PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

LOAD-DISPATCHING STRATEGY ON PARALLEL INFERENCE MACHINES

M.Sugie, M.Yoneyama, N.Ido and T.Tarui
Central Research Laboratory, Hitachi, Ltd.
Higashi-Koigakubo, Kokubunji, Tokyo 185, Japan

ABSTRACT

Load-dispatching strategies for a parallel inference machine prototype are described, and their performance is evaluated by simulation based on the loosely-coupled cluster model, using the 6-queens benchmark. The sender-initiate concept is applied to the bunch layer of a parallel inference machine prototype. The strategy in which the cluster with maximum ready goals dispatches a goal to the cluster with minimum ready goals brings the lowest load-dispatching rate limit. Stable performance during real program execution is not expected in this strategy, because it covers too narrow load-dispatching rate region . A strategy in which the goal dispatch target cluster is determined at random, but conditionally aborted based on dynamic loads at the dispatching cluster and the target cluster, brings the second lowest load-dispatching rate limit and is expected to have high performance and stability. More than 70% average utilization is achieved when the load-dispatching rate is higher than 5%. It is confirmed that 0.54 of the maximum performance of a parallel inference machine prototype can be achieved by applying this load-dispatching strategy.

1 INTRODUCTION

The Fifth Generation Computer Project has developed knowledge and information processing systems based on a predicate logic programming language [Fuchi and Furukawa 87], [Nakashima and Nakajima 87], [Taki 86]. The hardware of these systems has been dubbed an "Inference Machine". In the project's initial stage, sequential architectural inference machines were developed, and various parallel architectural concepts were designed and evaluated [Ito et al. 86], [Kumon et al. 86], [Onai et al. 85]. The 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[Goto and Uchida 86].

The main research areas of PIM are parallel processing overhead and processing-element utilization. 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 finer the granularity, the larger the utilization, so if 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. Parallel logic programming languages such as KL1 have a suspend/resume processes feature for concurrent process control [Ueda 86]. This feature causes much parallel processing overhead. Therefore, the PIM prototype load granularity is of coarse design. Load-balancing feature research is important for improving processing-element utilization.

Several load-balancing methods have been developed [Sakai et al. 86], [Hiraki et al. 86], in which load dispatch targets are determined dynamically by selecting the processing-element with minimum load. Once the processing-element with minimum load is determined, all processing-elements prepare to dispatch loads to it. If there is a time delay between load status detection and modification, load concentration on one processing-element occurs and the load concentration degrades the performance of the PIM prototype [Sugie et al. 88].

In this paper, load-balancing features of the PIM prototype, especially the load-dispatching strategies based on the sender-initiate concept are investigated. Several strategies which are distinguished by load dispatch conditions and targets are assessed from the point of view of granularity limit .

2 CONCEPT OF SYSTEM ORGANIZATION AND LOAD BALANCING OF A PIM PROTOTYPE

Parallel architecture may be used for improving processing ability. To improve this ability efficiently, program localization of closely related sequences must be considered whenever possible.

KL1, the PIM prototype target language, has a suspend/resume processes feature. This feature makes it possible to express concurrent process control flow explicitly in programs, but is a burden on inference machine processing ability. Therefore, occurrences of suspension/resumption have to be reduced at program execution time. In some cases, simple depth-first process activation scheduling can reduce the occurrence of suspension/resumption. That is to say, when a process is activated, the cause of process suspension is eliminated by past process activation. Even in the case of parallel architecture, this sort of scheduling implementation is important, since process suspension/resumption would be too great a parallel processing overhead.

A hierarchical structure, as is shown in Fig. 1, is useful to reduce occurrences of suspension/resumption, namely, parallel processing overhead 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), where a cluster is a group of small elements and a bunch is a group of large elements.

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 high a capacity for too little work. This would be too great an overhead burden.

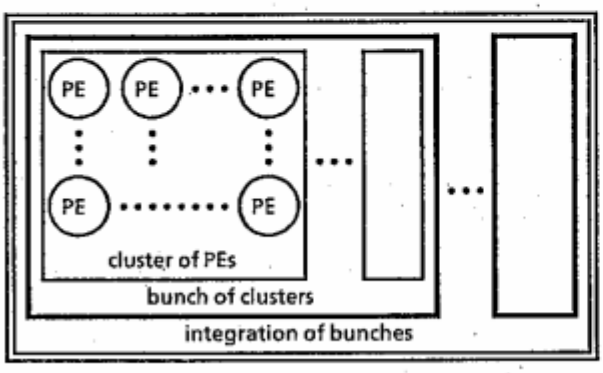

Fig. 1 Hierarchical structure for PIM

Current technology makes it possible to construct a PIM prototype with 2-layer hierarchy [Goto and Uchida 86]. The bunch of loosely-coupled clusters is the top layer. It consists of about 10 clusters coupled loosely through some sort of equal-length network such as a crossbar. The cluster consists of about 10 processing-elements coupled tightly through shared storage and caches.

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, load balancing is achieved by frequent communication between processing-elements, and the occurrence of suspension/resumption can be reduced by process activation scheduling using a common ready process queue stored in the shared storage. In the bunch layer, clusters communicate by sending/receiving messages. Communication between clusters should be restricted since such communication causes an overhead burden.

There are two basic concepts of dynamic load balancing, receiver-initiate and sender-initiate. In the former case, loads are dispatched to processing-elements when requested because of idling. In the latter case, load dispatch is determined only by the dispatcher's situation, regardless of whether the target processing-elements have loads or not. If a small number of processing-elements are installed in a PIM, receiver-initiate method is more efficient, because there is no wasted communication. However, this method is not appropriate for a PIM with a large number of processing-elements, because too much throughput is needed for the channels broadcasting load requests. The sender-initiate method is appropriate in this case.

It is difficult to extend the cluster size to much more than 10 processing-elements. For example, a PIM bunch layer with 1000 processing-elements would be composed of about 100 clusters. Therefore, the receiver-initiate method could not be applied to the bunch layer. For future large scale design of a PIM, the sender-initiate method will be used for the

bunch layer load balancing of the PIM prototype.

3 LOAD-DISPATCHING STRATEGY

In the PIM prototype, bunch layer load balancing is controlled by determining whether goal dispatch occurs and determining dispatch target cluster by a strategy, every time a goal reduction creates new goals.

In the bunch layer of the PIM prototype, load dispatch waste needs to be avoided so as to reduce occurrences of suspension/resumption. There are two useful ideas to avoid load dispatch waste, namely, to anticipate the clusters which may need load dispatch (idea A) and to stop load dispatch under bad conditions (idea B). The following four load-dispatching strategies are examined.

strategy A : The cluster to which goals are dispatched is determined at random.

strategy B : The cluster to which goals are dispatched is determined by selecting the cluster with minimum ready goals.

strategy C : 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 cluster has more ready goals than the dispatching cluster.

strategy D : The cluster with maximum ready goals dispatches a goal to the cluster with minimum ready goals.

Strategy B is a realization of idea A, strategy C is a realization of idea B and strategy D is a realization of both idea A and B. Strategy A is classified as "blind". It is examined to evaluate the performance of strategies B,C and D which are classified as "informed".

Goal dispatch should be aborted if the dispatching cluster has insufficient ready goals. The goal dispatch under this condition may make the dispatching cluster idle. Such a bad goal dispatch can be avoided by aborting goal dispatch when the dispatching cluster has fewer ready goals than some threshold. Strategy B' and strategy C', strategies in which this idea, strategy B, and strategy C are combined are also examined.

At KL1 program execution time, the initial query is assigned to one cluster and created new goals are dispatched to the other clusters. Therefore, utilization of clusters can be improved by dispatching more goals in the initial stage of program execution than in the medium and final stage. Strategy B" and strategy C", strategies in which this idea, strategy B, and strategy C are combined are also examined.

4 SIMULATION

To achieve dynamic load balancing, load-dispatching strategies in the bunch layer of the PIM prototype are examined by simulation.

4.1 Simulator

Simulation is made on the PIM-R hardware simulator [Sugie et al. 85] using an interpreter for KL1.

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.

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. As the purpose of this simulation is bunch-layer simulation, detailed structure and operation inside the cluster is not simulated. During simulation, SBC works as a single processing-element with high performance.

In the hardware simulator, the event-driven method is employed to eliminate the idling time during simulation. 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

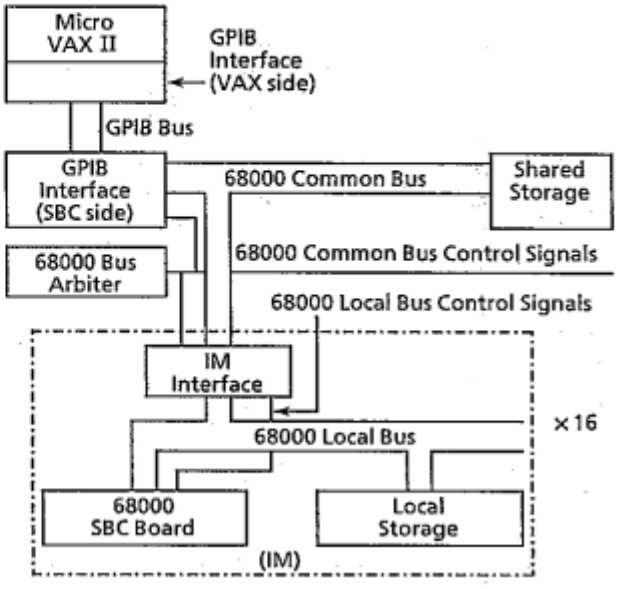

Fig.2 Hardware block diagram of a simulator

every time a transaction of any one of several functions is executed. When messages are sent to other clusters, network delay time is added to 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 count when it accepts the message. During simulation, all data measurements and some operations such as queue controls are based on the cluster software timer.

4.2 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 at head unification time.
(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.

4.3 Results

The relationship between utilization and granularity in the PIM prototype composed of 16 clusters is measured and the above-mentioned load-dispatching strategies are evaluated from the point of view of granularity limit.

Fig. 3 shows the normalized processing time of strategy A,B,C and D as a function of the load-dispatching rate for 6-Queens benchmark. The normalized processing time is defined as the ratio of the processing time for plural clusters to the processing time for 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. The load-dispatching rate is varied by changing the simulation parameter which controls load dispatch probability. 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.

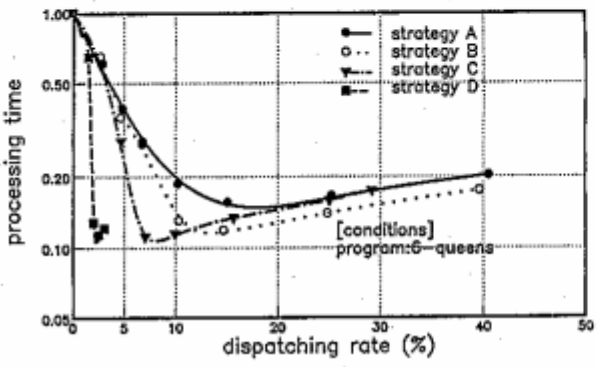

Fig.3 Processing time as a function of dispatching rate

The normalized processing time is expressed by

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

Figures 4 and 5 show the parallel processing overhead and the average utilization, as a function of the load-dispatching rate.

Fig. 4 shows that the parallel processing overhead is expressed by two straight lines with different gradients, namely, with a high gradient in the low load-dispatching rate region and with low gradient in the high load-dispatching rate region. An optimization is introduced into the KL1 interpreter on the simulator which can reduce the communication between clusters by storing values in a cluster which are instantiated in other clusters and sent to the cluster through messages. Once such values are stored in the cluster, no more communication is needed to get them. When load-dispatching rate is low, so few variables are shared between clusters that the above-mentioned optimization is not effective and parallel processing overhead is expressed by a line with a higher gradient than that in the high load-dispatching rate region.

In Fig. 4, dots which express data measured in

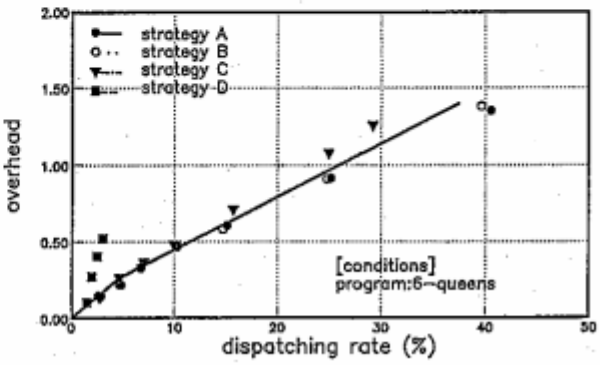

Fig.4 Parallel processing overhead as a function of dispatching rate

strategy D deviate from the straight line. This is due to the simulation mechanism. In the simulator, strategy D is implemented by dispatching a goal to the dispatching cluster itself to abort goal dispatch, on the condition that the dispatching cluster does not have maximum ready goals. This method can abort goal dispatch to other clusters, but cannot eliminate overhead of sending and receiving messages. This causes a large overhead in spite of the low final load-dispatching rate. Such an overhead can be removed by determining goal dispatch before preparing the message and not sending the message under inappropriate conditions. In Fig. 4, the parallel processing overhead in strategy C is larger than those in strategy A and B. This is also due to the implementation of aborting goal dispatch on the simulator.

Fig. 4 suggests that the load-dispatching rate should be limited to 5 % if the permissible parallel processing overhead is designed to be within 0.3.

It should be noted that strategy C has higher performance than strategy B. This indicates that strategy C can eliminate more load dispatch waste than strategy B. Strategy D has the highest performance, but the load-dispatching rate is low. Using strategy D, each cluster checks load distribution at reduction intervals and dispatch a goal on the condition that it has maximum ready goals. Therefore, goals cannot be dispatched at a higher rate than once per 16 reductions. On the contrary, plural clusters have a chance to dispatch goals at the same time in other strategies.

Figures 6 and 7 show average utilization, as a function of load-dispatching rate in strategies B', B," C' and C". These figures show that all four strategies improve performance, B' and C' more than B" and C".

Simulation was also carried out using other sample programs, such as BUP (Bottom-Up Parser) or kernel benchmark. The results using these benchmark programs are quantitatively different but qualitatively the same. Queens program can distinguish suspension/resumption features in the plural cluster case since it causes only one suspension/resumption if it is executed on one processing-element.

5 DISCUSSION

In the PIM prototype bunch layer, load granularity must be kept coarse to reduce occurrences of suspension/resumption. Therefore, the load-dispatching strategies should be evaluated on the basis of how low the load-dispatching rate can be, keeping sufficiently high utilization. Table-I shows the load-dispatching rate limits, defined as the load-dispatching rate which gives 70 % utilization in each load-dispatching strategy. Strategy D has the lowest load-dispatching rate limit. However, this strategy covers too narrow a load-dispatching rate region, which should be wide to realize high and stable performance. In strategy D, too low a load-dispatching rate causes high

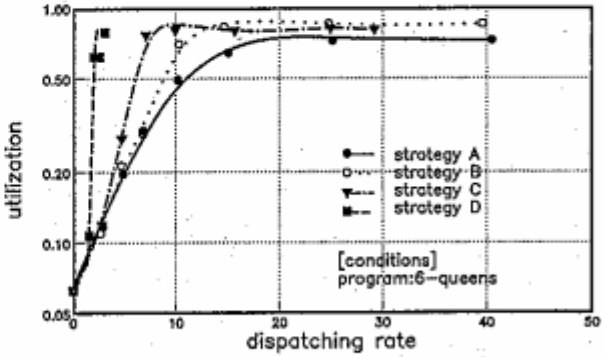

Fig.5 Utilization as a function of dispatching rate

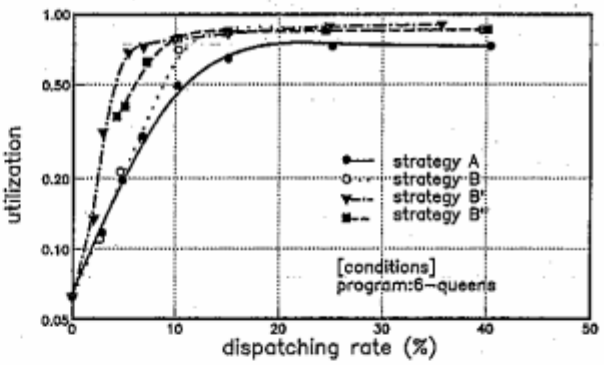

Fig.6 Utilization as a function of dispatching rate

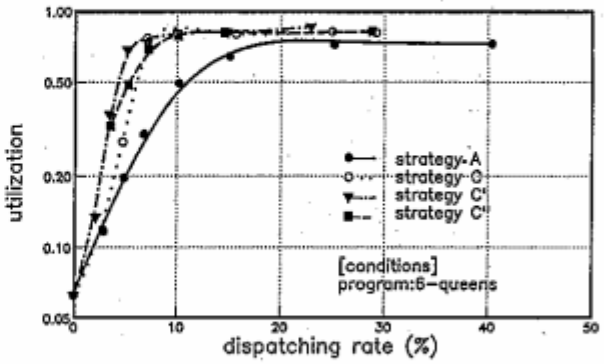

Fig.7 Utilization as a function of dispatching rate

Table-I Dispatching rate limit

strategy	A	B	C	D	B'	C'	B"	C"
dispatching rate limit (%)	15.0	10.5	6.5	2.5	5.5	5.0	8.0	7.5

performance degradation.

Except for strategy D, strategies B' and C' have the lowest load-dispatching rate limits and can achieve approximately the same performance as each other. Strategy C' is expected to have more stable performance than strategy B' during real program execution, since the performance difference between strategies C and C' is smaller than between strategies B and B'. The difference between strategies B and B' and between strategies C and C' is that strategies B' and C' abort goal dispatch if the dispatching cluster has fewer ready goals than the threshold. In strategies B' and C', performance depends on threshold and an optimum threshold for each application program exists. Figures 6 and 7 show that strategy C' has less threshold dependency than strategy B', and that strategy C' has more stable performance than strategy B'.

The performance of the PIM prototype is estimated, on the basis of simulation results. Table-I shows that the load-dispatching rate can be reduced to 5 %. Fig. 4 shows that parallel processing overhead becomes 0.3 when the load-dispatching rate is 5 %. Substituting 16, 0.7 and 0.3 for number of clusters, average utilization and parallel processing overhead, respectively, equation (1) gives normalized processing time as 8.6, which is more than half of the maximum performance. Assuming that the cluster performance equals 1 MLips, the performance of the PIM prototype is 8.6 Mlips.

So far, load-dispatching strategies have been being discussed from the point of view of granularity limit. To assess load-dispatching strategy, it is assumed that the PIM prototype has sufficient ability for inter-cluster communication processing. This assumption may differ from the real design. The granularity limit may also be assessed from the point of view of inter-cluster communication processing ability. In the PIM prototype, a cluster controller is introduced into the cluster for inter-cluster communication processing. The cluster controller is of approximately the same processing ability as the processing-element.

Suppose that a cluster controller and eight processing-elements are installed in the cluster, that the cluster controller has the same processing ability as the processing-element and that inter-cluster communication processing costs twice as high as reduction, then the load-dispatching rate limit should be less than 0.0625 ($=1/2\times8$). In this paper, an automated dynamic load-dispatching method is used and the simulation results show that a satisfactory load-dispatching rate limit was achieved. If inter-cluster communication processing dominates the load-dispatching rate limit, measures to assist the load-balancing method are necessary. Task division techniques using program characteristics and monitoring load distribution globally are useful. A big task may be divided into several sub-tasks. These sub-tasks are assigned into sub-bunches composed of less than 16 clusters. Inside the sub-bunch, the load-dispatching strategies can reduce the load-dispatching rate limit so low that the cluster controller can handle inter-cluster communication.

The bunch size is fixed in this simulation. However, average utilization is a function of bunch size, too. In the future, bunch size influence should be investigated to predict PIM behavior in case of too high-cost inter-cluster communication processing or large scale PIM with more than 16 clusters.

The subjects of this paper are new load-dispatching strategies on the dynamic load balancing of the PIM prototype. They are applicable to parallel systems in which the context switch is a relatively heavy burden to normal processing.

6 CONCLUSION

Several load-dispatching strategies based on the sender-initiate concept were developed and evaluated in the bunch layer of the PIM prototype. The lowest load-dispatching rate limit was achieved when the goal dispatch target was determined at random and then this goal dispatch was aborted if the dispatch target cluster had more ready goals than the dispatching cluster or the dispatching cluster had fewer ready goals than the threshold. It was confirmed that more than half of the PIM prototype maximum performance can be achieved by applying this load-dispatching strategy. This strategy is expected to realize stable performance in the wide load-dispatching rate region.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Shun'ichi Uchida, chief of 4th ICOT Laboratory for his guidance and support and Dr. Atsuhiro Goto, senior researcher of ICOT Laboratory for his 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," 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.Kishi, E.Kuno and K.Rokusawa, "The architecture and preliminary evaluation results of the experimental parallel inference 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] K.Ueda, "Guarded horn clauses: A parallel logic programming language with the concept of a guard," TR 208, ICOT, 1986

[9] 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

[10] 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

[11] M.Sugie, M.Yoneyama, and A.Goto, "Analysis of parallel inference machines to achieve dynamic load balancing," Proceedings of International Workshop on Artificial Intelligence for Industrial Applications, pp.511-516, 1988

[12] 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