IXM2: A PARALLEL ASSOCIATIVE PROCESSOR FOR SEMANTIC NET PROCESSING — PRELIMINARY EVALUATION —

Tetsuya Higuchi
Tatsumi Furuya
Kenichi Handa
Akio Kokubu

Reprinted from PROCEEDINGS OF THE 2ND INTERNATIONAL IEEE CONFERENCE ON TOOLS FOR ARTIFICIAL INTELLIGENCE, Herndon, VA, November 6-9, 1990

IXM2: A parallel associative processor for semantic net processing – preliminary evaluation –

Tetsuya Higuchi*, Tatsumi Furuya, Kenichi Handa, Akio Kokubu

Electrotechnical Laboratory
1-1-4, Umezono, Tsukuba, Ibaraki, Japan 305

Abstract

This paper describes an associative parallel processor, IXM2, for AI applications using network structured data (e.g. a semantic network). IXM2 consists of 64 associative processors and 9 network processors, having in total 256 Kwords by 40 bits of associative memory. IXM2 employs a new interconnection mechanism based on complete connection, realizing both high communication bandwidth and expansibility into a larger system. It is shown that associative memory is very effective in suppressing the computational explosion in large semantic network processing. Arithmetic and logical operations also can be done in parallel on all the words of associative memory; for example, achieving 7200 MOPS for the *less than* operation. IXM2 executes programs written in the IXL language, a superset of Prolog, so that IXM2 can be utilized as a back-end to AI workstations.

1 Introduction

Representing information as networks of interconnected nodes is one of the fundamental techniques in AI applications and has been widely used in areas such as knowledge based systems, natural language procesing, and human memory models [Quillian, 1967]. Frame and semantic network formalisms are used for this kind of representation.

However, it is very difficult to develop practical applications using large network structures. This is because the total amount of computation increases in an exponential fashion as the network size grows larger. In order to suppress the effect of the computational explosion, it is important to reduce the order of algorithms used in processing. The most promising solution for this is through the use of massively parallel processing.

In this paper, the authors propose a new approach to massively parallel computing based on large associative memories and describe the architecture of the associative parallel processor IXM2 which has been developed mainly for semantic network applications.

A traditional approach to massively parallel computing is to develop multiple processor systems consisting of thousands of 1-bit PEs[Hillis, 1985]. However, associative memories have the following advantages with respect to massively parallel computing;

(1) Arithmetic and logical operations are performed in SIMD manner on all the words contained in the memories. In addition, data need not be transferred to a CPU because the procesing can be done at each place the data is stored. This significantly reduces the amount of data traffic in the system, especially when large amounts of data are processed. In this usage, associative memories are free from Von-Neumann bottlenecks.

(2) Association, set operations and marker propagation, which are fundamental operations in knowledge retrieval from network structured data, are efficiently performed; in these operations, associative memories are extremely effective at reducing the order of algorithms. Therefore, the explosion of computation is suppressed in large semantic network applications which, traditionally, have been difficult to handle.

Parallelism in practical AI applications often exists at the data level rather than at the program level. Such data parallelism can not be exploited with current Lisp machines or workstations. To solve this, IXM2 is designed to work as an AI-coprocessor of workstations and to have a versatile programming interface. Functions or tasks which involve massive parallelism at the data level are distributed from the host to IXM2; IXM2 executes the functions called from a user program (Prolog or Lisp) on the host. (In this sense, the usage of IXM2 is similar to CM-2.)

IXM2 is currently utilized in two ways: (1) a macro instruction processor for arithmetic and logical operations with arrays of data, (2)a semantic network processor.

Macro instructions such as addition and less-than are executed in word-parallel manner on the data stored in the associative memories. As there are 256Kwords X 40 bits of fully parallel associative memory, there exists 256K parallelism in the execution of macro instructions.

In semantic net processing, IXM2 enables up to 64k parallel operations with the use of large associative memories. This is achieved by directly mapping the semantic network onto associative store allowing it to be processed in parallel. The programs which the IXM2 machine executes are written in the knowledge representation language IXL, a superset of Prolog. In addition to ordinary Prolog predicates, IXL includes special predicates for semantic network processing. Therefore programmers who have experience of Prolog can utilize the IXM2 very easily. The special predicates are passed from the host to the IXM2 machine and there all the solutions to a query are found in parallel. In addition to the Prolog interface, the Common Lisp interface to IXM2 is being developed at CMU where IXM2 is currently installed.

Section 2 of this paper describes the architecture of IXM2 and the details of the implementation. Section 3 describes the

*Center for Machine Translation, Carnegie Mellon University. higuchi@zen.mt.cs.cmu.edu

CH2915-7/90/0000/0667$01.00 © 1990 IEEE

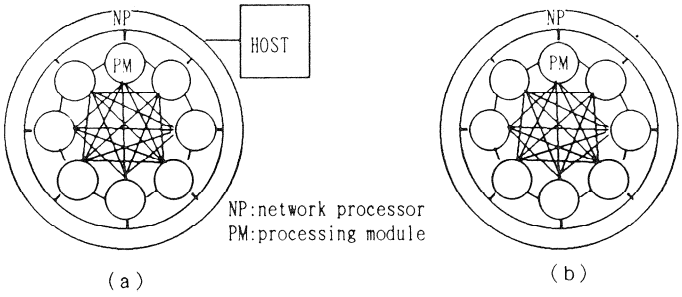

Figure 1: The general structure of IXM2

characteristics of semantic network processing and the efficient use of associative memories for this application. Section 4 discusses the programming systems for IXM2, including a brief description of a knowledge representation language IXL, which describes and handles the queries of semantic networks. Section 5 shows the initial performance of IXM2.

2 IXM2 system architecture

2.1 IXM2 architecture

IXM2 can consist of a large number of associative processors (APs). The interconnection of IXM2 is proposed for the realization of both high communication bandwidth among APs and expansibility into a system with a large number of APs.

Figure 1(a) shows the concept of the IXM2 architecture. Viewed from the host machine, IXM2 consists of Processing Modules (PM) and a network processor (NP). PMs are completely connected with each other and they are also connected with a network processor (NP) connected to the host machine.

PM is also composed of a NP and PMs in a recursive fashion as shown in Figure 1(b). Although the depth of the recursion varies by implementations, at the innermost level of the structure, PM is an AP.

Thus, because of this recursive structure, the system configuration can be expanded according to the size of the problem to be dealt with.

Figure 2 shows the structure of the IXM2 which has been constructed. 8 PMs are completely connected with each other. Each PM is composed of 8 APs connected completely. Figure 3 shows the IXM2 and the host machine, a SUN-3/260 in which an IMS B014 transputer board is installed to control the IXM2.

Although complete connection attains the best performance in various interconnection topologies, it is not usually used because of its high implementation cost.

However, complete connection among 8 PMs via high-speed serial links (20 Mbits/sec) is simple to implement. The maximum transfer rate per link is 2.4 Mbytes/sec.

In message passing systems, the delay in message transfer caused by intervening processors, between the source and the destination, often degrades the overall performance of the system considerably. Although IXM2 employs message passing, most of the communication can be performed via direct connections between the 8 APs. When communication

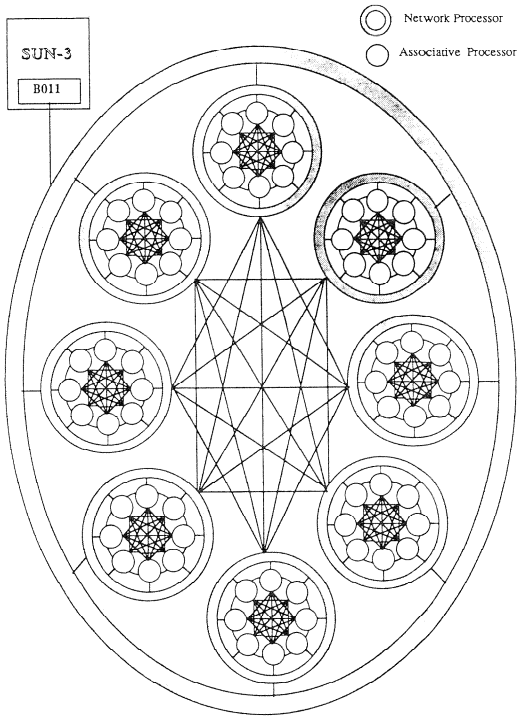

Figure 2: The structure of IXM2

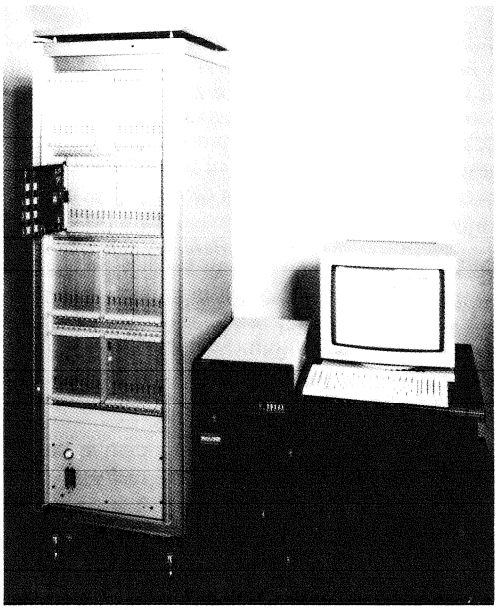

Figure 3: The IXM2 machine (left) and the host (SUN-3/260)

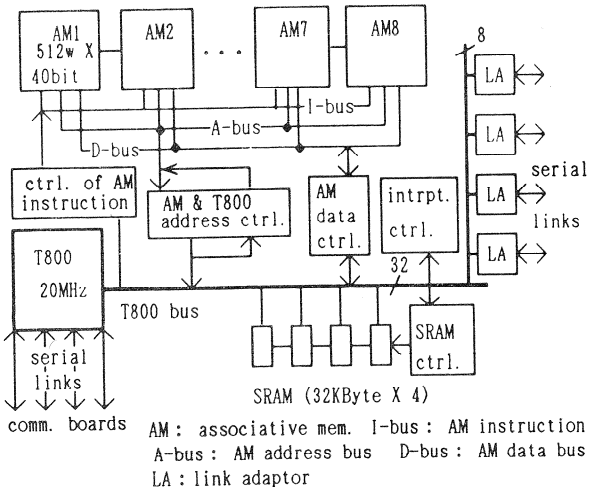

Figure 4: The blockdiagram of the associative processor

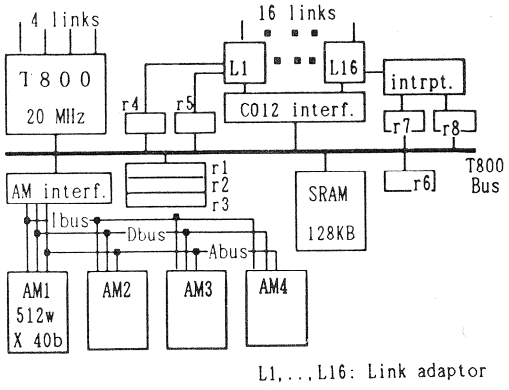

Figure 5: The blockdiagram of a network processor

between different PMs occurs, the NP relays the message from one PM to the other and may cause delays. However, with the use of an efficient semantic net allocator, the semantic network can be allocated so as to keep the communication locality within one PM as much as possible.

2.2 Associative processor (AP)

An associative processor (AP) consists of an IMS T800 transputer, associative memories, RAM, link adaptors, and associated logic as shown in Figure 4. The performance of the IMS T800 is 10 MIPS with a 20 MHz clock; it also includes 4Kbyte of on-chip SRAM with 50 nano second cycle time [Inmos, 1987]. External memory is 32 kword by 32 bit static RAM with a cycle time of 200 ns.

A T800 chip contains 4 serial links for communications, providing point to point connection between transputers. A link is bidirectional, and a link speed of 20 Mbits/second corresponds to a communication rate of 2.4 Mbytes/second in each direction.

To implement complete connections among 8 APs, it is required for an AP to have 8 serial links. Therefore, four link adaptors are attached to the bus of the transputer [Inmos, 1987]. Each link adaptor converts byte-wide parallel data into bidirectional serial link data. With these 8 links, an AP is connected to other 7 APs and to a NP.

The associative memory chip is a NTT 20 Kbit CAM (512 words X 40 bits) [Ogura, 1989]. The chip executes 26 instructions which are classified into four modes: search, read, write, and garbage collection. The combination of these instructions provides powerful functions such as relational search and long word data processing. Especially, the parallel write instruction, which writes a pattern in parallel into the all the words just selected by a search instruction, is extremely useful in reducing the order of pattern matching algorithms.

In IXM2, the associative memories store the semantic network and perform operations on stored data utilizing the marker bit field of each word as if they were registers in a microprocessor. The associative memory interface is designed for the T800 to be able to access the associative memory in the same manner as RAM; the associative memory is mapped into the transputer's 4GByte address space. Each AP board has 8 associative memory chips giving a total of 4096 words by 40 bits with an average cycle time of 375 nano seconds.

2.3 Network processor

The network processor consists of a T800, 16 link adaptors, 4 AM chips and SRAMs, as shown in figure 5 .

The network processor broadcasts the execution command from the host to APs. It also accepts the results from the APs and sends them back to the host. In addition, when a message is transferred from one PE module (PM) to the other PM, the NPs relay the message. For these purposes, an NP has the capability of broadcasting to up to 16 destinations; an upper NP, 7 NPs at the same level of IXM2 hierarchy, and the 8 APs connected to this NP.

The destinations of broadcast is controlled by setting the broadcast destination register (BDR). For example, when the execution command is sent from the host, the command is broadcast to the 8 APs connected to the NP by setting the corresponding 8 bits in the BDR.

When a message from an AP has multiple destinations, broadcasting is also used for speeding up the message transfer. In this case, the destination of the message is searched for in the look-up table in associative memory. The entry in the table corresponds to a BDR value which specifies the destinations of the message. The entry is retrieved from the associative memory and is written into the BDR for the broadcasting of the message.

3 Semantic network processing with associative memory

3.1 Associative memory

Instead of associative memories, hashing is often used to speed-up associative operations; it reduces the time for a search from O(N) to O(c). On the other hand it takes O(N) time to make the hash table and it is thus impractical in knowledge based systems to prepare hash tables for all the properties which are likely to be searched.

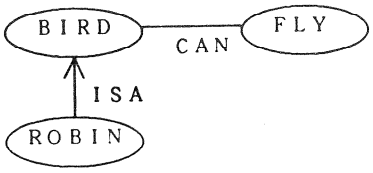

Figure 6: A semantic network

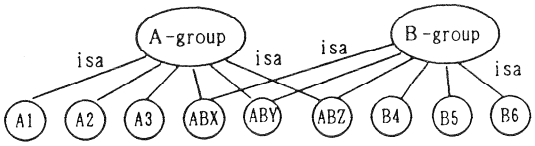

Figure 7: A semantic network representing two sets

Recent associative memories can provide powerful functions for massively parallel computing which can not be obtained by hashing. For example, a parallel write operation is extremely useful for set operations. When it is necessary to identify all the words which are selected by a search operation, a parallel write operation is done in one memory cycle to write an identifier into all the selected words.

Furthermore, progress in VLSI technology has made it easy to develop large associative memories. If the total memory size is of order, say, one hundred thousand or more, computational power comparable to supercomputers could be obtained from arithmetic algorithms devised for associative memories[Foster, 1976]. The algorithms for arithmetic and logical operations using this architecture are basically done in bit-serial manner, so that operations such as addition, multiplication, or logical operations(e.g. greater than) take longer than on conventional computers. However, because the operation is done on every word, there exist as many parallel operations as total words of associative memory. Therefore, if it takes 100 microseconds for an operation and there are 100,000 words of associative memory, the performance will be 1000 MOPS. Thus, associative memory suggests a new approach to massively parallel computing.

3.2 Characteristics of semantic network processing

A semantic network is a knowledge representation scheme which uses a *node* to represent a *concept* and a *link* to represent the *relationship* between two concepts. Using *isa* links, taxonomical knowledge can be represented more naturally than in other schemes such as predicate logic and relational models. In addition, an *isa* link also allows the lower concepts to inherit properties from super classes. The inheritance mechanisms are useful in two respects. The first is the saving of memory space; a property which holds for all the lower concepts of a particular concept A is specified just once at the description of A. Therefore, a semantic network is especially suitable for the description of large knowledge bases. The second is inference. Suppose a question is issued to the semantic network of figure 6 to ask whether a robin can fly or not. The robin node has no properties, but the inference succeeds in this case as the node is connected via a chain of *isa* links to the upper concept, the *bird* node, and the property of the bird node thus also holds for the robin node.

Most of the operations in semantic network processing are pattern matching operations. In addition to exact matching, traversing of *isa* links to find inheritable properties is included in these matching operations. For pattern matching in networks, three fundamental operations are used intensively: *association, set*, and *marker propagation*. These operations are performed efficiently by assigning *marker bits* to each node of the semantic network[Fahlman, 1980]. A marker bit is a one-bit flag in which the result of processing is stored; for example, if a marker bit is set in a node, it may show that the node is a member of a particular set found by search operations. Suppose the question, "Which member belongs to both A-group and B-group ?" is asked of the semantic network of Figure 7.

In order to find the solution, the association operation is executed first to set the marker bit 1 of the A-group node. Then marker propagation is executed to set marker bit 1 of all the nodes which belong to the 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. Similarly, marker bit 2 of the B-group members are set using association and marker propagation. Finally, the set operation is executed to find the intersection of the A-group and the B-group by ANDing marker bits 1 and 2 of each node.

3.3 Parallel processing in associative memory

This subsection describes the implementation of three fundamental operations in semantic networks utilizing associative memories.

In order to map semantic networks onto associative memories, they are decomposed linkwise. Each link of a semantic network is stored in a word of associative memory. A node with N links is represented using N words of associative memory. As shown in Figure 8(a), each word of associative memory consists of 4 fields: identifier field, destination field, link field, and marker bit field. For example, the *isa* link between the A-group node and A1 node in Figure 7 is represented in the memory as shown in Figure 8(b), because the identifier A1 is connected to the destination node, A-group, via an *isa* link.

Using this representation, the association operation is done very easily by inserting the identifier to be searched for into the corresponding position of the search register of the associative memory. Other fields of the search register are masked. For example, in order to find the A-group node, the search register is set as in Figure 8(e).

Marker propagation time is proportional to the the fanout from the node in sequential computers. For example, in order to get the set members of the A-group in figure 7, marker propagations are repeated as many times as the fanout from the A-group node (6 times). As the semantic network deals with taxonomical knowledge, there can appear nodes having a fanout of several hundred or more. This would be the bottleneck in sequential machines. However, associative memories are also very useful in reducing the overhead of marker propagation.

For example, the set members of the A-group node are found in constant time as follows. The words representing the set member nodes specify A-group as the destination field and also specify *isa* as the link field (ex. Figure 8(b)). Therefore, by setting the search register as shown in Figure 8(f), all the set members are found with only one search operation; hit flags of words representing nodes (such as A1 and A2) are set simultaneously. Then, in order to identify them, a parallel

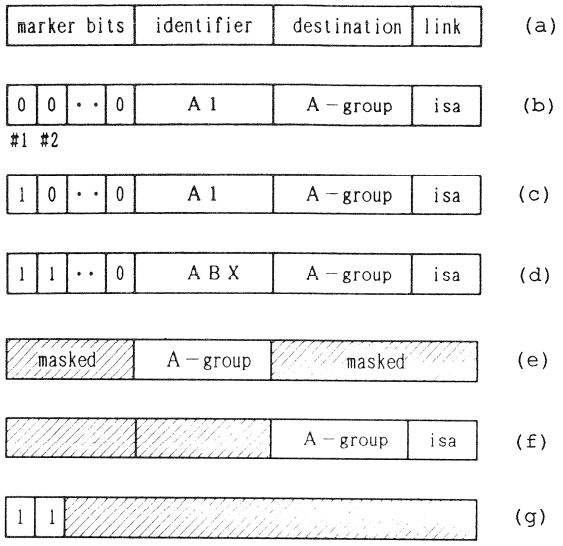

Figure 8: Representations of semantic networks in the associative memory and contents of search register

write operation is performed to set a particular marker bit in all the words just matched. If marker bit 1 is specified in the operation, the word for the A1 node becomes as shown in Figure 8(c). The parallel write operation takes only 1 memory cycle.

The set operation to get the intersection of two sets is also very efficient. In the result nodes, both marker bits 1 and 2 are set as shown in Figure 8(d). Therefore, the search operation is performed using the search register set as shown in Figure 8(g) and all the intersections are found at once.

Thus, associative memories are very efficient at performing fundamental operations required in semantic network processing. In addition, as mentioned above, arithmetic operations are done in parallel on every word of associative memory. In the IXM2, total MOPS are extremely high because there are 256K words of associative memory.

4 Programming for IXM2

Currently, IXM2 can be used in two ways: (1) as a macro instruction processor, (2) as a semantic network processor. The macro instructions defined for IXM2 are arithmetic and logical operations such as add, sub, multiplication, equal, less-than, greater-than. They are executed by bit-serial, but word-parallel algorithms. Therefore, there is as much parallelism as the number of words in the associative memory and so this suggests a new and efficient approach to number crunching operations. Currently, macro instructions can be called from C programs on the host machine and be executed in word-parallel manner on IXM2. Also, these macro instructions are to be incorporated in the semantic network processing in order to process queries which involves arithmetic and logical operations; those operations can be done at each place the relevant data is located. Therefore,it is not always necessary

```
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).
```

```
To connect nodes by a link:
  link(is_a, X, Y).
  link(not_isa. X, Y).
  link(instance_of, X, Y).
  link(not_instaceof, 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):-. . . .)).
```

Figure 9: The IXL commands

to retrieve all relevant data from large knowledge bases. Some preliminary results are given in section 5.

Semantic network processing is one of the principal applications of IXM2. The rest of this section describes this application.

4.1 Knowledge representation language IXL

IXM2 executes programs written with a knowledge representation language IXL[Handa, 1986]. IXL is a superset of Prolog. It includes special predicates for semantic network processing in addition to Prolog predicates. The additional predicates are shown in figure 9, and are called *IXL commands*.

Using these, IXL can perform (1) descriptions, (2) inquiries, and (3) modifications of knowledge bases. The IXL interpreter on the host is written in Quintus-Prolog, so that Prolog programs can be executed by the IXL interpreter.

Therefore, a programmer using the host computer who has experience of Prolog can easily perform parallel processing of semantic networks on the IXM machine. For example, suppose that the next statement is entered by a user on the host to get all the lower concepts of bird:

?- isa(X, bird), write(X), fail.

The IXL interpreter on the host interprets 'isa(X, bird)' as an IXL command and so the command is passed to the IXM2 machine which finds all the answers of the command in parallel and sends them back to the IXL interpreter on the host. The IXL interpreter binds the answers to the variable X of 'isa(X, bird)' one at a time whenever the backtracking comes to 'isa(X, bird)'.

4.2 Semantic net processing on IXM2

In IXM2, data and programs for semantic net processing are allocated as follows (Figure 10):

(1) A large semantic network to be processed by IXM2 is partitioned into semantic sub-networks and stored in the associative memory of each AP.

(2) Subroutines to execute IXL commands are stored in the local memory of each AP.

The network processors broadcast an IXL command simultaneously to all the APs. NPs also accept results from each

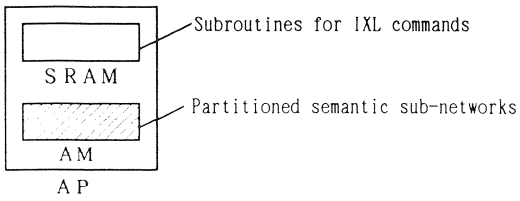

Figure 10: Data and program allocation on each AP

AP and pass them back to the host computer for the IXM2, in this case a SUN-3/260.

An IMS B014 transputer VMEbus board is installed in the SUN-3 to control the IXM2; loading occam2 programs into the IXM2; collecting answers returned from the IXM2; error handling; and so on.

4.2.1 Execution of an IXL command

When an IXL command is broadcast, each AP begins execution of the command using subroutines stored in its local memory. The majority of the operations are performed on the sub-networks stored in the associative memory of the AP. As IXL commands are entered from the host one at a time, the behaviour is SIMD at the command level. However, once each AP starts, APs operate independently in MIMD manner. During the execution of an IXL command, APs require no synchronization because of the way the IXM instruction works; an instruction in the subroutine for an IXL command (the instruction is called an IXM machine instruction) specifies a particular marker bit as the qualification for execution and the instruction only becomes executable when the specified marker bit is set. This is similar to situations in dataflow computers where the arrival of data initiates the execution of an operation. Because of this MIMD nature, answers are sent back to the host via the NPs as soon as they are found.

For example, an IXL command, isa(X,bird), is written with the following IXM machine instructions:
ASSOC(bird,1),
MARK(1, isa, 1),
REPORT(1).
The first IXM machine instruction finds the bird node and sets marker bit 1 of the node. The second indicates that if a node has a marker bit 1 set and the node has isa connections, then set marker bit 1 of the destination nodes connected via the isa links. The third instruction sends back to the host computer the names of all the nodes which have marker bit 1 set.

Suppose there is the semantic network of figure 11(a) and each node is stored in IXM2's AP as shown in figure 11(b). When the isa(X, bird) command is broadcast, then all the APs execute ASSOC(bird, 1) but only AP1 has the bird node and so only AP1 succeeds in setting marker bit 1. As the marker bit 1 is set and an *isa* link exists at the bird node, then MARK(1,isa,1) becomes executable. Execution of this instruction on AP1 sets the marker bit 1 of the robin node in AP2. This is performed by sending a message from AP1 to AP2. Then, AP2 is activated and it executes MARK(1, isa,1) to set the marker bit 1 of the robin-1 node in APn. REPORT(1) is also executed in AP2 and the identifier of the robin node is sent back to the host computer. Thus, executions go on in MIMD mode; IXM2 machine instructions are executed in marker-driven manner to exploit the concurrency in IXL command execution.

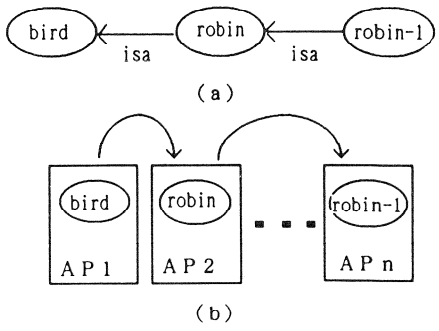

Figure 11: Execution of an IXL command on IXM2

5 Preliminary performance

This section discusses the initial performance of the IXM2, using results obtained on a single AP board. Section 5.1 discusses the execution of macro instructions. Section 5.2 shows the preliminary results of semantic network processing in two application fields; the knowledge base search and the natural language processing. The results are compared with SUN-4/330 executing Prolog and C progrrams.

5.1 Results of macro instructions

The execution time of a *less than* operation was measured on one AP board. Here a data value is compared with all the data contained in the associative memories(4096 words). It takes on average 36 micro seconds for the comparison of 32 bit data. The execution speed looks very slow, when it is compared with the execution time on sequential computers; for example, it takes 1.25 micro seconds in an occam2 program run on a T800 transputer at 20 MHz. However, the operation using the associative memory can be executed in parallel on 256 kwords in the IXM2. This means that the *less than* operation can be repeated 256K times in 36 micro seconds, corresponding to an execution time per word of 0.14 nano seconds. This performance is equivalent to 7200 MOPS which would not be attainable by any sequential computer. Even in a single AP board, the execution time per word corresponds to 8.8 nano seconds.

The execution time for addition was also measured on one AP board. This takes 46 micro seconds for 8 bit data and 115 micro seconds for 16 bit data. The execution time per word for the latter corresponds to 28 nano seconds on a single AP and 0.44 nano seconds on IXM2 (64 APs) ; it takes 2.1 micro seconds for an addition in an occam2 program on a T800 transputer at 20 MHz.

As shown above, bit-serial algorithms for associative memories have longer execution times than sequential computers. However, if the total size of the associative memory is of the order of thousands or more, the total performace is improved considerably. Thus, large associative memories suggest a new way of performing massively parallel computation.

5.2 Initial results of semantic net processing on an AP

IXM2 can handle up to 64K semantic network links in parallel allowing set and associative operations to be performed in O(c) processing time for any size of semantic network as long as data is stored in associative memory. Therefore, the performance of IXM2 must be discussed in the situations where a large semantic network is processed and computational explosion might occur on a sequential machine. However, the preliminary results of processing a small semantic network using a single AP board were promising in both of two experiments.

The first benchmark processes a query to a semantic network of 2700 links, which represents a knowledge base of wine as an isa hierarchy of 5 layers. It marks members of two sets and then gets the intersection of the two sets. The benchmark for one AP of IXM2 is written with IXM machine instructions. For SUN-4, the benchmark is written in C and is optimized with -O4.

It takes 0.79 mili seconds to process the benchmark on a single AP, compared with 3.03 mili seconds on the SUN-4/330. Also, the performance when Prolog is used for semantic net processing, has been examined for reference because IXM2 is intended to operates as a back-end to a Prolog processor. The same query was given to the knowledge base represented in Prolog and was processed by an IXL interpreter implemented in Quintus Prolog running on a SUN-4/280s. This takes 49050 mili seconds, excluding I/O. The execution time is very long, compared with the cases mentioned above, but it should be noted that the IXL interpreter involves overheads for checking negative knowledge and inconsistent knowledge, whereas those checks are omitted in the cases of the AP and the C program. These results show that IXM2 will be very useful as a processor for Prolog programs handling large data bases.

The second benchmark is the memory-based parsing used in the natural language processing for spoken inputs; syntactic recognition is carried out for an input sentence, using syntactic structures pre-expanded in terms of semantic network[Kitano, 1990]. For a sentence of length 8, it takes 1.3 mili seconds on a single AP, whereas the C program on SUN-4/330 takes 21.6 mili seconds.

According to these results, when applied to large semantic networks, IXM2 with 64 APs is expected to attain performance comparable to main-frame or supercomputers.

6 Conclusion

There are many AI applications using network data structures which can be represented in terms of semantic networks. However, large semantic networks have not been put into practical use due to the computational requirements. In order to solve this, the authors have proposed a new approach based on large associative memories. The associative processor IXM2 employs a new interconnection network based on complete connections which realize both high communication bandwidth and expansibility into a larger vsystem.

It has been proved that arithmetic and logical operations with large associative memories can attain high performance. These algorithms can be used for query processing of large knowledge bases. Because in many cases the processed data need not be transferred from associative memory, the total traffic is reduced. This characterestic would be very effective especially in multi-processor configurations.

Additionally, IXM2 has a versatile programming interface because the IXL language is a superset of Prolog and it can exploit parallelism inherent in Prolog programs which deal with network data structures. The result of initial experiments performed with the hardware shows that IXM2 can be a powerful back-end to a Prolog processor.

In order to utilize IXM2 fully for large knowledge based systems, it is very important to develop an optimizing allocator which partitions a large semantic network and allocates each semantic sub-network onto an AP, considering the structure of the network.

This is because the throughput of the IXM2 is greatly influenced by the allocation algorithms; the communication pattern in IXM2, the parallelism in marker propagation, and conflicts in the data transfer on the interconnection network vary considerably. Therefore, research on optimal allocation algorithms has to be done, using different types of semantic network in various AI applications.

In addition, studies of the representation of semantic networks should be continued for more efficient use of associative memories.

Acknowledgment

Authors thank Dr. Hiroshi Kashiwagi(ETL) for providing the opportunity to pursue this research, Dr. Akio Tojo (ETL) for assisting this research, Dr. Takesi Ogura (NTT) for providing associative memory chips, Prof. Jaime Carbonell and Prof. Masaru Tomita (CMU) for various supports for our research activities.

References

[Quillian, 1967] M. Ross Quillian," Word concepts: a theory and simulation of some basic semantic capabilities", *Behavioral Science*, 12, 1967, pp.410-430.

[Fahlman, 1980] S.E. Fahlman, "Design sketch for a million NETL machine", *Proceedings of First Annual National Conference on Artificial Intelligence*, 1980.

[Handa, 1986] K. Handa, T. Higuchi, A. Kokubu and T. Furuya, "Flexible Semantic Network for knowledge Representation", *Journal of Information Japan*, Vol.10, No.1, 1986.

[Higuchi, 1986] T. Higuchi, T. Furuya, H. Kusumoto, K. Handa and A. Kokubu, "The IX supercomputer for knowledge based systems", *Fall Joint Computer Conference*, November 1986.

[Higuchi, 1989] T. Higuchi, T. Furuya, K. Handa and A. Kokubu, "The prototype of a semantic network machine IXM", *International Conference on Parallel Processing*, August 1989.

[Ogura, 1989] T. Ogura, J. Yamada, S. Yamada and M. Tanno, "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines ", *IEEE Journal of Solid-State Circuits*, Vol.24, No.4, August 1989.

[Kitano, 1990] H. Kitano, T. Higuchi, M. Tomita, "Massively Parallel Spoken Language Processing using a Parallel Associative Processor IXM2", *International Coneference on Spoken Language Processing* , November 1990.

[Foster, 1976] C.C. Foster, *Content Addressable Parallel Processors*, Van Nostrand Reinhold Company, New York, 1976.

[Hillis, 1985] D. Hillis, *Connection Machine*, MIT Press, 1985.

[Inmos, 1987] Inmos, *IMS T800 Transputer*, April 1987.

[Inmos, 1988] Inmos, *Occam2 reference manual*, Prentice Hall, 1988.

[Inmos, 1987] Inmos, *IMS C012 link adaptor*, August 1987.