A NOTE ON THE ELEMENTARY EXECUTION UNIT IN A PARALLEL INFERENCE MACHINE

T.Maruyama, K.Hirata, H.Tanaka and T.Moto-oka

Department of Electrical Engineering, The University of Tokyo

Bunkyoku, Tokyo 113, JAPAN

Abstract

When we design a parallel inference machine which executes logic programs, there are some important problems that greatly influence the machine performance. The most important problem is how to decide the elementary execution unit (granule). The less the size of the elementary execution unit is, the less is the overhead of copying and transferring, but the control among the units becomes more complicated.

Until now, a few models of parallel inference machines have been proposed. However, we think that this problem is not discussed enough in these models. In this paper, we discuss the problem in detail.

1. Introduction

When we design a parallel inference machine, there are some important problems that greatly influence the machine performance. The most important problem is how to decide the elementary execution units that will be executed in parallel (we call this unit simply a goal). The less the size of the goal is, the less is the overhead of copying and transferring the goals, but the control among the goals becomes more complicated and may decrease the machine performance.

Until now, a few models of parallel inference machines have been proposed (Haridi 83; Ito 84; Onai 85). However, we think that the problem is not discussed enough in these models. We discuss the problem in detail, according to the research work on a highly parallel inference engine - PIE - (Moto-oka 84; Yuhara 84).

2. Data Sharing in a Parallel Inference Machine

In general, there are two kinds of data sharing in a parallel inference machine. One is the sharing of the structure data which compose the goals. This sharing aims to decrease the goal size and the overhead of copying and transferring the goals by storing structures in the shared memory, and sharing them among the descendant goals. This sharing further aims to decrease the amount of memory to

store goals. However, if we try to share too many structures, the overhead of separating them from the goals and the access conflict on the shared memory may decrease the performance against our will. Structure memory will be needed to implement this sharing method efficiently.

The other is the sharing of the goal itself. In a parallel inference machine, a goal may be shared among its descendant goals produced as the result of OR-parallelism, according to the level of the goal (elementary execution unit). For example, if a goal is composed of a literal, a goal is unified with several definitions, and a few descendant goals are produced, then the parent goal (which was unified) is shared among the descendant goals. In this case, the unifications of descendant goals may change the values of the parent goal variables. The parent goal has to be copied, or only the variables whose values change have to be copied at this time. If the level (granularity) of the goal is too low, this goal sharing may cause access conflict and load concentration. This sharing is tightly related to the granularity of the goals. If the goal has all environments that will be needed for its further unifications, the sharing of goals like this will not happen.

3. Elementary Execution Unit

There are two important points when we decide the elementary execution units (goals).

(1) the granularity of the goal
(2) copying/sharing the skeletons of the literals and structures

The first point is how to decide the granularity of the goal. The lower the granularity (the less the size) is, the less is the overhead of copying and transferring the goals. However, the access conflict and load concentration caused by the sharing may decrease machine performance. The second point is how to represent the goal in a parallel inference machine. Sharing of literal/structure skeletons decreases goal size, but the access conflict to the memory that stores the environments may decrease the performance.

The parallelisms that will be mainly realized in a parallel inference machine are

(1) inter-argument parallelism,
(2) AND parallelism,
(3) OR parallelism,

and

(4) inter-goal (elementary execution units) parallelism.

We think that stream parallelism is one form of inter-goal parallelism. We mainly discuss how to decide the granularity and internal representations of goals which are suitable for parallelism 2, 3 and 4.

3.1. Level of the Goals

In a sequential inference machine, OR parallelism is replaced by backtracking. If we try to make a parallel inference machine by slightly modifying a sequential inference machine, we can think of a model that regards parts of a stack as a goal, and transfer the goals among the sequential machines. In this case, parts of a stack in a machine may be shared by the goals distributed to the other machines. For example, if three goals are generated by unification of their parent goal, these goals share the parent goal environment (parts of a stack). These descendant goals may change the values of the parent goal environment one after another. The parent goal environment must be copied on every change of the values. The access conflict to the memory that stores the parent environment becomes a problem. In a parallel inference machine, the granularity of a goal decides the independency among the goals. The lower the goal granularity is, the smaller is its size, and the less independent are the goals. It is difficult to decide the granularity of the goal.

In general, we can consider four kinds of goal granularity as follows.

(1) argument
(2) literal
(3) clause body
(4) the environment from the initial goal

According to the increase of the item number, the granularity becomes higher. No sharing of goals occurs in 4. In 1 ~ 3, we can regard several arguments, several literals and several clause bodies as a goal. For example, if we regard several literals as a goal, and the number of literals in the goal is the same as the number of literals in the clause body, this goal is equal to the one in 3. This kind of enlargement of goals is taken for the increase of the machine performance. We think that our classification above is adequate for the discussion below.

We show the execution phases of each goal level (granularity) in Fig. 1 ~ 4. Though each figure is simply an example of the execution of each goal level, we think each figure represents the features well. Consistency check becomes necessary in Fig.1 because the arguments which are AND-related are unified in parallel. To avoid the overhead of it, we can dynamically cluster the arguments, but this is also rather heavy. This inter-argument parallelism can be combined with the executions of the other parallelisms.

In fig.2, each goal puts back the values of the bound variables into the parent goals (Fig.2a) and the next literal of the clause is separated from the clause as a new goal (Fig.2b).

The example shown in Fig.3 is almost same as the one in Fig.2. But there is no need to separate new goals explicitly as shown in Fig. 2 because the level of the goal corresponds to the clause body. The timing to put back the

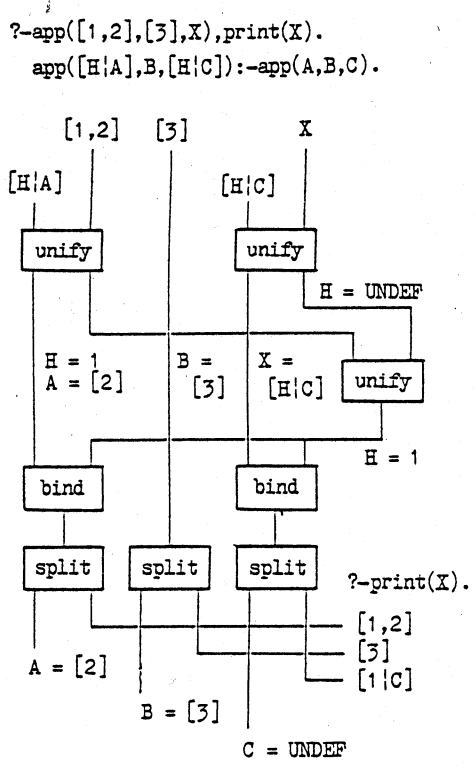


Fig. 1 An Example of a Data Flow Graph

```
?-f(X,Y),g(Y,Z).

f(X,a) :- p(X), q(X).
f(X,b) :- r(X), s(X).

p(a).  q(a).  r(a).  s(a).
p(b).  q(b).  r(b).  s(b).
```

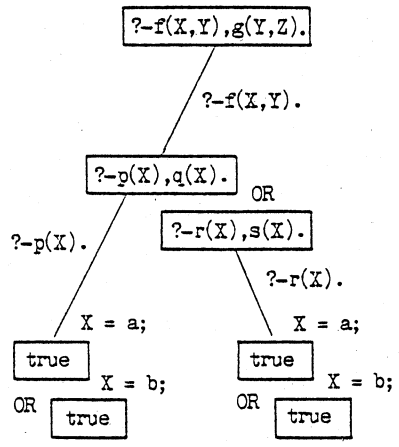


Fig. 2a An Example of the Execution of Literal Level

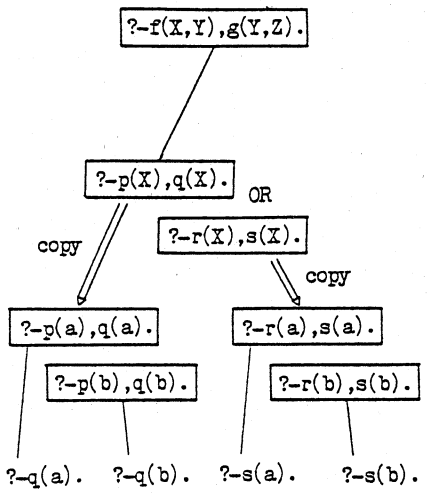


Fig. 2b An Example of the Execution of Literal Level (the generation of new goals)

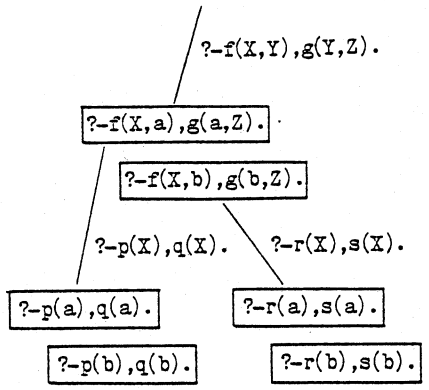


Fig. 3a An Example of the Execution of Clause Body Level

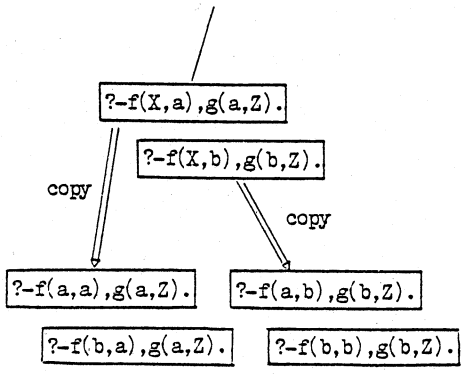


Fig. 3b An Example of the Execution of Clause Body Level (the generation of new goals)

values of the bound variables also differs from the one in Fig.2. The size of the goal in Fig.2 is usually smaller than the size in Fig.3, but the overhead of separating the next literal as a new goal becomes necessary.

In Fig.4, because all environments needed for further unifications are included in each goal, there is no need to put back the values of the bound variables into the parent goal. The control among the goals greatly differs from the one in Fig. 2 and 3.

3.2. Skeleton Sharing / Copying

In sequential inference machines or in systems written on sequential machines, skeletons of literals are shared in most cases. The overhead of literal skeleton copying is too heavy in a sequential machine, but if we consider the overhead of copying in a parallel inference machine, it may become negligible through pipeline execution of copying and unification. If it becomes negligible, we can find a few advantages in literal skeleton copying. Dereferences during the unification becomes fewer, and the initialization of the variable area becomes unnecessary, and so on. We can think of four kinds of the internal representations of the goal for each goal level (granularity).

(1) literal skeleton sharing / structure skeleton sharing
(2) literal skeleton sharing / structure skeleton copying
(3) literal skeleton copying / structure skeleton sharing
(4) literal skeleton copying / structure skeleton copying

In these combinations, 3 is obviously inefficient and we will not discuss it. We regard the execution of the argument level as included in 2 (or 1), because it is natural to

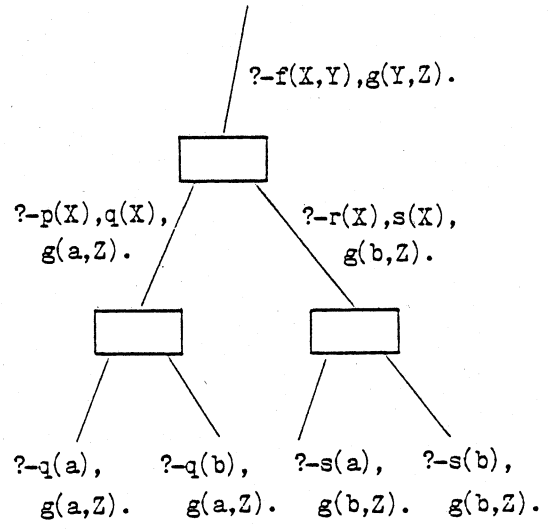
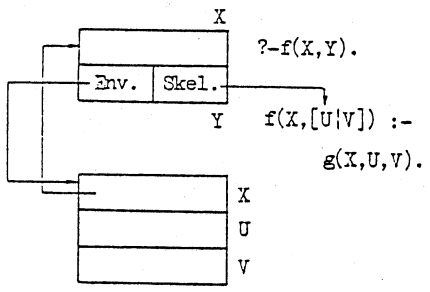


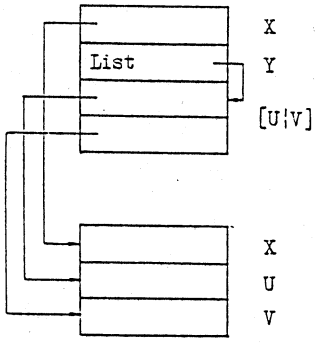
Fig. 4 AN Example of the Execution when the Goals has all Environments

think that each literal is preprocessed into data flow graphs and that literal skeletons will not be copied. The examples of the internal representations of the clause body level goals are shown in Fig.5.

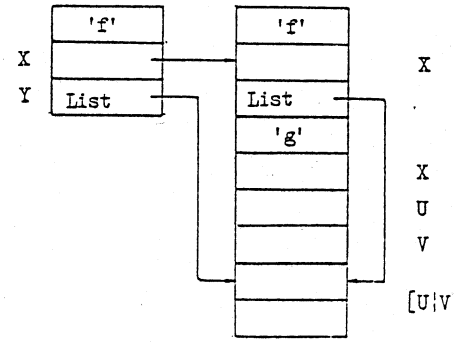
In combination 1, a molecule will be constructed when a variable is bound to a structure, and changing of the values of the parent goal variables by the descendant goals will happen if the descendant goals don't have all environments needed for their unifications. We can think of two methods for changing the values of the parent goal variables: The first method is that the variable area (only the variables whose values will change) is copied during the unification if the environment pointer (Env) of the molecule points to the parent goal variable area, while during the unification the values of the parent goal variables are referred to; The other method is that the part of the parent variable area that will be needed for further unifications is copied, to avoid the overhead of the reference to the parent goal.



(a) Literal Share / Structure Share



(b) Literal Share / Structure Copy



(c) Literal Copy / Structure Copy

Fig. 5 An Example of Internal Representation of the Goal

Anyway, both methods are inefficient. In the first method, the access conflict to the memory that manages the parent goals seems to become too heavy. In the second method, the number of the variable is used to access variables through the molecule, and the difficulty of renumbering of the variables tends to copy all the structures that were once referred to and will not be referred to any more.

In the combination 2 and 4, we can think of the same methods as in the 3. But in these cases, only the structures needed for further unifications can be copied easily, because the variables are accessed not using the skeleton of the structures (the value of each variable in the copied structure is changed to 'ref' etc). The size of the goals in 4 becomes larger than the size in 2.

4. Internal Representation of the Goals

In chapter 3, we discuss the granularity of a goal and sharing/copying of skeletons. In this chapter, we discuss the combination of them. We can think of the internal representations of the goals as shown in Table 1. When we try to evaluate each representation, there are some important points, as follows:

(1) size of the goal
The less goal size is better, as far as the dependency among the goals caused by the less size (lower granularity) does not decrease the performance.

(2) independency among the goals
The dependency among the goals strongly depends on the granularity of the goal. Higher independency makes the load distribution easier.

(3) faster unification
The elementary operation of the inference machine is unification. The representation that makes unification faster and also makes deterministic unification faster is better.

In the following sections, we discuss the internal representations shown in Table 1 from the aspects above.

4.1. Inter-argument Parallelism

We can take two approaches for inter-argument parallelism. One is the approach that aims to exhibit maximum parallelism. This approach seems to be suitable for data flow machines. The overhead of consistency check (or dynamic clustering of the arguments) and the fact that the input/output of a variable will be decided dynamically become problems. The other approach aims only to execute the arguments in a literal in parallel. This approach can be combined with the other level parallelism, but the degree of parallelism in a literal is small and the effect of this

approach is not evident. It is necessary to study dedicated hardware to realize the inter-argument parallelism.

4.2. Literal Sharing / Structure Sharing

In 2 and 3 in Table 1, the environment pointer (Env) of the molecule may point to the environments of the other goals. As mentioned above, it is inefficient to refer to the parent goal during the unification and it is also inefficient to copy the structures needed for further unifications because it is difficult to copy only the structures that will be needed. Therefore, the methods 1 and 2 seem to be less efficient than methods 6 and 7. If we consider the size of stacks of sequential inference machines, the method that copies all the data like 4 is not acceptable.

Anyway, the sharing of structure skeletons in parallel inference machines is inefficient unless we aim to execute only a few degree of the parallelism.

4.3. Skeleton Copying

In 6 ,7 ,9 and 10 in Table 1, we can consider two methods like 2 and 3. The method that changes the values of the parent goal variables does not suit parallel processing, but the method that copies the structures needed for further unifications is efficient in this case (Fig. 6).

First, we compare 7 and 10. The goal size of 10 is generally larger than the size of 7. But in 10, the unification may be faster if the machine executes the unification in a interpretive manner (if definitions are compiled, there is no difference). The merit of 10 compared with 7 is

Table 1 The Elementary Execution Units

		argument	literal	clause-body	Env.
structure share	literal share	1	2	3	4
	literal copy		x	x	x
structure copy	literal share	5	6	7	8
	literal copy		9	10	11

Env. is the environments from the initial goal.

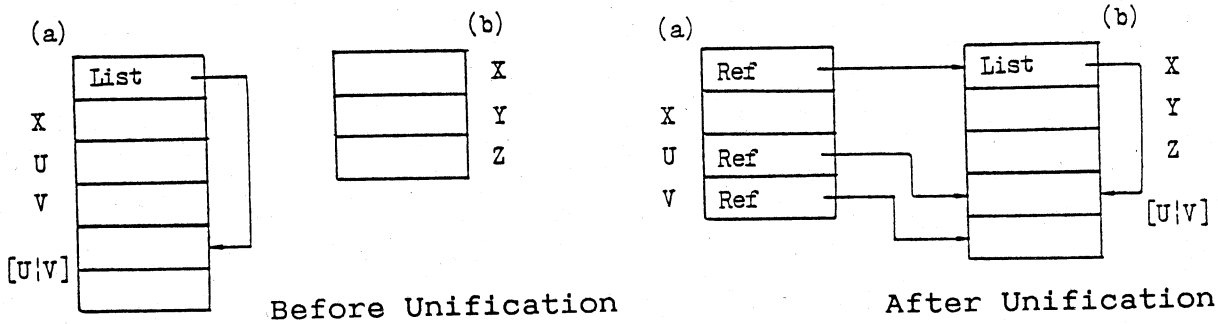
that we can execute deterministic unification faster by regarding the goal as a part of a stack of a sequential inference machine.

Next, we compare 6 and 7, 9 and 10 respectively. In 6 and 9, the goal size is smaller, and the overhead of copying and transferring is less. But the overhead of extracting literals from the clauses as new goals and the overhead of changing values of the parent goal variables become necessary. If we can find a fast algorithm for them, 6 and 9 may be efficient methods, but it is a subtle problem. 6 and 9 are different with respect to copying/sharing of literals but really have very similar form. In 6, the arguments for unification will be copied like the arguments of a subroutine call. In 6 and 9, it is difficult to execute deterministic unifications fast, owing to the rather complicated control among the goals (literals).

It may seem that the goal size of 11 is larger than the size of 8, but in fact, in 8, it is difficult to discard the unnecessary data because the variables are referred to through the variable number in literals. In the worst case, the size of the goal amounts to the size of the stacks of the sequential inference machine. Therefore, the goal size of 8 and 11 strongly depends on the application programs.

The goal size of 8 and 11 is larger than the size of 7 and 10. However, if we can cover the overhead through pipeline execution, 8 and 11 becomes more efficient than 7 and 10. No operation is needed to put back the values of the bound variables into the parent goal in 8 and 11, but the goal size may become too large.

To realize AND parallelism, we must basically lower the level of the goal to the literal level. If we can neglect the overhead of copying, we don't have to lower the level. In 7 and 10, the size of the goals seems to be not so much larger than the size in 6 and 9, but in 8 and 11 maybe the size becomes too large.




```
?-f([U|V],X),g(U,X),h(V,X).    (a)

f(X,Y) :- p(X,Z),q(Z,Y).       (b)
```

4.4. Multi Sequential Inference Machines

The methods mentioned above intend to construct highly parallel inference machines. To construct a parallel inference machine, we can think different approach that we connect sequential inference machines. In this model, it becomes most important to enhance the working rate of each sequential machine. If the all machines are executing their jobs, there is no need to distribute goals to other machines. So, the distribution of the goals doesn't happen so many. It is not so important to decrease the overhead of the goal distribution. 3 and 7 in Table 1 may seem to be good goal representations for these machines, but the amount of unifications caused by the goals of 3 and 7 is not so many. 4 and 8 are suitable for this machine. At this time, it is better to distribute the alternatives that is near to the root of the search tree as goals.

5. Summary

In this paper, the internal representation of the goals in a parallel inference machine is discussed. The selection of the representation is the trade off problem between the overhead of the copy and transfer of the goals and overhead of the complicated control among the goals. We think that a combination of methods like 7 and 8 in Table 1 that is changed dynamically according to the size of the goals and to the load of the system is the best method.

The evaluation of each overhead mentioned above through simulation is future work.

References

Haridi,S. and Ciepielewski,A., An OR-parallel Token Machine, Logic Programming Workshop 83, 1983.

Ito,N. and Masuda,K., Parallel Inference Machine Based on the Data Flow Model, Proc. of International Workshop on Highlevel Computer Architecture 84, 1984.

Moto-oka,T.,Tanaka,T. et al, The Architecture of a Parallel Inference Engine - PIE -, FGCS'84, ICOT, 1984.

Onai,R. et al, The Architecture and Software Simulation of the Parallel Inference Machine PIM-R , ICOT Technical Report TR-077, 1985 (in Japanese).

Yuhara,M. et al, A Unify Processor Pilot Machine for PIE, Proc. of the Logic Programming Conference '84, Tokyo, 1984.