Junctoids also differ from traditional, back-propagation based architectures than back-propagation algorithms. The main resulting advantage of conjunctoids over TNNs is that parameters being estimated in the TNN case. These parameters are updated as each new datum is received, based on statistical theory. Simultaneously, these parameters may be used to estimate values of unknown variables, which would correspond to the performance phase.

The two components that implement conjunctoid learning and performance are separable and they lead themselves easily to parallel implementation. Thus, a massively parallel machine such as the NCUBE at USC may be effectively used to exploit conjunctoid properties. To further increase speed, a VLSI implementation comprising a multitude of processors operating in parallel is also feasible. As will be shown, learning and performance require only a few mathematical computations, and thus each of the required processing elements may be relatively simple and fast.

This paper addresses several ongoing efforts at USC to implement conjunctoid algorithms on parallel platforms. These include: (a) a PC simulation of a full-blown conjunctoid model functioning in a pattern completion setting (serial implementation); (b) a description of the full-blown conjunctoid implemented on a NCUBE machine; and (c) a conceptual design of a special purpose integrated circuit.

II. Conjunctoid Implementation:

A. Full-blown conjunctoids for small scale problems:

A pattern completion example [2] will be illustrated to explain how the full-blown conjunctoid operates. Figure 1 shows a 7-segment LED array that may be used to represent user-defined symbols, such as the numbers 0 through 9. Each of the seven LED segments may each have a value of 0 (indicating that it is off), 1 (indicating that it is on) or possibly 0.5 (indicating that it may be either off or on). In this setting, an appropriately specified conjunctoid would learn to recognize the digits 0 through 9, and then perform by identifying or completing the pattern given incomplete information. A total of 2^7 combinations are present, of which 10 numbers are to be learned corresponding to the numbers 0 through 9. The learning phase involves updating a numerical value associated with each possible combination. The appropriate probability model may be formulated as,

\[
Pr \{ W = w ; \alpha \} = \alpha_w \cdot 0 \leq w \leq 0.1^7
\]

The statement thus reads: The probability that a given input vector is equal to one from the 2^7 combinations is the value associated with that possibility combination. To start with, since there are 2^7 equally possible combinations, the \( \alpha \) values associated with each of these combinations will be 1/2^7.
remains at unity.

digits 0 through 9, after which an input vector. Assume that the device has learned the pattern for the
from the so-called possibility set, identifying the \( w(OLiT) \) value with the highest probability (or one
likely output possible when data (bits) are missing in the
0, 1) corresponding to "3"; since "3" has been learned and has
mance. The effect on learning is to distribute parameter
completed pattern. In general, the device will perform by iden-
ty within its realm. Each node then passes its maximum
usory information” is available, in which case other
start values may be used [3]. Learning involves reading a
pecific input pattern, \( w^{(0)} \), say that corresponding to "0", which is \( \{1, 1, 1, 1, 1, 0\} \), along with a certain learning
weight. The corresponding probability \( \alpha_{c(1,1,1,1,1,0)} \) is increased while values for all other combinations are
decreased, reflecting the fact that the LED pattern for "0" has
been learned. Other numerical combinations are similarly learned, with the learning weight so adjusted that as each
number is learned, the probabilities associated with the learned values are increased while at the same time reducing
the probability of unlearned combinations.

When input values are not missing, that is the pattern is
known completely, simple input statistics are of the form,
\[
I(w) = \begin{cases} 1 & \text{if } w = w^{(0)} \\ 0 & \text{if } w \neq w^{(0)} \end{cases} \quad (2)
\]
Output parameters are then updated as per the equation
\[
\alpha_{c}^{(w^{(0)})} = \frac{L \cdot I(w^{(0)}) + \alpha_{c}^{(w^{(0))}}}{1 + L}, \quad w \in \{0, 1\}^{7}. \tag{3}
\]
It can be seen that by using equations (2), (3) and (4),
throughout the learning process, the sum of all \( \alpha \) values remains at unity. Performance would correspond to determining the most
likely output possible when data (bits) are missing in the \( w^{(0)} \) vector. Assume that the device has learned the pattern for the
digits 0 through 9, after which an input \( w^{(0)} \) with the value of
\( \{1, 1, 1, 1, 0, 0, 1\} \) is presented to the device, where 0.5 implies that the value is unknown. Two outcomes are possible,
\( \{1, 1, 1, 1, 0, 0, 1\} \) corresponding to "C" and \( \{1, 1, 1, 1, 0,
0, 1\} \) corresponding to "3"; since "3" has been learned and has a higher \( \alpha \) value, pattern \( \{1, 1, 1, 1, 0, 0, 1\} \) is chosen as the
completed pattern. In general, the device will perform by identifying
the \( w^{(0)} \) value with the highest probability (or one such value in case of a tie) among all values that are possible from the so-called possibility set,
\[
P(w^{(0)}) = \{ u : u \in \{0, 1\}; w_{k}^{(0)} = 0, 1 \to w_{k} = w_{k}^{(0)}; \}
\tag{5}
\]
Missing data have an effect on learning as well as perfor-
ance. The effect on learning is to distribute parameter
updating uniformly among all patterns in the possibility set. If
\( w^{(0)} \) was \( \{5, 5, 5, 1, 0, 0, 5\} \), along with a positive learning
weight, for instance, the overall effect would be to increase the
8 elements of \( \alpha \) having subscripts in the corresponding possi-
bility set and to decrease all others. One way of representing
this type of learning is to replace the sufficient statistic for-
formula of equation (4) with the general formula,
\[
x_{w}^{(0)} = \sum_{k=1}^{L} L_{k} I(w(w^{(0)}) - w_{k}) \alpha_{k}(w^{(0)}), \quad w \in \{0,1\}^{7}. \tag{6}
\]
It is straightforward to verify that (4) corresponds to (6) when
no \( w^{(0)} \) values are missing and that (6) produces the desired learning distribution effect otherwise.

Besides learning to perform pattern completion tasks,
conjunctoids can learn to perform error detection and correc-
tion tasks as well. It is also possible to alter learning weights
so that certain patterns are given greater emphasis, relative to
other patterns. This is helpful for applications where an
incomplete pattern would in general give more than one
acceptable pattern but only the one with the greatest frequency
is to be chosen (i.e., the one with a greater probability is to be
recognized). In addition, it is possible to "unlearn" previously
learned information through the use of negative learning
weights [2].

B. NCUBE implementation:

The NCUBE is a 1024 processor machine with a SUN4
host interface. The node processors are connected in the form
of a 10 dimensional hypercube. Programs for the nodes are
written and compiled on the host and downloaded to the node.
A matter of concern that arises while programming on parallel
processors is that of communication delays in using informa-
tion residing among individual processors. Fortunately, how-
ever, the communication involved in the NCUBE environment
is a bare minimum.

Figure 2 gives an outline of an algorithm that is analo-
gous to the previous 7-segment display example excepting
that the dimensions of the problem are greatly increased, with
the intent being to learn alphanumeric character patterns on a
20-segment LED. Figure 3 shows one possible layout for a
20-segment LED array. The entire vector space would thus
consist of \( 2^{20} \) combinations. Provided that all of the \( 2^{10} \) =
(1024) NCUBE processors are utilized, each processor can be
made to work upon \( 2^{10}/2^{10} = 2^{6} \) computations. The user may
specify that only \( 2^{6} \) processors be used instead. \( \alpha^{(0)} \) =
\( 1, 2, \ldots, 10 \) in which case \( 2^{5} \) computations would have
be performed on each one. Thus, the computational
load would be evenly distributed among all the processors.
The host control and the node computational program are
written in either concurrent FORTRAN or C.

Figure 2 also illustrates the high level constructs for the
host and node programs. The host program downloads the
node program to each of the nodes. It also passes data to the
nodes by calling certain functions. Once the nodes have
finished the processing (learning and performance) and are
ready with the results, this information is sent back to the host.
During the learning phase, the node program updates the \( \alpha 
values of the bit combinations allocated to it. Here again, as in
the earlier example, all members of the possibility set within
the domain of a node are determined and the corresponding \( \alpha 
values are updated as per equations (4), (5) and (6). During the
performance phase, each node computes its possibility set and
determines the vector combination with the greatest proba-
bility within its realm. Each node then passes its maximum \( \alpha 
value and address (\( \alpha \) subscript) to the host; the host in turn
goes through a loop that computes the global maximum from
among the local maxima provided by the nodes. The address
Determine the number of nodes to be used.

Read incomplete pattern and learning weight (if any) from the controller.

Provide incomplete pattern and learning weight (if any) to the nodes.

Do learning and performance as per data received.

Read local maxima from each node.

Compute the local maxima.

Compute the global maxima.

Return the local maxima to the control processor.

Algorithm for host processor.

Algorithm for node processors.

NCUBE Host and Node operations for the full-blown conjunctoid implementation.

One possible layout for a 20-Segment Display.

vector corresponding to that maximum $\alpha$ value is the best determined complete pattern.

C. VLSI implementation:

The proposed VLSI architecture consists of a controller chip and 1024 node chips. All node chips are identical, except that their identity or node numbers are hard wired to permit addressing them. The nodes may be thought of as functionally analogous to the nodes on a hypercube, except there is no form of internode communication. The control processor interfaces to a host machine and the various nodes. Alternative nodes were designed earlier by Wei [4] for serial implementation. The node processors below are refinements and extensions to handle parallel implementation with missing data. Thus, given a vector size of 20, $2^{20}$ nodes are designed to perform $2^{20}$ computations each, resulting in a total of $2^{20}$ computations for an application problem of size 20. Each node would have a 10 bit counter running from 0 through $2^{10} - 1$ counts. The $\alpha$ parameters are continually updated at every clock cycle. Learning and performance occur simultaneously, so that after $2^{10}$ cycles, one should have the completed pattern (for an incomplete one) and its corresponding $\alpha$ value.

A VLSI design approach is outlined in Figure 4 to parallelize the task of pattern completion and similar applications. The design consists of a controller chip and component node chips. The pattern vector, $m^{(20)}$, refers to the 20 bit pattern vector of 0's and 1's, corresponding to the kind encountered in the NCUBE example. The missing status vector, $w^{(20)}$, refers to the vector pattern that indicates what bit positions are unknowns in the original binary pattern $w^{(20)}$ ($w^{(20)}$ is broken down in this way to get around the difficulty of having to deal with tristates). The learning weight, $L$, refers to an integer number that is user specified. The pattern with the maximum $\alpha$ is the completed or constructed pattern.

The control processor in Figure 4 acts as the interface between the host and the nodes. It manages the flow of data between them. It accepts information from the host such as $m^{(20)}$, $w^{(20)}$ and $L$. These would be specified by the applications user, working at a high level (on a PC possibly). Figure 5 shows the structure of a finite state machine for the control processor, which runs it through the states indicated. Figure 6 is a high level functional block diagram for the controller. The control processor receives the pattern vector, the status vector, and the learning weight information from the host signaled by the STROBE signal. It then makes this information available to the node processors on the bus, strobing it with
the ALE signal and activating WRITE. While the nodes go through a loop counting from \(0\) through \(2^9 - 1\) to update their \(\alpha\) parameters, the controller reads local maximum \(\alpha\) values from the nodes for the preceding \(m^{(n)}\). The control processor does this by stepping through a similar count synchronously with the node processors. At each count, the \(\alpha\) value from the addressed node is read and checked against the greatest \(\alpha\) value, \(\alpha_{\text{max}}\), value thus far obtained. If the \(\alpha\) value read is greater than \(\alpha_{\text{max}}\), \(\alpha_{\text{max}}\) is updated, provided that the node is in the possibility set of the preceding pattern. Thus, while each
of the nodes is busy computing the $\alpha$ values and the maximum $\alpha$ value within their domain, the controller is kept busy determining the maximum $\alpha$ value (along with the pattern that goes with it) for the preceding $m^{(b)}$. At the end of the count, it would have the completed pattern stored in a register and would strobe it on the data bus with the DONE signal for the host to receive.

The node processor essentially performs the learning (and performance) of a given pattern vector, $m^{(b)}$. Every node processor first reads $m^{(b)}, \mu^{(b)}$, and $l$ as strobed by the control processor. Then, as the control processor steps through its $2^b$ counts to determine the global maxima, the node processors begin their count to update that $\alpha$ value with count as its subscript. The learning factor, as modified by unknown values at the input, is accounted for during the learning and performance phases. Figure 7 and Figure 8 give a state machine and a high level block diagram respectively for the node processor. A simple advantage of this structure is that a bigger input vector may easily be accommodated by utilizing a larger number of node processors.

III. Future Directions

Besides the full-blown model, a variety of families may be specified to exploit the conjuncts among input variables. As different families are identified for various applications, their implementation on parallel platforms will be carried on. The conceptual design that has been laid out in this paper raises interesting issues for further study. These include new board layouting and signal buffering that has been described in this paper. VLSI implementation issues such as optimal system timing and control signal generation, as well as timing synchronization between the control processor and node processors need also be addressed. A suitable network interconnection topology for the nodes may reduce considerably the process of generating $\alpha_{max}$. It would also be practical to devise an architecture wherein each of the node processors does its part of the computation in parallel, rather than in the serial fashion that we have described in this paper.

References