MASS: A Masking-aware Search Framework for Reliable QC-LDPC Code Construction in SSDs

Xiaolu Li, Dingxin Wang, Zhengyao Ding, Jinye Wu, Qingnan Hu
Huazhong University of Science and Technology, Wuhan, China

Patrick P. C. Lee
The Chinese University of Hong Kong
Hong Kong, China

Yuchong Hu
Huazhong University of Science and Technology
Wuhan, China

Dan Feng
Huazhong University of Science and Technology
Wuhan, China

***Abstract*—Quasi-Cyclic Low-Density Parity-Check (QC-LDPC) codes are widely used in flash-based solid-state drives (SSDs) to ensure storage reliability due to their efficient encoding and decoding, as well as their compact structure. However, as SSD capacities grow, existing QC-LDPC codes face higher error rates, leading to both lifetime and performance degradation. We propose MASS, a new QC-LDPC code construction framework that leverages masking to remove error-prone substructures from the coding matrix during construction, thereby improving SSD reliability. MASS further adopts a smart decoding policy that selectively skips decoding operations to boost I/O performance. We implement MASS in the MQSim SSD simulator, where evaluation shows up to 93% reduction in decoding failure rates, and up to 47.5% lower read response latency.**

***Index Terms*—SSD reliability, ECC, LDPC codes**

I. Introduction

Advances in solid-state drive (SSD) technology, particularly multi-level cell storage, have significantly increased SSD capacities by enabling a single flash cell to store multiple bits [3], [16], [20], [41]. However, this higher density also reduces reliability, with the raw bit error rate (RBER) increasing from 10^{-8} in 2009 [12] to 0.01 in 2017 [8], and further climbing to 0.015 in a recent study [57].

To ensure reliability under increasing RBERs, SSDs use error correction codes (ECCs), which encode user data with additional parity bits for error recovery. Traditionally, SSDs rely on Bose-Chaudhuri-Hocquenghem (BCH) codes [6], [26], [30], [31], [38]. As RBERs increase, low-density parity-check (LDPC) codes become the preferred choice of ECCs due to their superior error-correction capability [60]. In particular, quasi-cyclic LDPC (QC-LDPC) codes are the most commonly used LDPC codes in modern SSDs, as their quasi-cyclic structure simplifies encoding and decoding operations with improved computational efficiency, and further reduces storage overhead in the SSD controller via a more compact matrix representation.

This work was supported in part by the National Natural Science Foundation of China (62302175). Corresponding author: Patrick P. C. Lee (pclee@cse.cuhk.edu.hk).

As RBERs continue to grow in large-capacity flash-based SSDs, the reliability of QC-LDPC codes becomes a critical issue, especially when the number of errors may exceed the error correction capability of the QC-LDPC codes. To enhance the error correction capability of QC-LDPC codes, one approach is to reduce the *code rate*, defined as the number of user data bits divided by the *code length* (i.e., the total number of user data bits and parity bits). However, a lower code rate results in higher storage overhead for parity bits. This is particularly problematic for large-capacity SSDs. For example, for a 100 TiB SSD, reducing the code rate from 0.9 to 0.8 doubles the storage for parity bits from 10 TiB to 20 TiB.

An alternative approach is to improve the QC-LDPC code design with higher error correction capability, while maintaining a high code rate and low storage costs. However, designing reliable QC-LDPC codes is non-trivial. We observe that commonly used QC-LDPC codes are based on deterministic constructions, which restrict the potential for further reliability optimization. Existing theoretical constructions and optimization techniques (§II-C) often fall short of delivering sufficient reliability improvements for large code lengths, which are typical in SSDs. Consequently, designing reliable QC-LDPC codes and leveraging their strong error correction capability for I/O performance improvements remain unresolved.

We propose MASS, a QC-LDPC code construction framework designed to enhance SSD reliability with stronger error correction capability and improve I/O performance based on an enhanced error correction model. Our contributions are summarized as follows.

- We propose a masking-aware search framework to construct more reliable QC-LDPC codes for SSDs. MASS leverages masking to remove error-prone substructures from the coding matrix during QC-LDPC code construction.
- We design a smart decoding technique that selectively bypasses decoding based on an enhanced error correction model of MASS, so as to improve I/O performance.
- We implement MASS into the open-source SSD simulator, MQSim [50]. Our evaluation shows that MASS reduces the

decoding failure rate by up to 93% compared to state-of-the-art QC-LDPC code constructions. Based on Alibaba block traces [28], our smart decoding technique reduces the read response latency by up to 47.5% compared to the default decoding approach.

MASS is now open-sourced at **https://github.com/ukulililixl/mass**.

II. Background

A. Basics of SSDs

SSDs perform I/O in units of *pages*. Each page consists of a *user area* for storing data and a *spare area* for metadata, such as parity data for fault tolerance. Page sizes vary by manufacturer, but mainstream SSDs typically allocate 16 KiB for the user area and about 2 KiB for the spare area [46]. For example, MICRON's NAND flash implementation designates 2,208 bytes for the spare area [1].

To write pages, the SSD controller generates ECC *codewords*, comprising user data and corresponding parity data. Due to the high encoding complexity of large codewords, a page is typically segmented into multiple smaller codewords, such as 2 KiB or 4 KiB of user data per codeword [39], [60].

To read pages, the SSD controller first senses the raw page data and applies *hard-decision decoding* (or *hard decoding* in short), a fast and simple error correction method based on bit values (0 or 1). If successful, the user data is returned. If errors remain, the SSD controller applies multiple rounds of *soft-decision decoding* (or *soft decoding* in short). Each round starts with a *read retry* to re-sense the raw data [10], [32], [40], followed by more robust, computationally intensive error correction. If soft decoding cannot correct all errors after a certain number of rounds, a read error is returned.

B. Basics of QC-LDPC Codes

LDPC codes are widely adopted ECCs in modern SSDs [60]. An (N,M) LDPC code encodes $N-M$ information bits into M parity bits, with a *code length* N and a *code rate* $\frac{N-M}{N}$. A common code rate in production SSDs is 0.89 [11], [14], with 4-KiB user data and 512-byte parity data per codeword.

Parity-check matrix. An LDPC code is defined by a parity-check matrix $\mathbf{H}$ with M rows and N columns, where the N columns correspond to the N bits of a codeword $(v_0, v_1, \ldots, v_{N-1})$, while the M rows represent M parity-check equations $(c_0, c_1, \ldots, c_{M-1})$. The matrix $\mathbf{H}$ is characterized by the *column weight* (i.e., number of 1's per column) and the *row weight* (i.e., number of 1's per row). An LDPC code is *regular* if all columns have the same column weight and all rows have the same row weight; otherwise, it is *irregular*. LDPC codes in SSDs often have a column weight of 4 or 5 for enhanced reliability [13], [53], [59].

Tanner graph. A Tanner graph $G=(V,E)$ is a bipartite graph that describes an (N,M) LDPC code, which contains N *variable nodes* (representing codeword bits) and M *check nodes* (representing parity equations). Each check node c_i connects to variable nodes involved in its parity equation. The error correction algorithms are performed based on the Tanner graph (e.g., bit-flipping algorithm for hard decoding [51] and min-sum algorithm for soft decoding [44]). For example, Equation (1) defines an irregular $(8,4)$ LDPC code, which encodes four information bits v_0, v_1, v_2, and v_3 into four parity bits v_4, v_5, v_6, and v_7, and the four rows of the parity-check matrix define four parity equations in Equation (2), which are all equal to zero for a correct codeword (i.e., $c_i=0$ for all $i=1,2,3,4$).

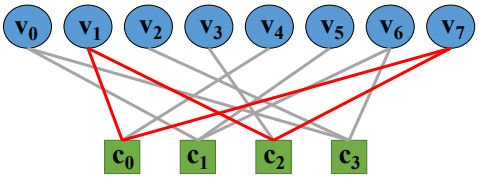


Fig. 1: Example of the Tanner graph for the $(8,4)$ LDPC code in Equation (1), where variable nodes are in the top layer and check nodes are in the bottom layer.

$$\mathbf{H}=\left[\begin{array}{cccc|cccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right]. \quad (1)$$

$$\begin{aligned} c_0 &= v_1+v_4+v_7,\ c_1 = v_0+v_5+v_6, \\ c_2 &= v_1+v_3+v_7,\ c_3 = v_0+v_2+v_6. \end{aligned} \quad (2)$$

Figure 1 shows the corresponding Tanner graph. For example, the check node c_0 connects to variable nodes v_1, v_4, and v_7, so $c_0=v_1+v_4+v_7$, which equals zero in an error-free codeword.

The error correction capability of an LDPC code mainly depends on three metrics:

- **Cycles:** *2ℓ-Cycle* is a closed path in the Tanner graph that comprises ℓ variable nodes and ℓ check nodes. For example, (v_1, c_0, v_7, c_2, v_1) form a 4-cycle in Figure 1. Shorter cycles are more prone to uncorrectable errors due to limited information available for error correction. Thus, reducing the number of short cycles improves the error correction capability.
- **Girth:** The *girth* is the length of the shortest cycle in a Tanner graph. In Figure 1, the girth is 4. A larger girth implies longer cycles, so increasing the girth improves the error correction capability.
- **Trapping sets:** A *trapping set* defines a group of bits that, if simultaneously erroneous, cannot be corrected [43]. Short cycles with *chords* (i.e., edges that connect two non-adjacent vertices of a cycle) are often the root causes of trapping sets [4]. Thus, reducing the number of chords reduces the number of trapping sets and improves the error correction capability.

QC-LDPC codes. QC-LDPC codes are a special family of LDPC codes that have been popularly used in SSDs [54], [58]. QC-LDPC codes are configured by parameters (N, M, q), where the $M\times N$ parity-check matrix can be divided into several square sub-matrices of size $q\times q$. Each sub-matrix is either a permuted identity matrix or a zero matrix. For example, the $(8,4)$ LDPC code in Equation (1) corresponds

to an $(N = 8, M = 4, q = 2)$ QC-LDPC code, with its matrix divided into eight 2×2 sub-matrices.

QC-LDPC codes achieve small storage footprints, as the parity-check matrix of size $M \times N$ can be compressed into an *exponent matrix* of size $\frac{N}{q} \times \frac{M}{q}$. Each item in the exponent matrix means a zero matrix (denoted by $infty$), or a permuted identity matrix (denoted by integers in $[0, q-1]$). For example, the parity-check matrix in Equation 1 can be compressed into the following exponent matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & \infty & 0 & 1 \\ 1 & 1 & \infty & 1 \end{bmatrix} \tag{3}$$

QC-LDPC codes also enable parallel processing by dividing the parity-check matrix into sub-matrices, so as to allow multiple processing units to perform computations on different parts of the code simultaneously.

C. Limitations of Existing QC-LDPC Codes

We review existing QC-LDPC construction approaches and their limitations.

Finite-field-based QC-LDPC codes. QC-LDPC codes based on finite fields, such as Galois Field (GF), are commonly used in SSDs [13], [19], [27], with exponent matrix elements derived from GF [19], [27]. While some GF-based constructions are specifically designed for NAND flash [5], [15], [22], [29], [33] and enable rapid code generation, they do not optimize cycles, girth, and trapping sets for reliability enhancements.

Search-based QC-LDPC codes. Search-based QC-LDPC constructions use algorithms to optimize code structures. While effective for short codes (e.g., shorter than 512 bytes) [47]–[49] or trapping set elimination [4], they face challenges with longer codes used in SSDs (longer than 4 KiB), as they often result in smaller girth [47]–[49] and have difficulty in removing small trapping sets [4].

Masking. *Masking* improves QC-LDPC reliability by strategically replacing exponent matrix values with ∞ (converting certain 1's to 0's in the parity-check matrix), to remove Tanner graph edges to eliminate cycles, increase girth, and reduce trapping sets. Existing masking techniques include *random masking*, which inserts ∞ randomly [29], and *structural masking* which inserts ∞ based on predefined patterns (e.g., cyclic shifts [13], [29]). Masking can be implemented before [52] or after [55] the construction of a QC-LDPC code. However, existing masking approaches do not simultaneously optimize cycles, girths, and trapping sets.

III. MASS

A. QC-LDPC Construction Framework

Since increasing girth, reducing cycles, and reducing trapping sets enhance reliability, MASS aims to construct a QC-LDPC code by finding an exponent matrix with a larger girth, fewer cycles, and fewer trapping sets, given parameters N, M, q, and column weight w. MASS's key idea is to incorporate masking into its construction process. It returns an $\frac{M}{q} \times \frac{N}{q}$ exponent matrix with entries in $[0, q-1]$ or ∞, following these steps:

Step 1: Initialization. We initialize a random exponent matrix $\mathbf{A}$ of size $\frac{M}{q} \times \frac{N}{q}$, with each entry chosen from $[0, q-1]$ or ∞. To meet the column weight constraint, each column contains exactly w values from $[0, q-1]$. We then evaluate the girth g, the number of g-cycles r_g, and the number of chords under girth g, denoted by t_g.

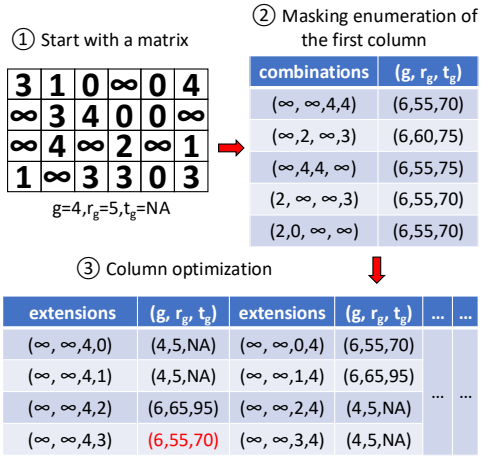
Step 2: Matrix optimization. We search for a better exponent matrix with a larger girth g, fewer g-cycles r_g, and fewer chords t_g in multiple rounds. In each round, we update the exponent matrix column by column, starting from the leftmost column in $\mathbf{A}$. The search for a column comprises the following steps, and is executed in multiple rounds until it cannot find an exponent matrix that provides a larger girth, fewer cycles, or fewer chords.

Step 2.1: Masking enumeration. When optimizing a column, we first enumerate all possible combinations of positions to fill with ∞. Given the column weight w, the number of positions filled with ∞ in a column should be $\frac{M}{q} - w$. For each combination, we randomly fill the remaining positions with values from $[0, q-1]$ and evaluate the corresponding girth g' as well as the number of g'-cycles r'_g, and the number of chords under girth g', denoted by t'_g.

Step 2.2: Column optimization. We optimize each column based on the combinations enumerated in Step 2.1. For each combination, we change the value of one position out of the w positions that are not ∞ to another value from $[0, q-1]$ as an *extension* of the combination. We then examine whether the extension provides a larger girth, fewer cycles, and fewer chords. Once we encounter a combination that provides a better exponent matrix, we update $\mathbf{A}$ accordingly.

Priority in comparison. When comparing the girth g, the number of g-cycles, and the number of chords under girth g of two exponent matrices, we give the highest priority to the girth, such that an exponent matrix with a larger girth is considered better than the other. For the number of g-cycles and chords, we provide two comparison policies: *cycle-first*, where the number of g-cycles has higher priority, and *chord-first*, where the number of chords has higher priority.

Example. Figure 2 shows an example of how we optimize a column in a round of Step 2 for the $(6,4,5)$ QC-LDPC code with $w = 2$. Suppose that we start with a randomly initialized exponent matrix (①), with the girth $g = 4$ and the number of 4-cycles $r_4 = 5$. Next, we optimize the first column. We enumerate $\binom{4}{2} = 6$ combinations of the two ∞ positions in the first column. For each combination, we randomly fill the remaining positions and evaluate the corresponding metrics of girth, cycles, and chords (②). Then, we generate extensions starting from the first combination of $(\infty, \infty, 4, 4)$ and examine the metrics one by one (③). Note that $(g = 6, r_g = 55, t_g = 70)$ is the best result among all the extensions of $(\infty, \infty, 4, 4)$, with the first occurrence in $(\infty, \infty, 4, 3)$. Thus, we update the first column of $\mathbf{A}$ with $(\infty, \infty, 4, 3)$ after examining the extension of the first combination of ∞. For brevity, we omit the details of examining the extensions of the other five combinations of the first column in this example.



3	1	0	∞	0	4
∞	3	4	0	0	∞
∞	4	∞	2	∞	1
1	∞	3	3	0	3

combinations	(g, r_g, t_g)
(∞, ∞,4,4)	(6,55,70)
(∞,2, ∞,3)	(6,60,75)
(∞,4,4, ∞)	(6,55,75)
(2, ∞, ∞,3)	(6,55,70)
(2,0, ∞, ∞)	(6,55,70)

extensions	(g, r_g, t_g)	extensions	(g, r_g, t_g)	...	...
(∞, ∞,4,0)	(4,5,NA)	(∞, ∞,0,4)	(6,55,70)		
(∞, ∞,4,1)	(4,5,NA)	(∞, ∞,1,4)	(6,65,95)	...	...
(∞, ∞,4,2)	(6,65,95)	(∞, ∞,2,4)	(4,5,NA)		
(∞, ∞,4,3)	(6,55,70)	(∞, ∞,3,4)	(4,5,NA)		

Fig. 2: Example of the optimization of a column in a round of step 2 for the $(6,4,5)$ QC-LDPC code with $w=2$, where g, r_g, and t_g refer to the girth, the number of g-cycles, and the number of chords under girth g, respectively.

We show that MASS improves reliability in terms of the girth, number of cycles, number of chords (see Exp#1 in §IV-A), and the decoding failure rate (see Exp#2 in §IV-A), compared with several state-of-the-art QC-LDPC constructions.

B. Smart Decoding

At high RBERs, hard decoding often fails, requiring multiple soft decoding rounds. Skipping hard decoding at a high RBER improves read performance, but determining when to skip hard decoding is difficult, as the QC-LDPC code acts as a black box in SSDs. MASS adopts *smart decoding* by selectively bypassing hard decoding based on the error correction model of the constructed QC-LDPC code.

Latency model. We first model the latency of a single decoding operation, which includes three components. (i) *data read latency*, the time to read data from the flash chip, denoted by t_{Rh} for hard decoding and t_{Rs} for soft decoding; (ii) *data transfer latency*, the time to transfer data from the flash chip to the decoder, denoted by t_{DMAh} for hard decoding and t_{DMAs} for soft decoding; and (iii) *decoding latency*, the time to execute the decoding algorithm, denoted by t_{ECCh} for hard decoding and t_{ECCs} for soft decoding.

Thus, the latency of hard decoding is

$$t_h(RBER) = t_{Rh} + t_{DMAh} + t_{ECCh}(RBER), \tag{4}$$

while the latency of a round of soft decoding is

$$t_s(RBER) = t_{Rs} + t_{DMAs} + t_{ECCs}(RBER). \tag{5}$$

Smart decoding. Based on the latency model, we can make a decision on when to skip the hard decoding. We mainly consider only one round of hard decoding and one round of soft decoding, as multiple rounds of soft decoding indicate a high RBER where hard decoding is unlikely to succeed. Given the error correction model, $P(RBER)$, which denotes the failure probability of hard decoding at a given RBER. The expected latency of performing a round of hard decoding followed by a round of soft decoding (if hard decoding fails) is:

$$T = t_h(RBER) + P(RBER) \times t_s(RBER). \tag{6}$$

By comparing T and $t_s(RBER)$, we can find a crosspoint, where on the left of the point, $T < t_s(RBER)$, meaning that the default decoding is expected to finish within less time compared with skipping the hard decoding, while on the right of the point, skipping hard decoding provides less latency. In real deployment, we can figure out the RBER of this point given the system configurations and the error model of an SSD to enable smart decoding (See our experiment Exp#3 in §IV-B).

C. Implementation

The implementation of MASS comprises two parts: (i) an offline QC-LDPC constructor based on our design in §III-A; and (ii) the integration of ECC modules, including an *error injector*, and a pair of *encoder* and *decoder* per channel into the MQSim SSD simulator [50] to show how we deploy the QC-LDPC code generated by MASS in a general SSD. The implementation involves 3.9 K LoC in MQSim.

Error injector. The error injector simulates errors in raw data read from flash chips based on a published error model [21] and the Additive White Gaussian Noise (AWGN) model [23]. It predicts RBERs from SSD statistics, halving the RBER with each read retry. It injects errors accordingly and initiates states for hard decoding and soft decoding operations.

Encoder. The encoder simulates the encoding process by adjusting the data size to account for parity data, as MQSim does not store actual data. The encoding procedure is integrated into the write flow, redirecting write requests to the encoder to modify the data size before it reaches the back end.

Decoder. The decoder handles error correction, supporting the default decoding procedure that begins with hard decoding followed by multiple rounds of soft decoding, and the smart decoding method (§III-B). The hard decoding uses the Gradient Descent Bit-Flipping algorithm [51], while the soft decoding employs the min-sum algorithm [44].

Latency model modifications. We update MQSim's end-to-end latency model to support MASS by incorporating *encoding latency* (data size divided by the encoding throughput of specific hardware), *decoding latency* (Equations (4) and (5)), and *pipelining* (where encoding or decoding of one codeword occurs concurrently with the data transfer of another codeword).

Discussion. We assume sufficient memory to store the full address mapping table, enabling in-memory operations to enhance I/O performance. MASS integrates ECC operations into the read/write flows, ensuring compatibility with standard garbage collection procedures. For overhead, MASS aligns with conventional QC-LDPC ECC processes, introducing no additional computational overhead and requiring only minimal extra memory to store RBER values for smart decoding.

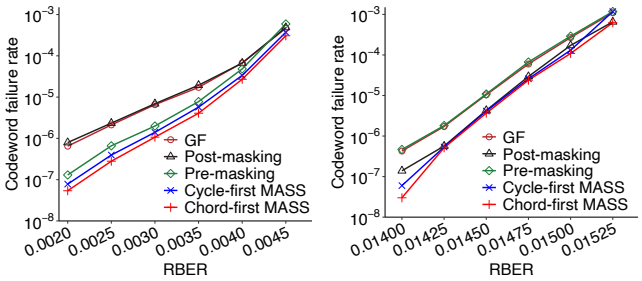
IV. Evaluation

A. Reliability Analysis

We compare MASS's reliability with several state-of-the-art QC-LDPC code constructions: (i) *GF*, which constructs an

TABLE I: (Exp#1) Girth, cycles, and chords.

Approach	(g, r, t)	Running time
GF	(6, 228417, 93513)	2 s
Post-masking	(6, 66170, 6617)	18 s
Pre-masking	(6, 2560, 0)	175 s
Cycle-first MASS	(8, 15008256, 689072128)	330 s
Chord-first MASS	(8, 14994432, 686882816)	330 s



(a) Hard decoding (b) Soft decoding

Fig. 3: (Exp#2) Decoding failure rate.

exponent matrix based on Galois Field [13], [19], [27]; (ii) *post-masking*, which applies masking after forming an exponent matrix to eliminate small cycles [55]; (iii) *pre-masking*, which applies masking before forming an exponent matrix to eliminate small cycles [52]; (iv) *cycle-first MASS*, configured with the cycle-first policy; and (v) *chord-first MASS*, configured with with the chord-first policy. By default, we configure ($N = 36864, M = 4096, q = 512$) QC-LDPC with a column weight of four (i.e., code rate of 0.89). Each codeword comprises 4 KiB of information bits and 512 bytes of parity bits.

(Exp#1) Girth, cycles, and chords. We evaluate the girth (denoted as g), the number of g-cycles, and the number of chords under g-cycles for QC-LDPC codes constructed by different methods. Table I shows the results. MASS achieves a girth of $g = 8$, surpassing other methods limited to $g = 6$, thus offering higher reliability at the cost of increased algorithm running time. As a search-based algorithm, MASS incurs longer runtimes, which we deem acceptable for offline exponent matrix construction. Among the two MASS variants, chord-first MASS shows slightly higher reliability than cycle-first MASS. Both achieve $g = 8$, but chord-first MASS has fewer cycles and chords, indicating that chords have a slightly greater impact on QC-LDPC code error correction capability.

(Exp#2) Decoding failure rate. We evaluate the decoding failure rate of a codeword across various RBERs, where the decoding failure rate specifies the probability of a failed read operation in an SSD. For each test, we encode information bits based on Gaussian elimination [7], [25], simulate errors by the error injector for different RBERs, and then apply the decoder for error correction. The decoding failure rate is calculated as the number of failed tests divided by the total number of tests, with up to 500 million tests for hard decoding and 100 million tests for soft decoding. As shown in Figure 3(a), MASS outperforms other methods in both hard and soft decoding.

For hard decoding, chord-first MASS achieves decoding failure rate reductions of 91.7%, 93.2%, 58.6%, and 30.9% compared to GF, Post-masking, Pre-masking, and cycle-first MASS, respectively, at an RBER of 0.002. At an RBER of 0.0045, the reductions are 34.1%, 37.0%, 48.2%, and 17.5%, respectively. As Figure 3(a) shows, chord-first MASS offers higher reliability over other methods when the RBER is below 0.0045, where an SSD operates in healthy mode after many program-and-erase (P/E) cycles. At RBERs above 0.0045, MASS's decoding failure rate is comparable to other methods due to excessive errors, with soft decoding ensuring reliability.

For soft decoding, similar trends are observed, with reductions of 50%-93.6% at an RBER of 0.014 and 4.6%-46.8% at an RBER of 0.01525 compared to other methods. Figure 3(b) shows that MASS outperforms other approaches and provides higher reliability when the RBER is below 0.015.

MASS's reliability improvements stem from its QC-LDPC construction that optimizes girth, cycles, and trapping sets through masking awareness. We omit results beyond specified RBERs due to small differences at high RBERs or insufficient errors at low RBERs.

B. Performance Evaluation in MQSim

Default settings. We use the SSD configuration from [9], featuring a 2 TiB capacity, 8 channels (2 chips per channel), and each chip with 2 dies (4 planes per die). Each plane has 1,888 blocks with 576 pages per block, and each page is 18 KiB (16 KiB user space and 2 KiB spare space). Write and erase latencies are set to 400 µs and 3,500 µs, respectively, while read latency varies based on hard and soft decoding. The error model from [21] is integrated into MQSim to estimate RBER from page statistics. Each codeword encodes 4 KiB of data with a code rate of 0.89 and 12.4 Gb/s encoding throughput [2], following the default QC-LDPC configuration in §IV-A. Hard decoding runs up to 200 iterations at 0.16 μs per iteration, while soft decoding runs up to 20 iterations at 1 us per iteration [24]. With 2-bit quantization, soft decoding's page read latency is three times that of hard decoding ($t_{Rh} = 13.33\,\mu$s, $t_{Rs} = 40\,\mu$s) [9], [13], [29], [33]. Data transfer latency for soft decoding is double that of hard decoding due to 2-bit quantization, resulting in $t_{DMAh} = 15.36\,\mu$s and $t_{DMAs} = 30.72\,\mu$s based on PCIe 4.0. We perform a trace-driven evaluation using 10 volumes from the Alibaba block trace [28], with characteristics detailed in Table II.

(Exp#3) MASS correction model. We assess when to skip hard decoding by comparing three policies: (i) *default*, which performs hard decoding followed by soft decoding; (ii) *soft-only*, which directly performs soft decoding; and (iii) smart decoding proposed in §III-B. As Figure 4 shows, for RBERs below 0.00575, the default policy offers lower read latency due to efficient hard decoding. For RBERs above 0.00575, soft-only excels as hard decoding runs too many iterations, while soft decoding's stronger correction runs fewer iterations. Smart decoding dynamically switches between the two, achieving the lowest latency (i.e., the red curve in Figure 4).

TABLE II: Volumes selected from Alibaba block trace.

Volume ID	# Requests	Read ratio	Avg. read bytes	Avg. write bytes
1	363248	3%	47357	14618
2	293618	10%	26268	17157
3	117498	19%	38637	18979
4	70515	31%	39374	39295
5	37394	40%	30756	15890
6	27598	50%	33138	21489
7	37250	60%	37589	33229
8	36315	70%	33717	80425
9	33344	77%	33356	79615
10	475806	99%	4095	5964

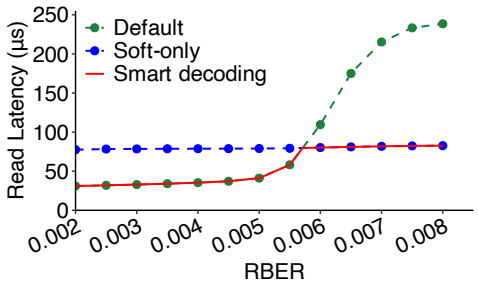


Fig. 4: (Exp#3) MASS correction model.

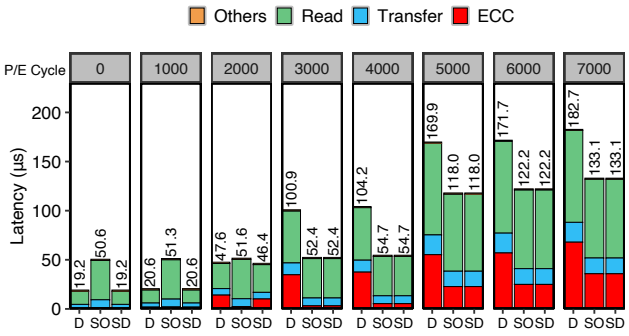


Fig. 5: (Exp#4) Impact of P/E cycles. *D* refers to default; *SO* refers to soft-only; *SD* refers to smart decoding. *Read* refers to the time to read data from flash chips; *Transfer* refers to the time to transfer data from flash chips to the decoder; *ECC* refers to the time spent on error correction; *Others* refers the to time spent in other aspects.

(Exp#4) Impact of P/E cycles. We evaluate the impact of P/E cycles on smart decoding, focusing on read response latency on Volume 10, which has the highest read ratio. Figure 5 shows the results. At P/E cycles below 1,000, both default and smart decoding achieve optimal performance with low error rates, while soft-only underperforms due to quantization overhead. At 2,000 P/E cycles, default and smart decoding show increased ECC time due to higher error rates but still outperform soft-only. At 3,000 P/E cycles, smart decoding matches soft-only's by skipping hard decoding, significantly reducing ECC time. At 4,000 P/E cycles, smart decoding reduces read response latency by 47.5% , effectively mitigating latency under high RBERs.

(Exp#5) Impact of different workloads. We evaluate the performance of smart decoding across 10 selected volumes

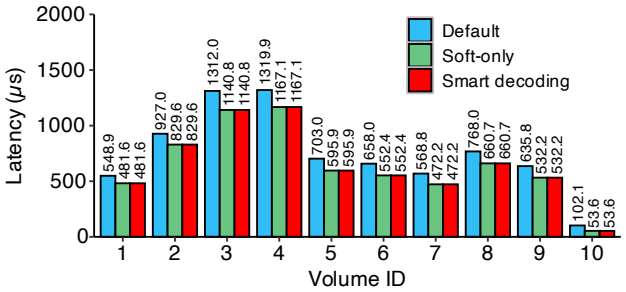


Fig. 6: (Exp#5) Impact of different workloads.

from the Alibaba block trace, with the SSD configured at 3,000 P/E cycles. We measure the average response latency for read and write operations, with results shown in Figure 6. The response latency varies across workloads due to differences in read ratios, request counts, and average read and write byte sizes. Smart decoding improves performance by over 10% compared to the default policy for all workloads. At 3,000 P/E cycles, smart decoding skips hard decoding, matching the performance of soft-only policy.

V. Related Work

In §II-C, we reviewed various QC-LDPC code constructions. Coding-based redundancy enhances fault tolerance at different SSD levels. Cell-based coding, applied within a flash cell, encodes information into distinct states using approaches such as WOM-v codes [17], [18], [37], [56], Gray codes [34], [36], [42], and Multi-Gray-Codes [35]. These can be used with ECCs, which operate within a page to correct raw bit errors from flash chips. While earlier ECCs relied on BCH codes [26], [31], [38], modern flash storage predominantly uses LDPC codes for superior error correction.

Zhao *et al.* [60] examine LDPC codes in SSDs, focusing on reducing LDPC-induced response delays. Song *et al.* [45] propose an adaptive approach that lowers the code rate for hot data to enhance reliability, thereby reducing read failure rates and associated read response latency. Some studies [13], [33] design hardware architectures for LDPC decoders to improve decoding performance in SSDs. Globally-Coupled LDPC (GC-LDPC) codes [5], [29], a special type of QC-LDPC codes, have been proposed to further improve decoding efficiency in SSDs. These performance-driven approaches can be integrated with MASS to enhance reliability.

VI. Conclusion

We present MASS, a framework for constructing reliable QC-LDPC codes for SSDs by applying masking during exponent matrix construction, to increase girth, reduce short cycles, and reduce small trapping sets. Leveraging MASS's error correction model, we propose smart decoding, which selectively bypasses hard decoding for better I/O performance. MASS outperforms state-of-the-art QC-LDPC codes in reliability, while smart decoding significantly improves read performance, particularly at high P/E cycles and enhances write performance.

REFERENCES

[1] 4Tb NAND flash memory; UT81NDQ512G8T. https://www.frontgrade.com/sites/default/files/documents/NAND_Basics_AppNote_1_2%20(2).pdf.

[2] Xilinx LDPC Encoder/Decoder v1.0. https://docs.amd.com/v/u/en-US/pb052-ldpc.

[3] Y. Aiba, H. Tanaka, T. Maeda, K. Sawa, F. Kikushima, M. Miura, T. Fujisawa, M. Matsuo, and T. Sanuki. Cryogenic operation of 3D flash memory for new applications and bit cost scaling with 6-bit per cell (HLC) and beyond. In *Proc. of IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, 2021.

[4] F. Amirzade, M.-R. Sadeghi, and D. Panario. Construction of protograph-based LDPC codes with chordless short cycles. *IEEE Trans. on Information Theory (TIT)*, 2023.

[5] B. Bao, W. Guan, L. Liang, and X. Qiu. An efficient GC-LDPC encoder architecture for high-speed NAND flash applications. *IEICE Electronics Express*, 21(2):1–6, 2024.

[6] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

[7] D. Burshtein and G. Miller. Efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel. *IEEE Trans. on Information Theory (TIT)*, 50(11):2837–2844, 2004.

[8] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proc. of the IEEE (Proc. IEEE)*, 2017.

[9] M. Chun, J. Lee, M. Kim, J. Park, and J. Kim. Rif: Improving read performance of modern SSDs using an on-die early-retry engine. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[10] J. Cui, Z. Zeng, J. Huang, W. Yuan, and L. T. Yang. Improving 3-D NAND SSD read performance by parallelizing read-retry. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 42(3):768–780, 2023.

[11] L. Cui, F. Wu, X. Liu, M. Zhang, R. Xiao, and C. Xie. Improving LDPC decoding performance for 3D TLC NAND flash by LLR optimization scheme for hard and soft decision. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 27(1):1–20, 2021.

[12] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *Proc. of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[13] K.-C. Ho, C.-L. Chen, and H.-C. Chang. A 520k (18900, 17010) array dispersion LDPC decoder architectures for NAND flash memory. *IEEE Trans. on Very Large Scale Integration (VLSI) systems (TVLSI)*, 24(4):1293–1304, 2015.

[14] J.-X. Hou and L.-P. Chang. Improving read performance for LDPC-based SSDs with adaptive bit labeling on Vmka_{th} mstates. In *Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023.

[15] Y.-L. Hsu, L.-W. Liu, Y.-C. Liao, and H.-C. Chang. GC-like LDPC code construction and its NN-aided decoder implementation. *IEEE Open Journal of Circuits and Systems (OJCAS)*, 2024.

[16] K. Ishimaru. Future of non-volatile memory-from storage to computing. In *Proc. of IEEE International Electron Devices Meeting (IEDM)*, 2019.

[17] S. Jaffer, K. Mahdaviani, and B. Schroeder. Rethinking WOM codes to enhance the lifetime in new SSD generations. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2020.

[18] S. Jaffer, K. Mahdaviani, and B. Schroeder. Improving the reliability of next generation SSDs using WOM-v codes. In *Proc. of 20th USENIX Conference on File and Storage Technologies (FAST)*, 2022.

[19] J. Kang, Q. Huang, L. Zhang, B. Zhou, and S. Lin. Quasi-cyclic LDPC codes: An algebraic construction. *IEEE Trans. on Communications (TCOMM)*, 58(5):1383–1396, 2010.

[20] A. Khakifirooz, S. Balasubrahmanyam, R. Fastow, K. H. Gaewsky, C. W. Ha, R. Haque, O. W. Jungroth, S. Law, A. S. Madraswala, B. Ngo, V. Naveen Prabhu, S. Rajwade, K. Ramamurthi, R. S. Shenoy, J. Snyder, C. Sun, D. Thimmegowda, B. M. Pathak, and P. Kalavade. 30.2 a 1Tb 4b/cell 144-tier floating-gate 3D-NAND flash memory with 40MB/s program throughput and 13.8Gb/mm2 bit density. In *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2021.

[21] B. S. Kim, J. Choi, and S. L. Min. Design tradeoffs for SSD reliability. In *Proc. of the 17th USENIX Conference on File and Storage Technologies (FAST)*, 2019.

[22] J. Kim, D.-h. Lee, and W. Sung. Performance of rate 0.96 (68254, 65536) EG-LDPC code for NAND flash memory error correction. In *Proc. of IEEE International Conference on Communications (ICC)*, pages 7029–7033. IEEE, 2012.

[23] Y. Kou, S. Lin, and M. P. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Trans. on Information Theory (TIT)*, 47(7):2711–2736, 2001.

[24] S.-H. Kuo. Ultra MMI: an LDPC decoder that doubles throughput at end-of-life. https://old.flashmemorysummit.com/Proceedings2019/08-07-Wednesday/20190807_CTRL-202-1_Kuo.pdf.

[25] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology-CRYPTO' 90*, pages 109–133. Springer Berlin Heidelberg, 1991.

[26] Y. Lee, H. Yoo, I. Yoo, and I.-C. Park. 6.4 gb/s multi-threaded BCH encoder and decoder for multi-channel SSD controllers. In *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2012.

[27] J. Li, S. Lin, K. Abdel-Ghaffar, W. E. Ryan, and D. J. Costello. Globally coupled LDPC codes. In *Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE, 2016.

[28] J. Li, Q. Wang, P. P. C. Lee, and C. Shi. An in-depth analysis of cloud block storage workloads in large-scale production. In *Proc. of IEEE International Symposium on Workload Characterization (IISWC)*, 2020.

[29] Y.-C. Liao, C. Lin, H.-C. Chang, and S. Lin. A (21150, 19050) GC-LDPC decoder for NAND flash applications. *IEEE Trans. on Circuits and Systems I: Regular Papers (TCSI)*, 66(3):1219–1230, 2018.

[30] S. Lin and D. Costello. *Error control coding: fundamentals and applications.* Prentice Hall, 2004.

[31] Y.-M. Lin, C.-L. Chen, H.-C. Chang, and C.-Y. Lee. A 26.9 k 314.5 mb/s soft (32400, 32208) BCH decoder chip for DVB-S2 system. *IEEE journal of solid-state circuits (JSSC)*, 45(11):2330–2340, 2010.

[32] C.-Y. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, and C. R. Das. SOML read: Rethinking the read operation granularity of 3D NAND SSDs. In *Proc. of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[33] L.-W. Liu, M.-H. Yuan, Y.-C. Liao, and H.-C. Chang. A 38.64-gb/s large-CPM 2-KB LDPC decoder implementation for NAND flash memories. *IEEE Open Journal of Circuits and Systems (OJCAS)*, 3:180–191, 2022.

[34] S. Liu and X. Zou. QLC NAND study and enhanced Gray coding methods for sixteen-level-based program algorithms. *Microelectronics Journal*, 66:58–66, 2017.

[35] Y. Lv, L. Shi, Q. Li, C. Gao, Y. Song, L. Luo, and Y. Zhang. MGC: Multiple-gray-code for 3D NAND flash based high-density SSDs. In *Proc. of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[36] Y. Lv, L. Shi, C. J. Xue, Q. Zhuge, and E. H.-M. Sha. Latency variation aware read performance optimization on 3D high density NAND flash memory. In *Proc. of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2020.

[37] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann. The devil is in the details: Implementing flash page reuse with WOM codes. In *Proc. of 14th USENIX Conference on File and Storage Technologies (FAST)*, 2016.

[38] R. Micheloni, R. Ravasio, A. Marelli, E. Alice, V. Altieri, A. Bovino, L. Crippa, E. Di Martino, L. D'Onofrio, A. Gambardella, et al. A 4Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36mb/s system read throughput. In *Proc. of IEEE International Solid State Circuits Conference-Digest of Technical Papers (ISSCC)*, 2006.

[39] S. Nie, Y. Zhang, W. Wu, and J. Yang. Layer RBER variation aware read performance optimization for 3D flash memories. In *57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[40] J. Park, M. Kim, M. Chun, L. Orosa, J. Kim, and O. Mutlu. Reducing solid-state drive read latency by optimizing read-retry. In *Proc. of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[41] T. Pekny, L. Vu, J. Tsai, D. Srinivasan, E. Yu, J. Pabustan, J. Xu, S. Deshmukh, K.-F. Chan, M. Piccardi, et al. A 1-Tb density 4b/cell 3D-NAND flash on 176-tier technology with 4-independent planes for read using CMOS-under-the-array. In *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2022.

[42] R. Pletka, N. Papandreou, R. Stoica, H. Pozidis, N. Ioannou, T. Fisher, A. Fry, K. Ingram, and A. Walls. Improving NAND flash performance with read heat separation. In *Proc. of International Symposium on*

Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2020.

[43] T. Richardson. Error floors of LDPC codes. In *Proc. of the Annual Allerton Conference on Communication Control and Computing*, volume 41, pages 1426–1435, 2003.

[44] W. Ryan and S. Lin. *Channel codes: classical and modern*. Cambridge university press, 2009.

[45] Y. Song, Y. Lv, and L. Shi. DECC: Differential ECC for read performance optimization on high-density NAND flash memory. In *Proc. of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023.

[46] R. Stoica, R. Pletka, N. Papandreou, N. Ioannou, S. Tomic, and H. Pozidis. Enabling 3D QLC NAND flash. *IBM Corporation, Flash Memory Summit*, 2019.

[47] M. H. Tadayon, A. Tasdighi, M. Battaglioni, M. Baldi, and F. Chiaraluce. Efficient search of compact QC-LDPC and SC-LDPC convolutional codes with large girth. *IEEE Communications Letters*, 22(6):1156–1159, 2018.

[48] A. Tasdighi, A. H. Banihashemi, and M.-R. Sadeghi. Efficient search of girth-optimal QC-LDPC codes. *IEEE Trans. on Information Theory (TIT)*, 62(4):1552–1564, 2016.

[49] A. Tasdighi and E. Boutillon. Integer ring sieve for constructing compact QC-LDPC codes with girths 8, 10, and 12. *IEEE Trans. on Information Theory (TIT)*, 68(1):35–46, 2021.

[50] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu. MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *Proc. of 16th USENIX Conference on File and Storage Technologies (FAST)*, 2018.

[51] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi. Gradient descent bit flipping algorithms for decoding LDPC codes. *IEEE Trans. on Communications (TCOMM)*, 58(6):1610–1614, 2010.

[52] D. Wang, L. Wang, X. Chen, A. Fei, C. Ju, and Z. Wang. Construction of QC-LDPC codes based on pre-masking and local optimal searching. *IEEE Communications Letters*, 22(6):1148–1151, 2017.

[53] X. Wang, G. Dong, L. Pan, R. Zhou, and I. Stievano. Error correction codes and signal processing in flash memory. *Flash Memories*, pages 57–82, 2011.

[54] C. Wu. LDPC for SSD - from theory to practice. https://www.inc.cuhk.edu.hk/wp-content/uploads/INC/default/files/seminars/slides/SSD_LDPC%20Talk.pdf, 2022.

[55] H. Xu, D. Feng, R. Luo, and B. Bai. Construction of quasi-cyclic LDPC codes via masking with successive cycle elimination. *IEEE Communications Letters*, 20(12):2370–2373, 2016.

[56] G. Yadgar, E. Yaakobi, and A. Schuster. Write once, get 50% free: Saving SSD erase costs using WOM codes. In *Proc. of 13th USENIX Conference on File and Storage Technologies (FAST)*, 2015.

[57] G. Yang, M. Zhang, P. Guo, X. Zhan, S. Yang, X. Zhao, X. Guo, P. Sang, J. Wu, F. Wu, et al. High-precision error bit prediction for 3D QLC NAND flash memory: Observations, analysis, and modeling. *IEEE Trans. on Computers (TC)*, 2025.

[58] X. Yu, J. He, B. Zhang, X. Wang, Q. Li, Q. Wang, Z. Huo, and T. Ye. Interleaved LDPC decoding scheme improves 3-D TLC NAND flash memory system performance. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 42(11):4191–4204, 2023.

[59] W. Zhang, J. Kang, Z. Dong, and Y. Zhu. RS-LDPC concatenated coding for NAND flash memory: Designs and reduction of short cycles. In *IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP)*, 2020.

[60] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang. LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives. In *Proc. of the 11th USENIX Conference on File and Storage Technologies (FAST)*, 2013.