THE IX SUPERCOMPUTER FOR KNOWLEDGE BASED SYSTEMS

Tetsuya Higuchi, Tatsumi Furuya, Hiroyuki Kusumoto, Ken'ichi Handa, and Akio Kokubu

Reprinted from the Proceedings of the
Fall Joint Computer Conference
November 2-6, 1986 Dallas Texas

THE IX SUPERCOMPUTER FOR KNOWLEDGE BASED SYSTEMS

Tetsuya HIGUCHI, Tatsumi FURUYA, Hiroyuki KUSUMOTO,
Ken'ichi HANDA, and Akio KOKUBU

Electrotechnical Laboratory
1-1-4 Umezono, Sakura-mura, Niihari-gun, Ibaraki 305, Japan

Abstract. This paper describes an IXM machine for the parallel processing of large knowledge based systems written in a semantic network language, IXL. The potential performance of IXM is thousands faster than conventional machines, as IXM can find all the solutions of the query simultaneously. IXM consists of a pyramid-shaped network(PYN) and one thousand proccesing elements(PEs). IXM utilizes two thousand associative memories in order to exploit the concurrencies in the fundamental operations of semantic network: association, set operation, and marker propagation. A network processor at each node of the PYN has an associative memory for parallel marker propagation. A PE also has two types of associative memories; one for the parallel processing of association and set operation, the other for exploiting the parallelism in IXL language interpreter described in non-procedural instructions.

1. INTRODUCTION

The semantic network is one of the most important representation schemes to develop knowledge based systems[1,2]

The authors are developing the IX[iks] system which includes a semantic network language for knowledge representation (IXL)[3,4] and a massively parallel hardware (IXM)[5]. The objective of the IX system is to support all the processes of semantic network processing in an integrated fashion: from using IXL in the modeling and description of knowledge based systems to their parallel execution on an IXM machine.

This paper focuses on the architecture of IXM. IXM is a high performance multiprocessor thousands of times faster than current machines used for semantic network processing. IXM consists of a pyramid-shaped network and one thousand processing elements.

It is very important for the design of the semantic network machines[6,7] to utilize the parallelism in three fundamental operations in the semantic network: marker propagation, association, and set operation.

Therefore, the network processor located at every node of the pyramid-shaped network employs an associative memory in order to exploit the concurrency in marker propagation.

Furthermore, the processing element also includes two kinds of associative memory; one to perform association and set operations in the SIMD manner, and the other to exploit the concurrency in an IXL language interpreter.

Section 2 describes the background of the semantic network and the programming language IXL since the IXM architecture design is IXL oriented. Section 3 is an overview of IXM. The design features are discussed in Section 4. Section 5 describes the processing element and Section 6 describes the pyramid-shaped network.

2. BACKGROUND

2.1 Semantic network

The advantage of the semantic network is the representation of declarative knowledge. Taxonomical knowledge and inheritance can be represented more naturally than predicate logic. Inheritance of properties from the super class is very useful to describe especially large knowledge bases because it can save memory spaces. Compared with frame systems, the meaning of the relation between concepts can be defined more flexibly. Procedural knowledge, however, can not be expressed well using only the semantic network. Therefore, semantic network languages have to include methods for procedure addition.

The semantic network can be processed efficiently by assigning marker bits to each node of the network2. A marker bit is a one-bit flag on which the result of processing is stored. Fundamental operations in the semantic network are performed using maker bits. For example, the question, "Which member belongs to both A-group and B-group ?" is asked in the semantic network of Figure 1.

In order to find the solution, the association operation is executed firstly to set the marker bit No.1 of the A-group node. Then the marker propagation is executed to set the marker bits No.1 of the nodes which belong to the lower hierarchy of the A-group along the 'isa' links, starting from the A-group node going downward. These nodes represent members of the A-group. As well as the A-group members, the marker bits No.2 of the B-group members are set using association

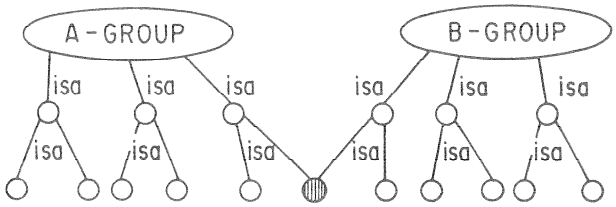


Figure 1. A Semantic Network

and marker propagation. Finally, the set operation is executed to find the intersection of the A-group and the B-group; marker bits No. 1 and No.2 of each node is ANDed.

Parallelism in marker propagation can be utilized to greatly reduce the execution time in large knowledge bases. This is because marker propagation can find all the members of a set in the length of time proportional to the depth of the tree. Therefore, if the set is represented as a binary tree of a million nodes, only 20 steps are required to mark all the members. Association includes as many concurrencies as there are nodes in the semantic network. This also applies to set operations. As will be mentioned later, the IXM machine can exploit these parallelisms fully by using associative memories.

2.2 IXL

IXL is a semantic network language for knowledge representation. The design philosophy of IXL is to solve the problems of existing semantic network languages: the description capability of relations themselves, the method of procedure addition, and the treatment of negative knowledge.

As shown in Figure 2, IXL is a command language which looks like a logic predicate. IXL can perform (1) modifications, (2) inquiries, (3) descriptions of knowledge bases. For example, an IXL command for knowledge description, 'link(is_a, penguin, bird)', is used to state that a penguin is a bird. Once this knowledge is stored and an IXL command for inquiry,'isa(penguin,X)', is issued, the answer becomes bird.

The main features of IXL are:

1) User-defined relation represented not by a link but by a network of nodes in order to define the meaning of the relation precisely and flexibly.

2) Procedural knowledge described in logic-based expressions.

3) Negative knowledge explicitly described.

A problem with the semantic network is the expression of procedural knowledge which is used to describe complicated inference rules for knowledge bases. IXL defines procedural knowledge in the form of a clause in Prolog. For example, the following procedural knowledge represents an inference rule to determine whether a restaurant is classified as high-class or not:

```
isa( X, high_class_restaurant):-
                    asst( entree, X, Y),
                    asst( main_dish, X, Z),
                    Z + Y > 100.
```

This rule classifies a restaurant, X, as a high-class restaurant if the sum total of the entree and the main dish exceeds 100 dollars. The predicates in the above clause are all IXL commands. Arithmetic and logical expressions are also allowed to appear in the body of the clause. Notice that IXL commands can be used to describe both declarative and procedural knowledge. Therefore, once the IXM machine is designed to execute IXL commands efficiently, the IXM can process both types of knowledge uniformly.

```
To connect nodes by a link:
 link(is_a, X, Y).
 link(not_isa, X, Y).
 link(instance_of, X, Y).
 link(a_kind_of, [X,Y,..], Z).
 link(source, R, X).
 link(destination, R, Y).
 link(rule, X, ((
   asst(R, X, Y):- ....
   prop(R, X, Y):- ....
   isa(X, Y):- ....
   instance(X, Y):- ....)).

To construct a relation:
 assertion(R, X, Y).
 property(R, X, Y).
```

```
To inquire about a link:
 isa(X, Y).
 instance(X, Y).
 ako(X, Y).
 source(R, X).
 destination(R, Y).

To inquire about a relation
 asst(R, X, Y).
 prop(R, X, Y).

System-supported primitive links;
 is_a,  not_isa,  instance_of
 not_instanceof,  a_kind_of,
 source,  destination

X,Y: concept node
R: relational node
```

Figure 2. The list of IXL commands

3. IXM SYSTEM ARCHITECTURE

3.1 Overall structure

IXM operates under the control of the host computer. IXM consists of the pyramid-shaped network and processing elements (PEs) as shown in Figure 3.

The pyramid-shaped network has a network processor at each node of the network. One network processor at the bottom layer of the pyramid-shaped network connects four PEs. Each upper network processor is connected to four lower network processors. In addition, a network processor is connected to four adjacent network processors of the same layer. A network processor includes message routing logic and an associative memory to exploit the concurrency in marker propagation.

The PE includes two kinds of associative memories. The first one is for the storage and the processing of the partitioned semantic networks, for a large semantic network for knowledge bases is partitioned into sub-semantic networks and distributed among the PEs. Association and set operations are executed using the function of the associative memory in parallel and equal to the number of the memory words.

The second one is to store the IXL language

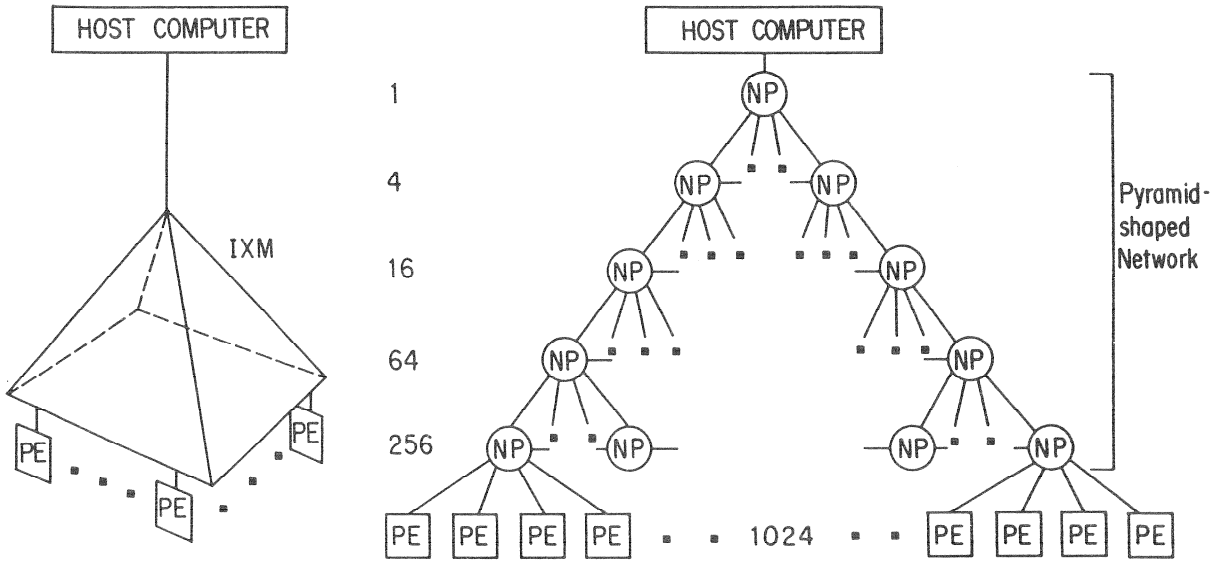


Figure 3. System organization

interpreter. The interpreter is described in the IXM machine instruction set of which the execution is asynchronous; the associative memory is used to fetch asynchronous IXM machine instructions.

A microprogrammed control unit is also included in the PE to execute IXL commands and arithmetic operations specified in procedural knowledge.

3.2 Instruction levels

There are three instruction levels in the IXM; (1) IXL commands, (2)IXM machine instructions, and (3) IXM micro instructions.

The interface between the IXM and its user is an IXL command. For example, user inputs an IXL command, 'is_a(penguin, X)' into the host computer in order to find out what a penguin is. The host computer accepts the IXL command and broadcasts it to each PE via the pyramid-shaped network, as shown in Figure 4. The IXL command is input into the IXM one at a time from the host computer; synchronization is needed each time a new IXL command is input into IXM. All the PEs execute the same IXL command in parallel because each PE stores the IXL language interpreter and the sub-semantic network in two associative memories.

The IXL language interpreter consists of subroutines; a subroutine is a non-procedual program and executes an IXL command asynchronously. Because subroutines are written in IXM machine instructions, an IXL command is executed in IXM by the IXM machine instructions.

Execution of IXM machine instructions are asynchronous to exploit the parallelism existing in IXL commands. During the execution of an IXL command, PEs require no synchronization with each other; a processing element can return the answer to the host computer as soon as the solution of the IXL instruction is found in the PE. This concurrency speeds up the IXL interpreter. Notice here the difference in execution modes; the IXM machine instructions are executed asynchronously (MIMD mode) , while the IXL commands are executed synchronously (SIMD mode).

IXM micro instructions are used mainly for the emulation of the IXM machine instructions.

4. DESIGN FEATURES

4.1 Nodes of equivalence for parallel marker propagation

Marker propagation can mark all the members of a set with O(log N) if the set is represented in a tree-shaped semantic network of N nodes. It seems to be very effective in speeding up large semantic network processing. However, it is not necessarily true when many links concentrate on one node, since such a node would have to repeat propagations as many times as the number of the links connected to it and would become a bottleneck in marker propagation. Nodes with hundreds of links appear frequently in semantic

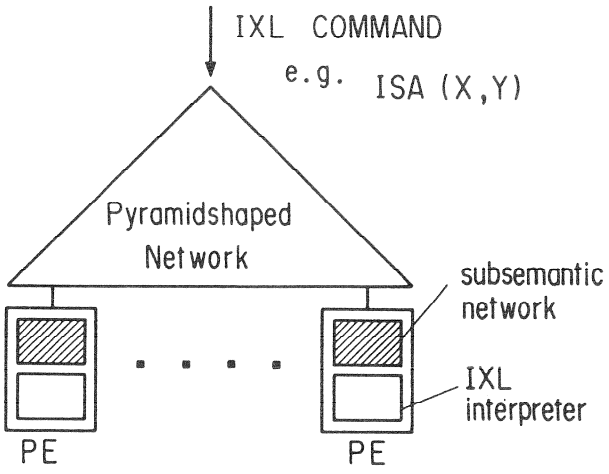


Figure 4. Execution of an IXL command

network applications because semantic networks are useful to represent taxonomical knowledge.

The authors introduced the idea of 'nodes of equivalence' in order to solve this problem and exploit concurrency in marker propagation. The idea is to partition the bottleneck node into nodes with a smaller number of connecting links and to distribute them among the PEs. A partitioned node is called a node of equivalence. For example, the student node in Figure 5(a) is the bottleneck of marker propagation, because it must repeat propagations 26 times. Therefore, the student node is divided,for example,in this case, into three nodes of equivalence as shown in Figure 5 (b). If the bottleneck node is partitioned into N nodes of equivalence, the parallelism increases N times.

If a message is sent to one of the nodes of equivalence, the message hasto be duplicated and sent to the other nodes of equivalence. For example, the hatched PEs in Figure 6 (i.e. B,C,D) contain the nodes of equivalence. If a message comes from the PE A and the destination is B, the message is duplicated and sent to C and D by the network processors of the pyramid-shaped network. The details are described in Section 6.

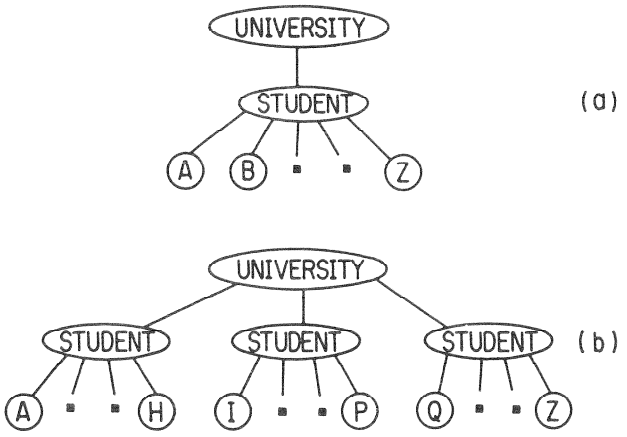


Figure 5. The nodes of equivalence

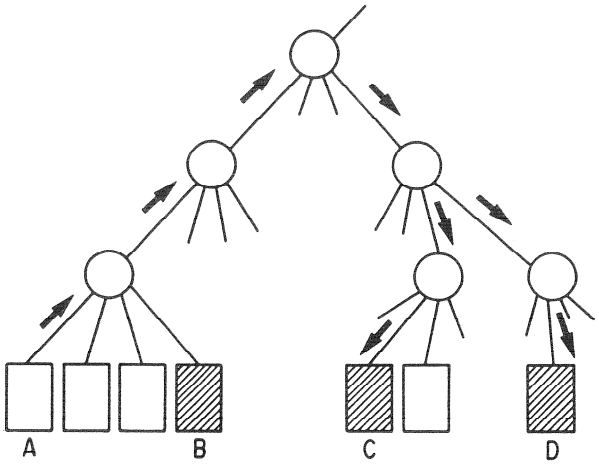


Figure 6. Message transfer using nodes of equivalence

4.2 Asynchronous IXM machine instructions

The authors have proposed an asynchronous execution mechanism of IXM machine instructions in order to utilize the parallelism in the IXL interpreter. As mentioned earlier, the IXL interpreter consists of subroutines written in IXM machine instructions; a subroutine corresponds to an IXM command. The parallelism in the IXL interpreter can reduce considerably the total execution time of the IXM.

For example, suppose the IXL command, 'is_a(X,animal)', is entered in order to find the solutions which come under the heading of animal. In Figure 7, the solutions are bird, penguin, and robin. Marker propagation marks these solutions, starting from the animal node. Therefore, the bird node is marked earlier than any other node.

So, if the bird node is returned immediately to the host computer as a solution (without waiting for the completion of marker propagation), the turn around time can be reduced to a large extent. In order to realize this, IXM machine instructions must be asynchronous.

Now we shall explain the asynchronous execution mechanism. Here it is assumed that each node has a subroutine for an IXL command to be executed; the hatched rectangular in Figure 7 represents a subroutine. The subroutine is written in asynchronous IXM machine instructions. An IXM machine instruction in the subroutine specifies a marker bit in the input marker field of the instruction format; as soon as the marker bit of a node is set, the IXM machine instruction can be started to the node, independent of the other nodes.

In Figure 7, if the marker bit No.1 of the bird node is set by marker propagation from the animal node, the bird node searches its subroutine for the IXM machine instruction which specifies the marker bit No.1 as the input marker. If there are two such IXM machine instructions in the subroutine and if they are the return and the marker propagation instructions, the bird node can immediately return its identifier to the host controller and then mark the marker bit No.1 of the penguin node. Thus,the asynchronous execution of the IXM machine instructions utilizes the parallelisms existing in the IXL interpreter.

The PE employs an associative memory to find

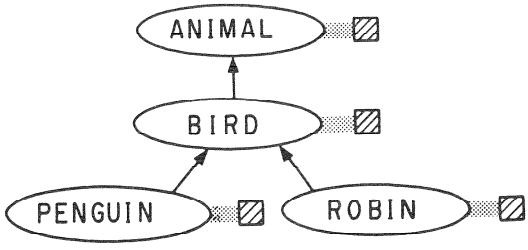


Figure 7. Taxonomical knowledge

executable IXM machine instructions quickly. The details are described in Section 5.

4.3 Associative memory to store and process semantic networks

The association operation and the set operation are very important in the IXM design because they appear in semantic network processing very frequently as well as marker propagation. The IXM employs an associative memory for each PE to utilize the parallelism in these operations.

The large semantic network of the knowledge base is partitioned into sub-semantic networks. Each sub-semantic network is stored in the associative memory of a PE. Therefore, multiple nodes are stored in a PE. For these nodes, the association operation and the set operation are executed in the SIMD manner, using the function of the associative memory.

Because the IXM consists of one thousand PEs, it is an advantage that the IXM design highly utilizes the associative memories suitable for VLSI implementation.

5. PROCESSING ELEMENT

As shown in Figure 8, the PE of the IXM consists of three components: an associative memory for semantic networks (SNAM), an associative memory for the IXL interpreter (IRAM), and a microprogrammed control unit (CU).

SNAM stores a sub-semantic network which is a partition of the large semantic network. Each word of SNAM contains not only a node but also the marker bit field where the processing results to the node are stored. Using its association function, the set operation and the association operation are executed simultaneously for all the data in the memory.

IRAM stores the IXL interpreter described in the IXM machine instructions. If a marker bit of a node in an SNAM word is set by the semantic network operation, then it causes the execution of another IXM machine instruction. Therefore, IXM machine instructions are asynchronous just like dataflow instructions[8]. IRAM finds executable IXM machine instructions by its associative function.

CU is a microprogrammed processor and it controls SNAM and IRAM to perform three functions: the execution of IXL commands, arithmetic and logical operations, and the processing of communication packets sent from either the host controller or other PE's. CU executes the IXM micro instructions.

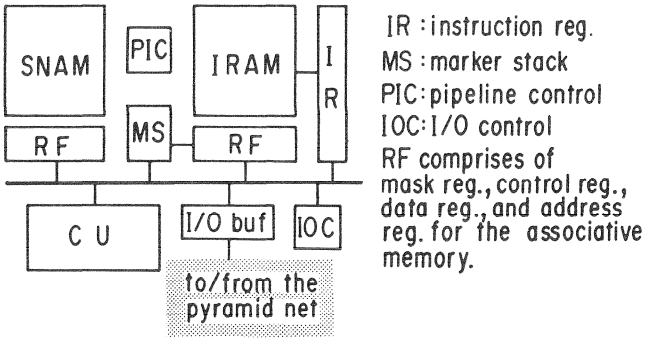


Figure 8. PE block diagram

5.1 Execution cycles of an IXM machine instruction

The microprogrammed control unit (CU) controls the fetch and the execution of an IXM machine instruction, as shown in Figure 9. Firstly, an IXM machine instruction is fetched from IRAM. Secondly, the IXM machine instruction is executed by CU, using SNAM. The marker bit of SNAM is updated as the result of the execution.

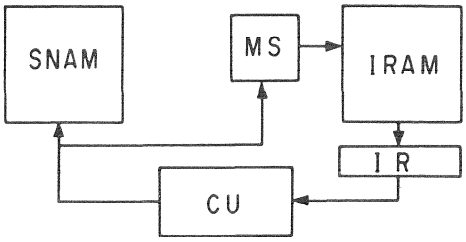


Figure 9. The execution mechanism of an IXM machine instruction

When the marker is updated, the marker bit number is also stored into the marker stack (MS). Thirdly, CU searches IRAM for the next executable IXM machine instruction, using the marker bit number in the marker stack, since the update of a marker bit enables the execution of another IXM machine instruction. These cycles are repeated to find the solutions of an IXL command.

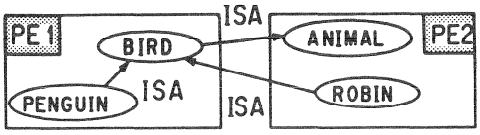
5.2 SNAM

Figure 10 shows the organization of SNAM and how the partitioned semantic networks are stored in SNAMs. The semantic network in the figure is partitioned into two sub-networks and stored separately in two SNAMs.

SNAM is 2K-word associative memory with a width of 64 bits. One word consists of four fields: the identifier field (21 bits), the destination field (21 bits), the link name field (4 bits), and the marker bit field (18 bits).

An SNAM word represents a link and the node which is connected to the link. A link occupies two SNAM words , since a link has two nodes on both edges. For example, as shown in Figure 10, the 'isa' link between the animal node and the bird node occupies the two words: the first word of SNAM1 and the first word of SNAM2. The identifier field distinguishes between the two million links stored in the IXM. The link name field indicates a system-support link such as 'isa' and 'instance_of' ; four bits are required to distinguish between the system-support links (a user-defined link is translated into system-support links). The destination field consists of the PE number and the displacement within SNAM where the node is stored; for example, the destination of the robin node at the first word of SNAM2 is represented as 'PE1(1)', because the animal node, which is the destination of the robin node, is located at the first word of SNAM1 in PE1.

The marker bit field contains 18 marker bits (numbered from 0 to 17). Each marker bit holds the result of the semantic network processing such as association and set operations. For example,



IDF(21bit)	DSF(21bit)	LN(4b)	MBF(18b)
BIRD	PE2(1) ANIMAL	ISA	
BIRD	PE1(4) PENGUIN	RISA	
BIRD	PE2(2) ROBIN	RISA	
PENGUIN	PE1(1) BIRD	ISA	
SNAM1 (PE1)			

ANIMAL	PE1(1) BIRD	RISA	
ROBIN	PE1(1) BIRD	ISA	
SNAM2(PE2)			

LEGEND IDF: identifier field LN: link name
DSF: destination field -> PE No.(displacement)
MBF: marker bit field RISA: reverse isa

Figure 10. SNAM organization and the partition of a semantic network

suppose an IXM machine instruction, 'ASSOC(bird, 2)', is executed in the IXL interpreter program, this instruction is for the association operation. It searches for the SNAM words which contain the bird node. If it exists, each marker bit No.2 of the corresponding words is set; in this example, the No.2 marker bits of the three words in SNAM1 are set.

When the marker bit is set in SNAM, the marker bit number is also stored in the marker stack. The marker stack is accessed by IRAM to find IXM machine instructions which have become executable.

Using SNAM has two advantages. The first one is the parallel processing of the association and the set operations. The IXM can process both of them in O(c) steps, while the algorithms for them on a sequential machine take O(n) and O(n*log n) steps respectively. The second advantage is that the establishment of new links is relatively easy. The two new words of SNAM are added in order to establish a new link between the two concepts.

5.3 IRAM

Figure 11 shows the organization of IRAM. Each word consists of the operation code field (3 bits), the input marker field (5 bits), the argument field (21 bits) and the resulting marker field (5 bits). The input marker field specifies a marker bit of SNAM. If an IXM machine instruction specifies a marker bit in the input marker field, the IXM machine instruction becomes executable as soon as the marker bit is set. The resulting marker field specifies the marker bit which is to be set as the result of the execution.

The IXL interpreter consists of subroutines for IXL commands and each subroutine is written with these IXM machine instructions. Figure 12 shows an IXL command, 'isa(penguin,X)', and its

op-code field (3bit)	input marker field (5bit)	argument field (21 bit)	resulting marker field (5bit)
ASSOC	*	PENGUIN	0
MARK	0	ISA	0

Figure 11. IRAM organization

subroutine written with IXM machine instructions. This IXL command searches for the higher-class concepts of the penguin. In order to find the solution, two steps are required. First, the penguin node is determined. Then, 'isa' links are traversed starting from the penguin node. The nodes on the traversed 'isa' links become the solutions of 'isa(penguin,X)'; they are the bird node and the animal node.

```
IXL command:    ISA(penguin, X)

IXM machine instruction:
                ASSOC(penguin,0)
                MARK(0, isa, 0)
```

Figure 12. An IXL command and the subroutine for it

The subroutine to implement this IXL command is stored in IRAM as shown in Figure 11. At first, 'ASSOC(penguin, 0)' is selected from IRAM. After the execution, the marker bit No.0 of the fourth word of SNAM1 in Figure 10 is set because it is the penguin node. After the marker bit No.0 is set, IRAM is searched to find the IXM machine instruction which has the marker bit No.0 in the input marker field. Then, 'MARK(0,is_a,0)' is selected.

Figure 13 explains the MARK instruction. Suppose 'MARK(n1,is_a,n2)' is to be executed. If node A has the marker bit No.n1 which is set and if an 'isa' link emerges from node A, the MARK instruction becomes executable. As the result of the MARK instruction, the marker bit No.n2 of node B which is connected to node A with the 'isa' link is set.

Therefore, if MARK(0,is_a,0) is executed to the example of Figure 6, the marker bit No.0 of the bird node is set. Then, MARK(0,is_a,0) is selected once again from IRAM and is executed. So, the marker bit No.0 of the animal node is set. Thus, the execution of the IXM machine instruction proceeds asynchronously.

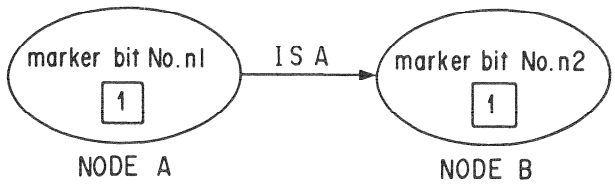


Figure 13. MARK instruction

5.4 Control unit (CU)

Processing by CU is roughly classified into the following two types. The first type is the emulation of IXL machine instructions. The IXM machine instruction such as MARK is executed in terms of IXM micro instructions. As mentioned in Section 5.1, read and write operations to SNAM and IRAM are frequently done in order to fetch and execute IXM machine instructions. These associative memories are controlled with IXM micro instructions.

The second type of CU processing is the arithmetic and logical operations which appear in the description of procedural knowledge; the inference rule mentioned in Section 2.2 is shown again as an example.

```
isa( X, high_class_restaurant):-
                    asst( entree, X, Y),
                    asst( main_dish, X, Z),
                    Z + Y > 100.
```

The first IXL command in the body, 'asst(entree,X,Y)', looks up the price of the entree,'Y'. The second looks up the price of the main dish, 'Z'. The third calculates the sum and decides whether the sum exceeds 100 dollars or not.

Although the rule looks like a clause in Prolog, notice that all the solutions of an IXL command can be determined simultaneously. Solutions of a predicate in Prolog are determined sequentially. Therefore, the IXM can be regarded as a parallel logic machine taking this viewpoint.

6. PYRAMID SHAPED NETWORK

The structure of the pyramid-shaped network (PYN) is shown in Figure 3. Network processors are connected in the shape of a pyramid. However, the connection path between the PEs of the same layer of the network is not used in the first version of the IXM for the convenience of implementation.

PYN is used for the following three types of packet communication.

(1) Host computer-to-PE: Broadcast of an IXM command.

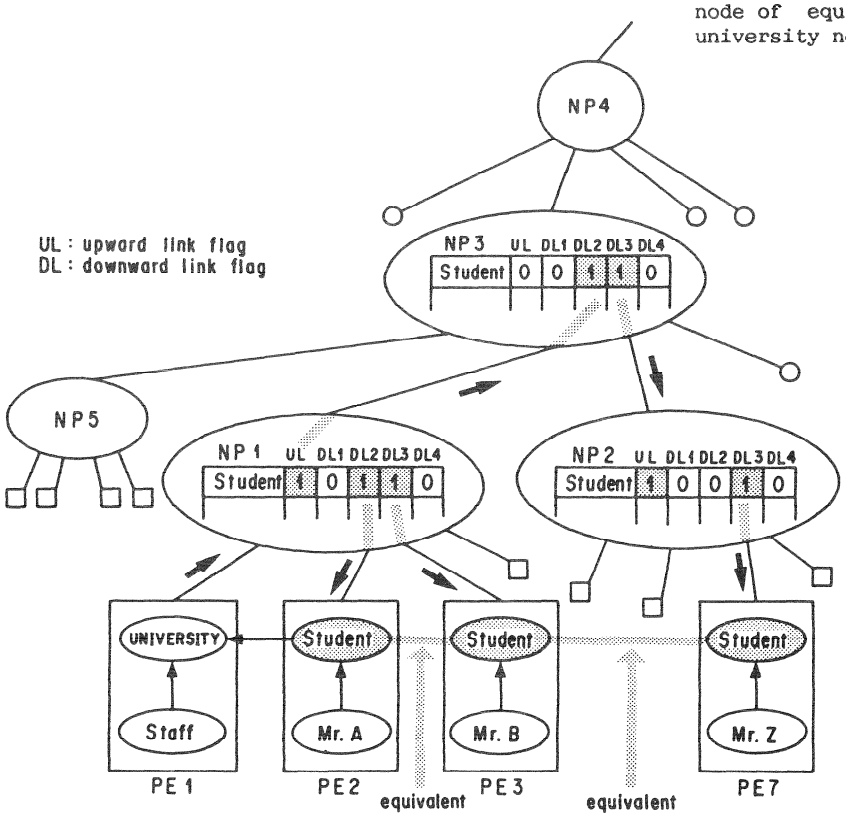
(2) PE-to-PE : Marker, value, and (or), node identifier are transferred from PE to PE.

(3) PE-to-Host computer : Collection of computation results.

The destination of the packet in the PE-to-PE communication is the node identifier to which the packet is sent. The node identifier consists of the PE number and the displacement in the SNAM where the node is stored. The network processor routes the packet according to the node identifier. The network processor always watches the node identifier to distinguish whether the node is the member of the particular nodes of equivalence or not. If the packet is sent to a member of the nodes of equivalence, the network processor copies the packet and send the copies to all the members of the nodes of equivalence. For this operation, a table for nodes of equivalence is stored in an associative memory of each network processor.

6.1 The node of equivalence

Figure 14 shows the marker propagation to the node of equivalence. It shows the case of the university node in PE No.1 sending a packet to the



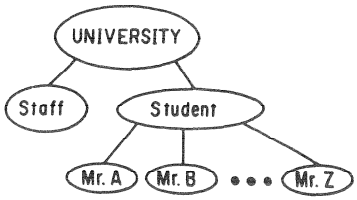


Figure 14. Marker propagation using nodes of equivalence

nodes of equivalence,'student'. The 'student' nodes are distributed in processing elements No.2, No.3, and No.7. Therefore, the packet has to be copied and the copied packets have to be sent to these PE's by the network processor No.1, No.2, and No.3.

The network processor has a table for nodes of equivalence (EQNT). Each item of EQNT consists of the node identifier of the node of equivalence and the routing information to deliver the copied packets.

The routing information is in five flags indicating the distribution of the nodes of equivalence among PEs. Four bits correspond to four downward links to four lower network processors or PE's, and the remaining one bit corresponds to one upward link to an upper network processor in the network hierarchy.

It is required to determine which network processors have the EQNT items for a particular node of euivalence. For example, network processors No.1, No.2, and No.3 have the EQNT items for 'student' node as shown in Figure 14.

The algorithm to select the network processors where the EQNTs are to be located is as follows:

(A1) Select the PEs which contain the nodes of equivalence.

(A2) Find a root network processor of the minimum sub-tree which has all the PEs selected in (A1) as its leaf.

(A3) Traverse the connection path starting from each PE of (A1) to the root network processor selected in (A2).

(A4) Select the network processors traversed in (A3).

The routing information of each EQNT item is determined as follows:

(B1) If a network processor has downward links which are connected to either the network processors of (A4) or the PEs of (A1), then the downward link flags corresponding to the downward links are set.

(B2) If a network processor has an upward link which is connected to the network processor of either (A2) or (A4), then the upward link flag is set.

For example, the downward link flags No.2 and No.3 of the 'student' item in the network processor No.1 are set as shown in Figure 14.

6.2 Structure of network processor

The network processor consists of the following components:

(1) Input/output buffer queues for asynchronous communications between network processror and a PE, or between two network processors

(2) decision logic for packet routing

(3) the associative memory to store the table for nodes of equivalence (EQNT)

(4) the control processor

Figure 15 shows the block diagram of the network processor. A packet arriving at a network processor is queued into input buffer registers. Then, the destination of the packet is examined using the associative memory in order to identify whether the destination is a node of equivalence or not. If it is identified, the packet is copied

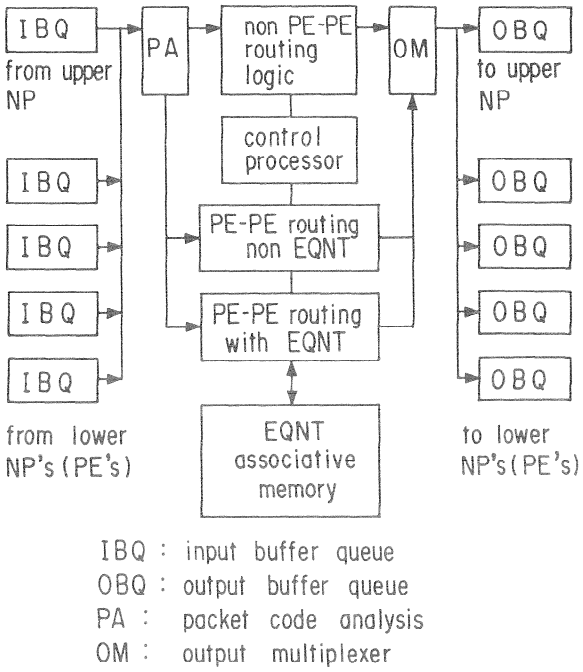


Figure 15. Network processor - block diagram

and sent to the other member of the node of euivalence according to the routing information in EQNT. If the destination is not a node of equivalence, the packet is delivered using simple routing logic.

7. PERFORMANCE

We show here the simulation results of the IXM machine and discuss its performance. At first, the execution time of semantic network processing on a conventional computer (Micro VAX II) is shown for the comparison with IXM.

We developed the IXL interpreter in K-Prolog on Micro VAX II. A French wine knowledge base has been written in IXL. A user can issue an IXL command to get various knowledge about wine. Table 1 shows the execution time of two types of queries to three knowledge bases of different size.

The query (1) asks the knowledge base whether a particular fact holds or not. For example, an IXL command, "prop(rate, 'chablis_wine', '**')",is issued to ask whether the chablis wine is rated as two-starred or not. As shown in Table 1, a user can get an answer (yes or no) in about two second, although the execution time depends on where the fact is located in the knowledge base.

The query (2) asks the knowledge base all the solutions of the query. For example, "bagof(X, prop(rate, X, '**'),L)" command replies all the wines rated as two-starred in the variable L. The execution time of this type of query increases abruptly as the knowlede base grows larger. A user have to wait fifteen minutes for the query to the knowledge base of 4500 links which is far from practical knowledge base. This is because of the exponential explosion of the search space. Even

Table 1. Execution time of the knowledge base(KB) search by IXL language processor on Micro VAX II

KB size	1600 links	3300 links	4500 links
query (1)	66 - 600 msec	66 - 1183 msec	66 - 1633 msec
query (2)	2.1 min.	8.7 min.	14.6 min.

Table 2. IXM machine cycles for the knowledge base search

KB size	1600 links	3300 links	4500 links
query (1)	1560	2112	2404
query (2)	2452	2756	3556

if the IXL interpreter is implemented not in Prolog but in Lisp, the result will show the same tendency as in Table 1.

On the other hand, as shown in Table 2, the result of IXM machine simulation shows that the total of IXM machine cycles required for query (2) does not increase abruptly even if the knowledge base grows. The potential performance of IXM is thousands faster than conventional computers. This is because IXM machine can find all the solutions in parallel using the procedures written in asynchronous IXM machine instructions.

We are currently designing the details of the IXM machine of which machine cycle is 100 nano second, assuming the associative memory of which access time is 300 nano second and 6 mega byte/sec transfer rate between network processors.

8. CONCLUSION

Knowledge bases of a few million nodes described in the semantic network will be used in practical applications within a decade. The authors have proposed a massively parallel architecture which highly utilizes associative memories to exploit the parallelism in semantic network processing.

The associative memories in IXM are used for three purposes. The first is the utilization of parallel structure in association and set operation. The second is the parallel marker propagation. The third is the exploitation of the parallelism in IXL interpreter.

The IXM machine is also a language (IXL) oriented machine. IXL gives a method to treat declarative and procedural knowledge uniformly. IXM machine can process both declarative and procedural knowledge efficiently by adopting the IXL oriented approach.

The simulation result shows that the potential performance of IXM machine is thousands faster than conventional machines. In order to utilize IXM fully for large knowledge based systems, it is also very important to develop an optimizing allocator which partitions a large semantic network and allocates each sub-semantic network onto a PE. This is because the throughput of IXM is greatly influenced according to the allocation algorithms; the communication pattern in IXM, the parallelism in marker propagation, and the conflicts on the pyramid-shaped network change considerably.

Associative memories can be implemented with a high integration density because of its regular cell structure. The authors have started the VLSI implementation of IXM[9]. Though associative memories of large capacity are not in practical use at present, the associative memory approach will lead to one of the most promising architectures for large knowledge based systems.

ACKNOWLEDGEMENT

The authors thank Dr. Hiroshi Kashiwagi for providing the opportunity to pursue this research.

REFERENCES

[1] J.R.Carbonell: "AI in CAI: An artificial intelligence approach to computer-assisted instruction", IEEE Trans. on Man-Machine Systems, MMS-11:190-202,1970.

[2] S.E.Fahlman: "Design sketch for a million NETL machine", Proc. of First Annual National Conf. on AI, 1980.

[3] K.Handa, T.Higuchi, A.Kokubu and T.Furuya: "Flexible Semantic Network for knowledage Representation", to appear in Journal of Information Processing Society of Japan.

[4] K.Handa, T.Higuchi, A.Kokubu and T.Furuya: "Semantic Memory System IX - Knowledge Representation in IXL", WGAI Rep. of IPSJ, Mar. 1985. (in Japanese)

[5] T.Higuchi,K.Handa,T.Furuya and A.Kokubu: "A Semantic Network Language Machine", Proc. of EUROMICRO, 1985.

[6] G.E.Hillis : "Connection machine", TR-646, MIT, 1981.

[7] D.I.Moldovan et al:"Semantic Network Array Processor",Univ. of Southern California, Tech. Rep. PPP-84-2,1984.

[8] J.B.Dennis : "Data Flow Schemas", Lecture note in computer science, Vol.15,1972.

[9] A.Kokubu et al: "Orthogonal Memory- A Step Toward Realization of large Capacity Associative Memory", Proc. of VLSI85, 1985.