TR-352

Analysis of Parallel Inference Machines to Achieve Dynamic Load Balancing

by
S. Sugie, M. Yoneyama & A. Goto

March, 1988

ICOT
Mita Kokusai Bldg. 21F
4 28 Mita 1 Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Analysis of Parallel Inference Machines to Achieve Dynamic Load Balancing

M.Sugie, M.Yonezyama
Central Research Laboratory, Hitachi, Ltd.
Higashi-Koigakubo, Kokubunji, Tokyo 185, Japan
and
A. Goto
Institute for New Generation Computer Technology (ICOT)
Mita Kokusai Bldg. 21F, Mita, Minato-ku, Tokyo 108, Japan

Abstract-A parallel inference machine prototype modelled on the loosely-coupled clusters is simulated on a hardware simulator, using Queens and a kernel benchmarks. Utilization depends on the load status modification delay. It is confirmed that the load status modification delay should be less than half of the reduction time to limit the degradation to within 5 %.

INTRODUCTION

The Fifth Generation Computer Project has developed knowledge/information processing systems based on a predicate logic programming language [1],[2],[3]. The hardware of these system have been dubbed an "Inference Machine", since the predicate logic principle is inference. In the project's initial stage, not only sequential architectural inference machines were developed, but also various parallel architectural concepts were proposed and evaluated [4],[5],[6]. That project is now in the intermediate stage. A parallel inference machine (PIM) prototype composed of about 100 processing-elements is being designed for the target language KL1[7].

The main research themes of PIM are parallel processing overhead and processing-element utilization, since the same ideas can be applied to inference processing itself as have been developed for sequential inference machines. Both processing-element utilization and parallel processing overhead depend on load granularity. Generally, the utilization becomes larger as the granularity becomes finer. If a fine load granularity is designed, it will be easy to get high processing-element utilization, but difficult to reduce parallel processing overhead. Utilization depends on the load-balancing feature of parallel systems as well as the granularity and several load-balancing methods have been proposed [8],[9]. In those methods, load dispatch targets are determined dynamically by detecting the load distribution imbalance of processing-elements. Parallel logic programming languages such as KL1 have a suspend/resume processes feature for concurrent process control. This feature causes much parallel processing overhead. Therefore, the PIM prototype load granularity is of a coarse design. The load-balancing feature is an important research theme for improving the processing-element utilization.

This paper investigates load-balancing features of the PIM prototype, especially the time delay effect which exists between load status detection and modification. Once the processing-element with minimum load is determined, all processing-elements prepare to dispatch loads to that processing-element. In case there is some time delay between load status detection and modification, load concentration on one processing-element occurs. Performance of the PIM prototype is limited by suspension/resumption overhead in the fine granularity region and by low utilization, due to load distribution imbalance, in the coarse granularity region. Maximum performance is obtained between these two regions, but the time delay causes a greater performance degradation in this region. The time delay margin is discussed.

CONCEPT ON SYSTEM ORGANIZATION OF A PIM PROTOTYPE

Parallel architecture is a promising means for improving processing ability. However, to improve this ability efficiently, program localization of closely related sequences should be considered whenever possible.

KL1, the PIM prototype target language, has suspend/resume processes feature [10]. This feature makes it possible to express concurrent process control flow explicitly in programs, but causes much burden on inference machine processing ability. Therefore, occurrences of suspension/resumption should be reduced at the program execution time. In some cases, simple depth-first process activation scheduling can reduce occurrences of suspension/resumption. That is to say, when a process is activated, those causes which would suspend that process are eliminated by past process activation. Even in parallel

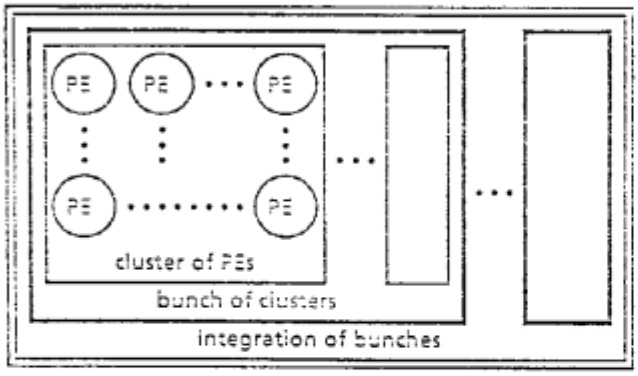

Fig. 1 Hierarchical structure for PIM

architecture's case, implementation of this sort of scheduling is important, since process suspension/resumption would be too great a parallel processing overhead.

In order to reduce ocurrencces of suspension/resumption, namely, parallel processing overhead, such a hierarchical structure as is shown in Fig. 1 is useful for the PIM prototype. Hardware/software investment in PIM prototype components should have the following priority order: processing-element (bottom-layer component) → cluster of tightly-coupled processing-elements (2nd-layer component) → bunch of loosely-coupled clusters (3rd-layer component) → integration of bunches (top layer).

The alternative is a uniform nonhierarchical configuration. An unequal-length network such as a mesh or hypercube can implement a uniform configuration with 100 processing-elements. In this case the only way to reduce suspension/resumption overhead using program localization is to make a group of neighboring processing-elements. To achieve high processing-element utilization, this group of neighboring elements must be dynamically modified to avoid assigning too much capacity for too little work. This would cause too great an overhead burden.

Current technology makes it possible to construct a PIM prototype with 2-layer hierarchy [7]. The bunch of loosely-coupled clusters would be the top layer. It could consist of about 10 clusters coupled loosely through some sort of equal-length network such as a crosshar. The cluster can consist of about 10 processing-elements coupled tightly through shared storage and caches.

SIMULATION

In the PIM prototype configuration, parallel processing overhead and processing-element utilization are much more significant in the bunch layer than in the cluster layer, because about 10 processing-elements are coupled tightly through shared storage and caches in the cluster layer. Inside the cluster, each processing-element could communicate with others with small overhead, and occurrences of suspension/resumption could be reduced by process activation scheduling using a common ready process queue stored in the shared storage. To understand the load-balancing features, the bunch layer of the PIM prototype has been simulated.

Simulator

Simulation has been made on the hardware simulator of PIM-R [11] on which an interpreter for KL1 is implemented.

Fig. 2 shows the hardware simulator organization. It is composed of 16 single board microcomputers (abbreviated as SBC) using MC68000, local storage, shared storage and Micro VAX II, which works as a supervisor. The shared storage, 16 IM Board Interfaces and GPIB Interface are connected to a common bus. The GPIB Interface handles the communication between Micro VAX II and the shared storage. but the IM Board Interfaces handle the communication between SBCs and the shared storage.

As for bunch layer simulation of the PIM prototype, the cluster of processing-elements is simulated by SBC and the network through which the clusters are connected is simulated by the shared storage. According to the purpose of this simulation, namely, bunch-layer simulation, detailed structure and operation inside the cluster is not simulated. On the simulation, SBC works like a single processing-element with high performance.

All SBCs have their own clocks and operate asynchronously in parallel. In the shared storage and the

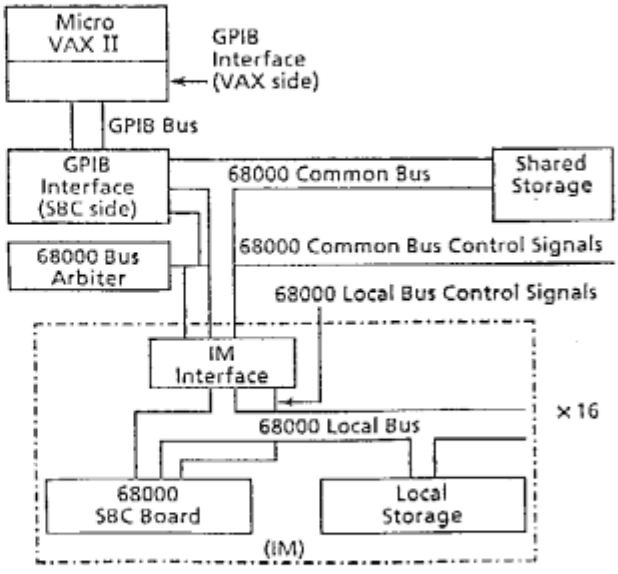

Fig. 2 Hardware block diagram of a simulator

local storage, dynamic RAMs, which need to be refreshed at intervals, are used. Therefore, this hardware simulator works with some uncertainty and realizes good parallel environment like a real system.

In this hardware simulator, the event-driven method is employed so as to eliminate the idling time during simulation. Concerning the timer, the simulator does not have a TOD (Time of Day Clock), which uniformly manages time over the whole system, but it does have a software timer in each cluster simulated by SBC. The timer count renews by adding a certain value every time a transaction of anyone of several functions is executed. When messages are sent to other clusters, network delay time is added on the timer count, and this value is attached to the sent message to indicate the arrival time. The cluster which receives the message controls the timer count by comparing this arrival time and its own timer when it accept the message. On the simulation, all data measurements and some operations such as queue controls are based on the cluster software timer. However, the rest of the operations are controlled by physical time, because of the event-driven method and real parallel operation.

Conditions

The simulation assumes the following:

(1) 16 clusters are coupled through a collision free, equal-length network with sufficiently large throughput.

(2) The cluster has a sufficiently large input/output buffer and waiting time, due to the input/output buffer overflow not being taken into account.

(3) The cluster's sending and receiving message overhead is 10 % of reductions in case of 4 clusters and the 4-Queens benchmark (adjusted by using parameters).

(4) OR-clauses are tried sequentially in head unification.

(5) A new goal is dispatched to clusters when AND-fork occurs in the clause body.

(6) Built-in predicates are not dispatched to other clusters.

Results

The following relationships on the PIM prototype composed of 16 clusters have been measured .

(1) processing time vs. granularity

(2) parallel processing overhead vs. granularity

(3) utilization vs. granularity

(4) utilization vs. load status modification delay

(5) processing time deviation vs. load status modification delay

Two load-dispatching strategies are examined. The cluster to which goals are dispatched in one is determined at random (strategy A) and in the other by selecting the cluster with minimum ready goals (strategy B).

Fig. 3 shows normalized processing time in strategy A as a function of the load-dispatching rate for 6-Queens and 8-Queens benchmarks. The normalized processing time is defined as the ratio of the processing time in case of plural clusters to the processing time in case of a single cluster. The load-dispatching rate is defined as the ratio of all goals dispatched to other clusters to all reduced goals. Granularity is expressed by this rate, namely, as a reciprocal of the load-dispatching rate. Parallel processing overhead dominates the processing time in the high load-dispatching rate region and utilization dominates the processing time in the low load-dispatching rate region.

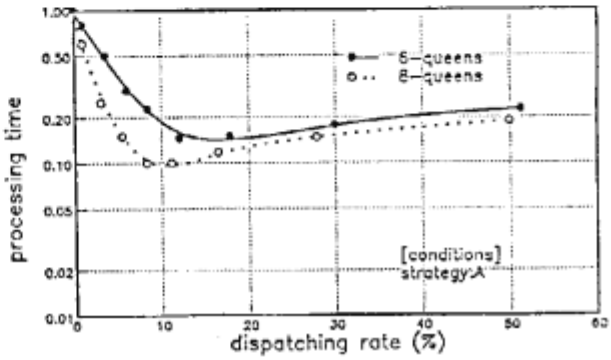

Fig.3 Processing time as a function of dispatching rate

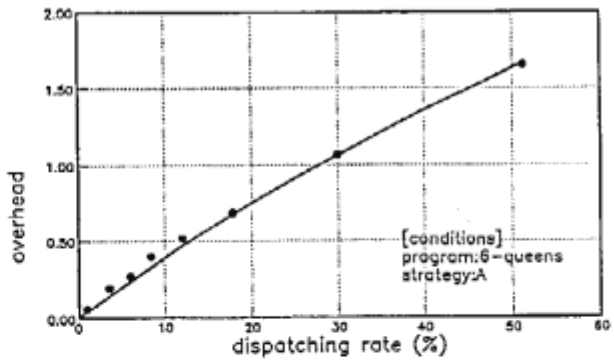

Fig.4 Overhead as a function of dispatching rate

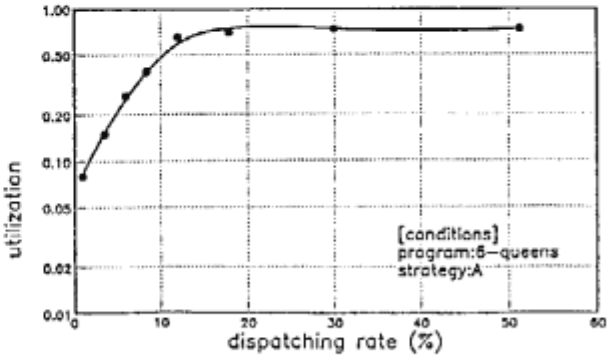

Fig.5 Utilization as a function of dispatching rate

Here, let us introduce the following form to the normalized processing time.

normalized processing time =
(number of clusters) × (averaged utilization)
÷ {1 + (parallel processing overhead)}.

Figures 4 and 5 show the parallel processing overhead and the averaged utilization, respectively, as a function of the load-dispatching rate. Fig. 4 suggests that the load-dispatching rate should be limited to up to 5 % if the parallel processing overhead is designed to be permitted within 0.2.

Figures 6 and 7 show the normalized processing time and the averaged utilization, respectively, as a function of the load-dispatching rate in strategy B, where a dotted line and a chain line indicate the cases that the load status modification delay are equal to 0 and the reduction time, respectively. At the measurement, the delay time was set by means of a hardware timer and interrupt to SBC. That is to say, a value is set on the timer just after the cluster with minimum load is determined , and the load status is changed in the interrupt routine which is executed in the interval of the set value. The delay time is normalized by the measured physical time of the reduction. As shown in Fig. 6 strategy B brings higher performance than strategy A, because it determines the cluster to which goals are dispatched dynamically taking the load distribution imbalance into account. However, it results in about the same performance as strategy A in case the load status modification delay is equal to the reduction time and the load-dispatching rate is high or low. It is shown in Fig. 7 that this degradation is due to the averaged utilization decrease.

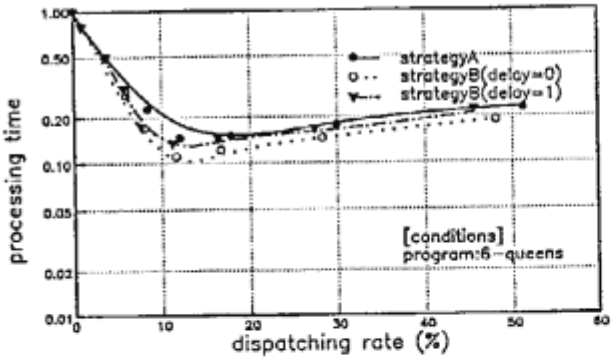

Fig.6 Processing time as a function of dispatching rate

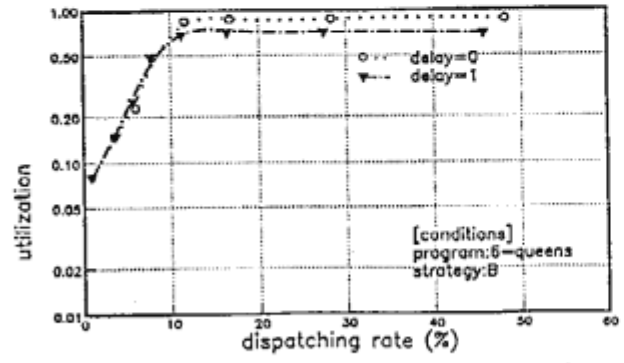

Fig.7 Utilization as a function of dispatching rate

Fig. 8 shows the averaged utilization as a function of the load status modification delay on the condition that the load-dispatching rate = 11 %. Averaged utilization decreases as delay increases. In startegy B the cluster with minimum load is determined and then a goal is dispatched to it. Between one load status modification and the next load status modification, the same cluster is selected as the cluster with minimum load. After a goal is dispatched to the cluster with minimum load, the load distribution changes. However, in case there exists some time delay between load distribution status detection and modification, plural clusters have chances to compete for the same minimum load cluster, even though the load distribution is changed by a goal dispatch. This causes load concentration on a certain cluster, decreases averaged utilization and degrades performance.

At the time of measurement, it was found that data deviated among trials in some conditions. Fig. 9 shows the relationship between the deviation of the averaged utilization and the load-dispatching rate. The deviations

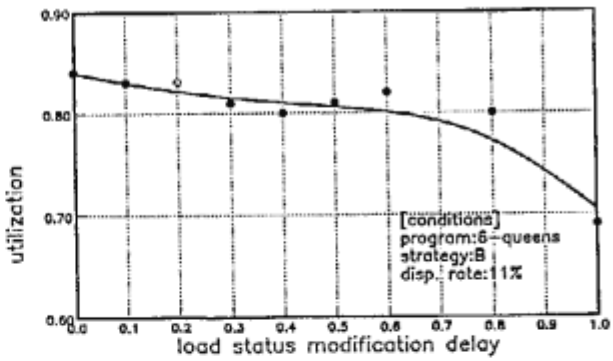

Fig.8 Utilization as a function of load status modification delay

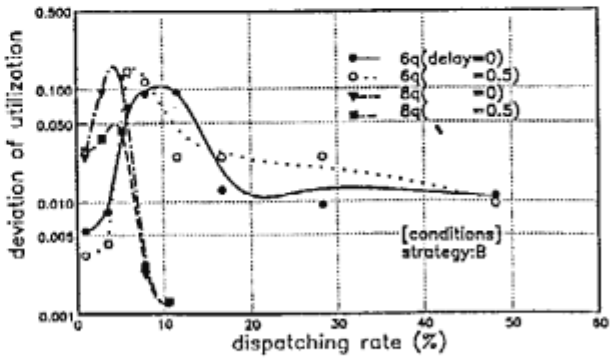

Fig.9 Deviation of utilization as a function of dispatching rate

among trials increase in a slightly lower load-dispatching rate region than the one which gives maximum performance. Fig. 10 shows the deviation of the averaged utilization as a function of the load status modification delay on the condition that the load-dispatching rate = 8 and 11%. In Fig. 10, this deviation looks independent from the load status modification delay.

Fig. 6 indicates that even in the load-dispatching strategy B only half of the ideal performance is achieved for the 6-Queens benchmark in the case of 16 clusters. As mentioned previously, the load-dispatching rate should be kept to less than 5 % if 20 % degradation due to parallel processing overhead is permitted. In the 6-Queens program, sufficient utilization cannot be obtained in this region. In order to confirm the features of the PIM prototype for application programs with higher parallelism, a kernel benchmark program composed of 6-Queens × 16 has been introduced. Figures 11, 12 and 13 show the normalized processing time vs. the load-dispatching rate, the utilization vs. the load status modification delay and the deviation of utilization, respectively. At the execution of the kernel benchmark program on the simulator, top-level 16 goals are assigned to each of 16 clusters statistically. As shown in Fig. 11, in the kernel programs, a maximum performance of 90 % of the ideal value is obtained in 1 ~ 3 % load-dispatching rate. The load status modification delay degrades the utilization, but the degradation is small because the kernel benchmark has sufficient parallelism.

DISCUSSION

The deviation of averaged utilization among trials decreases in the high and low load-dispatching rate regions. In those regions the averaged utilization is rather low regardless of the load status modification delay, because too few goals are dispatched to other clusters in the low load-dispatching rate region, many process suspensions occur, and the goal reductions are somehow serialized in the high load-dispatching rate region. Therefore, goal reduction sequence fluctuation could not influence the utilization of each cluster. In the intermediate region between those two regions, utilization is high because parallelism can be extracted; in other words, because granularity matches the system parallelism. In this region, a good load dispatch to a suitable cluster at a suitable time can decrease the cluster idling time without increasing the working time. Also, in this region, goal reduction sequence fluctuation greatly influences load distribution and cluster utilization.

The utilization deviation among trials may work to decrease the average because the maximum value is limited, namely, it cannot be more than 100 %. Also, it

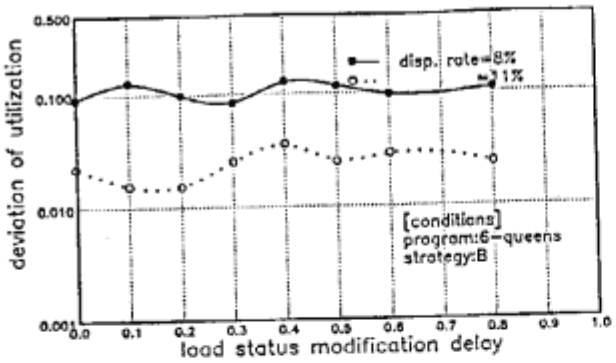

Fig.10 Deviation of utilization as a function of load status modification delay

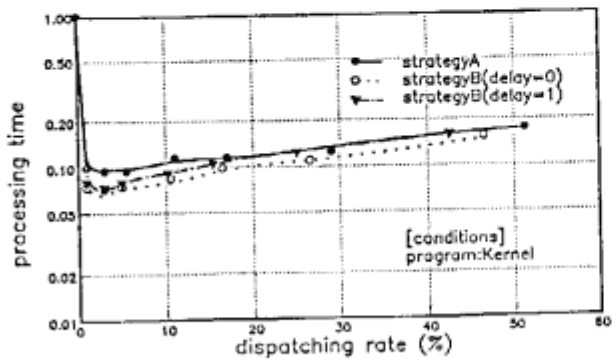

Fig.11 Processing time as a function of dispatching rate

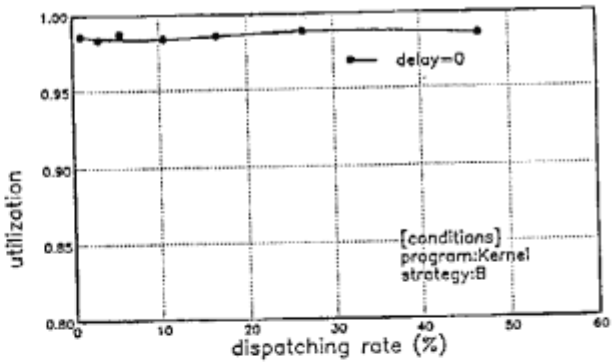

Fig.12 Utilization as a function of dispatching rate

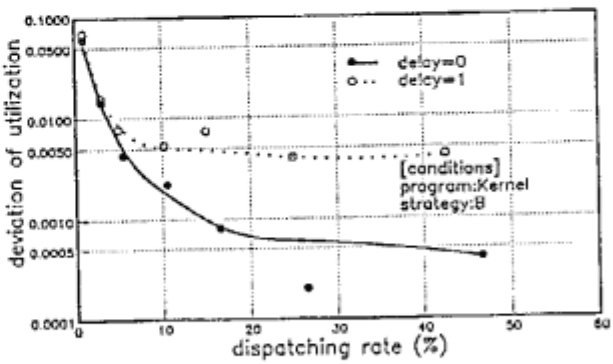

Fig.13 Deviation of utilization as a function of dispatching rate

cannot be ignored that too much deviation may cause a rather high probability of insufficient performance at a trial. Therefore, deviation among trials should be within a tolerable value. From Fig. 9, it is confirmed that at a program execution time the load granularity should be controlled so as not to exceed the 50 % larger than the optimum value in order to limit the deviation to within 0.1.

The performance degradation due to the load status modification delay should be designed to be tolerable. It is confirmed from the result shown in Fig. 8 that the load status modification delay should be less than half of the reduction time to limit the performance degradation to within 5 %. In the PIM prototype, the cluster performance is supposed to achieve 1.6 MLips. In this case, it is not easy to realize a load status modification delay less than half of the reduction time by software monitoring of dispatched goal arrivals, since a reduction is done in about 600 nsec. Hardware support may be necessary. Messages requesting goal reduction are sent through the network. Load distribution status monitoring by the network [11] can be effective and useful.

CONCULSIONS

The bunch layer simulation of the PIM prototype was made, and data on parallel processing overhead and utilization dependencies on granularity, and utilization and its deviation dependences on load status modification delay, were measured and analyzed. From these data, it is confirmed that the load dispatch strategy in which loads are dispatched to the cluster with minimum loads at an AND-fork time is effective on the loosely-coupled cluster level, resulting in 20 % higher performance than in the random dispatch strategy, and that the load status modification delay should be less than half of the reduction time to limit the degradation to within 5 %.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Shun'ichi Uchida, chief of 4th ICOT Laboratory and Dr. Tsuneyo Chiba, head of the 8th Department of Central Research Laboratory, Hitachi, Ltd., for their guidance and support.

REFERENCES

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

[2] K.Nakashima and H.Nakajima, "Hardware architecture of the sequential inference machine: PSI-II," Proceedings of 1987 Symposium on Logic Programming, pp.104-113, San Francisco, 1987

[3] K.Taki, "The parallel software research and development tool: Multi-PSI system," France-Japan Artificial Intelligence and Computer Science Symposium 86, Oct. 1986

[4] N.Ito, M.Sato A.Hishi, E.Kuno and H.Nakazawa, "The architecture and preliminary evaluation results of the experimental parallel inferance machine PIM-D," Proceedings of the 13th Annual International Symposium on Computer Architecture, June 1986

[5] K Kumon, H.Masuzawa, A.Satoh and Y.Sohma, "A new parallel inference method and its evaluation," COMPCOM Spring 86, pp.168-172, IEEE Computer Society, San Francisco, Mar. 1986

[6] R.Onai, H.Shimizu, K.Masuda, A.Matsumoto and M.Aso, "Architecture and evaluation of a reduction-based parallel inference machine: PIM-R," LNCS 221, Springer-Verlag, pp.1-12, 1985

[7] A.Goto and S.Uchida, "Toward a high performance parallel inference machine - The intermediate stage plan of PIM -," Future Parallel Computers, LNCS 272, Springer-Verlag, 1986

[8] S.Sakai, H.Koike, H.Tanaka and T.Motooka, "Interconnection network with dynamic load balancing facility," Transaction of Information Processing, vol.27, no.5, pp.518-524, (in Japanese), 1986

[9] K.Hiraki, S.Sekiguchi and T.Shimada "Load scheduling mechanism usig inter-PE network," Transaction of IECE Japan vol.J69-D, no.2, pp.180-189, (in Japanese), 1986

[10] K.Ueda, "Guarded horn clauses: A parallel logic programming language with the concept of a guard," TR 208, ICOT, 1986

[11] M.Sugie, M.Yoneyama, T.Sakabe M.Iwasaki, S.Yoshizumi, M.Aso, H.Shimizu and R.Onai, "Hardware simulator of reduction-based parallel inference machine PIM-R," LNCS 221, Springer-Verlag, pp.13-24, 1985