TR-704

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

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

October, 1991

© 1991, ICOT

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

Institute for New Generation Computer Technology

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

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

ABSTRACT

This paper proposes and evaluates the hardware implementation required for dynamic load balancing in the prototype PIM/c of the Parallel Inference Machine (PIM).

On one hand, PIM/c is configured along a hierarchical structure of loosely coupled TCMP clusters (TCMP - Tightly-Coupled Multi-Processors) in order to utilize the high locality of logic programs.

On the other hand, an LCMP (Loosely-Coupled Multi-Processors) approach enables us to build a more scalable machine using a crossbar network.

Load balancing algorithms and corresponding hardware suitable for each hierarchy are used.

First, for dynamic load balancing in the TCMP hierarchy, we propose a register with broadcast facility to request load dispatching. The evaluation determines the overhead due to memory polling in order to detect the request. The proposed hardware reduces the execution time of logic programs by 15%.

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

The results obtained confirm that the use of hardware mechanisms reduces the overhead due to the dynamic load balancing.

1. INTRODUCTION

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

Presently, we are developing the PIM/c [4] as a KL1-based machine. Dynamic load balancing is one of the main research areas for PIM [5][6]. As a result of the fact that only logical relations are present in a KL1 program and they never define their process of execution with determinacy, dynamic load balancing must be used in PIM.

The main problems of dynamic load balancing are ensuring a high processor utilization and simultaneously reducing the overhead. The high overhead of load equalization can be reduced by minimizing the data needed. Another possible solution exists in utilizing the locality of KL1 programs. The locality could restrict the interactions to groups of several processors and thus reduce the communications among groups. Thus, the hardware can be concentrated in a limited numbers of processors. In this way, a double hierarchical (both TCMP and LCMP) organization is used in PIM/c. Consequently, we must use a suitable load balancing algorithm in each hierarchy.

In small scale parallel machines like the TCMP part of PIM, a receiver-initiate algorithm is suitable because there is no wasted dispatching. In such machines the main problem is the way to implement a broadcast facility in order to avoid centralization of load request.

In large scale parallel machines like the LCMP part of PIM, a sender-initiate algorithm is suitable because the use of broadcasting would demand an exagerated throughput. In such machines, the problem is reducing the overhead to collect the load information of the receiver.

Consequently, we propose and evaluate appropriate hardware mechanisms for dynamic load balancing in each hierarchy. Thus, we propose a register with a broadcast facility to request load dispatching in TCMP, and a shortcut path in the LCMP hierarchy to reduce the overhead related to requesting the value of a total load.

We evaluate the real hardware in order to determine the critical interactions in the cache-processor-network complex. We carried out the evaluation using an artificial load model in order to focus on the speedup produced by the proposed hardware mechanisms.

2. OUTLINE OF PIM/c HARDWARE FEATURES

PIM/c has the following distinguishing hardware features:

A. Hierarchical structure of TCMP and LCMP.

Figure.1 shows the configuration of PIM/c. PIM/c is organized along a hierarchical structure of loosely coupled TCMP clusters to utilize the localities of KL1 programs.

Thus, the TCMP hierarchy consists of processors combined in a cluster. Each processor has its own cache, and they share a common bus. It has been proven by software simulation that the common bus might be a bottleneck. We concluded that the number of processors in a cluster should be limited to approximately eight, and that the two-way-interleaved common bus [7] should be possible in PIM/c.

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

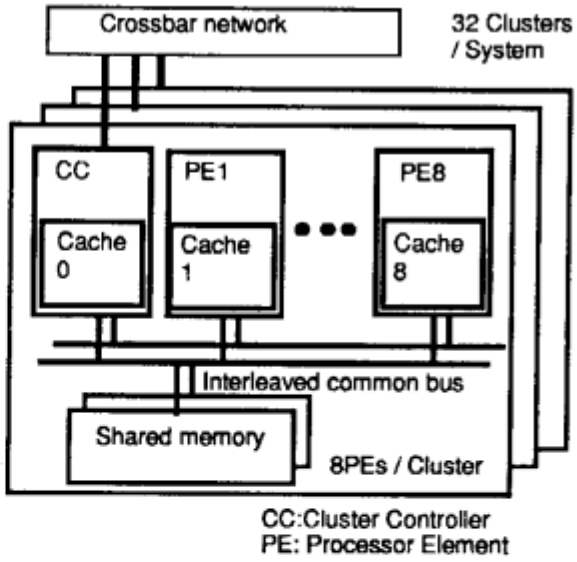

Fig.1. The configuration of PIM/c.

B. Snooping cache.

In order to utilize the access locality in TCMP, PIM/c uses a snooping cache, as this is

known to reduce the common bus usage [8], using a bit to denote the shared state of data.

There are two types of cache coherency protocols: invalidation-type protocols and broadcast-type protocols. The PIM/c cache protocol is a invalidation type protocol similar to the Berkeley protocol [9][10]. For KL1 programs, invalidation protocols are more suitable than broadcast protocols from the stand point of the maintenance cost of shared data. There is no need to rewrite data distributed on other caches. Because of the high access locality and the high write ratio of KL1 programs, the data on other caches will not be used.

C. Crossbar network.

In order to obtain a high throughput in LCMP, a crossbar network is employed in PIM/c [11]. Using this crossbar network, each processor can communicate with other processors with a throughput of 20Mbytes/s independently of other communications. In order to maintain this throughput, message queues are also used. However, we have to note that the existence of queues means, on the other hand, an increase in latency.

3. LOAD BALANCING IN EACH HIERARCHY

Because PIM/c is configured in a double hierarchy, both TCMP and LCMP, there must be a suitable load balancing algorithm in each hierarchy.

3.1 Load Balancing in TCMP

Because the response time in TCMP is short, a receiver-initiate load balancing algorithm is suitable. This algorithm avoids the wasted load dispatching by applying the rule "only fully idle processors require load dispatching". In order to implement this algorithm efficiently, we concentrate on reducing the overhead resulting from cache misses.

In the following, we will present several load dispatching schemes for this purpose:

A. The common goal pool scheme.

In this conventional scheme [12], a record of load waiting to be executed will be pooled in one data structure, and any processor will extract the record to execute the load described in it. The contents of a record are transferred through the cache mechanism implicitly by reading them. We identify this data structure as the *goal pool.*

In fine grain multiprocessing like in PIM/c, the use of this scheme causes some problems (Fig.2).

With a invalidation cache like in PIM/c, read accesses will cause cache misses when the processor that is accessing a given record changes.

If PIM/c caches were of a broadcast type, write accesses would cause broadcasting through the common bus.

With a broadcast cache, it would be impossible to avoid the frequent data transfers through the bus anyway.

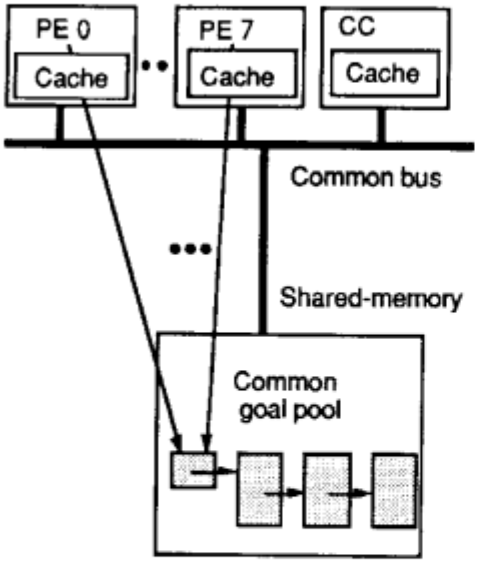

Fig.2. The common goal pool.

The distributed goal pool scheme [13], in which each processor has its own goal pool, on the other hand, can help avoid cache misses for each individual processor. Consequently, an explicit load balancing communication method for the distributed goal pools should be introduced in the TCMP hierarchy. This algorithm is based on requesting a load dispatching. The cache miss overhead is thus reduced to be proportional to the number of loads that were transfered by the explicit communications.

The next problem is related to the communication style in a KL1 system. In a KL1 implementation, events can interrupt processors only in a gap between load executions for which the overhead of context switches is low. Therefore, if a load request is sent to a specified processor, it takes many cycles (100 cycles in PIM/c on average) to respond to the request. In order to shorten the response time, a new type of communication, the AR (Arbitrary Responder) communication was introduced in PIM/c [14].

Fig.3 presents the concept of AR communication. This communication is sent to any processor which has more than one load in its goal pool. The responder is defined as the processor which detects the communication first. As the timing to detect requests differs in each PIM/c processor, this communication method is expected to shorten the response time proportionally to the number of processors in TCMP.

In order to implement this communication method, there are 2 implementation alternatives: memory polling and register polling.

Fig.3. Arbitrary responder communication.

B. The memory-polling scheme.

This new scheme uses a bit-map data. A bit-map pool is a data array in which each bit corresponds to a processor requesting load. This array is located in the shared memory, and is updated using exclusive access methods: Fetch-and-Or and Fetch-and-Zero [15]. All processors perform the following sequence:

- Idle processors set their corresponding bits in the bit-map word using a Fetch-and-Or operation.
- All but the idle processors are polling the bit-map word, and are checking whether the value of the bit-map is all-zero or not.
- If one processor detects a non-zero value for the bit-map word, that processor will be a responder. It keeps the value into its register, and clears the bit-map word using a Fetch-and-Zero operation.
- The responder sends one record in its load pool per one idle processor using a ordinary communication method with its responder specified.

- If the number of loads in goal pool of the responder becomes one, the bit-map word will be updated with the value of bit-map in the register using a Fetch-and-Or operation.

Two problems exist in the memory polling scheme. One is the overhead to read and check the bit-map word serially. The second is the overhead produced by cache misses in all irrelevant processors after each communication is finished.

C. The register-polling scheme

This is another new scheme using additional special purpose registers dedicated for interprocessor communications [14], as well as using the load request. In order to reduce the polling overhead, those registers have a broadcast feature. The reason for using registers is that registers can be easily implement the broadcast feature through the common bus. We denote these registers as RFR (Request Flag Registers). These registers have the following features:

- They have a one-bit width to indicate a request, and one of them in each processor is dedicated for the load request.
- They can be read only by their corresponding processor, and they can be written by any processor in the cluster.
- They also have the facility of broadcast write; therefore, a load request register in all processors can be written simultaneously.

As shown in Fig.4, the AR communication can be established by the following sequence:

- An idle processor sets simultaneously the load request bit in all processors using the broadcast feature.
- The earliest processor detecting a request resets simultaneously the request bit in all processors using the broadcast feature, and performs dispatching using the bit-map data in the shared memory.

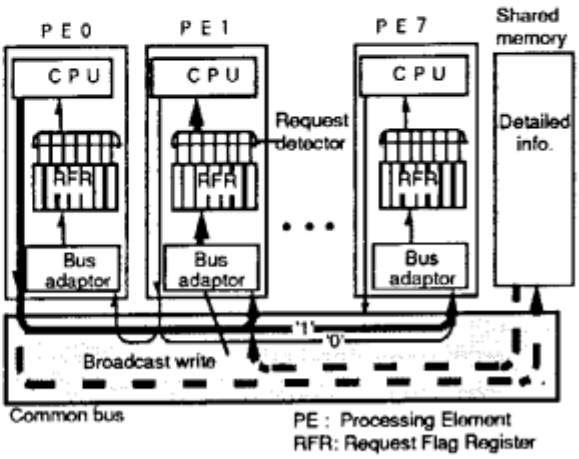

Fig.4. Dynamic load balancing support in TCMP.

When using registers, there is no overhead due to cache misses. Each PIM/c processor has 16 RFR registers, and also has a fast detection feature of control jump just in case one of RFRs corresponding to the processor is set to one.

3.2 Load Balancing in LCMP

Because implementing the broadcast feature in the LCMP context via the network demands an exagerated throughput, we adopted a sender-initiate load balancing algorithm. Sender-initiate algorithms involve wasted dispatchings, but simulation results proved that the "Smart Random Load Dispatching" [6] scheme, which uses a sender-initiate algorithm, works efficiently using local load information. Fig.5 illustrates the concept of this scheme.

Fig.5. Smart random load dispatching.

In order to avoid wasted dispatchings, the "Smart-Random" scheme requires the load amount related to a specified cluster. The scheme consists of the following sequence:

- The sender processor determines a receiver candidate randomly.
- The sender processor inquires about the load amount of the receiver candidate.
- If the load amount is less than that of the sender processor, load dispatching is carried out. Otherwise, dispatching is stopped.

In order to reduce the overhead due to requesting the load amount, a shortcut path for the messages inquiring load amount in comparison to the ordinary messages is introduced in the network (Fig.6) [11]. The hardware used for load balancing in LCMP has the following features:

- A shortcut path to message queues.
- Eight-bit wide registers to indicate the total load in a corresponding cluster.

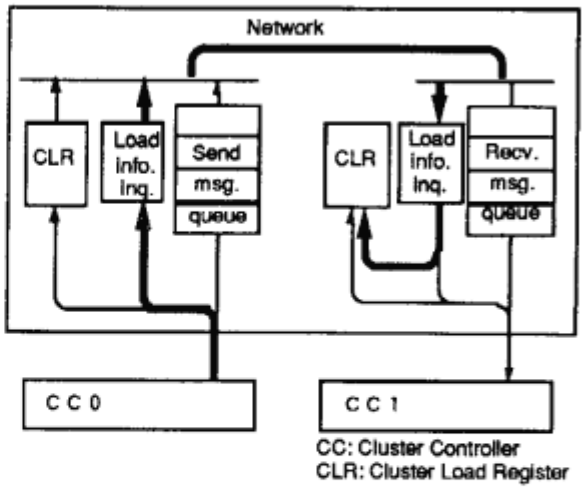

Fig.6. Dynamic load balancing support in LCMP.

The register should be written with the total load amount by its corresponding processor.

As the load amount is inquired without waiting at message queues and without waiting for the cluster controllers to receive, specified registers can be read in constant time.

4. EVALUATION STRATEGY

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

4.1 Evaluation on the Real Hardware

We used real hardware for evaluation purposes as the software simulation is almost impossible for the following reasons:

- The cache and the network are present and many parameters are in effect.

  There are many hardware parameters related to the internal states of the cache and network. The common bus arbitration time, and the message packet switching time are examples in this respect. The overhead of cache misses and the network latency is important in this evaluation. Thus, covering the cache and network effects would have

taken a good deal of time in software simulation concurrently with processor activities.

4.2 Evaluation using an Artificial Load Model

We carried out the evaluation on an artificial load model for the following reasons:

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

5. EVALUATION RESULTS

We carried out the evaluation of the proposed load balancing hardware in both TCMP and LCMP hierarchies.

5.1 Evaluation in the TCMP hierarchy

We carried out this evaluation focusing on the effect of cache misses.

A. The load model.

This model reflects the following characteristics of KL1 program executions:

- A unit load.
  We denote the unit as the *reduction*. We assume that the unit is approximately 200 cycles in PIM/c.
- Indeterminacy in the granularity of loads.
  We define the *goal* as consisting of an arbitrary number of reductions (1 to 16).
- Indeterminacy in the number of goals.
  We assume that each processor generates an arbitrary number of goals (1 to 4096).
- A high write ratio and a high share ratio.
  Accesses performed within the reductions have the following parameters: write ratio - 0.5, share ratio - 0.5.
  With the common pool scheme, the share ratio is set to 1.0, and the size of the working set is multiplied by the number of processors in a TCMP, in order to simulate the overhead of accessing the common goal pool.
- A high access locality.
  In our experiment, we define the locality as a number of successive accesses to the same address. The value is set to 4 in order to simulate free list manipulation in KL1, which consists of allocating, instantiating, referring and deallocating a memory cell.

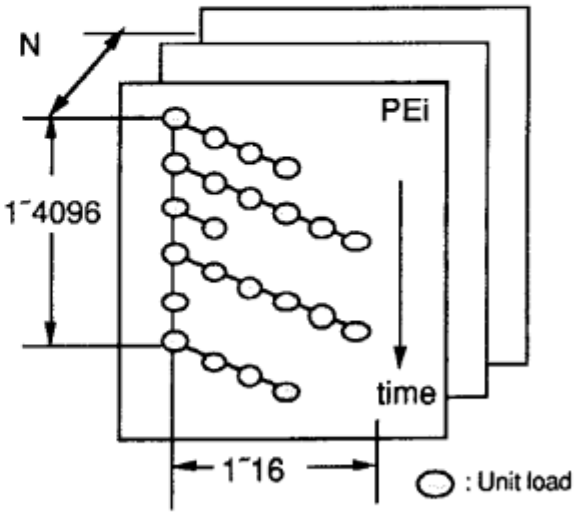

Fig.7. A Load model with variable granularity.

B. Load balancing schemes.

In order to determine the effectiveness of the proposed hardware, we carried out the evaluation using the following five load balancing schemes:

- A STATIC_DIST (static distribution) scheme.
  We assume a static load distribution of loads produced by a compiler. No dynamic load balancing is performed with this scheme.
- A COMMON_POOL scheme.
  As described before, a common goal pool is assumed with this scheme.
- A DIST_POOL (distributed pool) scheme.
  With this scheme, the distributed goal pools are introduced without the AR communication method. The responder is selected as a non-idle processor using bit-map data and is specified in the memory area where its processor number is written.
- A MEM_POLL (memory-polling) scheme.
  As described before, we assume the use of the distributed goal pools and the AR communication method.
- A REG_POLL (register-polling) scheme.
  As described before, in this scheme, each processor polls the special-purpose register with broadcast facility in order to reduce the overhead due to the memory-polling.

C. Results of the evaluation at the TCMP level.

We control the initial load amount in each processor. According to the variation of the initial load amount, 14 cases are simulated on each load balancing scheme with a 8 processor cluster. The resulting data are the total elapsed time (T), the total idle time (I), the total wait time after requesting for load (i), the total dispatching time (t), the total reduction count (R) and the load request count (r).

The following measures are defined:

Nominal reduction cost = T / R .....(5.1.1)

Reduction cost = (T - I - t) / R .....(5.1.2)

Utilization = (T - I) / T .....(5.1.3)

Fig.8 shows the effect of the utilization on reduction cost. The nominal reduction cost with the five above mentioned schemes is plotted as a function of utilization. The nominal reduction cost varies related to the utilization. However, the utilization is not the only significant parameter in TCMP schemes. Fig.8 shows that the results of applying the five schemes can be classified into the following three groups:

- Group I: This group corresponds to applying only one scheme, the COMMON_POOL scheme. The results indicates the overhead due to cache misses compared with the results of the DIST_POOL scheme.
- Group II: This group consists of results corresponding to the MEM_POLL and the DIST_POOL schemes. The corresponding results indicate the overhead due to the memory-polling compared with the results of the REG_POLL scheme.
- Group III: This group consists of results corresponding to the REG_POLL and STATIC_DIST schemes. Note that the REG_POLL scheme is almost free from both polling overhead and cache misses compared with the results of the STATIC_DIST scheme.

As a result of this experiment, it is confirmed that the overhead caused by memory-polling and cache misses is not negligible, and can be reduced using the proposed registers.

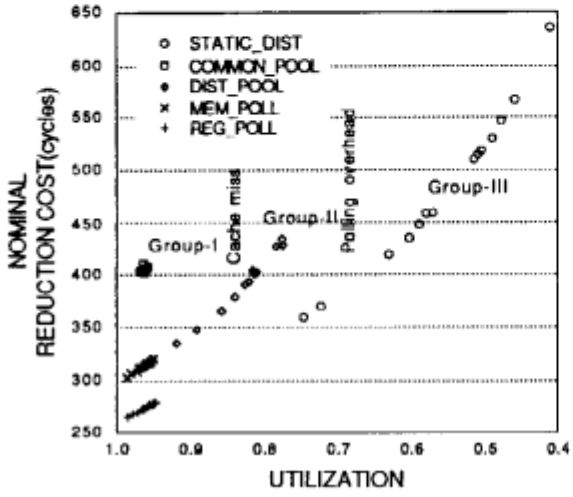

Fig.8. The effect of utilization.

Fig.9 shows the effect of the memory-polling. In relation to the request count, the nominal reduction cost is almost unchanged. This fact indicates that the memory-polling overhead caused by checking request occurrences is larger than the overhead due to cache misses caused by both the load requesting and the load dispatching.

The speedup obtained is 15% in comparison to the MEM_POLL scheme and 30% in comparison to the COMMON_POOL scheme.

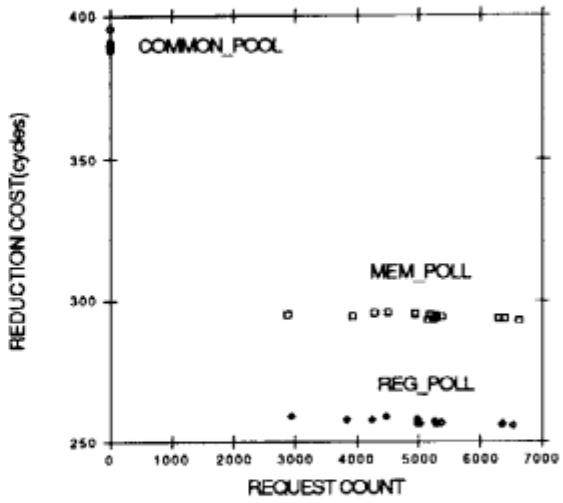

Fig.9. The effect of memory polling.

Fig.10 presents the effect of broadcasting. Both of the wait time and the dispatching time are plotted as a function of request count. It is confirmed that RF (Request Flag) registers with a broadcast feature reduce both the wait times and dispatching times. The RF registers reduce the dispatching overhead by 20%, and reduce the idle time by 15%, respectively.

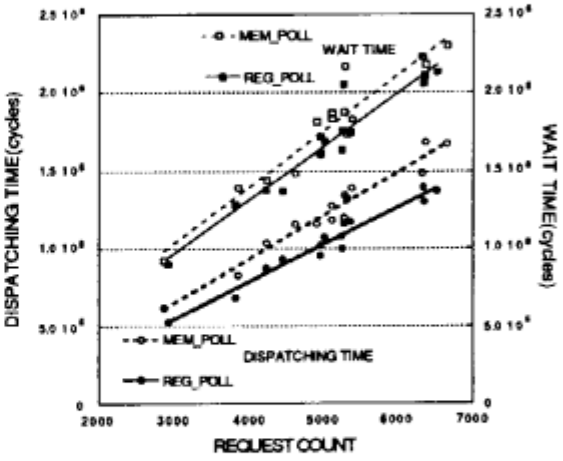

Fig.10. The effect of broadcasting.

5.2. Evaluation in the LCMP hierarchy

In the LCMP hierarchy, the evaluation is focused onto the effectiveness of avoiding the wasted dispatchings by using the correct load information.

A. The Load model.

The load model in LCMP is defined in a way as to reflect the changes in the number of load in the goal pool. The load model is as the following:

- An initial goal is denoted as L(16) (Fig. 11).
- The execution of goal L(i) produces (i-1) subgoals, L(i-1),..., L(1), L(1). Thus, the goal L(i) has 2^{i-1} reductions.
- Each reduction takes 300 cycles to execute using network messages.
- The message length required for the load dispatching is 27 bytes long. Thus, it takes 27 cycles to send this message through the 9 bit-wide network interface. The message length requesting the load amount is 2 bytes long.

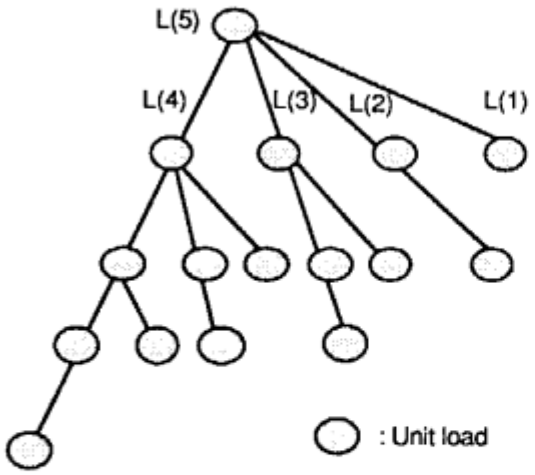

Fig.11. A load model with floating load amount.

B. Load balancing cases.

We assumed the use of the "Smart Random Dispatching" [6]. There are only two cases with respect to the use of the proposed hardware mechanisms:

- the case with hardware.
- the case without hardware.

C. Results of the LCMP level evaluation.

We control the dispatching rate by changing the interval of the dispatching control. In order to determine the efficiency of load dispatching, the total elapsed time (T), the total idle time(I) and the dispatching rate (d) are measured. As the load balancing algorithm is the same for the two schemes except for the hardware support, differences in the results are produced only by the latency in control information.

Fig.12 shows the results obtained by applying the smart random load dispatching scheme without its hardware support. The processing time and the utilization are plotted as a function of the dispatching rate. The dispatching rate is defined as the ratio of all goals dispatched to other cluster to all executed goals. In order to compare the results in the two cases, we assume that the dispatching rate is controlled to be 20%. Because the safe control exists in the upper side of the minimum point. Without the hardware support, the resulting speedup is approximately 3.3 at the point of 0.2 dispatching rate.

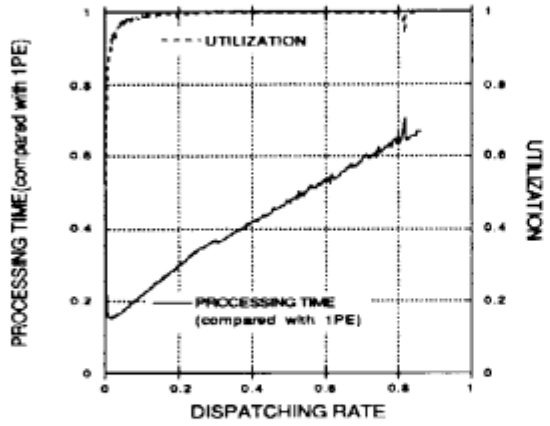

Fig.12. Smart random dispatching without its hardware support .

Fig.13 shows the results applying the smart random load dispatching scheme with its hardware support. The processing time and the utilization are plotted as a function of the dispatching rate. With the hardware support active, the processor can reduce the overhead due to requesting the load amount. The resulting speedup is approximately 5.5 at the point of 0.2 dispatching rate.

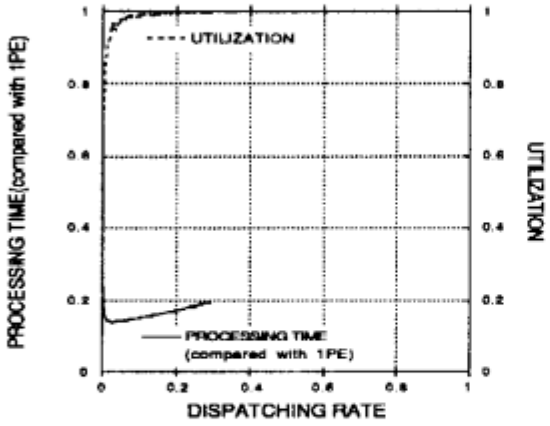

Fig.13. Smart random dispatching with its support hardware.

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

We must note that the shortcut path can be also used for other load balancing schemes, including the minimum load distribution scheme [16]. The evaluation of these schemes is under contemplation.

6. CONCLUSION

Hardware implementation of dynamic load balancing is proposed in both TCMP and LCMP.

We propose a register with broadcast facility to request load dispatching in TCMP. Also, in LCMP, the network unit uses a shortcut path to request the value of a total load register. The evaluation was carried out on the real hardware using an artificial load model.

The evaluation results in the TCMP hierarchy determine the overhead due to memory polling in order to detect the request. The proposed hardware reduces the execution time of logic programs by 15%.

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

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

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Shun'ichi Uchida, the manager of the research

department of ICOT, for his guidance and support, and Dr. Kazuo Taki, chief of 1st ICOT laboratory, for helpful discussions. This research was sponsored by ICOT.

REFERENCES

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

[2] K. Ueda , "Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard," TR208, ICOT, 1986.(Also in Programming Feature of Future Generation Computers, North Holland, Amsterdam, 1987.).

[3] T. Chikayama, H. Sato, T. Miyazaki, "Overview of the Parallel Inference Machine Operating System (PIMOS)," Proc. of the FGCS, vol.1, 1988, pp. 230-250.

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

[5] K. Kumon, H. Masuzawa, A. Satoh, Y. Sohma, "A new Parallel Inference method and its Evaluation," COMPCON Spring 86, IEEE Computer Society, San Francisico, Mar, 1986, pp 168-172.

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

[7] Radolph and Z. Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," Proc. of the 11th ISCA, June, 1984, pp. 340-347.

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

[9] P. Bitar and A. M. Despain, "Multiprocessor Cache Synchronization Issues, Innovations, Evolution," Proc. of the 13th ISCA, June, 1986, pp 424-433.

[10] T. Tarui, N. Ido, H. Maeda, T. Nakagawa, M. Sugie, "Parallel Inference Machine PIM/c - Snooping Cache Organization -," the 40th Annu. Convention IPS Japan, 2L-3, (in Japanese) .

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

[12] IBM System/370 Extended Architecture, "Principles of Operation," IBM manual, SA22-7085-0, pp. A-14.

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

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

[15] H. S. Stone, "Database Applications of the FETCH-AND-ADD Instruction," IEEE Trans. on Computers, Vol. C33, No. 7, July, 1984.

[16] S. Sakai, H. Koike, H. Tanaka, T. Motooka, "Interconnection network with

dynamic load balancing facility," Trans. of Information Processing, Vol. 27, No. 5, pp. 518-524, 1986, (in Japanese).