Context-Aware Resiliency: Unequal Message Protection for Random-Access Memories

Clayton Schoeny, *Member, IEEE*, Frederic Sala, *Member, IEEE*, Mark Gottscho, *Member, IEEE*, Irina Alam, *Student Member, IEEE*, Puneet Gupta, *Senior Member, IEEE*, and Lara Dolecek, *Senior Member, IEEE*

***Abstract*— A common way to protect data stored in DRAM and related memory systems is through the use of an error-correcting code such as the extended Hamming code. Traditionally, these error-correcting codes provide equal protection guarantees to all messages. In this paper, we focus on *unequal message protection* (UMP), in which a subset of messages is deemed as special, and is afforded additional error-correction protection while maintaining the same number of redundancy bits as the baseline code. UMP is a powerful approach when the special messages are chosen based on the knowledge of data patterns in context. Our objective is to construct deterministic, algebraic codes with guaranteed UMP properties, derive their cardinality bounds using novel combinatorial techniques, and to demonstrate their efficacy for realistic memory benchmarks. We first introduce a UMP alternative to the single-bit parity-check code, and then we generalize to a broader UMP code family, including a UMP alternative to the extended Hamming code, offering full double-error correction protection to special messages. Our UMP constructions, applied to main memory in high-performance computing applications, could lead to significant system-level benefits such as less frequent checkpoints in supercomputers and decreased risk of catastrophic failure from erroneous special messages.**

***Index Terms*— Error correction codes, Hamming distance, random access memory.**

I. Introduction

ERROR-CORRECTING codes (ECCs) play a critical role in memory resiliency. Traditionally, one of the most important metrics of interest is the minimum distance of a code, which provides guarantees on error-correction and error-detection capabilities. Intriguingly, *side-information* about the underlying data and communication channel can be used to enhance error-correction probabilities, extending the traditional notions from classical coding theory.

Recently, we proposed *Software-Defined Error-Correcting Codes* (SDECC), a class of heuristic techniques to recover from detected-but-uncorrectable errors (DUEs) [3]–[5]. SDECC can be considered as a highly practical list-decoding ([6]–[8]) framework that utilizes any linear code capable of correcting t errors and detecting $t + 1$ errors. Traditionally, when a DUE occurs, the memory system will either crash or restore to a checkpoint [9]. In our SDECC framework, when a DUE occurs, we first compute a list of *candidate codewords*—the closest neighboring codewords—and then probabilistically decode based on available side-information. SDECC is applicable to a wide variety of memory applications and systems ranging from large-scale servers in data centers to embedded systems in Internet-of-Things devices.

In this work, we take a different approach and focus on the encoding-side of SDECC: instead of using side-information to heuristically decode, we *a priori* designate specific messages to have extra protection against errors. We designate two classes of messages, *normal* and *special*, and they are mapped to normal and special codewords, respectively. When dealing with the underlying data, we refer to the *messages*; when discussing error detection/correction capabilities we refer to the *codewords*. Within the SDECC framework, special codewords can be viewed as a set of codewords with the property that no two elements from the set are ever in the same candidate list, i.e., when a DUE occurs, there will never be two or more special codewords among the neighboring codewords.

This type of *unequal message protection* (UMP) is fundamentally different from unequal error protection (UEP) [10], in which all codewords have extra protection for specific bit positions or certain error patterns (such as adjacent bit errors). UMP is a powerful approach when the special messages are chosen with regard to both the relative frequency and meaning of the stored data. In particular, UMP is useful when compression is not feasible, yet specific messages—or parts of specific messages—are very frequently stored/transmitted, as is the case in modern random-access memories [4], [5]. Additionally, given side-information about the system-level meaning of underlying messages, we may add extra protection to those messages whose miscorrections would be very costly.

Manuscript received March 15, 2018; revised January 13, 2019; accepted May 1, 2019. Date of publication May 22, 2019; date of current version September 13, 2019. This work was supported in part by the NSF under grant CCF 1718389, in part by the 2017 Qualcomm Innovation Fellowship, and in part by the 2017–2018 UCLA Dissertation Year Fellowship. This paper was presented in part at the 2017 IEEE Information Theory Workshop [1] and in part at the 2018 IEEE/IFIP International Conference on Dependable Systems and Networks Workshops [2].

C. Schoeny, I. Alam, P. Gupta, and L. Dolecek are with the Electrical and Computer Engineering Department, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: cschoeny@ucla.edu; irina1@ucla.edu; puneetg@ucla.edu; dolecek@ee.ucla.edu).

F. Sala is with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA (e-mail: fredsala@stanford.edu).

M. Gottscho was with the University of California at Los Angeles, Los Angeles, CA 90095 USA. He is now with Google, Mountain View, CA 94043 USA (e-mail: mgottscho@ucla.edu).

Communicated by J. Kliewer, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2019.2918209

For practical applications, most recent processors with large capacity on-chip caches have ECC protected L2 and/or L3 caches. Some common and recent examples include Qualcomm Centriq 2400 [11], AMD Athlon [12], AMD Opteron [13], and IBM Power 4 [14] processors. Additionally, in random-access memories, such as DRAM and SRAM [15], [16], the three most commonly used ECC classes are the single-bit parity-check code, the extended Hamming code, and the ChipKill (or equivalent) code [17]. The choice of appropriate ECC class depends on many system-level requirements including latency, energy, storage overhead, etc. For each of these codes, we create an alternative UMP code with enhanced error-correcting features, while still using the same number of redundancy bits. For example, the extended Hamming code is capable of correcting any single-bit error and detecting any double-bit error; our UMP alternative code sacrifices the universal double-bit detection in order to grant double-bit correction to special codewords. For a given set of code parameters (i.e., code size, dimension, and detection/correction properties), our goal is to maximize the number of special messages.

One crucial property of our proposed UMP scheme is that both classes of messages have the same length (as well as both classes of codewords). This property allows our coding scheme to be directly applicable to the vast majority of memory systems, which use fixed bit-length architectures. While it is possible to achieve a similar protection outcome by using, for example, a Hamming code for normal messages and a BCH code for special messages, this scheme would increase the codeword length for special codewords, thereby not fitting into fixed memory widths.

The paper is organized as follows. The remainder of this section is an overview of related work. In Section II, we provide preliminaries, notation, and objectives. Both Sections III and IV contain—for their respective codes—a derivation for the modified sphere-packing bounds on the number of special messages, an explicit code construction, a proof of correction properties, and a walkthrough of the decoding process. Section III focuses on the UMP alternative to the parity-check code, in which we trade-off single-bit detection in favor of single-bit correction for special codewords. Additionally, we show how an additional redundancy bit can be used to revive the single-error detection property for normal codewords. Section IV deals with the UMP alternative for the extended Hamming code. In Section V, we derive a novel programming bound for the number of special messages for our UMP codes. In Section VI, we discuss various strategies for the special message mapping and we investigate the benefits of our UMP codes on real-world memory benchmarks. We conclude in Section VII.

A. Related Work

The majority of research into UEP codes has focused on *bit-wise* UEP, in which specific positions of a codeword are more robust to errors [10], [18]. Masnick and Wolf [10] created a framework for constructing linear bit-wise UEP codes, in which each bit in a codeword is assigned an error protection level. Bit-wise UEP codes are useful when errors in specific bit positions are more severe, e.g., the most significant bit of a binary integer or the destination address header of a packet. A particular bit is then guaranteed to be decoded correctly if its error protection level is greater than or equal to the total number of errors in the codeword.

Another type of UEP is *error-wise* UEP, in which specific error patterns are guaranteed to be correctable. Error-wise UEP codes are useful when bit-error locations are not independent. A code that is designed to correct burst errors can be thought of as an error-wise UEP code. For example, *single-error-correcting/double-error-detecting/double-adjacent-error-correcting* (SECDED-DAEC) codes guarantee correction in the case of a single-bit error or a double-bit error given that the erroneous bits are adjacent [19], [20]. Error-wise UEP codes can also be useful when different sections of the codeword are stored in different chips in computer hardware, in which case a faulty chip only causes errors on a specific subsection of the codeword [21], [22].

In this work, we focus on UMP, i.e., *message-wise* UEP, in which specific messages have extra protection from errors. In this setting, Broade *et al.* used an information-theoretic approach to prove that it is possible to encode many special messages, even at rates approaching the channel capacity [23].

Shkel *et al.* [24] also examined the UMP problem. The main distinction between their work and ours is their work is concerned with producing information-theoretic bounds (achievability and converse) for such codes with average and maximal error probability over a probabilistic channel. Shkel *et al.* followed the line of work considered in Broade *et al.*, but they also looked at the finite-length regime by applying the finite blocklength framework from Polyanski *et al.* [25] to the UMP setting. Nevertheless, theirs is a different setting compared to ours: we are interested in adversarial, not probabilistic, errors and we wish to produce short, explicit non-randomized code constructions. Additionally, this work is the first—to the best of our knowledge—to implement UMP coding schemes in practical memory systems.

Our approach also complements recent research on data compression in cache and main memory systems, an emerging topic that aims to meet the energy and storage demands brought upon by the exponential growth of produced data. Techniques include frequent value compression [26], frequent pattern compression [27], and base-delta-immediate compression [28]. These techniques add considerable complexity and overhead that may not always be tolerable; they nevertheless clearly demonstrate that there is a tremendous amount of correlation and redundancy inherent in the data used in main memory systems, which we seek to capitalize on, not for compression, but instead for resilience. This inherent data correlation is a key factor allowing our UMP coding framework to be innovative and useful.

Our work also relates to past research on joint source/channel coding. Works in this area observe that although the source/channel coding separation theorem states that optimally there is no loss from separately removing redundancy from a source (source coding) then independently encoding the resulting output (channel coding), for practical

finite-length codes, the source coding process still leaves some redundancy. This remaining redundancy, called *residual redundancy*, which is intrinsic to the source, can be exploited via channel coding schemes to improve performance. Such works include those of Sayood and Borkenhagen [29], Phamdo and Farvardin [30], and Hagenauer [31], who added a Viterbi-like decoder that takes advantage of the residual redundancy in lieu of channel codes. To better handle low error-probability cases, Otu and Sayood [32] added constraints to the source-coder output, further increasing the residual redundancy.

A number of such papers are concerned with the variable-length code (VLC) setting commonly used in source coding. Papers that focus on memoryless sources include [33], while other more recent works focus on the first-order Markov sources, using trellis decoding to take advantage of residual redundancy for error-correction [34], [35]. The source model is extended to Markov Random Fields (MRFs) in [36]. More recent efforts along these lines include Jiang *et al.*, where machine learning methods and the inherent redundancy in language-based sources are used to improve the rate of Polar codes and the performance of LDPC codes for non-volatile memories, [37]–[39].

There are several significant differences between our UMP approach and residual redundancy approaches. Such works typically employ and modify a source coder, while our approach does not involve source coding at all. Residual redundancy codes either replace channel coding entirely, or else use it in complement with iterative schemes to exploit the residual redundancy. Our strategy is to modify the extant channel code; we are particularly concerned with systems that employ fast, simple codes, where an expensive iterative joint source/channel coding scheme would not be practical.

Finally, UMP is related to the *red alert problem*, in which a specific message not only requires a small probability of missed detection, but also a small probability of false alarm [40]. In this work, we are not concerned with mitigating false alarms of our special messages.

II. Preliminaries

A code $\mathcal{C}$ is a subset of $\{0, 1, \ldots, q-1\}^n$, where $q \geq 2$ is the alphabet size and n is the code length. We set $M = |\mathcal{C}|$ to be the cardinality of the code. As usual, for linear block codes the parameter k is the code dimension (so that $M = q^k$ messages can be represented). Code $\mathcal{C}$ has minimum distance d if $d = \min_{\mathbf{x},\mathbf{y}\in\mathcal{C},\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y})$, where d_H is the Hamming distance. If $\mathcal{C}$ has minimum distance d, it can correct $t = \lfloor (d-1)/2 \rfloor$ errors. We use the standard (n, k, d) notation to denote code length, dimension, and minimum distance parameters. We use $d(\mathcal{C})$ as shorthand for the minimum distance of $\mathcal{C}$. In this paper, logarithms are base 2. When dealing with cyclic codes, let α be a primitive element in GF(2^p), $p \geq 1$, where the code length is $n = 2^p - 1$, and let $\phi_i(x)$ be the minimum polynomial of α^i.

When discussing the inputs and outputs to a channel, let $\boldsymbol{m}$ be the original message, $\boldsymbol{c}$ be the transmitted codeword, $\bar{\boldsymbol{c}}$ be the received (possibly erroneous) vector, $\hat{\boldsymbol{c}}$ be the decoded codeword, and $\hat{\boldsymbol{m}}$ be the final de-mapped message. Let $\boldsymbol{e}_i$ represent the error-locator vector with 0's at every index except index i, which has a value of 1. We use the notation $\boldsymbol{H}(i, j)$ to refer to the element on the ith row and jth column of matrix $\boldsymbol{H}$ (and $\boldsymbol{H}(:, j)$ to refer to the jth column of $\boldsymbol{H}$).

We partition the M codewords into the sets $\mathcal{M}_i$, where the codewords in $\mathcal{M}_i$ have the property that they are guaranteed to be correctable in the presence of up to (and not necessarily more than) i errors. Additionally let $M_i = |\mathcal{M}_i|$. The values of i will depend on the code at hand. For example, the UMP alternative to the extended Hamming code partitions the M codewords into the sets $\mathcal{M}_1$ and $\mathcal{M}_2$, in such a way that M_2 is maximized.

The basic approach in our UMP constructions involves the use of *subcodes*, in which every codeword in the subcode—the special codewords—is a member of a larger, overall code. All codewords not in the subcode are considered to be the normal codewords. There are two key points worth noting about the code design. First, the overall code should not be a *perfect* code, i.e., there should be received (erroneous) vectors that are not inside the Hamming sphere of any codewords. This allows us to increase the Hamming spheres around our special codewords in order to capture these erroneous vectors. For example, if our overall code is a Hamming code—a perfect code—then increasing the Hamming spheres around any choice of special codewords would necessarily eliminate the single-error-correction guarantees of some of the normal codewords. However, the *extended* Hamming code is thus a *quasi-perfect* code and is a suitable choice for the overall code. Second, the subcode property of our coding framework must be designed at the generator matrix as opposed to the parity-check matrix. With the subcode structure explicitly represented in the generator matrix, we can encode our choice of special messages in a straightforward manner. Narrow-sense BCH codes are nested (have subcodes that are also BCH codes); however, the subcode structure is traditionally explicitly embedded in the parity-check matrix.

As an initial upper bound on the number of special codewords for our UMP parameters, we use the sphere-packing bound (also known as the Hamming bound). For a code $\mathcal{C}$, the sphere-packing bound can be written as

$$|\mathcal{C}| \leq \frac{q^n}{\sum_{\ell=0}^{t} \binom{n}{\ell}(q-1)^\ell}.$$

For our purposes, we rewrite the sphere-packing bound by splitting up $\mathcal{C}$ into the different classes of codewords,

$$|\mathcal{C}| = \sum_j M_j,$$

thus yielding our modified sphere-packing bound:

$$\sum_{j:\mathcal{M}_j\neq\emptyset} M_j \sum_{\ell=0}^{j} \binom{n}{\ell}(q-1)^\ell \leq q^n.$$

Depending on the code at hand, we fix n, k, and the desired codeword partitions, in order to derive an upper bound on the number of special codewords. However, the sphere-packing bound is naïve in the sense that it does not take into account the geometry of the codespace. In Section V, we derive a more

sophisticated bound building upon Delsarte's linear programming bound [41]. Additionally, using the same rationale for a lower bound on the number of special codewords produces a modified Gilbert-Varshamov bound [42] as follows. Let us say we are focusing on class M_i and that we wish to have $M_j = \alpha_j M_i$ for $j \neq i$ be the relative sizing for our desired partition. Then, the optimal size of M_i that satisfies this partition is lower bounded as

$$M_i \geq \left\lfloor \frac{q^n}{\sum_{j\neq i} \alpha_j \sum_{\ell=0}^{2j} \binom{n}{\ell}(q-1)^\ell + \sum_{\ell=0}^{2i} \binom{n}{\ell}(q-1)^\ell} \right\rfloor.$$

Once again, this bound is loose since it does not take into account the geometry of the codespace.

III. Parity Check UMP Alternative: (sm)SEC

A. Basic Properties

The single-bit parity-check code is a simple code that ensures every codeword has even weight. As a result, the minimum distance of the code is 2 and any single-bit error is detectable. A single-bit parity-check code is often systematic, but it can also be employed as the equivalent cyclic redundancy check CRC-1 code. In our UMP alternative, we give up single-error-detection for special message single-error-correction: *(sm)SEC*. We partition the M codewords into the sets $\mathcal{M}_0$ and $\mathcal{M}_1$. Note that the codewords in $\mathcal{M}_0$ are still uniquely decodable in the presence of no errors, i.e., the code mapping is injective.

Definition 1. *A $(k+1, k)$ (sm)SEC code is a code whose codewords are partitioned into $\mathcal{M}_0$ and $\mathcal{M}_1$ with the following minimum distance properties*:

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_0, \mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y}) \geq 1, \tag{1}$$

$$\min_{\mathbf{x}\in\mathcal{M}_0, \mathbf{y}\in\mathcal{M}_1} d_H(\mathbf{x},\mathbf{y}) \geq 2, \tag{2}$$

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_1, \mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y}) \geq 3. \tag{3}$$

As an initial example, let us examine the codewords of the linear $(5, 4)$ single-bit parity-check code. We partition the codewords to transform this single-error-detection code into an (sm)SEC code. Note that the codewords themselves are the same; the partition is simply equivalent to a new decoding procedure.

Example 1. $\mathcal{C} = \mathcal{M}_0 \cup \mathcal{M}_1$ *is an* $(5, 4)$ *(sm)SEC code*:

$$\begin{aligned}\mathcal{M}_0 = \{&(00011), (00101), (00110), (01001),\\ &(01010), (01100), (01111), (10001),\\ &(10010), (10100), (10111), (11000),\\ &(11011), (11101)\},\\ \mathcal{M}_1 = \{&(00000), (11110)\}.\end{aligned}$$

Note that the partition in the above example meets the minimum distance requirements for a (sm)SEC code. Any received vector that is Hamming distance 1 away from a codeword in $\mathcal{M}_1$ will be decoded to that codeword. The expansion of the Hamming spheres around the special codewords eliminates the single-error detection guarantee for codewords in $\mathcal{M}_0$; however, note that detection is still possible in many cases, just not *guaranteed* for all cases. For example, an error is detected if the transmitted and received words are $\boldsymbol{c} = (00011)$ and $\bar{\boldsymbol{c}} = (00111)$. Additionally, note that there is no possible partition of the $(5, 4)$ single-bit parity-check code that results in an (sm)SEC code with $M_1 > 2$, i.e., in Example 1, there is no combination of three or more codewords for $\mathcal{M}_1$ that would satisfy Conditions 1-3. The previous fact can be shown by individually eliminating all possible codeword weight trios for $\mathcal{M}_1$ as being able to satisfy Condition 3. However, the following example demonstrates that we can construct a nonlinear code that has a higher number of special codewords.

Example 2. $\mathcal{C} = \mathcal{M}_0 \cup \mathcal{M}_1$ *is a* $(5, 4)$ *(sm)SEC code*:

$$\begin{aligned}\mathcal{M}_0 = \{&(11010), (11001), (10110), (10101),\\ &(01110), (01101), (10011), (01011),\\ &(10010), (01010), (10001), (01001), (11111)\},\\ \mathcal{M}_1 = \{&(00000), (11100), (00111)\}.\end{aligned}$$

To arrive at an initial upper bound on the number of possible special messages in a (sm)SEC code, we use the sphere-packing bound as follows ($|\mathcal{B}_i|$ is the size of a Hamming sphere with radius i):

$$\begin{aligned} & M_0|\mathcal{B}_0| + M_1|\mathcal{B}_1| \leq 2^n \\ \Longrightarrow\ & (2^k - M_1) + M_1(n+1) \leq 2^n \\ \Longrightarrow\ & (2^k - M_1) + M_1(k+2) \leq 2^{k+1} \\ \Longrightarrow\ & M_1 \leq \frac{2^k}{k+1}. \end{aligned} \tag{4}$$

A comparison between the sphere-packing bound and our code constructions is provided later in Table I.

B. Explicit Construction

Assume our message size, k, is a power of 2. The general strategy for this construction will be to use a shortened version of the extended Hamming code as the subcode of a single-bit parity-check code. Essentially, we are replacing some of the rows of the generator matrix for the single-bit parity-check code with those from a Hamming code, so that the submatrix and overall matrix have the desired minimum distance properties.

We begin the construction of our $(k+1, k)$ (sm)SEC code by first creating the generator matrix for the smallest Hamming code whose dimension is larger than k. A Hamming code has the parameters $(2^r, 2^r - 1)$, where r is the redundancy in bits. In our scenario, k is a power of 2, so we can convert the Hamming code parameters to be in terms of k. We set $2k = 2^r$, yielding $r = \log(k) + 1$. Thus, the Hamming code we seek has parameters $(2k - 1, 2k - \log(k) - 2)$. Let $\phi_i(x)$ be the minimum polynomial of α^i, where α is a primitive element of GF($2k$). The generator polynomial for the associated Hamming code is simply $g_1(x) = \phi_1(x)$. Let $\boldsymbol{G}_1'$ be the generator matrix whose rows are formed, as is usual with cyclic codes, by cyclic shifts of the coefficients of the generator polynomial. (Any

TABLE I
SPECIAL MESSAGES (MEASURED IN BITS)

		Sphere-Packing Bound		
Message Bits (k)	**Constructions 1 & 2**	**$(k+1,k)$ (sm)SEC**	**$(k+2,k)$ SED-(sm)SEC**	**$(k+\log(k)+2,k)$ SEC-(sm)DEC**
4	1	1.68	2.68	2
8	4	4.83	5.83	5.88
16	11	11.91	12.91	13.51
32	26	26.96	27.96	28.93
64	57	57.98	58.98	60.20

valid Hamming code generator polynomial could be used to generate $\boldsymbol{G_1'}$, but we use choose $g_1(x) = \phi_1(x)$ in order to be explicit in our construction.)

Throughout this paper, we will use the notation presented in [43] to illustrate the generator matrix of a cyclic code. Specifically, if the generator polynomial has the form $g(x) = g_0 + g_1 x + \cdots + g_r x^r$, then we represent the generator matrix as

$$\boldsymbol{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_r & & 0 \\ & g_0 & g_1 & \cdots & g_{r-1} & g_r & \\ & & & \cdots & \cdots & & \\ 0 & & g_0 & \cdots & \cdots & & g_r \end{bmatrix} = \begin{bmatrix} g(x) & & & & \\ & xg(x) & & & \\ & & & \cdots & \\ & & & & x^{n-r-1}g(x) \end{bmatrix}.$$

We now *shorten* $\boldsymbol{G_1'}$ from a $(2k - \log(k) - 2) \times (2k - 1)$ matrix to a $(k - \log(k) - 1) \times k$ matrix. This is accomplished by removing the bottom $k - 1$ rows and the right $k - 1$ columns, respectively, yielding the following generator matrix for a shortened Hamming code:

$$\boldsymbol{G_1} = \begin{bmatrix} g_1(x) & & & & \\ & xg_1(x) & & & \\ & & x^2 g_1(x) & & \\ & & & \cdots & \\ & & & & x^{k-\log(k)-2} g_1(x) \end{bmatrix}. \tag{5}$$

Now we turn our attention to the overall code. We will add an overall parity-bit at a later step, so the generating polynomial for the remaining rows is simply the identity function. At this stage, the remaining rows of the overall code are represented by

$$\boldsymbol{G_0} = \begin{bmatrix} x^{k-\log(k)-1} & & & & \\ & x^{k-\log(k)} & & & \\ & & x^{k-\log(k)+1} & & \\ & & & \cdots & \\ & & & & x^{k-1} \end{bmatrix}. \tag{6}$$

Let $\mathcal{C}_1$ and $\mathcal{C}_0$ be the codes represented by $\boldsymbol{G_1}$ and $\boldsymbol{G_0}$, respectively. At this point, we have $d(\mathcal{C}_1) = 3$ and $d(\mathcal{C}_0) = 1$, thus the addition of an overall parity bit increases each minimum distance by 1. Our final generator matrix for our (sm)SEC code is simply a vertical concatenation of matrices $\boldsymbol{G_1}$ and $\boldsymbol{G_0}$, extended with an overall parity bit. Each row of $\boldsymbol{G_1}$ has odd weight due to the properties of the minimum polynomial $g_1(x) = \phi_1(x)$, and each row of $\boldsymbol{G_0}$ has odd weight since each row is simply a monomial. Thus, the parity bit at the end of each row is always a '1'.

Construction 1. *With $\boldsymbol{G_1}$ and $\boldsymbol{G_0}$ defined in (5) and (6), respectively, we define the overall generator matrix:*

$$\boldsymbol{G} = \left[\begin{array}{c|c} \boldsymbol{G_0} & \boldsymbol{1} \\ \boldsymbol{G_1} & \boldsymbol{1} \end{array}\right].$$

Here and elsewhere in the paper, $\boldsymbol{1}$ represents a column vector of all 1's, of appropriate dimension dictated by the number of rows of the submatrix it is appended to. We place $\boldsymbol{G_0}$ above $\boldsymbol{G_1}$ so that the special mapping, explored further in Section VI, is more convenient.

Theorem 1. *Let $\mathcal{M}_1$ be the set of codewords corresponding to the set of messages that begin with $\log(k)+1$ 0's. Then, $\boldsymbol{G}$, from Construction 1, is the generator matrix for a $(k+1,k)$ (sm)SEC code.*

Proof: We prove that the three conditions in Definition 1 are satisfied when the special messages are those that begin with $\log(k) + 1$ 0's.

For special messages, any non-zero bits are entirely contained in the part of the message that multiplies $[\boldsymbol{G_1}|\boldsymbol{1}]$ in the encoding step. Since $\boldsymbol{G_1}$ is the generator matrix for a shortened Hamming code, Condition 3 is trivially satisfied.

For Condition 1 to be true, we need each of the 2^k messages to be encoded into unique codewords, i.e., if $\boldsymbol{m_1 G} = \boldsymbol{c}$ and $\boldsymbol{m_2 G} = \boldsymbol{c}$, then $\boldsymbol{m_1} = \boldsymbol{m_2}$. For this property to hold, we simply need the rows of $\boldsymbol{G}$ to be linearly independent. Individually, it is evident that $\boldsymbol{G_0}$ and $\boldsymbol{G_1}$ each have linearly independent rows. Let $\tilde{\boldsymbol{G}}$ denote $\boldsymbol{G}$ without the final column of 1's. Note that $\boldsymbol{G_0}$ can be expressed as $[\boldsymbol{0}|\boldsymbol{I}]$, where the identity matrix $\boldsymbol{I}$ has dimensions $(\log(k) + 1) \times (\log(k) + 1)$. Thus, to show that $\tilde{\boldsymbol{G}}$ has linearly independent rows, it is sufficient to show that no linear combination of rows in $\boldsymbol{G_1}$ results in a vector whose weights are entirely in the final $\log(k) + 1$ bits. For a $(k+1,k)$ (sm)SEC code, the generator polynomial in $\boldsymbol{G_1}$ is written as $g_1(x) = \phi_1(x) = 1 + x + x^{(log(k)+1)}$. Since each row is a cyclic shift of $g_1(x)$, any combination of rows necessarily contains weights that span at least $\log(k)+2$ bits. Condition 1 is satisfied since $\tilde{\boldsymbol{G}}$ has linearly independent rows (the addition of the final 1's column does not affect this property).

A generator matrix in which each row has even weight produces a code in which all codewords have even weight.

Condition 1 being true implies that distinct messages are encoded into distinct codewords, hence Condition 2 also holds since each row in $\boldsymbol{G}$ has even weight. ∎

Corollary 1. *Using $\boldsymbol{G}$ from Construction 1 with the mapping from Theorem 1, there are $2^{k-(\log(k)+1)}$ special messages, i.e., $M_1 = 2^{k-(\log(k)+1)}$.*

In order to gauge the number of special messages of an (sm)SEC code, we introduce the following definition.

Definition 2. *An (sm)SEC code, with M_1 special messages, is bitwise optimal if there does not exist an (sm)SEC code with*

$$2^{\lceil \log(M_1) \rceil + 1}$$

or more special messages.

Comparing Corollary 1 to the sphere-packing bound in Equation (4), we arrive at the following result concerning the optimality of our (sm)SEC construction.

Corollary 2. *The code in Construction 1 is a bitwise optimal (sm)SEC code.*

Proof: We calculate the difference between the maximum number of information bits for M_1 from the sphere-packing bound and the number of information bits for M_1 in our construction as follows:

$$\begin{aligned} &\log\left(\frac{2^k}{k+1}\right) - \log\left(2^{k-(\log(k)+1)}\right) \\ &= k - \log(k+1) - (k - \log(k) - 1) \\ &= \log\left(\frac{k}{k+1}\right) + 1 < 1, \end{aligned}$$

for positive values of k. ∎

As a concrete example, let us briefly walk through the construction of the $(33, 32)$ (sm)SEC code. We first construct the cyclic generator matrix for the $(63, 57)$ Hamming code. Let α be a primitive element of GF(2^6) such that $1 + x + x^6 = 0$, then our generator polynomial is simply $g_1(x) = \phi_1(x) = 1 + x + x^6$. We create the matrix $\boldsymbol{G_1}$ and then shorten the code to $(32, 26)$. Above it we add a 6×6 identity matrix, padded on the left with 0's, i.e., we concatenate $\mathbf{0}^{6\times 26}\mathbf{1}^{6\times 6}$ on top of $\boldsymbol{G_1}$. Lastly, we add a column of 1's.

C. Decoding

The decoding process for a (sm)SEC code is relatively simple. A slight caveat is that the overall code is not systematic. Thus, to retrieve $\hat{\boldsymbol{m}}$ from $\hat{\boldsymbol{c}}$ requires a de-mapping, which is accomplished by using the right pseudo-inverse of $\boldsymbol{G}$ as follows: $\hat{\boldsymbol{c}}\boldsymbol{G}^{-1} = \hat{\boldsymbol{m}}$.

The cyclic construction of $\boldsymbol{G_1}$ was helpful for the proof of Theorem 1, but in practice we convert $\boldsymbol{G_1}$ to a systematic form by using elementary row operations, and easily obtain the corresponding parity-check matrix $\boldsymbol{H_1}$. Note that we don't need an overall $\boldsymbol{H}$ since the overall code is simply a single-bit parity check.

There are three possible events in the decoding process. First, if the received vector $\bar{\boldsymbol{c}}$ has even weight, then it is a valid codeword and we declare $\hat{\boldsymbol{c}} = \bar{\boldsymbol{c}}$. Second, if the syndrome $\boldsymbol{s_1} = \boldsymbol{H}_1\bar{\boldsymbol{c}}^T$ is equal to column j in $\boldsymbol{H_1}$, then we flip the jth bit in $\bar{\boldsymbol{c}}$, i.e., $\hat{\boldsymbol{c}} = \bar{\boldsymbol{c}} + \boldsymbol{e}_j$, similar to how syndrome decoding is performed on Hamming codes. Lastly, if $\boldsymbol{s_1}$ is nonzero and is not equal to a column in $\boldsymbol{H_1}$, then we declare a DUE, i.e., there was no special message reachable from an erroneous vector with a single bit-flip. This final outcome occurs when a normal codeword is transmitted, a single-bit error occurs, and there is no special codeword within a Hamming distance of 2 from the original codeword. The following steps provide a concise summary of the decoding process for a (sm)SEC code:

Algorithm 1 Decoding algorithm for the (sm)SEC code

```
if wt(c̄) = 0 (mod 2) then
    ĉ ← c̄
else if s_1 = H_1(:, j) then
    ĉ ← c̄ + e_j
else
    Declare DUE
end if
```

D. SED-(sm)SEC

Recall that for the (sm)SEC code we give up the single-error detection guarantee. This loss in protection might cause the trade-off to be undesirable for certain systems. However, we can extend the (sm)SEC code by a single redundancy bit in order to guarantee SED for normal codewords. The minimum distance requirements for SED-(sm)SEC are as follows.

Definition 3. *A $(k+2, k)$ SED-(sm)SEC code is a code whose codewords are partitioned into $\mathcal{M}_0$ and $\mathcal{M}_1$ with the following minimum distance properties*:

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_0,\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y}) \geq 2, \tag{7}$$

$$\min_{\mathbf{x}\in\mathcal{M}_0,\mathbf{y}\in\mathcal{M}_1} d_H(\mathbf{x},\mathbf{y}) \geq 3, \tag{8}$$

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_1,\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y}) \geq 3. \tag{9}$$

Using Construction 1, from the previous subsection, we meet the above requirements with the addition of a single bit that takes the value of 1 for normal messages and a value of 0 for special messages. The redundancy bit is simple to implement as it is just the logical NOR of the first $\log_2(k)+1$ bits in the message.

Comparing Definitions 1 and 3, notice that the requirements from Conditions (1) and (2) increase by 1 while the requirement from Condition (3) remains the same. With the addition of the nonlinear redundancy bit, it is obvious that any code satisfying (2) now satisfies (8). While the nonlinear parity bit does not affect (1), our (sm)SEC code from the previous subsection already satisfies (7) as it is an even weight code.

The sphere-packing bound requires modification to be applicable to the SED-(sm)SEC. Each special codeword still has $(n+1)$ n-dimensional points within its Hamming sphere. However, each point at distance 1 away from a normal codeword is at distance 1 away from at most n normal codewords. Each of these points can be thought of as being *shared* by at

most n normal codewords. Thus, each normal codeword has a claim to at least $(1/n)n = 1$ points (not including itself). Our modified sphere-packing bound is as follows:

$$\begin{aligned} &2M_0 + (n+1)M_1 \leq 2^n \\ \implies &2(2^k - M_1) + M_1(k+3) \leq 2^{k+2} \\ \implies &M_1 \leq \frac{2^{k+1}}{k+1}. \end{aligned} \tag{10}$$

The decoding process is slightly more involved than that of the (sm)SEC code. Let $\bar{\boldsymbol{c}}$ represent the received codeword, not including the nonlinear redundancy bit, and let η represent the value of that bit. Again, let $\boldsymbol{s_1} = \boldsymbol{H_1}^T \bar{\boldsymbol{c}}$. As in the case prior, if the received codeword has even weight, then we assume no errors have occurred. Similar to before, but with an extra condition, if $\eta = 0$ and $\boldsymbol{s_1} = \boldsymbol{H_1}(:, j)$, then the received codeword is reachable from a special codeword with a single-bit error, thus we set $\hat{\boldsymbol{c}} = \bar{\boldsymbol{c}} + \boldsymbol{e}_j$. However, the addition of η allows us to detect single-bit errors from normal codewords. For example, a single-bit flip on a normal codeword guarantees either that $\eta = 1$ and $\boldsymbol{s_1} = \boldsymbol{H_1}(:, j)$, or that $\eta = 0$ and $\boldsymbol{s_1} \neq \boldsymbol{H_1}(:, j)$. The SED-(sm)SEC decoding process is as follows:

Algorithm 2 Decoding algorithm for the SED-(sm)SEC code

```
if wt(c̄) = 0 (mod 2) then
    ĉ ← c̄
else if η = 0 and s_1 = H_1(:, j) then
    ĉ ← c̄ + e_j
else
    Declare DUE
end if
```

Even though for decoding purposes it is useful to view the nonlinear redundancy bit in a unique light, it is not given a special channel and it is susceptible to a bit-flip in the same manner as any other bit in the codeword.

IV. Hamming Code UMP Alternative: SEC-(sm)DEC

The extended Hamming code is a single-error-correcting/double-error-detecting (SECDED) code. We give up the universal DED guarantee in favor of granting special codewords double-error-correction (DEC). We thus partition the M codewords into the sets $\mathcal{M}_1$ and $\mathcal{M}_2$. We call such a code a SEC-(sm)DEC (*single-error-correcting/special-message-double-error-correcting*) code. It is convenient to formally define the code in terms of the minimum Hamming distances between pairs of codewords.

Definition 4. *A SEC-(sm)DEC code is a code whose codewords are partitioned into $\mathcal{M}_1$ and $\mathcal{M}_2$ with the following minimum distance properties*:

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_1, \mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x}, \mathbf{y}) \geq 3, \tag{11}$$

$$\min_{\mathbf{x}\in\mathcal{M}_1, \mathbf{y}\in\mathcal{M}_2} d_H(\mathbf{x}, \mathbf{y}) \geq 4, \tag{12}$$

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_2, \mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x}, \mathbf{y}) \geq 5. \tag{13}$$

We present the following simple example; we use the same codewords from the $(8, 4)$ extended Hamming code, but we consider the Hamming sphere around the all-1's codeword and the all-0's codeword to have radius 2.

Example 3. $\mathcal{C} = \mathcal{M}_1 \cup \mathcal{M}_2$ *is an* $(8, 4)$ *SEC-(sm)DEC code*:

$$\begin{aligned} \mathcal{M}_1 = \{&(11100001), (10011001), (01010101), (00101101), \\ &(00110011), (01001011), (10000111), (01111000), \\ &(10110100), (11001100), (11010010), (10101010), \\ &(01100110), (00011110)\}, \\ \mathcal{M}_2 = \{&(00000000), (11111111)\}. \end{aligned}$$

Our objective is to fully partition the code into the sets $\mathcal{M}_1$ and $\mathcal{M}_2$ and maximize M_2. That is, we require that every codeword is correctable given a single error, and we seek to maximize the number of codewords that are correctable in the presence of up to two errors.

To arrive at an upper bound on the number of possible special messages we use the sphere-packing bound as follows:

$$\begin{aligned} &M_1|\mathcal{B}_1| + M_2|\mathcal{B}_2| \leq 2^n \\ \implies &(2^k - M_2)\sum_{j=0}^{1}\binom{n}{j} + M_2\sum_{j=0}^{2}\binom{n}{j} \leq 2^n \\ \implies &M_2 \leq \frac{2^n - 2^k(n+1)}{\binom{n}{2}}. \end{aligned} \tag{14}$$

The resulting bound is intuitive: there are $2^n - 2^k(n+1)$ points outside of the radius-1 Hamming spheres, which can become radius-2 Hamming spheres with the addition of $\binom{n}{2}$ points. While the SEC-(sm)DEC code is meant as a direct alternative to the SECDED code, the above bound makes sense for any code with parameters (n, k) with a redundancy level in between the respective Hamming code and $t = 2$ BCH code.

A. Explicit Construction

Once again, we assume our message length, k, is a power of 2. Thus, our SEC-(sm)DEC code has parameters $(k + \log(k) + 2, k)$. We take a similar approach to the (sm)SEC construction, but here the subcode is an extended $t = 2$ BCH code and the overall code is an extended Hamming code.

We first begin by creating the narrow-sense $t = 2$ BCH code with parameters $(2k-1, 2k-2\log(k)-3)$. The generator polynomial for the BCH code is $g_2(x) = LCM\{\phi_1(x), \phi_3(x)\}$, used to generate $\boldsymbol{G_2'}$.

Similarly to the previous case, we *shorten* $\boldsymbol{G_2'}$ from a $(2k - 2\log(k) - 3) \times (2k - 1)$ matrix to a $(k - \log(k) - 1) \times (k + \log(k) + 1)$ matrix. This is accomplished by removing the bottom $k - \log(k) - 2$ rows and the right $k - \log(k) - 2$ columns, respectively, yielding the following generator matrix for a shortened BCH code:

$$\boldsymbol{G_2} = \begin{bmatrix} g_2(x) & & & & \\ & xg_2(x) & & & \\ & & x^2g_2(x) & & \\ & & & \cdots & \\ & & & & x^{k-\log(k)-2}g_2(x) \end{bmatrix}. \tag{15}$$

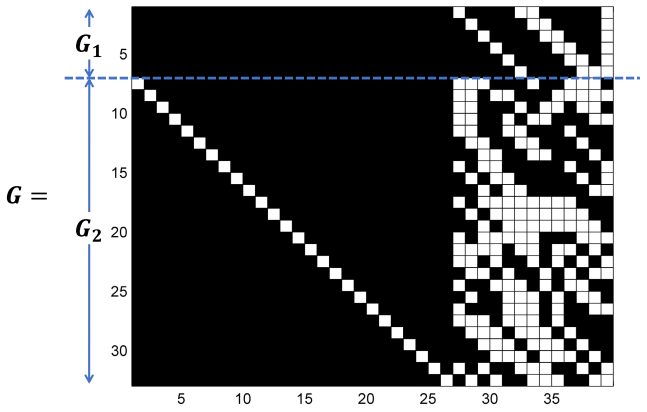

Fig. 1. The generator matrix $\boldsymbol{G}$ for our (39, 32) SEC-(sm)DEC code. The white and black squares represent 1s and 0s, respectively. Note that $\boldsymbol{G_2}$ has been converted to systematic form.

We build the additional $\log(k)+1$ rows using $g_1(x)=\phi_1(x)$, the generator polynomial for the corresponding Hamming code:

$$\boldsymbol{G_1}=\begin{bmatrix} x^{k-\log(k)-1}g_1(x) & & & \\ & x^{k-\log(k)}g_1(x) & & \\ & & \cdots & \\ & & & x^{k-1}g_1(x) \end{bmatrix}. \tag{16}$$

We combine the matrices as before:

Construction 2. *With $\boldsymbol{G_2}$ and $\boldsymbol{G_1}$ defined in (15) and (16), respectively, we define the overall generator matrix:*

$$\boldsymbol{G}=\left[\begin{array}{c|c} \boldsymbol{G_1} & \boldsymbol{1} \\ \boldsymbol{G_2} & \boldsymbol{1} \end{array}\right].$$

An example of Construction 2 is shown in Figure 1, with $\boldsymbol{G_2}$ converted to systematic form for easier usage in practical systems. As with the (sm)SEC case, we have the following theorem and lemma.

Theorem 2. *Let $\mathcal{M}_2$ be the set of codewords corresponding to the set of messages that begin with $\log(k)+1$ 0's. Then, $\boldsymbol{G}$, from Construction 2, is the generator matrix for a $(k+\log(k)+2, k)$ SEC-(sm)DEC code.*

Proof: Similarly to the proof of Theorem 1, we need to prove that the conditions in Definition 4 are satisfied when the special messages are those that begin with $\log(k)+1$ 0's. Since $\boldsymbol{G_2}$ is the generator matrix of a shortened $t=2$ BCH code, it is trivially true that Condition 13 is true.

Once again, let $\tilde{\boldsymbol{G}}$ denote $\boldsymbol{G}$ without the final column of 1's. Recall that $g_1(x)=\phi_1(x)$ and $g_2(x)=LCM\{\phi_1(x)\phi_3(x)\}$. Since $\phi_1(x)$ and $\phi_3(x)$ are irreducible and distinct, we have $g_2(x)=\phi_1(x)\phi_3(x)$. Thus, a vector is a codeword of $\tilde{\boldsymbol{G}}$ (in polynomial form) if and only if it is divisible by $\phi_1(x)$; hence, $\tilde{\boldsymbol{G}}$ is the generator matrix for a shortened Hamming code with minimum distance 3 and Condition 11 is satisfied. The addition of the column of 1's makes every row have even weight, and thus Condition 12 is also true. ■

Corollary 3. *Using $\boldsymbol{G}$ with the mapping from Theorem 2, there are $2^{k-(\log(k)+1)}$ special messages, i.e., $M_2=2^{k-(\log(k)+1)}$.*

B. Decoding

As with the previous code, we focus on decoding $\hat{\boldsymbol{c}}$ from the received vector $\bar{\boldsymbol{c}}$; an additional de-mapping step with the pseudo-inverse of $\boldsymbol{G}$ is required to arrive at $\hat{\boldsymbol{m}}$. We convert $[\boldsymbol{G_2}|\boldsymbol{1}]$ into systematic form, using elementary row operations, to easily retrieve the associated parity-check matrix, $\boldsymbol{H_2}$. Additionally, we convert $\boldsymbol{G}$ into systematic form to retrieve the overall parity-check matrix $\boldsymbol{H}$. Note that converting $\boldsymbol{G}$ to systematic form destroys the explicit subcode partition in $\boldsymbol{G}$; however, as shown in Algorithm 3, the parity-check matrix $\boldsymbol{H}$ is used to correct single-bit errors (decoding to a normal or special codeword), while $\boldsymbol{H_2}$ is used to correct double-bit errors (decoding only to a special codeword).

Let $\boldsymbol{s}=\boldsymbol{H}^T\bar{\boldsymbol{c}}$ and $\boldsymbol{s_2}=\boldsymbol{H_2}^T\bar{\boldsymbol{c}}$. The following pseudocode outlines the logical flow of the decoding process.

Algorithm 3 Decoding algorithm for the SEC-(sm)DEC code

```
if s = 0 then
    ĉ ← c̄
else if s = H(:, j) then
    ĉ ← c̄ + e_j
else if s_2 = H_2(:, j) + H_2(:, i) then
    ĉ ← c̄ + e_j + e_i
else
    Declare DUE
end if
```

The process above outlines the correct order of steps in the decoding process. There are a variety of physical implementations and algorithms to choose from for the BCH decoding process for the step involving $\boldsymbol{H_2}$.

C. SECDED-(sm)DEC

As in Section III-D, we can use an additional nonlinear parity bit to create a code strictly better than the base code, i.e., not giving up the double-error detection guarantee for any codewords. The minimum distance requirements for the SECSED-(sm)DEC code are as follows.

Definition 5. *A $(k+\log(k)+3, k)$ SECDED-(sm)DEC code is a code whose codewords are partitioned into $\mathcal{M}_1$ and $\mathcal{M}_2$ with the following minimum distance properties*:

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_1,\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y})\geq 4, \tag{17}$$

$$\min_{\mathbf{x}\in\mathcal{M}_1,\mathbf{y}\in\mathcal{M}_2} d_H(\mathbf{x},\mathbf{y})\geq 5, \tag{18}$$

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{M}_2,\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x},\mathbf{y})\geq 5. \tag{19}$$

Using Construction 2, we meet the above requirements with the addition of a single bit, which we denote as η, that takes the value of 1 for normal messages and a value of 0 for special messages. As before, the redundancy bit is simple to implement as it is just the logical NOR of the first $\log_2(k)+1$ bits in the message. The nonlinear parity-bit affects only the distances between normal and special codewords, thus only Condition 18 is different than before; Conditions 17 and 19

were already satisfied by our original Construction 2. The decoding procedure is very similar to Algorithm 3.

Algorithm 4 Decoding algorithm for the SECDED-(sm)DEC code

```
if s = 0 then
    ĉ ← c̄
else if s = H(:, j) then
    ĉ ← c̄ + e_j
else if η = 0 and s_2 = H_2(:, j) + H_2(:, i) then
    ĉ ← c̄ + e_j + e_i
else
    Declare DUE
end if
```

Using the above decoding algorithm, the SECDED-(sm)DEC protection properties of the code still hold even if η is one of the bits in error.

V. Upper Bound on Special Codewords

We first recap our current results with Table I, which compares the number of special messages for our constructions with their respective sphere-packing bounds. Each row is indexed by a value of k, and the second column represents the results from Corollaries 1 and 3. The third column helps to demonstrate Corollary 2, that the (sm)SEC code is bitwise optimal.

For traditional codes, the sphere-packing bound is not the tightest upper bound available in either the finite-length or asymptotic regimes. A better bound is provided in both cases by Delsarte's linear programming (LP) bound [41]. The LP bound considers the *distance distribution* vector $\mathbf{g} = (g_0, g_1, g_2, \ldots, g_n)$, where

$$g_i = |\{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{C}, d_H(\mathbf{x}, \mathbf{y}) = i\}|/|\mathcal{C}|.$$

All values of g_i are nonnegative, $g_0 = 1$, and $g_i = 0$ for $1 \leq i < d_{min}$. The remaining condition in the LP bound is $\mathbf{gQ} \geq 0$, where $\mathbf{Q}$ is the so-called *second eigenmatrix* of the Hamming association scheme on $\mathbb{F}_2^n$. Delsarte showed that $\mathbf{Q}$ can be formed by the relation $\mathbf{Q}_{i,j} = K_j(i)$, with the Krawtchouk polynomial defined as

$$K_k(x) = \sum_{j=0}^{k} (-1)^j \binom{x}{j}\binom{n-x}{k-j}. \tag{20}$$

Clearly, we have that $\sum_{i=0}^{n} g_i = |\mathcal{C}|$, and thus maximizing $\sum_{i=0}^{n} g_i$ also maximizes the size of the code.

We are ready to introduce our modified programming bound for UMP codes that are partitioned into two classes of codewords. The core idea is to use multiple distance distribution vectors to represent the distances within and between codeword partitions.

Theorem 3. *Let $\mathcal{C}$ be a binary code whose codewords are partitioned into normal codewords, $\mathcal{N}$, and special codewords, $\mathcal{S}$, with the following minimum distance properties*:

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{N},\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x}, \mathbf{y}) \geq D_1, \tag{21}$$

$$\min_{\mathbf{x}\in\mathcal{N},\mathbf{y}\in\mathcal{S}} d_H(\mathbf{x}, \mathbf{y}) \geq D_2, \tag{22}$$

$$\min_{\mathbf{x},\mathbf{y}\in\mathcal{S},\mathbf{x}\neq\mathbf{y}} d_H(\mathbf{x}, \mathbf{y}) \geq D_3. \tag{23}$$

We define distance distribution vectors $\mathbf{a}, \mathbf{b}$, *and* $\mathbf{c}$ *to contain distances between two normal codewords, one normal codeword and one special codeword, and two special codewords, respectively*:

$$\begin{aligned} a_i &= |\{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{N}, d_H(\mathbf{x}, \mathbf{y}) = i\}|/|\mathcal{C}|, \\ b_i &= |\{(\mathbf{x}, \mathbf{y}) : \ \mathbf{x} \in \mathcal{N}, \mathbf{y} \in \mathcal{S} \text{ or} \\ &\qquad \mathbf{x} \in \mathcal{S}, \mathbf{y} \in \mathcal{N}, d_H(\mathbf{x}, \mathbf{y}) = i\}|/|\mathcal{C}|, \\ c_i &= |\{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{S}, d_H(\mathbf{x}, \mathbf{y}) = i\}|/|\mathcal{C}|. \end{aligned}$$

Then, we have

$$|\mathcal{S}| \leq \sqrt{2^k \textstyle\sum_{i=0}^{n} c_i^*},$$

where $\mathbf{c}^*$ *is the solution to the following nonlinear program*:

maximize: $\sum_{i=0}^{n} c_i$ *subject to:*

Inequality Constraints	*Equality Constraints*
$\mathbf{a} \geq 0$,	$a_i = 0, \ \ 1 \leq i \leq D_1 - 1$,
$\mathbf{b} \geq 0$,	$b_i = 0, \ \ 0 \leq i \leq D_2 - 1$,
$\mathbf{c} \geq 0$,	$c_i = 0, \ \ 1 \leq i \leq D_3 - 1$,
$\mathbf{aQ} \geq 0$,	$a_0 + c_0 = 1$,
$\mathbf{cQ} \geq 0$,	$\sum_{i=0}^{n}(a_i + b_i + c_i) = 2^k$,
$(\mathbf{a}+\mathbf{b}+\mathbf{c})\mathbf{Q} \geq 0$,	$2^k(a_0)^2 - \sum_{i=0}^{n} a_i = 0$,
	$2^k(c_0)^2 - \sum_{i=0}^{n} c_i = 0$.

Proof: This proof largely follows that from Delsarte's LP bound [41]; however, due to the multiple distance distribution vectors, there are a number of substantial differences in the constraints of our programming bound.

The overall goal of this program is to maximize $|\mathcal{S}|$. Summing over all the entries in $\mathbf{c}$, we have:

$$\sum_{i=0}^{n} c_i = \frac{|\mathcal{S}|^2}{|\mathcal{C}|} \implies |\mathcal{S}| = \sqrt{2^k \sum_{i=0}^{n} c_i},$$

and thus, for given n and k, our objective function is to maximize $\sum_{i=0}^{n} c_i$.

We first establish *inequality* constraints. Note that $\mathbf{a}$, $\mathbf{c}$, and $\mathbf{a} + \mathbf{b} + \mathbf{c}$ are valid (scaled) distance distribution vectors of codes. Thus, our first three inequality constraints are $\mathbf{aQ} \geq 0$, $\mathbf{cQ} \geq 0$, and $(\mathbf{a}+\mathbf{b}+\mathbf{c})\mathbf{Q} \geq 0$, where $\mathbf{Q}$ is the same eigenmatrix based on the Krawtchouk polynomial in Equation 20. Similarly to the LP bound, we require all a_i, b_i and c_i to be nonnegative.

We now establish the *equality* constraints. We have $\sum_{i=0}^{n} b_i = \frac{2|\mathcal{N}||\mathcal{S}|}{|\mathcal{C}|}$. Thus our total codewords condition is:

$$\sum_{i=0}^{n}(a_i + b_i + c_i) = \frac{|\mathcal{N}|^2 + 2|\mathcal{N}||\mathcal{S}| + |\mathcal{S}|^2}{|\mathcal{C}|} = \frac{|\mathcal{C}|^2}{|\mathcal{C}|} = 2^k.$$

Due to the minimum Hamming distances in the distribution vectors, we have $a_i = 0$ for $1 \leq i \leq D_1 - 1$, $b_i = 0$ for $0 \leq i \leq D_2 - 1$ and $c_i = 0$ for $1 \leq i \leq D_3 - 1$. Additionally, since $a_0 = |\mathcal{N}|/|\mathcal{C}|$ and $c_0 = |\mathcal{S}|/|\mathcal{C}|$, we have that $a_0 + c_0 = 1$.

Unfortunately, while the condition $a_0 + c_0 = 1$ is necessary, it is not specific enough to guarantee a solution consistent with our distribution vector definitions. We require an extra condition on a_i and c_i, as follows:

$$a_0 = \frac{|\mathcal{N}|}{2^k} \implies 2^k(a_0)^2 - \sum_{i=0}^{n} a_i = 0,$$

$$c_0 = \frac{|\mathcal{S}|}{2^k} \implies 2^k(c_0)^2 - \sum_{i=0}^{n} c_i = 0.$$

■

The final two equality constraints are not affine, and thus our program is no longer a convex optimization. However, given the smoothness of our quadratic constraints, there are many efficient optimization techniques for this nonlinear program (NLP) [44]. The NLP bound correctly returns *infeasible solution* for any parameters (n, k) with less redundancy than the associated Hamming code. Additionally, for any (n, k) with more redundancy than the associated $t = 2$ BCH code, the program correctly returns $M_2 = 2^k$. Note that only the first three equality constraints are dependent on the specific UMP code. In the special case that $D_1 = 1$, as is the case with the (sm)SEC code, a_i is never forced to be 0 since there are no values of i that satisfy the constraint $1 \leq i \leq D_1 - 1 = 0$.

Unlike the relationship between Delsarte's LP bound and the sphere packing bound, our NLP bound is not always at least as strong as the analogous sphere packing bound. Our NLP bound does not improve on the sphere-packing bound when we use the minimum number of redundancy bits required for our UMP constructions. However, our NLP bound often results in tighter bounds with the usage of additional redundancy bits. For example, with $k = 16$ message bits, the optimal SECDED code has parameters $(22, 16, 4)$, and the optimal DEC code has parameters $(26, 16, 5)$. Codes with lengths in between these are largely unexplored since the minimum distance of the code cannot increase from 4 to 5. However, since we are interested in more than just the overall minimum distance of the code, it is useful to obtain bounds on the number of special messages for these code parameters as well. Table II provides the NLP results for the SEC-(sm)DEC codes with parameters in between SECDED and DEC for $k = 8$ and $k = 16$.

TABLE II
SEC-(SM)DEC UPPER BOUNDS ON $\log(M_2)$

(n, k)	Sphere-Packing Bound	Programming Bound
$(14, 8)$	7.11	7.00
$(15, 8)$	8.09	7.99
$(23, 16)$	14.72	14.56
$(24, 16)$	15.74	15.56
$(25, 16)$	16.90	16.00

VI. Special Mapping Strategies and Results in Random-Access Memories

Now that we have established the code constructions and bounds, we switch our focus toward their practical usage in real-life systems. We have designed our UMP codes to function as a black box—the user does not need to know the intricate details of the error-correction mechanisms. However, the user is responsible for a pre-mapping of the messages that are to be designated special. As stated in Theorems 1 and 2, the messages that will be treated as special, for all of our UMP codes presented here, are those messages that start with $\log(k) + 1$ 0's. Thus, the exact method used for the pre-mapping is dependent on the underlying data and the desired special messages. In terms of simplicity, the best-case scenario is that the underlying data is often lead-padded with 0's so no mapping has to be done (see Table III). The worst-case scenario is structureless data, in which case a look-up table would be needed to store a paired list containing the messages to be deemed special and messages that begin with $\log(k) + 1$ 0's that we do not wish to be special. However, data or instructions stored in memory are generally structured, so we can use clever techniques to specially encode large sets of messages instead of individually

Data in memory is usually low-magnitude signed or unsigned data of a certain data type. These low magnitude values get inefficiently represented by fixed size data type, for e.g., a 4-byte integer type used to represent values that usually need only 1-byte. This means in most cases the MSBs would be a leading pad of 0's or 1's. Also, frequencies of instructions in most applications follow a power law distribution [5]; some instructions are much more frequently accessed than the other instructions. If the *opcode*, which primarily determines the action taken by the instruction for a certain instruction set architecture (ISA), is for example, the first x bits, then the relative frequency of the opcodes of the common instructions are high. Thus, most instructions in the memory would have the same prefix of x-bits.

We collected dynamic memory access traces of various benchmarks that were compiled for both the 64-bit and 32-bit RISC-V instruction sets v2.0 and analyzed them to determine the most frequent opcodes (in instruction memory) and the relative frequency of common patterns (in data memory) over the entire suite; the results are shown in Table III. For both the 64-bit and 32-bit RISC-V ISAs, the opcode is 7 bits long and occurs in bit-positions 0-6. We find that the distribution of opcodes is highly asymmetric—the two most frequent instructions, `LOAD` and `OP-IMM` [45], comprise an average of 51% and 56% of the instructions in the AxBench [46] and SPEC CPU2006 suites, respectively. For data memory the majority of stored vectors begin with a run of 0's consistently throughout each benchmark (as demonstrated by the low variance values).

Due to the popularity of $(39, 32)$ SECDED codes in byte-oriented architectures, we seek a $(39, 32)$ SEC-(sm)DEC coding framework that efficiently maps special messages of our choice to special codewords. For a $(39, 32)$ SEC-(sm)DEC

TABLE III
FRACTION OF SPECIAL MESSAGES PER BENCHMARK WITHIN SUITE

	32-bit Architecture				64-bit Architecture			
	Most Freq. 2 opcode (Instruction Memory)		First 6 bits are 0 (Data Memory)		Most Freq. opcodes (Instruction Memory)		First 7 bits are 0 (Data Memory)	
Suite	Max	Mean	Max	Mean	Max	Mean	Max	Mean
AxBench	0.51	0.46	0.92	0.86	0.27	0.26	0.89	0.82
SPEC CPU2006	0.56	0.37	0.99	0.89	0.31	0.22	0.99	0.60

code formed via Construction 2, Lemma 3 yields $\log(M_2) = 26$, i.e., we can have 2^{26} special messages. Given the structure of the underlying data, there are two natural choices for our special messages. First, the 32-bit RISC-V ISA is comprised of 7 bits for the opcode and 25 bits for the rest of the message, therefore, we are able to offer DEC protection to 2 opcodes and all of their associated messages. The messages containing either of the opcodes that we be deemed special would simply need to be mapped/swapped with 0000000 and 0000001. This swapping would occur prior to the encoding process, and once again after the decoding process. An alternative strategy to focusing opcodes is to focus on data with a leading run of zeros, since this is a very common pattern. Again, since we have $\log(M_2) = 26$, we can offer full DEC protection to any message beginning with a run of 0's of length at least $32 - 26 = 6$ bits. We can apply the same analysis for the $(72, 64)$ SEC-(sm)DEC code, for which Lemma 3 yields $\log(M_2) = 57$. Since $64-57 = 7$, we can offer DEC protection to the single most likely opcode (and the associated messages), or alternatively, to any message whose first 7 bits are 0's.

Using the same number of redundancy bits as the SECDED code, our SEC-(sm)DEC coding scheme offers full DEC protection to special messages based on a customizable mapping scheme. Depending on the user goal, our construction could lead to system-level benefits such as less frequent checkpoints in supercomputers and decreased risk of catastrophic failure from erroneous special messages. Our results indicate that the $(39, 32)$ SEC-(sm)DEC scheme can improve the overall failure rate (in systems where DUEs are critical) by up to 9x with no additional redundancy using the leading run of 0's mapping technique.

Implementing SEC-(sm)DEC coding would require changes to the hardware that already supports SECDED. The encoding latency and energy when writing to the memory are almost identical for the two protection schemes. The decoding requires an additional clock cycle for the non-special messages in the case of SEC-(sm)DEC. This is because for SEC-(sm)DEC, the first 6-bits of a 32-bit message is non-systematic. For special messages, this first 6-bits is the special prefix that is known and hence, the trailing systematic 26-bits can simply be truncated from the received message when there is no error and the special prefix can be added to construct the original message. However, for non-special messages the first 6-bits is not known and hence, the entire received codeword needs to go through an additional cycle of matrix multiplication to retrieve the original message, incurring an additional cycle latency during decoding.

To understand the performance impacts of the proposed codes and the additional cycle latency due to non-systematic non-special messages, we evaluated SED-(sm)SEC in last level cache (LLC) over applications from the SPEC 2006 benchmark suite and compared it against LLC with SECDED code. The processor is a lightweight single in-order core architecture with a 32kB L1 cache for instruction and 64kB L1 cache for data. Both the instruction and data caches are 4-way associative. Since SED-(sm)SEC has 3.5x lower redundancy storage overhead, for the same area, it allows for a larger capacity LLC ($\sim$ 10% larger) than SECDED. Hence, for the system with SED-(sm)SEC protected LLC, the size of the LLC is 1152kB while the system with SECDED protected LLC has 1024kB of last level cache. The increased capacity of LLC results in fewer cache misses during read/write operation which helps in improving the overall system performance. For LLC with SED-(sm)SEC, the non-systematic non-special messages have one extra cycle during read/write operations from/to the LLC that was taken into consideration for our simulation.

From the results shown in Figure 2, it can be seen that the system with SED-(sm)SEC has up to $\sim$4% better performance (lower execution time) than the one with SECDED. The applications showing higher performance benefits are mostly memory intensive. This is because even though SED-(sm)SEC has slightly higher average cache access latency (due to the non-systematic non-special messages), it gets more than offset by the increased cache hit rate due to the higher LLC capacity coming from the lower storage overhead of SED-(sm)SEC.

We also evaluated the impact of loss of guaranteed protection on approximation-tolerant applications. The (sm)SEC code is expected to correct single-bit errors in special messages while any single-bit error in non-special messages goes undetected and hence un-corrected. The approximation friendly applications are expected to tolerate most of the single-bit undetected errors and have minimal (benign) impact on the output. However, (sm)SEC is expected to result in fewer crashes/hangs as compared to SED since it has the ability to correct single-bit flips in special messages. To evaluate this we used 6 applications from AxBench [46], an approximate benchmark suite. The AxBench benchmarks were compiled for the open-source 64-bit RISC-V (RV64G) instruction set v2.0 [45] using the official tools [47]. Each benchmark was ran until completion 1000 times on top of the RISC-V proxy kernel [48] using the Spike simulator [49] that was modified to produce representative memory access traces.

For each run, a single bit error was randomly injected on a demand data memory read. In case of non-special messages in (sm)SEC, the program continued with the wrong message. For SED, even though all single-bit errors were detected, the program continued with the wrong message

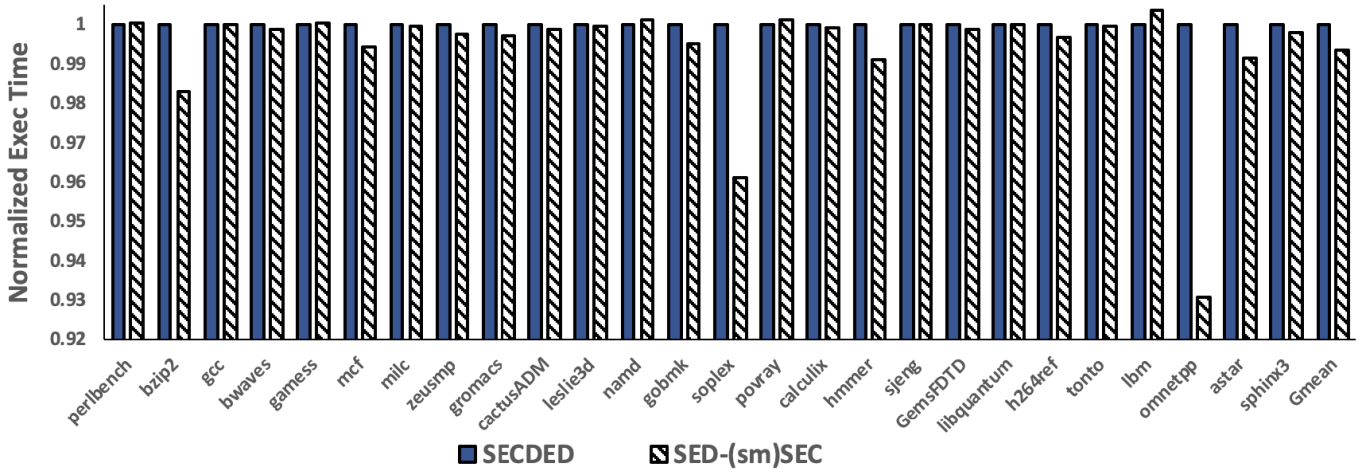

Fig. 2. Comparing Normalized Execution Time of SPEC 2006 benchmarks with SECDED protected and SED-(sm)SEC protected last level caches.

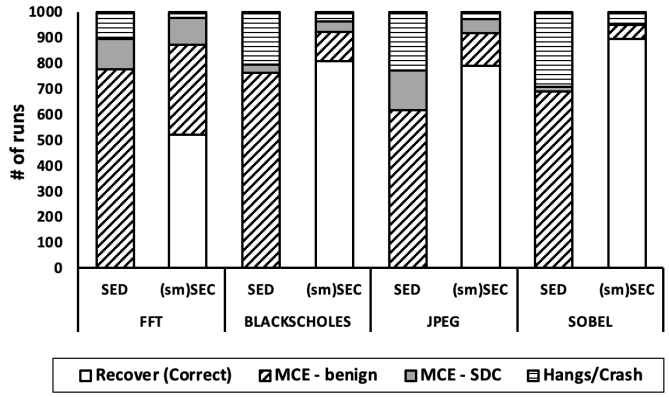

Fig. 3. Output Quality of AxBench benchmarks for memory with SED vs with (sm)SEC.

instead of crashing immediately since these applications are approximation-tolerant. From the results shown in Figure 3, it can be seen that (sm)SEC reduces intolerable Silent Data Corruption (SDC), that is, an SDC with more than 10% output error, by up to 84.2% (avg. 32.5%). It significantly reduces the number of crashes/hangs by up to 95.3% (avg. 85.6%). This means with (sm)SEC the system will have many fewer hangs/crashes in case of unpredictable single bit flips during runtime.

VII. Conclusion

Unequal message protection codes are unique in that they offer varying protection levels to different messages (as opposed to information contained in specific bit positions). Messages that are appropriate for extra protection include frequently occurring messages and critical messages. This UMP framework is an effective alternative to our previously proposed Software-Defined Error-Correcting Codes, which is a class of heuristic recovery techniques. Instead of probabilistically decoding from a set of candidate codewords, our UMP framework allows us to *a priori* select messages that receive extra protection and map them to special codewords, ensuring that no two special messages are confusable given a small number of bit errors.

In this paper, we explored a novel class of UMP codes along with potential applications. After establishing the notation, we provided explicit constructions for four specific UMP codes. Two of these codes, the (sm)SEC code and the SEC-(sm)DEC code, provide direct alternatives to two widely used codes—the single-bit parity-check code and the extended Hamming code, respectively. The other two codes, the SED-(sm)SEC code and the SECDED-(sm)DEC code, enable new possibilities in a previously unexplored redundancy space in which the additional redundant bit does not provide for an increase to the overall Hamming distance. In addition to the explicit constructions, we proved that the (sm)SEC code is bit-wise optimal and provided both a modified sphere-packing bound as well as a nonlinear programming upper bound on the number of special codewords given the UMP code parameters.

Lastly, we conducted extensive simulations of these codes in real-world benchmarks. With very simple encoding schemes, we were able to denote large percentages of real messages as *special* in both instruction memory and data memory. Additionally, these extra protection levels had minimal impact on the latency while improving overall system resiliency.

There are many paths for future work with UMP codes. Tighter bounds on the number of special codewords for a given UMP code parameter set have yet to be discovered. Additionally, it is likely that similar techniques to those presented in this paper can be used to construct UMP codes that have many levels of protection, as opposed to just two. Future work also non-binary UMP constructions which would serve as an alternative to Chipkill [22].

References

[1] C. Schoeny, F. Sala, M. Gottscho, I. Alam, P. Gupta, and L. Dolecek, "Context-aware resiliency: Unequal message protection for random-access memories," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Kaohsiung, Taiwan, Nov. 2017, pp. 166–170.

[2] I. Alam, C. Schoeny, L. Dolecek, and P. Gupta, "Parity++: Lightweight error correction for last level caches," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Luxembourg City, Luxembourg, Jun. 2018, pp. 114–120.

[3] M. Gottscho, C. Schoeny, L. Dolecek, and P. Gupta, "Software-defined error-correcting codes," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshop*, Toulouse, France, Jun./Jul. 2016, pp. 276–282.

[4] M. Gottscho, "Opportunistic memory systems in presence of hardware variability," Ph.D. dissertation, Dept. Elect. Eng., Univ. California, Los Angeles, Los Angeles, CA, USA, Jun. 2017.

[5] M. Gottscho *et al.*, "Software-defined ECC: Heuristic recovery from uncorrectable memory errors," Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep., Oct. 2017. [Online]. Available: https://escholarship.org/uc/item/0gt7j9qj

[6] P. Elias, "List decoding for noisy channels," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. 335, Sep. 1957.

[7] M. Sudan, "List decoding: Algorithms and applications," in *Proc. IFIP Int. Conf. Theor. Comput. Sci.*, Sendai, Japan, Aug. 2000, pp. 25–41.

[8] V. Guruswami and A. Rudra, "Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy," *IEEE Trans. Inf. Theory*, vol. 54, no. 1, pp. 135–150, Jan. 2008.

[9] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," in *Proc. ACM SIGMETRICS*, Seattle, WA, USA, vol. 37, no. 1, Jun. 2009, pp. 193–204.

[10] B. Masnick and J. Wolf, "On linear unequal error protection codes," *IEEE Trans. Inf. Theory*, vol. 13, no. 4, pp. 600–607, Oct. 1967.

[11] *Qualcomm Centriq 2400 Processor*. Accessed: Mar. 15, 2018. [Online]. Available: https://www.qualcomm.com/products/qualcomm-centriq-2400-processor

[12] J. Huynh, "White paper: The AMD athlon MP processor with 512KB L2 cache," Adv. Micro Devices, Sunnyvale, CA, USA, Tech. Rep., May 2003.

[13] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway, "The AMD opteron processor for multiprocessor servers," *IEEE Micro*, vol. 23, no. 2, pp. 66–76, Mar. 2003.

[14] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM J. Res. Develop.*, vol. 46, no. 1, pp. 5–25, Jan. 2002.

[15] P. Nikolaou, Y. Sazeides, L. Ndreu, and M. Kleanthous, "Modeling the implications of DRAM failures and protection techniques on datacenter TCO," in *Proc. ACM Int. Symp. Microarchitecture*, Dec. 2015, pp. 572–584.

[16] J. Chang *et al.*, "The 65-nm 16-MB shared on-die L3 cache for the dual-core Intel Xeon processor 7100 series," *IEEE J. Solid-State Circuits*, vol. 42, no. 4, pp. 846–852, Apr. 2007.

[17] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 397–404, Sep. 2005.

[18] I. Boyarinov and G. Katsman, "Linear unequal error protection codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 2, pp. 168–175, Mar. 1981.

[19] N. Abramson, "A class of systematic codes for non-independent errors," *IRE Trans. Inf. Theory*, vol. 5, no. 4, pp. 150–157, Dec. 1959.

[20] P. Reviriego, J. Martínez, S. Pontarelli, and J. A. Maestro, "A method to design SEC-DED-DAEC codes with optimized decoding," *IEEE Trans. Device Mater. Rel.*, vol. 14, no. 3, pp. 884–889, Sep. 2014.

[21] S. Kaneda and E. Fujiwara, "Single byte error correcting-8212; double byte error detecting codes for memory systems," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 596–602, Jul. 1982.

[22] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," *IBM Microelectron. Division*, vol. 11, pp. 1–23, Nov. 1997.

[23] S. Borade, B. Nakiboglu, and L. Zheng, "Unequal error protection: An information-theoretic perspective," *IEEE Trans. Inf. Theory*, vol. 55, no. 12, pp. 5511–5539, Dec. 2009.

[24] Y. Y. Shkel, V. Y. F. Tan, and S. C. Draper, "Unequal message protection: Asymptotic and non-asymptotic tradeoffs," *IEEE Trans. Inf. Theory*, vol. 61, no. 10, pp. 5396–5416, Oct. 2015.

[25] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.

[26] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Monterey, CA, USA, Dec. 2000, pp. 258–265.

[27] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches," Univ. Wisconsin, Madison, Madison, WI, USA, Tech. Rep. 1500, 2004.

[28] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Minneapolis, MN, USA, Sep. 2012, pp. 377–388.

[29] K. Sayood and J. C. Borkenhagen, "Use of residual redundancy in the design of joint source/channel coders," *IEEE Trans. Commun.*, vol. 39, no. 6, pp. 838–846, Jun. 1991.

[30] N. Phamdo and N. Farvardin, "Optimal detection of discrete Markov sources over discrete memoryless channels—Applications to combined source channel coding," *IEEE Trans. Inf. Theory*, vol. 40, no. 1, pp. 186–192, Jan. 1994.

[31] J. Hagenauer, "Source-controlled channel decoding," *IEEE Trans. Commun.*, vol. 43, no. 9, pp. 2449–2457, Sep. 1995.

[32] H. H. Otu and K. Sayood, "A joint source/channel coder with block constraints," *IEEE Trans. Commun.*, vol. 47, no. 11, pp. 1615–1618, Nov. 1999.

[33] R. Bauer and J. Hagenauer, "Symbol-by-symbol MAP decoding of variable length codes," in *Proc. ITG Conf. Source Channel Coding*, Munich, Germany, Jan. 2000, pp. 111–116.

[34] K. Sayood, H. H. Otu, and N. Demir, "Joint source/channel coding for variable length codes," *IEEE Trans. Commun.*, vol. 48, no. 5, pp. 787–794, May 2000.

[35] J. Kliewer and R. Thobaben, "Iterative joint source-channel decoding of variable-length codes using residual source redundancy," *IEEE Trans. Wireless Commun.*, vol. 4, no. 3, pp. 919–929, May 2005.

[36] J. Kliewer, N. Goertz, and A. Mertins, "Iterative source-channel decoding with Markov random field source mode," *IEEE Trans. Signal Process*, vol. 54, no. 10, pp. 3688–3701, Oct. 2006.

[37] Y. Wang, M. Qin, K. R. Narayanan, A. Jiang, and Z. Bandic, "Joint source-channel decoding of polar codes for language-based sources," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[38] Y. Wang, K. R. Narayanan, and A. A. Jiang, "Exploiting source redundancy to improve the rate of polar codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 864–868.

[39] P. Upadhyaya and A. A. Jiang, "LDPC decoding with natural redundancy," in *Proc. Non-Volatile Memory Workshop (NVMW)*, San Diego, CA, USA, Mar. 2017, pp. 1–2.

[40] B. Nazer, Y. Y. Shkel, and S. C. Draper, "The AWGN red alert problem," *IEEE Trans. Inf. Theory*, vol. 59, no. 4, pp. 2188–2200, Apr. 2013.

[41] P. Delsarte, "An algebraic approach to the association schemes of coding theory," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. Catholique de Louvain, Louvain-la-Neuve, Belgium, Jun. 1973.

[42] E. N. Gilbert, "A comparison of signalling alphabets," *Bell Syst. Tech. J.*, vol. 31, no. 3, pp. 504–522, May 1952.

[43] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam, The Netherlands: Elsevier, 1977.

[44] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.

[45] A. Waterman and K. Asanović, "The RISC-V instruction set manual, User-level ISA, version 2.0," DTIC, Fort Belvoir, VA, USA, Tech. Rep. UCB/EECS-2014-54, 2014, vol. 1.

[46] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Design Test*, vol. 34, no. 2, pp. 60–68, Apr. 2017.

[47] Q. Nguyen. *RISC-V Tools (GNU Toolchain, ISA Simulator, Tests)—Git Commit 816a252*. Accessed: Mar. 15, 2018. [Online]. Available: https://github.com/riscv/riscv-tools

[48] A. Waterman. *RISC-V Proxy Kernel—Git Commit 85ae17a*. Accessed: Mar. 15, 2018. [Online]. Available: https://github.com/riscv/riscv-pk/commit/85ae17a

[49] A. Waterman and Y. Lee. *Spike, a RISC-V ISA Simulator—Git Commit 3bfc00e*. Accessed: Mar. 15, 2018. [Online]. Available: https://github.com/riscv/riscv-isa-sim

Clayton Schoeny (S'09–M'19) received his Ph.D. in the Electrical & Computer Engineering Department at the University of California, Los Angeles (UCLA) where he was a recipient of the 2018 Distinguished PhD Dissertation Award in Signals & Systems. He received his B.S. (cum laude) and M.S. degrees in Electrical Engineering from UCLA in 2012 and 2014, respectively. His research interests include coding theory and information theory, and he is associated with the LORIS and CoDESS labs. He is a recipient of the Henry Samueli Excellence in Teaching Award, the 2016 Qualcomm Innovation Fellowship, and the UCLA Dissertation Year Fellowship. He is currently a Data Scientist at Fair Financial Corp.

Frederic Sala is a postdoctoral scholar in the Stanford Computer Science Department. His research interests span machine learning, data storage systems, and information and coding theory, and in particular problems related to the analysis and design of algorithms that must operate on unreliable (incomplete, noisy, corrupted) data. He received the Ph.D. and M.S. degrees in Electrical Engineering from UCLA, where he received the Outstanding Ph.D. Dissertation in Signals & Systems Award from the UCLA Electrical Engineering Department. He is a recipient of the NSF Graduate Research Fellowship.

Mark Gottscho is a Senior Hardware Engineer at Google, where he works on the architecture and microarchitecture of TPU chips for datacenter AI platforms. He received the PhD degree in Electrical Engineering from the University of California, Los Angeles (UCLA) in 2017, where all of his contributions to this work were performed. Mark has authored more than 15 papers and one US patent. He is a recipient of the 2016 Qualcomm Innovation Fellowship and the 2016 UCLA Dissertation Year Fellowship. Mark's research interests are focused on hardware acceleration, memory systems, hardware reliability, and agile ASIC design methodologies.

Irina Alam is a third year PhD student in the Electrical and Computer Engineering department at University of California, Los Angeles. She received her bachelors in Electrical and Electronic Engineering from Nanyang Technological University, Singapore in 2014 and M.S. in Electrical and Computer Engineering from UCLA in 2018. She worked at Micron Technology Inc for two years as a Product Engineer where she was involved in testing and debugging of design and manufacturing issues of NAND based memory devices. At UCLA, her primary research focus is memory fault tolerance and opportunistic memory architectures for power and performance benefits. She has been recently working on lightweight memory resilience in context of embedded/internet-of-things systems. Her interests lie in computer architecture, test, design automation, and system software.

Puneet Gupta (SM'16) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi, New Delhi, India, in 2000, and the Ph.D. degree from the University of California at San Diego, San Diego, CA, USA, in 2007. He is currently a Faculty Member with the Electrical and Computer Engineering Department, University of California at Los Angeles. He Co-Founded Blaze DFM Inc., Sunnyvale, CA, USA, in 2004 and served as its Product Architect until 2007. He has authored over 160 papers, 17 U.S. patents, a book and a book chapter in the areas of design-technology co-optimization as well as variability/reliability aware architectures. Dr. Gupta was a recipient of the NSF CAREER Award, the ACM/SIGDA Outstanding New Faculty Award, SRC Inventor Recognition Award, and the IBM Faculty Award. He led the multi-university IMPACT+ Center which focused on future semiconductor technologies.

Lara Dolecek (S'05–SM'12) is a Full Professor with the Electrical and Computer Engineering Department at UCLA, where she leads the Laboratory for Robust Information Systems. She holds a B.S. (with honors), M.S,. and Ph.D. degrees in EECS as well as an M.A. degree in Statistics, all from UC Berkeley. She received several research and teaching awards, including the 2007 David J. Sakrison Memorial Prize for the most outstanding doctoral research in the EECS Department at UC Berkeley, an NSF CAREER Award, Intel Early Career Award, IBM Faculty Award, Okawa Research Grant, and the Northrop Grumman Excellence in Teaching Award, among others. With her research group and collaborators, she is a recipient of several best paper awards, including 2018 and 2016 Best of SELSE Paper Awards, 2018 NVMW Memorable Paper Award, 2016 IEEE Data Storage Society Best Paper Award, 2015 IEEE Globecom – Data Storage Track Best Paper Award, among others. She is currently an Associate Editor for Coding Theory for IEEE TRANSACTIONS ON INFORMATION THEORY.