A Preliminary Evaluation of the Activity Control Mechanism in PIE

Tsutomu Maruyama, Aida Hitoshi, Atsuhiro Goto

Hidehiko Tanaka, Tohru Moto-oka.

Faculty of Engineering, University of Tokyo

Abstract

This paper describes the features of the Activity Control Mechanism in PIE and the preliminary evaluation of its prototype. Because the Activity Control Mechanism in PIE is independent of Resolution Mechanism, the Activity Control Mechanism is simple and powerful enough.

1. Introduction

The OR-parallel execution of logic programs generates a lot of activities. Usually the number of them is enormous, because it is equivalent to the width of the search tree. Logic programs can be executed employing different problem solving strategies without changing their meanings. The strategy employed, however, affects the efficiency of execution radically. On the other hand, the extended features of logic programming languages like 'not' and 'guard' are needed in practical applications.

Therefore it is important to establish suitable Activity Control Mechanism in parallel machines. The roles of the Activity Control Mechanism are to manage the activation of the parallel activities according to both the problem solving strategy and the extended language features.

PIE (Parallel Inference Engine) is a highly OR-parallel inference machine based on a goal-rewriting model [Got83a] [Got83a]. PIE has many OR-parallel Unify Processors executing the activities in parallel. An activity in PIE is called a "goalframe", which is a reduced form of the combination of an initial goal, applied definitions, and binding environment. The relation between goalframes is too complicated to be recorded in goalframes themselves. Therefore, a tree-formed structure, called a "relation tree" is established to represent the relation between goalframes. Each goalframe corresponds to a leaf node of the tree. Operations on the tree node are carried out using the "node control commands".

The Activity Control Mechanism of PIE is simple because of the independence from the Resolution Mechanism. In this paper, we present the features of the Activity Control Mechanism in PIE, and discuss the performance of the prototype implemented on a simulator [Mar83,84].

2. The Feature of the Activity Control Mechanism

2.1. Feature

The global architecture of PIE is shown in Fig.1. The most principal modules of the Activity Control Mechanism are the Activity Controllers coupled with Memory Modules. All Activity Controllers are connected with one another through the Command Network. The Activity Control Mechanism in PIE is realized using the relation tree and node control commands. The overall feature of the Activity Control Mechanism is shown in Fig.2. Each Activity Controller has a Tree Node Memory to store a part of the relation tree. Goalframes in each Memory Module are managed through the corresponding tree node (goal-node) by the Activity Controller. An Activity Controller operates on the nodes of the relation tree, transferring the node control commands through the Command Network.

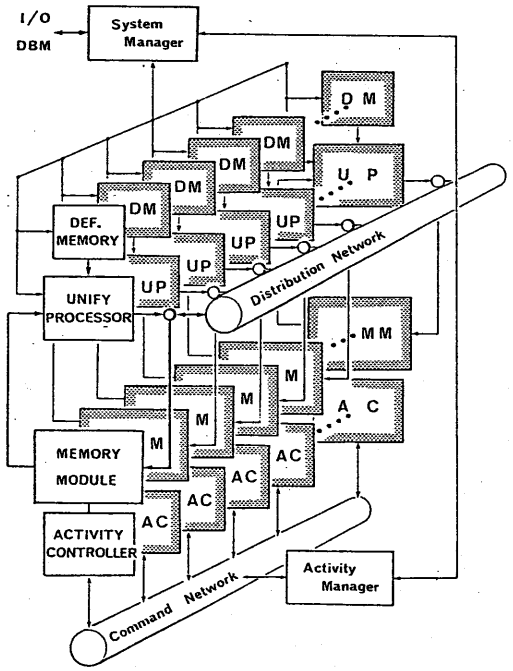

Fig. 1
The Global Architecuture of PIE

The roles of the Activity Control Mechanism in PIE is

(1) to control the activation of the parallel activities, namely goal-frames, according to the problem solving strategy and the language features,

(2) to manage the hardware resources.

The Activity Control Mechanism needed to realize these roles is simple, and the kinds of operations are few because of the independence.of the Activity Control Mechanism from the Resolution Mechanism. These roles are realized through only the three kinds of basic operations and a few kind of other operations for meta-logical predicates.

In the following sections, the features of the relation tree, node control commands and the operations on the relation tree are described.

2.2. Relation Tree and Node Control Commands

The nodes of the relation tree are broadly classified into four types, as below.

(1) a root-node

(2) relay-nodes (non-terminal)

(3) goal-nodes (terminal)

(4) fully expanded-nodes (terminal)

The root-node corresponds to the input process of an initial goal. In a multi-job environment, several root-nodes may exist. The relay-nodes are used to relate the goalframes in a system. Each relay-node specifies the relation among its parent-node and son-nodes as its attributes. A relay-node having only one son-node can be eliminated as an unnecessary node. Each goal-node corresponds to a goal-frame in Memory Module. Goal-nodes spread branches or are pruned through the resolutions of corresponding goal-frames. When OR-parallel unify process in Unify Processor ends in failure, or when a complete frame (null clause) is generated, the branch is fully expanded and is replaced by a fully expanded-node. Fully expanded-nodes can be pruned to save hardware resources.

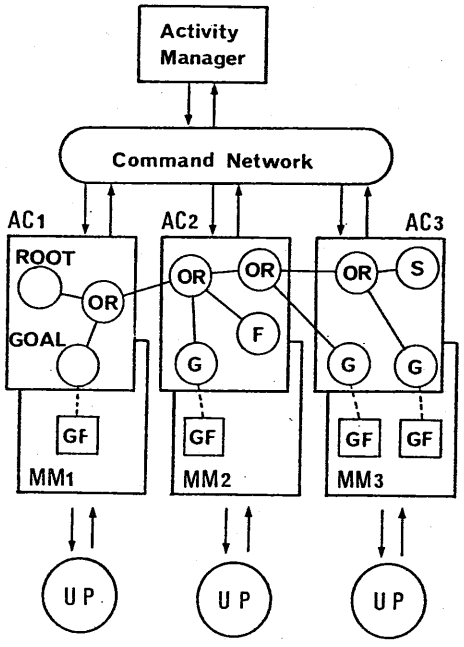

Fig. 2
An Overall Feature of
the Activity Control Mechanism in PIE

Each operation on tree node is executed by the Activity Controllers through sending/receiving node control commands among them. The operations of node control commands are specified by both commands themselves and the attributes of the destination nodes.

2.3. Operations

Basic operations on tree nodes are as follows.

(1) expanding the relation tree

(2) pruning fully expanded-nodes

(3) eliminating unnecessary relay-nodes

The Elimination of an unnecessary relay-node is not the essential operation, but useful to transfer the node control commands speedily and to save hardware resources. The extended language features such as meta-logical predicates 'not' and 'guard' need other operations. For example, following operations are needed to execute the 'not' and 'guard'.

(4) aborting goalframes

(5) suspending the execution of goal-frames

(6) resuming the execution of suspended goalframes

3. Prototype

In the previous chapter, we introduce the overall features of the Activity Control Mechanism in PIE. In this chapter, we describe the implementation example of the Activity Control Mechanism. This prototype is actually implemented as a software simulator [Mar83,84]. Because of the simplicity of the Activity Control Mechanism in PIE, this prototype can be implemented easily and have enough power to control activates. The following description is based on the specification of the simulator.

3.1. The Specification of Prototype

The extended features of logic programs need some special operations.

To explain clearly, a simple prototype logic programming language is presented. This prototype is mainly base on pure-Prolog. The execution order of the goal literals and the selection order of alternative definitions are principally free from the program expression. If no specification is given, the execution order of literals is usual left-to-right. However, the goal literals may be dynamically selected according to the problem solving strategy. If no specification is given, all solutions of an input goal are searched.

Three special notation are employed as meta-logical in this prototype language. Note that they are introduced for presenting the Activity Controlling Power of PIE and for using the conventional Prolog programs as sample programs of the PIE simulator. The language construct of logic programming should be considered through careful examination.

(1) negation as failure [Cla78a]:

not(P, Q,.., R)

Inverts the success and failure of the AND-connective of goal literals P, Q,..,R. Note that 'not' does not instantiate any variables.

(2) guarded clauses [Dij75a][Hoa78a]:

$$A :- G_1 \mid B_1.$$
$$A :- G_2 \mid B_2.$$
$$\vdots$$
$$A :- G_n \mid B_n.$$

The guarded parts are examined in parallel, then the body part whose guarded part has succeeded first is selected in a don't care nondeterministic sense.

(3) sequential application of definitions:

$$\text{seq } \{\ A :-- B_1,..,B_n.$$
$$A :-- C_1,..,C_n.$$
$$\vdots$$
$$A :-- D_1,..,D_n.\ \}$$

The definitions for the predicate 'A' should be sequentially applied. This feature is introduced in order to simulate some of the execution feature of sequential Prolog. The sequential application is also used to avoid resource exhaustion.

In addition, several built-in predicates, such as 'print', 'plus', 'minus' etc. are provided.

3.2. Node Attributes and Node Control Commands

A tree node constituting the relation tree of the prototype has a format shown in Fig.3. In this node format, the node attribute consists of a type and some flags, in order to express some kinds of complex attributes easily. The types and the flags of node attributes are classified into five groups, as listed in Fig.4-(1) to (5).

my type	my flags	my node id.
number of son-nodes		number of inactive son-nodes
----		parent-node id.
son type	son flags	son-node id.
son type	son flags	son-node id.
son type	son flags	son-node id.

[my type, my flags] : node attribute

[son type, son flags] : son-node attribute

node id. = { AC # | index to node memory }

Fig. 3
Tree Node Format in Prototype

```
types
    [AND]  : all success -> [SUCCESS],
             one failure -> [FAIL]
    [OR]   : one success -> [SUCCESS],
             all failure -> [FAIL]
    [AOR]  : all failure -> [FAIL],
             one success -> continue
    [NOT]  : not-node

flags
    [Sq]   : activate son-nodes
             sequentially
    [G]    : guard select node
    [E]    : eliminating node
    [P]    : pruned node
```

Fig. 4 (1) Relay-node Attributes

```
types
    [GOAL] : goal-node
    [GSS]  : guard success
             and suspend goal-node

flags
    [G]    : goal-node including
             more than one guard_bar
    [Su]   : suspended goal-node
    [O]    : OR flag
    [W]    : waiting for
             {END_RESOLVE} message
```

Fig. 4 (2) Goal-node Attributes

```
types
    [SUCC] : success-node
    [FAIL] : failure-node
```

Fig. 4 (3) Expanded-node Attributes

```
types
    [SON]  : son-node
    [GSS]  : guard success
    [SUCC] : success son-node
    [FAIL] : failure son-node

flags
    [W]    : waiting for <son> command
    [S]    : suspended
    [P]    : pruned
    [E]    : eliminating node
```

Fig. 4 (4) Son-node Attributes

```
types
    [Sq]    : sequentially applied
    [G]     : guarded clauses
```

Fig. 4 (5) Definition Attributes

The relay-node attributes, the goal-node attributes and the expanded-node attributes are exclusive with one another. The son-node attributes in Fig-4(4) are used in each relay-node to express the attributes of its son-nodes. The definition attributes in Fig-4(5) are used in order to realize the guarded clauses and the sequential applications. These attributes are introduced through the pre-processing of source programs and are attached to definition templates.

The node control commands in the prototype have a format as shown in Fig.5. They are listed in Fig.5-(1) and (2). Fig.5-(1) shows the commands from a son-node to a parent-node. Fig.5-(2) shows the commands from a parent-node to a son-node. In Fig.5 and in following sections, the node control commands are expressed as:

< destination node id.,
command name, argument list .. >

or simply as:

< command >.

COMMAND	some args	destination node id.
some arguments		source node id.

Fig. 5
Node Control Command Format

< d, success, a > :
node(a) succeeds and is pruned

< d, suspended-success, a > :
node(a) succeeds and is pruned,
but is suspended on parent-node

< d, failure, a > :
node(a) fails and is pruned

< d, guard-success, a > :
guard-part succeeds at node(a)
or at descendants of node(a)

< d, son, a, n > :
node(a) is the n-th son-node
of node(d)

< d, suspended-son, a, n > :
node(a) is the n-th son-node
of node(d), but is suspended

< d, son-change, a, b > :
son-node of node(d) is changed
from node(a) to node(b)

< d, eliminate > :
node(d) can be eliminated

Fig. 6 (1)
The Node Control Commands
from Son to Parent

< d, s-stop > :
receiver node or its descendants
should be pruned on success

< d, f-stop > :
receiver node or its descendants
should be pruned on failure

< d, activate > :
receiver node or its descendants
should be activated

< d, suspend > :
receiver node or its descendants
should be suspended

< d, parent-change, a, b > :
parent-node of node(d) is changed
from node(a) to node(b)

Fig. 6 (2)
The Node Control Commands
from Parent to Son

3.3. Basic Operations

In chapter 2, three basic operations are described. In this section, we describe the implementation example of these basic operations.

3.3.1. Expansion of The Relation Tree

The resolution in each Unify Processor corresponds to the relation tree expansion in the Activity Control Mechanism. Tree expansion features are implemented as the message transfer between the Activity Controller and the Unify Processor. The relation tree is expanded as shown in Fig.7-(1),(2).

(1) First, an Activity Controller sends a 'Resolve' message which includes a goal-node id. (P in Fig 7-(1)), its node attribute ([GOAL]) and the goalframe to the Unify Processor, and set [W] flag on the goal-node to indicate that the tree expansion for this goal-node is not complete yet. The Unify Processor resolves this goalframe as an OR-parallel unify process, and generates new goal-frames, if there are. The Unify Processor returns new goalframes as 'NewGF' messages. A 'NewGF' message includes the parent-node id. (P), the new node attribute ([GOAL]) of a new goal-node (S1 or S2), and a new goalframe. 'NewGF' messages can be sent to other Activity Controllers/Memory Modules according to the goalframe distribution strategy.
The Unify Processor also returns an 'End-Resolve' message, when the OR unify process for the input goal has terminated. The 'End-Resolve' message includes the input goalframe id. (P), the new attribute ([AOR]) and the number of son goalframes (2). When the number of son is zero, it means the OR-parallel unify process ends in failure. The Activity Controller which receives the 'End-

Resolve' message changes the node attribute and resets [W] flag. If some commands for the node are stored in the Waiting Command Memory, they are executed.

(2) The Activity Controller which receives the 'NewGF' message stores the new goalframe into the Memory Module. The Activity Controller makes a new goal-node in its Tree Node Memory. At this point, its node id. (S1, S2) is created. The Activity Controller sends a <son> command to the Activity Controller that has the parent-node (P) in order to link with the parent-node.

During the tree expansion, [W] flag is set on the goal-node (P). If a <son>, <success> or <failure> command arrives this node before the flag is reset, it is stored into the Waiting Command Memory, and wait until [W] flag is reset by 'End-Resolve' message. Actually, the Waiting Command Memory is used more generally. When Activity Controller receives a certain command, the Activity Controller checks the contents of the Waiting Command Memory whether any command is waiting or not.

3.3.2. Pruning Fully Expanded Node

An example pruning fully expanded nodes is shown in Fig.8-(1),(2). When the resolution in a Unify Processor ends in failure (node S2 in Fig.8-(1)), or when a complete frame (null clause) is generated, the branch of the relation tree node is fully expanded. Expanded tree nodes are pruned towards upwards. When a goal-node changes to a expanded-node, the Activity Controller sends a <failure> or <success> command to the Activity Controller which has its parent-node, in order to cut the link to its parent-node and to prune the fully expanded node.

A relay-node whose son-nodes are all pruned (S1 and P in Fig.8-(1)) also becomes a failure-node. Then, the fully expanded branches of the relation tree are dynamically pruned, sending <failure> commands towards the root-node in the relation tree.

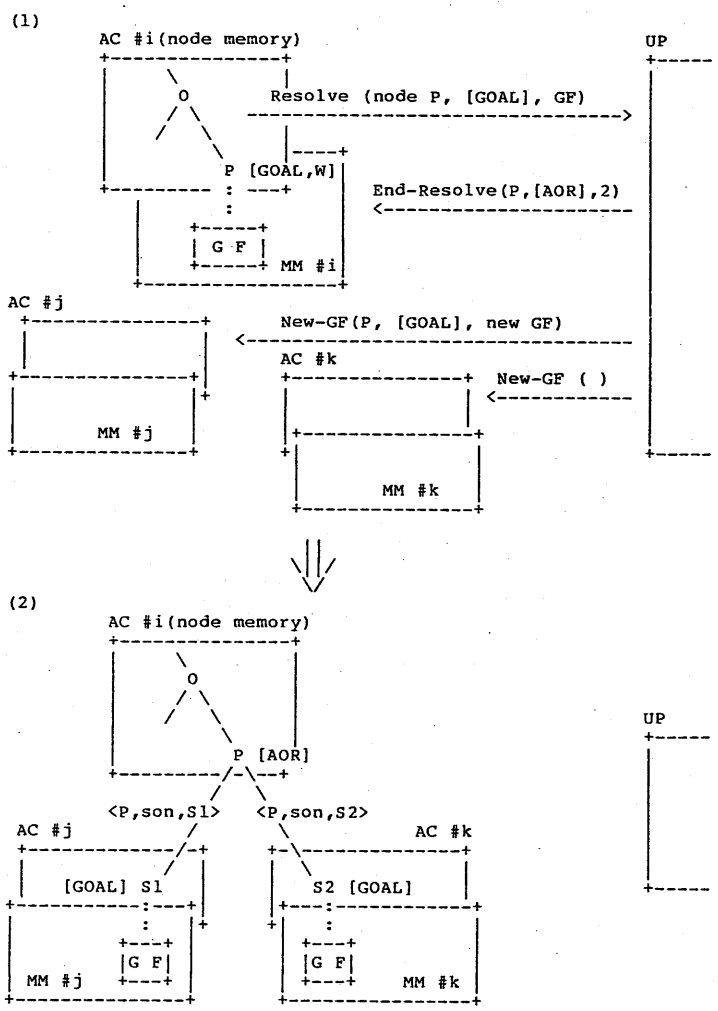

Fig. 7
Tree Expansion

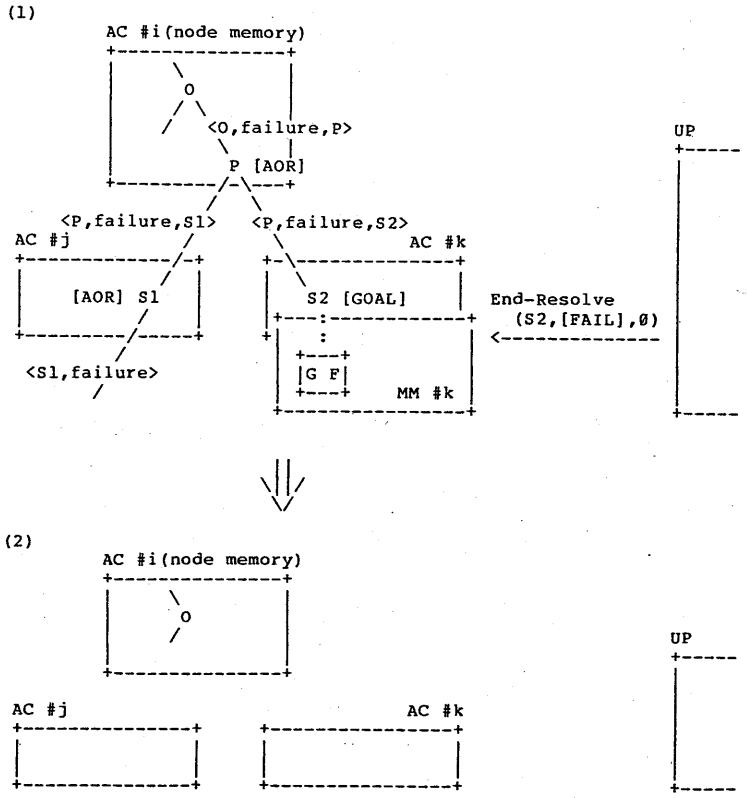

Fig. 8
Pruning an Expanded-node

3.3.3. Elimination of Unnecessary Relay-nodes

The nodes having only one son-node are unnecessary. These nodes should be eliminated in order to propagate node control commands speedily and to save the memory space. However, it is not easy to eliminate unnecessary relay-nodes because various commands pass on the nodes. Practically, four times command transfer is necessary for eliminating relay-nodes asynchronously. Therefore the elimination of these relay-nodes should be executed in lower priority than other commands in order to avoid both the congestion of the command execution and destination conflicts of commands in the Command Network.

An example of unnecessary relay-node elimination is shown in Fig.9-(1)~(3).

(1) If a node becomes unnecessary relay-node (B in Fig.9), the Activity Controller sets an eliminating flag [E] on its attribute, then sends a <son-change> command to the Activity Controller which has its parent-node A. When node A receives the <son-change> command, it changes son node id. to C and sets [E] flag on the son-node attribute, and sends a <parent-change> to its new son-node C.

(2) When node C receives the <parent-change> command, it changes its parent-node id. to A, and then sends an <eliminate> command to node B. When node B receives the <eliminate> command, it prunes itself, and passes the command to node A.

(3) When node A receives the <eliminate> command, it resets the [E] flag on the son-node attribute.

If a node receives a command such as <failure>, during its elimination, it passes the command to its parent-node and eliminates itself. If it receives a command such as <s-stop>, <f-stop>, <activate> or <suspend>, it simply passes the command to its son-node.

Even if the two contiguous nodes become unnecessary relay-nodes at the same time, they can't be eliminated at a time. One of them must wait until another relay-node elimination finishes, in order to perform the elimination safely. In Fig.9, if node C becomes unnecessary relay-node during the elimination of node B, node C sends a <son-change> command to node B. When node B receives the <son-change> command, it stores the command into the Waiting Command Memory until the elimination finishes. When node B is eliminated by the <eliminate> command, the Activity Controller which has node B passes the <eliminate> command to node A.

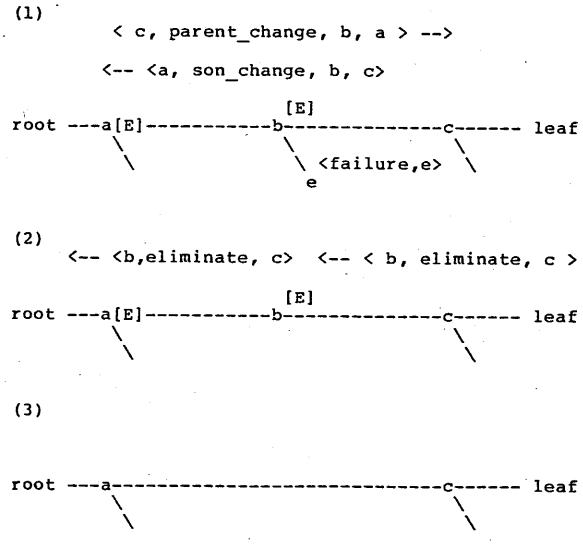

Fig. 9
Elimination of an Unnecessary Relay-node

3.4. meta-logical predicate The meta-logical predicates described in chapter 3, are executed as follows. The details of the execution can be found in [Mar84].

3.4.1. not

An example of the execution of "not" is shown in Fig.10.

(1) A goalframe including "not" (?- not(A), B. in Fig.10) is AND-divided into two sub-goalframes (?- not(A). and ?- B.), which are executed in parallel. On the relation tree, the node attribute is replaced by the attribute "AND". "AND" node is the special node. If one of son-nodes fails, "AND" node stop other branches of it using <stop> commands as shown in Fig.11.

(2) When the resolution of the sub-goalframe ?- not(A). ends in failure, A <success> command is sent to its parent AND-node. If a complete-frame (null clause) is generated, a <failure> command is sent to its parent AND-node, and the parent AND-node send a <stop> command to the sub-goalframe ?-B. and aborts it, or its descendent goalframes. Otherwise the goal-node attribute is replaced by the node attribute "NOT". "NOT" is a special node and inverts the result of its son nodes. If all descendents of a "NOT" node end in failure, the "NOT" node success. Otherwise, it fails.
The son goalframes of a NOT-node inherits the [O] flag and the descendent nodes of the NOT-node become [OR] nodes instead of [AOR].

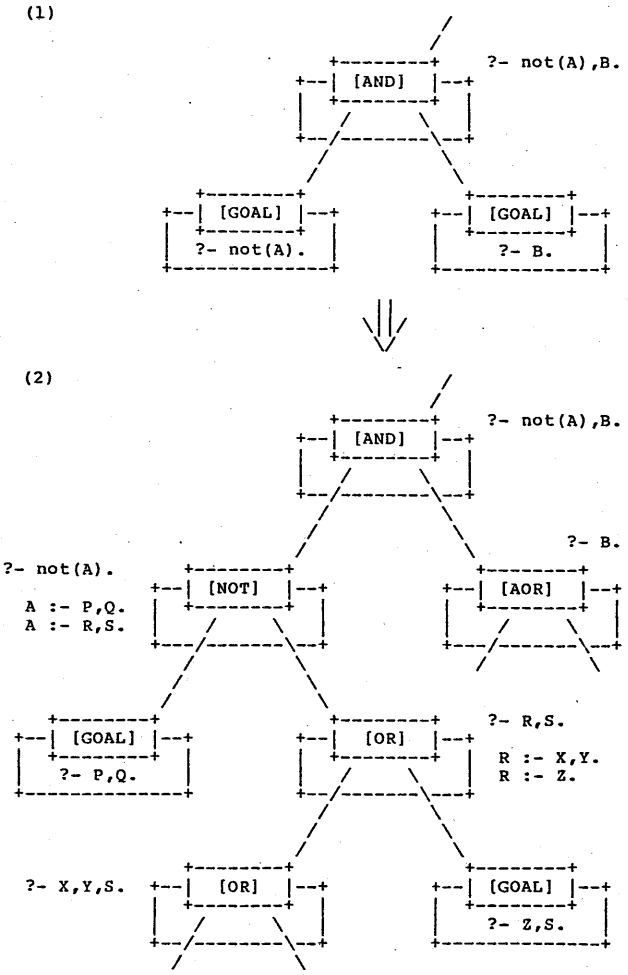

Fig. 10
An Execution Feature of Meta-Predicate "Not"

3.4.2. guarded clause

If definitions have the guard flag [G], this flag is set on the node attribute. The [G] flag on a relay-node indicates the introduction point of guarded clause. The descendent goalframes of the node having [G] flag inherit [G] flag which means that the goalframes include at least one guard-bar. When one of the descendent goal-frames executes guard-bar, a <guard-success> command is sent to its parent-node. when the parent-node receives the <guard-success> command, it sends <stop> commands to its other son-nodes and aborts the execution of its descendent goalframes. If the parent-node has [G] flag, it resets the flag and sends a <activate> command to the son-node from which the <guard-success> command arrived. If the parent-node doesn't has [G] flag, it only passes the <guard-success> command to its parent-node.

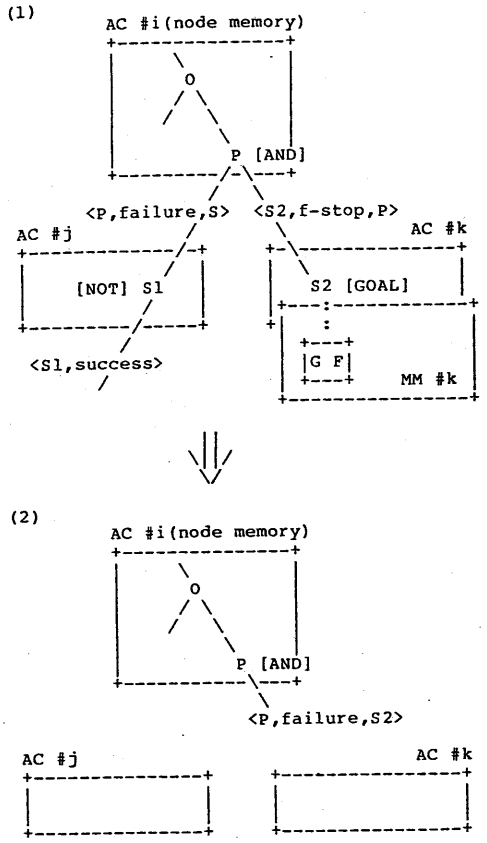

Fig. 11
Abortion of a Goalframe

3.4.3. sequential application of definitions

If definitions have the sequential execution flag [Sq], son-goalframes except for the first son-goalframe are set the suspended flag a [Su]. The goalframes having a [Su] flag are not executed until the goalframe receives a <activate> command. When the node having [Sq] flag receives a <success> or <failure> command from its first son, it sends a <activate> command to the next son-node.

4. Evaluation of The Prototype

In this chapter, we evaluate the the performance of the prototype described in the previous chapter.

4.1. factors

Node control commands are transferred among the Activity Controllers through the Command Network. The traffic of commands largely influences the total performance of PIE. Some kinds of commands, such as <stop> command for aborting goalframes and <activate> command for reactivating suspended goalframes, should be transferred as fast as possible. Consequently the following points should be considered.

(1) the strategy of goalframe distribution to minimize the command traffic

(2) the faster transfer of the node control commands through the Command Network

However, the goalframe distribution strategy must be investigated as the 'trading off' with effective usage of Unify Processors. PIE can not employ the distribution strategy only for minimizing the command traffic.

On the other hand, the facility of eliminating unnecessary relay-node is employed. Although the eliminating operation is rather costly, it seems to be effective because of the following reason.

(1) The Activity Controllers are not always busy, especially it seems that they are rather free in the first stage of execution. The eliminating operation can be performed during the free time.

(2) Each node control command has its priority according to its object. Commands such as <stop>, <activate> have a higher priority than the commands used for relay-node elimination. Therefore the commands for eliminating operation do not disturb other important commands because of their low priority.

(3) The length of command's path is shortened through relay-node elimination, making the propagation time reduced.

Therefore the main factors which decide the performance of the Activity Control Mechanism are considered as follows.

(1) the frequency of commands

(2) command traffic

(3) the execution time of the commands by Activity Controllers and the delay time of the Command Network

In the following sections the performance of the prototype of the Activity Control Mechanism examined on the factors above.

4.2. simulator

In this section, the simulation assumptions are presented. The detailed description of the simulator can be found in [Mar84].

Through the preliminary examination of the Unify Processor pilot machine [Yuh83,84a,84b], the time to generate a new goalframe is almost in proportion to the length of the generated goalframe. Its execution time of it is about 4~7 micro steps per one cell. In this simulator, 7 clocks are assumed.

The effect of the delay time necessary for transferring a goalframe through the Distribution Network is examined, compared the with the execution time in a Unify Processor. In this simulator, the Distribution Network is assumed that

(1) the network delay time is almost determined by the number of transferred cells

(2) the network delay increases in proportion to the logarithm of the number of Unify Processors.

The activation strategy of goalframes, especially the goalframes including meta-predicates, in the Memory Module may influences the frequency of the commands. In the following simulations, FIFO (first in first out) is assumed.

4.3. The Frequency of Commands

The performance of the Activity Control Mechanism should be influenced by not the total number of commands but the peak frequency. The simulation results are shown in Fig.12~Fig.16. In Fig.12~Fig.14, the sample programs are [LC] (verification program of a small logic circuit), and in Fig.15~16, [LL2P] ((lisp + logic) * 2 = prolog). Fig.12 and Fig.13 show the frequency of commands in the case of 16 Unify Processors. Fig.14 shows the number of relation tree nodes in the same case. Fig.15 and Fig 16 show the frequency of commands in the case of 256 Unify Processors. In Fig.13 and Fig.16, unnecessary relay-nodes are eliminated. In Fig.12~16, frequency is measured by the number of commands generated in each 2000 simulation clocks. In Fig.12, a high peak is observed in the last stage of the execution. This peak is mainly consists of <success> and <failure> commands. On the other hand, there is no peak in Fig.13, because the elimination of unnecessary relay-node keeps the size of the relation tree small, as shown in Fig.14. In this result, the elimination of unnecessary relay-nodes increases the total number of commands but equalizes the frequency in each stage of program execution.

Fig.15 and Fig.16 shows that the elimination of unnecessary relay-node increases not only the total number of commands but also the frequency in all stages. However, the number of the <failure> commands are decreased by the elimination of unnecessary relay-nodes. In this prototype, these commands have lower priority than other commands, and if a <failure> or <success> command passes on the eliminating nodes, the node is pruned immediately, and the commands for this node elimination aren't executed no more.

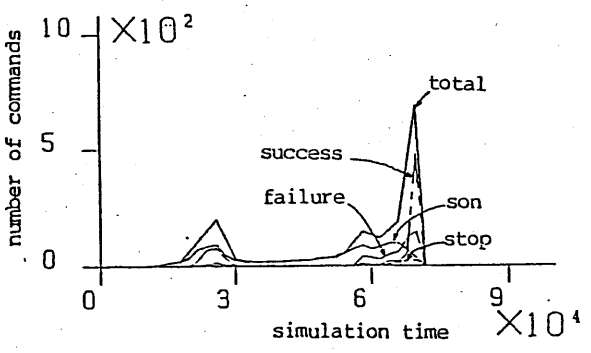

Fig. 12
Frequency of Commands
without Elimination of Unnecessary Relay-nodes
[LC] : Up = 16

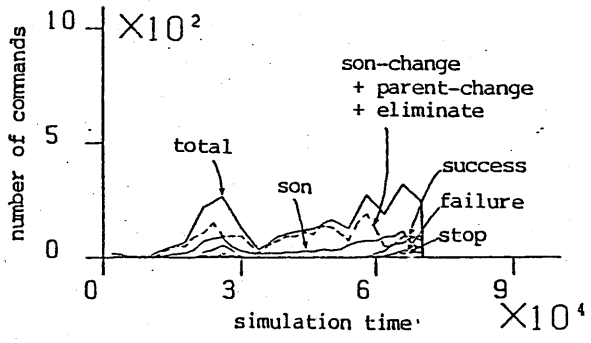

Fig. 13
Frequency of Commands
with Elimination of Unnecessary Relay-nodes
[LC] : Up = 16

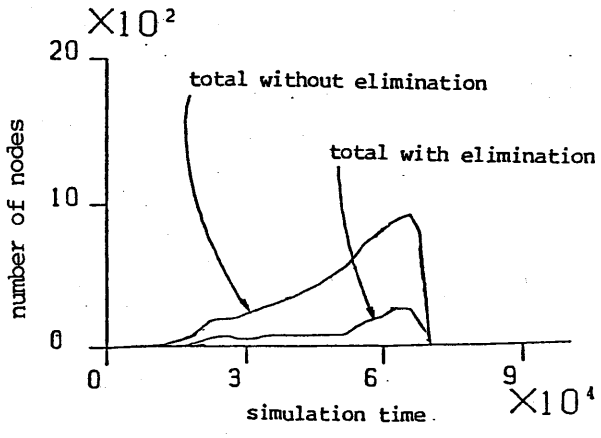

Fig. 14
Total Number of Tree nodes
[LC] : Up = 16

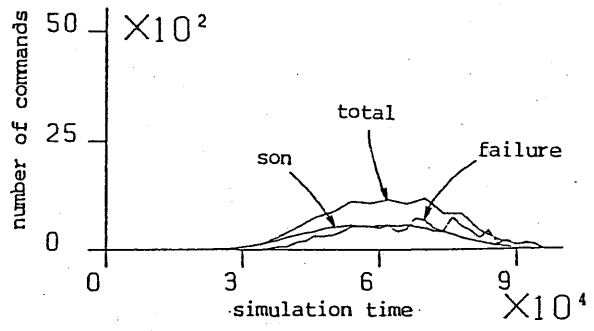

Fig. 15
Frequency of Commands
without Elimination of Unnecessary Relay-nodes
[LL2p] : Up = 256

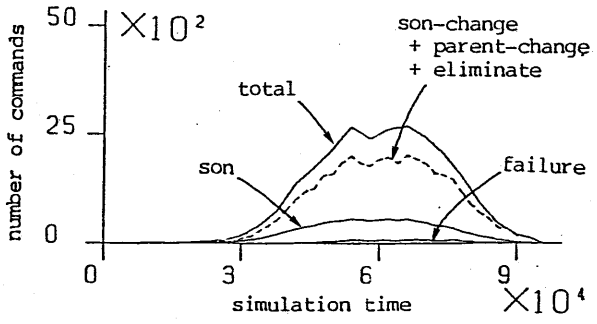

Fig. 16
Frequency of Commands
with Elimination of Unnecessary Relay-nodes
[LL2p] : Up = 256

In Fig.12~Fig.16, an Activity Controller must execute a command 4~6 times faster (130~200 clocks) than the resolution speed of a Unify Processor at the peak of the frequency. It is not difficult to realize this execution speed on the real machine.

4.4. the execution time of a command and the Command Network delay

The increase of the execution time of a command and the delay time of the Command Network decrease the performance of the Activity Control Mechanism. The main reasons are as follows.

(1) After all the goalframes are resolved, relation tree is pruned. The time pruning the whole tree increases almost in proportion to the network delay and the command execution time.

(2) The network delay of <stop> commands may causes wasteful execution of goalframes.

The simulation results are shown in Table.1, with the sample program [LL2P], in the case of 64 Unify Processors. Table.1 shows that the Activity Control Mechanism doesn't decrease the total performance if the unnecessary relay-nodes are pruned. Sample program [LL2P] doesn't include meta-logical predicates. All relay-node attributes are [AOR], and all relay-nodes are candidates for eliminating. The size of relation tree is kept always small. If the sample program includes the meta-logical predicates, the elimination of unnecessary nodes doesn't always realize better performance than the performance without the elimination. The reasons are as follows.

(1) During the execution of the program including meta-logical predicates such as "not", a lot of <stop> commands are used to abort execution of goalframes. If the goalframes to be aborted are distributed to many Activity Controllers, the abortion take a lot of time because of the Command Network delay.

(2) In this simulation, [AND] and [NOT] nodes are not eliminated. The elimination of node of these types are rather complicate. If these types nodes are distributed in many Activity Controllers, it takes a lot of time to prune the whole relation tree in the last stage of program execution.

Therefore the goalframe distribution strategy considering of the meta-logical predicates is important. The distribution strategy considering the meta-logical predicates are not implemented on the simulator yet. This kind of distribution strategy is one of the future works.

4.5. Command Traffic

The command traffic depends on goalframe distribution strategy. The effects of the following strategies on the Resolution Mechanism have been examined through the simulations.

(1) first self:
When a Unify Processor resolves a plural number of goalframes from one input goalframe, the first resolvent is returned to the coupled Memory Module, and others are distributed among all Memory Modules randomly.

(2) empty self:
When a Unify Processor generates a goalframe, the goalframe is returned to the coupled Memory Module if there is no goalframe in it. If there are any, the goalframe is distributed to others.

(3) random:
The goalframes are distributed quite randomly.

Through the simulation of the Resolution Mechanism, the effect of goalframe distribution strategies can be summarized as follows:

(1) An effective distribution strategy should be employed in the first stage of execution.

(2) A simple distribution strategy using the number of goalframes in the coupled Memory Module is enough effective in the processing environment of PIE.

(3) After the number of goalframes exceeds the number of available Unify Processors, the distribution strategy does not influence to the total performance radically.

The simulation results of command traffic are shown in Table.2, where sample programs are [LC] and [LL2P]. The three strategies mentioned above are examined. Table.2 shows the total number of commands transferred by the Command Network in each 2000 simulation clocks. The command traffic is very little, even if the distribution strategy is empty self. The number of the commands sent out to the Command Network is at most 8 per one Activity Controller per 2000 clocks. However, if sample programs have more parallelism, the command traffic would become heavier.

Anyway the command traffic will not become so heavy. From the point of the command traffic, it is important to distribute the goalframes according to meta-logical predicates as described in the previous section.

5. Conclusion

We have described the features of the Activity Control Mechanism in PIE and the prototype implemented on a simulator. The Activity Control Mechanism in PIE is basically independent of The Resolution Mechanism. The prototype shows that the basic controls of the parallel activities are executed easily, and meta-logical predicates such as introduced in the prototype can be executed simply, through the con-

Table.1
The Effect of
the the Execution Time of Command
and the Delay Time of the Command Network

parameters (clock)		Simulain time (clock)	
Exectuion Time of a Command	Delay Time of The Commands Network	No Eliminaitin of Relay-node	Elimination of Relay-node
0	0	159182	159182
50	200	165132	160282
50	500	170532	162802

Table.2
Command Traffic:
Number of Commands
per each 2000 Simulation Clocks

sample program	destinatin of commands	distribution strategies		
		random	first self	empty self
[LC] UP = 16	to Network (to self)	153 (69)	40 (88)	122 (100)
[LL2P] UP = 64	to Network (to self)	804 (196)	253 (756)	829 (472)

trolling of the activities by the Activity Controllers independent of the Resolution Mechanism. We will examine the performance of the Activity Control Mechanism in detail.

6. Acknowledgements

The authors would like to express the thanks to the member of the private group SIGIE (special interesting group on inference engine) for helpful discussions.

References

[Cla78]. Clark, K.L., "Negation as Failure," pp.293-322 in Logic and Data Bases, ed. Gallaire, H.and Minker, J.,Plenum Press, New York (1978).

[Dij75] Dijkstra, E.W., "Guarded Commands, Nondeterminacy and Formal Derivation of Programs," CACM Vol. 18(8) pp. 453-457 (Aug. 1975).

[Got83a]. Goto, A., Aida, H., Maruyama, T., Yuhara, M., Tanaka, H., and Moto-oka, T., "On the Parallel Inference Engine PIE (Japanese)," Proc. of the Logic Programming Conference '83, ICOT, (March 1983).

[Got83b]. Goto, A., "A Highly Parallel Inference Engine Based on Goal-rewriting Model: PIE," Doctorial Dissertation, Department of Information Engineering, University of Tokyo, (Dec. 1983).

[Hor78]. Hoare, C.A.R., "Communicating Sequential Processes", CACM Vol.21(8) pp. 666-677 (Aug.1978).

[Mar83]. Maruyama, T., Yuhara, M., Aida, H., Goto, A., Tanaka, H., Moto-oka, T., "A Higly Parallel Inferance Engine : PIE -- An Evalution of Effective Parallelism and Performance -- (Japanese)," Technical Research Report, EC83-39, IECE of Japan, (Dec. 1983).

[Mar84]. Maruyama, T. "A Preliminary Evaluation of a Highly Parallel Inference Engine PIE" (Japanese)," Master's Thesis , Department of Information Engineering, University of Tokyo, (Feb. 1984).

[Yuh83]. Yuhara, M., Aida, H., Goto, A., Tanaka, H., and Moto-oka, T., "Unify Processor and its Reduction Algorithm of the Highly Parallel Inference Engine -- PIE (Japanese)," Technical Research Report, EC83-30, IECE of Japan, (Oct. 1983).

[Yuh84]. Yuhara, M., "A Unify Processor of Highly Parallel Inference Engine PIE (Japanese)," Master's Thesis , Department of Electrical Engineering, University of Tokyo, (Feb. 1984).