CLUSTER-BASED ASSOCIATIVE MEMORIES BUILT FROM UNRELIABLE STORAGE

François Leduc-Primeau[⋆] *Vincent Gripon*[†] *Michael G. Rabbat*[⋆] *Warren J. Gross*[⋆]

[⋆] Dept. of Electrical & Computer Engineering, McGill University, Montréal, Canada
[†] Electronics Department, Télécom Bretagne, Brest, France

ABSTRACT

We consider associative memories based on clustered graphs that were recently introduced. These memories are almost optimal in terms of the amount of storage they require (efficiency), and allow retrieving messages with low complexity. We study an unreliable implementation of the memory and compare its error rate and storage efficiency with that of a reliable implementation. We present analytical and simulation results that indicate that the proposed memory structure can tolerate a large number of faults at a reasonable cost, thereby making it a good candidate for achieving highly efficient circuit implementations of associative memories.

1. INTRODUCTION

Associative memories are devices that are able to retrieve previously stored messages given part of their content. They are used in a variety of applications ranging from CPU caches [1] to database engines [2], and intrusion detection systems [3]. Recently, Gripon and Berrou [4, 5] proposed a novel architecture for associative memories that is almost optimal in terms of storage efficiency [6]. The number of messages it is possible to store and then retrieve with high probability grows quadratically with the number of vertices in the underlying graph. Moreover, the retrieval complexity is limited, even when the number of stored messages is large.

Several applications require dedicated hardware, and some circuit implementations of cluster-based associative memories have been proposed, e.g. [7]. In this paper, we consider the performance of this associative memory when implemented on unreliable hardware. Considering unreliable hardware is motivated by the fact that, as the feature size of integrated circuits decreases, it is becoming increasingly hard to control the variability associated both with the fabrication process and with the circuit's operating conditions [8, 9]. As a result, larger safety margins must be used to maintain yield and performance guarantees. Fault tolerance provided by the algorithm allows reducing these safety margins, thereby increasing the implementation's efficiency. However, it is important to characterize the cost of providing fault tolerance at the algorithm level in order to assess whether such an approach is more efficient than maintaining large safety margins.

An important performance metric for associative memories is their storage efficiency, which measures the amount of storage redundancy that is required to implement the associative retrieval mechanism. We first present analytical results for the retrieval performance, that is the probability of retrieving a message in terms of the number of messages stored in the memory. We then discuss the fault tolerance of the memory in terms of its storage efficiency, and show that it can be implemented with very faulty storage with only a small loss in efficiency.

2. CLUSTER-BASED ASSOCIATIVE MEMORIES

2.1. Message storage

Consider that we want to store a set $\mathcal{M}$ of messages, each composed of c symbols in the alphabet $\mathcal{A} = \{1, 2, 3, \dots, \ell\}$. We are interested in retrieving a message $m \in \mathcal{M}$ given a partial version of it, that is a copy of m where some of the symbols have been replaced by an erasure symbol $\perp$, $\perp \notin \mathcal{A}$. To represent the structure of the memory, let us consider an undirected unweighted graph made of $c \cdot \ell$ vertices. The graph is partitioned into c clusters, each containing the same number ℓ of vertices. Any vertex in the graph is uniquely identified given a pair (i, j), where $1 \le i \le c$ and $1 \le j \le \ell$.

To represent messages to store in the graph, we define a function f that projects a message or a partial message m onto a subset of vertices in the graph. We have $f(m) = \{v_{i,j} | m_i = j\}$, where $v_{i,j}$ is the vertex associated with (i, j) and m_i represents the i-th symbol of m. To store a message $m \in \mathcal{A}^c$ in the graph, we add all the edges between vertices in $f(m)$, with the exception of self-loops. Therefore, $f(m)$ becomes a clique in the graph. The graph being unweighted, edges that are already in the graph remain unchanged.

2.2. Message retrieval

A stored message can be retrieved when only a part of its symbols are projected onto the graph. The retrieval algorithm takes as input a partially erased message $\widetilde{m}$ that is obtained from a message $m \in \mathcal{M}$ by replacing any $c_e < c$ symbols with $\perp$. We call "erased cluster" a cluster corresponding to an erased symbol in $\widetilde{m}$. Note that $f(\widetilde{m})$ does not contain any vertex in erased clusters. The algorithm is iterative, and its state at iteration $t \in \mathbb{N}$ can be fully described by a set $V_{\widetilde{m}}^{(t)}$, whose initial value is given by $V_{\widetilde{m}}^{(0)} = f(\widetilde{m})$. A vertex $v \in V_{\widetilde{m}}^{(t)}$ will be referred to as an *active* vertex at iteration t.

Two data objects are needed to describe the retrieval algorithm: first the set $V_{\widetilde{m}}$ of active vertices, and second a representation of the graph edges. The edges can be represented using a set of $\ell \times \ell$ bi-adjacency matrices, each representing the edges between a given pair of clusters. We will denote the matrix representing the edges from cluster i to cluster j by $W_{i,j}$. We have that $W_{i,j} = {W_{j,i}}^{\mathsf{T}}$. Since the graph edges are unweighted, the elements of $W_{i,j}$ are binary. The number of memory bits required for storing $\{W_{i,j}\}$ is at most

$$Q = \ell^2 \binom{c}{2} = \frac{\ell^2}{2}(c^2 - c). \qquad (1)$$

To match the organization of the adjacency data, we choose to represent the vertex states separately for each cluster. We use a set of vectors $\{a_1, a_2, \ldots, a_c\}$, where each $a_i \in \{0,1\}^{\ell}$ represents the state (active or not) of the vertices in cluster i. An element of a_i is denoted $a_i[j]$ and defined as

$$a_i[j] = \begin{cases} 1 & \text{if } v_{i,j} \in V_{\widetilde{m}} \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

The retrieval process is described by Algorithm 1. The initial state vectors are obtained by using (2) with $V_{\widetilde{m}} = f(\widetilde{m})$, and upon completion, the algorithm outputs the updated state vectors. The pseudo-code uses the following notation: the zero vector is denoted $\underline{0}$, $w(x)$ denotes the Hamming weight of a binary vector x, and the operator $\otimes$ denotes a matrix product where addition is replaced by logical *or*, and multiplication by logical *and*.

In each iteration of the algorithm, we select the vertices that will remain active based on their *score*. We express the score achieved by all vertices in a cluster i as a vector s_i of length ℓ. As described on line 9, the score of a given vertex is incremented once for every cluster in which it has at least one neighboring active vertex. We then find the maximum score $n_{\max}$ achieved in a given cluster. A vertex remains activated if it achieves the maximum score in its cluster.

Algorithm 1 is guaranteed to converge to a fixed point [6], but the fixed point may be such that some clusters have more than one active vertex. Since, by definition, each message stored involves exactly one active vertex per cluster, this needs to be resolved. In most applications, we will be interested in a decoder that generates an estimate $\hat{m} \in \mathcal{A}^c$, as opposed to $\{\mathcal{A} \cup \{\perp\}\}^c$. If we assume that the messages are independent and identically distributed (i.i.d.) with a uniform distribution, it is optimal to select, for each cluster, an arbitrary symbol value from the set of active vertices. Therefore, we define an estimator $\hat{m} = h(\{a_1^{(t)}, \ldots, a_c^{(t)}\})$ such that for each i, $\hat{m}_i = j$, where j is an arbitrary index that satisfies $a_i^{(t)}[j] = 1$.

```
   input : $\{a_1^{(0)}, a_2^{(0)}, \ldots, a_c^{(0)}\}$
1  begin
2      $E \leftarrow \{i \mid w(a_i^{(0)}) \neq 1\}$
3      $t \leftarrow 0$
4      VALID $\leftarrow E = \emptyset$
5      while $t < L$ and not VALID do
6          for each $i \in E$ do
7              $s_i \leftarrow \underline{0}$
8              for each $k \in [1, c], k \neq i$ do
9                  $s_i \leftarrow s_i + a_k^{(t)} \otimes \widetilde{W}_{k,i}$
10             define $n_{\max} = \max_j s_i[j]$
11             for j from 1 to $\ell$ do
12                 if $s_i[j] = n_{\max}$ then $a_i^{(t+1)}[j] \leftarrow 1$
13                 else $a_i^{(t+1)}[j] \leftarrow 0$
14         $t \leftarrow t + 1$
15         if $\forall i, w(a_i^{(t)}) = 1$ then VALID $\leftarrow$ true
16     return $\{a_1^{(t)}, a_2^{(t)}, \ldots, a_c^{(t)}\}$
```

Algorithm 1: Message retrieval algorithm

3. DEVIATION MODEL

We now describe how we model the effect of circuit faults on the algorithm. A *fault* refers to the incorrect operation of a physical component, while *deviation* refers to a change in the algorithm's behavior as a result of a fault.

The matrices $\{W_{i,j}\}$ must be stored in a memory. In a hardware implementation, we should expect the storage associated with $\{W_{i,j}\}$ to represent a large part of the complexity. Indeed, it is straightforward to show that the amount of data that must be accessed from $\{W_{i,j}\}$ in order to perform one iteration of Alg. 1 is much less than the amount of stored data. Furthermore, very few operations must be performed on each bit of data, and we can therefore expect the processing circuit to be small compared to the memory. For this reason, in this short version of the paper, we only consider deviations occurring in the storage of $\{W_{i,j}\}$. The matrices with deviations are denoted by $\widetilde{W}_{i,j}$. For a reliable implementation of the algorithm, we define $\widetilde{W}_{i,j} = W_{i,j}$.

Integrated circuit memories such as SRAMs can suffer from both permanent and transient faults that cause incorrect values to appear at the output [10, 11]. In this short version of the paper, we only consider that the storage of $\{W_{i,j}\}$ is affected by permanent faults. The associative memory is used in two phases. First, we store the message set $\mathcal{M}$, which fixes $\{W_{i,j}\}$. We model the deviations by saying that $\{W_{i,j}\}$ is transmitted through Q parallel binary deviation channels with output $\{\widetilde{W}_{i,j}\}$, where Q is defined in (1). If we assume that deviations occur independently on each bit, the parallel channels are independent. Once $\mathcal{M}$ has been stored, and with $\{\widetilde{W}_{i,j}\}$ fixed, we perform a number of retrieval operations.

A simple model for permanent defects in bit cells, but which nonetheless covers more than 50% of SRAM defects [10], is to treat faulty cells as stuck at 0 or stuck at 1. One can then define a corresponding "stuck-at" deviation channel. Since faults are permanent, consecutive uses of a given binary channel in a given fabricated device are not independent. However, we analyze the performance for a given $\mathcal{M}$, which means that $\{\widetilde{W}_{i,j}\}$ is fixed. Therefore, each of the Q binary channels is only used once. If the stuck-at-0 and stuck-at-1 events are equiprobable and the channel is only used once, the transmission can be equivalently modeled using a binary symmetric channel (BSC) with cross-over probability ψ. If the Q bits of $\{W_{i,j}\}$ are transmitted through this BSC, we obtain

$$\widetilde{W}_{i,j} = \begin{cases} W_{i,j} + D \mod 2 & \text{if } i < j \\ \widetilde{W}_{j,i}^{\mathsf{T}} & \text{if } i > j, \end{cases} \tag{3}$$

where D is the indicator matrix of deviation events, that we define as a random matrix of size $\ell \times \ell$, where each element $d_{i,j}$ is an independent Bernoulli random variable with $\mathbb{P}(d_{i,j} = 1) = \psi$. For each (i, j) with $i < j$, we only store one of $W_{i,j}$ or $W_{j,i}$. Therefore, when considering faults, the identity $\widetilde{W}_{i,j} = \widetilde{W}_{j,i}^{\mathsf{T}}$ remains valid.

Note that $\{\widetilde{W}_{i,j}\}$ is only sampled once, and we are therefore interested in the performance for a particular realization of $\{\widetilde{W}_{i,j}\}$ (e.g. a specific chip), rather than the average performance. However, in simulations the performance of various realizations of $\{\widetilde{W}_{i,j}\}$ has been observed to remain very close to the mean, which motivates considering the expected performance over $\{\widetilde{W}_{i,j}\}$. This is the approach taken for the analysis in Section 4.

4. RETRIEVAL PERFORMANCE

We are interested in retrieving a previously stored message given a partially erased version $\widetilde{m}$. Let $P_m^{(t)}$ be the probability that the correct message is retrieved after t iterations. The retrieval algorithm is such that $P_m^{(t)}$ is non-decreasing, and therefore we can obtain a lower bound on $P_m^{(t)}$ by deriving an expression for $P_m^{(1)}$.

Throughout the analysis, we assume that the stored messages are i.i.d. uniform. Denote by $M = |\mathcal{M}|$ the number of messages to store. Note that during the storing process, each message to store adds – or does not modify if it already exists – one edge between every pair of clusters. Let us fix a pair of vertices from distinct clusters. The probability that a message obtained using i.i.d. uniform random variables adds the edge between them while being stored is trivially $1/\ell^2$. Thus, the probability that this edge is added to the graph after storing M i.i.d. uniform messages is[1]

$$d = 1 - \left(1 - \frac{1}{\ell^2}\right)^M . \tag{4}$$

[1]For simplicity, we assume that $\mathcal{M}$ may contain repeated messages.

The existence of edges is not independent since a message adds multiple correlated edges while being stored. However, we can show that they can be asymptotically regarded as independent [12], and make this approximation to simplify the analysis.

Let us now fix a message m in the set of stored messages, and erase $c_e > 0$ symbols uniformly at random to obtain a partial message $\widetilde{m}$. We will consider the probability of retrieving m correctly when the algorithm is used for a single iteration, and when the bi-adjacency matrices are affected by deviations. As mentioned in Section 3, we consider a typical realization of $\{\widetilde{W}_{i,j}\}$. Since $\widetilde{W}_{i,j} = \widetilde{W}_{j,i}^{\mathsf{T}}$, we can equivalently model the deviations in terms of the graph model. For each pair of vertices, if an edge exists, we remove it with probability ψ, and if none exists, we add one with probability ψ.

Let us first consider the probability $P_s^{(1)}$ that a single erased symbol is successfully retrieved after one iteration. From the definition of the estimator h at the end of Sect. 2, we have that for a reliable implementation, $P_s^{(1)} = \mathbb{E}[1/A]$, where A represents the number of active vertices in the cluster. However, because of the deviations on the edges, the correct vertex is not guaranteed to achieve the maximum score. Let n_{v_0} be the score achieved by the correct vertex v_0, and let F denote the event that at least one incorrect vertex in the cluster achieves a score higher than n_{v_0}. If F occurs, the probability of identifying the correct vertex is 0. Otherwise, an error could still occur if some incorrect vertices obtain a score of n_{v_0}. $P_s^{(1)}$ becomes

$$P_s^{(1)} = \mathbb{P}(F) \cdot 0 + \mathbb{P}(\neg F) \cdot \mathbb{E}\left[\frac{1}{A}\middle|\neg F\right] \tag{5}$$

where $\neg F$ denotes the negation of event F.

Let $c_k = c - c_e$. We first consider the probability that the correct vertex v_0 achieves a score of n_0, $n_0 \le c_k$:

$$\mathbb{P}(n_{v_0} = n_0) = \binom{c_k}{n_0}(1-\psi)^{n_0}\psi^{c_k - n_0}.$$

On the other hand, using the edge independence assumption, the probability that any other vertex is connected to the active vertex of a known cluster is $P_+ = \psi(1-d) + (1-\psi)d$. We can then express the probability mass function of the score of incorrect vertices as

$$\mathbb{P}(n_v = x) = \binom{c_k}{x} {P_+}^x (1-P_+)^{c_k - x}, 0 \le x \le c_k.$$

The scores of vertices are independent because of the edge independence assumption. The probability that the correct vertex is in the active set is then

$$\mathbb{P}(\neg F) = \sum_{n_0=0}^{c_k} \mathbb{P}(n_{v_0} = n_0) \left[\sum_{x=0}^{n_0} \mathbb{P}(n_v = x)\right]^{\ell - 1}.$$

Given $\neg F$, we have $A = A_s + 1$, where A_s is the number of vertices that are incorrect but nonetheless active. Therefore

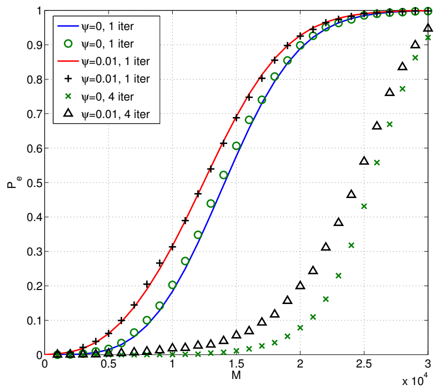

Fig. 1. Message error rate for 1 and 4 iterations as a function of the number M of stored messages, for $c = 8, \ell = 256, c_e = \frac{c}{2}$. The solid curves represent the analytical results.

$\mathbb{E}[1/A|\neg F] = \mathbb{E}[1/(A_s + 1)]$ is given by

$$\mathbb{E}\left[\frac{1}{A}\middle|\neg F\right] = \sum_{n_0=0}^{c_k} \mathbb{P}(n_{v_0} = n_0) \cdot$$

$$\left[\sum_{k=0}^{\ell-1} \frac{1}{k+1}\binom{\ell-1}{k}\mathbb{P}(n_v = n_0)^k \left(\sum_{x=0}^{n_0-1} \mathbb{P}(n_v = x)\right)^{\ell-1-k}\right]$$

Because edges are assumed independent and subsets of edges that target each erased symbol in the first iteration are disjoint, the probability that the complete message is retrieved successfully after the first iteration is simply $P_m^{(1)} = \left(P_s^{(1)}\right)^{c_e}$.

Figure 1 shows the message retrieval performance when messages are composed of 8 symbols of 8 bits each (i.e. $\ell = 256$), and half the symbols are erased. Simulation results for a single iteration confirm the analytical expressions. Even when the deviation probability is as high as 1%, the retrieval performance remains reasonably close to that of a reliable implementation. At $\psi = 10^{-3}$, the difference in performance with a reliable memory becomes negligible.

5. STORAGE EFFICIENCY

Storage efficiency compares the amount of information that can be reliably stored in the associative memory with the amount of storage required to represent it. For message-wise efficiency, we propose that the amount of information stored should be based on the expected number of messages that can be retrieved perfectly, and on the expected number of erased symbols that can be retrieved perfectly for symbol-wise efficiency. Message-wise reliable storage efficiency is given by $\eta_m^{(t)} = (c \cdot \log_2(\ell) \cdot P_m^{(t)} \cdot M)/Q$, and similarly for the symbol-wise efficiency $\eta_s^{(t)}$, replacing $P_m^{(t)}$ with $P_s^{(t)}$. It is interesting to consider the maximum efficiency that can be

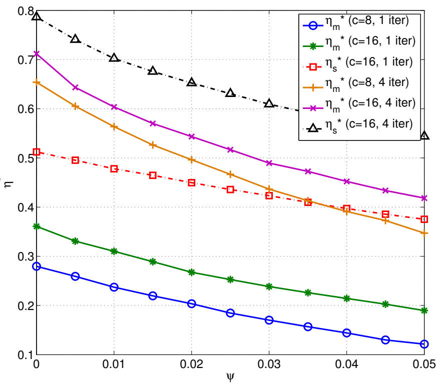

Fig. 2. η_m^* (solid curves) or η_s^* (dashed curves) as a function of ψ for 1 and 4 iter. For all curves, $c_e = \frac{c}{2}$ and $\ell = 256$.

achieved by the memory. For message-wise efficiency, we have

$$\eta_m^* = \max_M (\eta_m) = \frac{c \cdot \log_2(\ell)}{Q} \cdot \max_M (P_m M), \qquad (6)$$

where the iteration superscript is omitted to simplify the notation. The symbol-wise maximum efficiency is defined similarly.

For a single iteration of the algorithm, we can evaluate η_m^* by combining (5) and (6) and optimizing numerically. The efficiency can be greatly improved by using more than one decoding iteration. In that case, we evaluate η_m^* based on simulation results. Results for $c \in \{8, 16\}$, $\ell = 256$, and half the symbols erased are shown in Figure 2. As expected, the efficiency is higher if we do not require complete messages, but only consider the probability of retrieving an individual erased symbol. We see that large deviation rates can be tolerated at the cost of a small reduction in the maximum efficiency. For example, when ψ goes from 0 to 0.04, the memory with $c = 16$, $\ell = 256$ looses 27% of its efficiency when measured symbol-wise, or 36% when measured message-wise. On the other hand, for a 65nm CMOS process, tolerating an error rate of 4% in an SRAM can reduce energy consumption by close to one order of magnitude [13].

6. CONCLUSION

We studied the fault-tolerance of associative memories based on a clustered graph model [4]. We presented a detailed implementation strategy and a model describing how circuit faults can introduce *deviations* in the representation of the graph. We derived an analytical expression for the probability of correctly retrieving messages from the faulty memory, and showed that there is little degradation in the storage efficiency of the memory even when 1% of the adjacency storage is affected by faults. Our results therefore show that these associative memories can lend themselves to efficient circuit implementations with reduced safety margins.

7. REFERENCES

[1] N. P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers,” in *Proceedings of the 17th annual International Symposium on Computer Architecture*, 1990, pp. 364–373.

[2] C. S. Lin, D. C. P. Smith, and J. M. Smith, “The design of a rotating associative memory for relational database applications,” *ACM Trans. Database Syst.*, vol. 1, pp. 53–65, Mar. 1976.

[3] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, “Improving the accuracy of network intrusion detection systems under load using selective packet discarding,” in *Proceedings of the Third European Workshop on System Security*, ser. EUROSEC ’10, New York, NY, USA, 2010, pp. 15–21.

[4] V. Gripon and C. Berrou, “Sparse neural networks with large learning diversity,” *IEEE Trans. on Neural Networks*, vol. 22, no. 7, pp. 1087–1096, 2011.

[5] ——, “A simple and efficient way to store many messages using neural cliques,” in *Proceedings of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, Paris, France, April 2011, pp. 54–58.

[6] ——, “Nearly-optimal associative memories based on distributed constant weight codes,” in *Proceedings of Information Theory and Applications Workshop*, San Diego, CA, USA, February 2012, pp. 269–273.

[7] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, “Reduced-complexity binary-weight-coded associative memories,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 2523–2527.

[8] S. Ghosh and K. Roy, “Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era,” *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718 –1751, Oct. 2010.

[9] S. S. Sapatnekar, “Overcoming variations in nanometer-scale technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, March 2011.

[10] R. Dekker, F. Beenker, and L. Thijssen, “A realistic fault model and test algorithms for static random access memories,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 6, pp. 567–572, 1990.

[11] T. Karnik and P. Hazucha, “Characterization of soft errors caused by single event upsets in CMOS processes,” *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128 – 143, april-june 2004.

[12] F. Leduc-Primeau, V. Gripon, M. G. Rabbat, and W. J. Gross, “Fault-tolerant associative memories based on clustered graphs,” McGill University, Tech. Rep., 2013.

[13] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, “Near-threshold computing: Reclaiming moore’s law through energy efficient integrated circuits,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

8. ACKNOWLEDGEMENT

The authors would like to thank ReSMiQ for their support.