Integer-Encoded Massively Parallel Processing of Fast-Learning Fuzzy ARTMAP Neural Networks

Hubert A. Bahr[a], Ronald F. DeMara[b] , Michael N. Georgiopoulos[b]

[a]HQ STRICOM, AMSTI-ET, 2350 Research Boulevard, Orlando, FL 32826
[b]Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816

ABSTRACT

In this paper we develop techniques that are suitable for the parallel implementation of Fuzzy ARTMAP networks. Speedup and learning performance results are provided for execution on a DECmpp/Sx-1208 parallel processor consisting of a DEC RISC Workstation Front-End (FE) and MasPar MP-1 Back-End (BE) with 8,192 processors. Experiments of the parallel implementation were conducted on the Letters benchmark database developed by Frey and Slate. The results indicate a speedup on the order of 1000-fold which allows combined training and testing time of under four minutes.

KEYWORDS: Fuzzy ARTMAP Neural Network, Parallel Processing, DEC/mpp, Letters Benchmark, Massively Parallel

1. INTRODUCTION

Adaptive resonance theory was introduced by Grossberg in 1976 as a means of describing how recognition categories are self-organized in neural networks[1]. Since this time, a number of specific neural network architectures based on ART have been proposed. Many of these architectures originated from Carpenter, Grossberg, and their colleagues at Boston University. A major separation amongst the ART architectures developed so far is in the categories of *unsupervised* versus *supervised* ART architectures. A prominent member of the class of unsupervised architectures is Fuzzy ART[2], while a prominent member of the class of supervised architectures is Fuzzy ARTMAP[3]. Fuzzy ART can cluster arbitrary collections of binary or analog input patterns, while Fuzzy ARTMAP can implement any mapping from an input space of arbitrary dimensionality to an output space of arbitrary dimensionality. There is a high degree of correlation between the Fuzzy ART and the Fuzzy ARTMAP architectures, because a number of components of Fuzzy ARTMAP are Fuzzy ART modules.

Our primary focus in this paper is the parallel implementation of Fuzzy ARTMAP. Analog and digital VLSI implementations of simpler than the Fuzzy ARTMAP architectures (e.g., ART1) have appeared in the literature[4-7]. To the best of our knowledge though, no effort has been made so far to implement Fuzzy ARTMAP on general purpose massively parallel hardware.

The parallel implementation of Fuzzy ARTMAP is applied on a DECmpp/Sx-1208 parallel processor consisting of a DEC RISC Workstation Front-End (FE) and a MasPar MP-1 Back-End (BE) with 8,192 processors. Processing is divided into FE routines that perform input/output (I/O), integer scaling, and data set randomization, and BE routines that train and test the network. Experiments performed on the Letters benchmark database[8], developed by Frey and Slate, indicated a speed-up of 1000-fold which resulted in a combined training and testing time of under 4 minutes.

The organization of the paper is as follows. In Section 2 we describe the Fuzzy ART neural network since it constitutes the building block in the design of a Fuzzy ARTMAP neural network. In Section 3, we continue with the description of the Fuzzy ARTMAP neural network. In Section 4, we present the implementation environment where Fuzzy ARTMAP will be implemented. In Section 5, we emphasize issues pertaining to the parallel implementation of Fuzzy ARTMAP. Finally, in Section 6 we discuss the experimental results, while in Section 7 we provide some concluding remarks.

Further author information –
H.A.B.: e-mail: hab@ece.engr.ucf.edu; Telephone: 407-384-3874; Fax: 407-679-9329
R.F.D.: e-mail: rfd@ece.engr.ucf.edu; Telephone: 407-823-5916; Fax: 407-823-5835
M.N.G.: e-mail: mng@ece.engr.ucf.edu; Telephone: 407-823-5338; Fax: 407-823-5835

2. Fuzzy ART

A brief overview of the Fuzzy ART architecture is provided in the following sections. For a more detailed discussion of this architecture, the reader should consult G. A. Carpenter[2].

2.1 Fuzzy ART Architecture

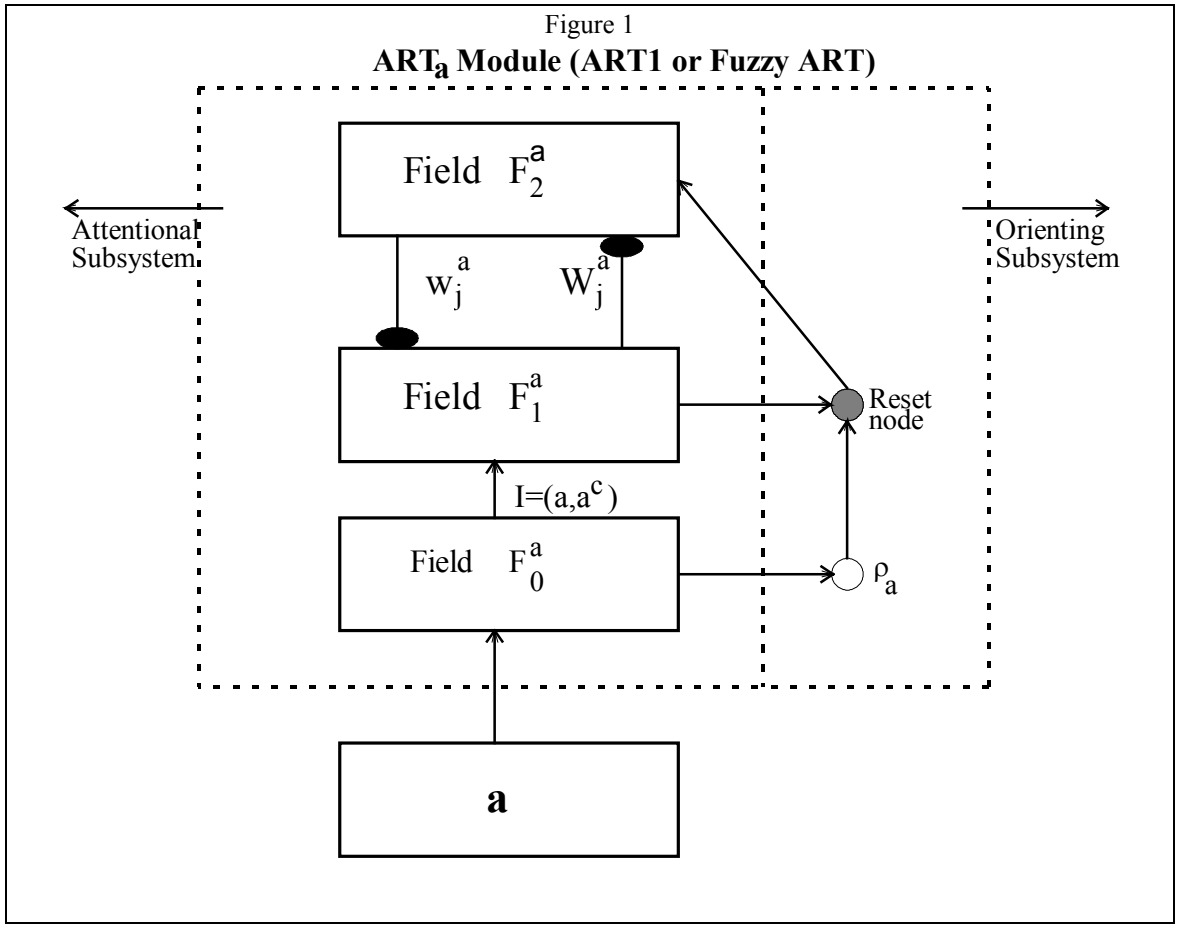
The Fuzzy ART neural network architecture is shown in Figure 1. It consists of two subsystems, the *attentional subsystem*, and the *orienting subsystem*. The attentional subsystem consists of two fields of nodes denoted F_1^a and F_2^a . The F_1^a field is called the *input field* because input patterns are applied to it. The F_2^a field is called the *category* or *class representation field* because it is the field where category representations are formed. These categories represent the clusters to which the input patterns belong. The orienting subsystem consists of a single node (called the *reset node*), which accepts inputs from the F_1^a field, the F_2^a field (this input is not shown in Figure 1), and the input pattern applied across the F_1^a field. The output of the reset node affects the nodes in the F_2^a field.

Some preprocessing of the input patterns of the pattern clustering task takes place before they are presented to Fuzzy ART. The first preprocessing stage takes as an input an M_a-dimensional input pattern from the pattern clustering task and transforms it into an output vector $\mathbf{a} = (a_1,\ldots, a_{Ma})$, whose every component lies in the interval [0, 1] (i.e., $0 \le a_i \le 1$ for $1 \le i \le M_a$). The second preprocessing stage accepts as an input the output a of the first preprocessing stage and produces an output vector I, such that

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1,\ldots, a_{Ma}, a^c{}_1,\ldots, a^c{}_{Ma}), \tag{1}$$

where

Figure 1



$$a_i^c = 1 - a_i \ ; 1 \le i \le M_a \tag{2}$$

The above transformation is called complement coding. The complement coding operation is performed in Fuzzy ART at a preprocessor field designated by F_0^a (see Figure 1). We will be refer to the vector **I** formed in this fashion as the *input pattern*.

We denote a node in the F_1^a field by the index i ($i \in \{1, 2,\ldots, 2M_a\}$), and a node in the F_2^a field by the index j ($j \in \{1, 2,\ldots, N_a\}$). Every node i in the F_1^a field is connected via a bottom-up weight to every node j in the F_2^a field; this weight is denoted W_{ij}^a. Also, every node j in the F_2^a field is connected via a top-down weight to every node i in the F_1^a field; this weight is denoted w_{ij}^a. The vector whose components are equal to the top-down weights emanating from node j in the F_2^a field is designated $\mathbf{w}_j^a$, and is referred to as a *template*. Note that $\mathbf{w}_j^a = \left(w_{j1}^a, w_{j2}^a, \ldots, w_{j,2M_a}^a\right)$ for $j = 1,\ldots, N_a$. The vector of bottom-up weights converging to a node j in the F_2^a field is designated $\mathbf{W}_j^a$. Note that in Fuzzy ART the bottom-up and top-down weights corresponding to a node j in F_2^a are equal. Hence, in the forthcoming discussion, we will primarily refer to the top-down weights of the Fuzzy ART architecture. Initially, the top-down weights of Fuzzy ART are chosen to be equal to the "all-ones" vector. The initial top-down weight choices in Fuzzy ART correspond to the values of these weights prior to presentation of any input pattern to the Fuzzy ART architecture.

Before proceeding, it is important to introduce the notations $\mathbf{w}_j^{a,old}$ and $\mathbf{w}_j^{a,new}$. Quite often, templates in Fuzzy ART are discussed with respect to an input pattern **I** presented at the F_1^a field. The notation $\mathbf{w}_j^{a,old}$ denotes the template of node j in the F_2^a field of Fuzzy ART prior to the presentation of **I**. The notation $\mathbf{w}_j^{a,new}$ denotes the template of node j in the F_2^a after the presentation of **I**. Similarly, any other quantities defined with superscripts {*a, old*} or {*a, new*} will indicate values of these quantities prior to or after a pattern presentation to Fuzzy ART, respectively.

2.2 Operation of Fuzzy ART

As mentioned previously, we will use I to indicate an input pattern applied at F_1^a, and $\mathbf{w}_j^a$ to indicate the template of node j in F_2^a. In addition, we will use $|\mathbf{I}|$ and $|\mathbf{w}_j^a|$ to denote the size of **I** and $\mathbf{w}_j^a$, respectively. The size of a vector in Fuzzy ART is defined to be the sum of its components. Furthermore we define $\mathbf{I} \Lambda \mathbf{w}_j^a$ to be the vector whose i-th component is the minimum of the i-th I component and the i-th $\mathbf{w}_j^a$ component. The operation Λ is called the *fuzzy-min* operation.

Let us assume that an input pattern **I** is presented at the F_1^a field of Fuzzy ART. The appearance of pattern **I** across the F_1^a field produces bottom-up inputs that affect the nodes in the F_2^a field. These bottom-up inputs are given by the equation:

$$T_j^a(\mathbf{I}) = \frac{\left|\mathbf{I} \Lambda \mathbf{w}_j^{a,old}\right|}{\left(\alpha_a + \left|\mathbf{w}_j^{a,old}\right|\right)} \tag{3}$$

where α_a, which takes values in the interval $(0, \infty)$, is called the *choice parameter*. It is worth mentioning that if in the above equation $\mathbf{w}_j^{a,old}$ is equal to the "all-ones" vector, then this node is referred to as an *uncommitted node*; otherwise, it is referred to as a *committed node*.

The bottom-up inputs activate a competition process among the F_2^a nodes, which eventually leads to the activation of a single node in F_2^a, namely the node which receives the maximum bottom-up input from F_1^a. Let us assume that node j_{max} in F_2^a has been activated through this process. The activation of node j_{max} in F_2^a indicates that this node is considered as a potential candidate by Fuzzy ART to represent the input pattern **I**. The appropriateness of this node is checked by examining the ratio

$$\frac{\left|\mathbf{I} \Lambda \mathbf{w}_{j_{\max}}^{a,old}\right|}{|\mathbf{I}|} \tag{4}$$

If this ratio is smaller than ρ_a, then node j_{max} is deemed inappropriate to represent the input pattern I, and as a result it is reset (deactivated). The parameter ρ_a, called the *vigilance parameter*, takes values in the interval [0, 1]. The deactivation process is

carried out by the orienting subsystem, and in particular by the reset node. If a reset happens, another node in F_2^a (different than node j_{max}) is chosen to represent the input pattern **I**; resets last for the entire input pattern presentation. The above process continues until an appropriate node in F_2^a is found, or until all the nodes in F_2^a have been considered. If a node in F_2^a is found appropriate to represent the input pattern **I**, then learning ensues according to the following rules:

Assuming that node j_{max} has been chosen to represent **I**, the corresponding top-down weight vector $\mathbf{w}_{j_{\max}}^{a,old}$ becomes equal to $\mathbf{w}_{j_{\max}}^{a,new}$, where

$$\mathbf{w}_{j_{\max}}^{a,new} = \left(\mathbf{I} \Lambda \mathbf{w}_{j_{\max}}^{a,old}\right) \tag{5}$$

It is worth mentioning that in equation (5) we might have $\mathbf{w}_{j_{\max}}^{a,new} = \mathbf{w}_{j_{\max}}^{a,old}$; in this case we say that no learning occurs for the weights of node j_{max}. Also note that equation (5) is actually a special case of the learning equations of Fuzzy ART that is referred to as *fast learning*[2]. In this paper we only consider the fast learning case. We say that node j_{max} has coded input pattern **I** if during **I**'s presentation at F_1^a , node j_{max} in F_2^a is chosen to represent **I**, and j_{max}'s top-down weights are modified as equation (5) prescribes. Note that the weights converging to or emanating from an F_2^a node other than j_{max} (i.e., the chosen node) remain unchanged during **I**'s presentation.

3. Fuzzy ARTMAP

A brief overview of the Fuzzy ARTMAP architecture is provided in the following sections. For a more detailed discussion of this architecture, the reader should consult G. A. Carpenter et. al.[3]

3.1 Fuzzy ARTMAP Architecture

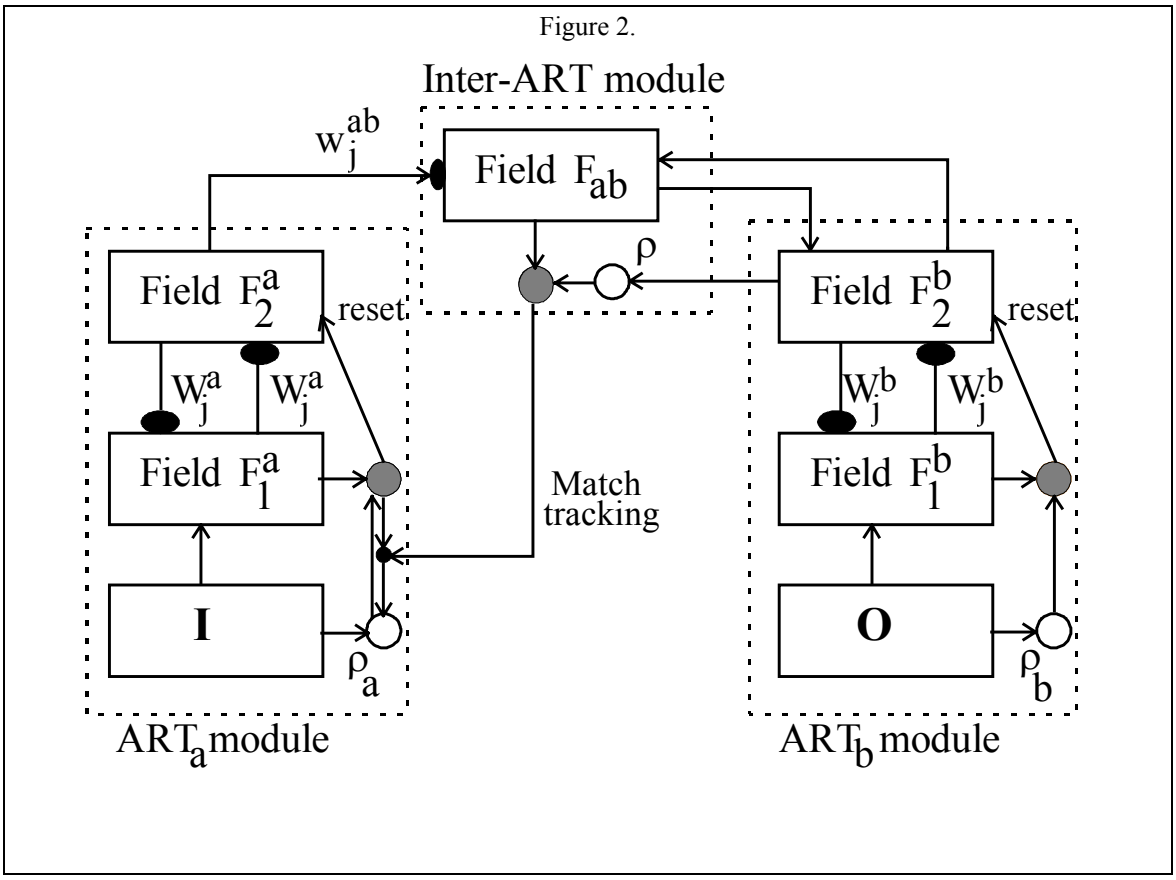
A block diagram of the Fuzzy ARTMAP architecture is provided in Figure 2. Note that two of the three modules in Fuzzy ARTMAP are Fuzzy ART architectures. These modules are designated ART_a and ART_b in Figure 2. The ART_a module accepts as inputs the input patterns, while the ART_b module accepts as inputs the output patterns of the pattern classification task. All of the details in Section 2 are valid for the ART_a module without change. The same for the ART_b module by if the superscript *a* of Section 2 is replaced with the superscript *b* to emphasize the fact that we are referring to weights and parameter values of the $FuzzyART_b$ module. One of the differences between the ART_a and the ART_b modules in Fuzzy ARTMAP is that for pattern classification tasks (many-to-one maps) it is not necessary to apply complement coding to the output patterns presented to the ART_b module.

As illustrated in Figure 2, Fuzzy ARTMAP contains a module that is designated the inter-ART module. The purpose of this module is to make sure the appropriate mapping is established between the input patterns presented to ART_a, and the output patterns presented to ART_b. There are connections (weights) between every node in the F_2^a field of ART_a, and all nodes in the F_{ab} field of the inter-ART module. The weight vector with components emanating from node *j* in F_2^a , and converging to the nodes of F_{ab} the field is denoted $\mathbf{w}_j^{ab} = \left(w_{j1}^{ab},\ldots,w_{jk}^{ab},\ldots,w_{jN_b}^{ab}\right)$, where N_b are the number of nodes in F_{ab} (the number of nodes in F_{ab} is equal to the number of nodes in F_2^b). There are also fixed bidirectional connections between a node k in F_{ab}, and its corresponding node k in F_2^b .

3.2 Operation of Fuzzy ARTMAP

The operation of the Fuzzy ART modules in Fuzzy ARTMAP is a slightly different from the operation of Fuzzy ART described in Section 2. For instance, resets in the ART_a module of Fuzzy ARTMAP occur either because the category chosen in F_2^a does not match the input pattern presented at F_1^a , or because the appropriate map has not been established between an input pattern presented at ART_a, and its corresponding output pattern presented at ART_b. This latter type of reset is enforced by the inter-ART module via its connections with the orienting subsytem in ART_a (see Figure 2). This reset is accomplished by forcing the ART_a architecture to increase its vigilance parameter value above the level that is necessary to cause a reset of

Figure 2.



the activated node in the F_2^a field. Hence, in the ART_a module of Fuzzy ARTMAP, we identify two vigilance parameter values, a baseline vigilance parameter value $\overline{\rho}_a$ which is the vigilance parameter of ART_a prior to the presentation of an input/output pair to Fuzzy ARTMAP, and a vigilance parameter ρ_a that corresponds to the vigilance parameter that is established in ART_a via appropriate resets enforced by the inter-ART module. Also, the node activated in F_2^b due to a presentation of an output pattern at F_1^b can either be the node receiving the maximum bottom-up input from F_1^b, or the node that is designated by the F_{ab} field in the inter-ART module. The latter type of activation is enforced by the connections between the F_{ab} field and the F_2^b field.

All of the equations in Section 2 for the Fuzzy ART module are valid for the ART_a and ART_b modules in Fuzzy ARTMAP. In particular, the bottom-up inputs to the F_2^a field and the F_2^b field are given by:

$$T_j^a(\mathbf{I}) = \frac{\left|\mathbf{I} \Lambda \mathbf{w}_j^{a,old}\right|}{\left(\alpha_a + \left|\mathbf{w}_j^{a,old}\right|\right)} \tag{6}$$

and

$$T_k^b(\mathbf{O}) = \frac{\left|\mathbf{O} \Lambda \mathbf{w}_k^{b,old}\right|}{\left(\alpha_b + \left|\mathbf{w}_k^{b,old}\right|\right)} \tag{7}$$

where in equation (7), **O** stands for the output pattern associated with the input pattern **I**, while the rest of the ART_b quantities are defined as they were defined for the ART_a module in Section 2. Similarly, the vigilance ratios for ART_a and ART_b are computed as follows:

$$\frac{\left|\mathbf{I} \Lambda \mathbf{w}_J^{a,\,old}\right|}{\left|\mathbf{I}\right|} \tag{8}$$

and

$$\frac{\left|\mathbf{I} \Lambda \mathbf{w}_K^{b,\,old}\right|}{\left|\mathbf{I}\right|} \tag{9}$$

The equations that describe the modifications of the weight vectors $\mathbf{w}_j^{ab}$ can be explained as follows: A weight vector emanating from a node in F_2^a to all the nodes in F_{ab} is initially the "all-ones" vector and, after training that involves this F_2^a node, all of its connections to F_{ab}, except one, are reduced to the value of zero.

3.3 Operating Phases of Fuzzy ARTMAP

Fuzzy ARTMAP may operate in two different phases: training and performance (testing). The *training phase* of Fuzzy ARTMAP works as follows: Given the training list $\{\mathbf{I}^1; \mathbf{O}^1\}$, $\{\mathbf{I}^2; \mathbf{O}^2\}$, : : :, $\{\mathbf{I}^P ; \mathbf{O}^P\}$, we want Fuzzy ARTMAP to map every input pattern of the training list to its corresponding output pattern. In order to achieve the aforementioned goal, present the training list repeatedly to the Fuzzy ARTMAP architecture. That is, present $\mathbf{I}^1$ to ART_a and $\mathbf{O}^1$ to ART_b, then $\mathbf{I}^2$ to ART_a and $\mathbf{O}^2$ to ART_b, and eventually $\mathbf{I}^P$ to ART_a and $\mathbf{O}^P$ to ART_b; this corresponds to one list presentation. Present the training list as many times as is necessary for Fuzzy ARTMAP to classify the input patterns. The classification (mapping) task is considered accomplished (i.e., the learning is complete) when the weights do not change during a list presentation. The aforementioned training scenario is called *off-line training*.

In the *performance phase* of Fuzzy ARTMAP the learning process is disengaged, and input/output patterns from a *test list* are presented in order to evaluate the classification performance of Fuzzy ARTMAP. In particular, during the performance evaluation of Fuzzy ARTMAP, only the input patterns of the test list are presented to the ART_a module of Fuzzy ARTMAP. Every input pattern from the test list will choose a node in the F_2^a field. If the output pattern to which the activated node in F_2^a is mapped matches the output pattern to which the presented pattern should be mapped, then Fuzzy ARTMAP classified the test input pattern correctly; otherwise Fuzzy ARTMAP committed a misclassification error.

4. IMPLEMENTATION ENVIRONMENT

These routines are implemented for DECmpp/Sx model 1208 computer which consists of a DEC RISC Workstation as a Front end and the MasPar MP-1 model 1208 Back end. Communication between the Front end and Back end computers is through DMA channels over a VME bus.

The Back end machine as described by MasPar[9] is a SIMD massively parallel machine consisting of 512 4x4 clusters of processor elements (PE) arranged in an 16 x 32 cluster array. This gives us 8192 PEs in a 128 x 64 PE array. To control and transfer data to and from the PE array the MP-1 has a RISC 32-bit processor with 32 32-bit registers, 128 K bytes of data memory and 1 megabyte of ram. Each PE consists of a 4- bit processor with 40 32-bit registers and 16 kilobytes of RAM. A key component of any implementation is the movement of data to and from the processor. In a single processor system this movement is implied. In a parallel processing environment it can be the driving contribution to the execution time. For the MP-1 there a two types of communications, a. communication between the ACU and the PEs, and b. communication between the PE's.

Communication between the ACU and the PEs are all broadcast, i.e., each PE receives the identical information at the same time. Incoming data is transferred from all active PEs simultaneously and its values are "logically reduced (global ORed)". Communication between PEs can either be transferred X-Net (a mesh type connection) or through the Global Router. X-Net communications are made in a straight line to any one of the 8 major compass directions with toroidal wrap around. Global Router communications are direct point to point simultaneous transfers between units. They are simultaneous from the viewpoint that all transfers will occur before the processors will execute the next instruction, however they take place over a

switched network that allows many transfers to occur in parallel but in reality the worst case condition can cause serial transfers. Although a very powerful mechanism, global routing may not be the best available choice[11]. One of the restrictions of the router mechanism is that only one input and one output communication can occur from any given 16 processor cluster at one time, thus the identification of the cluster element in the architecture. For more details on the MP-1 see references[9].

The programming environment for this implementation is MPL[10]. MPL is an extended version of ANSI C that has added the data type modifier plural that refers to any variable operated on or stored in the PEs. It has also added keywords to control the active set of processors and implement inter-processor communications. In MPL, control statements take on the additional implication of determining which processors are executing instructions at a given time. If the variables that form part of the control expression are all singular the active set is not modified. If any of them are plural the active set is dependent on value of the plural variable that is contained in the individual PE. A processor element either executes the sequence of instructions inside the block governed by the control statement or does nothing. As long as any processor is executing code within the control structure that code is executed. I.e., if a plural variable is used in a control structure such as if (plural var) {...} else {....} both sets of code will probably be executed.

To allow the implementations to be used for any size problem the routines were virtualized by treating the processor array as if it were 3 dimensional. It is made up of additional layers of the total processor array. The communication between layers was through the shared memory and a limited set of registers. Only the processors were layered not the PE memory.

In order for an algorithm to benefit from parallel processing it must exhibit a high degree of calculation independence. A neural network exhibits this characteristic in that product of each weight with the input can be calculated simultaneously and the sum can be calculated in any order. In addition when the inputs are vectors each element of the vector can be calculated simultaneously. The key purpose of using a parallel computer is to reduce the total amount of time required to perform the computation. The primary hardware independent measure of this gain is speedup which is defined as Total number of calculations divided by the total number of steps. A massively parallel computer (normally defined as a SIMD machine with 1024 processors or more or a MIMD machine with 64 processors or more) can conveniently be considered as having an infinite number of processors which allows a program design that takes advantage of maximum calculation independence, where number of physical processors is exceeded, their number can be extended by virtualization. While an infinite number processors allows a large number of calculations to be performed simultaneously, it does not address the data flow or communication of the results. In this area the design of the program becomes architecture dependent. A Single Instruction Multiple Data (SIMD) architecture machine executes that the same operation on all active data streams at the same time. While this reduces the number instruction decode units and provides a highly synchronized environment, it limits the processor independence. The primary advantage of a SIMD architecture is the simplicity of design. The Multiple Input Multiple Data (MIMD) architecture allows each processor to perform an independent instruction sequence on the data stream, however inter-process synchronization and communication become a bigger challenge. The DECmpp/Sx is MIMD from the viewpoint of Front end, Back end relationship, but SIMD for the massively parallel array. In most applications you use at least two program files, one with the main program written in ANSI-C which also call functions written in MPL from the second file. The ANSI-C routines execute on the Front end and the MPL routines execute on Back end. There is nothing that dictates where the main program runs as it can be either a “C” or “MPL” routine as it is simply the first entered and last exited, all the routines are linked together into one executable. Generally, the Front end is used for the main routine as it is the device that communicates with user and other general I/O and is more efficient at executing strict sequential actions such as reading character streams. This MIMD configuration can execute in either synchronous or asynchronous mode as primitives supporting both operations are provided.

5. PARALLEL IMPLEMENTATION

This implementation attempts to take full advantage of the parallel machine while also reducing the complexity of the operations (Integer vs. Floating Point) as a demonstration of the feasibility of special purpose highly parallel architectures for Fuzzy ARTMAP neural networks. The program is broken into two files fuzzmapfe.c and fuzzmapp.m which contain the Front end (FE) and Back end (BE) routines respectively. The FE routines perform all the I/O and pre and post scaling operations. They also perform the randomization of the data set between trials. The BE routines perform the training and testing of the neural networks. The BE is configured as two sets of vector processors. The first set handles all of the ART_a calculations simultaneously with the second set handling all of the ART_b calculations. Each set is further broken down into

N template vector processors where N is the number of committed templates + 1. Each template vector processor takes ceiling(M/2) physical processors where M is the dimension of input patterns. Thus each physical processor handles two elements of the input plus their complements times the number of virtual layers. The justification for this organization is that the communication between adjacent processors was faster than from the memory to a processor. However, operations from registers were the fastest. There are more total operations on a per template basis than on a per element basis so some compromise was desired to promote processor efficiency. Each additional virtual processor layer generates an additional set of execution steps. The division of the physical processors into ART_a and ART_b sets was made arbitrarily due to the configuration of the PE array. The last row was set aside for the ART_b processing. Although this reduces the number of processors available for ART_a processing since it is only 1.56% maximum loss, this was considered better than leaving 99% idle during separate ART_b processing and the simplification of the program gained by dedicating a total row was advantageous. If this ratio of ART_a to ART_b processors is inadequate the program will generate an error message to that extent and the program will have to be modified. Since, the program computes the maximum number of required ART_a patterns as being 1+ the number of training patterns (the result if $\rho_a = 1$) and expects the maximum number of ART_b templates to be an input parameter this error should not occur.

To demonstrate the potential to use simplified hardware (integer arithmetic) as well as the processing speed to be gained on the PE all of the values are pre and post scaled prior to submission to the PE array. All operations were performed using 32 bit integer arithmetic except solving for the ratio. In this case the numerator is scaled to 64 bits so the resulting division will yield a 32 bit quotient.

The FE processor initiates the program by reading in the parameters and the data set, that it scales and places into arrays for passing to the BE processor. The FE also calculates the dimensions for the PE array configuration. After all input and output values, both training and testing are in place the FE calls the BE main routine. The BE copies all the parameters and data to the PE array and ACU. If an additional randomly ordered run is required it initiates an asynchronous call to randomize routine on the FE which will execute concurrently with Training. It then calls the training routine which finishes configuring the PE array and then calculates the templates.

The templates are calculated on an iterative basis on a pattern by pattern basis until all patterns are presented with no further change to the template set. Each pattern is generated by reading the input element by element from the data array, each value is broadcast to each template processor where the element and its compliment are stored in the appropriate vector register. The vector element location is determined by a modulo operation which activates only the processors that act on the specified vector element. Note the compliment is recalculated for each pattern presentation since it faster than the reading and broadcast of an additional element. After the input pattern is generated the corresponding output pattern is likewise generated.

If it takes more than one layer of processors to handle all the templates this section is repeated for each layer. Simultaneously the fuzzy min operation is performed between the pattern and committed and first uncommitted template for both input and output patterns. Since each PE element provides processing for 2 elements and their compliments, this operation is repeated 4 times. The magnitude of the intersection is calculated first by summing the four elements in each processor then by pair wise summing the partial sums from each processor of the template vector. The pair wise summation takes $\log_2(M/2)$ steps to complete. This magnitude TW is saved for each template until the best template is selected. TW is then divided by the sum of beta and the old template magnitude TWO and saved as TR (template ratio) (this s a scaled value as discussed in the previous section). If TW is greater than $\rho \cdot M$ and TR is greater than T_{max} of the previous layer then TR and the virtual processor number are saved in T_{max} and num. After all template layers are processed, the T_{max} register holds the largest value across the layer. The template identified by the virtual processor number also meets the vigilance criteria but we still need to select T_{max} across the set of processors.

Selecting T_{max} across all processors is accomplished by a pair wise binary reduction process first between all processors of each row which leaves the T_{max} and num at processor 0 in each row and then the column reduction that leaves the answer at processor 0. Note the last row is not included in the final step since it is used to process ART_b only and its solution was complete after the row reduction. The temporary variable num in processor 0 for ART_a and processor Sb for ART_b contain pointers to the respective input and output templates. This pointer is used to fetch the mapping index for ART_a which is compared to the template number of ART_b. If the ART_a result pointed to the uncommitted template its mapping index is set to the ART_b template number and the template magnitude TW is saved as the old template magnitude TWO and the intersection elements are saved as the template elements.

If the mapping index does not agree with the ART_b template number then the T_{max} selection process must be repeated with a revised ρ. This is implemented by setting the $\rho \cdot M$=TW+ε where TW was the intersection weight of the template that yielded T_{max} and ε is the smallest possible value. Using the previously calculated values of TW and TR the selection process of T_{max} is continued resetting $\rho \cdot M$ as required until a suitable mapping index is obtained. Note that if two patterns are identical but map to different outputs this result would never converge. A test for this condition is included and if it occurs an error message is generated and the program is aborted. Once the correct template is found the intersection is recalculated since that was faster than saving all the intersections. This intersection replaces the previous elements of the template and the old template magnitude is replaced by the current template magnitude. This completes the operations for one pattern presentation.

After all patterns have been processed in the current iteration the value of the each template's old magnitude is compared to the saved magnitude. If all do not match, then the saved magnitude is updated to the old magnitude and another iteration is initiated. This approach is much faster than setting a flag each time a change occurs as it at most only takes L (the number of layers) steps per iteration. Otherwise, there could be up to N (the number of patterns) changes per iteration. Once all saved magnitudes match the old magnitudes the training is complete and the program returns to the parallel main routine.

After learning, the BE synchronizes its actions with the FE and passes the learning results to the FE. If output of the learned templates has requested the BE initiates an asynchronous process on the FE to scale, format, and output those templates. Concurrently the BE proceeds into test phase. The test phase is very similar to the learning phase except no templates are modified. The test patterns are presented the same as in the learning phase. However, after T_{max} is determined and a template is selected if the uncommitted template is the best match the unmatched counter is incremented. If the mapping index of ART_a does not agree with ART_b template number then mismatched counter is incremented. If the mapping index of ART_a does agree with ART_b template number then matched counter is incremented. After each testing pattern is presented one time the results are returned to the BE main routine.

The BE main routine synchronizes with the FE and transfers the results to the FE and initiates an asynchronous test report routine on the FE. If additional passes were requested based on a reordering of the data set the BE now transfers the updated data set from the FE then synchronizes with the FE and initiates another random shuffling of the data set concurrently with proceeding with another learn, report, test and report cycle. This continues until all requested passes are completed or the program times out. Since, it is common for larger jobs to be presented than can be accomplished during the preset time limits the program is designed to generate multiple output files and close them incrementally during the process. It also will accept an input parameter to tell it where to restart. The program can be restarted before the initiation of any arbitrary random pass. It does this by repeatedly executing the random shuffling routine until ready for the prescribed pass. It is assumed that since these passes are performed on the FE prior to activating BE they will not deleted from the maximum time limit.

6. EXPERIMENTAL RESULTS

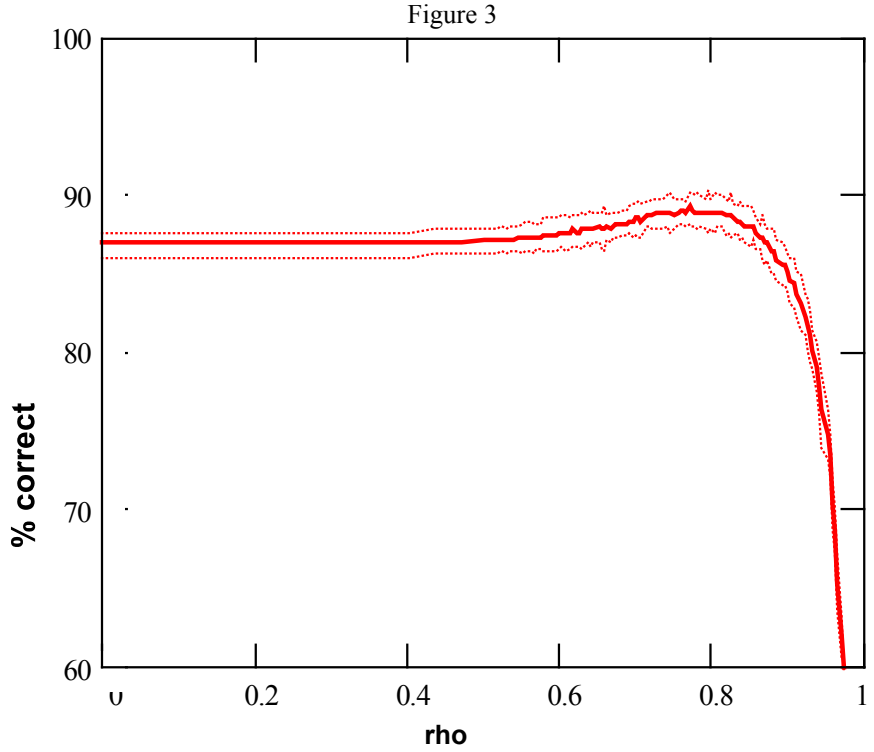
The experiments presented in this paper were performed on the Letters benchmark developed by Frey and Slate[8]. This benchmark consisted of a database of 20,000 patterns derived from 20,000 unique black-white pixel image. Sixteen numerical feature attributes were obtained from each character image, and each attribute value was scaled to range of 0 to 15. Each of patterns consists of the desired output character and the sixteen attribute values. The output characters were changed to an integer value of 1 through 26 to represent the alphabet from A to Z. The output was represented as one element with values from 1 to 26. A special FE process was created to deal with database allowing the input of the integer values rather than normalized values. This allowed a smaller database size than using a 6 digit decimal fraction representation. This FE process first normalized the input data prior to scaling data. The output used the reverse operation to allow the direct comparison to the input data. This experiment consisted of twenty separate runs of using the same order of the data set varying only the $\overline{\rho}_a$ value of the ART_a section. Values of $\overline{\rho}_a$ were selected to minimize the number of classification errors during the test run. The test was initially ran for $\overline{\rho}_a$ values of 0, .5, and 1 with further values selected at mid points of previous intervals until a local minimum was found. This data is presented in table 1. The data set was presented by using the first 16000 patterns to train the network and the last 4000 patterns to test the network..

TABLE 1 LETTERS DATABASE RESULTS						
TEMPLATES	ITERATIONS	$\overline{\rho}_a$	β_a	CORRECT	INCORRECT	UNKNOWN
786	6	0.00	0.01	3509	491	0
786	6	0.50	0.01	3509	491	0
791	5	0.571	0.01	3535	465	0
788	6	0.629	0.01	3538	462	0
865	5	0.692	0.01	3543	457	0
940	6	0.75	0.01	3559	441	0
913	5	0.754	0.01	3587	413	0
950	5	0.758	0.01	3572	428	0
959	5	0.763	0.01	3561	439	0
945	5	0.767	0.01	3575	425	0
987	5	0.771	0.01	3608	392	0
998	4	0.775	0.01	3575	425	0
1021	6	0.779	0.01	3581	419	0
1004	5	0.783	0.01	3578	422	0
1069	5	0.792	0.01	3581	419	0
1129	5	0.808	0.01	3561	439	0
1258	5	0.821	0.01	3540	460	0
1479	5	0.842	0.01	3530	469	1
1874	4	0.871	0.01	3476	524	0
4539	3	0.941	0.01	3124	529	347
15093	2	1.00	0.01	390	0	3610

7. CONCLUSIONS AND PLANS FOR FURTHER INVESTIGATION

The initial results generated more questions than answers. However, they provided some insight to guide further investigation. The data suggests the presence of multiple minimums suggesting that a more exhaustive study with multiple orderings is necessary before any conclusions are reached. It also demonstrates, that its sensitivity to $\overline{\rho}_a$, is stepwise rather than continuous. This is related to the fact that each element has only 16 possible values over the range 0-15. This allows an exhaustive study of $\overline{\rho}_a$ as it can be set to $\frac{M \cdot 15 - n}{M \cdot 15}$, $where\ n = 0..M \cdot 15.$ n an integer. As the number of templates only changes by 3 between 0 and .5 and number of iterations required to train remains constant, it suggests that the majority of the study concentrate on the range of .5 to 1.0. For that reason we studied 130 values of $\overline{\rho}_a$ with 121 of them from .5 to 1.0 selected each possible template weight in that range, and the rest chosen to illuminate the range between 0 and .5. For each of twenty different data sets I evaluated each of the 130 values of these values for a total of 2600 computer runs. We were also interested in observing the characteristics of the temporary values of ρ used during the resets, so we collected the template # and the value of $\overline{\rho}_a$ used for each of these occurrences over all of these runs. After this data is collected and analyzed, results and conclusions will be updated. Each run took between 3 to six minutes to complete with approximately 10 hours total per data set. Each data set was generated by splitting the data into half and exchanging each element of the first half with a randomly selected element of the second half. Successive data sets were generated by repeating this shuffling process. This exhaustive study with results presented in figures 3 and 4 showed that adjusting $\overline{\rho}_a$ can provide a template set with a higher probability of correctly identifying the input. This appears to occur between $\overline{\rho}_a$ values of .70 to .85 but it would take an exhaustive search within this range to select an optimum but is also dependent on the ordering of the data set as is indicated by the variations noted between runs. Figure 3 shows the mean and two sigma confidence region on the percentage of correct mappings versus $\overline{\rho}_a$ for the twenty different shuffles. Figure 4 shows the mean and two sigma confidence region of number of templates versus $\overline{\rho}_a$.

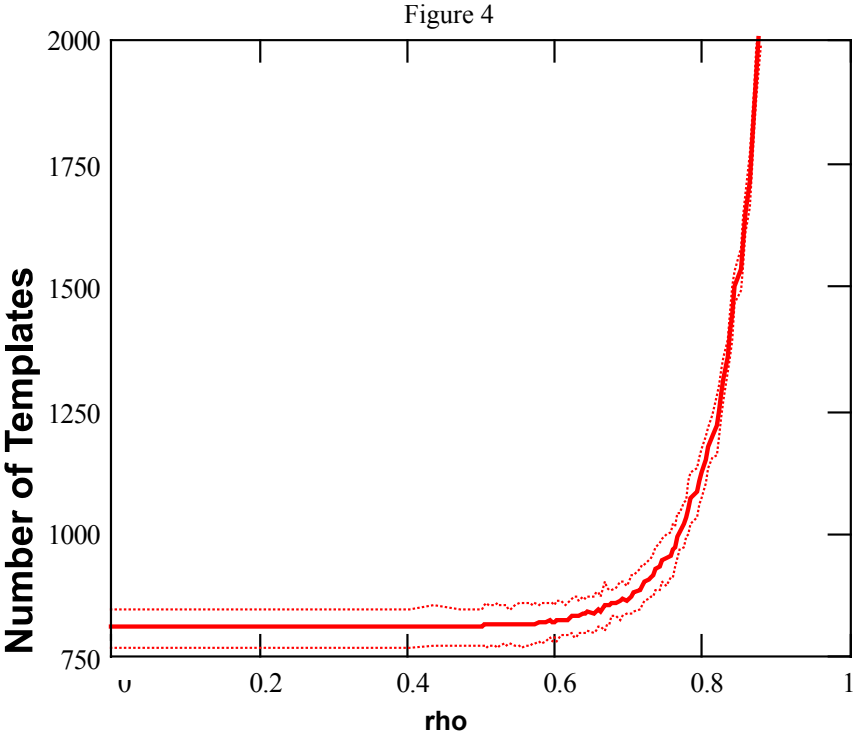
Figure 3



7.1 Computer architecture insight:

This implementation demonstrates that this type of neural network can take advantage of parallel processing. Furthermore since it is primarily based on comparison as its primary operation it can take advantage of simple processors. As the majority of its processing is the same it is also highly adaptable to the SIMD architecture. This also implies that a node or template has low computational, or hardware cost. From this viewpoint it appears that in reality that a one to one comparison between a Fuzzy ARTMAP node and a Back Propagation node is not totally correct. From a computational cost viewpoint one Back

Figure 4



Propagation node costs the same as N Fuzzy ARTMAP nodes, with N > 1. This combined with the fast learning characteristics, combined with the simple parallel architecture could make it a feasible component for adaptive control mechanisms or other intelligent applications where fast learning is required. As identified in table 2 for a $\overline{\rho}_a$ = .771 the number of templates required was 987. This set of templates generated correct answers within .05% of the peak found in the study yet used almost 200 templates less. This set would probably be chosen as optimum for the presented computer architecture since it used 987/1008 sets of processors or 98% of the available processors for each presentation of each pattern, yet it gave better performance than most larger sets of templates. No operations would be saved for a smaller set of templates unless they were less than 512.

Further investigation is planned in more precisely determining the amount of speedup that can be expected from the parallel processing approach. In addition more analysis will be performed to determine whether the range of exhaustive $\overline{\rho}_a$ study can be predicted by the size of templates generated by a run of $\overline{\rho}_a$ =0.

References

1. S. Grossberg. "Adaptive pattern recognition and universal recoding II: Feedback, expectation, olfaction, and illusions", *Biological Cybernetics*, 23, pp. 187-202. 1976

2. G. A. Carpenter, S. Grossberg, and D.B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system", *Neural Networks*, 4(6), pp. 759-771, 1991.

3. G. A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised Learning of analog multidimensional maps", IEEE Transactions on *Neural Networks*, 3(5), pp. 698-713, 1992.

4. A. Rao, M. R. Walker, L. T. Clark, and L. A. Akers, "Integrated circuit emulation of ART1 networks", *Proc. First IEEE Conf. On Artificial Neural Networks*, pp. 37-41, 1989.

5. S. W. Tsay, and R. W. Newcomb, "VLSI implementation of ART1 memories", *IEEE Trans on Neural Networks*, 2, pp. 214-221, 1991.

6. C. S. Ho, J. J. Liou, M. Georgiopoulos, G. L. Heileman, and C. Christodoulou, "Analog circuit design and implementation of an adaptive resonance theory (ART) neural network architecture", *International Journal of Electornics, 76, pp. 271-291*, 1994.

7. C. S. Ho, J. J. Liou, and M. Georgiopoulos, "Mixed Analog/Digital VLSI Implementation of ART1 Memories", *Journal of Microelectronic Systems Integration*, 2(1), pp. 23-40, 1994.

8. P. W. Frey and D. J. Slate. "Letter recognition using Holland-style adaptive classifiers", *Machine Learning*, 6, pp 161-182, 1991.

9. *MasPar System Overview*, MasPar Computer Corporation, Sunnyvale, CA, 1992.

10. *MasPar Parallel Application Language (MPL) Reference Manual, Software Version 3.2*, MasPar Computer Corporation, Sunnyvale, CA, 1993.

11. *MasPar Parallel Application Language (MPL) User Guide, Software Version 3.2*, MasPar Computer Corporation, Sunnyvale, CA, 1993.