Annual Report of the Engineering Research Institute
Faculty of Engineering, University of Tokyo
Vol.45
Received June 24, 1986

A Design of Structure Memory Pilot Machine of PIE

Keiji HIRATA* and Hidehiko TANAKA**

SYNOPSIS

In this paper, we describe the hardware design of the structure memory pilot machine which we are now developing. This system is composed of an inference unit and a structure memory, and is a prototype of the Parallel Inference Engine PIE. The system has various dedicated hardware to support a structure data sharing technique. As the inference unit has two associative memory chips, we can search tables efficiently. Since the structure memory has two independent sequencers, lazy fetch operations can be executed immediately. Moreover, garbage collection is also hardware-supported.

1. Introduction

We have been developing a Parallel Inference Engine called PIE [1]. A computational model for PIE is the goal rewriting model [2], and this model is basically implemented with a complete copying method. PIE is a MIMD type machine. Hence, processing elements communicate with each other by transferring data through networks. The data express goals in the Prolog sense. In order to estimate machine performance in detail, we have so far built two hardware simulators, which operate as a unification processor and a processing element (Inference Unit : IU) respectively [3][4].

We are now constructing a structure memory hardware simulator, which consists of one IU and one Structure Memory (SM) [5]. The SM is introduced to support a structure data sharing technique on hardware level [6]. The simulator will be able to give us information about network traffic between IU and SM, and the optimal configuration of SM.

2. A Model of Structure Memory

2.1 Structure Data Sharing Technique

As an execution goes on, ground instances within goals usually grow up. For the structure data sharing technique, the ground instances are forced to be separated from goals dynamically, and then stored into a memory shared by IUs. The stored data are fetched afterwards when necessary. The merits of this technique are as follows :

1. reducing goal size, therefore small transferring overhead
2. small copying overhead when generating goals
3. reducing unification time

In order to execute the three original kinds of operations efficiently, we should introduce SM with a high level interface and provide dedicated hardware for IU.

---

* Graduate Student, Fac. of Eng., the University of Tokyo
** Assoc. Professor, ditto.

株式会社富士通研究所の委託研究「推論マシン方式の調査研究」の報告

2.2 Lazy Fetch

PIE uses complete copying when generating goals, so each goal always contains the whole environment and structures. According to this method, each goal carries its structure data lazily which include undefined variables, and fetches the rest (ground instances) from SM when necessary. We call the operation lazy fetch (fig.1). Since lazy fetch operations occur during unification, we require short response time. The transmission for lazy fetch between IU and SM is like direct memory access (DMA). We will provide a lazy fetch buffer which works like a cache memory within the IU.

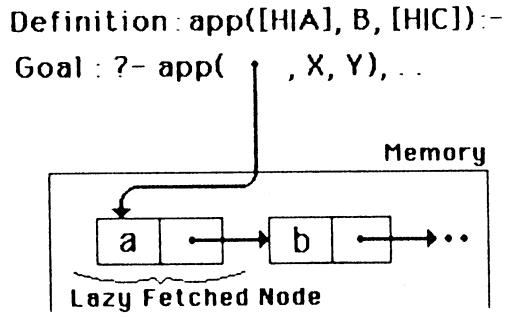


Fig.1 Lazy Fetch

2.3 Separating and Storing Ground Instance

When generating new goals, leaf nodes are checked if they are ground instance or not (fig.2). Then new goals have to know the addresses into which isolated ground instances have been stored. The SM should deliver the empty address to every IU in advance. Therefore, a store command includes a ground instance and an empty address (fig.3).

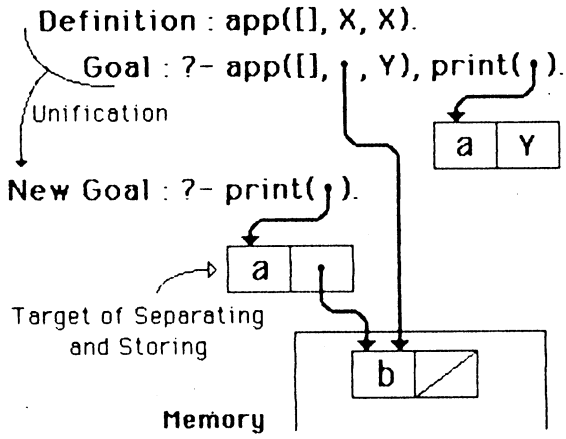


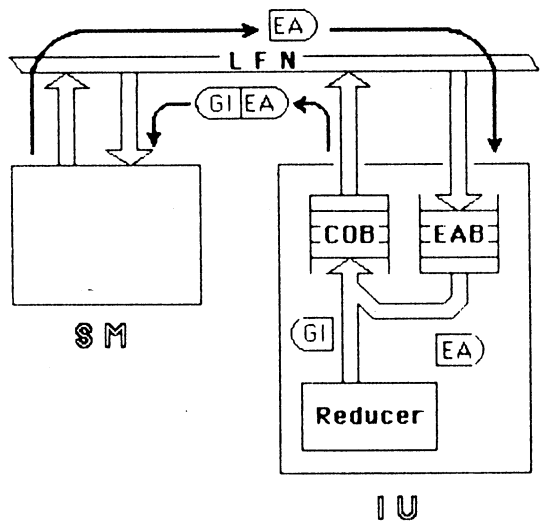
Fig. 2 Separating and Storing of Ground Instance

2.4 Reference Count Command

When producing new goals, IUs issue reference count commands to SM. Reference count commands are optimized within IU before they are transferred (fig.4).

2.5 Logical Image of Structure Memory

In our system, one cell means one word (32 bits), so we express a list node by two cells: a vector node by the number of the arguments + 2. Since we have to manage vari-sized nodes, compacting garbage collection (GC) is required. In a naive implementation, every time GC occur, all the pointers which refer to the nodes stored in SM must be adjusted. To avoid this, we introduce an Address Translation Table (ATT) and store the node itself into the Vari-sized Cell Memory (VCM) (fig.5).



LFN Lazy Fetch Network
COB Command Output Buffer
EAB Empty Address Buffer

Fig.3 Management of Empty Address

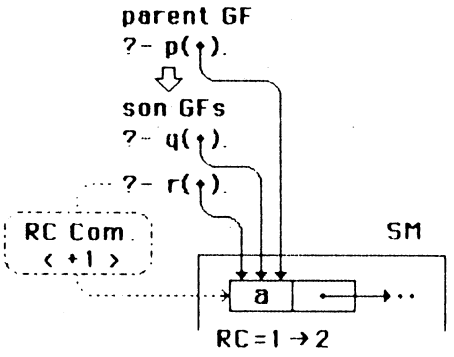
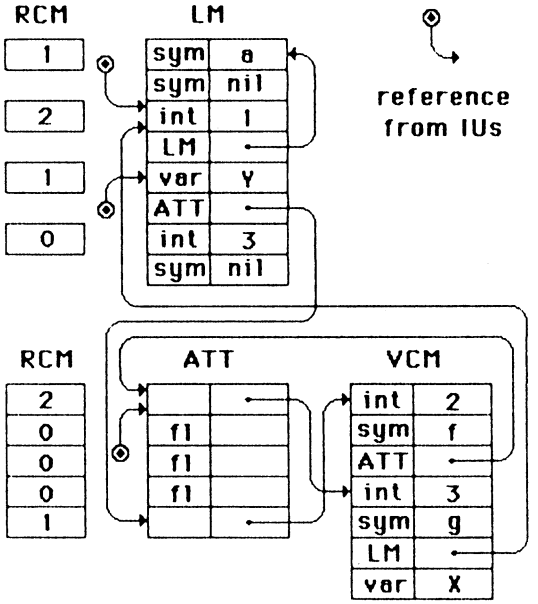


Fig.4 Processing of RC Command

Accordingly, the references from goals to SM necessarily go through the ATT, and so do inner SM accesses. Using ATT, we can shut pointer replacements for GC in SM. On the other hand, we should treat whole list nodes only in the List Memory (LM) to avoid such indirect accesses.

2.6 Garbage Collection

Basically, we adopt reference count garbage collection (GC). Free nodes in LM and ATT are administered by a free list in each.



LM : List Memory
ATT : Address Translation Table
VCM : Vari-sized Cell Memory
RCM : Reference Count Memory

fl : free list tag

Fig.5 Logical Image of Structure Memory

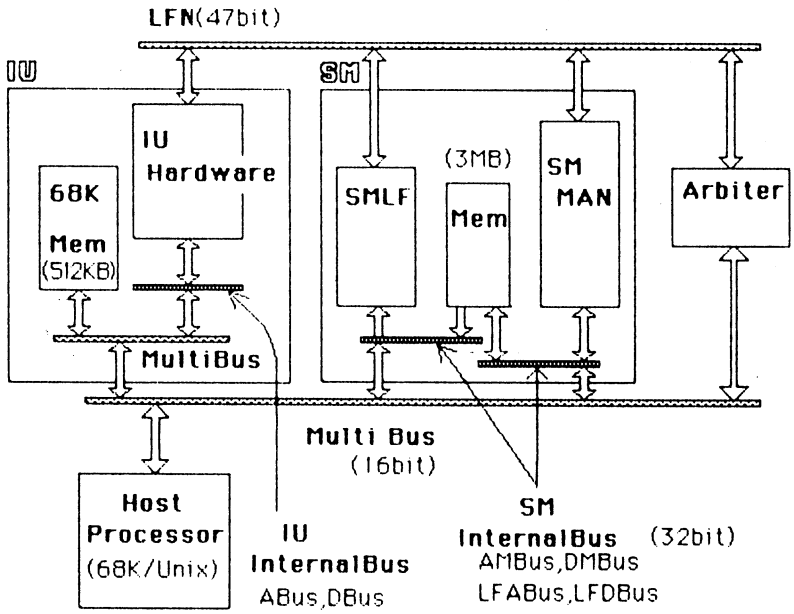


Fig.6 Simulator Hardware Configuration

Since nodes of ATT and VCM correspond to each other, the moment an ATT node becomes garbage, so does the corresponding VCM node. Then, free nodes of VCM are appended to different free lists according to size. This method can allocate a VCM node quickly, and make compaction of VCM space delayed as possible until VCM is too segmented.

3. Configuration of Structure Memory Hardware

3.1 Organization

Our simulator has a IU and a SM, which are connected with lazy fetch network (LFN) (fig.6). On the IU side, only the part to support the structure data sharing technique is constructed as hardware. That is, the dedicated hardware processes lazy fetch operations, separating ground instances, management of empty addresses, and generating the command packets.

As lazy fetch requires short response time, SM has two sequencers independently working on the same clock signal : one only for lazy fetch and another for the other operations (storing ground instances, mangement of empty addresses and GC). We call the two blocks SMLF and SMMAN, respectively. 3Mbytes shared memory is banked and used as LM, ATT, VCM and reference count memory (RCM). It may happen that SMLF and SMMAN access the same bank simultaneously. If that happens, the access conflict is detected on hardware level and the SMLF is able to fetch data with no delay, while the SMMAN is forced to wait for one micro cycle.

There are three kinds of requests to the LFN arbiter:

(a) lazy fetch (from IU)
(b) transferring command packet (from IU and SMMAN)
(c) transferring empty addresses (from SMMAN)

The request (a) has the highest priority. Once IU makes a lazy fetch request, LFN is occupied until SMLF returns a node. That is, a lazy fetch request sends an SM address from IU to SMLF, and returns a ground instance node at a time. In (b), the command packet consists of commands for ground instance storing, reference count and empty address request, and is transmitted together on proper timing. In response to the request (c), SM sends a packet which consists of empty addresses of LM or ATT to an IU. We briefly summarize the specification of the simulator in table 1.

We want to make the simulator easy

Table 1
Specificaton of The Simulator

```
Sequencers : Am2910A x 3
ALU        : Am29116s (cascaded) x 3
Clock      : 5 MHz
Word size  : 32 bits

Inference Unit side
    * IU hardware
        500 TTL ICs
        7 boards
        Memory : 112 KB
        Two associative memory chips
                    (made by NTT)
        micro instruction width
                        : 168 bits
    * 68K micro processor + 512 KB
Structure Memory side
    Shared memory : 3 MB
    * SMLF
        200 TTL ICs
        2 boards
        micro instruction width
                        : 92 bits
    * SMMAN
        400 TTL ICs
        5 boards
        micro instruction width
                        : 128 bits
Lazy Fetch Network
    width : 47 bits
    transfer rate : 20 MB/Sec
```

to modify and debug We have designed horizontal type micro instruction formats. The system includes some hardware dedicated for debugging.

3.2 Inference Unit

Fig.7 is the block diagram of the dedicated hardware of IU side. In fig.7, A Motorola 68000 processor, which executes unification, goal mangement and so on, can access only the shaded registers and memories. For example, during a lazy fetch operation, 68K writes an SM address into g_cell. Then the IU hardware is evoked and the actual lazy fetch occurs. As a result, IU hardware fetches the desired ground instance node into lazy fetch buffer (LFB) and sets g_cell to the LFB index of the stored node. As we mentioned above, IU hardware has two associative memory chips : one for the lazy fetch buffer table (LFBT) and another for the reference count command buffer. LFB behaves like a cache memory (fig.8).

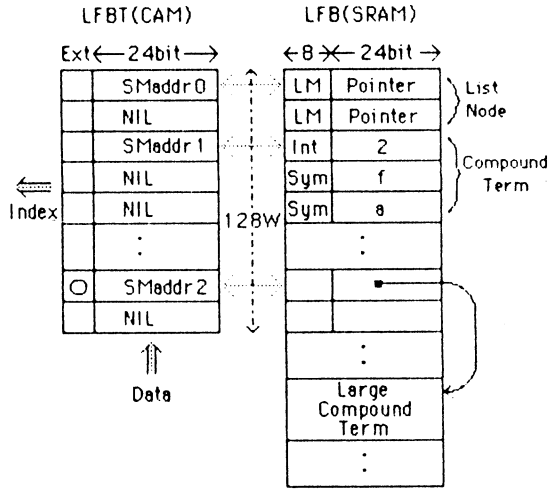


Fig. 8 Operation on LFBT and LFB

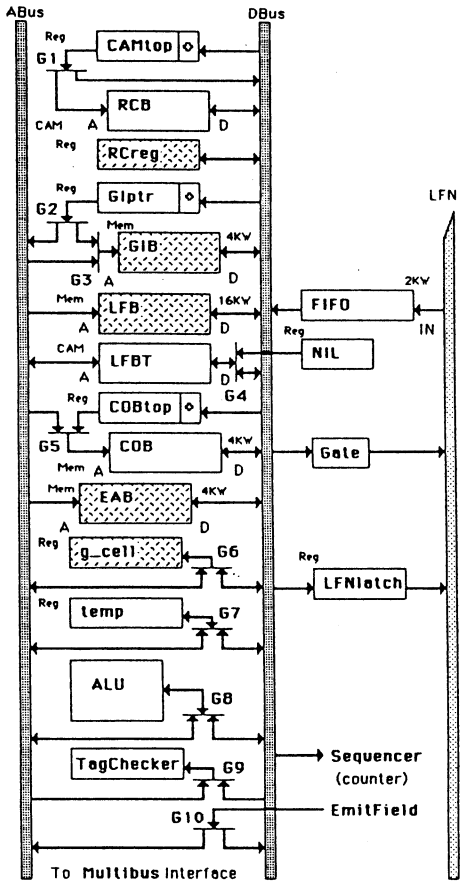


Fig. 7 IU Hardware Block Diagram

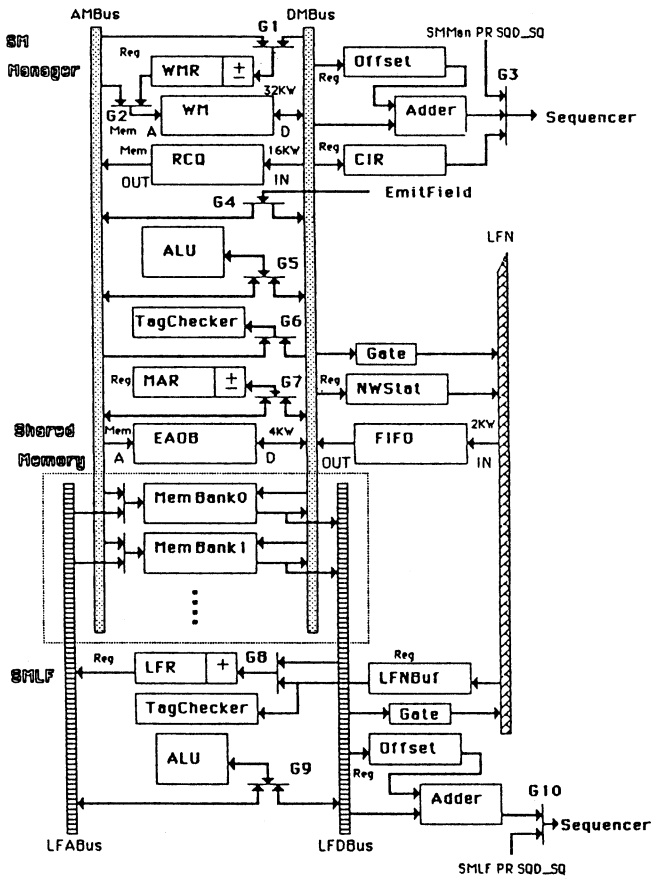


Fig. 9 SM Hardware Block Diagram

LFBT holds SM addresses on which lazy fetched nodes have been stored in the SM. An index of LFBT corresponds to an index for a node stored in LFB. We use an associative memory chip as LFBT, whose size is 32 bits x 128 words. We can collect reference count commands to the same address and add them together. In order to do such optimizations, we use the same kind of associative memory chip as a reference count command buffer.

3.3 Structure Memory

Fig.9 is the block diagram of structure memory. SMLF is usually idle and works only when receiving an SM address. Since register LFR is auto-incrementable, we are able to read shared memory at 1 word on every micro step.

SMMAN interprets command packets from IU, then stores ground instance nodes, sends out empty address packets and manages reference count commands. The detailed algorithms are omitted because of space limitation. Zero propagation may follow an updating reference count and VCM compaction may occur after an allocation of a VCM node. It is a problem to schedule above jobs in optimum. According our principle, only when SMMAN has nothing to do other than an updating reference count, SMMAN can do the job. And SMMAN manipulates shared memory as blocks allocated on demand.

4. Remarks

We described the hardware design of structure memory hardware simulator. The simulator is now under debugging. We believe that the debugging will finish by this fall. After that, we will begin to evaluate the system. We will also study structure memory support for higher level functions and process communication.

REFERENCES

[1] Moto-oka, Tanaka, Aida, Hirata and Maruyama : "The Architecture Of A Parallel Inference Engine - PIE -", FGCS'84, ICOT (Nov.1984).

[2] Goto, Tanaka and Moto-oka : "Highly Parallel Inference Engine PIE - Goal Rewriting Model and Machine Architecture -", New Generation Computing, Vol.2, No.1, OHMSHA (1984).

[3] Yuhara, Koike, Tanaka and Moto-oka : "A Unify Processor Pilot Machine for PIE", Logic Programming Conference '84, ICOT (March 1984).

[4] Tarui, Koike, Tanaka and Moto-oka : "An Evaluation of Inference Unit Hardware Simulator of PIE", The 32nd Annual Convention, 1R-5, ISPJ (March 1986).

[5] Hirata, Inomata, Tarui, Matsubara, Koike, Tanaka and Moto-oka : "A Design of Structure Memory Pilot Machine of PIE", Tech. Res. Rep. EC85-64, IECE (Feb. 1986).

[6] Hirata, Tanaka and Moto-oka : "A Model of Structure Memory for PIE", Architecture Workshop In Japan '84, ISPJ (1984).