Fault Tolerant Memories Based on Expander Graphs

Shashi Kiran Chilappagari
Dept. of Electrical and Computer Eng.
University of Arizona
Tucson, AZ 85721, USA
Email: shashic@ece.arizona.edu

Bane Vasić
Dept. of ECE and Dept. of Mathematics
University of Arizona
Tucson, AZ 85721, USA
Email: vasic@ece.arizona.edu

***Abstract*— In this paper we consider memories built from components subject to transient faults. We propose a fault-tolerant memory architecture based on LDPC codes and show the existence of memories which can tolerate constant fraction of failures in all the components. Our proof relies on the expansion property of the underlying Tanner graph of the code. We illustrate our results with specific numerical examples.**

I. Introduction

During the past four decades, the decrease in transistor size and the increase in integration factor have lead to very small, fast, and power efficient chips. As the demand for power efficiency continues, a wide range of new nano-scale technologies is being actively investigated for processing and storage of digital data. Although it is difficult to discern which of these approaches will become a technological basis for computers in the future, it is widely recognized that due to their miniature size and variations in technological process, the nano-components will be inherently unreliable. Even in more traditional semiconductor technologies, reducing transistor size has already started affecting circuit reliability, and it is widely believed that transistor failures (both transient and permanent) will become one of the main technological obstacles as the trend of increasing the integration factor continues. In this paper, we consider storage circuits built from such unreliable (faulty) components. We consider an unreliable component (a logic gate or a memory element) to be a component that is subject to *transient faults*, i.e., faults that manifest themselves at particular time steps but do not necessarily persist for later times [1].

Von Neumann [2] was the first to study computation using faulty gates. Von Neumann showed that, under certain conditions, increased gate redundancy can lead to increased reliability of a circuit. However, it was shown that, in general, computation by faulty gates with non-zero computational capacity is not possible (see [3], [4]). The study of storage circuits made of unreliable components led to much more optimistic results. Taylor in [5] proved that a memory has an associated information storage capacity, C, such that arbitrarily reliable information storage is possible for all memory redundancies greater than $1/C$. The methodology of the proof, however, does not allow one to explicitly calculate the storage capacity. Taylor [5] considered two models of component failures and proposed construction of fault-tolerant memories based on low-density parity-check (LDPC) codes. In the first model, the failures of a particular component are assumed to be statistically independent from one use to another. In the second model, the components fail permanently but bad components are replaced with good ones at regular intervals. The failures in different components are assumed to be independent in both the models. This construction was further studied by Kuznestsov in [6] and we will refer to it as the Taylor-Kuznetsov (TK) scheme. Hadjicostis [1] was able to generalize Taylor's scheme to fault tolerant linear finite state machines. Spielman [7] obtained the best result for a general model of computation, by marrying the ideas of von Neumann with Reed-Solomon (RS) codes.

The fundamental contribution of this paper is to show existence of reliable memories built entirely from unreliable components and which have finite redundancies. We consider the adversarial failure model in which only a fixed fraction of the components can fail at any given time. Our fault tolerant memory architecture is also based on LDPC codes but differs from the TK scheme in the decoding algorithm employed. The TK scheme can be shown to be an implementation of the Gallager B decoding algorithm for LDPC codes (the proof will be given in the longer version of the paper). We use the parallel bit flipping decoding algorithm proposed in the context of expander codes by Sipser and Spielman [8]. Expander codes also belong to the class of LDPC codes and are asymptotically good error correcting codes with linear time decoding algorithms which can correct a linear fraction of errors. Expander graph based arguments have been successfully applied for message passing algorithms [9] as well as for linear programming decoding [10]. At the time of their discovery, explicit construction of graphs with expansion required for parallel bit flipping algorithm were not known. Capalbo *et al*. [11] recently gave an explicit construction of expander graphs based on randomness constructors. Hence, our method can be seen as a constructive proof in contrast to Taylor's method which is an existence proof.

The rest of the paper is organized as follows. In Section II we provide the necessary definitions and a brief overview of LDPC Codes. We explain the proposed memory architecture and characterize it in terms of complexity and redundancy. In Section III we introduce the model of failure of the components and prove our main result showing the existence of memories which can tolerate failures in all the components. In Section IV we provide a few numerical examples and we conclude in Section V with a few remarks.

II. The System Description

In this section we give a detailed description of the memory system. We start by introducing the terminology used to characterize memories and proceed to discuss the importance of LDPC codes. We explain the coding scheme and the error correction scheme employed in the proposed memory architecture. We then calculate the redundancy and complexity associated with the memories.

A. Definitions

A memory is a device in which information is stored at some time and retrieved at a later time [5]. A memory circuit consists of registers (memory elements) which can store a single bit. The information storage capability of a memory is the number of information bits it stores. A k-bit memory circuit built out of reliable registers has information storage capability of k bits. Such a memory is termed as an irredundant memory. Now, consider the problem of information storage with unreliable memory elements. Due to the component failures, the information read out of the memory may not be identical to the information stored originally. Hence, to ensure reliable storage, the information needs to be stored in coded form (see [5] for an excellent discussion on the importance of coded form). Initially, a codeword from some error correcting code is stored in the memory. The unreliable nature of the memory elements introduces errors in the registers and the contents of the memory differ from the initial state. A correcting circuit is employed which performs error correction and updates the contents of the registers with an estimate of the original codeword. Hence, a fault-tolerant memory system consists of a memory circuit and a correcting circuit. The correcting circuit is also built of unreliable components. The coding of information along with the correcting circuit introduce redundancy into the memory system. Such redundant memories are characterized by two closely related parameters, namely, complexity and redundancy. The *complexity* of a memory is the number of components within the memory (a component is a device which either performs an elementary operation or stores a single bit where an elementary operation is any Boolean function of two binary operands [5]). The *redundancy* of a memory is the ratio of the complexity of the memory to the complexity of an irredundant memory which has the same information storage capability. It should be noted that there can be many memory architectures with the same information storage capability.

Another important characteristic of a memory is reliability. We say that arbitrarily reliable information storage is possible in a memory if the probability of memory failure can be made arbitrarily small. To quantify the reliability of a memory system, it is important to first define what constitutes a memory failure. Let a memory failure be defined as an event in which the word read out of memory is not equal to the original codeword. Arbitrarily reliable information storage is not possible with such a definition of memory failure. This is due to the fact that the probability of failure is lower bounded by the probability of failure of components in the final step of extracting the information bits. Hence, we define a failure in the following manner. Associated with each codeword in a code is a decoding equivalence class, i.e., the set of words which decode to that particular codeword when decoded with a decoder built of reliable components. If the contents of the memory do not belong to the decoding equivalence class of the original codeword, we say a memory failure has occurred. The *storage capacity*, C, of a memory is a number such that for all memory redundancies greater than $1/C$, arbitrarily reliable information storage is possible [5].

B. LDPC Codes

The information storage capability of a memory depends on the type of the code employed in the correcting circuit. The memories under consideration store information in form of bits and therefore we restrict our attention to binary codes in this paper. An (n, k) binary block code maps a message block of k information bits to a binary n-tuple [12]. The rate r of the code is given by $r = k/n$. Hence, a memory employing an (n, k) block code has information storage capability of k bits. An (n, k) binary linear block code, $\mathcal{C}$, is a subspace of $GF(2)^n$ of dimension k [12]. A parity check matrix H of $\mathcal{C}$ is a matrix whose columns generate the orthogonal complement of $\mathcal{C}$, i.e., an element $\mathbf{w}$ of $GF(2)^n$ is a codeword of $\mathcal{C}$ iff $\mathbf{w}H^T = \mathbf{0}$ [13].

Taylor in [5] argues that no decoding scheme other than iterative decoding of LDPC codes can achieve non-zero storage capacity. LDPC codes [14] are a class of linear block codes which can be defined by sparse bipartite graphs [15]. Let $\mathcal{G}$ be a bipartite graph with two sets of nodes: n variable (bit) nodes and m check (constraint) nodes. The check nodes (variable nodes) connected to a variable node (check node) are referred to as its neighbors. The degree of a node is the number of its neighbors. This graph defines a linear block code of length n and dimension at least $n-m$ in the following way: The n variable nodes are associated to the n coordinates of codewords. A vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ is a codeword if and only if for all each check node, the sum of its neighbors is zero. Such a graphical representation of an LDPC code is called the Tanner graph [16] of the code. The adjacency matrix of $\mathcal{G}$ gives H, a parity check matrix of $\mathcal{C}$. It should be noted that the Tanner graph is not uniquely defined by the code and when we say the Tanner graph of an LDPC code we only mean one possible graphical representation. The Tanner graph of an (n, γ, ρ) regular LDPC code has n variable nodes each of degree c and $n\gamma/\rho$ check nodes each of degree ρ. This code has length n and rate $r \geq 1 - \gamma/\rho$ [15].

C. The Proposed Fault Tolerant Memory Architecture

The complexity and redundancy of a fault-tolerant memory depend on the coding scheme as well as the decoding algorithm employed in updating the contents of the memory. We now explain our memory architecture in detail.

At time $t = 0$, a codeword from an (n, γ, ρ) regular binary LDPC code is written into the memory circuit consisting of n registers each of which can store a single bit. The n bits of the codeword correspond to the n variable nodes in the Tanner graph, $\mathcal{G}$, of the code and are denoted by v_i, $1 \leq i \leq n$. Let c_i, $1 \leq i \leq n\gamma/\rho$, denote the i^{th} check node. The contents of

the registers are updated at times $\tau, 2\tau, \ldots, L\tau$, $L \in I\!N$. The update rules can be explained by defining messages along the edges in $\mathcal{G}$. For a variable node v (check node c), let $E(v)$ ($E(c)$) denote the edges incident on v (c). Each edge e is associated with a variable node v and a check node c. Let $\overrightarrow{\mathrm{m_t}}\ (e)$ and $\overleftarrow{\mathrm{m_t}}\ (e)$ represent the messages passed on an edge e from variable node to check node and check node to variable node at time t respectively. Let $v(t)$ denote the value of variable node v at time t. Then the update at time t is given by the following algorithm:

Algorithm A

- For each edge e and corresponding variable node v

$$\overrightarrow{\mathrm{m_t}}\ (e) = v(t^-)$$

- For each edge e and corresponding check node c

$$\overleftarrow{\mathrm{m_t}}\ (e) = \left(\sum_{e' \in E(c) \setminus \{e\}} \overrightarrow{\mathrm{m_t}}\ (e') \right) \mod 2$$

- For each variable node v

$$v(t^+) \quad = \quad \begin{cases} 1, & \sum_{e \in E(v)} \overleftarrow{\mathrm{m_t}}\ (e) \geq \lceil \gamma/2 \rceil \\ 0, & \gamma - \sum_{e \in E(v)} \overleftarrow{\mathrm{m_t}}\ (e) \leq \lfloor \gamma/2 \rfloor \\ v(t^-), & \text{otherwise} \end{cases}$$

Remarks: We assume that the update is instantaneous and use $v(t^-)$ and $v(t^+)$ to denote the value of variable v just before and after the update respectively. We note that the algorithm presented above is a slight modification of the parallel bit flipping algorithm proposed in [8].

D. Complexity and Redundancy

LDPC codes can achieve non-zero capacity due to the fact that the redundancy of the LDPC codes memory increases linearly with the information storage capability. The complexity of the logic gates needed to perform decoding depend only on γ and ρ and not on the length of the code (as is the case with sequential decoders). So the redundancy remains bounded even as the code length tends to infinity. We now calculate the complexity and redundancy associated with our fault-tolerant memory architecture. The memory circuit consists of n registers each of which can store a single bit. The correcting circuit consists of logic gates (built from components) needed to implement the update algorithm. The algorithm can be interpreted in the following manner. Every variable node sends an estimate of its value to the neighboring check nodes. A check node calculates an estimate of a neighboring variable node by computing the modulo two sum of all the remaining $\rho - 1$ neighboring variable nodes. This requires a $(\rho-1)$-input XOR gate which can be implemented using $\rho - 2$ two input XOR gates (a two input XOR gate calculates modulo two sum of two bits). A variable node receives γ estimates, one from each neighboring check and the majority of these estimates is the updated value of the node. This requires a γ-input majority logic gate whose complexity we denote by D_γ. Hence, for every variable node the correcting circuit requires $\gamma(\rho-2)$ two-input XOR gates and a γ-input majority logic gate. The complexity of the entire system is therefore

$$\mathcal{C} = n(1 + D_\gamma + \gamma(\rho - 2))$$

The memory has information storage capability of rn bits and the complexity of an irredundant memory with the same information storage capability is rn. The redundancy of the fault-tolerant memory is therefore

$$\begin{aligned} R &= n(1 + D_\gamma + \gamma(\rho - 2))/rn \\ &\leq (1 + D_\gamma + \gamma(\rho - 2))/(1 - \gamma/\rho) \end{aligned}$$

III. Analysis of the Memory System

The storage capacity of a memory depends on the type of failures in the components. A logic gate is said to have failed if its output is flipped. A register is said to have failed if the bit stored in it is flipped. In this paper we consider the adversarial failure model also referred to as bit flipping channel model. In the adversarial model, the failures occur in the worst case fashion but no more than a fixed fraction of the components fail at any given time. In other words, the number of failures is bounded for a given number of components. As the number of components increases so do the number of failures. We denote the fraction of memory element failures in a time interval τ by α_m, fraction of two input XOR gate failures for every use by $\alpha_\oplus$ and fraction of γ-input majority logic gate failures for every use by α_γ. As mentioned before, the component failures are transient and independent from one use to another.

A memory system is said to tolerate a constant fraction of errors in all components if at any time at most a constant fraction of components can fail and no memory failure occurs in the system at all times $t < \infty$. Recall that, from our definition, a memory failure occurs if the contents in the memory do not belong to the decoding class of the originally stored codeword. In this section, we prove that the memory architecture proposed in Section II can tolerate a constant fraction of failures in all the components. Our proof is based on the expansion property of the underlying Tanner graph, $\mathcal{G}$, of the code.

Definition 1: [8] A Tanner graph $\mathcal{G}$ of a (n, γ, ρ) LDPC code is a $(\gamma, \rho, \alpha, \delta)$ expander if for every subset S of at most an αn variable nodes, at least $\delta|S|$ check nodes are incident to S.

The definition of expander is much more general but we restrict our attention to Tanner graphs of LDPC codes. Sipser and Spielman in [8] proposed a class of asymptotically good error correcting codes based on expander graphs known as expander codes. LDPC codes are a special case of expander codes in which the expander graph is the Tanner graph of the LDPC code. In [8] it is shown that when the underlying graph has sufficient expansion, the code can correct a fixed fraction of errors. Sipser and Spielman proposed two simple bit flipping algorithms: serial and parallel. We describe the parallel bit flipping algorithm and interested readers are referred to [8] for details about serial bit flipping. We say that a constraint is satisfied by a setting of variables if the

sum of the variables in the constraint is even; otherwise, the constraint is unsatisfied.

Parallel Bit Flipping Algorithm

- In parallel, flip each variable that is in more unsatisfied than satisfied constraints.
- Repeat until no such variable remains.

Suppose a codeword is stored is stored in a memory. The word read out from the memory may differ from the original codeword. The set of variable nodes (bits) which differ from their original value are known as corrupt variables. The following theorem from [8] gives the sufficient conditions for the parallel bit flipping algorithm to correct a constant fraction of errors.

Lemma 1 ([8], Theorem 11): Let $\mathcal{G}$ be a $(\gamma, \rho, \alpha, (3/4 + \epsilon)\gamma)$ expander over n variable nodes, for any $\epsilon > 0$. Then, the simple parallel decoding algorithm will correct any $\alpha_0 < \alpha(1 + 4\epsilon)/2$ fraction of error after $\log_{1-4\epsilon}(\alpha_0 n)$ decoding rounds. Also, if V denotes the set of corrupt variables in the input and $|V| < \alpha n(1 + 4\epsilon)/2$, then the parallel decoding algorithm produces a word with at most $|V|(1 - 4\epsilon)$ corrupt variables after one decoding round.

Proof: See [8] ■

From Lemma 1, it is clear that a word belongs to the decoding class of a codeword as long as the fraction of corrupt variables (bits) is less than $\alpha(1+4\epsilon)/2$. Note that Algorithm A is a slight modification of one iteration of the parallel bit flipping algorithm of [8]. In the parallel bit flipping algorithm, every check node indicates to its neighboring variable node if it is satisfied or not. In Algorithm A every check node gives an estimate of the variable node. Theoretically both the algorithms are equivalent but we use the Algorithm A as it has lesser redundancy. We now state and prove our main theorem.

Theorem 1: Let $\mathcal{G}$ be a $(\gamma, \rho, \alpha, (3/4 + \epsilon)\gamma)$ expander for any $\epsilon > 0$. The proposed memory architecture can tolerate constant fraction of errors in all the components if

$$\alpha_m + \gamma(\rho - 2)\alpha_\oplus + \alpha_\gamma < \alpha(1 + 4\epsilon)(4\epsilon)/2$$

Proof: At $t = 0$, a codeword from an (n, γ, ρ) LDPC code with Tanner graph $\mathcal{G}$ is read into the memory. The contents are updated at times $\tau, 2\tau, \ldots, L\tau$, $L \in I\!N$, by running Algorithm A. We bound the number of corrupt variables at time t. Let $\alpha_v(t)$ denote the fraction of corrupt variables at time t. We establish bounds on $\alpha_v(t)$ for all t. We first prove the following. Let $\delta > 0$. If

$$\alpha_v((l-1)\tau - \delta) < \alpha(1 + 4\epsilon)/2,$$

then

$$\alpha_v(l\tau - \delta) < \alpha(1 + 4\epsilon)/2$$

Let $V(t)$ denote the set of corrupt variables at time t.

$$|V((l-1)\tau - \delta)| = \alpha_v((l-1)\tau - \delta)n.$$

Since $\alpha_v((l-1)\tau - \delta) < \alpha(1 + 4\epsilon)/2$, a decoder built with reliable gates outputs a word with at most $|V((l-1)\tau - \delta)|(1 - 4\epsilon)$ corrupt variables (by Lemma 1). We now bound the number of errors introduced due to the faulty nature of the decoder. Each XOR gate failure can corrupt at most one variable and each majority logic gate failure can corrupt at most one variable. So,

$$\begin{aligned} |V((l-1)\tau)| &< |V((l-1)\tau - \delta)|(1 - 4\epsilon) \\ &+ \gamma(\rho - 2)\alpha_\oplus n + \alpha_\gamma n \end{aligned} \quad (1)$$

Eq. 1 bounds the number of corrupt variables at the end of $(l-1)^{th}$ correcting cycle. However, in the time interval $[(l-1)\tau \;\; l\tau)$, at most $\alpha_m n$ variables can get corrupted due to failures in memory elements. Therefore, the time at which there are maximum number of corrupt variables is just before the start of a correcting cycle, i.e.,

$$\alpha_v(l\tau - \delta) = \max\{\alpha_v(t) : (l-1)\tau \le t < l\tau\}$$

Hence, it suffices to bound $\alpha_v(t)$ for $t = l\tau - \delta, l = 1, 2, \ldots, L$.

$$\begin{aligned} |V((l\tau - \delta)| &< |V((l-1)\tau - \delta)|(1 - 4\epsilon) \\ &+ \gamma(\rho - 2)\alpha_\oplus n + \alpha_\gamma n + \alpha_m n \end{aligned} \quad (2)$$

Dividing Eq. 2 by n gives

$$\begin{aligned} \alpha_v((l\tau - \delta) &< \alpha_v((l-1)\tau - \delta)(1 - 4\epsilon) \\ &+ \gamma(\rho - 2)\alpha_\oplus + \alpha_\gamma + \alpha_m \\ &< \alpha(1 + 4\epsilon)(1 - 4\epsilon)/2 + \alpha(1 + 4\epsilon)(4\epsilon)/2 \\ &= \alpha(1 + 4\epsilon)/2 \end{aligned}$$

Since

$$\alpha_v(\tau - \delta) \le \alpha_m < \alpha(1 + 4\epsilon)/2,$$

it follows that

$$\alpha_v(l\tau - \delta) < \alpha(1 + 4\epsilon)/2 \qquad \forall l \in I\!N.$$

Hence,

$$\alpha_v(t) < \alpha(1 + 4\epsilon)/2 \qquad \forall t < \infty$$

and no memory failure occurs. ■

It is instructive to see the behavior of the memory in the absence of the correcting circuit. In any time interval of τ seconds, at most $\alpha_m n$ fraction of the memories may fail. After sufficiently long time, the fraction of corrupt variables becomes more than $\alpha(1 + 4\epsilon)/2$ and a memory failure occurs. The presence of a correcting circuit ensures that at any time the number of corrupt variables remains less than the correcting capability of the code. However, for a given expander there is a loss in the tolerable memory failure due to the faulty nature of the gates as well as the iterative nature of the decoder. Consider the case of where decoder is reliable and failures occur only once. The tolerable fraction of errors for a given expander is close to $\alpha(1 + 4\epsilon)/2$. In the case of memories with unreliable memory elements but reliable logic gates, the tolerable fraction of memory errors is close to $\alpha(1 + 4\epsilon)(4\epsilon)/2$. The reduction by a factor of 4ϵ occurs due to the fact that decoder is iterative in nature and needs multiple rounds to converge to the codeword. One round of error correction decreases the errors by a factor of $(1 - 4\epsilon)$ and $\alpha_m n$ new errors might be introduced due to memory failures. In the extreme case of $\epsilon = 1/4$ we have a decoder which takes just one step to correct all the corrupt

variables, in which case the tolerable failure rate is arbitrarily close to $\alpha(1+4\epsilon)/2$. The faulty nature of the decoder further reduces the tolerable memory failure rate. Given the values of $\alpha_m, \alpha_\oplus, \alpha_\gamma$, a code based on graph with sufficient expansion can be chosen to build a fault tolerant memory. It is well known that a random graph is a good expander with high probability (see [8] and references therein). In the next section, we illustrate this fact with a few examples.

IV. Numerical Results

In this section we illustrate with specific numerical examples the redundancies and tolerable failure rates associated with different values of γ and ρ. We first make the following observations. The redundancy of a memory system depends on the parameters γ and ρ of the LDPC code used. Different values of γ and ρ can result in same redundancy. To compare across different values of γ and ρ, the values of D_γ and α_γ have to be chosen consistently. How D_γ and α_γ scale with γ depends on the technology and implementation. Assuming that all gates are built out of universal NAND gates also does not answer the question fully as different implementations can lead to different values. Hence for the sake of illustration we consider a specific implementation. It should be noted that the subsequent discussion is for illustration purpose only. Accurate analysis for a given case can be carried out along the lines of the method we present in this section. For a given implementation, we fix the the values of γ and ρ thereby fixing the redundancy as well as α_γ and $\gamma(\rho-2)\alpha_\oplus$. We then use the bounds on the achievable expansion of a (γ,ρ) regular bipartite graph to find bounds on the value of $\alpha_{total} = \alpha(1+4\epsilon)(4\epsilon)/2$. This in turn provides bounds on the value of α_m for fixed γ and ρ.

A. Redundancy

Recall that the redundancy of a memory system is given by

$$\begin{aligned} R &= n(1+D_\gamma+\gamma(\rho-2))/rn \\ &\leq (1+D_\gamma+\gamma(\rho-2))/(1-\gamma/\rho) \end{aligned}$$

For a fixed γ, R is minimum for a certain ρ depending on the value of D_γ. For example, if $D_\gamma = 2\gamma-1$, then it can be shown that $\rho = 2\gamma$ minimizes the redundancy. This implies that a rate $1/2$ code has the least redundancy for a given γ.

Figure 1(a) and Figure 1(b) show the dependence of the redundancy on ρ for a given value of γ.

B. Bounds on Expansion

We make use of the following theorem from [8] to find an upper bound α_{total} for a given γ and ρ.

[Theorem 25, [8]]: Let B be a bipartite graph between n c-regular vertices and $(c/d)n$ d-regular vertices. For all $0<\alpha<1$, there exists a set of αn c-regular vertices with at most

$$n\frac{c}{d}(1-(1-\alpha)^d)+O(1) \text{ neighbors}$$

It should be noted that the upper bound is tight for higher values of c.

Using this theorem, we can find an upper bound on α_{total} for a given γ and ρ. It should also be noted that we look for graphs which expand by at least a factor of $(3/4+\epsilon)$.

The following proposition from [10] addresses the issue of existence of expanders.

[Proposition 6, [10]]: Let $0<r<1$ and $0<\delta<1$ be any fixed constants, and let c be such that $(1-\delta)c$ is an integer which is at least 2. Then for any n, m such that $r = 1-m/n$ there is a Tanner graph with n variable nodes, m check nodes, and regular left degree c which is an $(\alpha n, \delta c)$-expander, where

$$\alpha = (2e^{\delta c+1}(\delta c/(1-r))^{(1-\delta)c})^{-\frac{1}{(1-\delta)c-1}}$$

It should be noted that the notation for expanders is different in [10]. Also, the proof does not guarantee that all the check nodes have same degree.

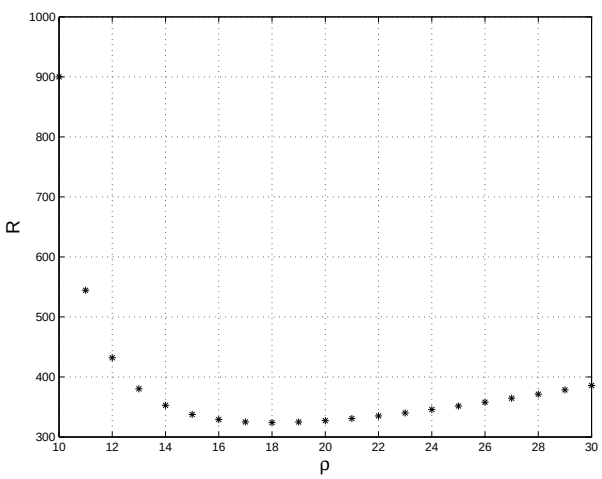
This proposition guarantees the existence of graphs with sufficient expansion and can be used to derive a lower bound on α_{total} for given γ and ρ. This in turn proves existence of memories which can tolerate $\alpha_m, \alpha_\oplus$ and α_γ fraction of failures in respective components as long as $\alpha_m + \gamma(\rho-2)\alpha_\oplus + \alpha_\gamma < \alpha_{total}$. Figures 1(c) and 1(d) illustrate the upper bounds and lower bounds on α_{total} for $\gamma = 9$ and $\gamma = 34$ respectively. We remark that the bounds have been derived numerically and we do not attempt to give closed form expressions for the bounds as the results are for illustration purpose only.

V. Conclusion

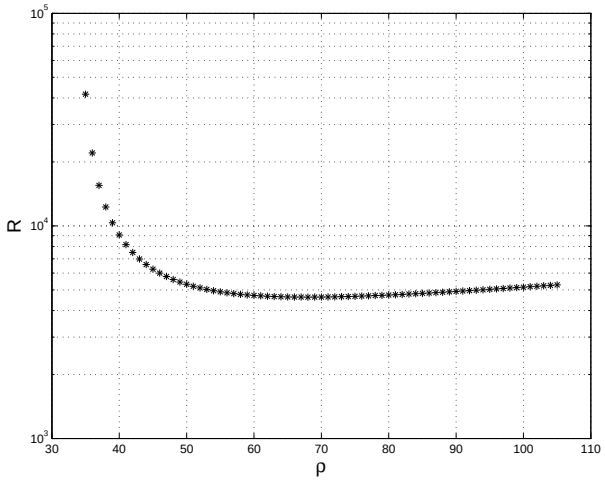
We have shown the existence of memories built entirely from unreliable components which can tolerate failures in all components. The results can be extended to the independent failure model in which each component can fail with a certain probability by using Chernoff bounds [17] (the proof will appear in longer version of the paper). The existence of finite length LDPC codes with required expansion as well as explicit methods to construct such expanders makes our method constructive as opposed to the TK scheme in which results hold only in the case of infinite length. Our memory architectures have lesser redundancies compared to the TK scheme (the proof will appear in longer version of the paper). We remark that explicit calculation of storage capacity still remains a challenging problem due to the complex interdependence of various parameters. Improving the bounds on expansion can lead to better estimates. The fraction of tolerable failures can be also improved. Expander codes with modified decoding algorithms have been proposed in [8], [18], [19], [20] which can recover from larger fraction of failures in linear time. Investigation of fault-tolerant implementation of these codes can lead to increasing the tolerable failure rates. The study of error exponents for the independent failure model, the use of irregular LDPC codes and employing message passing decoding in place bit flipping decoding can lead to further improvements and is a focus of our current research.

Acknowledgment

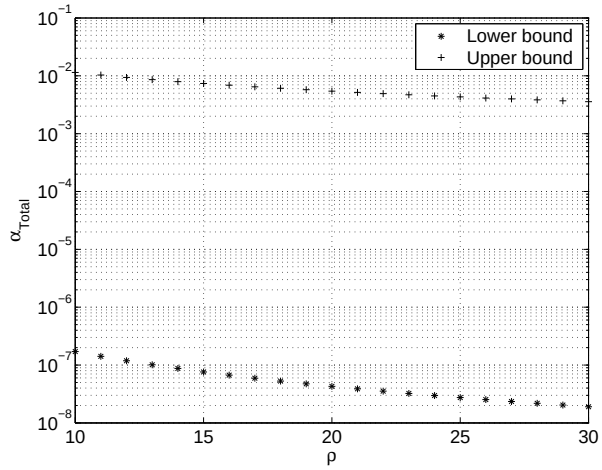
This work was supported by the NSF under Grant CCF-0634969.



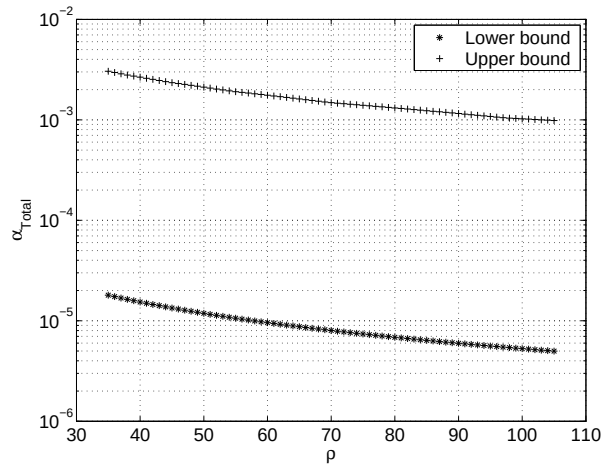
(a) Redundancy for $\gamma = 9$



(b) Redundancy for $\gamma = 34$



(c) Bounds on α_{total} for $\gamma = 9$



(d) Bounds on α_{total} for $\gamma = 34$

Fig. 1. Redundancies and bounds on expansion for different values of γ

REFERENCES

[1] C. N. Hadjicostis and G. C. Verghese, "Coding approaches to fault tolerance in linear dynamic systems," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 210–228, Jan. 2005.

[2] J. V. Neumann, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*, ser. Automata Studies. Princeton: Princeton University Press, 1956, pp. 43–98.

[3] R. L. Dobrushin and S. I. Ortyukov, "Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements," *Probl. Inform. Transm.*, vol. 13, pp. 59–65, 1977.

[4] N. Pippenger, "Developments in 'the synthesis of reliable organisms from unreliable gates'," in *Symposia in Pure Mathematics*, 1990, pp. 311–324.

[5] M. Taylor, "Reliable information storage in memories designed from unreliable components," *Bell System Technical Journal*, vol. 47, pp. 2299–2337, 1968.

[6] A. Kuznetsov, "Information storage in a memory assembled from unreliable components," *Problems of Information Transmission*, vol. 9, pp. 254–264, 1973.

[7] D. Spielman, "Highly fault-tolerant parallel computation," in *IEEE Conference on Foundations of Computer Science*, 1996, pp. 154–163.

[8] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[9] D. Burshtein and G. Miller, "Expander graph arguments for message-passing algorithms," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 782–790, Feb. 2001.

[10] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright, "LP decoding corrects a constant fraction of errors," *IEEE Trans. Inform. Theory*, vol. 53, no. 1, pp. 82–89, Jan. 2007.

[11] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, "Randomness conductors and constant-degree lossless expanders," in *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 2002, pp. 659–668.

[12] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[13] D. Terr, "Parity check matrix." [Online]. Available: http://mathworld.wolfram.com/ParityCheckMatrix.html

[14] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[15] A. Shokrollahi, "An introduction to low-density parity-check codes," in *Theoretical aspects of computer science: advanced lectures*. New York, NY, USA: Springer-Verlag New York, Inc., 2002, pp. 175–197.

[16] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, Sept. 1981.

[17] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, vol. 23, pp. 493–507, 1952.

[18] V. Guruswami and P. Indyk, "Linear-time encodable/decodable codes with near-optimal rate," *IEEE Trans. Inform. Theory*, vol. 51, no. 10, pp. 3393–3400, Oct. 2005.

[19] A. Barg and G. Zemor, "Error exponents of expander codes," *IEEE Trans. Inform. Theory*, vol. 48, no. 6, pp. 1725–1729, Jun. 2002.

[20] G. Zemor, "On expander codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 835–837, Feb. 2001.