TR-431

System Size Dependency of Minimum Load-Dispatching Rate in Parallel Inference Machines

by
M. Sugie and N. Inoue(Hitachi)

November, 1988

SYSTEM SIZE DEPENDENCY OF MINIMUM LOAD-DISPATCHING RATE IN PARALLEL INFERENCE MACHINES

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

The area of interest : Evaluation of parallel systems by simulation

ABSTRACT

System size dependency of the minimum load-dispatching rate for dynamic load balancing in parallel inference machines is described, where system size is the number of processor nodes connected through a network and the minimum load-dispatching rate for load balancing is the lowest load-dispatching rate in which a certain level of utilization is maintained. Relationship between the minimum load-dispatching rate and system size is measured in two load-dispatching strategies based on the sender-initiate concept by simulation on the loosely-coupled multi-processor model. The minimum load-dispatching rate has sub-linear (increase more slowly than linear) dependency on the system size in smart-random and max-min strategies. In high average utilization cases, it is approximately proportional to n in the former strategy and $(n - 1) / n$ in the latter strategy, where n is the system size. By decreasing system size, the minimum load-dispatching rate cannot be reduced so much in max-min strategy, but can be reduced in smart-random strategy. The load dispatch may also be assessed from the point of view of communication processing ability. In smart-random strategy, task division techniques can adjust the minimum load-dispatching rate for load balancing to communication processing capacity.

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". Various parallel architectural concepts for the inference machine were designed and evaluated [Ito et al. 86], [Kumon et al. 86], [Onai et al. 85]. Now, 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 the granularity of parallel systems. Generally, the finer the granularity, the larger the utilization, so if fine 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. Load-balancing feature research is important for improving processing-element utilization, since the PIM prototype granularity is of coarse design.

Several load-balancing methods have been developed [Sakai et al. 86], [Hiraki et al. 86], [Yamauchi and Tanaka 88], 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 a PIM [Sugie et al. 88]. Two load-balancing methods which can avoid this load concentration and realize higher performance than these methods were developed [Sugie et al. 88]. In one load-balancing method (smart-random strategy), load dispatch target is determined at random and then this goal dispatch is aborted on the condition that the dispatch target has more loads than the dispatching processing-element. In the other (max-min strategy), loads are dispatched to the processing-element with minimum load by the processing-element with maximum load.

To utilize parallel architecture efficiently, program localization of closely related sequences must be considered whenever possible. A hierarchical structure is introduced in the PIM prototype to utilize this program localization efficiently. Fig. 1 shows the block diagram of the PIM prototype. Current technology makes it possible to construct a PIM prototype with 2-layer hierarchical hardware. A processing-element is a bottom-layer component, and a cluster is a group of processing-elements. In the bunch layer, 16 clusters are connected through a network. Each cluster has eight processing-elements and a cluster controller for inter-cluster communication.

In the cluster layer, processing-elements and the cluster controller are tightly coupled through shared memory and coherent caches, and so, only a small amount of overhead is required for each processing-element to communicate with others. In programs written in KL1, there is frequent communication among closely related parts. Therefore, by assigning these program parts to the same cluster, we can efficiently utilize the processing ability of the PIM prototype.

In the PIM prototype configuration, communication overhead is small in the cluster layer. Inside the cluster, load balancing is achieved by frequent communication between processing-elements. In the bunch layer, clusters communicate by sending/receiving messages. Communication between clusters should be restricted since such communication causes an overhead burden. Communication between clusters comes from load dispatch. Information is necessary to execute program parts corresponding to dispatched load and some of it should be obtained in execution time through communication between the dispatch cluster and the dispatch target cluster. Therefore, load dispatch should be as infrequent as possible to reduce communication between clusters. Frequent load dispatch is preferable for load balancing. The load dispatch limit for dynamic load balancing in the PIM bunch layer is investigated in this paper. The bunch layer of the PIM prototype is a usual parallel system which is composed of plural processor nodes connected through a network. The load-balancing feature in the bunch layer resolves into the same problem in a general parallel systems.

The load dispatch in parallel systems may also be assessed from the point of view of communication processing ability. If parallel systems do not have sufficient communication processing ability, frequent load dispatch cannot be realized. If communication processing ability dominates the load-dispatching rate, which is defined in a PIM as the ratio of all dispatched goals to all reduced goals, it is necessary to decrease the minimum load-dispatching rate for load balancing, to the load-dispatching rate determined by communication processing ability. The minimum load-dispatching rate for load balancing may depend on the number of processor nodes. Task division techniques in which a large task is divided into several sub-tasks and these sub-tasks are assigned to sub-systems of the whole parallel system may reduce the minimum load-dispatching rate for load balancing. The number of nodes in a PIM was fixed in the evaluation of smart-random and max-min strategies [Sugie et al. 1988]. The purpose of this paper is to investigate the system size influence on the minimum load-dispatching rate for dynamic load balancing. The system size can be expressed by the number of processor nodes. The minimum load-dispatching rates for load balancing are measured in smart-random and max-min strategies and in different system sizes by simulation and analyzed.

2 LOAD-DISPATCHING STRATEGY

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 processor nodes have loads or not. If a small number of processor nodes are installed in a parallel system, receiver-initiate method is more efficient, because there is no wasted communication. However, this method is not appropriate for a parallel system with a large number of processor nodes, because very high throughput is needed for the channels broadcasting load requests. The sender-initiate method is appropriate in this case.

It is difficult to extend the PIM cluster size to much more than 10 processing-elements, because of tight couple through coherent caches. For example, a PIM bunch layer with 1000 processing-elements would be composed of about 100 clusters. Therefore, it is difficult to apply the receiver-initiate method 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.

In the sender-initiate method, load balancing is controlled by determining whether a load is dispatched and determining dispatch target. Load-dispatching strategies manage these load dispatch conditions and targets. In the load-dispatching strategies, load dispatch waste needs to be avoided so as to reduce parallel processing overhead. Sugie et al. [1988] showed that smart-random and max-min strategies were efficient. Max-min strategy has the highest performance. However, this strategy covers too narrow a region in the low load-dispatching rate. Even if a network has sufficient throughput and a parallel system has sufficient communication processing ability, max-min strategy cannot utilize this capacity well. On the contrary, smart-random strategy covers wide load-dispatching rate region and is expected to achieve high performance stably. The following three load-dispatching strategies are examined.

strategy A (baka-random) : The node to which loads are dispatched is determined at random.

strategy B (smart-random) : The node to which loads are dispatched is determined at random and then this load dispatch is aborted on the condition that the dispatch target node has more loads than the dispatching node.

strategy C (max-min) : The node with maximum loads dispatches a load to the cluster with minimum loads.

Strategy A is classified as "blind". It is examined to evaluate the performance of strategies B and C which are classified as "informed".

3 SIMULATION

The minimum load-dispatching rates for load balancing are examined in three load-dispatching strategies and in parallel systems with different sizes by simulation based on the loosely-coupled multi-processor model. Benchmark is 6-queens.

3.1 Simulator overview

Simulation is made on the PIM-R hardware simulator composed of 16 MC68000 [Sugie et al. 85], using an interpreter for KL1. As the purpose of this simulation is investigation of load balancing in the PIM bunch layer, detailed structure and operation inside the PIM cluster is not simulated.

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 processor. The timer count renews by adding a certain value every time a transaction of any one of several functions is executed. When messages are sent to other processors, network delay time is added to the timer count, and this value is attached to the sent message to indicate the arrival time. The processor which receives the message updates 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 managed, based on the cluster software timer.

3.2 Conditions

The simulation assumes the following:

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

(2) The processor has a sufficiently large input/output buffer and waiting time, due to the

input/output buffer overflow not being taken into account.

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

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

(5) Built-in predicates are not dispatched to other processors.

3.3 Results

The relationship between utilization and load-dispatching rate in parallel systems composed of 2, 4, 8, 16 and 32 nodes is measured. The load-dispatching rate is defined as the ratio of all goals dispatched to other nodes, to all reduced goals, as defined previously.

Fig. 2 shows the normalized processing time of strategy A, B, B', B " and C as a function of the load-dispatching rate, where the number of nodes is 16. The difference between B and B' is the selection of dispatch goals, which is described in detail in the next paragraph. Strategy B" is a combination of strategy B and the strategy in which goal dispatch is aborted on the condition that dispatching cluster has fewer ready goals than the threshold. The normalized processing time is defined as the ratio of the processing time for plural processors, to the processing time for a single processor. The load-dispatching rate is varied by changing the simulation parameter which controls load dispatch probability.

In order to reduce the minimum load-dispatching rate for load balancing, it is useful to dispatch heavy goals, which need many reductions before no their descendant goals are left. They can improve utilization of the dispatch target node. Fig. 3 shows 6-queens program written in KL1. In strategy A, B, B" and C, "queen(U,[P|C],L,X,O)", underlined in Fig. 3, is dispatched every time it is created. In strategy B', no goal is automatically dispatched, but goals are still dispatched according to the probability. Comparing results of strategies B and B' in Fig. 2, it is shown that heavy goal selection for dispatch reduces the optimum load-dispatching rate to 1/2.

Strategy B" has higher performance than strategy B, when heavy goals are not selected for dispatch [Sugie et al. 88]. However, strategy B" has approximately the same performance as strategy B in case of heavy goal selection for dispatch, as Fig. 2 shows, because a dispatched heavy goal creates more ready goals than the threshold.

The normalized processing time is expressed by

normalized processing time =

(number of clusters) × (average utilization)

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

Fig. 4 shows the parallel processing overhead as a function of the load-dispatching rate, where parameter is the number of processor nodes. This figure indicates that the parallel processing overhead is expressed by a straight line and that it is independent from the number of nodes and is determined only by the load-dispatching rate. As the load-dispatching rate increases, occurrence of process suspension/resumption increases more steely than linear. This tends to give super-linear relationship between parallel processing overhead and the load-dispatching rate. An optimization is introduced into the KL1 interpreter on the simulator which can reduce the communication between nodes by storing values in a node which are instantiated in other nodes and sent to the node through messages. Once such values are stored in the node, no more communication is needed to get them. When load-dispatching rate is low, so few variables are shared between nodes that the above-mentioned optimization is not effective. This tends to give parallel processing overhead sub-linear (increase more slowly than linear) dependency on the load-dispatching rate [Sugie et al. 88]. These two factors are thought to compensate each other and give parallel processing overhead linear dependency on the load-dispatching rate on this simulation condition.

Figures 5, 6 and 7 show the average utilization as a function of the load-dispatching rate for strategies A, B and C. In these figures, average utilization is saturated in the high load-dispatching rate region. Fig. 8 shows the saturated utilization as a function of the system size, which is expressed by the number of nodes. As the system size increases, the saturated average utilization decreases, because 6-queens program do not have sufficient parallelism. The saturated average utilization decrease is approximately same in strategies B and C, and is steeper in strategy A than in those "informed" strategies. Fig. 8 indicates that "informed" load-dispatching strategies B and C can utilize parallelism more efficiently than in "blind" load-dispatching strategy A.

Here, let us introduce the load-dispatching rate limit (d_{lim}), defined as the load-dispatching rate which gives 80 % of the saturated average utilization. This load-dispatching rate limit can express the minimum load-dispatching rate for load balancing. Fig. 9 shows the load-dispatching rate limit as a function of the system size. In Fig. 9, the load-dispatching rate limit has sub-linear dependency on the system size in three load-dispatching strategies A, B and C and its gradient is highest in strategy A, medium in strategy B and lowest in strategy C. When the system size is 2 nodes, these two strategies have the same load-dispatching rate limit, because they are the same. In the 2 node configuration, one node is the dispatcher and the other is the dispatch target. The dispatcher in strategy B has more ready goals than the dispatch target, and so, it is the node with maximum load in strategy C.

4 DISCUSSION

The load-dispatching rate limit determines how many goals must be at least dispatched during a certain period, so as to maintain the average utilization during this period as high as the past periods. Suppose that no goal dispatch during a certain period would make one busy node idle. Then, a goal must be dispatched to this node or $n \cdot (1 - u)$ nodes which are idle, where n and u the number of nodes and the saturated average utilization.

In strategy A (baka-random), dispatch target nodes are determined at random and goals are dispatched regardless of whether the dispatch target nodes have more ready goals than the dispatching nodes or not. Therefore, probability with which a goal is dispatched to one of those $n \cdot (1 - u) + 1$ nodes is $\{ n \cdot (1 - u) + 1 \} / n$, and so, $n / \{ n \cdot (1 - u) + 1 \}$ goals must be dispatched to maintain the average utilization. Length of the period during which no goal dispatch would make one busy node idle is proportional to the reciprocal of $n \cdot u$, the number of reductions during unit time is proportional to $n \cdot u$, and so, the number of reductions during that period is constant, namely, independent from n and u. Therefore, the load-dispatching rate limit in strategy A is expressed by

load-dispatching rate limit $=$ const. $\times\, n / \{ n \cdot (1 - u) + 1 \}$. $\cdots$ (2)

In strategy B (smart-random), dispatch target nodes are determined at random but restricted to nodes with more ready goals than the dispatching nodes. Strategy B has higher probability with which a goal is dispatched to suitable nodes than strategy A. In the first order approximation, this probability may be expressed by constant $\times \{ n \cdot (1 - u) + 1 \} / n$, and then, the load-dispatching rate limit is expressed by

load-dispatching rate limit $=$ c' $\times$ const. $\times\, n / \{ n \cdot (1 - u) + 1 \}$, $\cdots$ (3)

where c' is a constant less than 1.

In strategy C (max-min), the only node with maximum ready goals can dispatch goals. Suppose such a special situation as one node keeps maximum ready goals and delivering goals to the other nodes. This assumption may approximate well real operation of a PIM using this load-dispatching strategy, because the initial query is usually assigned to one node and new goals created from it are dispatched to the other nodes. Assuming that all dispatched goals have the same lifetime, which is defined as the average reduction counts before their descendant goals are left, $n \cdot u - 1$ goals must be dispatched during t_l so as to maintain the average utilization during this period as high as past periods, where t_l is the lifetime of the dispatched goal. The number of all reduced goals is equal to $t_l \cdot n \cdot u$. Therefore, the load-dispatching rate limit is expressed in this case by

load-dispatching rate limit $= (n \cdot u - 1) / n \cdot u \cdot t_l$. $\cdots$ (4)

Results in Fig. 9 fit expression (2), (3) and (4).

The load-dispatching rate limit may also be assessed from the point of view of 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. If communication processing ability is not sufficient to the load-dispatching rate limit which is determined from the point of view of load balancing, the load-dispatching rate limit must be reduced. The system size dependency of the load-dispatching rate limit shown in fig. 9 suggests important directions.

The line for strategy C in Fig. 9 and expression (4) indicates that the load-dispatching rate limit in strategy C is expressed by the function of the system size which quickly saturate as the system size increases. Therefore, it cannot be reduced so much by decreasing the system size. For example, in Fig. 9, the load-dispatching rate limit is reduced to 90 % of the value in 32 node case by decreasing the number of nodes from 32 to 16. When a program has sufficient parallelism, the saturated average utilization becomes 1. By substitution of 1 into u in expression (4), is given

load-dispatching rate limit $= (n-1)/n \cdot t_1 \cdot \cdots (5)$

This function gives approximately constant load-dispatching rate limit in large n region. In strategy C, task division techniques are not useful to adjust the minimum load-dispatching rate for load balancing to communication processing capacity. It can be adjusted only by dispatching goals with long lifetime.

On the contrary, the load-dispatching rate limit can be reduced by decreasing the system size in strategy B. By substitution of 1 into u in expression (3), is given

load-dispatching rate limit $= c' \times \text{const.} \times n \cdot \cdots (6)$

In the high saturated average utilization case, the load-dispatching rate limit is proportional to the system size, as shown in expression (6), and half of the load-dispatching rate can be achieved by decreasing the system size to half. In strategy B, task division techniques can be applied to adjust the minimum load-dispatching rate for load balancing to communication processing capacity.

5 CONCLUSION

The influence of the PIM bunch size on the load-dispatching rate limit for the bunch layer load balancing was measured and analyzed in baka-random, smart-random and max-min strategies. The load-dispatching rate limit has sub-linear dependency on the bunch size in smart-random and max-min strategies. By decreasing the bunch size, it cannot be reduced so much in max-min strategy, but can be reduced in smart-random strategy. In smart-random strategy, task division and assignment to sub-bunches can adjust the load-dispatching rate limit for load balancing to communication processing capacity. In this adjustable feature, smart-random strategy is preferable to max-min strategy.

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] T.Yamauchi and H.Tanaka, "An evaluation of the load balancing mechanism for a parallel inference machine," IEICE Technical Report, CPSY-23, (in Japanese), 1988

[12] 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

[13] M.Sugie, M.Yoneyama, N.Ido and T.Tarui, "Load-dispatching strategy on parallel inference machines," Proceedings of FGCS'88 (to be published in December, 1988)

[14] 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

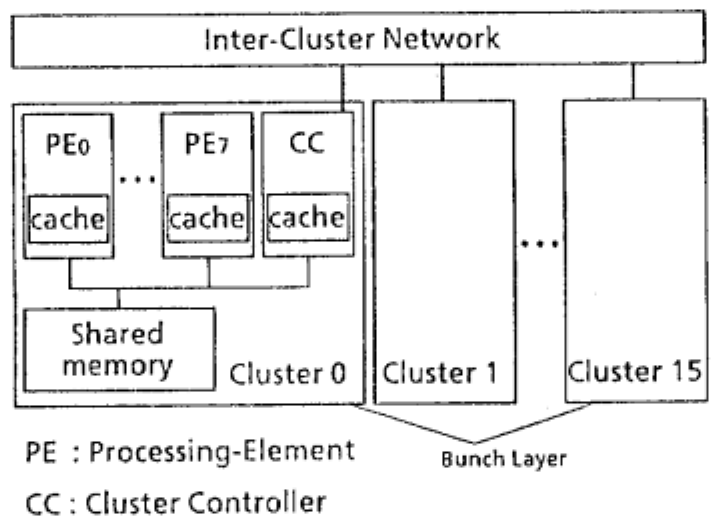


Fig. 1 PIM prototype organization

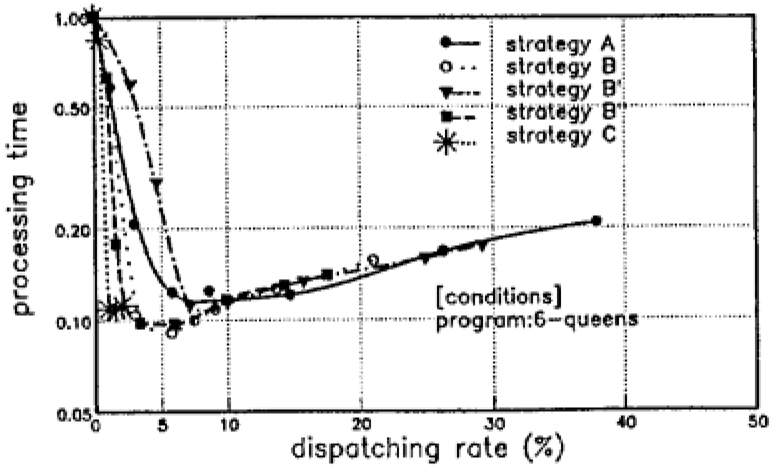


Fig.2 Processing time as a function of dispatching rate

```
go:- true | queen([1,2,3,4,5,6],[],[],X,[]),outstream(X).
append([A | X],Y,Z):- true |
        Z = [A | Z1] , append(X,Y,Z1).
append([],Y,Z):- true | Z = Y.

queen([P | U],C,L,I,O):- true |
        append(U,C,N) , c1(P,1,N,L,L,I,X) ,
        queen(U, [P | C],L,X,O).
queen([],[A | B],C,I,O):- true | I = O.
queen([],[],L,I,O):- true | I = [L | O].

c1(T,D,N,[P | R],B,I,O):- T = ¥ = P + D,T = ¥ = P-D |
        D1: = D + 1,c1(T,D1,N,R,B,I,O).
c1(T,D,N,[P | A],B,I,O):- T = : = P + D | I = O.
c1(T,D,N,[P | A],B,I,O):- T = : = P-D | I = O.
c1(T,D,N,[],B,I,O):- true |
        queen(N,[],[T | B],I,O).

? :- true | go.
```

Fig.3 6-queens program in KL1

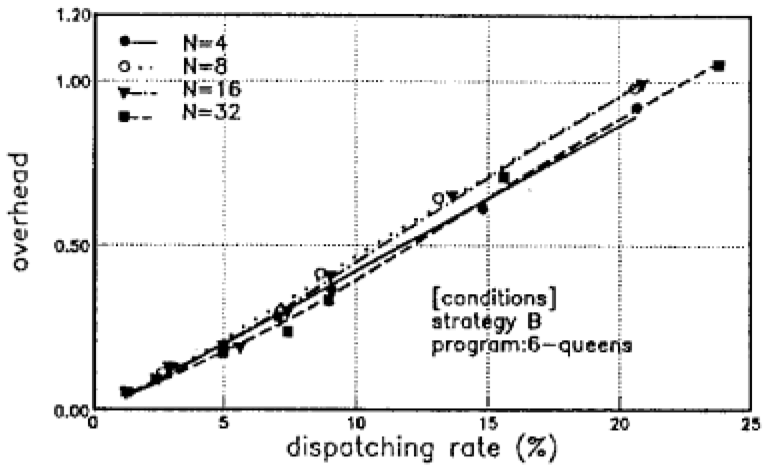


Fig.4 Parallel processing overhead as a function of dispatching rate

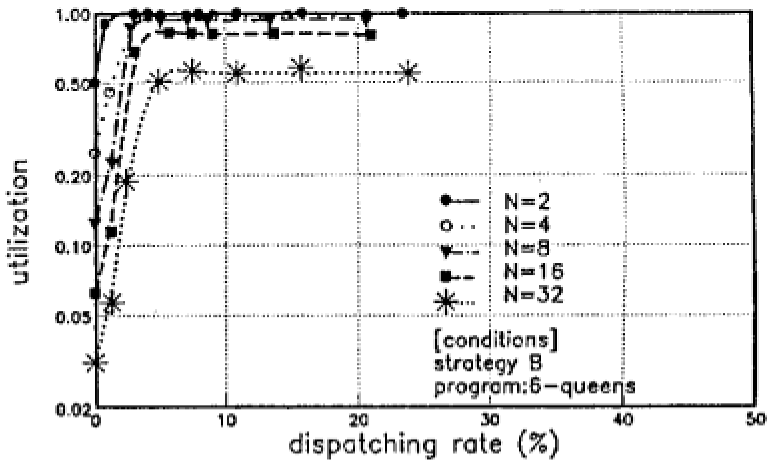


Fig.6 Utilization as a function of dispatching rate

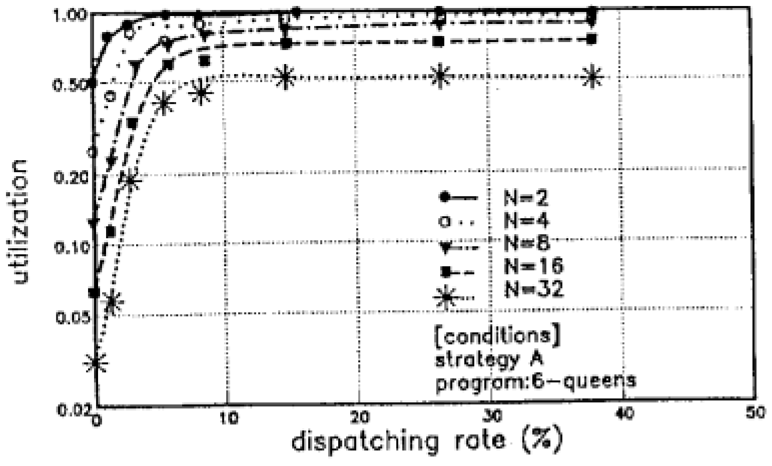


Fig.5 Utilization as a function of dispatching rate

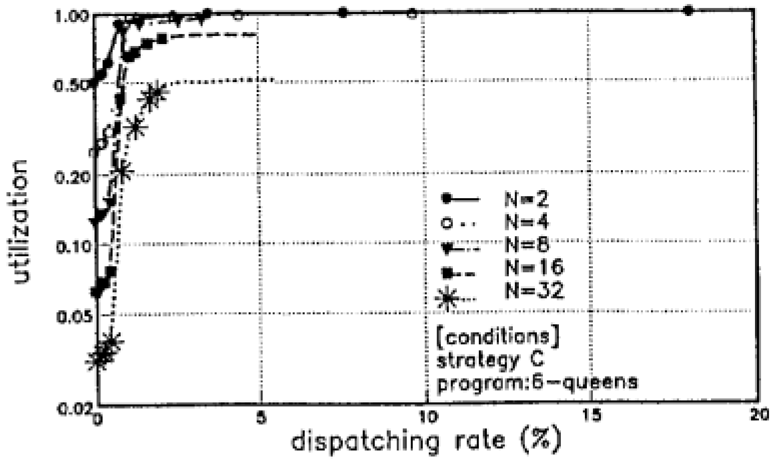
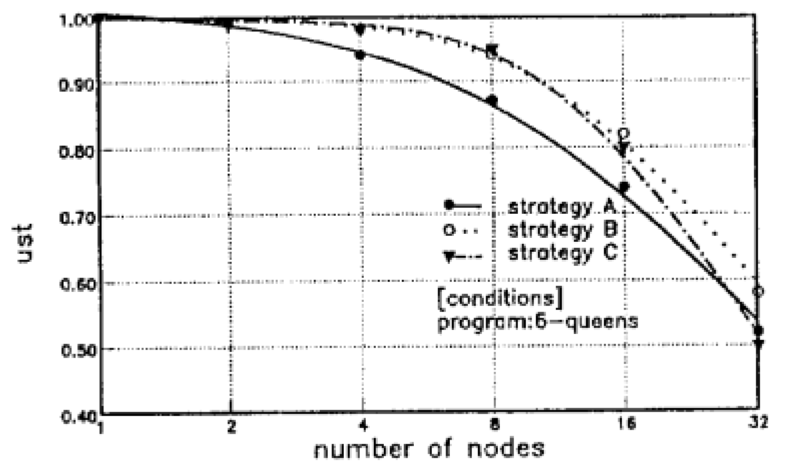
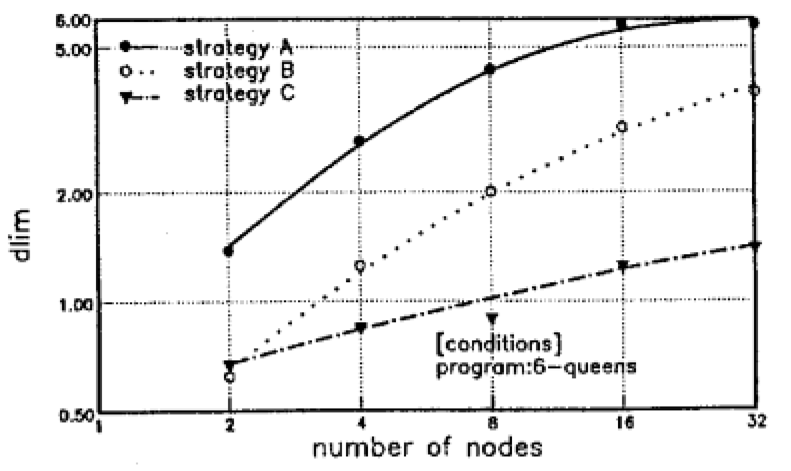


Fig.7 Utilization as a function of dispatching rate



ust :saturated utilization
Fig.8 Saturated utilization as a function of system size



dlim :dispatching rate limit
Fig.9 Dispatching rate limit as a function of system size