ICOT Technical Report: TR-371

TR-371

Load-dispatching Strategy on Parallel Inference Machines

by
M. Sugie, M. Yoneyama,
N. Ido and T. Tarui(Hitachi)

May, 1988

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

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 are proposed for a parallel inference machine prototype, and their performance are evaluated by the simulation based on the loosely-coupled cluster model, using 6-queens benchmark. Sender-initiate concept is appropriate 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, but it is not expected to realize stable performance at real program execution, since it covers too narrow load-dispatching rate region. The strategy in which the goal dispatch target cluster 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 or on the condition that the dispatching cluster has fewer ready goals than the threshold brings the second lowest load-dispatching rate limit and is expected to realize high performance stably at real program execution. More than 70% averaged utilization is achieved in the region that 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.

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 systems has been dubbed an "Inference Machine". 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. Parallel logic programming languages such as KL1 have a suspend/resume processes feature for concurrent process control [10]. 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.

Several load-balancing methods have been proposed [8],[9], 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 that processing-element. In case there is some time delay between load status detection and modification, load concentration on one processing-element occurs and this load concentration degrades the performance of the PIM prototype [11].

This paper investigates load-balancing features of the PIM prototype, especially the load-dispatching strategies based on the sender-initiate concept. Several strategies which are discriminated by load dispatch conditions and targets are discussed from the granularity limit point of view.

CONCEPT ON SYSTEM ORGANIZATION AND LOAD BALANCING 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. 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 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 crossbar. The cluster can consist 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 could be achieved by frequent communications between processing-elements, and occurrences of suspension/resumption could 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 a communication would cause an overhead burden.

There are two basic concepts on dynamic load balancing, namely, receiver-initiate and sender-initiate ones. In case small number of processing-elements are installed in a PIM, receiver-initiate method is efficient, because there is no waste communication. However, this method is not appropriate for a PIM with large number of processing-elements, because too many request channels are needed. The receiver-initiate method could be appiled to the cluster layer of the PIM prototype.

Assuming a PIM with 1000 processing-elements, it would be composed of about 100 clusters. Therefore, the receiver-initiate method cannot be applied to the bunch layer. Considering the future organization of a PIM, the sender-initiate method is introduced to the bunch layer load balancing of the PIM prototype.

LOAD-DISPATCHING STRATEGY

In the bunch layer of the PIM prototype, it is intensively requested to avoid waste load dispatch so as to reduce occurences of suspension/resumption. To avoid waste load dispatch, there are two useful ideas, namely, to anticipate the clusters which may need load dispatch (idea A) and to stop load dispatch under bad situation (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 read 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, respectively. Strategy A is not an intelligent one, but it is examined to evaluate the performance of strategies B,C and D.

Goal dispatch should be aborted on the condition that the dispatching cluster has few 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 on the condition that the dispatching cluster has fewer ready goals than the threshold. Strategies (strategy B' and C') in which this idea and strategy B and C are combined are also examined.

At KL1 program execution time, the initial query is assingned 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. Strategies (strategy B" and C") in which this idea and strategy B and C are combined are also examined.

SIMULATION

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

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.

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.

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.

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 relationships between utilization and granularity on the PIM prototype composed of 16 clusters have been measured and the above-mentioned load-dispatching strategies have been evaluated from the granularity limit point of view.

Fig. 3 shows normalized processing time in 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 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.

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

normalized processing time =

(number of clusters) × (averaged utilization)

÷ {1 + (parallel processing overhead)}. ·····(1)

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

It is shown in Fig. 4 that the parallel processing overhead is expressed by two straight lines with different gradients, namely, with large gradient in the low load-dispatching rate region and with small gradient in the high load-dispatching rate region. Such an optimization is introduced into the KL1 interpreter on the simulator as could reduce the communication beween clusters by storing those 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 communications are needed to get them. In the low load-dispatching rate region, so few variables are shared beween clusters that the above-mentioned optimization is not effective and parallel processing overhead is expressed by the line with larger gradient than that in the high load-dispatching rate region.

In Fig. 4, dots which express data measured in strategy D deviate from the straight line. This is due to the simulation mechanism. On the simulator, strategy D is implemented by dispatching a goal to the dispatching cluster itself so as to abort goal dispatch on the condition that the dispatching cluster does not have maximum ready goals. This implementation can realize aborting goal dispatch to other clusters, but cannot eliminate overhead of sending and receiving messages. This causes large overhead in spite of low final load-dispatching rate. Such an overhead can be removed by deremining goal dispatch before preparing a message and avoiding sending a message on an inappropriate condition. In Fig. 4, 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 up to 5 % if the parallel processing overhead is designed to be permitted within 0.3.

It should be noted that startegy C brings higher performance than strategy B. This indicates that strategy C can eliminate more waste load dispatch than strategy B. Strategy D brings the highest performance, but the load-dispatching rate is limited to be low in this strategy. In 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 of clusters have chances to dispatch goals at the same time in other strategies.

Figures 6 and 7 show the averaged utilization, as a function of the load-dispatching rate in strategies B' and B" and in strategies C' and C", respectively. As shown in those figures, the four strategies improve the performance, but strategies B' and C' bring higher performance than strategies B" and C".

DISCUSSION

In the bunch layer of the PIM prototype, it is intensively requested to keep load granularity coarse so as to reduce occurrences of suspension/resumption. Therefore, the load-dispatching strategies should be evaluated on the point how low the load-dispatching rate can be, keeping sufficiently high utilization. In Table-I, are shown the load-dispatching rate limits that are defined as the load-dispatching rates which give 70 % utilization in each load-dispatching strategy. As shown in that table, strategy D brings the lowest load-dispatching rate limit. However, too narrow load-dispatching rate region can be covered by this strategy. In order to realize sufficient performance stably at real program execution, high utilization should be kept in wide load-dispatching rate region. In strategy D, a little too low load-dispatching rate causes much performance degradation.

Excluding strategy D, strategies B' and C' bring the lowest load-dispatching rate limit and can achieve about the same performance. In these two strategies, however, strategy C is expected to realize more stable performance at real program execution, since the performance differnce between strategies C and C' is smaller than that between strategies B and B'. The difference between strategies B and B' and between strategies C and C' is to abort goal dispatch on the condition that the dispatching cluster has fewer ready goals than the threshold. In sytrategy B' and C', performance depends on the threshold and there exists the optimum threshold for each application program. Figures 6 and 7 show that strategy C' has less threshold dependency than strategy B' and that strategy C' realize more stable performance than strategy B'.

Here, let us estimate the performance of the PIM prototype. From Table-I, the load-dispatching rate can be reduced to 5 %. From Fig. 4, parallel processing overhead becomes 0.3 on the condition that the load-dispatching rate is 5 %. By substitution of 16, 0.7 and 0.3 into number of clusters, averaged utilization and parallel processing overhead, respectively, equation (1) gives us 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 becomes 8.6 Mlips.

CONCLUSIONS

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

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.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] 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, and A.Goto, "Analysis of parallel inference machines to achieve dynamic load balancing," Proceedings of International Workshop on Artificial Intelligence for Industrial Applications (to be published in May, 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

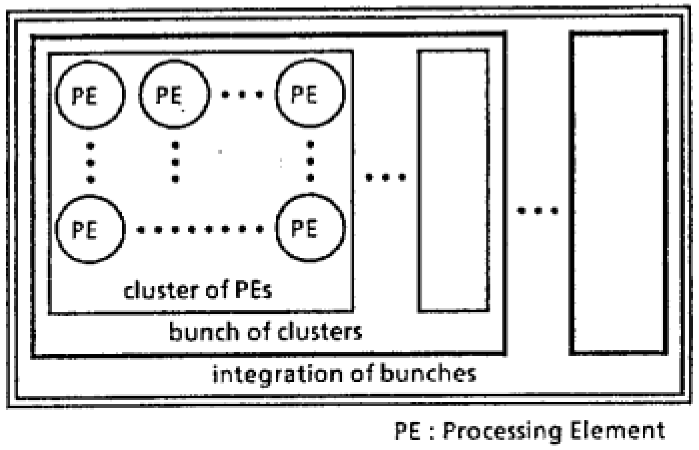

Fig. 1 Hierarchical structure for PIM

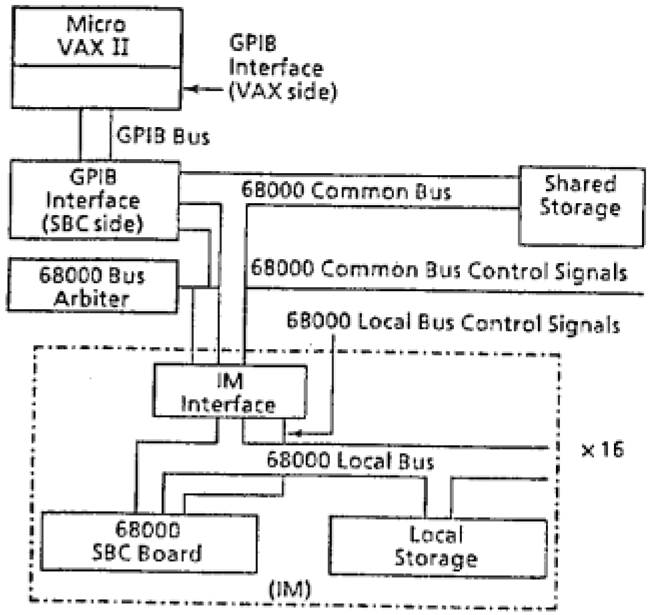

Fig.2 Hardware block diagram of a simulator

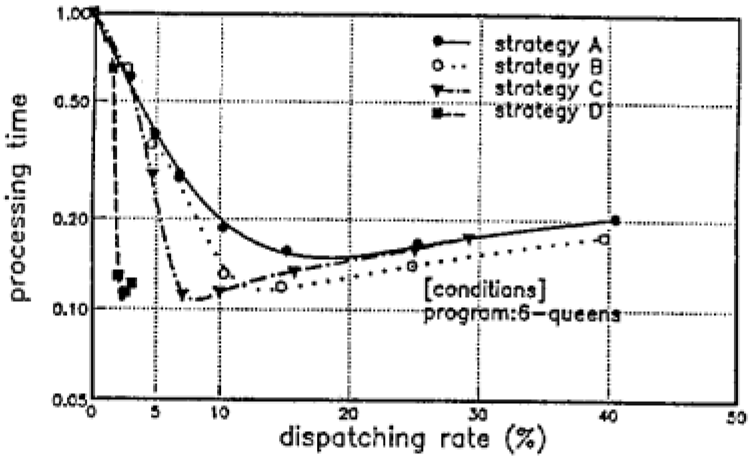

Fig.3 Processing time as a function of dispatching rate

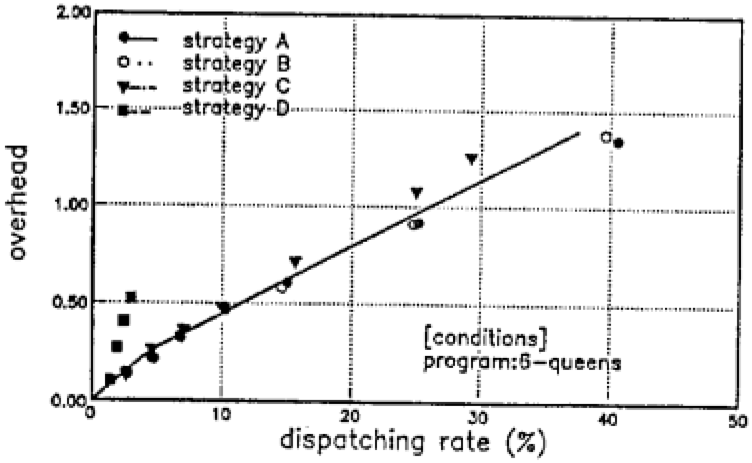

Fig.4 Paralle procesing overhead as a function of dispatching rate

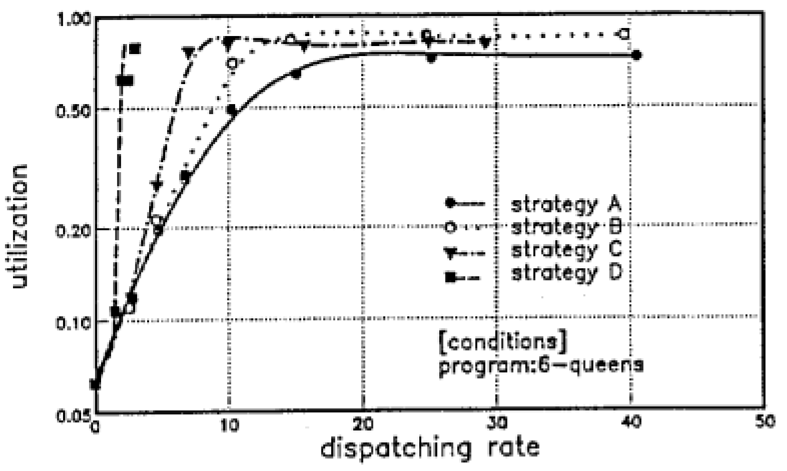

Fig.5 Utilization as a function of dispatching rate

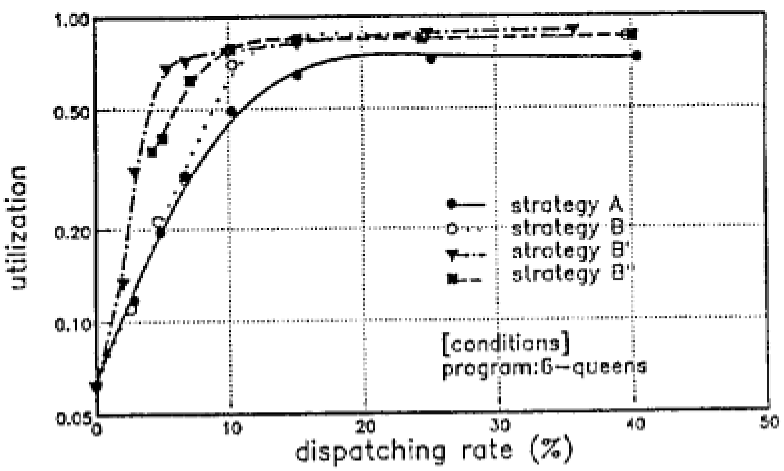

Fig.6 Utilization as a function of dispatching rate

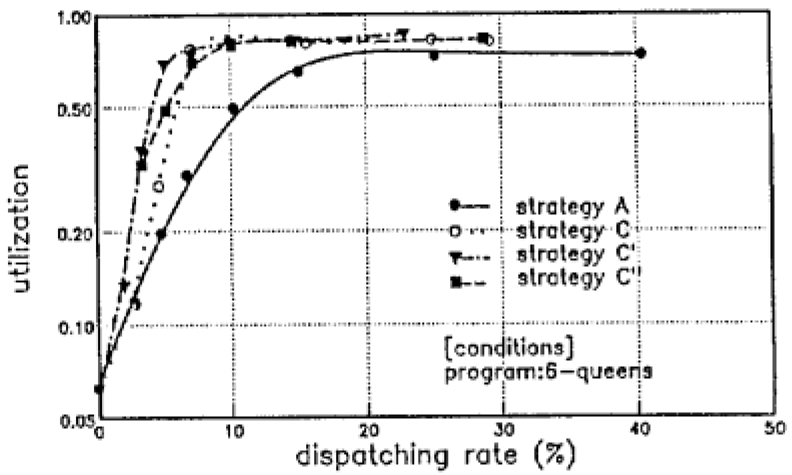

Fig.7 Utilization as a function of dispatching rate

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