In Proceedings of the *IASTED International Symposium on Expert Systems and Neural Networks*, pp. 12-15, 1989.

DNA: A New ASOCS Model With Improved Implementation Potential

George L. Rudolph and Tony R. Martinez
Department of Computer Science
Brigham Young University
Provo, Utah 84602

Abstract. A new class of high-speed, self-adaptive, massively parallel computing models called ASOCS (Adaptive Self-Organizing Concurrent Systems) has been proposed. Current analysis suggests that there may be problems implementing ASOCS models in VLSI using the hierarchical network structures originally proposed. The problems are not inherent in the models, but rather in the technology used to implement them. This has led to the development of a new ASOCS model called DNA (Discriminant-Node ASOCS) that does not depend on a hierarchical node structure for success. Three areas of the DNA model are briefly discussed in this paper: DNA's flexible nodes, how DNA overcomes problems other models have allocating unused nodes, and how DNA operates during processing and learning.

Introduction. A new class of high-speed, massively parallel, adaptive computational models called ASOCS (Adaptive Self-Organizing Concurrent System) has been proposed [3]. These present neural networks with fast, well-bounded learning algorithms using simple, asynchronous digital nodes [4] - [7]. One area of current ASOCS research focusses on building VLSI implementations of ASOCS using current technology. Chang and Vidal describe one possible implementation of an ASOCS model in [1]. More recent analysis of ASOCS shows that current ASOCS models may make inefficient use of limited network resources. This is not due to a problem inherent in the models, but in VLSI implementation. This paper introduces a new ASOCS model, called DNA (Discriminant-Node ASOCS), that is designed to overcome these inefficiencies. Some basic ASOCS terms are defined, and ideas briefly discussed. An example of how ASOCS models learn is presented, leading into a discussion of the problems of inefficient use of nodes in a hierarchical implementation of a network. The section following that discusses how DNA overcomes these problems. The paper concludes with a brief summary of DNA's advantages and limitations, possible applications for DNA and finally, the direction of current research.

Definitions and Ideas. An ASOCS network can be viewed as a function machine that can compute a mapping of arbitrary Boolean inputs to arbitrary Boolean outputs. A network is composed of computing nodes that operate asynchronously and in parallel. ASOCS networks have two modes of operation--processing and learning. During processing, the network acts like a parallel hardware circuit. During learning, nodes may change the functions they compute, or be deleted from the network altogether, as new information is presented. The network adapts itself so that the overall function it computes is consistent with the information it has received. The network learns **incrementally**, using rules that express a particular functional mapping. A given mapping or rule is known as an **instance**. The following are examples of instances:

AB'C -> Z   ACD -> Z'   ED' -> Y'

The variables on the left side of the arrow represent the inputs, and those on the right side the desired outputs for those inputs. Instances may specify high or low outputs for one or more output variables. An instance has **positive polarity** for a given output variable if it specifies that variable to be high, and **negative polarity** for that output variable if the variable is specified to be low. Nodes that specify opposite polarities for the same output variable are called **discordant** instances. In the examples above, AB' -> Z is positive for Z, while ACD -> Z' is negative for Z. These instances are also discordant instances. ED' -> Y is positive for X. It is not discordant with either of the other two instances because the specified outputs are different variables. For simplicity, the instances presented in the discussion will be limited to a single output variable (i.e. all outputs will be Z or Z' only). Implementing multiple outputs is an easy extension of the ideas presented here [3], but will not be addressed.

The collection of instances presented to the network is the **instance set**. The instance set is not stored explicitly in most models. Rather, the network will store some internal representation that allows it to execute the same function as the particular instance set it has been given. It has been shown that the number of instance sets that potentially represent the same functional information is infinite, and that therefore instance sets are not unique [3]. Furthermore, internal representations for the same instance set are diferent for different models. A network is said to **fulfill** a given instance set when the function it computes matches that specified by tthat instance set. Conflicts occur when a set of inputs could cause the network to output both high and low for some variable at the same time. The network resolves conflicts (is made **consistent**) by adding new nodes and variables to the network or deleting old conflicting ones. Precedence is given to the most recent instance presented to the network which is referred to as the **new instance** (NI).

Whenever a NI is presented to the network, each node *independently* determines its relationship to the NI. The nodes act in parallel, *without* information from any other nodes. This means they can also act asynchronously. Nodes take different adaptive actions based on their relationships to the NI. Each node, and by extension the network, reconfigures without control external to the network, the network is **self-organizing**.

After enough variables have been added so that the instance set is consistent with the NI, the network can enter the **Self-Deletion** (SD) phase. Nodes and variables that are *no longer needed* in discriminating inputs are deleted. The network does not lose information during SD, and will work properly without doing SD. Whereas consistency is absolutely necessary, SD is optional. SD allows the network to free nodes, an important factor when resources are limited (as in any physical implementation).

Learning in ASOCS. What follows is a brief example of the learning paradigm used in current ASOCS models. This is a brief overview of the process, presented to help the reader understand the sections that follow. This discussion deals only with instances and instance sets, not with particular models or networks, because internal representations differ from model to model. (It should be noted however, that as the "instance set" changes, a network must reconfigure to reflect those changes.) Assume the following instance set:

AB' -> Z

Then, the instance AB'CD -> Z' is added. There would be a conflict between the NI and the instance set when AB'.is matched To resolve the conflict, variables C and D must be added to the instance set. When this is completed, the instance set above will be replaced by the following one:

AB'C' -> Z

AB'D' -> Z

AB'CD -> Z'

During SD all, or only parts of some instances may be eliminated from the instance set. Two simple cases are given to illustrate this (by no means an exhaustive explanation). Suppose the following two instances are contained in some instance set:

ABC -> Z

ABCD -> Z

The second can be deleted because that case is covered by the first one. Consider the instances:

AB' -> Z

AB -> Z

These two can be replaced by:

A -> Z

which covers the same cases with less variables (and instances).

Once consistency is established, the instance set fulfills the NI asnd all parts of previous instances that do not contradict the NI. Once SD is complete, the instance set may contain fewer variables and instances than before. The NI presentation, comparison, adaptive actions and possible self-deletions occur *only once* for each NI. No iterative loops are necessary to obtain the correct instance set. This is reflected in the networks in that no iterative training loops are necessary to cause a network to learn properly.

Problem: Implementing Current Models in VLSI. The basic structure of a node in current models has two parts: a two-input logic gate capable of performing the 16 possible Boolean functions on two variables, and a control unit that tells it which function it is to perform at any given time. The node structure is illustrated in figure 1. Current models use hierarchies of the two-input nodes to represent instances in the current instance set. The problem that occurs is not a problem inherent in using the hierarchical structures in these models. Rather it is a problem of implementing the hierarchy in VLSI: the technology does not fit the models.

Recall that instances can have positive or negative polarities. Current models use two types of nodes to represent instances. The first type is called a **primitive node**, or **P-Node**. These "building-block" nodes have no real polarity, but compute their functions and feed their output to other nodes Figure 2 shows a representation of a P-Node, which has the function it computes written in, but no letter at the top representing the output and polarity. (Representing the nodes this way does not imply that this information is stored in the node itself--it may not be.) The other type is called a **discriminant node** or **D-Node**. Figure 2 also shows a D-Node, which has Z or Z' (Z bar) at the top, indicating the output variable and its polarity in addition to showing the function it computes In DNA, there are no "building-block" nodes. Every node is a D-Node. This distinguishes it from all previous models. Even though DNA is different, it still uses the ASOCS learning paradigm. Hence the name "Discriminant-Node ASOCS", or DNA.

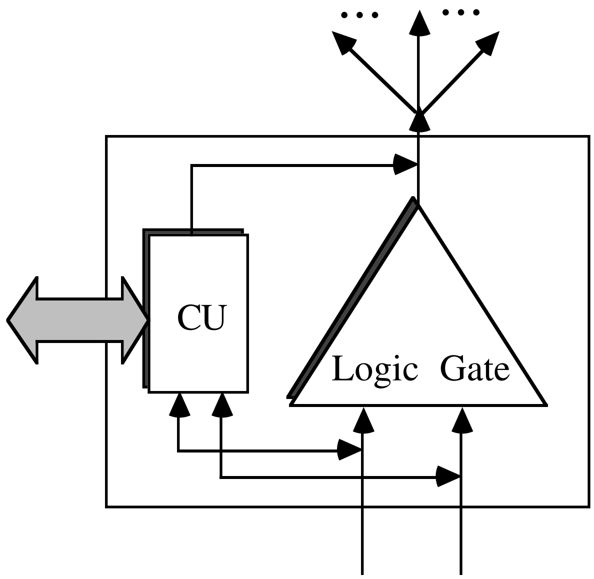

Fig. 1 - General Structure of ASOCS node

Both dynamic and static interconnect schemes [2] are found among current ASOCS models [3]. In considering a VLSI implementation of these hierarchical models, a problem associated with allocating unused nodes efficiently arises because the technology does not fit the models. Figure 3 shows how the instance set:

AB'C' -> Z

AB'D' -> Z

AB'CD -> Z'

might be represented using a dynamic scheme, which uses a router to change and set the links between nodes.

Implementing dynamic interconnect, routing through or around other nodes will become a problem as the number of allocated nodes in the network increases. A static interconnect is also preferred because it would be cheaper and simpler to implement. Figure 4 shows the

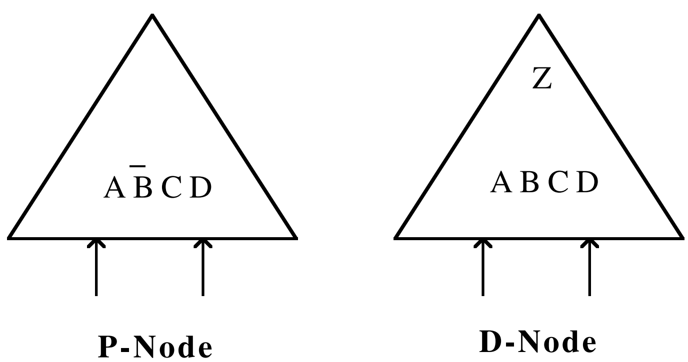

Fig. 2 - Representaion of an ASOCS P-Node and D-Node

instance set above represented using a static scheme. This one is a Binary Decision Tree [11] in which the links between nodes are fixed.

The problem is that it is not possible to decide beforehand which physical branch will require more nodes than the others, and thus which nodes will remain unused. In both the dynamic and static cases, allocating nodes from different parts of the network complicates the logic of the network. Each node represents only a small piece of some instance in the current instance set. Different pieces, i.e. nodes, have to be logically ordered so that the instance set is fulfilled. The physical position of a node in the network thus becomes important to the success of the network. Where the unused nodes are has an impact on how easily they can be allocated by a logical chain that may need them. It is the hie-rarchy of nodes that causes this problem. Yet it is the very nature of the computing nodes that makes the hierarchy necessary.

The basic structure of an ASOCS node was discussed earlier (see Fig. 1). The fact that these computing nodes have only two inputs is very important. The nodes are simple and computing their functions is very fast. They fit naturally into a hierarchical scheme that allows for bounded, O(log n) training and processing times. Because the nodes are limited to two inputs, each node is limited in the amount of information it can physically represent. As shown in figures 3 and 4, it may take several nodes to represent an instance or an instance set. (Refer again to figures 3 and 4. In these models each node computes the *AND* of its inputs). Representations of instances are "built" by feeding the output of some nodes into the inputs of others.

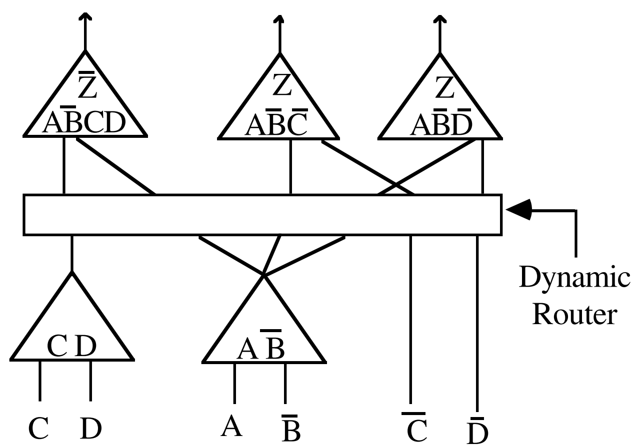

Fig. 3 - Dynamic Interconnect

Thus, it is clear that the problem with allocating unused nodes is tied to the existence of the hierarchy of nodes, which exists in turn because of the structure of the nodes. DNA's solution to the problem is to get rid of the hierarchical structure. As will be seen in the next section, this necessitates a change in the node structure.

DNA's Solution. DNA offers a solution to the problem of the unused nodes. There is no need for direct communication between nodes, so there is no hierarchical structure between nodes. This is because each DNA node in the network represents a complete instance in the *current representation* of the instance set. (For this reason DNA and similar models that are currently being developed are called Node-Per-Instance, or NPI models.) DNA nodes are capable of dynamically changing to represent new and modified instances. DNA nodes contain more information than nodes in other models. This is an important difference from previous models which, as explained above, must use a hierarchy of nodes to represent an instance. DNA nodes are logically independent of one another, so nodes can be placed anywhere in the network. Figure 5 shows the instance set of Figures 3 and 4 as it might be represented in the DNA model:

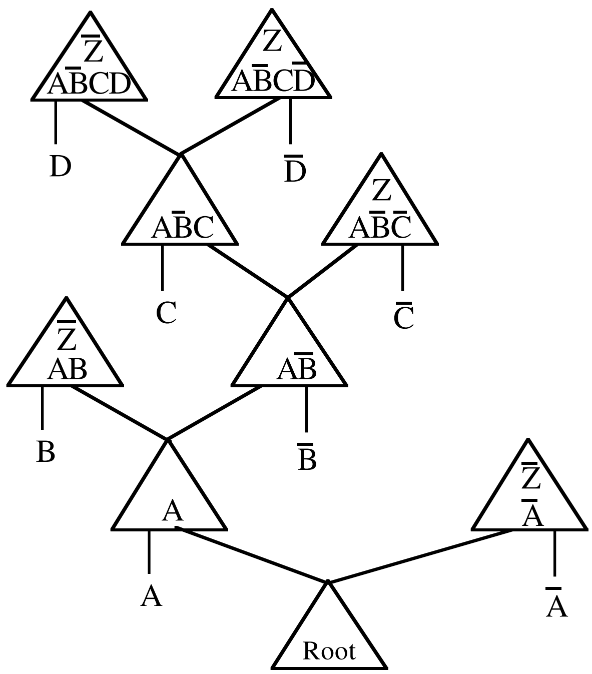

Fig. 4 - Static Interconnect

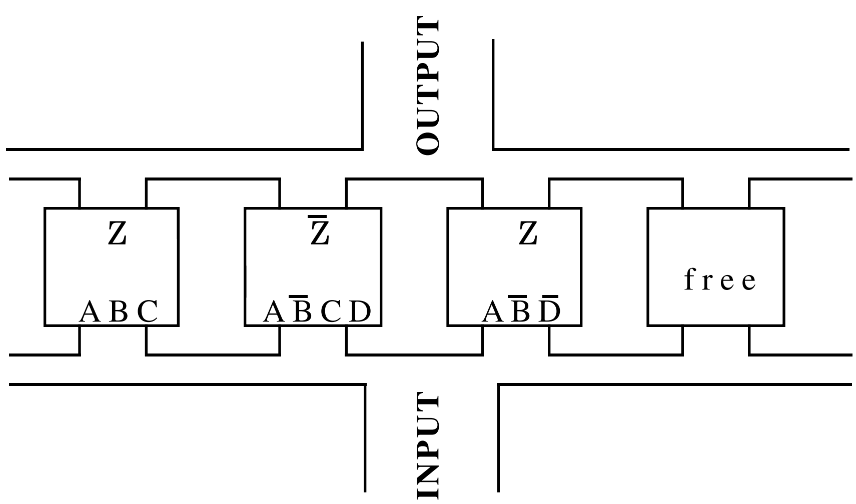

Fig. 5 - DNA nodes are independent

The bus represents the communication mechanism for broadcast of inputs and collection of outputs. ("Bus" here describes the functionality of the interconnect, but does not imply a specific implementation.) DNA has the ability to use any node in the network easily to represent instances, whereas this is not true of other models.

In order to get this flexibility and still maintain the advantages that other models have, the DNA node structure must capture what the hierarchy was doing in other models. There are a variety of ways to do this. One is to build each node as a separate hierarchical structure. Another is to time-multiplex input variables across a two-input logic gate with some extra memory at the node. A third is to use a PLA-like structure. How ever the node is actually implemented, it can logically be thought of as a control unit and an n-input logic gate, shown in figure 6 (as compared to the two-input gate of Fig.1). This difference in nodes (whether logical, physical or both) is the foundation of the differences between the DNA model and models that use node hierarchies.

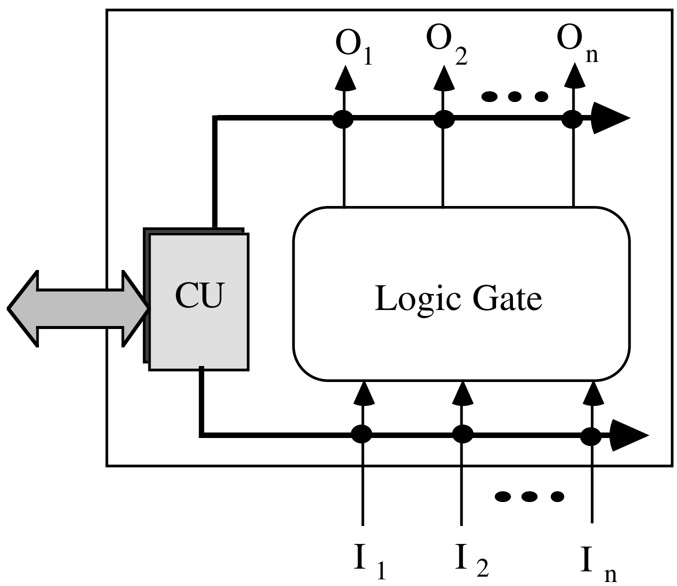

Fig. 6 - Logical Structure of DNA node

Processing and Learning in the DNA model. As in other ASOCS models, DNA has two modes--processing and learning. In processing mode, the inputs are broadcast to each of the nodes in parallel. Each node, *independent of all others*, determines whether or not it matches the inputs, and thus whether or not to put its outputs on the output bus. Since the representation of a given instance set is consistent, the outputs will always be consistent [3]. In the case that the input is matched by no node, two choices are available: either the network can output a "don't know" state, or choose the output for the instance that most closely matches the input. Which one is used is a matter of preference in implementation, but does not affect the model.

Figure 5 uses a bus, which has linear propagation time, to connect the nodes. However, because they are logically independent of one another, it does not matter how the nodes are arranged topologically as long as they can receive input and send output. Therefore a mesh, a cube, a tree, or some other polynomial or logarithmic structure can easily be used as an interconnect. It is important to note that while signals may have to pass through nodes to get to other nodes, the latter's computation is not influenced by the former.

In learning mode, the network uses the same hardware as in processing mode. However, the output must be broadcast to the nodes as well as the input (i.e. a NI is presented to the network). As above, each node independently determines how well it matches the information. A node does not put its outputs on the output bus as it might do above at this point. Instead, a node may change its function based on the result. The goal of any change is to make the overall function represented by the network consistent, so that there are no conflicting outputs. Since each node has all the information it needs, adaptation can take place asynchronously and concurrently among nodes (as in the other models).

Conclusion. The development of the DNA model is an important step from the abstract model toward the physical realization of an ASOCS machine that can be used to solve problems. It is designed to better fit the requirements for VLSI implementation than its predecessors. The advantages and limitations of the model should be seen with this in mind. Its advantages are:

1. It alleviates the unused node allocation problem that current models have. Unused nodes are easier to use in DNA.
2. The network no longer depends on a logical hierarchy of nodes for success. Physical location in the network is no longer a factor.
3. As a result of #2, instances can be placed anywhere in the network, and easily relocated without affecting the logic.
4. Different interconnect schemes may be used: dynamic, static, trees, hypercubes, busses, etc. without affecting the success of the network (although some topologies may be preferred over others).
5. As a result of #4, it is still possible to obtain fast processing and learning times as in other models, using the proper interconnect.

Its limitation is:

1. The node will be slightly more complex than nodes in current models, owing to the need for flexibility in what it represents.

Applications for ASOCS have been found in adaptive logic, system fault management, robotics and dynamic control. As an ASOCS model, and potential implementation, DNA could be used for any of these.

Current research focusses on five main areas:

1. VLSI implementation of DNA and other NPI models.
2. Use of a priority scheme with instances to limit the growth of a network.
3. New learning algorithms that improve and extend the power of current models.

4. Methods of extending functionality beyond Boolean to higher-order functions and more complex data.

5. Finding new applications for DNA.

Bibliography

[1] Chang, J. and J.J. Vidal., "Inferencing in Hardware," *Proceedings of the MCC University Research Symposium,* Austin TX. July 1987.

[2] Feng, Tse-yun, "A Survey of Interconnection Networks," *Computer,* Vol. 14, #12. pp. 12-27. December 1981.

[3] Martinez, T.R., *Adaptive Self-Organizing Logic Networks*., UCLA, Ph.D.Dissertation. May 1986.

[4] Martinez, T.R., "Models of Parallel Adaptive Logic," *Proceedings of the 1987IEEE Systems Man and Cybernetics Conference*, pp.290-296, October 1987.

[5] Martinez, T.R., J.J. Vidal, "Adaptive Parallel Logic Networks," *Journal of Parallel and Distributed Computing*, Vol. 5, #1, pp. 26-58, 1988.

[6] Martinez, T.R., "Digital Neural Networks," *Proceedings of the 1988 IEEE Systems Man and Cybernetics Conference*, pp. 681-684, August 1988.

[7] Martinez, T.R., "Memory Efficient Adaptive Logic," submitted.