High-Density Image Storage Using Approximate Memory Cells *

Qing Guo[⋆] Karin Strauss[† ‡] Luis Ceze[‡ †] Henrique S. Malvar[†]

[⋆]NVIDIA [†]Microsoft Research [‡]University of Washington

Abstract

This paper proposes tailoring image encoding for an approximate storage substrate. We demonstrate that indiscriminately storing encoded images in approximate memory generates unacceptable and uncontrollable quality degradation. The key finding is that errors in the encoded bit streams have non-uniform impact on the decoded image quality. We develop a methodology to determine the relative importance of encoded bits and store them in an approximate storage substrate. The storage cells are optimized to reduce error rate via *biasing* and are tuned to meet the desired reliability requirement via *selective error correction*. In a case study with the progressive transform codec (PTC), a precursor to JPEG XR, the proposed approximate image storage system exhibits a 2.7× increase in density of pixels per silicon volume under bounded error rates, and this achievement is additive to the storage savings of PTC compression.

Categories and Subject Descriptors I.4.2 [*Image Processing and Computer Vision*]: Approximate methods; B.3.4 [*Memory Structures*]: Reliability, Testing, and Fault-Tolerance

Keywords Approximate Storage; Image Encoding; Multi-level Cells

1. Introduction

Images and video consume significant storage space in both consumer devices and in the cloud. The emergence of new applications and new devices with even greater image and video recording capabilities will only reinforce this trend. This will result in pressure on storage system capacity, which is exacerbated by replicas of the data in multiple user devices and the cloud. Luckily, this also relaxes the requirements on the quality of data stored in devices because there is always a high-fidelity copy stored in the cloud. This paper proposes an image encoding algorithm co-designed with a solid-state storage system that supports multiple reliability levels, resulting in high-density image storage. This allows users to have more images stored locally, potentially at a lower quality, all backed up by full-quality images in the cloud.

Solid-state memories with multi-level cell (MLC) capability improve storage density by subdividing the range of values in a cell into a greater number of levels, each of which being representative of a digital state. Creating more levels to store multiple bits in a single cell improves density, but it also increases the cost of storage dramatically because read and write circuits with much higher precision are required to ensure that data can be reliably read and written.

Recently proposed approximate storage techniques [27] relax the costly reliability requirements by allowing occasional errors. Examples of data that can tolerate these errors are samples of signal inputs that are already noisy, such as image sensors – an error in one pixel in a raw image affects quality in a localized manner. However, images are rarely stored in raw format long-term; instead, they are typically encoded in compressed form to save space.

Image compression techniques such as JPEG trade-off image quality against image size via quantization techniques. This quantization step is lossy, yet applied in a deterministic way so that all loss happens at encoding time. In contrast, approximate storage errors are inherently non-deterministic and accrue over time. This paper makes the observation that the latter type of error may create large distortions once the image is decoded because it has not been accounted for at encoding time. Because image encoding schemes typically transform pixels to the frequency domain (where some components matter more than others) and employ entropy encoding (lossless compression), changes in different bits in the compressed file lead to different kinds of distortion in the output image. Hence, to preserve quality in the decoded image, different bits should be subject to different levels of approximation (*i.e.*, exposed to different error rates).

An intuitive solution for reducing the error rates of an unreliable substrate is to treat it as a noisy channel and uniformly apply an error correcting code to it. This approach treats source and channel encoding independently, and is optimal when the conditions for the well-known separation

* This work was done when the author Qing Guo was an intern at Microsoft Research and a student at the University of Rochester.

ASPLOS '16 April 2–6, 2016, Atlanta, Georgia, USA.

DOI: http://dx.doi.org/10.1145/2872362.2872413

theorem apply [8, 9]. Unfortunately, at least one of these conditions does not apply when the source is image data (which in general cannot be modeled as being generated from a source with stationary statistics) or when the channel is MLC storage (where the error probability depends on the value stored and manufacturing variations). In other words, if we apply an error-correcting code (channel coding) uniformly to the entire storage space, we essentially make all cells similarly reliable, which is wasteful when storing parts of the encoded image data that can tolerate higher error rates.

This paper advocates for a cooperative design of image encoding algorithms and approximate storage systems to maximize storage density while minimizing image distortion. The key idea is to determine the relative importance of encoded bits created by the encoding algorithm, and store them into separate regions of approximate storage, each of which tuned to match the error tolerance of the bits it stores. This exploits the case that, when the conditions for the separation theorem do not apply, best compression is achieved by joint source-channel coding [8, 32].

We demonstrate our techniques via a case study of the progressive transform codec (PTC) [17] – a precursor to JPEG XR – and a phase-change memory (PCM) storage substrate. Our results show that the proposed scheme achieves extra quality versus space trade-off that is fundamentally *not* possible with the deterministic quantization in the algorithms of the JPEG family. Compared to plain PTC-encoded images stored in precise PCM cells, our proposed system increases the storage substrate density by over 2.7× with negligible quality degradation in the decoded images.

2. Background

This section presents background on image encoding, multi-level phase-change memory, and approximate storage.

2.1 Image Encoding

Image encoding algorithms use a variety of strategies to reduce the size of a raw image. These algorithms may be lossless or lossy. Often, the lossy algorithms (*e.g.*, JPEG [23]) are capable of achieving better compression rates with tolerable degradation in image quality. Hence, we focus on lossy image encoding. The specific algorithm used in this work is the progressive transform codec (PTC) [16, 17] – a precursor to the state-of-the-art JPEG XR [5]. PTC has been used in several practical applications, such as in game texture storage for Xbox games.

PTC processes the pixels in the image through typical steps including time-to-frequency transformation, quantization, coefficient mapping from a 2D array to 1D, and entropy encoding (shown in Figure 1). These steps are also used by most image codecs, including the popular JPEG [5, 17, 23]. What makes PTC unique is the use of the hierarchical lapped biorthogonal transform (HLBT) instead of the discrete cosine transformation (DCT) or wavelets, a ping-pong style scanning pattern for macroblocks, and an adaptive run-length/Golomb-Rice (RLGR) entropy encoding [17, 18].

PTC image encoding process. Figure 1 shows that PTC encodes an image in six steps: the raw image is partitioned into rectangular blocks (1), each of which is transformed into the frequency domain using the HLBT (2). Next, the HLBT frequency-domain coefficients are quantized, *i.e.*, scaled and rounded to the nearest integers (3). The quantization resolution and hence the number of bits used to represent the coefficients is determined by a target quality, typically measured by peak signal to noise ratio (PSNR). Then, coefficients with similar frequencies are spatially gathered and grouped via a 'clustering' step (4). Next, a reordering step visits the coefficient array in a hierarchical ping-pong manner (5), resulting in a vector with lower frequency coefficients clustered in the beginning and higher frequency coefficients clustered toward the end. Lower frequency coefficients tend to have greater absolute values and to be more important to image quality [15, 17]. Finally, this vector is divided into fixed-size macroblocks (MB), each of which is subsequently encoded using the adaptive RLGR algorithm (6).

The RLGR algorithm encodes all of the coefficients in a macroblock along the vertical direction, *i.e.*, *bit-plane encoding* [29]. The number of bit planes is determined by the maximum absolute value of the coefficients. The encoding process begins at the first bit plane (*e.g.*, bit plane 0 in Figure 1), and progressively encodes column-by-column toward the last bit plane (*e.g.*, bit plane 4). Notably, coefficients toward the beginning of a macroblock are likely to have greater absolute values than those toward the end of a macroblock (thanks to the clustering and the reordering steps), which translates into longer runs of *zeros* in the bit planes and results in a better compression rate. For each individual coefficient, the most important pieces of information are the *sign* and the bit position of the most significant *one* (underlined) that determine the magnitude of the coefficient. The remaining bits on the right hand side of the most significant *one* refine the corresponding coefficient value, and hence are called 'refinement bits'. Prior work observed that the entropy in the refinement bits is so high that there is little to gain in encoding them [16]. As such, refinement bits are simply appended to the coded bitstream of the macroblock.

A bit plane is encoded in one of the following three modes (Figure 1). In the 'no run' mode, the coder reads a single bit and encodes it. In the 'full run' mode, the coder finds m continuous 0s ($m = 2^k$) from the input and uses a single 0 to represent them. If the number of 0s is short for a full run ($m < 2^k$), the coder uses binary compression, $binary(m,k)$, to encode the value of m with k bits, and the coder is in 'partial run' mode. To achieve a good compression rate, the RLGR coder adaptively switches between run modes (controlled by parameter k) based on the inputs it has observed. The RLGR coder outputs 'control bits' and 'run-length' bits. The control bits inform the decoder the mode used in the encoding and

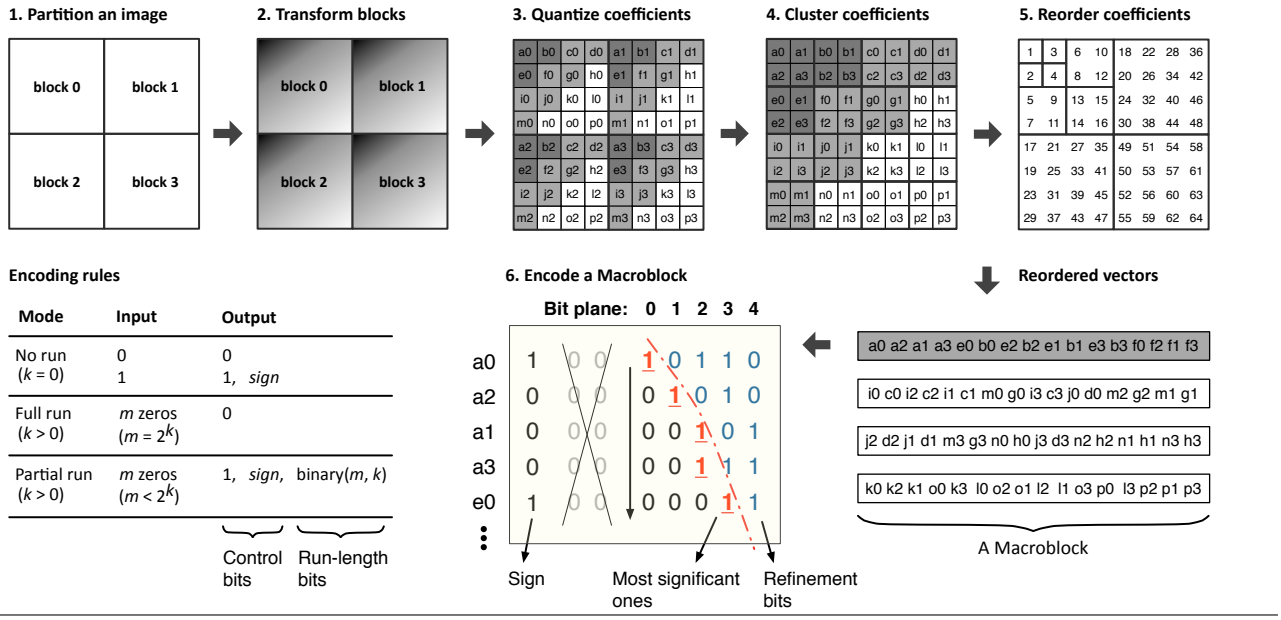

Figure 1. Simplified encoding steps in the PTC image codec.

the sign of the corresponding coefficient, and the run-length bits convey the number of 0s in a partial run. Both control and run-length bits are precision critical; an error in either a control or a run-length bit may affect the decoding of all of the subsequent bits in the bit plane, which results in the distortion of multiple spots in the decoded image.

2.2 Multi-Level Cells

Memory technologies with a large dynamic range (*e.g.*, between the minimum and maximum charges in a NAND Flash, or between the minimum and maximum resistances in emerging resistive memories) allow for dividing the analog range into more than two levels, each of the levels representing a distinct digital value. In contrast to a single-level cell (SLC) that stores binary states, a multi-level cell (MLC) encodes more than two bits into a single memory cell. Figure 2 illustrates a 4-level cell: the analog range is uniformly (in log scale) partitioned into four levels, L_0, L_1, L_2 and L_3, corresponding to binary values of '10', '11', '01', and '00', from lowest to highest resistance. A Gray code is used to minimize the Hamming distance between the codes from adjacent levels.

Phase-change memory (PCM) represents the stored information by the resistance of a chalcogenide material, which ranges from $10^3\Omega$ (in the crystalline state) to $10^{6.5}\Omega$ (in the amorphous state) [21]. An SLC PCM cell can be programmed into one of its two states by driving a write current through the chalcogenide material; the magnitude and the duration of the current pulse determine which resistance level the cell finally settles in. Programming an MLC PCM cell is more challenging because the dynamic range is partitioned into more levels, each mapping to a narrower 'band' than an SLC cell. This requires higher precision write and read circuitry. In practice, MLC PCM often employs a *write-and-verify* approach, which iteratively issues a write pulse and a subsequent checker read. The write process completes when the resistance of the cell is within the boundaries of the target level ($2T$) [3, 21]; alternatively, a write fails when the number of trials exceeds a predefined limit, which creates a write error. We model the final write resistance as a normal distribution around the target resistance of that level, as suggested in prior work [30, 33].

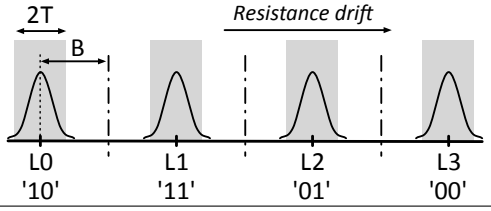

Figure 2. A 4-level cell: The analog (resistance) range (x axis) is in log scale, where levels are uniformly distributed. Write operations set cell resistance according to a normal distribution around the target resistance in the middle of the range.

A critical issue with MLC PCM is resistance drift [12, 13] – an effect of structural relaxation that causes the resistance of a cell to increase over time. Resistance drift is a major source of *retention errors*, in which a cell is written to one of the levels and, by the time it is read, it has shifted to another level (*e.g.*, an L_0 value drifts beyond $L_0 + B$ in Figure 2). The drift effect increases with the target resistance and the elapsed time from the last write. In a 4-level PCM cell like in Figure 2, for instance, level L_2 suffers the most resistance drift and often dominates the combined error rate of the cell. The drift from L_3 does not cause an error. Prior work proposes tri-level cells [30], in which the most drift-prone level (*i.e.*, L_2) is removed from a 4-level cell. Alternative solutions for mitigating retention errors include tackling

the time component of the drift, employing error-correcting codes [2, 14, 24, 36], or periodically scrubbing the entire memory [1]. Recently, IBM and Macronix proposed novel circuit techniques that allow reliable reads from 4-level and 8-level PCM cells in the presence of resistance drift [31, 34].

2.3 Approximate Storage

Approximate Storage [27] offers an alternative approach to the costly error control mechanisms. Instead of detecting errors and correcting them, an approximate storage system exposes errors to the application in a controlled manner: programmers are given the option to specify the objects that can tolerate errors. As a result, the precision-critical objects are still protected (yet in a more costly manner) whereas the error-tolerant objects are stored in memory regions with relatively low reliability for the benefits of increased density and performance.

Naive Image Mapping to Approximate Storage. As described previously, the encoded PTC bits have different importance, *i.e.*, errors in them have a different impact on the resulting decoded image. For example, an error in a low frequency coefficient affects the corresponding entire block and may be visible in the decoded image, making the quality degradation unacceptable in many cases.

If all bits of an encoded image are indiscriminately stored in unoptimized and untuned approximate cells, errors, already at a high rate due to the lack of optimization, affect all encoded bits equally. Such naive use of approximate storage for images invariably leads to either density loss due to wasteful overprovisioning of error correction resources to maintain image quality, or lower storage requirements at the risk of polluting precision-critical bits, which affects image quality.

Figure 3 illustrates an example where a 40 dB image (left) is stored in unoptimized 8-level approximate cells and is read, decoded, and reconstructed 28 seconds later (right). Despite improving density by 3×, this degrades image by about 22 dB. Without any cell optimization or error correction, maintaining the image quality (40 dB) at decode time would have required the use of a much higher target quality level (50 dB), which uses a smaller quantization factor at encode time, resulting in almost doubled encoded image size. Alternatively, error correction can be used to protect the image against errors at the cost of additional storage overheads (*e.g.*, about a third extra storage for BCH-16).

3. Effective Approximate Storage for Images

This paper advocates that bits with different importance in output image quality should be treated differently with respect to how they are stored and that cells storing these bits should be optimized and tuned to appropriate error levels. Careful matching of cell error properties with encoded bits via cooperative design of the encoding algorithm with selective error correction storage can significantly improve

Figure 3. Original encoded image (left, error rate = 10^{-16}, PSNR = 40 dB) and the same image naively stored in unoptimized and untuned approximate cells, with unacceptable quality degradation (right, error rate = 10^{-3}, PSNR = 18 dB).

image quality while getting most of the density benefit that approximate storage offers.

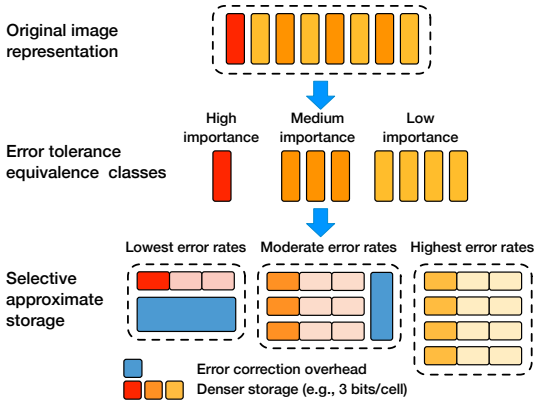

Figure 4. Example mapping of encoded bits into error tolerance classes, which are placed in storage with appropriately designed error rates.

As Figure 4 shows, we start by analyzing the PTC image encoding format (top) and identifying groups of bits with similar error rate requirements to form *error tolerance classes* (middle). Next, we optimize a multi-level PCM cell design for high density (*e.g.*, 3×) at reasonably high error rates (*e.g.*, 10^{-3}). Finally, bits in different error tolerance classes are stored by the modified algorithm into regions of a single optimized storage substrate (bottom), yet protected by appropriate error correction codes with different resulting error rates and storage overheads. The end result is high storage density with little image quality degradation.

This approach is most suitable to smartphones and other mobile devices. They make it very easy to take photos and users want to have all their pictures available to them on these devices. However, given the importance of small form-factor, the amount of solid-state storage available in these devices is limited. Since these devices are commonly backed by the cloud, it is acceptable to allow image quality degradation, as there can always be a full-quality version of the image stored in the cloud.

4. Approximation-Aware Encoding Algorithm

We analyze the PTC algorithm to identify bits in different error tolerance classes and make two observations. First, lower frequency coefficients, often higher in value, are the most important coefficients for image quality. Luckily, PTC (and other image encoding algorithms) already organizes coefficients based on the frequency which they refer to, typically from lower to higher frequency. As a result, lower frequency coefficients will be present in the first few macroblocks and can be easily mapped to memory that offers low error rates. Second, control bits affect output quality more than the run-length bits, and run-length bits affect the output quality significantly more than refinement bits. Consequently, these classes of encoded bits should be stored in memory regions of increasing density and error rates, respectively. Section 8 confirms this intuition, and Section 2.1 explained how PTC generates these types of bits. Back to Figure 4, the red shape represents control and run-length bits for the lowest frequency coefficients, the orange shapes represent control and run-length bits for other coefficients, and the yellow shapes represent refinement bits for all coefficients.

Modifications to PTC. The original PTC algorithm partitions data into macroblocks, which makes it straightforward to direct different macroblocks to different regions of storage. However, for each macroblock, PTC stores control, run-length and refinement bits in the same bitstream. During encoding, refinement bits are already segregated from control and run-length bits and appended at the end of a macroblock. However, the latter two need to be pulled apart into different bitstreams if they are to be stored into memory regions with different error characteristics. The algorithm uses a header – typically stored in the precise memory region – to record the mapping between macroblocks' bitstreams and the regions of memory that store them.

Implementation effort and quality control. We envision these modifications to be implemented by a programmer expert in image encoding who is familiar with the methodology proposed here and the various error correction options provided by the storage hardware, along with their corresponding error rates. The final algorithm is packaged in a library. This cooperative design effort may be undertaken by memory manufacturers, who ship their hardware along with supporting libraries, or by system integrators designing a product that includes digital imaging, where the library is never exposed. Final users of these products are exposed to a single knob that sets the expected quality of the image – this is exactly how users interface with lossy encoding algorithms (such as JPEG) today. This is possible because, from a user's perspective, errors caused by approximate storage are simply one more source of image quality loss.

Application to other encoding algorithms and data types. Although we use PTC as the base encoding algorithm, the same principles can be easily applied to other codecs that employ time-frequency transformation, quantization, and entropy encoding. For example, modern video codecs such as H.264 [39] apply motion estimation to predict the current video frame, and the difference frame between the original and its prediction is encoded with an image codec, which has the a similar structure to JPEG, JPEG XR, or PTC. In essence, the proposed scheme can be applied to other types of data where more important bits coexist with less important bits (*e.g.*, video, audio, and sensor data).

5. Storage Substrate Optimization

Our next goal is to optimize a PCM storage substrate to offer high density while maintaining reasonable error rates. We use a technique called *biasing* [33, 37] that, when used in combination with very low frequency scrubbing, achieves low error rates in a 4-level configuration (2 bits/cell) and reasonably low error rates in an 8-level configuration (3 bits/cell).

5.1 Biased Levels

A multi-level cell maps each of the levels within the analog range (in the case of PCM, the range between the highest and lowest resistances) onto a distinct digital value. Before biasing was proposed, each level in a multi-level PCM cell was assigned the same width in log space (Figure 2). However, such uniform partitioning of the resistance range can lead to unnecessary drift-related errors in dense approximate cells. Biasing addresses this problem by tuning the positions of the resistance levels for minimizing the combined (write and drift) error rate.

A cell suffers two types of errors. A *write error* occurs when the write circuitry is unable to set the cell resistance at the target level before exceeding a maximum number of trials, leaving the cell in an undesired resistance level. Material relaxation causes cell resistances to drift to higher resistance levels, resulting in the second type of errors – *drift errors*. Drift unidirectionally increases the cell resistance and its effect is more significant in higher resistance levels than the lower ones (Section 7). As such, the second highest level in a uniform cell (L_2 in Figure 2) becomes the most drift-prone level and dominates the drift error rate of the cell. The highest level (L_3 in Figure 2) does not suffer from drift errors because it is the highest range of resistances.

We use *biasing* to reposition and resize each resistance level (Figure 5). The drift error rate of the cell can be minimized by equalizing the drift error rate of each individual level (assuming the stored data maps to each level with an even probability). As shown in Figure 5, levels are wider in value ranges where drift is more likely to have an effect, *i.e.*, the higher resistance levels. Biasing is optimized based on a fixed elapsed time since the last write. This assumes that the system will scrub the storage contents and reset resistance levels to the target resistance at this scrubbing interval. It is worth noting that cells will work at different scrubbing

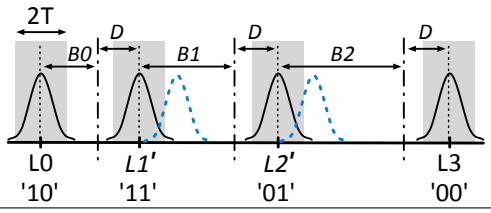

Figure 5. Cell configured with biased levels. The goal is to reduce drift-induced errors by equalizing the error rate of each individual level.

intervals, but they will suffer higher error rates compared to the interval they were optimized for because levels' error rates will not be completely equalized. To understand why this is the case, imagine an optimal biasing configuration for interval t. When this interval is extended to $t + t_\Delta$, resistances are allowed to drift longer distances. Since the drift for higher resistances is stronger than for lower ones, at interval $t + t_\Delta$ a cell optimized for interval t will experience higher error rates in higher levels. As such, to equalize error rates, the optimization needs to compensate by making the higher levels even wider than before, and the lower levels even narrower than before.

In addition, note that target resistances are no longer placed at the center of each level, dividing each level into two bands; the left band is made narrower (D) to leave more room for drift in the right band (B_i). However, as the target resistance is moved to lower values and D is reduced, the write error rate begins to increase because the tail of the write resistance distribution gets closer to the lower end of that level. The sizing of D and B_i is therefore a trade-off between write error rate and drift error rate. Errors are minimized by equalizing the number of errors of each type.

Once the resistance range is partitioned into biased levels, the next step is to map digital values to individual biased levels. Both in general and in our encoded images, *zeros* are the most common ('00' for 4-level cells and '000' for 8-level cells), so the value *zero* should be mapped to the highest level, which is immune to drift. We observe no other value that is more common than others for images, so the values for the remaining levels are assigned by using a simple Gray code.

Cell density. Based on the cell parameters we assume in this work (Section 7), we advocate for two cell configurations: 4- and 8-level cells. Notably, none of the two meet the uncorrectable bit error rate of commercial solid-state storage products today (10^{-16}) in their raw form [4, 20]; however, reasonably low error rates can be achieved via biasing and the industrial standard reliability can be reached with error correction codes. Even optimized with biasing, unfortunately, a 16-level cell suffers prohibitively high error rates (*i.e.*, the write error rate is around 10^{-4}, and the drift error rate is in the order of 10^{-1} after 1 second of writing), which cannot be brought down to a reasonable level by error correction codes that have storage overheads low enough to justify the increase in number of levels.

We use 2-level and 3-level cells as precise baselines since they both exhibit very low error rates. The 2-level cell has a simpler structure and presents a better performance in terms of read and write speed. In contrast, the 3-level cell offers higher density while maintaining a low error rate. However, additional circuits are required to convert the data from the rest of the system to the form that is stored in the 3-level cells. We use 4-level and 8-level cells as approximate memory cells.

5.2 Scrubbing to Reduce the Effect of Drift

After biasing is applied, drift may still be an issue in the long-term. To mitigate excessive drift, scrubbing can be used to rewrite a cell and bring the resistance level back down. Based on our PCM cell model, we expect the scrubbing period to be in the order of 10^7 seconds (3 months) – so infrequent that it is likely to be imperceptible. The average access bandwidth required by scrubbing at this frequency is on the order of 100 bits/second per gigabit of storage – a negligible figure. Also, if data is going to be scrubbed anyway, this may also be a good opportunity to perform wear leveling.

6. Providing Error Rate Diversity

Once cells are optimized, they can finally be tuned to provide different levels of reliability. The storage controller is responsible for offering a variety of error correction codes, each of which presenting a different trade-off between the error correction capability and the storage overhead. The complexity – and hence the area, power, and latency overheads – of the error correction circuits can be reduced by using codes within the same family (*e.g.*, BCH-4 and BCH-16).

Region configuration. The controller is responsible for organizing the storage into regions, each with a different error correction capability. The controller stores a region-to-configuration map in a table resident in the controller, which is backed by a preconfigured precise region of storage that persists during power cycles. System software sends special configuration commands to the controller for region allocation and configuration. Once configured, the controller uses the requested address and the information in the region-to-configuration map to determine which region the request targets and the appropriate error correction capability to use in servicing the request. The number of different regions is small (*e.g.*, 8), so the region-to-configuration map can support variable-size regions and be fully associative.

Hardware/software interface. The code implementing the modified algorithm must be able to allocate storage in different configurations. Assuming a storage system that is directly accessible through the processor address space, each bitstream can simply be allocated via a persistent object interface, and pointed to by the header. If a file system is used, then all storage in a block needs to be in the same configuration. The file system provides calls to specify the memory configuration when opening streams and to concatenate multiple streams to form a single file. If

no changes to the file system are possible or desirable, the multi-precision memory may be exposed as an independent volume, providing an 'image store' that maintains and serves the collection of images using approximate storage, where each image can be linked from the main file system. The operating system or the storage controller is responsible for any necessary maintenance tasks, such as wear leveling, refreshing the memory to limit degradation over time, and reconfiguring the storage after power failure.

Storage region sizing. Regions with different error correction capabilities have different storage overheads for metadata. As such, different regions will need different number of cells for storing the same amount of data. The entire storage space may be managed in one of the two ways. Static management simply partitions the storage into multiple regions at manufacturing time. This approach is inflexible in that it does not allow a different proportion of storage to be dedicated to a region. The second approach is to allow dynamic reconfiguration of regions to match application demands. In this case, region resizing causes additional complexity. Assuming the storage device leaves manufacturing with all regions initialized to the strongest available error correction by default, when a region is configured for the first time, it grows in density, and thus in usable size. A simple way to cope with this is to expose this region as two regions, one of the original size before reconfiguration, and a virtual one with the surplus storage. This makes addressing simpler. A region can only be reconfigured to a smaller size if the system can accommodate the contents of the surplus region elsewhere.

7. Experimental Setup

This section describes experimental setup, assumptions, and parameters used for this study.

Encoding algorithm. We implement our algorithm modifications on top of the PTC codec [16–18]. Besides its state-of-the-art performance, we choose PTC because of its relatively simple reference code, which makes it easier to design and implement modifications. In addition, we expect that the results we report here for PTC, in terms of the impact of bit errors on reconstructed image quality, should be similar if we implement the same modifications in other codecs. That is essentially because of the orthogonality of transforms. Developers modifying other codecs can follow a methodology similar to the one presented in this paper to identify and process most important and less important groups of bits separately.

Input images. All quality measurements are based on 24 grayscale raw images at 768×512 pixels resolution in the Kodak PCD image set [6]. These images provide diversity in types of features they include, which make them ideal for benchmarking image encoding algorithms. Color images have a color component in addition to the grayscale component. The color component is encoded in the same manner as the grayscale one and typically produces slightly more refinement bits. Higher resolution images with more pixels would have similar behavior. As such, we choose to use small grayscale images because this makes the algorithm simple to modify and experiments fast to run, yet they still provide a conservative lower bound on the benefits of cooperatively designing the algorithm and hardware.

Cell model. All experiments assume a PCM cell model with resistance range between $10^3\Omega$ and $10^{6.5}\Omega$ [21]. We use the power-law drift model from prior studies [12, 13]:

$$R_d(t) = R_w \cdot \left(\frac{t}{t_w}\right)^{\nu} \quad (1)$$

where R_w is the initial resistance at the time the write operation concludes (t_w), and R_d is the drift resistance after an elapsed time t. The drift exponent ν grows with $\log R_w$. Configurations and parameter settings for 4-level cells and 8-level cells are summarized in Table 1 and Table 2, respectively. Note that, compared to uniform cells, biased cells have target levels ($\log R_T$) and level boundaries ($\log R_B$) moved toward lower resistances by appropriate amounts, resulting in lower drift-induced errors at the cost of increased write errors (as discussed in Section 5.1). The overall drift error rate can be minimized by equalizing the drift error rates for all the levels (except for the first level and the last level). Cells are optimized for a scrubbing interval $t = 10^7$s (about 3 months) after they are written. During scrubbing, their original target resistances are restored.

We use an iterative *write-and-verify* writing mechanism [3, 38]: a *RESET* pulse followed by a series of *SET* pulses are generated to heat the PCM, with a checker read after each write trial until the desired resistance level has been reached. Each pulse has the same width (duration) but different amplitude. Equation 2 describes the in-house model that emulates the writing process:

$$R_w = R_0 + (R_w - R_0) + \sqrt{|\Delta \cdot (R_w - R_0)|} \cdot N(0,1) \quad (2)$$

where R_0 and R_w are the original and the final resistance, respectively, and N is the normal distribution. Δ measures the imprecision of the writing circuitry (in this case, 10%). Another parameter is T (shown in Figure 5). A writing process will succeed when the final resistance falls into the $2T$ range of a level, or abort after a predefined number of trials. This model generates a normal distribution of final write resistance values, with means shown in the $logR_T$ column (Table 1 and Table 2), and a standard variation of $T/3$. We use $T = 0.1$, like prior work [22, 25]. With the selected target write error rate (10^{-6}), our model shows that the maximum number of trials in a biased 4-level cell and a biased 8-level cell are seven and nine, respectively.

We model the read circuitry in Cadence SPECTRE, and conclude that a current-mode sensing mechanism (similar to that used by Bedeschi *et al.* [3]) and an analog-to-digital converter are capable of distinguishing the worst-case resistance difference for all cell configurations.

Level	Data	$\log R_T$		$\log R_B$		$\mu(\nu)$		$\sigma(\nu)$	Write BER		Drift BER	
		u	b	u	b	u	b		u	b	u	b
0	10	3.00	3.00	3.58	3.35	0.001	0.001	$0.4\mu(\nu)$	very low	2.5×10^{-6}	very low	very low
1	11	4.17	3.60	4.75	4.27	0.027	0.012				very low	very low
2	01	5.33	4.52	5.92	6.25	0.073	0.040				0.3656	very low
3	00	6.50	6.50	–	–	0.120	0.120				–	–

Table 1. Uniform (u) and biased (b) 4-level cell parameters. R_T denotes the mean resistance of a level, and R_B denotes the resistance at the upper boundary of the level. ν follows normal distribution, with $\mu(\nu)$ being interpolated from prior work [30, 38].

Level	Data	$\log R_T$		$\log R_B$		$\mu(\nu)$		$\sigma(\nu)$	Write BER		Drift BER	
		u	b	u	b	u	b		u	b	u	b
0	100	3.00	3.00	3.25	3.14	0.001	0.001	$0.2\mu(\nu)$	5.6×10^{-14}	5.6×10^{-6}	very low	very low
1	101	3.50	3.28	3.75	3.44	0.010	0.006				very low	0.0005
2	111	4.00	3.58	4.25	3.75	0.020	0.012				0.0058	0.0040
3	110	4.50	3.90	4.75	4.12	0.040	0.018				0.6774	0.0042
4	010	5.00	4.28	5.25	4.62	0.060	0.031				0.9700	0.0043
5	011	5.50	4.78	5.75	5.33	0.080	0.051				0.9960	0.0044
6	001	6.00	5.47	6.25	6.36	0.100	0.080				0.9991	0.0045
7	000	6.50	6.50	–	–	0.120	0.120				–	–

Table 2. Uniform (u) and biased (b) 8-level cell parameters.

Metrics. We evaluate the proposed scheme through a combination of two metrics: peak signal to noise ratio (PSNR) and memory density. PSNR is commonly used in the image processing community to measure the quality of reconstructed images. It compares the original image – pixel by pixel – against the decoded image that contains errors from lossy compression algorithm (*e.g.*, quantization) and memory subsystem (in this case, uncorrected write errors and drift errors). The higher the PSNR value, the smaller the difference between the original and the reconstructed images. We evaluate the approximate memory system with images from several PSNR targets, *i.e.*, 35 dB, 38 dB, 40 dB, and 42 dB. For most images, 40−42 dB range denotes high image quality, with distortion visually imperceptible; whereas, 38 dB and 35 dB represent moderate and low quality, respectively.

Memory density is defined as the average number of *data* bits stored by a cell. Error-prone memories (*e.g.*, PCM) commonly use error correction codes (ECC) to recover from certain number of errors. The storage overhead of error correction code reduces memory density.

Simulation. We conduct Monte Carlo simulations to model write-induced errors caused by circuit imprecision (Equation 2), in which a disturbance factor Δ is applied to the write target. Read-induced errors are modeled in a similar manner. Process variation is taken into account and is reflected by the parameter ν in the cell model (Equation 1, Tables 1 and 2) – the drift effect varies from cell to cell.

Due to the nondeterministic error patterns in the approximate memory system, we run each image in the benchmark through our stochastic model 100 times (errors appear in random locations every time) and report the minimum (rather than the average) PSNR for each image, which gives a conservative estimate of the quality loss.

8. Evaluation

In this section, we demonstrate the benefits of tailoring the PTC image encoding algorithm to a PCM approximate storage substrate. We start by showing the overall quality and space improvement of an optimized design over the baseline. We then show how errors affect different types of encoded bits and the resulting image quality, evaluate the benefit of optimizing the PCM substrate, and illustrate how hardware and software are tuned for best density versus image quality trade-off. We also show results obtained using the proposed algorithm and methodology on a Flash substrate.

8.1 Overall Gains

Figure 6 compares quality versus memory area of a fully optimized design and intermediate alternatives against PTC on SLC PCM. The memory area (x-axis) is measured by the number of memory cells being used, normalized to the image size. The y-axis shows the reconstructed image quality after having been encoded and stored in the approximate memory for a scrubbing interval (10^7s). As high-density memory cells often suffer higher error rates, image quality and memory area are at odds. As such, the further left a curve is, the better the quality versus memory area trade-off it represents.

From right to left, the worst performance comes from the PTC algorithm on SLC PCM (2LC), where the quality and area trade-off is solely controlled by the quantization factor; previously proposed tri-level PCM (3LC) results in a 1.58× (*i.e.*, $log_2 3$) increase in density without any quality loss from the encoded images. However, the circuitry to deal with base transformations adds complexity to the memory system interface. In comparison, the biased 4-level (Bias4LC) cell, which requires no complex base transformation circuitry, results in a higher density (1.85× over 2LC), while still maintaining the same level of image quality. The only cost of biasing is the optimization of cell levels and boundaries at design time.

Finally, we see three biased 8-level cell configurations on the left. Note that all of them use error correction because at this density the quality degrades dramatically if no error correction is used (10 dB range, which is unacceptable). The configurations shown include error correction applied thoroughly and equally to the entire storage (Bias8LC TC), selective error correction applied based on the algorithm's needs (Bias8LC SC), and an idealized correction mechanism that has no storage overhead and corrects all errors (8LC ideal). Despite gaining 2.28× in density over 2LC, the Bias8LC TC uses more correction strength than necessary to maintain high quality. In contrast, by carefully choosing the appropriate ECC scheme for each encoded bits class, Bias8LC SC achieves a density of 2.73× over 2LC (less than 10% lower then the ideal 8-level cell density gain, *i.e.*, 3×), while keeping quality loss at an acceptable level ($\leq$ 1 dB loss).

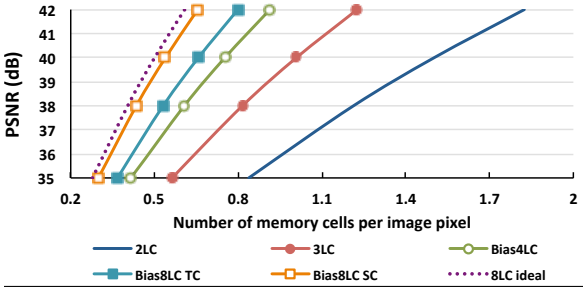

Figure 6. Overall benefits of the fully optimized design. Bias4LC, Bias8LC TC, and Bias8LC SC assume a scrubbing interval of 10^7 seconds.

8.2 Effect of Errors on Encoded Bit Types

We evaluate the impact of different coded bitstreams on the quality of reconstructed images. Our goal is to keep the quality loss within 1 dB of the encoded image. For example, a target quality of 40 dB will allow degradation only down to 39 dB.

We first evaluate the effects of control, run-length, and refinement bits across all the macroblocks. Intuitively, control bits are more precision-critical than run-length bits, then followed by refinement bits (Section 3). Figure 7 corroborates this by applying variable error rates over a subset of bits based on their types and measuring the resulting quality degradation. All images are encoded with a target quality of 40 dB. Each curve is labeled with three letters representing whether the error rate (in the x axis) is applied (A) or not (P) to each of control, run-length, and refinement bit types, respectively.

P-P-P does not suffer any failures and maintains the quality at the target 40 dB. Refinement bits affect image quality the least. As such, P-P-A can tolerate quite high error rates (this curve starts dropping only when BER reaches 10^{-4}). Next, P-A-P and P-A-A (with the introduction of run-length errors) can tolerate failure rates up to 10^{-7} with no or very little image quality degradation. Control bits are the most error-sensitive (A-P-P and A-A-A), also degrading quality quickly as failure rates go above 10^{-7}. This analysis supports the idea of adjusting error correction selectively, maintaining a lower error rate for control and run-length bits (10^{-7}), and a higher error rate (10^{-3}) for refinement bits. Due to the large gap between the error rate requirements for refinement bits and other types of bits, we fix refinement error rates at 10^{-3}, committing 0.2 dB of our 1 dB quality degradation budget.

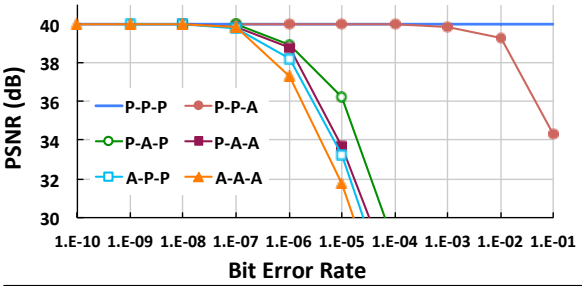

Figure 7. Image quality degradation with approximate bit streams. P denotes precise substrate, A denotes approximate substrate with BERs varying along the x-axis. The first letter represents control bits, the second represents run length bits, and the third represents refinement bits.

Next, we study the effect of different macroblocks on quality. Section 4 states that the first macroblock, which holds the lowest frequency coefficients, plays the most important role on the decoded image quality. Figure 8 verifies this statement by applying strong correction (at an error rate of 10^{-16}) to the first n macroblocks in a coded image with a total of 128 macroblocks, variable error rates for control and run-length bits in the remaining blocks, and the fixed error correction (at an error rate of 10^{-3}) for refinement bits in all blocks.

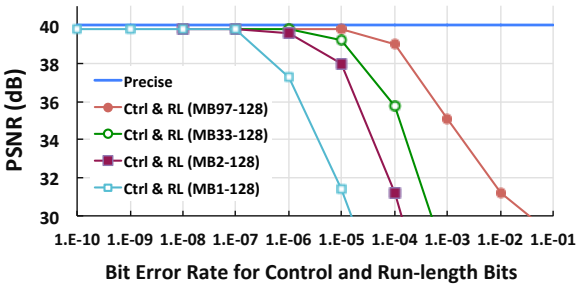

Figure 8. Image quality degradation with varying error rates in control and run-length bits. Ctrl & RL bits outside the denoted MBs use a BER of 10^{-16}, refinement bits use a BER of 10^{-3}.

As predicted, Ctrl & RL (MB2-128), which protects control and run-length bits in the first macroblock, improves quality substantially compared to Ctrl & RL (MB1-128), which leaves the first macroblock at the same error rate as other macroblocks. On the other hand, raising the precision level of additional macroblocks has diminishing returns. These results suggest that protecting the first macroblock's control and run-length bits with a strong error correction

to reach industrial standard error rates (10^{-16}), and then protecting the remaining control and run-length bits with an intermediary strength code (10^{-6}) keeps quality well within the 1 dB degradation limit.

8.3 Effect of Substrate Biasing

With target error rates for error tolerance classes in hand, we turn our attention to bringing the PCM substrate up to these standards. We start by making the substrate as good as it can possibly be for an arbitrary scrub rate ($10^7 s$, or approximately 3 months) by optimizing cells via biasing. Figure 9 illustrates the effect of biasing on error rates, for both 4-level and 8-level cells, reporting combined error rates across all levels. Error rates grow over time because of drift effects.

We initially focus on uniform cells (Uniform 4LC and Uniform 8LC). As expected, error rates for 4-level cells are always lower than for 8-level cells because fewer levels allow more room for drift in each level. However, both types of cells start showing excessively high error rates even only an hour after being written. In contrast, Biased 4LC maintains very low drift error rates during the entire range of time (10^{-20} at 10^{10}s). The raw bit error rate (RBER) of the biased 4LC is dominated by write errors. Biased 8LC provides an error rate of about 10^{-3}, which is two orders of magnitude lower than Uniform 8LC at 10^7s. Coincidentally, it also matches the needs of the most error tolerant bits (*i.e.*, the refinement bits). This allows us not to use any error correction at all for these bits, eliminating unnecessary metadata overhead.

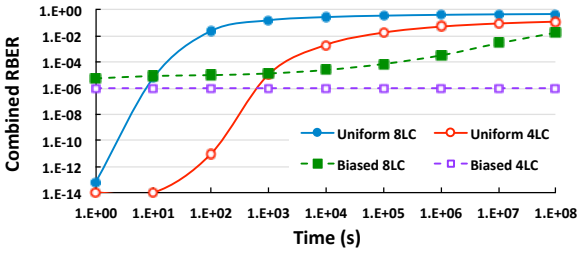

Figure 9. Combined RBERs of uniform and biased PCM cells.

Figure 10 provides insight on which cell configuration offers the best trade-off between overall density, including error correction to maintain the uncorrectable bit error rate (UBER) at the industrial standard rates (10^{-16}) [4, 20], and scrubbing overhead. SLC and 3LC cells have RBERs as low as precise memory, and hence do not require error correction. 3LC provides 1.58× higher density over SLC. The densities of uniform cells (*i.e.*, 4LC, 8LC, and 16LC), although high for short scrubbing intervals (so short that makes them unattractive), fall sharply at longer intervals, since drift-induced errors accrue fast. In contrast, biasing suppresses the growth of drift errors significantly: Bias4LC has a stable 1.86× density gain (the delta between this and the ideal 2× is caused by the ECC that protects against the write errors), and Bias8LC experiences a much smoother density degradation, achieving 2.28× density improvement at 10^7s (*i.e.*, the target scrubbing interval).

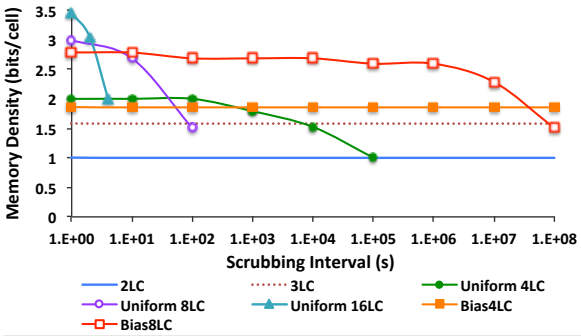

Figure 10. Density comparison (target UBER = 10^{-16}).

8.4 Assigning Error Correction to Encoded Bit Types

Once both the algorithmic error rate requirements are determined and the substrate is optimized for lowest possible error rates, we match the two via error correction. This relies on understanding the trade-offs between storage overhead of the error correction mechanism and its correcting power. Figure 11 presents a variety of error correction mechanisms (with storage overheads), and the correspondence between a variety of RBER values and their resulting UBERs when using each of these mechanisms. The biased 8LC cell already meets the demands of refinement bits, so they don't need any correction. For control and run-length bits in the first macroblock, we need a correction mechanism that accepts a RBER of 10^{-3} and produces a UBER of 10^{-16}. BCH-16 is the code that provides this capability with the lowest storage overhead (31.2%). Luckily, the bits that need such a strong error correction are only a small fraction (2.1%) of all bits. Finally, BCH-6 provides a 10^{-6} UBER at an overhead of 11.7% for the remaining bits that need any correction (81.7%).

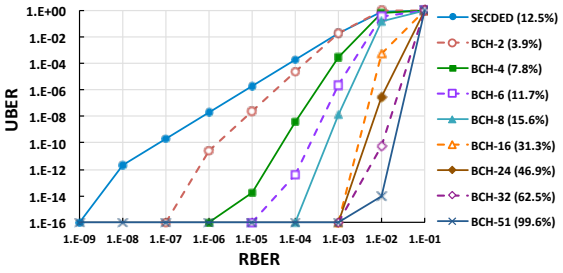

Figure 11. Capabilities and overheads of ECC (at 512 data bit blocks). SECDED corrects one error and detects up to two errors in 72 bits; each of the BCH codes corrects up to the denoted number of errors in 512 data bits plus overhead.

It is also worth noting that as RBER increases, the code strength required to maintain the same UBER grows rapidly. This highlights the value of biasing: had it not lowered the error rate by two orders of magnitude, the 8-level cell design would have offered RBER so high that the overhead of correcting all errors would have made it prohibitive.

8.5 Sensitivity to Scrubbing Period

The scrubbing period chosen for the biasing optimization is long enough to create insignificant overhead and short enough to preserve quality. This section shows what would happen if we used this cell design with other scrubbing intervals. Note that these results do not assume a cell redesign, they only show what would happen if designers decided to use these cells "out-of-spec" for different scrubbing intervals. Thus, if the interval is shorter than the specified, write errors dominate the RBER of the cell; if the interval is longer, drift errors dominate instead.

Figure 12 shows how error correction selection would have changed for different scrubbing intervals (assuming less than 1 dB quality degradation). It compares thorough correction (Bias8LC TC) with selective correction (Bias8LC SC) side-by-side at each interval. As the scrubbing interval increases (towards the right of x-axis), stronger ECC mechanisms must be employed to suppress the growth of drift errors, resulting in higher storage overheads. On the other hand, larger intervals reduce system energy and bandwidth overheads due to data movement and checker bits computation generated by the scrubbing. Although we select 10^7 seconds as the target scrubbing interval for the dense, approximate image storage system, shorter intervals might also be acceptable for other systems if higher density is the top priority.

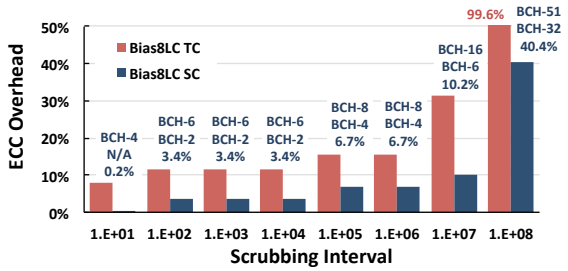

Figure 12. Trade-off between ECC storage overhead and scrubbing interval. Above each set of bars, the code in the first row is applied to all the bits in thorough correction (TC); the selective correction (SC) uses the first-row ECC for the control and run-length bits in MB1, and the second-row ECC for the control and run-length bits in other MBs, and leaves all refinement bits unprotected. The third row shows the total overhead.

The main takeaway from these results, however, is that selectively applying error correction only where needed can significantly reduce the loss in density while bringing the memory to the algorithmically-required error rates, as evidenced by the large difference in each pair of bars. Thanks to the biasing (optimized at the scrubbing interval of 10^7s), only 10.2% storage overhead (brought down from almost 32%) is required. This is what ultimately allows us to reach storage density 2.73× over the 2-level baseline.

8.6 Applicability to Flash

Although the proposed co-design of image encoding algorithm and approximate memory bases the study on multi-level PCM, the framework is readily applicable to other technologies, *e.g.*, Flash. Multi-level Flash (*e.g.*, TLC NAND Flash) is supported by major manufacturers [19, 28]. In such devices, ECCs (BCH and LDPC are common) are applied to a sector of 512 bytes (or greater, 1024 bytes). Figure 13 compares various BCH codes in terms of error correction capability and storage overhead (similar to Figure 11 but at a larger granularity).

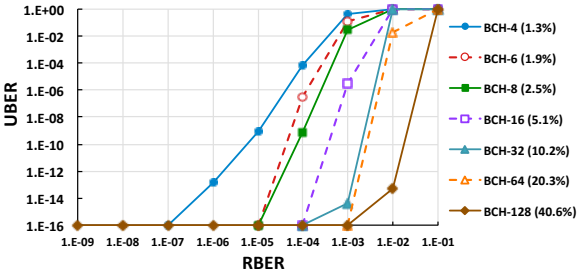

Figure 13. Capabilities and overheads of error correcting codes commonly used with Flash. Each code is able to correct the denoted number of errors in a 512-byte Flash sector and the associated ECC checker bits.

Prior studies report that TLC NAND Flash devices have an initial RBER of 10^{-4}, which increases gradually with the number of program/erase cycles [35]. Combining this knowledge with the methodology presented in this paper, a TLC Flash would use BCH-16 for the cells that store the control and run-length bits in MB1, and BCH-6 for the remaining control and run-length bits, and leave all refinement bits uncorrected. This results in 2.96× higher density than SLC while maintaining a less than 1 dB quality loss. In comparison, thorough correction, which uses BCH-16 uniformly, achieves 2.85× higher density. However, these numbers only apply to a brand new device. As RBER increases along with program/erase cycles, stronger ECCs are gradually required. For instance, RBER reaches 10^{-3} after approximately 3000 program/erase cycles. At this point, the density improvement of selective correction and thorough correction lower to 2.88× and 2.49×, respectively, making selective correction more attractive.

9. Related Work

Sampson *et al.* [27] observed that not all applications need high-precision storage (*i.e.*, storage that has really low error rates) for all kinds of data, and proposed trading-off storage accuracy for other desirable characteristics including density, performance, endurance, and energy efficiency. However, practical and crucial issues such as how to identify data structures that have different levels of error tolerance and how to manage them within a single storage system were not addressed. This paper advances prior work by demonstrating

a methodology of quantifying the reliability requirements of different parts of an encoded image and optimizing an approximate storage substrate to meet them. On top of the image quality versus storage space trade-off already enabled by the algorithms in the JPEG family, programmers are offered explicit control over mapping application data onto multiple approximation levels to gain extra space savings.

Seong *et al.* [30, 37] proposed tri-level cell PCM, which was motivated by the fact that the uniform 4-level cell suffered high error rates due to drift. The tri-level cell was architected as a precise memory in prior work (and is used as a baseline in this paper). In this work we exploit practical use of cells with more levels as approximate storage. Cell reliability is significantly improved via *biasing* and the storage density is enhanced via *selective error correction*.

Yoon *et al.* [37] proposed *biasing* to address the drift issue in PCM. We leverage this idea to tune MLC PCM cells and minimize the combined (*i.e.*, write and drift) error rate at a certain scrubbing interval. Xu and Tong [33] proposed an adaptive drift mitigation approach for PCM, which stores a 'timestamp' in a reference cell along with the cells that store the data. When reading the data, the controller uses the timestamp to decide the level boundries of the cells. Sala *et al.* [26] proposed dynamic thresholding for multi-level cells. This mechanism adapts read circuitry parameters according to the elapsed time since the cells were written. Unlike these and other techniques that dynamically tune the read circuits to offset the drift effect, *biasing* settles the optimal positions for levels at manufacturing time, requiring lower area overheads and hence lower costs.

Holcomb and Fu [11] proposed an automated method that maps a data word onto multiple approximate MLCs such that the worst-case error is minimized at the word-level. The scheme is based on uniformly partitioned levels and certain levels that cause high error rates are removed. In contrast, this work uses biased levels that make better use of the resistance range of storage cells. In addition, it leverages relative importance of bits to choose their mapping to different types of approximate storage cells, and observes that a non-uniform distribution of values may lead to higher than expected error rates if the mapping between values and levels in a cell is not favorable.

In multimedia communication systems, Guo *et al.* [10] and Frescura *et al.* [7] explored an unequal error protection approach to protect image and video data against channel errors during transmission. The idea is to choose different coding parameters and redundancy levels based on the parts of the compressed bitstream. Specifically, maximum redundancy is used for headers and lower redundancy for less important parts of the compressed data. While similar in spirit, this technique is different from our work in at least two key aspects. First, we focus on storage systems, which have very different properties compared to wireless channels (*e.g.*, packet drop/loss is the main issue in wireless channels). Second, our proposal adjusts error correction at a much finer granularity, considering component frequency as well as the kind of bit-stream (control, run-length, and refinement). This leads to a completely different algorithmic structure. To our knowledge, our proposal is the first to apply selective reliability to image storage.

10. Conclusion

We proposed to cooperatively design image encoding and storage mechanisms for denser approximate storage. The key idea is to identify the relative importance of encoded bits on output image quality and perform error correction accordingly. The result is over 2.7× the density of the image storage with little quality degradation (less than 1 dB). We focused on PTC image encoding and PCM, but our ideas also apply to other storage substrates and other algorithms that use time-frequency transformation followed by quantization and entropy coding (*e.g.*, audio and video).

Acknowledgments

The authors would like to thank Adrian Sampson and Kathryn McKinley for feedback and early discussions on this work and the anonymous reviewers for the useful comments and suggestions.

References

[1] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramanian, and V. Srinivasan. Efficient scrub mechanisms for error-prone emerging memories. In *Proceedings of the 18th International Symposium on High Performance Computer Architecture*, February 2012.

[2] R. Azevedo, J. Davis, K. Strauss, M. Manasse, P. Gopalan, and S. Yekhanin. Zombie memory: Extending memory lifetime by reviving dead blocks. In *Proceedings of the 40th International Symposium on Computer Architecture*, June 2013.

[3] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. C. Buda, F. Pellizzer, D. W. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE Journal of Solid-State Circuits*, 44(1):217–227, 2009.

[4] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Error patterns in MLC NAND Flash memory: Measurement, characterization, and analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 521–526, San Jose, CA, USA, 2012. EDA Consortium.

[5] F. Dufaux, G. J. Sullivan, and T. Ebrahimi. The JPEG XR image coding standard. *IEEE Signal Processing Magazine*, 26: 195–199, June 2009.

[6] R. Franzen. Kodak Lossless True Color Image Suite: PhotoCD PCD0992, 2002.

[7] F. Frescura, M. Giorni, C. Feci, and S. Cacopardi. JPEG2000 and MJPEG2000 transmission in 802.11 wireless local area networks. *IEEE Transactions on Consumer Electronics*, 49(4): 861–871, Nov 2003.

[8] M. Gastpar and M. Vetterli. Source-channel communication in sensor networks. *Information Processing in Sensor Networks*, pages 162–177, 2003.

[9] M. Gastpar, B. Rimoldi, and M. Vetterli. To code, or not to code: Lossy source-channel communication revisited. *Transactions on Information Theory*, pages 1147–1158, 2003.

[10] Z. Guo, Y. Nishikawa, R. Y. Omaki, T. Onoye, and I. Shirakawa. A low-complexity FEC assignment scheme for motion JPEG2000 over wireless network. *IEEE Trans. on Consum. Electron.*, 52(1):81–86, Feb. 2006.

[11] D. Holcomb and K. Fu. QBF-based synthesis of optimal word-splitting in approximate multi-level storage cells. In *Workshop on Approximate Computing Across the System Stack (WACAS), 2014*, March 2014.

[12] D. Ielmini, A. L. Lacaita, and D. Mantegazza. Recoverty and drift dynamics of resistance and threshold voltages in phase-change memories. *IEEE Transactions on Electron Devices*, 54(2):308–315, February 2007.

[13] D. Ielmini, S. Lavizzari, D. Sharma, and A. L. Lacaita. Physical interpretation, modeling and impact on phase change memory (PCM) reliability of resistance drift due to chalcogenide structural relaxation. In *Proceedings of the International Electron Devices Meeting*, December 2007.

[14] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin. Coset coding to extend the lifetime of memory. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 222–233, 2013.

[15] H. S. Malvar. *Signal Processing with Lapped Transforms*. Artech House, 1992. `http://research.microsoft.com/apps/pubs/default.aspx?id=77707`.

[16] H. S. Malvar. Fast progressive wavelet coding. In *Proceedings of the Data Compression Conference*, DCC '99, pages 336–343, Washington, DC, USA, 1999

[17] H. S. Malvar. Fast progressive image coding without wavelets. In *Proceedings of the Data Compression Conference*, pages 243–252, March 2000.

[18] H. S. Malvar. Adaptive Run-Length / Golomb-Rice encoding of quantized generalized gaussian sources with unknown statistics. In *Proceedings of the Data Compression Conference*, DCC '06, pages 23–32, Washington, DC, USA, 2006.

[19] Micron. Micron TLC NAND. `http://www.micron.com/products/nand-flash/tlc-nand`.

[20] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill. Bit error rate in NAND flash memories. In *IEEE International Reliability Physics Symposium, 2008. IRPS 2008.*, pages 9–19, April 2008.

[21] T. Nirschl, J. B. Philipp, T. D. Happ, G. W. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. C. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y.-C. Chen, Y. Zhu, R. Bergmann, H. L. Lung, and C. Lam. Write strategies for 2 and 4-bit multi-level phase-change memory. In *IEEE International Electron Devices Meeting, 2007. IEDM 2007.*, pages 461–464, 2007.

[22] N. Papandreou, H. Pozidis, A. Pantazi, A. Sebastian, M. Breitwisch, C. Lam, and E. Eleftheriou. Programming algorithms for multilevel phase-change memory. In *2011 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 329–332, May 2011.

[23] W. B. Pennebaker and J. L. Mitchell. JPEG: Still image data compression standard. Technical report, Springer, 1993.

[24] M. K. Qureshi. Pay-as-you-go: Low-overhead hard-error correction for phase change memories. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 318–328, 2011.

[25] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis. Morphable memory system: A robust architecture for exploiting multi-level phase change memories. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 153–162, New York, NY, USA, 2010.

[26] F. Sala, R. Gabrys, and L. Dolecek. Dynamic threshold schemes for multi-level non-volatile memories. *IEEE Transactions on Communications*, 61(7):2624–2634, July 2013.

[27] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage. In *Proceedings of the 46th International Conference on Microarchitecture*, December 2013.

[28] Samsung. Samsung SSD Product Story: 3-bit/cell MLC NAND Flash. `http://www.samsung.com/global/business/semiconductor/minisite/SSD/global/html/why/MlcNandFlash.html`.

[29] J. W. Schwartz and R. C. Barker. Bit-plane encoding: A technique for source encoding. *IEEE Transactions on Aerospace and Electronic Systems*, AES-2(4):385–392, July 1966.

[30] N. H. Seong, S. Yeo, and H.-H. S. Lee. Tri-level-cell phase change memory: Toward an efficient and reliable memory system. In *Proceedings of the 40th International Symposium on Computer Architecture*, June 2013.

[31] M. Stanisavljevic, A. Athmanathan, N. Papandreou, H. Pozidis, and E. Eleftheriou. Phase-change memory: Feasibility of reliable multilevel-cell storage and retention at elevated temperatures. In *Reliability Physics Symposium (IRPS), 2015 IEEE International*, pages 5B.6.1–5B.6.6, April 2015.

[32] S. Vembu, S. Verdú, and Y. Steinberg. The source-channel separation theorem revisited. *IEEE Transactions on Information Theory*, 41:44–45, January 1995.

[33] X. Wei and Z. Tong. A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(8):1357–1367, 2011.

[34] J. Y. Wu, W. S. Khwa, M. H. Lee, H. P. Li, S. C. Lai, T. H. Su, M. L. Wei, T. Y. Wang, M. BrightSky, T. S. Chen, W. C. Chien, S. Kim, R. Cheek, H. Y. Cheng, E. K. Lai, Y. Zhu, H. L. Lung, and C. Lam. Greater than 2-bits/cell mlc storage for ultra high density phase change memory using a novel sensing scheme. In *2015 Symposium on VLSI Technology*, 2015.

[35] E. Yaakobi, L. Grupp, P. Siegel, S. Swanson, and J. Wolf. Characterization and error-correcting codes for tlc flash memories. In *2012 International Conference on Computing,*

Networking and Communications (ICNC), pages 486–491, Jan 2012.

[36] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 466–477, 2011.

[37] D. H. Yoon, J. Chang, R. S. Schreiber, and N. P. Jouppi. Practical nonvolatile multilevel-cell phase change memory. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*, pages 21:1–21:12, 2013.

[38] W. Zhang and T. Li. Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system. In *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 197–208, June 2011.

[39] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard In *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, July 2003.