13th International Workshop on
Cellular Nanoscale Networks and their Applications (CNNA)
August 29-31 Turin, Italy

Non-Boolean Associative Architectures Based on Nano-Oscillators

Steven P. Levitan[*1], Yan Fang[1], Denver H. Dash[2], Tadashi Shibata[5], Dmitri E. Nikonov[3], George I. Bourianoff[4]
[1]University of Pittsburgh, Pittsburgh, PA, USA
Intel Corporation, [2]Pittsburgh, PA, [3]Hillsboro, OR, [4]Austin, TX, USA
[5]University of Tokyo, Tokyo, Japan
levitan@pitt.edu

***Abstract*— Many of the proposed and emerging nano-scale technologies simply cannot compete with CMOS in terms of energy efficiency for performing Boolean operations. However, the potential for these technologies to perform useful non-Boolean computations remains an opportunity to be explored. In this talk we examine the use of the resonance of coupled nano-scale oscillators as a primitive computational operator for associative processing and develop the architectural structures that could enable such devices to be integrated into mainstream applications.**

I. Introduction and Motivation

Computing systems based on Boolean logic can be viewed as fabrics of combinational logic with periodic restoring storage. The transient state of logical operations is captured in the voltages (charges) on the CMOS capacitive structures and wires. Static state is captured in cross coupled, or active, feedback structures. It is well known that current computing systems based on CMOS are approaching the limits of integration for two reasons: device scaling and power dissipation. However, investigations of nanotechnology to replace the CMOS transistor as the workhorse of Boolean computing structures with a more energy efficient device have been largely unsuccessful.

This leads us to re-think the use of Boolean logic based structures and the corresponding methodology we use to define "state." In this work, rather than using the charge on a wire, we consider another definition of state. We will use the relative phase relationship among a group of loosely coupled non-linear oscillators as a representation of state. Using phase as the basis of state, our primitive computational operations can be based on direct interactions between the frequency and phase relationships of clusters of oscillators. Comparing phase and changing phase become our two operations, as opposed to logic and latching.

Therefore, we are developing computing information processing systems based on the use of nano-scale oscillators to perform spatio-temporal recognition tasks. We use the temporal coherence of non-linear oscillators to perform "associative processing" between sets of stored patterns and input patterns. The effective resonance between the input patterns and the intrinsic phase relationships of the non-linear oscillators become our basic computation paradigm. Thus, if we "let the physics do the computing", we can design systems where the behavior of dynamic "attractors" will allow us to rapidly convergence to a computational solution.

In the rest of this paper we first describe the use of coupled oscillators to perform matching operations; we then discuss the use of pattern matching, or association for the task of image recognition. Next, we describe our associative architecture for performing these operations and give some results on a face recognition task. Finally, we present our plan for a hybrid oscillator/CMOS implementation of this processor.

II. Background

A. Using Coupled Oscillators as Associative Processors

Given that we can develop coupled oscillator modules for computation, mapping phase-based operations into Boolean logic is inefficient; rather Hoppensteadt and others have shown that oscillators can be used to directly perform computation in terms of associative operations [1-2]. While the associative processing paradigm does not provide the flexibility of Boolean logic, it does provide a mechanism for a broad range of pattern matching and classification tasks that are intrinsically "data-parallel." Comparisons between an input pattern and all stored patterns can be done in parallel, providing very fast computation.

Using coupled non-linear oscillators as computational primitives to perform comparison operations has two unique advantages. First, the same devices that are storing the patterns are also performing the comparisons. This eliminates the need for separate memory and processing devices and forms a rudimentary single instruction, multiple data parallel processor, or cellular automata. Second, the nature of the pattern matching operation provides a more robust comparison than that provided by traditional Boolean associative processors (or content addressable memories) based on exclusive-OR operations. In those processors, one typically computes the Hamming distance, or number of bit-mismatches, between input patterns and stored patterns.

This work was partially supported by the Intel Labs University Research Office.

Hamming distance gives a very crude measure of match, in the case of "near misses." Rather, it is generally more useful to use a higher level norm (such as Euclidian distance) as a measure of degree of match. The synchronization of coupled oscillators as an attractor basin can provide such a higher level norm, giving us a more robust pattern match computation.

Figure 1 shows 2D projections of four distance measures: Hamming, linear, Euclidian, and some possible attractor basins of a set of coupled oscillators. We can see that a "near miss" in different cases will be evaluated with different resolution results. For the Euclidian and oscillator curves, near-miss values will be "pulled in" to a match. Perhaps most important is the fact that the oscillator's characteristic attractor basin will give us a degree of match that spans all of the dimensions of a multi-dimensional input vector without the need for any algebraic calculations to be done in CMOS support circuitry. Specifically, if we look at a *k* element vector comparison to calculate the Euclidean distance we need to compute squares, sum, and take a square root of the *k* sub-comparisons. If we use the oscillators with each of the *k* elements one per oscillator, then the multidimensional attractor basin will do the equivalent computation "in the physics."

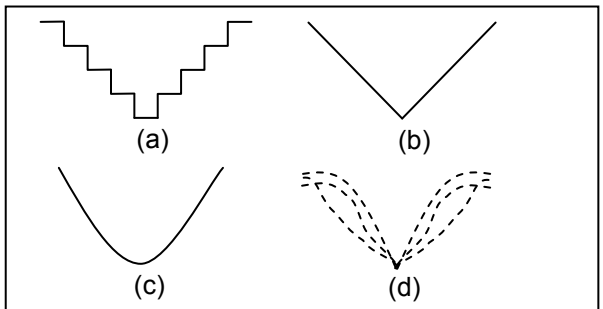

Figure 1. 2D representation of distance metrics for various possible pattern matching methods: (a) Hamming distance; (b) absolute value of difference; (c) Square of distance; and (d) models of oscillator convergence.

III. An Image Recognition Architecture

A. Associative Memory

Associative processing takes one of two forms: auto-association and hetero-association [3]. Auto-association processors are configured to store a set of patterns (vectors) and are required to be able to retrieve the original specific pattern when presented with the distorted or noisy version. Hetero-associative memory is the same except its output patterns are coded differently from the input patterns. Therefore, we can consider that a mapping between input and output patterns is stored in the network for pattern retrieval. In the architecture we are developing, the basic function unit is a hetero associative model where the output is the index of the most likely pattern based on the input pattern.

B. N-Tree Hierarchical Architecture

Assume there is a pattern set with *m* patterns, $\{p_1, p_2, \cdots, p_m\}$, required to be memorized by an associative memory system. When the number of patterns, *m,* is very large, it becomes difficult for a single associative memory (AM) unit based on oscillators to hold all the patterns. This is due to both fabrication and interconnect symmetry constraints. In order for the oscillators to work correctly as a pattern matching circuit, the oscillators need be matched in performance and the interconnect needs to be symmetrical. Both of these goals can only be reliably achieved if we keep the clusters of oscillators relatively small. Therefore, as a solution to this problem, we use a hierarchical AM model that organizes multiple AM units into a tree structure.

If each AM unit can only store *n* patterns, while $\mathrm{m} > n$, then the *m* patterns can be clustered hierarchically into an *n*-tree structure and stored in an AM model with the corresponding architecture. In this tree structure, every node is an AM unit and has at most *n* children nodes. Only leaf nodes (nodes that have no children) store specific patterns from the original input set: $\{\mathrm{p}_1, \mathrm{p}_2, \cdots, \mathrm{p}_\mathrm{m}\}$. The higher nodes are required to memorize the information associated with different levels of pattern clusters. Thus, every search operation, to retrieve one pattern, results in a path in the tree, from the root to a leaf. During the retrieval process, the key pattern is input recursively into the AM nodes at different levels of tree and finally the pattern with the highest degree of match is output.

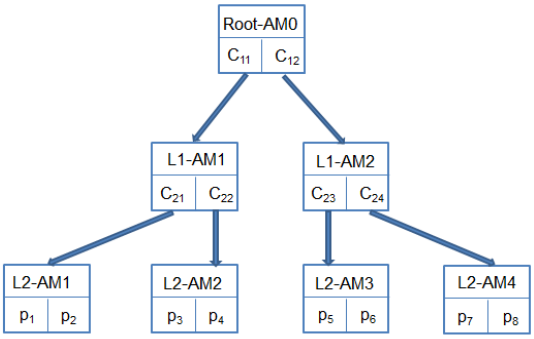

Figure 2. Binary tree hierarchical AM model

For example, if we take the simple case where, $\mathrm{m} = 8, \mathrm{n} = 2$, this case forms a binary tree AM model shown in Figure 2. At the root node, patterns are clustered into two groups, $\{p_1, p_2, p_3, p_4\}$ and $\{p_5, p_6, p_7, p_8\}$ respectively, associated with two representative patterns: C_{11} and C_{12}. There are several ways that the *C* patterns can be generated during training, as explained below. From the second level, these two groups of patterns split again into four groups and are stored in four AM units in the third level. In effect, each AM node has an associated pattern, *C*, stored in its parent node which represents the patterns in its local group.

C. Training

Once the hierarchical structure of AM model is fixed, the next question is how to organize the patterns into a tree structure and which features can be used for the recognition at each node? These questions determine the method of training and pattern retrieval. Notice the fact that the AM unit always

outputs the pattern with the highest degree of match, namely the one nearest to the input pattern in the data space. For this work, we use a hierarchical k-means clustering algorithm.

K-means clustering [4] is a method of cluster analysis which aims to partition *n* patterns into *k* clusters in which each observation belongs to the cluster with the nearest mean. Since these are multi-dimensional means we also refer to them as "centroids."

The steps of the algorithm can be summarized as:

1. Randomly select k data points as initial means (centroids)
2. Assign each data point to its closest mean (square Euclidian distance) and thus form k clusters
3. Recalculate current means of each cluster based on the data points assigned
4. Repeat (2) (3) until there are no changes in the means

Hierarchical k-means clustering is a "top-down" divisive clustering strategy. All patterns start in one cluster and are divided recursively into clusters by clustering subsets within the current cluster as one moves down the hierarchy. The clustering process ends when all the current subsets have less than k elements, and they are non-dividable. This algorithm generates a k-tree structure that properly matches our AM structure. The internal nodes are centroids of each cluster in different levels and the external nodes (leaf nodes) are exact patterns. During the clustering process, some clusters may become non-dividable earlier than others and have higher positions in the tree than the other leaf nodes.

Consider the previous example, the first clustering generates two clusters, $\{p_1, p_2, p_3, p_4\}$ and $\{p_5, p_6, p_7, p_8\}$ with centroids C_{11} and C_{12}. Then these two clusters split to four sub-clusters, $\{p_1, p_2\}$, $\{p_3, p_4\}$, $\{p_5, p_6\}$ and $\{p_7, p_8\}$. Their centroids are C_{21}, C_{22}, C_{23} and C_{24}. Since the number of patterns in one cluster is less than the threshold, we set (n=2), the clustering process ends. After the centroids and patterns are written into each AM node correctly, the training process is finished.

D. Recognition

The pattern retrieval process starts from the root node. When we input the pattern that needs to be recognized, the AM unit at the root node will output the nearest centroid as the winner. This centroid will lead to a corresponding node in the next level and the system will repeat the recognition process until the current node is a leaf node. The final output is a stored pattern. Considering the same example as above, assume we have a test pattern p_t, which is closest to p_5. The retrieval process is:

1. Input p_t to the root node AM0, output pattern is C12, point to the AM2 of level 1, with cluster $\{p_5, p_6, p_7, p_8\}$
2. Input p_t to L1-AM2, output pattern is C23, point to the AM3 of level 2, with cluster $\{p_5, p_6\}$
3. Input p_t to L2-AM3, final output pattern is p_5

IV. Experiments and Simulation

A. Test Data

We implemented an N-Tree Hierarchical AM model for an image recognition task. The data set is from the ATT Cambridge Image Database [5]. This data set contains 400 grayscale images of human faces, as shown in Figure 3. These images are photos of 40 people with 10 views for each person. The original images have 92x112 pixels with 256 gray levels. We convert the image size into 32x32 and average the intensity of all the images by the following method:

1. Calculate the means of all pixels for each image, $(x_1, x_2, \cdots, x_{400})$
2. Calculate the mean of $(x_1, x_2, \cdots, x_{400})$, $\mathrm{M} = \frac{\sum_{i=1}^{400} x_i}{400}$
3. For ith (1<=i<=400) image, every pixel is multiplied by the factor $\alpha = \frac{M}{x_i}$

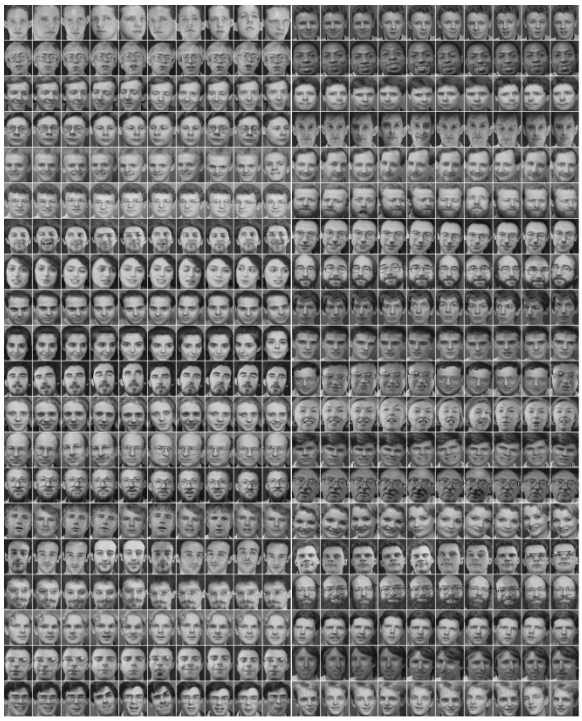

Figure 3. ATT image database, 10 images of each of 40 people

B. System Specification

In this case we define the size of our AM unit as 16x1024. One AM unit can store 16 images with 1024 (32x32) pixels. Thus the nodes of the tree structure can have at most 16 child nodes, instead of two. The definition of the algorithms in terms of both the training and recognition process remain unchanged.

V. Performance

A. Classification Test

In this task, the architecture is required to recognize the person on an input image after training. We use a hierarchical version of the ARENA algorithm of [6], which performs

nearest-neighbors in a reduced-resolution face image. For evaluation, first, we split the data set into two parts, a training data set and a test data set. For each subject, we pick 9 images for training and 1 for test. Then there are 360 images in the training data set and 40 images in the test data set. We use the images in the training data set to train our model.

Figure 4 shows a partial snapshot of the N-Tree after it has been trained. We can see 16 images at the root node. In fact these are not images from the input set, but rather the centroid "average images" that represent the 16 subtrees under the root node. Shown at node 2(5) is another set of centroid images for a lower level of the tree and at node 3(58) two final images that are from the original data. Also shown is node 2(14) which is a terminal node as well, with eight images. This occurs since the clustering operations create a tree which is not perfectly balanced.

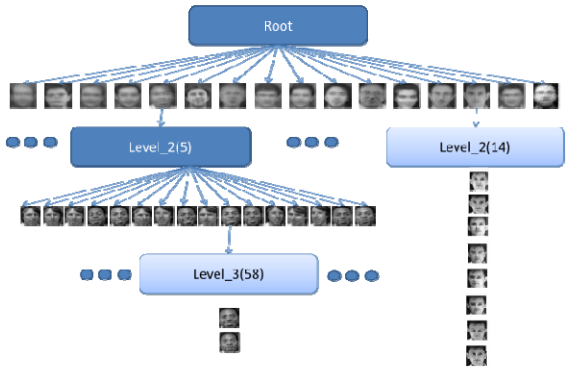

Figure 4. Snapshot of image database in the N-Tree

After the model has memorized all the images, we input images of the test data set sequentially. If the output image and the input image belong to the same subject (from different views) we say the recognition is successful; otherwise we call it a failure. The "hit rate" of this simulation is computed based on the 40 test results, one per subject. In order to evaluate the system's performance more accurately, we use a cross-validation technique [7]. For each run, we randomly pick images for training and test of each subject such that the model is trained and tested on different data sets for every run. The simulation includes 500 runs, and the distribution of hit rates (accuracy) is shown in Figure 5. For the 500 runs, the mean hit rate was 93.76%, with a variance of 0.12%. On average two or three misses happen in the 40 recognition cases.

For comparison, between our hierarchical method and a direct implementation of ARENA, we can assume we have a single large AM unit that can hold all 360 images and can classify each input image by directly comparing the degree of match to every memorized image. This model represents the best performance possible using a Euclidean distance to classify the face image data. We did the same simulation on this "flat" model for comparison. The results are shown in Figure 6. For the 500 runs, the mean hit rate was 97.05%, with a variance of 0.0063%. On average there was only one miss in the 40 recognition cases.

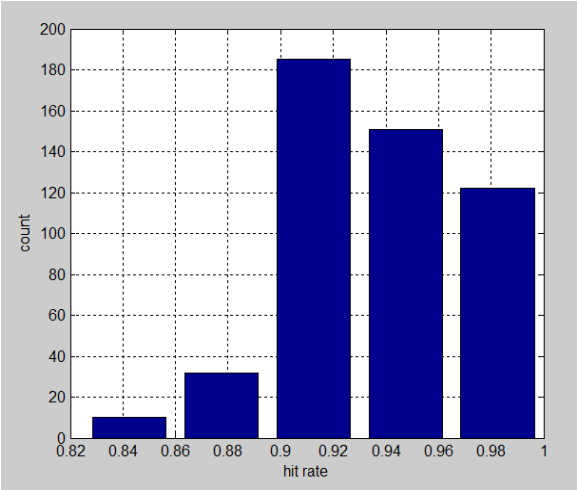

Figure 5. Hit rate distribution of hierarchical model

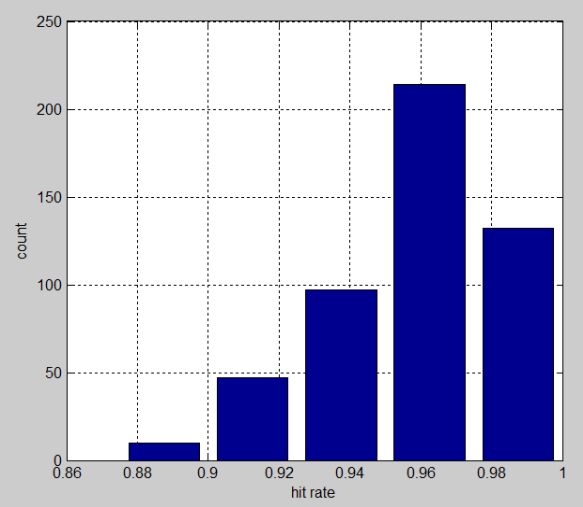

Figure 6. Hit rate distribution of single AM unit model

Compared with this model, our model is very close to the best performance with a higher efficiency. For each case, the large AM unit model has to compute the degree of match 360 times, while the N-Tree hierarchical model with a 3-level depth performs at most $3 \times 16 = 48$ operations of comparison.

VI. Design and Simulation of a Hybrid Nano-Oscillator/CMOS Processor

A. Design Constraints

We have designed a digital simulation of the N-Tree architecture to explore design tradeoffs in terms of the capabilities of the oscillator arrays and the digital CMOS support circuitry necessary to implement a complete system. There are several technologies that could be used to implement the non-linear oscillators including analog CMOS [8], resonance body transistors [9], and magnetic spin torque oscillators [10]. For the digital simulation we simply assume that there will be a cluster of oscillators and that they will be able to exhibit the comparison operations as attractor functions.

Figure 7 shows this abstract view of the associative module (AM). Here a set of *f* associative clusters (or words) each of size (or width) *k* constitutes one module. Each of these clusters will perform the comparison operation and the results will be converted into digital signals to be used by the control circuitry to enable successive layers of the N-Tree for the complete search operation.

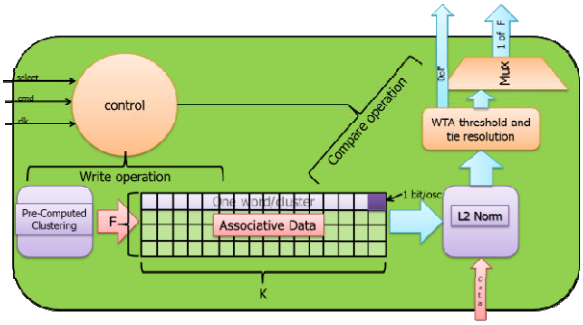

Figure 7. Abstract view of associative module

Figure 8 shows one possible implementation using CMOS ring oscillators. Each ring oscillator is coupled to *k* others to perform the multi-variable match. The output of each match is rectified, integrated, and compared with *f* other oscillator clusters, giving a winner-take-all result.

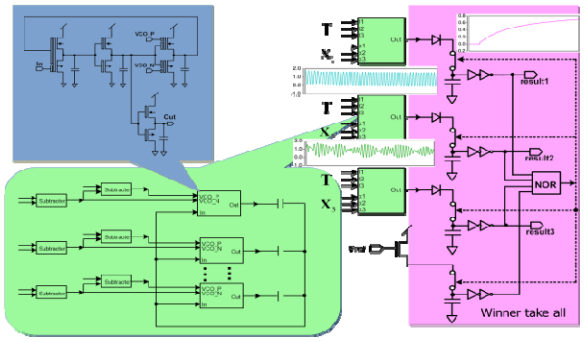

Figure 8. CMOS based associative oscillator module with winner-takes-all circuit

B. *Associative Module*

Irrespective of the implementation of the clusters we can abstract their behavior into the model shown in Figure 9. Here we show the *f* groups of *k* oscillators with their associated control circuitry. Based on our preliminary studies, we believe that both the size of the clusters *k* and the number of clusters *f* that can be resolved with a single analog winner-take-all circuit will be limited to values that are significantly less than the scale necessary to solve "interesting" problems. Therefore, we need to provide CMOS circuitry to collect the results of groups of *j* processing modules for each processing node of the N-Tree.

C. *Processing Nodes*

Shown in Figure 10 is a processing node corresponding to the nodes in Figure 2. Each processing node is composed of *j* associative modules and can perform the complete comparison between the full input vector and a set of *f* stored patterns. The

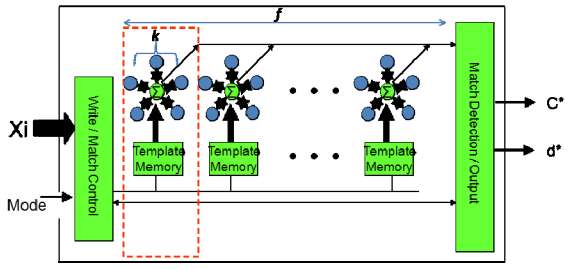

Figure 9. Processing Module

key component of the processing node is the "collaboration circuit" which performs parallel addition of each of the partial match results from each module. For *j* modules we need a *j*-way adder tree for each of the *f* values coming from each module. After the adder trees complete their work, a second circuit will compare the results and generate a 1-of-*f* result representing the index of the winner. As discussed above, the stored patterns are either centroids representing the subtrees of the data structure or particular patterns in the database. If this is a non-terminal node, then the result is used to select the appropriate child node. If this is a terminal node, then it holds the index of the final pattern. It is these processing nodes which make up the N-Tree itself.

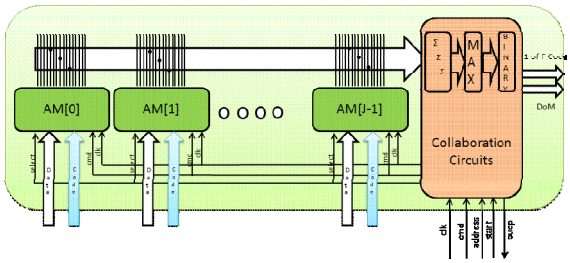

Figure 10. Processing Node

D. *N-Tree Architecture*

Figure 11 shows the full N-Tree architecture composed of the tree of processing nodes and the control circuitry. The control circuitry reads in both the operations (program) and the data to execute the image search application. It also provides the interconnect structure (both data and control) to build the hierarchical tree. Finally, the control circuitry provides the timing, handshaking, and select/enable control needed to first load the image database and then to perform the search operations by cascading the results of each comparison in the tree to the selected node in the next level of the tree, until the terminal nodes are reached with the final matching pattern identified.

E. *Simulation*

We have developed a digital simulator written in SystemC [11] of our N-Tree implementation. In this version we have modeled the N-Tree with the parameters shown in Table 1.

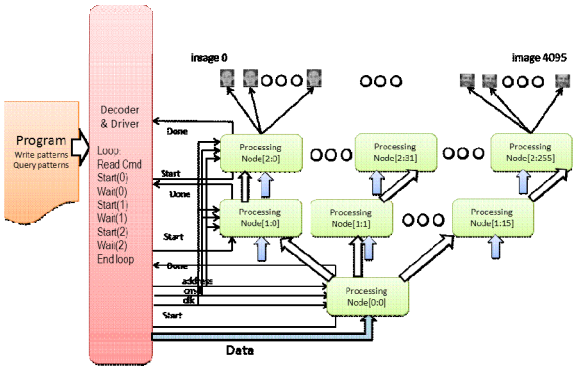

Figure 11. N-Tree Architecture

TABLE I. ARCHITECTURAL SIMULATION PARAMETERS

Input vector size	1024 (32x32 values of 8 bit gray scale images)
Oscillators in a cluster	16
Clusters in a module (controls fanout in tree)	16
Modules in a processing node	64
Database size	360 images (maximum 4096)
Tree size	3 levels with fanout of 16 at each level
Clock speed	5ns
Oscillator lock in time (analog comparison)	5-10ns
CMOS computation and propagation delay	20ns
Instruction cycle time	25-30ns

Figure 12 shows a snapshot of the output of the simulator. While the detailed timing of the digital simulation is not clear in the picture, we can see the process of writing data to each of the processing nodes at the first and second level of the tree. This is followed by 40 search operations, shown on the right section of the waveforms. Each "pulse" in the lower right is the enable signal from one processing node at the second level of the tree which is used as a "search enable" signal for the nodes at the third level of the tree.

VII. SUMMARY AND CONCLUSIONS

We have shown the design of an associative memory based on non-Boolean operations performed by non-linear oscillators. For any feasible realization of the memory based on nano-scale components, we assume that the associative clusters must be small relative to the scale of the problems of interest. Therefore, we have designed our memory as a partitioned hierarchical tree with a fixed fanout. We have shown that this design can perform with almost identical performance to a monolithic "flat" associative memory for a class of image recognition tasks. The use of a partitioned architecture for searching has another major advantage in terms of power. Only a small fraction of the processing modules need to be active for any given search. Using CMOS circuitry for control and interconnect, the architecture can be scaled to very large sizes.

REFERENCES

[1] Hoppensteadt, F. C.; and Izhikevich, E. M.; 1999 Phys. Rev. Lett. 82 2983.

[2] Hölzel, R.W. and K Krischer, K; New Journal of Physics 13 (2011) 073031.

[3] Hakin, S. *Neural Networks: A Comprehensive Foundation*, Prentice Hall, New Jersey, 1999.

[4] Duda, R. O.; Hart, P. E.; and Stork, D. G. *Pattern Classification*; John Wiley & Sons: 2000.

[5] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[6] T. Sim, R. Sukthankar, M. Mullin, S. Baluja. "Memory-based face recognition for visitor identification", Proceedings of International Conference on Automatic Face and Gesture Recognition, 2000.

[7] Devijver, Pierre A.; and Kittler, Josef; *Pattern Recognition: A Statistical Approach*, Prentice-Hall, London, GB, 1982.

[8] Asai, T. et al; Int. J. Unconventional Computing, 1 123-147.

[9] Weinstein, D. and Bhave, S.A.; Nano Lett., 2010, 10 (4), 1234–1237.

[10] Shehzaad Kaka, S. et al; Nature 437, 389-392.

[11] Black, D.C. and Donovan, J.; *SystemC: From the Ground Up*; Kluwer, Boston; 2004.

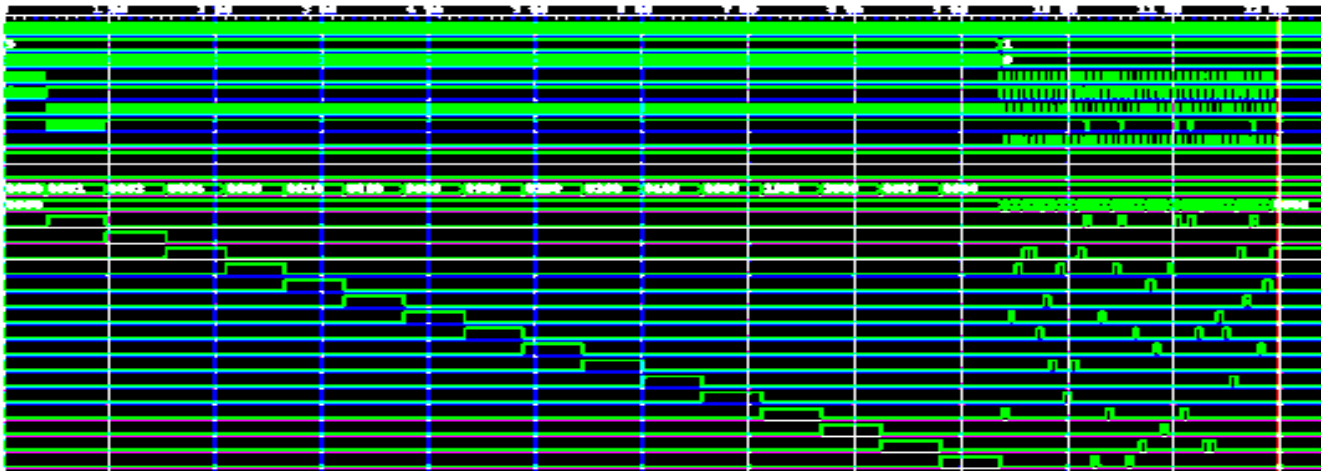

Figure 12. Digital Simulation of N-Tree Architecture