Architecture and Implementation of an Associative Memory Using Sparse Clustered Networks

Hooman Jarollahi, Naoya Onizawa, Vincent Gripon, and Warren J. Gross
Department of Electrical and Computer Engineering
McGill University, Montreal, Quebec H3A 2A7
Email: hooman.jarollahi at mail.mcgill.ca

***Abstract*—Associative memories are alternatives to indexed memories that when implemented in hardware can benefit many applications such as data mining. The classical neural network based methodology is impractical to implement since in order to increase the size of the memory, the number of information bits stored per memory bit (efficiency) approaches zero. In addition, the length of a message to be stored and retrieved needs to be the same size as the number of nodes in the network causing the total number of messages the network is capable of storing (diversity) to be limited. Recently, a novel algorithm based on sparse clustered neural networks has been proposed that achieves nearly optimal efficiency and large diversity. In this paper, a proof-of-concept hardware implementation of these networks is presented. The limitations and possible future research areas are discussed.**

I. Introduction

In associative memories, previously stored (trained) contents are accessed by linking other contents in the memory instead of using an explicit address as in indexed memories. They are therefore suitable for computational algorithms such as data mining and set implementations. When implemented in hardware, such memories could significantly improve both the speed and the efficiency of some applications such as retrieving data from part of its content when compared with indexed memories.

Classical Hopfield Neural Networks (HNN) used to be the state-of-the-art associative memories in terms of efficiency (the number of information bits stored per memory bit) [1]. However, the efficiency of HNN approaches zero as the amount of information learned is increased [2]. Furthermore, the number of messages that the network is capable to store (diversity) is limited since the length of each message needs to be unnecessarily the same size of the network. Therefore, the implementation of HNN is impractical and cannot compete with conventional indexed-based memories. Consequently, there has been little interest in implementing HNN in the past few years.

Recently, a novel algorithm for associative memories using sparse clustered networks has been introduced by [2], [3]. This methodology, we will call GBNN, provides a practical and nearly optimum efficiency and large diversity. As opposed to HNN, in GBNN both the neurons and the connection weights between the neurons are single bit binary numbers, with a much simpler functioning. In addition, the length of a learned message does not have to be the same as the size of network, leading to a much larger diversity. For a case study using 1.8×10^6 bits of physical memory, GBNN achieves an efficiency of 52% (which is 20 times the amount achieved in HNN) and a diversity 250 times larger than that of HNN.

In GBNN, the values of the neurons are computed concurrently. Using a software implementation with a limited number of processors will not allow full parallel capability and thereby will dramatically degrade the speed. This delay will grow at least linearly as the size of the network is increased. For example, retrieving a previously stored message of 105 bits in a GBNN containing 1Gb of data takes $\approx$ 80 ms using a 2.4 GHz Intel Core 2 Duo E6600. Thus, to benefit from the fully parallel capability of GBNN to reduce the delay, it is necessary to implement hardware solutions.

In this paper, we develop and demonstrate a proof-of-concept solution towards the hardware implementation of GBNN as associative memories. The algorithm is briefly explained from a hardware design perspective in Section II, and the hardware implementation is introduced in Section III. Section IV summarizes the results. Conclusions and future research are discussed in Section V.

II. Gripon-Berrou Neural Networks (GBNN)

Fig. 1 depicts the graphical representation of an example GBNN. The network consists of n neurons divided into c clusters each associated with an equally divided portion of a message to be learned or retrieved. Equal partitioning of the message is possible if and only if the clusters are of the same size [2]. The neurons in a cluster are also called *fanals*. The number of fanals is denoted by l. Only one fanal in each cluster is activated in the training process, and is desired to be activated in the message retrieval (decoding) process. In case of cluster or bit erasure in the input messages, more than one fanal may be temporarily activated in the message retrieval process, which will be corrected after an iterative decoding process in most cases. When a message is learned, single binary connection weights, w, between the activated fanals in all clusters are stored in a memory module to be later used in the retrieval process. All activated neurons will be connected constructing a *clique* of neurons. Fig. 1 shows a graphical representation of two learned cliques.

A. GBNN: Message Training

A message m of K bits is divided into c clusters each with l fanals. Therefore, the length of the sub-message associated

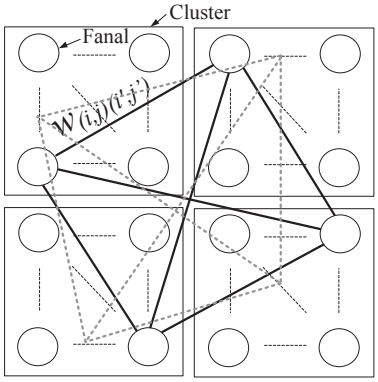

Fig. 1. Graphical representation of fanals, clusters and cliques.

with each cluster is $\kappa = K/c = log_2(l)$. In (1), $C(.)$ is a custom-designed function that maps a sub-message m_{c_i}, which is the part of the message associated with cluster i, to the index of fanal l_i to be activated. In this paper a simple function that maps binary values of κ bits to an integer number between 1 to l is considered. No two fanals in a cluster are allowed to be activated in the training process.

$$w_{(c_1,l_1)(c_2,l_2)} = \begin{cases} 1, & \text{if } \begin{cases} c_1 \neq c_2 \\ \text{and } \exists m \in \{m_1 ... m_M\} \\ C(m)_{c_1} = l_1 \text{ and } C(m)_{c_2} = l_2 \end{cases} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

B. GBNN: Message Retrieval

Once the network has been trained with the desired messages, i.e. the connection weights between fanals have been determined, and the cliques have been constructed, it is ready to retrieve messages given healthy or partially erased inputs. The retrieval process is initiated by performing the Local Decoding Type I (LDT-I), followed by an iterative process of Global Decoding (GD) and Local Decoding Type II (LDT-II). Local decoding deals with the determination of the index of the fanals to be activated in each cluster given a message. This is performed by using the scalar product of a pre-designed codeword of size κ bits, g, with the input message I, followed by finding the maximum fanal value that would exceed a certain threshold value σ.

The designated fanal is activated according to (2), and (3).

$$v(n_j) = \sum_{i=1}^{\kappa} g_{i,j} I(i) \tag{2}$$

$$v(n_j) \leftarrow \begin{cases} 1, & \text{if } v(n_j) = v_{max} \\ & \text{and } v_{max} \geq \sigma \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

GD is performed iteratively on each neuron in all clusters according to (4) to determine which fanal should become or remain activated. GD is performed on the results from local decoding, and is followed by LDT-II as shown (3). If we denote $n_{i,j} (i \leq c, j \leq l)$ the j'th fanal in the i'th cluster, v denotes a function that returns the numerical value of the neuron. γ is necessary to achieve good performance by enabling a memory effect such that a learnt message is distinguished from the non-learnt ones [2]. GD followed by LDT-II will iteratively reveal the correct fanals to be activated in the network.

$$\forall i, j, v(n_{i,j}) \leftarrow \sum_{i'=1}^{c} \sum_{j'=1}^{l} w_{(i',j')(i,j)} v(n_{(i',j')}) + \gamma v(n_{(i,j)}) \tag{4}$$

LDT-II is performed after each global decoding step and according to (3).

TABLE I
SELECTED DESIGN PARAMETERS FOR GBNN.

No. of Neurons (n)	128
No. of Clusters (c)	8
No. of Fanals per Cluster (l)	16
Message Length (bits)	32
κ	4
Maximum Diversity (Upper bound) (M_{max})	224

III. HARDWARE IMPLEMENTATION

Classical HNN have been implemented in hardware for many different applications such as in [4], [5], [6]. In order to implement a hardware model for GBNN, it is essential to specify the design parameters such that various cases can be tested especially when erasures would be present in the input messages. Therefore, while having scalability in mind as design feature, a sufficiently large network has been selected by using simulations in software and then designed in hardware using the FPGA. Some of the design specifications are shown in Table I. The upper bound value of the diversity of the network is given by (5) [2].

$$M_{max} = \frac{(c-1)n^2}{2c^2 log_2(n/c)} \tag{5}$$

A. Top Level Design Hierarchy

A simplified high-level block diagram of the network's hardware architecture is depicted in Fig. 2. Input messages are $K = c.\kappa$ bits long each. Since the input messages to be retrieved consist of either antipodal values (-1 or +1) when no erasure bit exists, or '0' when a bit erased, 2 bits are required to represent each bit of the message. Therefore, +1, -1, and 0 will be represented by 01, 10 and 00 respectively. Thus, for the design specification listed in Table I, 64 bits ($2.\kappa.c$) are required for the input messages I, whereas half of this amount, i.e. 32 bits are sufficient for training messages. The entire design has been implemented using internal logic registers and look-up tables of the FPGA although it is also possible to use other types of memories than the flip-flops including off chip ones for larger networks with the cost of speed degradation.

B. Message Training

The amount of memory required to store the binary weights is given by

$$\text{Memory (bits)} = \frac{c(c-1)l^2}{2} \tag{6}$$

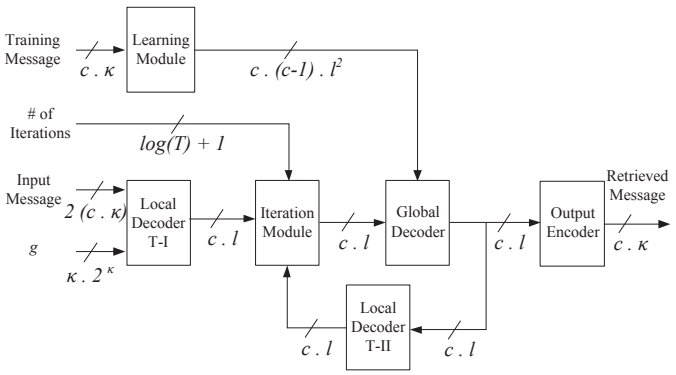

Fig. 2. Simplified block diagram of the top level hierarchy.

This memory accounts for the major part of the total memory usage of the network in the hardware implementation as also verified in Section IV for a sample network. As the size of the network is increased, this memory will dominate the silicon area when integrated with the dedicated logic portion.

In message retrieval, it is desired to have the same fanals that were previously learned to be activated after message decoding.

Hardware architecture of this part is depicted in Fig. 3. The training messages are to be stored in memory as binary weight values in a serial manner. A memory block made out of Flip-Flops (FF) with the indicated size (per cluster) is required to store these values where every FF can be accessed independently. An OR array is used to accumulate the connection weights when the inputs are serially transferred. The memory location to which the binary value is stored is determined by part of the message as shown with the vertical gray area beside the row decoder. The actual data values are stored through the OR array and using the rest of the message shown below the data decoder. The row and data decoders are one-hot decoders to decode the memory address and the actual data respectively. The OR-Mux block is a multiplexer that is used to assist the write process by reducing the necessity to OR all stored values with the new data only to the location where the new data is actually going to be stored.

C. Local Decoding Type I

The codewords, g, are given as inputs into the LDT-I. In this type of decoding $\sigma = \kappa$ is a good choice. Since the input messages and the codewords are antipodal, in order to implement (2), negative and positive bits of the input message need to separated, and the codeword g is applied to both of them individually using XOR gates. Then, the results for each group are added up using κ bit adders, and then subtracted from each other to perform full operation of the scalar product as shown in Fig. 4 (left). Once all the fanal values have been computed using the associated codewords, the Fanal activation module in Fig. 4 (right) will find the maximum value exceeding the threshold value, and activate the corresponding fanal accordingly while switching the rest of them off (3). The

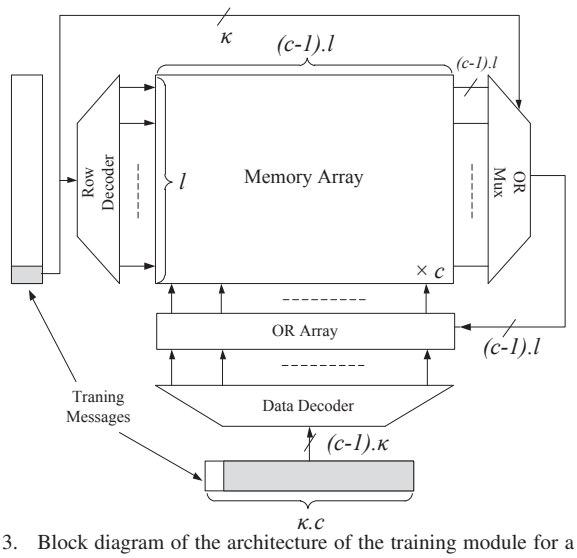

Fig. 3. Block diagram of the architecture of the training module for a single cluster.

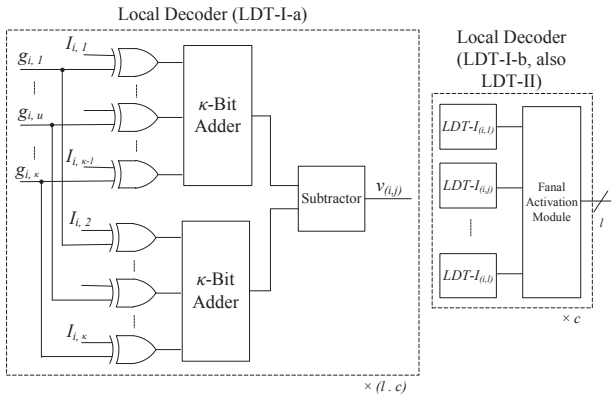

Fig. 4. Block diagram of the architecture of the local decoder Type I (left and right), and Type II (right).

maximum-function is implemented by using the compare-and-select methodology.

D. Local Decoding Type II

This type of local decoding is used in the iterative process after the global decoding as depicted in Fig. 2. The architecture corresponds to (3) which is the second part of LDT-I. The difference between this type of decoding and that of LDT-I is in the choice of the threshold value. In this case, $\sigma = 0$ is selected. Fig. 4 (right) shows this implementation as also discussed in the previous section.

E. Global Decoding

Fig. 5 depicts the hardware implementation of the global decoder. It consists of a Global Neural Computing Element (GNCE) that computes the integer values of the neurons including the memory effect (left), followed by a Global Cluster Computing Module (GCCM) that takes inputs from GNCE and outputs the binary fanal values (right). The architecture of GCCM is similar to that of LDT-II which is a max-function coupled with a threshold value comparator. Let us assume that the goal is to compute the binary value of neuron $n_{i,j}$, which

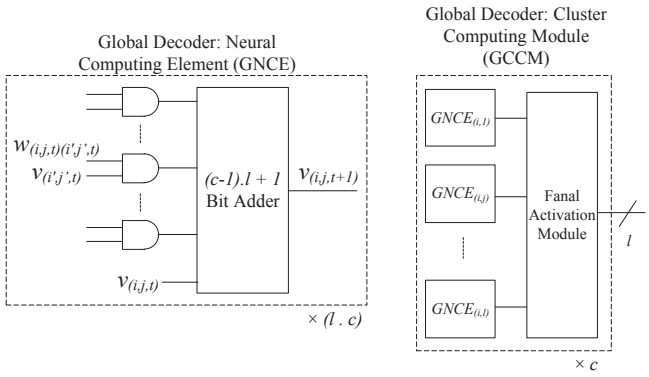

Fig. 5. Hardware implementation of the global decoder.

is the j'th fanal of the i'th cluster. GNCE has three types of inputs: (1) $w_{(i,j)(i',j')}$, the binary weights from the training module connecting all neurons excluding the i'th cluster to $n_{i,j}$, (2) $v_{(i',j')}$, the neuron's binary value in the clusters excluding the i'th cluster, and (3) $v_{(i,j)}$, the neuron's binary value of the computing fanal constructing the memory effect. t and $t+1$ denote the current and next iteration moments respectively. In the hardware implementation, the memory effect coefficient, γ, is considered to be equal to 1 (also discussed in [2]), which will also simplify the hardware design since all neuron values will be integers in all iterations.

IV. Results

The architecture described in this paper has been implemented using an Altera Stratix IV (EP4SGX230KF40C2) Field Programmable Gate Array (FPGA). Its functionality has been verified by matching the results with the software simulations using different iteration values. In Fig. 6, the results are demonstrated by varying the network density from 10% to 90% and measuring the message and bit error rates for 100 uniformly distributed trials for each density. Two scenarios are demonstrated, a 50% cluster erasure case where a portion of the message is erased, and a 10% bit erasure case where the location of the erased bits is based on a uniformly random distribution. The results look similar since the algorithm described in [2] is not suitable for the BE case. The maximum size of the network that can fit into the current FPGA with the same number of clusters and using only the internal logic is predicted to be 448 neurons using (6) and Table II. For this network, the ratio between the software simulation delay using an AMD Opteron 8387 (2.8 GHz) processor to that of the hardware implementation of this network using the FPGA for 100,000 trials is measured to be (10 sec / (100,000 × 5 clock cycles × clock period) ≈ 2000 when the number of iterations (it) is 4. Table II shows a summary of the resource allocation of the FPGA for the implementation of GBNN architecture.

V. Conclusion

In this paper, a fully-parallel hardware implementation of GBNN has been introduced. This architecture can be used in a variety of applications such as data mining, and can potentially be embedded inside processor chips to communicate with the

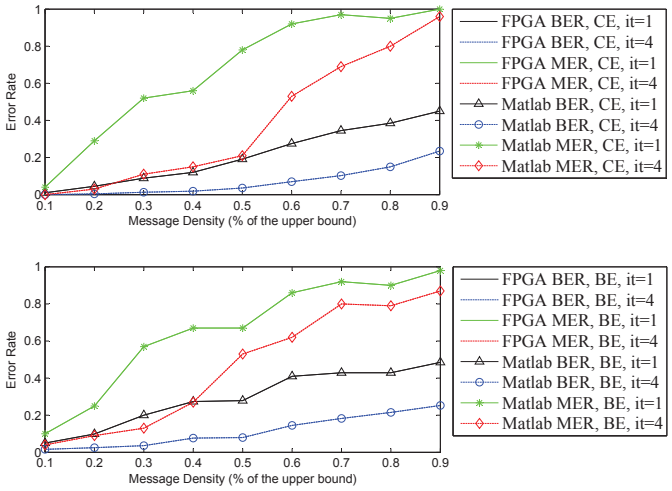

Fig. 6. MER and BER results for 50% Cluster Erasure (CE) vs. 10% Bit Erasure (BE) cases when varying the message density of the network.

TABLE II
FPGA results with design parameters shown in Table I.

Memory usage dedicated to w (bits)	14,336
Dedicated Logic Registers	$15,783/182,400(9\%)$
Combinational Look-up Tables (LUT)	$35,224/182,400(19\%)$
Total pins	$169/888(19\%)$
Slow 900mv 85C maximum frequency (Mhz)	107.15
Training, Retrieving delay (per message)	10 ns, 50 ns
Software to hardware delay ratio	≈ 2000

indexed memory units to achieve higher performance. Depending on the application, this architecture can be implemented on Application Specific Integrated Circuits (ASIC). The amount of memory required to store the binary connection weights, and also the number of interconnects are potential design limitation in large scales, similar to HNN. Future research will investigate partially-parallel architectures and efficient representation of sparse matrices for very large scaled applications. Also, the efficiency improvement of the maximum-function in local decoding can be investigated by using other techniques than the classical compare-and-select methodology. Improving the error rate results when bit-erasures exists instead of cluster erasures is possible and currently in research.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA*, 1982.

[2] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *Neural Networks, IEEE Transactions on*, vol. 22, no. 7, pp. 1087 –1096, july 2011.

[3] ——, "A simple and efficient way to store many messages using neural cliques," in *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*, april 2011, pp. 1 –5.

[4] R. Mandisodza, D. Luke, and P. Pochec, "VLSI implementation of a neural network classifier," in *Electrical and Computer Engineering, 1996. Canadian Conference on*, vol. 1, may 1996, pp. 178 –181 vol.1.

[5] I. Mihu, R. Brad, and M. Breazu, "Specifications and FPGA implementation of a systolic hopfield-type associative memory," in *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, vol. 1, 2001, pp. 228 –233 vol.1.

[6] S. Saif, H. Abbas, and S. Nassar, "An FPGA implementation of a competitive hopfield neural network for use in histogram equalization," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, 0-0 2006, pp. 2815 –2822.