TR-138

Architecture and Evaluation of a Reduction-Based Inference Machine: PIM-R

by
R. Onai (NTT),
K. Masuda (Mitsubishi Electric Corp.),
H. Shimizu, A. Matsumoto and M. Aso

September, 1985

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

Architecture and Evaluation of a Reduction-Based Parallel Inference Machine : PIM-R

Rikio ONAI[1] , Hajime SHIMIZU, Kanae MASUDA[2] , Akira MATSUMOTO and Moritoshi ASO

Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F, 4-28, Mita 1-Chome, Minato-ku, Tokyo 108, Japan

1 INTRODUCTION

This paper proposes a Reduction-based Parallel Inference Machine : PIM-R and describes the parallel machine architecture of PIM-R and the evaluation using software simulators.

Currently there are several proposals for parallel inference machine architecture (Moto-oka 84 ; Ito 84) and predicate logic languages (Shapiro 83 ; Clark 84 ; Pereira 84). We have chosen Prolog and Concurrent Prolog (Shapiro 83) as the target languages of PIM-R. These have been selected by ICOT as the base languages for its Kernel Language version 1 (KL1(84)). The basic operation of PIM-R consists of parallel generation of new resolvents. PIM-R executes Prolog programs in OR parallel and Concurrent Prolog programs in AND parallel. In PIM-R, if a process has multiple goals (the multiple goals, as a whole, are called the parent process), only the reducible goals, specified by various operators, are copied and reduced. Each resolvent generated contains a pointer to its parent process; the solution obtained is returned to the parent process using the pointer. That is, PIM-R executes Prolog and Concurrent Prolog programs by expanding and reducing a process tree. When the processing ends, the tree is logically deleted.

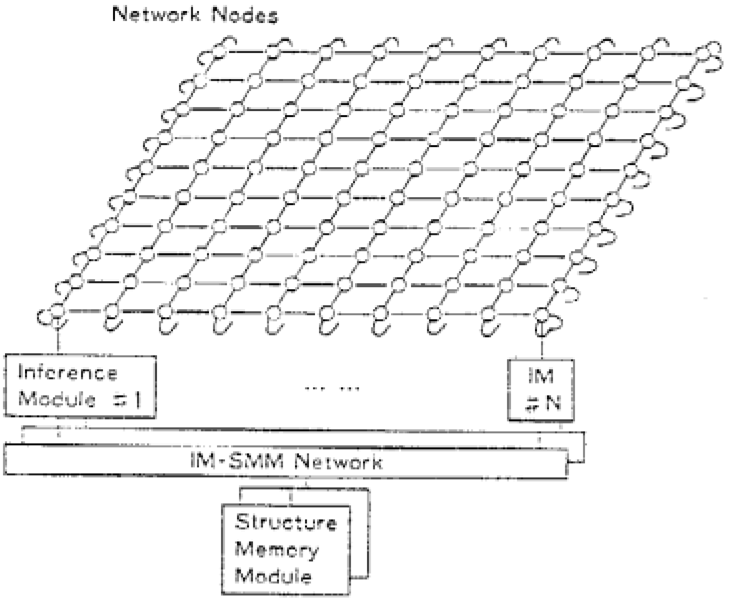

Fig. 1 Conceptual configuration of PIM-R

Present Address :
1.NTT Musashino ECL, Communication Principles Reseach Section 2,
3-9-11 Midori-cho Musashino-shi Tokyo 180 Japan
2.Mitsubishi Electric Corporation, Computer Works, New Product Development Dept.
325 Kamimachiya Kamakura-shi Kanagawa 247 Japan

2 PIM-R ARCHITECTURE

PIM-R uses the structure-copy method to increase the independence of individual processes and decrease the network traffic due to structure sharing. It also uses the only reducible goal copy method and a unique process-structuring method to decrease the amount of copying, and the number of packets passing through the network. PIM-R architecture features include the distributed shared memory for Concurrent Prolog, network nodes for efficient packet distribution, and a structure memory to store a part of the structured data to reduce copying overhead. As shown in Figure 1, PIM-R basically consists of two types of modules, an Inference Module and a Structure Memory Module, besides networks connecting these modules.

2.1 Inference Module (IM)

The Inference Module (Fig. 2) consists of two units: the Unification Unit (UU) and the Process Pool Unit (PPU).

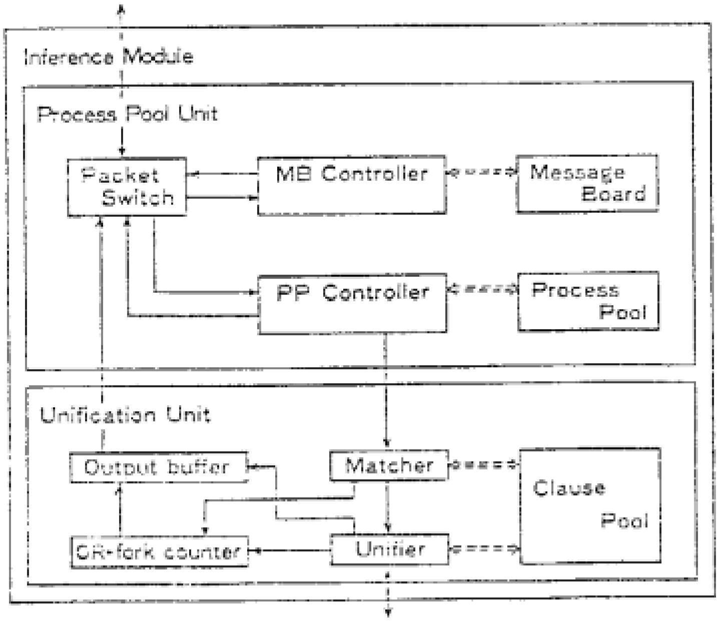

Fig. 2 Inference Module Configuration

2.1.1 Unification Unit (UU)

(1) Clause Pool (CP)

The Clause Pool (CP) in each IM stores the same clauses. Each word in CP is 32 bits. The Clause Pool consists of a Clause Definition Group Management Block and a Clause Definition Block. The Clause Definition Group Management Block stores the number of clauses in OR-relation, a pointer to the Clause Definition Block where each clause is stored, and the data type of the first argument of the head literal of a clause. The Clause Definition Block stores the definition of a clause and consists of a header, a variable area, a literal header, a literal area, and a structure area (Fig. 3).

(2) Matcher and Unifier

The Matcher chooses unifiable clauses according to the data type of the first argument of the goal passed from the Process Pool Unit. The Unifier stores the goal sent from the Matcher in the goal memory and a clause copied from the CP in the clause memory, unifies the goal with the clause, generates the final result in the goal memory, and passes it to the output buffer. Also the Unifier executes built-in predicates.

	Int	Clause length	Header
#0	Int	Head address of structure area	
	Int	Head address of literal header	
	Int	Number of variables	Variable area
		:	
	Int	Literal count	Literal header
	Type	Head literal	
	Type	Body literal N	
		:	
	Type	Body literal 1	
	:		Literal area
	:		Structure area

Fig. 3 The configuration of the Clause Definition Block (Type is either Poi,Lit,Sym, or Para)

2.1.2 Process Pool Unit (PPU)

The Process Pool Unit (PPU) consists of two types of memories (Process Pool and Message Board) and two types of controllers (Process Pool Controller and Message Board Controller).

(1) Process Pool and Process Pool Controller

The Process Pool (PP) is a memory for storing processes (32 bits/word, the same as the Clause Pool). PIM-R employs a process configuration method capable of minimizing communications between IMs. A process consists of Process Control Blocks (PCB) for storing control information of a goal sequence, a Process Life Block (PLB) to manage the number of Process Control Blocks, and Process Template Blocks (PTB; a PTB and a PCB makes a pair) to hold the template of a goal sequence. These blocks are assigned to an IM as a set.

The PLB is the top level block in a process and contains the commit tag, the number of PCBs under the PLB, and other information. At Concurrent Prolog execution, the commit tag is turned on by the PCB in the process that first succeeds in executing its guard part.

The PCB contains the state of a goal sequence (reducible (ready), run (unification is under way), wait (waiting for a solution to be sent from a child process), dead, or suspend (consumer process is waiting for a shared variable to be connected with a value)), reduction level, number of OR forks (when an OR-parallel Prolog program is executed) or AND forks (when a Concurrent Prolog program is executed), number of returns (i.e., return from forked processes), a pointer used to connect to the Ready Process Queue, etc. The reduction level means the relative depth of each process, assuming that the depth of the process which corresponds to the root of the process tree is 1. When under a PCB, a PTB has the same internal format as clauses in the Clause Pool (CP) except for the clause length. Simply put, a reducible goal in a PTB is sent to the UU, and when its solution is returned to a PCB, the goal sequence in the corresponding PTB is copied, the binding environment is assigned, and a new goal sequence (a PCB-PTB pair) is generated under the same PLB.

Process Pool Controller (PPC) is responsible for process creation, renewal, and deletion. When a new resolvent is returned from the UU, PPC creates a new process which consists of a PLB and a PCB-PTB pair. Process renewal is the updating of the fork count and return count and the creation of a new PCB-PTB pair. When a PCB enters the dead state (i.e., fork count = return count), the PPC sets the PCB state dead and increments the PCB return count in the corresponding PLB by 1.

(2) Message Board (MB) and Message Board Controller (MBC)

Each IM has a distributed shared memory called Message Board to store variables used as channels (these variables are called simply channels). The MBC is designed to reduce the load of PPC processing. The MB consists of channel cells, value cells, and a suspend process list. Channel cells consist of four words and contain a write tag, a suspend tag, a pointer to a value cell, the suspend process count, and the head address of the suspend process list. In Concurrent Prolog, when a consumer process suspends the PPC requires the MBC to check whether a producer process has already sent a message to that MB containing the cell of the channel causing the suspension. If a message has arrived, the MBC sends the message to the PPC to activate the consumer process; otherwise, the MBC writes the PP address of the consumer process in the suspend process list.

A channel can be described with two "channel information" words stored in the structure area in a PTB. Given a goal sequence "p(X), c(X?)", p(X), which is placed in the PP of an IM as a result of an AND fork, has a PTB in Fig. 4. ChIR is a data type which stores a pointer to channel information. The first word of Channel Information is usually a pointer to an appropriate channel on the MB. The second word stores local data. When unification on a guard succeeds and the commit operation is performed, this local data is written into the appropriate channel cell on the MB.

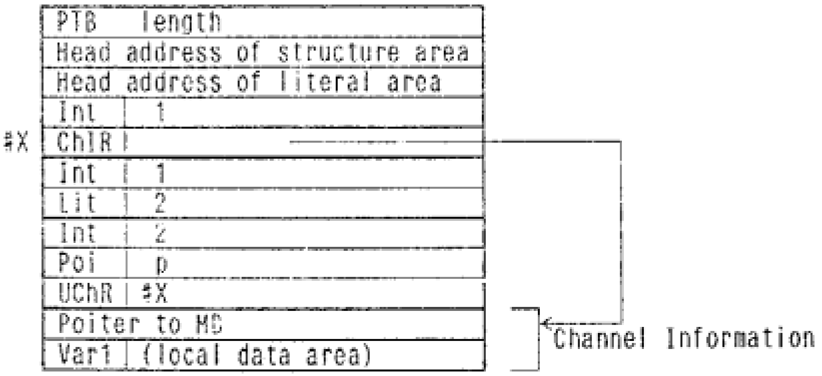

Fig. 4 Process Template Block of p(X).

2.2 Structure Memory Module

We have introduced the structure memory concept to reduce the copying overhead. The Structure Memory Module stores only a large structured data, such as large combined lists and vectors. Each Structure Memory Module (SMM) is connected to several Inference Modules (IM) via the IM-SMM Network and a part of the structured data is accessed by the Unification Unit (UU) in the IM on a unification-demand basis. Our SMM has the following characteristics.

(1) Ground instance sharing method

We have chosen the most basic method: that of sharing only ground instances which do not include any unbound variables. This sharing method can maintain highly parallel environments among processes.

(2) Combined representation of structured data

We use a combined representation scheme, which includes the list-based representation with pointers and the record-based representation storing data in successive memory cells.

(3) Lazy unification between arguments

For unification requiring the structured data stored in the SMM, PIM-R uses a lazy unification between arguments to avoid unnecessary unifications and accesses to the SMM. If there is an argument referring to the SMM in either the reducible goal or the unifiable clause, then this unification will be delayed and the other unifications between arguments not referring to the SMM will be executed with higher priorities.

2.3 Network

At execution of Prolog or Concurrent Prolog programs on PIM-R, when a goal succeeds in unification with a rule, a child process (new resolvent) is generated. However, even if a goal succeeds in unification with a fact, the result is only returned to the PCB in a parent process and a child process is not generated. (Onai 84) shows that the average OR-relation number is about 2 or 3 in Prolog programs that consist mainly of rules. Therefore, it is possible to map an average process tree onto a cyclic mesh structure.

In programs of logic languages such as Prolog, when there are several solutions, locations (depth) of each solution in the AND-OR tree are usually different and children forked simultaneously rarely return solutions to a parent process at the same time. Thus, the mesh-type network node, onto which the upper part of a process tree is mapped, only rarely gets overloaded by concentration of solutions returned to a parent process. Also all accesses except for those to the MB at Concurrent Prolog execution are between neighbor nodes. For these reasons, we chose mesh type structure for the inter-IM network (Fig. 1). In the inter-IM network, nodes are located at mesh points and an IM is connected to each node. The network between IMs and the SMM, the IM-SMM network, is of the equal distance type and at present is expected to be implemented with a shared bus.

3 BASIC SOFTWARE SIMULATION

The basic simulator is developed to confirm fundamental validity of PIM-R mechanisms and written in Prolog/C-Prolog, and runs on DEC2060 or VAX-11.

3.1 Simulation conditions

Simulation is performed under the following conditions:

(1) A network is ideal, excluding packet conflict.

(2) Each unit has a buffer of sufficient size.

(3) The performance of PPUs (process creation, renewal, and deletion) and UUs is determined by the number of steps required when they are written in an ordinary assembly language. (Moto-oka 84), who adopted the all goals copy method, shows that the average time of a unification and size reduction in a 6-Queens program is about 42μsec . Since PIM-R uses the only reducible goal copy method and the amount of copying for this method is less than that for the all goals copy method, we assume that UU in PIM-R is able to execute a unification and produce the packet to send to PPU in 100μsec at least. (We think that UU is able to execute those operations in under 50μsec if special hardware for the UU is implemented.) Since the average packet length through networks of 4-Queens is about 20 words (about 600 bits), we assume that network speed is 10 Mbps and average network delay is 60μsec .

(4) When unification with a unit clause succeeds in OR-parallel Prolog processing, the unification result is returned to the parent process in PP and no child process is created. On the other hand, when unification with a clause other than a unit clause succeeds, the new resolvent (new child process) is distributed. In Concurrent Prolog, goals in AND relation are distributed as different processes. This simulation adopts the following static and cyclic distribution strategy: IM concerned, East IM, South IM, IM concerned,... and so on. In simulations with two IMs, new processes are distributed to the IMs according to the following strategy: the first process is distributed to the IM concerned, the second to the neighboring IM, the third to the IM concerned,... and so on.

3.2 Simulation results

3.2.1 Prolog program

The 4-Queens program was run to collect the necessary data.

(1) Effect of number of Inference Modules (Fig. 5)

The 4-Queens program increases in performance as the number of Inference Modules (IM) increases. Processing time decreases as more Inference Modules are used up to seven or eight units. Then it levels off. This trends roughly corresponds to the average level of OR parallelism, about 6, resulting from a dynamic analysis (Onai 84). The results show that PIM-R is able to exploit the parallelism in the Prolog program.

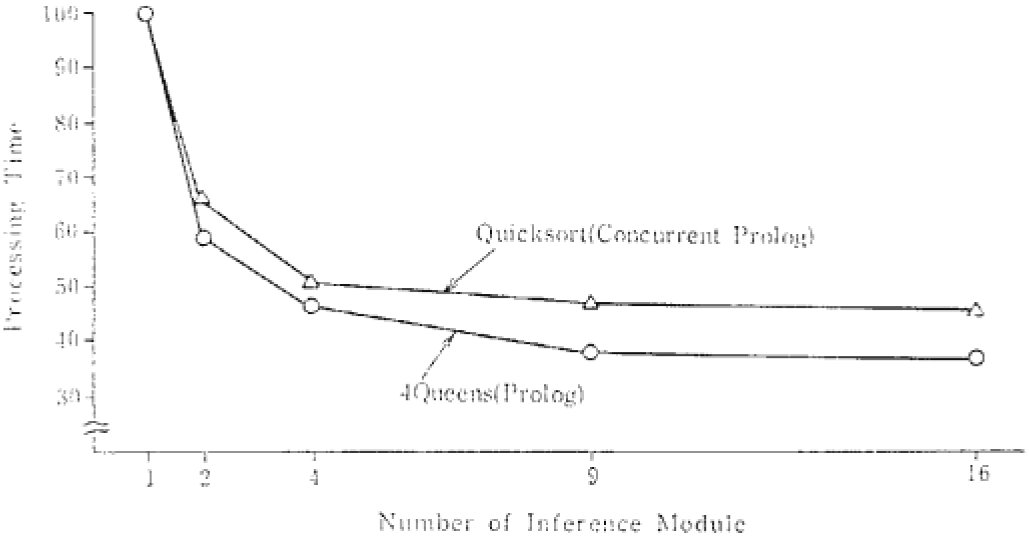

Fig. 5 Effect of Number of Inference Modules

(2) Halting dead child processing

The number of packets passing through the network is shown by their types in Table 1.

Table 1 Packet Number (through the network)

	2IMs	4IMs
Total number of packets	151	207
Number of true return packets	31	47
Number of OR-fork packets	60	80
Number of fork-down packets	60	80

The fork-down packet is used by a child process to inform its parent process that it has entered the dead state. It is concerned with garbage collection and is not directly related to the solution-obtaining processing. Therefore, if the PP has sufficient space, lower priorities can be given to dead child processing and the generation and transfer of fork-down packets in the PPC. Higher priorities are given to first executing processing required to obtain solutions and to transfer the respective packets. This can reduce the number of packets passing through the network by about 40% for two and four IMs. Also it alleviates processing load in the PPC, resulting in about 8% and 15% reductions in processing time to obtain the first and second solutions for four IMs respectively.

(3) Local execution control using reduction level

Local pseudo depth-first execution was tried giving higher priorities to Ready PCBs with deeper reduction levels and to processing of true-return packets from child processes. At the same time, lower priority is given to fork-down packet processing. The result was that 12% and 17% reductions in processing time to obtain the first and second solutions for four IMs respectively were achieved.

(4) Network load average and network speed

The load average of each link is about 10% for two IMs and 6% for four IMs. (There are 2 links for two IMs and 8 links for four IMs.) The more IMs the less load average. Simulation condition (1) is appropriate. The maximum input buffer length of the Packet Switch is 15 for two IMs and 14 for four IMs. If we speed up the network from 10Mbps to 60Mbps, we get about 10% reduction in processing time for four IMs.

(5) PPU load average and dynamic process distribution

When 60Mbps network speed for four IMs is achieved, the PPC in each IM recorded load averages of 42%, 55%, 54%, and 80%. Relatively wide discrepancies exist among these figures, because the child process distributing strategy at OR fork was fixed. In this case, the average input buffer lengths of the packet switches are 0.46, 0.69, 0.49, and 4.5 units. These results correlate with the load averages of the PPC. When network nodes are used to dynamically distribute child processes to the IM whose packet switch has the shortest input buffer length, the processing time necessary for obtaining the first and second solutions can be decreased further by 10% and 14% for four IMs; then the PPC in each IM has balanced load averages of 69%, 72%, 62%, and 65%.

3.2.2 Concurrent Prolog program

The Quicksort (ten elements) program was run to collect various data items.

(1) Effect of number of Inference Modules (Fig. 5)

The Quicksort program increases in performance as the number of IMs increases. Processing time decreases as more Inference Modules are used up to five or six units. Then it levels off. Since this example has a parallelism of about 4, the results show that PIM-R is able to exploit the parallelism in the Concurrent Prolog program.

(2) Halting dead child processing

Table 2 shows the number of reductions and other data at the execution of Quicksort.

Table 3 shows the number of packets passing through the network during Quicksort execution by their types.

Table 2 Reduction Number

Number of reduction	284
Number of successes	196
Number of failuers	19
Number of suspended reduction	69

Table 3 Packet number

	2IM	4IM
Total number of packets	141	211
Number of true return pakets	17	19
Number of AND-fork packets	21	26
Number of fork down packets	21	26
Number of MB-related packets	82	140

These tables show that, while 284 reductions occurred and 196 of them succeeded, 141 packets passed through the network for two IMs and 211 packets for four IMs. In other words, a packet passed through the network each time about 1.3 or 2 reductions occurred. Unlike the Prolog program, the MB-related packets account for 58% of the total for two IMs and 66% for four IMs, as shown in Table 3. This means that halting dead child processing and generation and transfer of fork-down packets, an effective approach in the Prolog program, could reduce the number of packets passing through the network by only 15% for two IMs and 12% for four IMs. Unlike the fork down packets, these MB-related packets cannot have a lower transfer priority attached; if they do, no solution will be obtained. Therefore, faster packet transfer is more critical for Concurrent Prolog.

(3) Effect of Message Board Controller (MBC)

A channel cell is allocated on the MB when unification succeeds in the UU, a new child process is returned to the PPU, and there is a new channel. At execution of Quicksort, 88 channels are stored on the MB and, as Table 2 shows, successful unifications are total 196. This requires the cell for a channel to be allocated on the MB every time about 2.2 unifications succeed. Thus the speed of allocating a channel cell on the MB influences PIM-R processing speed. This problem can be eliminated by introducing a MBC to handle MB-related processing. If the MB-related processing were handled by the PPC, instead of the MBC, processing time would increase by 13% for one IM and by 10% for four IMs.

(4) AND-OR parallel execution

When Concurrent Prolog programs are executed in AND-OR parallel, child processes distributed in the different IMs from the parent process have to check the commit tag in the parent process through networks when guard execution is successful. Child processes have to wait for the return packet from the parent process. This increases network traffic. In the case of four IMs at execution of Quicksort, since the number of commit tag check packets and return packets is 22, the total packet number increases about 10%. As a result, the processing time for AND-OR-parallel execution increases about 13% over that of AND-parallel execution. OR-parallel execution is not suitable for Concurrent Prolog programs.

4 DETAILED SOFTWARE SIMULATION

Detailed software simulators have been developed using Occam (INMOS 84), a language capable of describing multiple processes running concurrently and message communications between processes. The major emphasis is on simulation which precisely reflects the detailed structure of PIM-R, such as internal data formats, and which involves 16 to 64 or more IMs. The simulators are currently running on the VAX 11 to collect various pieces of data.

4.1 Simulation Conditions

(1) Dead process processing is not halted in PPC.

(2) If the PTB does not get longer at compaction (ONAI 85) when a solution to a built-in predicate is returned, direct overwriting to the process template is possible.

(3) The PPC attaches new Ready PCBs to the tail of the RPQ and transfers the head Ready PCB to the UU.

(4) Simulation clock is not introduced yet.

(5) Process distribution is the same as that of basic software simulation.

4.2 Simulation Results

4.2.1 Message Board (MB) - related packets

Packets passed through the inter-IM network include AND-fork packets and packets indicating successful/ failed unification of child processes. They also include MB-related packets such as the channel-value write/ read packets that are passed when Concurrent Prolog channel variables are not stored on the MB of the IM concerned and the activation packets are returned to suspended processes. Table 4 shows how the number of packets and the total length of the packets (in words) changes with the number of IMs for Quicksort (50 elements) simulation. Table 4 suggests that the MB-related packets, each consisting of a small number of data words, account for most of the packets transferred between IMs. Therefore, fust transfer of MB-related packets is important.

Table 4 Packet number and word length

IM number	1	2	4	9	16
packets in	8106[PKT]	6773	6127	5694	5683
IM	166258[W]	161267	158849	157162	157123
inter IM	0	1535	2247	2674	2681
packets	0	11110	15709	17261	17106
MB-related	0	1333	1979	2412	2423
packets in	-	(87%)	(88%)	(90%)	(90%)
inter IM	0	4991	7409	9096	9135
packets	-	(45%)	(47%)	(53%)	(53%)

4.2.2 Effect of shared Clause Pool (CP)

The software simulator involving two IMs ran 6Queens written in Prolog and Quicksort (50 elements) in Concurrent Prolog to collect data on the PTB and packets. The results are shown in Table 5 and Table 6 bellow ("literal length" = literal header length + literal area length).

Table 5 Effect of shered Clause Pool (in Prolog)

6Queens	Average total length in word(A)	Average literal length in word(L)	L/A(%)
PTB	44.8	21.7	48.4
OR-fork packets	43.7	23.4	53.5
true return packets	30.9	9.9	32.0

Table 6 Effect of shered Clause Pool (in Concurrent Prolog)

Quicksort	Average total length in word(A)	Average literal length in word(L)	L/A(%)
PTB	37.7	11.0	29.3
AND-fork packets	36.2	4.5	12.4
true return packets	24.3	5.5	22.5

If the PPC can access the CP, the length of the PTB can be shortened by about 30 to 50% by removing the literal header and literal area from the PTB. This results in a decrease in the amount of processes generated and renewed copying in the PPU as well as the amount of unification-related copying in the UU. In addition, OR-fork and true-return packets in 6Queens can be reduced in length by 54%and 32%, respectively, and AND-fork and true-return packets in Quicksort by 12%and 23%, respectively. This in turn leads to reduction in network traffic.

4.2.3 Effect of Structure Memory Module (SMM)

Various data items are being collected by running evaluation programs, such as the morphological analysis program, DCG program, a formula simplification program (Equiv2), and Quicksort (50 elements), on the prolog software simulator. The data obtained is used to examine the effect of the SMM on the structured area and the effect of lazy unification in particular.

(1) Reducing effect of structure area

As shown in Table 7, the use of the SMM for program 1 to 3 reduced the PTB length by 20 to 30% when the CP was not shared (A/B) and by 30 to 45% when it was shared (C/D). The SMM was also able to shorten (in words) the OR-fork and true-return packets by 10 to 45 percent for the non-shared CP (A/B) and by 20 to 50% for the shared CP (C/D). The combined effect of sharing the CP and the use of SMM (C/B) resulted in about a 50 to 60% decrease in length for the PTB and about a 50 to 60% decrease for OR-fork and true-return packets. Since the structure area generally accounts for a larger portion in true-return packets than in OR-fork packets, the introduction of the SMM was slightly more effective for true-return packets. Also, the simulation leads to the conclusion that the SMM will have a significant effect on programs, such as the morphological analysis, DCG, and formula simplification programs, which perform various processing operations on structured data bound with ground instances without changing the data, but an insignificant effect on programs like Quicksort which successively create new lists.

Table 7 Reducing effect of structure area

			Average length in words (Clause Pool not shared)			Average length in words (Clause Pool shared)		
	test programs		SMM used:A	SMM not used:B	A/B (%)	SMM used:C	SMM not used:D	C/D (%)
1.	Morphological analysis	PTB	44.2	64.2	68.8	23.6	43.7	54.1
		OR-fork	35.8	47.4	75.5	17.9	29.7	60.2
		true return	32.3	52.6	61.4	24.0	44.3	54.2
2.	DCG	PTB	29.3	37.9	77.3	14.5	23.1	62.9
		OR-fork	28.3	34.6	81.3	13.2	19.7	66.8
		true return	51.3	86.0	59.7	42.8	77.4	55.3
3.	Formula simplificaton	PTB	35.3	43.4	81.3	20.4	28.5	71.5
		OR-fork	35.9	40.8	88.0	20.5	25.4	80.9
		true return	54.5	98.5	55.3	46.9	90.9	51.6
4.	Quicksort	PTB	85.7	94.2	91.0	61.7	74.2	83.1
		OR-fork	63.3	73.0	86.7	42.7	52.4	81.6
		true return	132.7	144.5	91.8	122.7	134.5	91.2

(2) Effect of lazy unification

As Table 8 suggests, the use of lazy unification could lower the number of SMM read requests and the total length of SMM read returns by about 10% for the morphological analysis program. Lazy unification, however, cannot work effectively for some programs like the DCG program.

Table 8 Effect of lazy unification

Morphological analysis	lazy(L)	nomal(N)	(N-L)/N (%)
Number of SMM read requests	822	908	9.5
Total length of SMM read returns	2772	2956	6.2

4.2.4 Effect of writing programs in GHC and GHC-supporting data types

In programs written in GHC (Ueda 85), the base language of KL1(85), PTBs need not to have a local environment for a channel. The following table shows goals-related data (not built-in predicates) which were passed from the PPU to UU when 50-element Quicksort in Concurrent Prolog ran on the two-IMs. Table 9 suggests that the packet length can be reduced by 25% by writing programs in GHC (a pointer to the MB can be stored in the ChIR-type word in the variable area). Note that, when a predicate is called in GHC, binding which can be observed from the caller cannot be generated while the guard in a clause is being executed. Therefore, such a binding (unification) must be moved to the body of the clause.

Table 9 Effect of writing programs in GHC

Average goal length in words	35.4
Average structure area length in words	18.0
Average channel information (a pointer to the MB and local enviroment) length in words	8.8

For example, a clause "qsort([],[]):- true | true." in Quicksort in Concurrent Prolog is expressed in GHC as follows:

```
    qsort([],X):- true | X=[].
```

This causes the variable area to increase by one word for the new variable X and the literal area by four words because the internal format of "X=[]" is four words in length. These increases in the variable and literal areas in GHC over Concurrent Prolog require the CP to be accessed from the PC as well. This is also necessary to benefit from the GHC's ability to eliminate local environments (if the CP can be accessed from the PPC, the literal area can be removed from the PTB). Also, the introduction of three new GHC-supporting data types - TopCh (channel in the goal in the first calling process), WritableCh (channel capable of undergoing binding), and NestedCh (channel called from a guard directly or indirectly) - seems to make it possible to write programs in GHC without changing the Concurrent Prolog execution mechanism in PIM-R.

5 CONCLUSION

This paper described the architecture and evaluations using software simulatiors of PIM-R, a reduction-based parallel inference machine. The introduction of the PLB into a process can make it unnecessary to report each generation or deletion of a goal sequence to its parent process, irrespective of how many goal sequences are generated or deleted in that process. Therefore the use of such process-structuring methods permits a decrease in the number of packets passing through the inter-IM network.

As for architecture, PIM-R uses a Message Board to handle Prolog and Concurrent Prolog programs equally. It was confirmed that the MB permits PIM-R to execute Concurrent Prolog functions including back communication and finite-length buffer communications. The evaluation using PIM-R software simulators demonstrated that PIM-R is able to exploit parallelism in Prolog and Concurrent Prolog programs. In other words, the number of Inference Modules does affect the performance of parallel processing. It was also confirmed that the dynamic distribution of child processes by network nodes, introduction of the MBC, halting of dead child processing, stopping the generation and transfer of fork-down packets in the PPC, local pseudo-depth-first execution using reduction level, introduction of the SMM, and shared Clause Pool are all effective measures.

At Concurrent Prolog execution, over half the packets sent through networks are related to MB access. Therefore, we think that it is not sufficient to increase packet transfer speed. It is also necessary to decrease the relative number of MB-related packets to other packets. It is a way of introducing modularity into the language to enlarge grain size of AND-parallel processing of Concurrent Prolog. Since a channel in Concurrent Prolog is a logical variable, a producer and a consumer have to execute unification for sending and receiving a message respectively. It causes a reduction in speed of message transfer. Clearly, research into the implementation problem of communication and synchronization in logic-type languages not using logical variables is necessary.

At present we are developing a hardware simulation system consisting of sixteen MC68000s (Sugie 85). We plan to conduct various detailed simulations of many programs, with these tools to validate and enhance PIM-R.

ACKNOWLEDGMENTS

Finally, thanks are due to Director Kazuhiro Fuchi and Dr. Shunichi Uchida at ICOT for providing the opportunity to pursue this research. The discussion with and comments of Dr. Mamoru Sugie, Mitsugu Yoneyama and Masa-aki Iwasaki of the Central Research Laboratory Hitachi, Ltd. were of great benefit.

REFERENCES

Clark KL, Gregory S (1984) PARLOG:Parallel Programming in Logic, Research Report DOC 84/4, Dept. of Computing, Imperial College, London

INMOS Limited (1984) Occam Programming Manual, Prentice-Hall International Series in Computer Science

Ito N, Masuda K (1984) Parallel Inference Machine Based on the Data Flow Model, Proc. of the International Workshop on High Level Computer Architecture 84, Los Angeles

Moto-oka T, Tanaka H, et al (1984) The Architecture of a Parallel Inference Engine-PIE-, Proc. of Int. Conf. on Fifth Generation Computer Systems 1984, ICOT, Tokyo

Onai R, Shimizu H, Masuda K, Aso M (1984) Analysis of Sequential Prolog Programs, ICOT TR-048

Onai R, Aso M, Shimizu H, Masuda K, Matsumoto A (1985) Architecture of a Reduction-based Parallel Inference Machine:PIM-R, New Generation Computing, vol 3/2, Ohmsha, Springer-Verlag, p 197-228

Pereira LM, Nasr R (1984) DELTA-PROLOG: A Distributed Logic Language, Proc. of Int. Conf. on Fifth Generation Computer Systems 1984, ICOT, Tokyo

Shapiro EY (1983) A subset of Concurrent Prolog and Its Interpreter, ICOT TR-003

Sugie M, et.al. (1985) Hardware Simulator Implementation of PIM-R, Logic Programming Conference '85, Tokyo

Ueda K (1985) Guarded Horn Clauses, ICOT TR-103