The Design of a Marker Passing Architecture for Knowledge Processing

Wing Lee and Dan Moldovan

Department of Electrical Engineering - Systems
University of Southern California 90089-1115
Los Angeles, California
wlee@gringo.usc.edu, moldovan@gringo.usc.edu

Abstract

Knowledge processing is very demanding on computer architectures. Knowledge processing generates subcomputation paths at an exponential rate. It is memory intensive and has high communication requirements. Marker passing architectures are good candidates to solve knowledge processing problems. In this paper, we justify the design decisions made for the Semantic Network Array Processor (SNAP). Important aspects of SNAP are: the instruction set, markers, relations, propagation rules, interconnection network, and granularity. These features are compared to those in NETL and the Connection Machine.

1 Basic Operations in Knowledge Processing

The computations that are typical of knowledge processing require the generation of numerous computation paths that all could potentially be followed in parallel. The process of spawning a number of relatively independent subcomputations, each of which may spawn other subcomputations, is called *bifurcation*. Bifurcation processes appear to be important for a wide range of knowledge based systems. On a serial computer, the bifurcation of independent subprocesses leads to large computational demands. Even a parallel computer does not have the hardware resources to examine all of the parallel paths of a problem. The problem with using current parallel computers is that the problems bifurcate into multiple computation paths that share a considerable amount of context.

Another basic operation in knowledge processing is *inheritance*. Inheritance is the mechanism which locates properties attached to concepts within a certain "distance". Often, the inheritance is not obvious, or deals with conflicting properties.

Recognition is also important in knowledge processing. It deals with the ability to recognize concepts or situations in the knowledge base. Although similar to ordinary pattern matching, this problem is far more complex. For example, the properties may not be available locally and may have to be extracted via inheritance, or the exact pattern may not exist and the best match must be determined.

Classification is the process of placing a concept in the knowledge hierarchy. Once placed, it is very easy to retrieve information about the concept. However, classifying a concept is a non-trivial task. It involves comparing the properties of the new concept with properties of all the concepts in the knowledge base.

Unification is the process of generalizing two patterns to form a new pattern that match what both input patterns would have matched. For example, unification may be used to match an inference rule with a knowledge base to determine the applicability of that inference rule.

In *probabilistic reasoning*, probabilities are attached to concepts in the semantic network. The probabilites are modified by interactions with other nodes. Thresholding is used to filter out hypotheses

[0]This research has been funded by the National Science Foundation Grant No. MIP-89/02426

Problem	Operation
Bifurcation	Control the bifurcation process
Inheritance	Find all nodes connected to a node via some combination of relations
	Find the paths connecting two nodes
	Find the implicit properties of a concept
Recognition	Locate entities in the knowledge base that best match a set of features
	Find the pattern that best matches a given pattern
Classification	Place a pattern at the most appropriate place in the knowledge base
Unification	Bind concepts with compatible concepts in the knowledge base
Probablistic Reasoning	Associate numerical values to parts of the knowledge base
	Compute the strength of hypotheses and compare measures
Learning	Change the values of nodes or links according to some learning algorithm

Table 1: Important Problems and Operations in Knowledge Processing

that have low probabilities.

Learning is the ability of a system to adapt to a problem domain. Commonly, learning involves assigning weights to concepts or links, and being able to change the weights to match the characteristics of the problem domain.

Table 1 lists some of the important problems and operations in knowledge processing. Some other important aspectgs of knowledge representation and reasoning are described in [Brachman, 1988].

2 SNAP Design

2.1 Marker-Passing Architectures

Marker passing architectures provide efficient implementation of the operations indentified previously [Hendler, 1988],[Moldovan, 1989]. However, the class of marker passing architectures has been relatively unexplored. This is due primarly because knowledge processing operations and algorithms have not really been identifed.

Our design approach was to build a machine that would achieve efficient performance for the operations we described earlier and for a set of Natural Language parsing algorithms. In the rest of this paper, we will be describing some of the features of SNAP. In order to better understand these features, we will be comparing them with the features of two other architectures for knowledge processing: NETL and the Connection Machine.

SNAP is a parallel machine consisting of a central controller and a 16K processing nodes. A SNAP node is capable of storing a single fact, concept, rule pattern, etc. The nodes in the network connect to other nodes in the network by a way of relations. Each relation type denotes a different relationship between concepts (nodes). The primary means of computation in SNAP is the processing of markers.

NETL [Fahlman 1979] was one of the first architectures for knowledge processing. It consisted of a central controller and a collection of very simple processing nodes. NETL had 8 different node types and 8 link types for connecting nodes together. A physical wire served as the connection between two nodes. The computation model in NETL was marker-passing, with the controller playing an active role in the movement of markers. Although NETL was never built, it served as the basis for several architectures, including the Connection Machine and SNAP.

The **Connection Machine** [Hillis, 1985] was originally developed as an implementation of Fahlman's NETL. The Connection Machine is a fine-grained array processor with programmable connections between nodes. It consists of 64K single-bit processors, with each processor having 4K bits of memory and a serial ALU. The processors operate in SIMD fashion, with messages being the method of communication.

2.2 Relations

The knowledge base in SNAP is built upon relations between nodes. SNAP can support 64 user-defined relations. The user can extend the number of relations by using nodes to act as relations. This construct is called a Relation-node. Relation-nodes are not as efficent as the primitive relation, but they effectively enable the user to have as many relations as needed. Relations in SNAP also have a weight associated with them. The weight can represent the strength of the link, the cost of traversing the link, etc. Associating weights with relations is essential for implementing reasoning mechanisms such as probabilistic reasoning.

In NETL, relations are pre-defined. Thus, the knowledge base has to be defined using the 8 link types (VC, EQ, CANCEL, CANVC, SPLIT, EXFOR, EXIN, SCOPE). This puts a severe constraint on the type of knowledge that can be represented. NETL has no mechanism for extending the number of relations beyond these 8. In addition, NETL relations cannot carry weights. In fact, NETL has no numeric capabilities at all.

The relations in the Connection Machine are more general than those in both NETL and SNAP. Like SNAP, the relations are all user-definable. Unlike SNAP, the Connection Machine has no limit in the number of relations. However, the total amount of memory available to the node for storage is limited to 4K bits. In SNAP, we felt that 64 relations was more than adequate for most applications, and that it was not worth the hardware resources to extend beyond 64. We provided the Relation-node construct to support those cases where more than 64 relations are needed.

2.3 Markers and Value Passing

SNAP is a marker-passing architecture. The nodes in SNAP communicate by way of messages. The effect of a message at a destination node is to manipulate a marker and possibly generate more messages. Each SNAP node is capable of simultaneously storing up to 24 markers. Each marker consists of a bit indicating whether the node possesses that marker, a value register and a pointer register. The value register can store either data, the current strength of the marker, a probability, etc. A marker also contains a pointer value. This pointer value identifies the node that originated the marker. The marker pointer allows the same marker to be used for different hypothesis. The pointer "colors" the marker so that we can identify which hypothesis it refers to. The marker pointer also enables the easy creation of new relations between nodes. This can be used to solidify a hypothesis, or be the end result of a series of computations (classification of a concept is one example where the creation/deletion of links is the end result).

We created markers of this type to support probabilistic reasoning. Probabilistic reasoning requires the passing of not only markers but also values. The values contain the probabilities and cost associated with the network. They must be included in the marker messages that are sent between nodes. Otherwise, the system has no way to modify the probabilities in the network. Probabilistic reasoning has important applications in the areas of speech recognition and translation and natural language understanding.

The markers in SNAP differ from those in NETL and the Connection Machine. NETL markers consist of only a single bit. No pointer or value is associated with the marker. Consequently, NETL cannot distinguish between two or more hypotheses using the same marker nor can it support probabilistic reasoning. The Connection Machine is able to associate a pointer and value with a marker. However, this is a software construct and is not nearly as efficent in utilizing these features as the built-in hardware in SNAP.

2.4 Propagation Rules

SNAP nodes are capable of communicating with other nodes by way of messages. Each message type has a built-in "propagation rule" which determines the path messages take (i.e. on which relation links to place the messages on). When a destination node receives a message, it sets a marker, performs a corresponding action, and, depending on the propagation rule, can "propagate" the message to other nodes. Thus, propagation rules permit the transfer and bifurcation of messages to occur without intervention from the central controller. This allows many different message types, with different propagation rules, to travel in the network simultaneously.

In the SNAP design we felt that it was important to implement in hardware some key propaga-

tion rules. The five propagation rules listed below give the programmer considerable freedom to direct how markers are to be propagated.

1. SEQ(R1, R2): the SEQUENCE propagation rule allows the marker to propagate through R1 once, then to R2 once.
2. SPREAD(R1, R2): the SPREAD propagation rule allows the marker to traverse through a chain of R1 links. For each cell in the R1 path, if there exist any R2, the marker switches to R2 link and continues to propagate until the end of the R2 link.
3. COMB(R1, R2): the COMBine propagation rule allows the marker to propagate to all R1 R2 links without limitation.
4. END-SPREAD(R1, R2): This propagation rule is the same as SPREAD except that it marks only the last cells in the paths.
5. END-COMB(R1, R2): This propagation rule is the same as COMB except that it marks only the last cells in the paths.

By comparison, the propagation rules in NETL are primitive. Markers can propagate on only one relation type making a propagation rule like Combination difficult to achieve. In addition, the NETL controller plays an active role in monitoring and controlling the propagation of markers. Consequently, only one type of marker can be propagating in the network.

In the Connection Machine, the propagation rules can be quite flexible, which allows the Connection Machine to easily implement the SNAP propagation rules. However, the propagation rules in the Connection Machine are software mechanisms. This creates several drawbacks. First, marker propagation in the Connection Machine occurs much more slowly than in SNAP. In SNAP, the marker propagation and processing is built into the hardware. Second, since the Connection Machine must execute instructions to process markers, marker propagation must occur in the foreground. Thus, the Connection Machine cannot perform any other function during this time. In the majority of cases, only a small portion of the network participates in marker propagation, the rest of the network is idle. In SNAP, however, marker propagation occurs in the background. Consequently, the SNAP nodes that are not busy processing markers are free to do other things. Finally, because marker processing in the Connection Machine is software based, the Connection Machine can propagate only one type of marker at a time. A different type of marker would require a different set of software instructions, which would not be able to execute at the same time as the first marker instructions. In order to solve most problems, several different markers are typically required. SNAP, therefore can acheive results with less effort and time than the Connection Machine.

2.5 Instruction Set

We have designed for SNAP a set of 21 powerful instructions specific to knowledge processing for SNAP. These instructions are executed by the processing nodes and are divided into 6 groups: Node Maintenance, Search, Logical, Marker, Marker-Auxiliary, and Data Retrival. We felt that the instructions in these 6 groups represent the core functions required for knowledge processing. The Node Maintenance instructions (CREATE, DELETE, SET-COLOR) are used for loading and modifying the knowledge base. The Search instructions (SEARCH and SEARCH-COLOR) are used to select a node or a group of nodes in the array. The Logical functions (AND, OR, NOT) are used to manipulate the markers within a node. The Marker instructions (MARKER, MARKER-ADD, MARKER-SUB, MARKER-MULT, and MARKER-DIVIDE) introduce a marker into the network. Each of the Marker instructions has a propagation rule associated with it to provide decentralized control. In addition, an arithmetic function can be associated with a marker to enable manipulation of the numeric values in the marker value and the relation weight registers. This enables SNAP to support a wide range of numeric applications, including probabilistic reasoning and learning. The Marker-Auxiliary functions (CLEAR-MARKER, STOP-MARKER, CLEAR-STOP-MARKER, EQUATE, and CLEAR-EQUATE) are used to modify the operation of the Marker instructions. The STOP-MARKER enables a node to "eat" a marker and prevent it from propagating. This is an important mechanism for controlling the flow of markers and preventing the movement of markers into undesired areas. The EQUATE instruction enables a relation to be treated as if it were another relation type during marker propagation. The CLEAR instructions are used to reset the marker portions of nodes. Fi-

nally, the Data Retrieval (COLLECT, COLLECT-RELATION, COLLECT-MARKER) are used to obtain information from the nodes. For a more detailed look at the SNAP instruction set see [Moldovan 1989].

In NETL, there is no real instruction set per se. The nodes in NETL are very simple. Control of markers is governed by the NETL controller. Reasoning on NETL is done by retrieving patterns from the knowledge base.

The nodes in the Connection Machine have a basic instruction set. They can be combined to form higher level instructions like those in SNAP. For example, [Chung 1989] created an instruction set similar to SNAP's when he programmed some knowledge processing examples on the Connection Machine.

Our approach was to spend more hardware to create more complex primitive instructions. This allows us to perform basic knowledge processing operations in a minimal amount of time.

2.6 Granularity of a SNAP Chip

In SNAP, we have packaged 32 nodes into a single custom-designed chip. Each node can have an average of 10 relations. We placed these two limitations on SNAP to reduce the cost and to save space. The advances in VLSI techonology in the last decade have enabled us to place more logic onto a chip. Consequently, we felt it would be much easier to build a machine with 512 chips than it would be to build one with 16K chips. With the amount of logic that each node takes, we have estimated that 32 nodes and 320 total relations would easily fit onto a chip.

There is a tradeoff in terms of placing more nodes onto a chip. Each SNAP chip has only 4 data ports for communication with other chip. With 32 nodes in a chip, the data ports can become a bottleneck in the system. [Kim, 1989] has done some preliminary analysis into this area.

In both NETL and the Connection Machine, each chip consisted of only one node.

2.7 Interconnection Network

The SNAP interconnection network is used to connect the SNAP chips together. It is used to enable message passing between nodes. The SNAP interconnection network is a modified bus hypercube. A 16K network is made up of 512 SNAP chips. Each chip has 4 data ports for communication with other chips and a router for determining the message path. Messages in SNAP are 50 bits long and are sent as five 10-bit packets. The maximum number of intermediate chips a SNAP message must pass through is 3. More details on the SNAP interconnection network can be found in [Moldovan 1989]. The network has been software simulated and compared with other networks [Lee 1989]. The results show that the network performs favorably when compared to the performance of other networks.

The interconnection network in NETL is vastly different. In NETL, all connections between nodes are point to point. A physical wire is placed between two nodes that share a relation. Thus, NETL messages can travel very fast. However, a network of this type is unfeasible. In a dense network (more than a hundred nodes) it is almost impossible to place a physical wire between two nodes and be able to remove it later. A network where we cannot delete links can only support monotonic reasoning; in almost all cases, this is unacceptable.

The Connection Machine has special routing chips for sending messages. This frees the nodes from having to participate in message routing. In a 64K Connection Machine, there are 4K routers, with each router servicing 16 nodes. The routers are arranged in a 12-dimensional hypercube and can process 1-bit at a time. In SNAP, we chose not to create a special router chip, because we felt that incorporating the routing function into the node chip was a better alternative. Unlike the Connection Machine, marker propagation in SNAP can proceed in the background without intervention from the controller. Thus an integrated router fits right into that concept. In the Connection Machine, however, actual instructions have to be executed to process a message. Incorporating the routing function in the node would further complicate the long process of marker propagation.

3 Simulation Results

We have built a simulator of SNAP to test some of the concepts we have discussed in this paper. We have run several examples on the SNAP simulator. Table 2 summarizes the simulation results for these examples. Example 1 is a non-obvious inheritance problem involving 30 concepts (nodes). Example

	SNAP		CM	Tcm/
	Cycle	Time	Time	Tsnap
Example1 Inheritance	387	59.8us	202ms	3380
Example2 Inheritance	2888	446us	519ms	1163
Example3 Recognition	329	50.8us	379ms	7460
Example4 Classification	366	56.5us	NA	

Table 2: Comparsion between SNAP and Connection Machine

2 is a inheritance problem over an imaginary two-dimensional 10 × 10 network. Example 3 deals with recognition with multiple properties. Example 4 is a small classification problem [Lipkis, 1983]. A complete description of the examples can be found in [Moldovan 1990].

Some of these examples have been implemented on the Connection Machine at USC-ISI. Those times are listed along with the SNAP times in Table 2. Note, for SNAP we considered the speed to be the same clock as the Connection Machine (6.47443 MHz).

4 Conclusions

SNAP combines several features which collectively make SNAP a powerful knowledge processing engine. Some of these features are: a powerful instruction set implemented in hardware, marker passing architecture, associative array processing, and a modified hypercube interconnection network. The instruction set has been carefully designed to provide hardware implementation of the most often used knowledge processing operations. Special attention has been given to marker propagation rules.

The results shown in Table 2 between SNAP and the Connection Machine are not suprising considering the differences between the two machines described in this paper. The fundamental reason for the superior performance of SNAP is that it implements in hardware features that require software instructions in the Connection Machine.

References

Brachman, R. [1988]. "The basics of knowledge representation and reasoning", *AT&T Technical Journal*, 67:1, 7-24.

Chung, S., Moldovan, D. and Tung, Y. [1989]. "Reasoning on the Connection Machine", Technical Report CENG-89-13. University of Southern California Department of EE Systems.

Fahlman, S. [1979]. "NETL: A system for representing and using real-world knowledge". The MIT Press, Cambridge, MA.

Hendler, J. [1988] "Integrating Marker-Passing and Problem-Solving". Lawerence Erlbaum Associates, Inc.

Hillis, W. [1985] "The Connection Machine". The MIT Press, Cambridge, MA.

Kim, J. and Moldovan, D. [1989]. "Parallel Classificaton for Knowledge Representation on SNAP". *Proceedings of the 1990 International Conference on Parallel Processing.* Department of Electrical Engineering Systems, Univ. of Southern California.

Lee, W. "Bandwidth Analysis of Message Passing Networks". Technical Report CENG 89-24, Department of Electrical Engineering-Systems. University of Southern California.

Lipkis, T. and Schmolze, J. [1983]. "Classification in the KL-ONE knowledge representation system", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Vol. 1, 330-332.

Moldovan, D., Lee, W., and Lin, C. [1989]. "SNAP: A Marker-Propagation Architecture for Knowledge Processing", Technical Report No: 89-10. Department of Electrical Engineering Systems, Univ. of Southern California.

Moldovan, D., Lee, W., Lin, C., and Chung, S. [1990]. "Parallel Knowledge Processing on SNAP". *Proceedings of the 1990 International Conference on Parallel Processing.*