002-146, pages 1-6, 2003.

11. Grotker, T., Liao, S., Martin, G., Swan, S. System Design with SystemC, Kluwer Academic Publishers, 2003.

12. SystemC Community. http://www.systemc.org/ (March, 2004).

13. AMD Opteron Processor, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8825,00.html (Current March, 2006).

14. HyperTransport Consortium, http://www.hypertransport.org/ (Current March, 2006).

15. Pacheco, P., Parallel Programming with MPI. Morgan Kaufmann Publishers, CA, USA, 1997.

16. SUSE Linux, http://www.novell.com/linux/ (Current March, 2006)

17. Kane, G., Heinrich, J. MIPS RISC ARCHITECTURE. Prentice Hall PTR, New Jersey, USA, 1992.