Prolog Machine
Based on the Data Flow Mechanism

by

Noriyoshi Ito, Rikio Onai,

Kanae Masuda, Hajime Shimizu

Prolog Machine Based on the Data Flow Mechanism

N. Ito, R. Onai, K. Masuda, H. Simizu

Institute for New Generation Computer Technology

1. INTRODUCTION

Prolog (Programming in Logic) is a simple but powerfull language containing a basic inference capability. The execution process of Prolog is closely related to a data flow concept. A model based on the data flow concept can naturally realize the parallel processing and has the capability of functioning as the base of a highly parallel processing system[1]. The authers are investigating a parallel inference machine which executes Prolog based on the data flow concept. This paper describes the basic idea of the parallel processing mechanism and the conceptual architecture of the machine.

2. PARALLEL UNIFICATION

An unification, or a pattern-matching, is a basic function embedded in Prolog. Given a goal sentense, the unification function solves the goal sentence by matching, in terms of pattern, the goal sentence with a set of clauses. The parallelism concerning this unification can be divided into three types as follows:

(1)OR parallelism

(2)AND parallelism

(3)Parallelism among arguments

The first parallelism, the OR parallelism, can be accomplished in a straightforward manner. This parallelism is achieved, given a goal literal in the goal sentence, by unifying in parallel the clause whose head litaral has the same predicate as the goal literal (A goroup of such clauses is called a definition, and these clauses are combined by OR connectives each other.) with the goal literal with respect to all the clauses included in the definition [2].

In contrast, the second paralleism, the AND parallelism, involves consistency checking. In other words, when goal literals in a goal sentence have shared variables in there arguments, the complex communications are generally required among AND unfication processes running in parallel, in which each AND unification process tries to solve a goal literal, so that consistency of the binding status can be maintained for these shared variables.

The third parallelism, the parallelism among arguments, means, when a goal literal contains multiple arguments, unfications are performed in parallel on individual arguments. Although it can be considered a variant of the AND parallelism due to the necessity of consistency checking, this checking can be realized by a synchronization

operation which simply checks whether unfications for all arguments has successfully completed or not.

3. UNIFICATION BY THE DATA FLOW APPROACH

As described above, the introduction of AND parallelism may results in an extremely complicated operation. Therefore our machine will implement OR-parallel/AND-pipeline processing using a stream concept[3]. Also it acheives efficient parallel processing among arguments by decomposing an unification process into individual unification operators and representing it by a data flow graph.

A stream plays a role of a "pipe" for the flow of binding information (i.e. instances) for variables and provides a means of asynchronous communications between the binding information producer and consumer.

For example, assume that the following goal sentence and a set of clauses are given (The syntax below conforms to DEC-10 Prolog[4]):

```
?-p(X,Y),q(X),r(Y).

p(a,b).
p(c,d).
q(a).
q(e).
r(b).
r(d).
```

where X and Y denote variables and the literals in the goal sentence are processed from left to right. The binding

information for X and Y are passed along the streams represented by the arrows in the above figure, and processing between p(X,Y) and q(Y) and between p(X,Y) and r(Y) are performed on a pipeline basis. Since q(X) and r(Y) have no shared variable each other, they can be executed in parallel. If a clause such as p(Z,Z) is given in the definition of predicate p, however, a dynamic sequence control mechanism may be required between the processes q(X) and r(Y), because X and Y are to be bound to the same variable Z.

Our machine basically uses the breadth-first approach which simultaniously initiates executable unification processes. To prevent a deadlock due to an explosive resource exhaustion, the machine adopts priority control of processes by assigning a priority to each process, while it provides, if required, a swapping/termination function for designated processes. As for priority control, a search strategy based on a pseudo depth-first approach can be achieved by, for example, representing process identifiers in a search tree with numbers in the depth (vertical) direction and those in the width (horizontal) direction, and employing the width-direction numbers as priorities.

These capabilities can be accomplished by appending a process identifier field to a token which carries an operand data, and by providing a hardware function to control the process status by identifing this field.

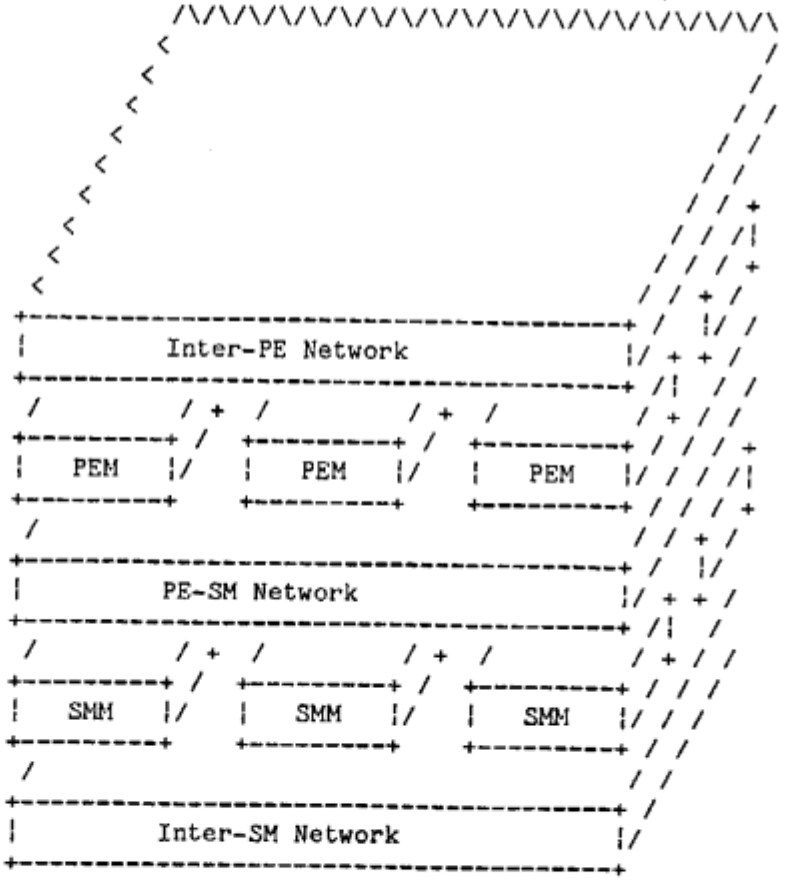
4. MACHINE ARCHITECTURE

Figure 1 shows a conceptual configuration diagram of our machine. The machine consists of Processing Element Modules (PEMs) which execute unification processes represented by the data flow graphs, Structure Memory Modules (SMMs) which store and manipulate structured data, and three types of network; Inter-SM, PE-SM, and Inter-SM Network.

(1) Processing Element Module (PEM)

The PEM executes unification processes represented by data flow graphs. A PEM consists of an Instruction Control Module (ICM) which controls the instrucution execution, and an Execution Module (EXM) which interpretes and executes instructions. The ICM is initiated by an arrival of any token on its input port. Each token has a process identifier field, a destination field which contains a desination instruction address, and a value field which contains operand data of the instruction. The ICM tests the executability of the instruction, i.e. tests if all the operands of destination instruction has arrived or not. If the instruction is executable, then the ICM constructs an executable instruction packet which contains an operation code, operands, and next destination fields which specify the destination instruction addresses of the results, and sends it to the EXM. Otherwise, the ICM stores the operand in its operand memory, until the instruction becomes

executable. The EXM receives the executable instruction packet, executes the instruction, generates new tokens with the result and the destination fields in the executable instruction, and sends back them to the ICM specified by the destinations. The ICM and EXM form a circular-pipeline structure. The EXM is further constructed by multiple Processing Units (PU) to balance the EXM with the ICM in the average processing speed. The PU executes built-in predicates, control instructions, etc., which does access no structured data, and generate the results as tokens.



PEM: Processing Element Module

SMM: Structure Memory Module

Figure 1 An abstract machine configuration

If a PU has received a structured data manipulation instruction, it sends the instruction to specified SMM (described below) and leave the instruction execution to it.

Unification processes are dynamically allocated to PEMs so that load balance will be maintained among PEMs. (A PEM in a relatively idle state will load a new process from the process queue and execute it.) In addition, a PEM is provided with a process priority control function as well as a swapping and deletion fuction for the tokens belonging to particular processes.

(2) Structure Memory Module (SMM)

An SMM has functions to store, control, and manipulate strctured data such as lists, vectors, and streams, and is shared among all PEMs. In order to avoid temporary load concentration or access contention on a particular SMM, a logical structured data may be distibuted and stored in multiple SMMs. In addition, for effective garvage collection control, a reference count method is used.

(3) Network Structure

There are three types of network, each based on an asynchronous communication method.

a. Inter-PE Network

This network is used for communications between processes assigned to different PEMs. Such unification processes are dynamically allocated to PEMs. When they are allocated, a strategy capable of making use of the locality of inter-PE communications is adopted. That is, a newly created process may be allocated to a PEM which is in a short distance from its parent's PEM. The topology of Inter-PE network, therefore, is a two-dimentional mesh configuration.

b. PE-SM Network

This network is used when a process in a PEM accesses structured data. Generally it seems difficult to maintain a load balance among SMMs while storing structured data in SMMs to localize communications between SMMs and PEMs. The PE-SM Network, therefore, is an equi-distance network consisting of four-input-four-output switching nodes in a multiple-stage configuration.

c. Inter-SM Network

This network is used when inter-SM communications are required to traverse a data structure as in the case of testing if the structure is a ground term or not (i.e. inclusion check of variables). Generally it is possible to introduce a strategy capable of making use of the locality of inter-SM communications when a data structure is mapped over multiple SMMs. Therefore, like the Inter-PE Network, this network also employs a two-dimensional mesh

configuration.

4. CONCLUSION

This paper has described the abstract configuration of a Prolog machine based on the data flow model. We will carry out more detailed investigations on its architecture using such methods as simulation. We would like to thank Mr. K. Murakami, Chief of the First Research Laboratory, for his valuable daily guidance.

<referneces>

[1] Arvind, Gostelow,K.P. and Plouffe,W.E.," An Asynchronous Programming Language and Computing Machine", TR 114a, Department of Information and Science, University of California, Irvine, Dec. 1978.

[2] Clark,K.L. and Gregory,S., "A Relational Language for Parallel Programming", Res. Rep. of Imperial College of Science and Technology, DOC 81/16, July, 1981.

[3] Conery,J.S. and Kibler,D.,"Parallel Interpretation of Logic Programming",Proc. of Conf. on Functional Programming Language and Computer Architecture, ACM, Oct. 1981.

[4] Pereira,L., Pereira,F. and Warren,D.,"User's Guide to DECsystem-10 Prolog", Department of Artificial Intelligence, University of Edinburgh, Sep. 1978.