Vol. 39 No. 7 Transactions of Information Processing Society of Japan July 1998

Regular Paper

Dynamic Scheduling Algorithms for Real-time Signal Processing on Ring-based Multiprocessor Systems

Hiroyuki Miyata,[†,☆] Masato Takahashi,[†,☆☆] Hiroyuki Takano[††] and Kazuhiro Aoyama[††]

This paper describes a study of the performance of dynamic scheduling algorithms used in the design of distributed real-time systems. The study is based on a simulation model assuming a ring-based shared memory multiprocessor. An application is a multiple-target-tracking algorithm. In order to design an optimal dynamic scheduling algorithm with several requirements, we classified scheduling schemes according to the essential procedures for dynamic scheduling, and proposed four scheduling algorithms. The results show that a scheduling algorithm with distributed qualification and a resource reservation mechanism significantly improves system performance.

1. Introduction

Recent technological advances in multiprocessors and VLSI are making hard real-time signal processing feasible. A key to realizing this goal will be research on the architecture of multiprocessors for signal processing.

The choice of an interconnection network is one of the most important considerations in the architecture of multiprocessors. Several schemes for interconnection of multiprocessors have been proposed, such as a shared bus, a cube network, and a ring network[1),2)]. Since a shared bus is connected to a shared memory and some processors, it is usually difficult to speed up the bus because of physical constraints, while a cube network becomes increasingly complex as the number of nodes grows large. A ring network, however, is a simple form of point-to-point interconnection and, since the function of its routers in the interface block is also simple, it could allow faster operation speeds and less expensive hardware. From this reason, we use a ring network for multiprocessors.

Use of a dynamic scheduling algorithm is another key technique for real-time signal processing on multiprocessors. There has been a great deal of discussion in the literature about dynamic scheduling algorithms on multiprocessors[3)~7)]. For example, Casavant, et al.[3)] presented a taxonomy of scheduling algorithms in distributed computing systems, Wang, et al.[4)] defined the quality of load sharing (Q-factor) and evaluated several scheduling algorithms using the Q-factor, and Stankovic, et al.[5)] presented a scheduling algorithm with hard real-time constraints. However, there have been few papers about dynamic scheduling algorithms with hard real-time constraints on ring-based multiprocessor systems. Though a ring network is a simple form, the maximum distance between two nodes is $n-1$ if unidirectional links are used in an n-node system. Any delay in communication between nodes may affect dynamic scheduling algorithms more than other interconnections of multiprocessors.

In this paper, we propose dynamic scheduling algorithms for real-time signal processing on ring-based multiprocessor systems, and evaluate them by means of simulation. In order to avoid communication delay between nodes, a reservation mechanism is adopted. If one node is qualified to accept a task, the resource in the node (for example, CPU power) is reserved before moving the task. Qualification mechanisms are classified as centralized or distributed. Our interest is to determine which type of mechanism is better for real-time signal processing.

In the next section, we explain a real-time signal processing algorithm called Multiple Hypothesis Tracking (MHT). After that, we give an overview of our system. Then, we discuss our proposals for dynamic scheduling algorithms in order to provide guidelines for system

† Information Technology R&D Center, Mitsubishi Electric Corporation
†† Kamakura Works, Mitsubishi Electric Corporation
☆ Presently with Kamakura Works, Mitsubishi Electric Corporation
☆☆ Presently with Communication Research Laboratory, Ministry of Posts and Telecommunications

design. Finally, we discuss classified dynamic scheduling algorithm schemes and explain our selection of the best scheme for our system.

2. Multiple Hypothesis Tracking: MHT

There have been several recent studies of a tracking algorithm called Multiple Hypothesis Tracking (MHT)[9),10)] in relation to multi-target tracking, robot vision, and automobile navigation. In MHT processing, the number of computational tasks can rapidly increase according to the number of target signals and clutter (unnecessary signals). The MHT algorithm theoretically generates hypotheses corresponding to all combinations of signals from targets and clutter. But it is difficult to implement a pure MHT algorithm, even if a multiprocessor is used, because of computational explosion. In order to avoid this problem, hypotheses are generally weighted according to their probability, which is calculated from the correspondence between a predicted position and an observed position. Thus, a hypothesis will be eliminated if its probability is below a pre-selected value.

3. System Overview

Figure 1 shows an overview of a system whose nodes are connected by SCI: Scalable Coherent Interface (IEEE Std 1596-1992)[8)]. Later, we will call a processor a "node".

SCI offers the following benefits: (1) low latency, (2) ease of building distributed shared memory, and (3) the fact that it is a public (IEEE) standard. We use a real-time OS that has the following benefits: (1) ease of enhancement into a parallel version, (2) good real-time performance, and (3) executability in a microprocessor. The microprocessor that we use offers the following benefits: (1) good programmability and (2) availability for general use.

The hardware design trade-off is described in Ref. 11) and 12).

4. Dynamic Scheduling Algorithms

We define dynamic scheduling as that in which an OS decides the scheduling of a task dynamically in the system, and static scheduling as that in which a compiler decides the scheduling of a task according to a program. Dynamic scheduling algorithms in real-time systems are classified in **Table 1**, according

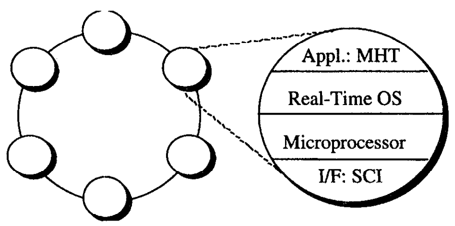


Fig. 1 System overview.

Table 1 Classification of dynamic scheduling algorithms.

	Static	Dynamic
Centralized System	A	B
Distributed System	C	D

to a procedure described by Stankovic, et al.[5)]. There have been several studies of type A and C. Optimal dynamic scheduling algorithms in A and C are not generally optimal in B and D. In type B, work is currently in progress. On the other hand, type D is described as a challenging area. Our study, which centers on schemes for dynamically allocating tasks on a system with distributed nodes to optimal execution nodes, is of type D.

4.1 Dynamic Scheduling Algorithms on Distributed Systems

Basically, dynamic scheduling algorithms on distributed systems are divided into two sequential portions: a global scheduling portion and a local scheduling portion. The global scheduling portion is a mechanism for distributing the load to other nodes. The local scheduling portion is a mechanism for guaranteeing the time conditions in each node. The load distribution mechanism decides in which node the task should be executed. The guarantee mechanism decides whether the node can satisfy the deadlines for the tasks.

In the global scheduling portion, it's usually difficult to know the exact current status of the all nodes because of communication delays in gathering the loads from each node. In addition, gathering load information frequently leads to heavy network traffic and low system efficiency.

On the other hand, in the local scheduling portion, our system uses a simple algorithm to judge whether or not the task can be executed within the pre-selected time based on the deadline time, the current time, and the execution time of the task. Therefore, our discussion below will concentrate on the global scheduling portion of the algorithm.

4.2 Classification of Dynamic Scheduling Algorithms on Distributed Systems

In this subsection, we classify dynamic scheduling algorithms in distributed systems. As regards the scheduling scheme, we specified that our system would use dynamic, symmetric, sender-initiated load distribution in a distributed system.

We propose to classify dynamic scheduling algorithms on the basis of three functions: qualification, determination, and reservation.

(1) Qualification is a procedure for deciding whether a node can execute a task by a deadline. Multiple qualified nodes are possible. A node investigated by a qualification procedure is called an objective node.

(2) Determination is a procedure for selecting one of the qualified nodes to which the task will be distributed.

(3) Reservation is a procedure for reserving hardware resources in some qualified nodes, and is explained in more detail in the next subsection.

We will classify scheduling algorithms according to the node in which "qualification" or "determination" is executed.

Qualification is also categorized into two types: centralized and distributed. In centralized qualification, load information is gathered into a sender node, and qualification is executed in the sender node. In distributed qualification, task information is distributed to objective nodes, and qualification is executed in the objective nodes. Centralized qualification seems to be easy to implement. However, in distributed qualification, the sender's load for distribution would be lighter than in centralized distribution, because the qualification procedure is executed concurrently.

Determination is also categorized into two types: centralized and distributed. In centralized determination, qualification information is gathered into a sender node, and determination is executed in the sender node. In distributed determination, determination is executed at each qualified node. Distributed determination is too complicated to implement, and we will therefore not discuss it further. Instead, we will now concentrate on centralized determination. Both centralized and distributed qualification algorithms have their own respective merits.

4.3 Reservation Mechanism

In distributed systems, communication delay should be considered. All arriving information may be different from that of the current states. All node states are changing concurrently. When a node (for example, node-A) succeeds in acquiring state information from another node (for example, node-B), the current state of node-B may have already changed on account of a communication delay. In the worst case, the task may be rejected by a selected node because of a state change during a communication delay. The sender then has to find another node, which may also reject the task. The task can thus be thought of as being sent around from one node to another.

Because of the above characteristic of distributed systems, reservation can be effective. Reservation means reserving some resources (e.g., CPU time) to execute a task on a node before distribution of the task. The timing of reservation in centralized qualification algorithms is different from that in distributed qualification algorithms. In our system, the reservation in centralized qualification is expected to be performed at the same time as acquisition of information on the load of each node, while the reservation in distributed qualification is expected to be done just after qualification at each node.

An advantage of these reservation mechanisms is that a task is guaranteed to be accepted in a selected node, if it arrives before the scheduled time. Two drawbacks of the reservation mechanism are the time overhead for the reservation and the possibility of wasted resource, since reserved resources are wasted if they are not used. To avoid this, a reservation release mechanism may be required. However, it may make the whole mechanism more complicated.

Our interest is to determine whether the benefits of the reservation mechanism outweigh the cost of the complication.

4.4 Four Dynamic Scheduling Algorithms

We propose four dynamic scheduling algorithms, shown in **Table 2**, where their names are abbreviated to two letters, of which the first identifies the qualification class, and the second existence or nonexistence of reservation.

- Qualification class: C stands for centralized. D stands for distributed.

Table 2 Scheduling classification.

	Qualification	
	Centralized	Distributed
Reservation	CR	DR
No reservation	CN	DN

- Reservation: R denotes the existence of reservation. N denotes the non-existence of reservation.

5. Evaluation Using Simulation

A simulation study of the above four algorithms was conducted, using several sets of parameters. This section describes the simulation model, the parameters, the workload model, and the criteria for the evaluation. The simulator was built by using a discrete simulator called SES/Workbench[14].

5.1 Simulation Model

The simulation model for the simulator is described below.

(1) There is a time frame within which execution of all tasks should be completed. This time frame is continuously repeated.

(2) The requirements for all tasks are known at the beginning of the time frame.

(3) The time frame is divided into three parts. In the first part, every node checks whether the tasks in the node can be executed within the time frame. In the second part, every node tries to find another node that can execute its overload of tasks. This part needs a dynamic scheduling algorithm, and is a main theme of this paper. In the last part, every node executes its allotted tasks.

(4) A preemptive scheduling algorithm is assumed, and we use a round-robin algorithm.

(5) The configuration of an input node is also specified. An input node serves both for computation and as an entrance for information from one or more sensors. For example, we adopted four configurations as alternative design choices in the initial phase of system development.

- One input node in every four nodes (1 : 4)
  There is just one input node in every four successive nodes in a ring network. In other word, there are three computation nodes between every two input nodes.
- One input node in every two nodes (1 : 2)
  Every second node is an input node.
- All nodes serve as equivalent input nodes (1 : 1)
  Every node is an input node.
- Only two input nodes in the entire system (only-2-node)
  There are only two input nodes in the system.

We adopted these four system configurations for two reasons: to allow a choice as regards the number of input nodes in the initial phase of our system development, and to reflect the characteristic of the simulator that hypotheses (i.e., tasks) in MHT generate new hypotheses (tasks). Namely, when the system configuration is set at 1 : 1, it corresponds to a simulation in which new tasks are generated equally in all system nodes. When the input node configuration is set at 1 : 4, it corresponds to generation of a new task at one in every four nodes.

(6) We evaluate the response characteristics of the system.

5.2 Parameters

A list of the main parameters is shown in **Table 3**. Since we have a prototype machine based on an SCI ring, we assumed that some parameters were the same as those of the real machine. We define all of the times in detail, and therefore there are no non-deterministic elements in the simulator. In other words, the estimated time in the simulator is always fixed.

The number of nodes in the evaluated system is also specified. We evaluated 4, 8, 16, 32 and 64 nodes in the simulator.

5.3 Workload Model

Table 4 shows the workload model. We used four conditions related to the ratio between the number of tasks already existing in the system and the number of tasks newly input. The ratio represents the CPU time consumption. For example, 50% means that half of the CPU time in the time frame will be used for the tasks. We assigned two priorities for tasks: high and low.

First, in Condition 0, the system is empty in the initial state. High-priority tasks whose size is 100% of the system capacity are then input. The expected final state is 100% high-priority tasks. The scheduling completion time (i.e., the time when the scheduling algorithm finishes the load-balancing procedure) is calculated.

Second, in Condition 1, the system is filled to capacity with low-priority in the initial state. High-priority tasks whose size is 50% of the system capacity are then input, and the scheduling completion time is calculated. In this condition, the scenario is that half of the existing tasks will be purged and replaced with newly input high-priority tasks. Since 50% of the system will be filled with high-priority tasks and

Table 3 Parameter list.

Categories	Parameters	Time (μs)
Communication delay	SCI-input	0.1
	SCI-bypass	0.7
	SCI-output	1.4
CPU execution time	Access to local memory	10
	Get-lock, Release-lock	10
	Create message	10
	Create task	10
	Judgment of guarantee	20
Round-robin	Quantum	1000
	Overhead time of exchange	7.2
Task	Time frame	32000
	Average execution time	8000

Table 4 Workload model.

	Initial State		Newly Input		Final State	
Priority	High	Low	High	Low	High	Low
Cond.0	0%	0%	100%	0%	100%	0%
Cond.1	0%	100%	50%	0%	50%	50%
Cond.2	50%	50%	50%	0%	100%	0%
Cond.3	0%	100%	100%	0%	100%	0%

50% with low-priority tasks, it is expected to be easier in this condition than in Conditions 2 and 3, described below, to look for qualified candidate nodes for task distribution.

Third, in Condition 2, the system is filled with low-priority and high-priority tasks (50% each) in the initial state. High-priority tasks whose size is 50% of the system capacity are then input, and the scheduling completion time is calculated. The scenario in this condition is that low-priority tasks existing in the system will be replaced by newly added high-priority tasks. The system is expected to execute only high-priority tasks. In this situation, it is expected to be more difficult and to take longer than in Condition 1 to look for qualified candidate nodes for task distribution.

Finally, in Condition 3, the system is filled with low-priority tasks in the initial state. High-priority tasks whose size is 100% of the system capacity are then input, and the scheduled completion time is calculated. The scenario in this condition is that all low-priority tasks existing in the system will be purged and replaced by newly added high-priority tasks. It is also expected to be more difficult and to longer in this condition than in Conditions 0, 1, and 2 to look for qualified candidate nodes for task distribution. Moreover, since 100% of low-priority tasks are to be purged, the total CPU load for the scheduling procedure is expected to be highest of these for the four conditions.

5.4 Criteria

Generally, the performance of real-time systems is calculated as the percentage of tasks completed by the deadline, and is known as the "guarantee ratio." We use the guarantee ratio as the first criterion for evaluating the performance. In addition, we will use two other criteria. The first is how quickly each scheduling algorithm completes its load balancing procedure, and the second is how often high-priority tasks are rejected.

6. Results and Discussion

In this section, we will compare the characteristics of the four scheduling algorithms, which are derived from the simulation results.

Figure 2 shows the scheduling completion time, plotted on the vertical axis, in relation to the four conditions described above, plotted on the horizontal axis. The number of nodes is fixed at 16. The input node configuration is set at 1 : 4. Before performing this simulation, we predicted that algorithms with a reservation mechanism might take longer to complete scheduling, because they generate a high volume of network traffic. We also predicted that algorithms without a reservation mechanism might take longer to locate a node that can finally guarantee and accept tasks that have been sent from one node to another.

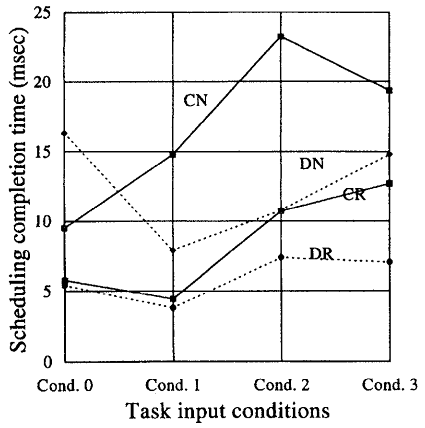


Fig. 2 Completion time (16-node system).

The result shown in Fig. 2 indicates that algorithms with a reservation mechanism perform better than those without such a mechanism. Furthermore, in the same situation with and without a reservation mechanism, it is shown that distributed qualification mechanisms perform better than centralized qualification ones. The DR algorithm, which has both of the features described here, achieves the best performance.

This pattern was always observed in similar figures when the values of the number of nodes or input node configuration were changed, within the range shown in Section 5.1 (5).

In term of the various conditions, it takes longer in Condition 3 than the others, except for CN algorithms. This result is reasonable because, from Table 4, all of the low-priority tasks are replaced by newly input high-priority tasks and the load is the highest in four conditions.

In real-time systems, it is important that scheduling be done by the deadline. If the scheduling completion time is long, it will not be possible to execute some tasks by the deadline, especially in systems with large numbers of nodes. In fact, in a 32-node system and a 64-node system, tasks whose deadline was not met were observed under the same conditions and input node configuration as described above. This phenomenon was observed only in CN and DN algorithms, whose scheduling completion time was longer, as shown in Fig. 2. Neither CN nor DN algorithms have a reservation mechanism. Since it is important that high-priority tasks should be completed by their deadline in real-time systems, a short scheduling completion time should be considered important.

Table 5 Rejection by receiver candidate [on Condtion 0].

Rejection	CR	CN	DR	DN
0	100.0%	75.0%	100.0%	45.3%
1	0.0%	21.9%	0.0%	29.7%
2	0.0%	3.1%	0.0%	15.6%
3	0.0%	0.0%	0.0%	6.3%

To analyze why non-reservation algorithms have a long scheduling completion time, we measured the frequency of rejection at a receiver candidate node and made histograms based on the ratio of the number of tasks rejected at a receiver candidate node to the total number of input tasks. (The term "reject" means that the task was not accepted by the candidate node because the node status had changed and the node could not afford to execute the task.)

Table 5 shows the results observed in Condition 0. The rejection count in the table represents the number of times that a task was rejected at a receiver candidate node. As a result, it affects the scheduling completion time. From Fig. 2 and Table 5, we were able to understand the importance of the reservation mechanism and why it results in a lower overhead time. Since the CR and the DR algorithms have reservation mechanisms, the rejection count at receiver candidate nodes is zero. The CN and the DN algorithms, however, do not have such a mechanism, and 75% of tasks in the CN algorithm and 45.3% of tasks in the DN algorithm were not rejected in receiver candidate nodes. Other tasks (25% in CN and 54.7% in DN) were rejected at least once at receiver candidate nodes. In addition, 25% of tasks in the DN algorithm were rejected more than twice at receiver candidate nodes.

Figure 3 shows the scalability of the system with the CR, CN, DR, and DN algorithms. We fixed the input node configuration at 1 : 4. and adopted Condition 3.

From the results described above, for systems of all scales from 4 to 64 nodes, we consider the DR algorithm to be the best of the four algorithms discussed in this paper. The reasons are as follows: (1) the reservation mechanism prevents a scheduling from generating re-try sessions, and effectively reduces the network load,

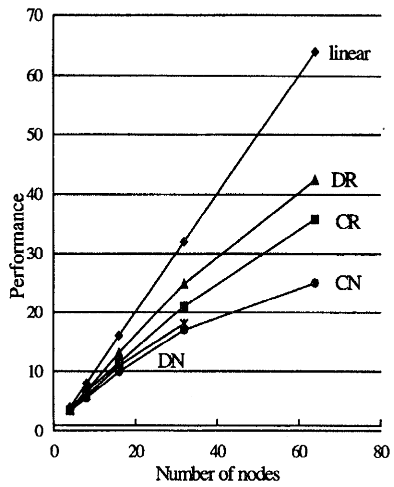


Fig. 3 Scalability.

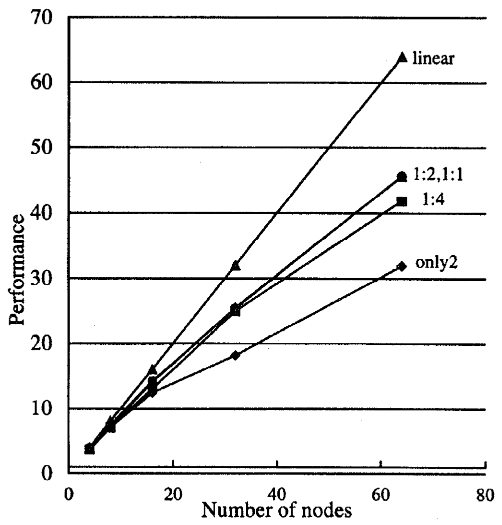


Fig. 4 Effect of Input Node Configuration on DR Algorithm.

and (2) concurrence of distributed qualification reduces the completion time.

The DR algorithm, which is considered the best so far, was examined further. **Figure 4** shows the scalability of the DR algorithm for input node configurations of 1 : 1, 1 : 2, 1 : 4, and only-2-node. Condition 2 is adopted here. Two curves, for the input node configuration of 1 : 1 and 1 : 2, are almost overlapped.

The performance of the only-2-node input node configuration is inferior to the others. It is thought that serialized processing at two input nodes bottlenecked the system.

It is considered effective to adopt an 8- or 16-node system using the DR algorithm, if the input node configuration of 1 : 4 is adopted. The input node configuration of 1 : 4 is a highly probable configuration from the point of view of a multi-sensor. In other words, it is not effective to use more than 32 nodes in a system under the above four conditions.

7. Conclusion

No systematic design methodology has been generally established for the dynamic scheduling algorithms used in distributed systems.

This paper proposes three classification axes (centralized/distributed qualification, centralized/distributed determination, and existence/absence of reservation) which are thought to be effective for designing optimal dynamic scheduling algorithms under certain conditions and parameters. We also evaluated algorithms in the categories outlined by those axes by means of simulation under certain conditions.

With regard to our target system for implementing Multiple Hypothesis Tracking (MHT), we obtained the following results with our current hardware, OS, and application parameters:

- Qualification: distributed qualification is superior to centralized qualification.
- Reservation: a reservation mechanism is superior to a non-reservation mechanism.

From the results described above, the following design decisions are suggested: It is suitable for our real-time system to use a dynamic scheduling algorithm that executes qualification in a distributed scheme and provides a reservation mechanism.

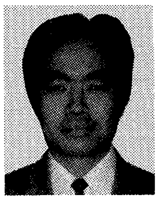
Our future plan is to verify our simulation results with a real machine, which we are currently developing. In addition, we are going to continue to study ways of improving the distributed qualification algorithm with a reservation mechanism.

References

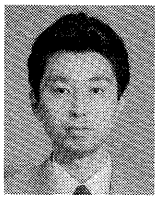
1) Tomita, S.: *Parallel Computer Architectures*, Shoko Dou (1986).
2) Okugawa, S.: *Parallel Computer Architecture*, Corona Publishing (1991).
3) Casavant, T.L. and Kuhl, J.G.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, *IEEE Trans. Softw. Eng.*, Vol.14, No.2, pp.141–154 (1988).
4) Wang, Y.T. and Morris, R.J.T.: Load Sharing in Distributed Systems, *IEEE Trans. Comput.*, Vol.c-34, No.3, pp.204–217 (1985).

5) Stankovic, J.A., Ramaritham, K. and Cheng S.: Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems, *Tutorial Hard Real-Time Systems*, pp.273–286 (1988).
6) Ma, R.P., Lee, E.Y.S. and Tsuchiya, M.: A Task Allocation Model for Distributed Computing Systems, *IEEE Trans. Comput.*, Vol.c-31, No.1, pp.41–47 (1982).
7) Ramamoorthy, C.V., Chandy, M.K. and Gonzalez, M.J.: Optimal Scheduling Strategies in a Multiprocessor System, *IEEE Trans. Comput.*, Vol.c-21, No.2, pp.137–146 (1972).
8) IEEE Standard for Scalable Coherent Interface (SCI), IEEE Computer Society, IEEE Std., 1596–1992 (1992).
9) Reid, D.B.: An Algorithm for Tracking Multiple Targets, *IEEE Trans. Automatic Control*, Vol.AC-24, No.6, pp.843–854 (1979).
10) Miller, M.L.: *Implementation Notes for MHT With Multiple Target Models*, NEC Research Institute (1993).
11) Takahashi, M., Aoyama, K., Miyata, H. and Kan, T.: A Performance Comparison of several Network Topologies Composed of Scalable Coherent Interface, *Proc. 50th Domestic Conference of IPSJ*, 6, Hardware-Systems, pp.6-25–6-26, in Japanese (1995).
12) Takahashi, M., Aoyama, K., Miyata, H. and Kan, T.: An Evaluation of Network Topologies Using SCI Interface, SWoPP-95, Eighth IPSJ Summer United Workshops on Parallel, Distributed, and Cooperative Processing, TR 95ARC-113, Vol.95, No.80, pp.73–80, in Japanese (1995).
13) Takahashi, M., Aoyama, K., Takano, H. and Miyata, H.: A Simulation Study of Dynamic Scheduling Algorithms of Real-Time Signal Processing on a Multi-Processor Systems, Hokke-96, Third IPSJ Workshop of High Performance Computing and Evaluation '96, TR 96-ARC-117, Vol.96, pp.13–18, in Japanese (1996).
14) SES/Workbench Reference Manual Release 3.0, SES, Austin, TX (1994).

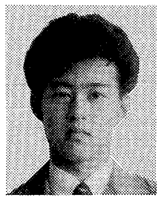
(Received March 24, 1997)
(Accepted April 3, 1998)



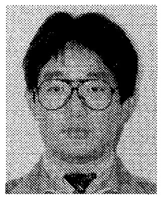
Hiroyuki Miyata was born in Nagoya Japan in 1957. He received M.S. degree in 1982, and Ph.D. degree in 1998 in Information Science from Kyoto University. He joined Computer and Information Systems Laboratory of Mitsubishi Electric Corporation in 1982. He was engaged in research and development of massively parallel processors. From 1990 to 1991, he was also a vising scholar at University of Illinois at Urbana-Champaign. He is a senior engineer for work on parallel computing systems in Kamakura Works of Mitsubishi Electric Corporation. He is a member of IEEE.



Masato Takahashi received M.S.degree in Coordinated Sciences from University of Tokyo in 1990. He joined Computer and Information Systems Laboratory of Mitsubishi Electric Corporation in 1990. He was engaged in research of multiprocessor systems. From 1996, he joined Communications Research Laboratory of Ministry of Posts and Telecommunications. He is a senior researcher on satellite communications. He is a member of IEEE.



Hiroyuki Takano was born in 1967. He graduated in Mechanical Engineering from Tokyo National College of Technology. He joined Mitsubishi Electric Corporation in 1988. He has a great interest in parallel processing of tracking radar system. He is engaged in development of avionics software at Kamakura Works of Mitsubishi Electric Corporation since 1988.



Kazuhiro Aoyama is an Engineer at Kamakura Works, Mitsubishi Electric Corporation. His research interests are real-time computing and multiprocessor systems. He received his B.Eng. and Ms.Eng. in instrumentation engineering from Keio University in 1983 and 1985 respectively, and his M.S. in computer science from the University of Illinois at Urbana Champaign in 1993.