TR-0755

Hardware Implementation of Dynamic Load Balancing in the Parallel Inference Machine PIM/c

by
T. Nakagawa, N. Ido, T. Tarui, M. Asaie
& M. Sugie (Hitachi)

March, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
(03)3456-3191～5
Telex ICOT J32964
Minato-ku Tokyo 108 Japan

Institute for New Generation Computer Technology

Hardware Implementation of Dynamic Load Balancing in the Parallel Inference Machine PIM/c

T. NAKAGAWA, N. IDO, T. TARUI, M. ASAIE and M. SUGIE
Central Research Laboratory, Hitachi Ltd.
Higashi-Koigakubo, Kokubunji, Tokyo 185, Japan

ABSTRACT

This paper proposes and evaluates the hardware implementation required for dynamic load balancing in the prototype PIM/c of the Parallel Inference Machine (PIM). In fine grain multiprocessing, dynamic load balancing is suffering from the high overhead due to the frequent access to load information. Proposed hardware can reduce the overhead by speeding up the access to the load information. In order to utilize the high locality of logic programs, PIM/c is configured along a hierarchical structure of network-connected clusters each of which is a bus-connected multiprocessor. Therefore two kinds of hardware suitable for each hierarchy are implemented for dynamic load balancing.

First, in the clusters, we propose a register with broadcast write feature. The evaluation determines the reduction of overhead due to memory polling which detects a load request. The proposed hardware reduces the execution time of logic programs by 15%.

Second, in the network, we propose the use of a shortcut path to request the value of the total load within a cluster. The evaluation shows that the overhead due to the request of that value is reduced as a result of introducing the shortcut path. The proposed hardware reduces the execution time by 50%.

The results obtained confirm that the use of hardware can reduce the high overhead of dynamic load balancing.

1. INTRODUCTION

Japan's Fifth Generation Computer project [1] has been centered around ICOT (the Institute for new generation COmputer Technology). ICOT has developed the parallel logic programming language KL1 (Kernel Language-1) [2] to describe knowledge and information processing systems. ICOT has also produced software in KL1, including the PIM operating system [3].

We are currently developing the PIM/c [4] as a KL1-based machine. A hierarchical structure of network-connected clusters each of which is a bus-connected multiprocessor is introduced to utilize high access locality of KL1 programs in PIM [5]. Use of locality could restrict the interactions to clusters of several processors and thus reduce the communications among clusters. Therefore, a double hierarchical organization is used in PIM/c.

Dynamic load balancing is one of the main research areas for PIM. As a result of the fact that logical relations are present in a KL1 program and they never define their process of execution with determinacy, dynamic load balancing must be used. For dynamic load balancing it is necessary to require load information, for example, the information about the existence of idle processors or the value of a total load within a cluster. The load information is updated and referenced by distributed processors. In other words the load information is global, therefore it has no locality.

A problem exists in that hardware for normal process execution in PIM/c is optimized to the access with locality. With this type of hardware the latency in accessing global information is large. In fine grain multiprocessing in KL1 programs, high frequency and large latency in accessing load information produces high overhead. Therefore, extensions in hardware are introduced in order to reduce the latency of load information in PIM/c.

In shared bus multiprocessors, snooping caches are known to reduce the memory latency observed by the processors [6,9]. There are two types of cache coherency protocols for rewriting shared data with

copies distributed in plural caches; invalidation-type protocols and broadcast-type protocols. The choice depends on whether it is preferable to invalidate old copies for rewriting by the same processor, or to broadcast the new data for rewriting by other processors.

Eggers [7] defined "per processor locality" as the average number of repeated write references to the same address by the same processor. For normal process execution in the KL1 system, an incremental garbage collection makes the same processor reuse the same address repeatedly for different data references [4]. Thus invalidation protocols are more suitable due to high "per processor locality".

For dynamic load balancing, broadcast protocols are preferable in order to access load information efficiently. Although protocols using both invalidation and broadcast features are known as "competitive snooping protocols [8]", the cache is insufficient to reduce the latency in accessing load information within the cluster of bus-connected multiprocessors. Thus the snooping cache in PIM/c utilizes an invalidation protocol and the implementation of broadcast feature is also considered, not for cache, but for registers to reduce the latency more efficiently.

In network-based multiprocessors, for normal process execution, it is more important to increase the throughput than to reduce the latency because the "non-busy-waiting" feature could overcome the large latency [4]. The PIM/c network unit has message queues to increase the throughput, although they produce an increase in latency. For dynamic load balancing, use of the old information may cause wasteful load dispatching. Therefore, a shortcut path to the message queues is introduced to reduce the latency in accessing load information through the network of PIM/c.

Hardware extensions in PIM/c require only a small amount of hardware because the addressable space for broadcasting is limited in the shared bus, and because the increase in the number of interconnections among clusters is less than that of a system with a special purpose network [10].

2. PIM/c HARDWARE FEATURES

PIM/c has the following hardware features:

A. Hierarchical structure of shared bus multiprocessor and network based multiprocessor.

Figure 1 shows the configuration of PIM/c. It is organized along a hierarchical structure of network-connected clusters to utilize the localities of KL1 programs. Thus, the shared bus hierarchy consists of processors combined in a cluster. Each processor has its own cache, and they share a common bus. Software simulation has proved that the common bus might be a bottle-neck. We concluded that the number of processors within a cluster should be limited to around eight, and that a two-way-interleaved common bus [11] should be possible in PIM/c.

We consider that utilizing the access locality makes it possible to reduce the amount of network hardware because of reducing the number of messages transferred among clusters. As a consequence, in PIM/c the network is connected only to cluster controllers (CC) instead of all processors in the cluster.

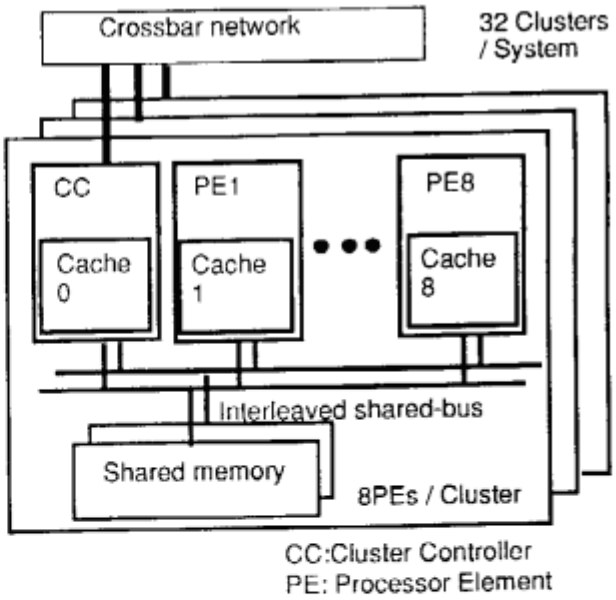

Fig. 1. The configuration of PIM/c. Each cache has a capacity of 80 Kbytes and consists of 20 byte blocks.

B. Broadcast registers in the shared bus hierarchy.

In order to reduce the access latency of load information in the shared bus hierarchy, registers with broadcast feature are introduced in PIM (Fig. 2) [12].

We denote these registers as EFR's (Event Flag Register). They have the following features:

- one-bit wide to indicate an event, and a fast detection feature for control jumps which checks the existence of events.
- feature of broadcast write; therefore, registers indicating the same request event to any processor can be written simultaneously.

The reference and jump can be done within a cycle. When using registers, there is no overhead due to cache misses. Each PIM/c processor has 16 EFRs.

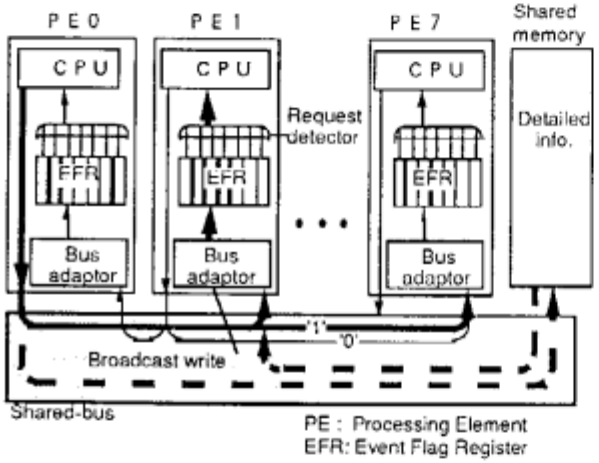

Fig. 2. Broadcast registers in the cluster. Bold lines show the propagation path of a request event to broadcast registers and the broken lines show the memory polling path without hardware support. The thin lines show the reset action of that event.

C. Shortcut path in the network hierarchy.

In order to reduce the access latency of load information in the network hierarchy, two kinds of features are introduced; a shortcut path for the specific messages (Fig. 3) [13] and the registers that hold the load information are called CIR's (Cluster Information Register). The hardware has the following features:

- a shortcut path to message queues.
- eight-bit wide registers to indicate load information in a corresponding cluster.

The register should be written with the load information by its corresponding cluster controller.

As the load information is required without waiting at message queues and without waiting for the cluster controllers to receive, specified registers can always be read in 11 cycles.

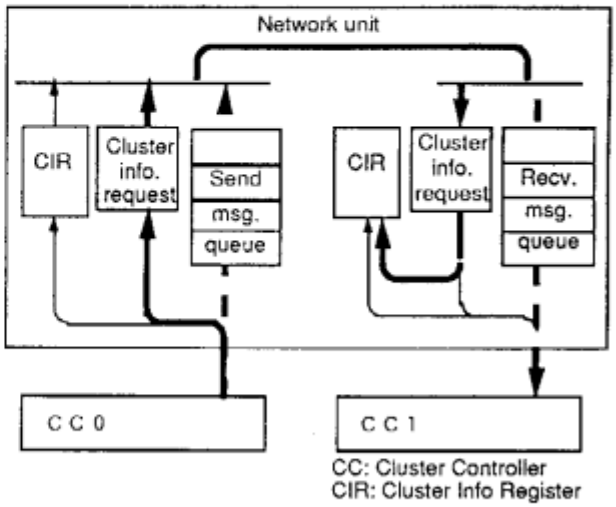

Fig. 3. Shortcut path in the network. The shortcut paths and the registers exist in the router board of the packet switching network. Broken lines show the normal path through the message queues to increase the network throughput and the bold lines show the shortcut path to bypass the queues.

3. EVALUATION STRATEGY

We defined the following two strategies to evaluate the effectiveness of the proposed load balancing hardware.

3.1 Evaluation on the Real Hardware

Real hardware was used for evaluation as the software simulation is almost impossible for the following reasons:

- The presence of the cache and the network introduce more parameters.

  There are many hardware parameters related to the internal states of the cache and the network. The common bus arbitration time, and the message packet switching time are examples. The overhead of cache misses and the network latency is important

in this evaluation. Thus, simulating the cache and network effects concurrently with processor activities would have taken a great deal of time in software simulation.

3.2 Evaluation using an Artificial Load Model

With an aim toward further improvement, we evaluated an artificial load model for the following reasons:

- to separate the effect of hardware alone.
  An evaluation independent of the specific application is necessary in order to isolate the speedup produced by the proposed hardware mechanisms.
- to separate the effect of load balancing.
  The real KL1 execution environment involves many new control sequences in addition to load balancing. For example, handling the priority of loads needs another polling action using EFR registers. The total performance depends on the usage of the proposed hardware in other control sequences.

4. EVALUATION RESULTS

We carried out the evaluation of the proposed hardware in both shared bus and network-based hierarchies.

4.1 Evaluation of broadcast registers in the shared-bus hierarchy

We carried out this evaluation by focusing on the reduction of the latency to access the information about the existence of the idle processors.

A. The load balancing scheme.

The load balancing scheme is explained below:

- Distributed load pool.
  Each processor has its own load pool in order to avoid implicit data transfers between caches due to updating a serial link in case of the generator processor of the load differs from its consumer processor using common load pool [14]. Consequently, an explicit load balancing communication for the distributed load pools is required.
- Receiver-initiated load balancing.
  The explicit load balancing communication for the distributed load pools should be initiated by fully idle processors in order to avoid wasteful dispatching. Thus the communication is request based.
- Communication with arbitrary responder.
  In order to reduce the response time without interrupting busy processors, a new type of communication, the AR (Arbitrary Responder) communication is introduced in PIM/c [12]. The request is sent to any processor which has more than one load in its load pool. In order to avoid the high overhead of context switching, every processor polls the request at intervals where the context switch overhead is low. Thus any processor which detects the request first responds to it. As the timing to detect requests differs in each PIM/c processor, this communication method is expected to reduce the response time proportionally to the number of processors in a cluster.

B. The load model.

This model reflects the following characteristics of KL1 program execution:

- Unit load.
  We denote the unit as the *reduction*. The unit is assumed to be 200 cycles in PIM/c (Fig. 4).
- Indeterminacy in the granularity of loads.
  In order to simulate "Tail Recursion Optimization" [17], we define the *goal* as consisting of an arbitrary number of reductions (1 to 16).
- Indeterminacy in the number of goals.
  In order to simulate the indeterminacy, we assume that each processor generates an arbitrary number of goals (1 to 4096).
- A high write ratio and a high share ratio.
  Accesses performed within the reductions have the

following parameters: write ratio is 0.5, share ratio is 0.5, where write ratio is defined as the ratio of write references to total memory references, and share ratio is defined as the ratio of references to shared data area to total memory references.

- A high access locality.

  We define the locality as the number of successive accesses to the same address. The value is set to 4 in order to simulate free-list manipulation, which consists of allocating, instantiating, referring and deallocating a memory cell.

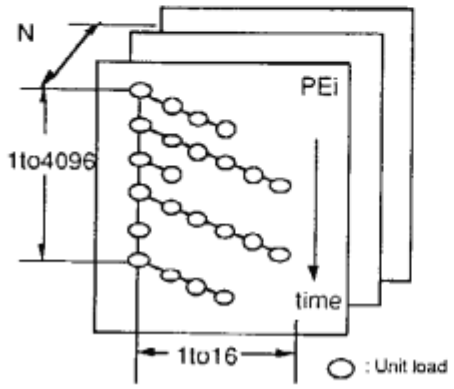

Fig. 4. A Load model with varying granularity.

C. Results of the evaluation in a cluster.

We control the initial load amount in each processor to vary load balancing conditions. According to the deviation of the initial load amounts within processors, 14 cases are simulated with an 8-processor cluster. The resulting data are the total elapsed time (T), the total idle time (I), the total wait time after requesting for load (i), the total dispatching time (t), the total reduction count (R) and the load request count (r). The total idle time includes the time spent waiting for load dispatching since requesting a load by updating a bit-map word until receiving a load by reading a non-zero value from its communication area, and the time to wait for termination of the whole program. The bit-map word is a data array in which each bit corresponds to a processor requesting load. The total dispatching time includes the time to select an idle processor by encoding the bit-map word to the address of its communication area, and the time to dispatch a load to each idle processor by updating their communication areas. The evaluation measures are i and t, and the reduction cost is defined as follows:

Reduction cost = (T - I - t) / R

Figure 5 shows the performance increase in reduction using registers. The total reduction cost and the load request count are varied in 14 simulation cases. In this figure, request ratio is introduced, which is defined as the ratio of the load request count r to the total reduction count R. The reduction cost is almost independent of the request ratio. This fact indicates that the memory-polling overhead caused by checking request occurrences is larger than the overhead due to cache misses using invalidation protocol. The speedup obtained is 15% due to the use of EFRs.

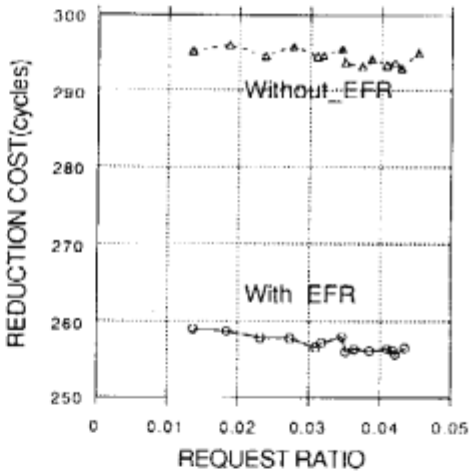

Fig. 5. The increase in speed using registers. The reduction cost is defined as the number of execution cycles per unit load. The result involves extra cycles for probing. The request ratio is defined as the number of request per reduction. Using memory polling the reduction cost is high due to the serial execution of a memory access and a branch. Using EFR, both the access and the branch can be done within a cycle. The polling is done for three kinds of events; load request, load dispatching and termination of the whole program.

Figure 6 shows the wait time i and the dispatching time t as a function of request count. It is confirmed that the use of EFR with broadcast feature reduces both the wait time and dispatching time. The use of EFR reduces the dispatching time by 20%, and reduces the wait time by 15%.

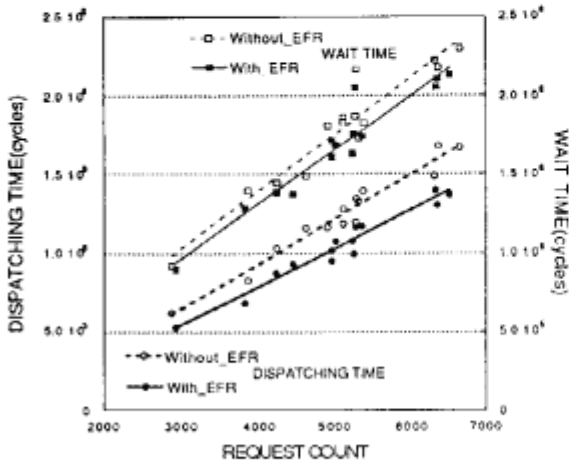

Fig. 6. The increase in speed using broadcasting. The dispatching time and the wait time increase due to the cache misses using an invalidation-type snooping cache. The use of broadcast feature eliminates the overhead due to the cache misses.

4.2. Evaluation of shortcut paths in the network-based hierarchy

We carried out this evaluation by focusing on the reduction of the latency in accessing the value of the total load in a cluster.

A. The load balancing scheme.

The load balancing scheme is described below:

- Sender-initiated load balancing.

  A study of the Multi-PSI system disclosed a problem of the receiver-initiated load balancing scheme in large-scale machines, namely that a load request contention may arise at busy processors [15]. In order to avoid this contention, an improved sender-initiated scheme, named "Smart Random Load Dispatching" [5] is efficient in reducing wasteful dispatching. In this scheme, the cluster to which goals are dispatched is determined at random and then this goal dispatch is aborted on the condition that the dispatch target has more loads in the pool than the dispatching cluster.

B. The Load model.

The load model among clusters is defined in such a way as to reflect the changes in the amount of loads in the load pool. The load model is as follows:

- An initial goal is denoted by L(16) (Fig. 7 shows L(5)).
- The execution of goal L(i) produces (i-1) subgoals, L(i-1),..., L(2), L(1). Thus, the goal L(i) has 2^{i-1} reductions.
- Each reduction takes 300 cycles to execute using network messages.
- The message length required for the load dispatching is 27 bytes long. Thus, it takes 27 cycles to send this message through the one-byte-wide network interface. The length of the message requesting the load amount is 2 bytes.

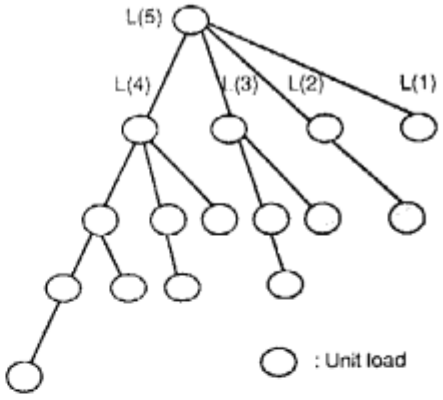

Fig. 7. A load model with floating amount of load.

C. Results of the evaluation among clusters.

We control the dispatching rate, which is defined as the ratio of all goals dispatched to other clusters to all executed goals, by changing the interval of the dispatching control. In order to determine the efficiency of load dispatching, the total elapsed time (T), the total idle time (I) and the dispatching rate (d) are measured. Differences result from the latency of load information.

Figure 8 shows the results obtained by applying the smart random load dispatching scheme to 8 cluster system without support hardware. The normalized elapsed time, which is defined as the ratio of elapsed time by 8 cluster system to elapsed time by single cluster, and the utilization of processors are plotted as a function of the dispatching rate. In order to compare the results in the two cases, we assume that the dispatching rate is controlled to be 0.2, because safe control occurs only at the upper side of the minimum point. Without the support hardware, the resulting increase in speed is approximately 3.3 in an 8-cluster system at a dispatching rate of 0.2.

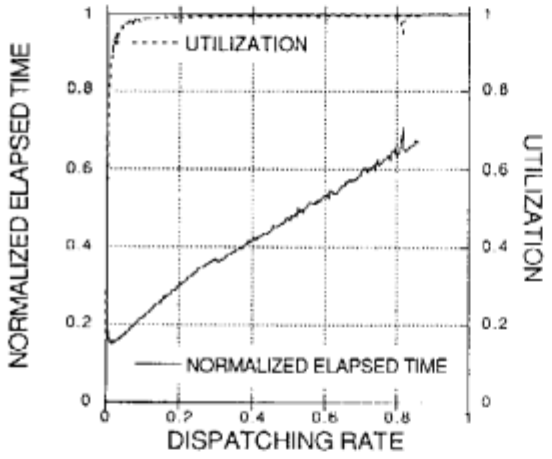

Fig. 8. Smart random dispatching without support hardware. The dispatching rate is defined as the ratio of all goals dispatched to other clusters to all executed goals. The normalized elapsed time varies considerably from 0.125 using 8 clusters connected via a network because the overhead for message handling is visible.

Figure 9 shows the results after applying the smart random load dispatching scheme with hardware support. The normalized elapsed time and the utilization of processors are plotted as a function of the dispatching rate. With the support hardware active, the processor can reduce the overhead due to requesting the load amount. The resulting increase in speed is approximately 5.5 in an 8-cluster system at a dispatching rate of 0.2.

Comparing the two results, the use of the proposed hardware halves the normalized elapsed time at 0.2 dispatching rate, where the control of dispatching rate seems to be possible.

It should be noted that the shortcut path can also be used for other load balancing schemes, including the minimum load distribution scheme [16]. These schemes will be evaluated in future work.

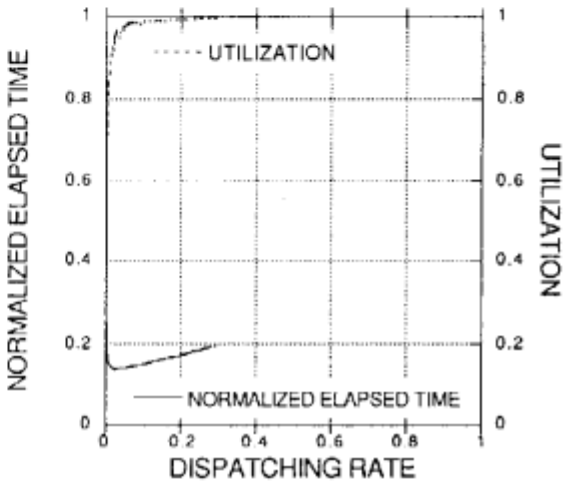

Fig. 9. Smart random dispatching with support hardware. The normalized elapsed time varies near 0.125 using 8 clusters connected via a network because the overhead for message handling is quite low.

5. CONCLUSION

Hardware for dynamic load balancing is implemented in both shared-bus and network-based multiprocessors.

We propose a register with broadcast write feature in shared-bus multiprocessors. Also, in network-based multiprocessors, the network unit uses a shortcut path. The evaluation was carried out using real hardware and an artificial load model.

The evaluation results in the shared bus hierarchy determine the overhead due to memory polling which detects a load request. The proposed hardware reduces

the execution time of logic programs by 15%.

The evaluation results in the network-based hierarchy show that the overhead due to requesting the load amount is reduced as a result of introducing the shortcut path. The proposed hardware reduces the execution time by 50%.

It is confirmed that the proposed hardware reduces the access latency of load information, and subsequently the overhead produced by dynamic load balancing.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Shun'ichi Uchida, the manager of the research department of ICOT, for his guidance and support, Dr. Kazuo Taki, chief of 1st ICOT laboratory, and Mr. Marius Hancu for helpful discussions. This research was sponsored by ICOT.

REFERENCES

[1] K. Fuchi and K. Furukawa, "The role of logic programming in the fifth generation computer project," Springer-Verlag, 1987, 1(5) pp 3-28.

[2] K. Ueda , "Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard," TR208, ICOT, 1986.

[3] T. Chikayama, H. Sato, T. Miyazaki, "Overview of the Parallel Inference Machine Operating System (PIMOS)," Proc. of the FGCS, vol.1, 1988.

[4] A. Goto, M. Sato, K. Nakajima, K. Taki, A. Matsumoto, "Overview of the Parallel Inference Machine Architecture (PIM)," Proc. of the FGCS, vol.1, 1988, pp 208-229.

[5] M. Sugie, M. Yoneyama, N. Ido, T. Tarui, "Load Dispatching Strategy on Parallel Inference Machines," Proc. of FGCS, Vol.3, 1988.

[6] J. Archibald and J. Baer, "Cache Coherence Protocols: Evaluation using a Multiprocessor Simulation Model," ACM Trans. on Comp. Systems, Vol.4, No.4, 1986, pp 273-298.

[7] S. J. Eggers and R. H. Katz, "Evaluating the Performance of four Snooping Cache Coherency Protocols," Proc. of the 16th ISCA, 1989.

[8] A. R. Karlin, M.S. Manasse, L. Rudolph and D. D. Sleator, "Competitive Snoopy Caching," Proc. of the 27th Annual Symposium on Foundation of Computer Science, Toronto, October, 1986.

[9] A. Gupta and J. Hennessy, "Comparable Evaluation of Latency Reducing and Tolerating Techniques," Proc. of the 18th ISCA, IEEE, 1991.

[10] H. Koike and H. Tanaka, "Multi Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64," Proc. of FGCS, Vol.3, 1988.

[11] L. Rudolph and Z. Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," Proc. of the 11th ISCA, June, 1984.

[12] T. Nakagawa, A. Goto, T. Chikayama, "Slit-Check Features to Speedup Interprocessor Software Interruption Handling," IEICE SIG Reports, July, 1989, pp 17-24, (in Japanese).

[13] N. Ido, H. Maeda, T. Tarui, T. Nakagawa, M. Sugie, "Parallel Inference Machine PIM/c -Load Balancing Support-," the 40th Annu. Convention IPS Japan, 2L-4, (in Japanese) .

[14] M. Sato and A. Goto, "Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor," Proc. of IFIP Working Conf. on Parallel Processing, Pisa, April, 1988.

[15] M. Furuichi, K. Taki, and N. Ichiyoshi, " A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI," In Proc. of the 2nd SIGPLAN Sympo. on Principles and Practice of Parallel Programming, pp 50-59, Mar. 1990.

[16] S. Sakai, H. Koike, H. Tanaka, T. Motooka, "Interconnection network with dynamic load balancing facility," Trans. of Information Processing, Vol. 27, No. 5, pp 518-524, 1986, (in Japanese).

[17] D. H. D. Warren, " An Improved Prolog Implementation which Optimises Tail Recursion," Research paper 156, Dept. of Artificial Intelligence, Univ. of Edinburgh, Scotland, 1980.