IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 6, 2020, accepted August 28, 2020, date of publication September 3, 2020, date of current version September 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3021498

A Stepwise Rate-Compatible LDPC and Parity Management in NAND Flash Memory-Based Storage Devices

SEUNG-HO LIM[1], (Member, IEEE), JAE-BIN LEE[1], GEON-MYEONG KIM[1], AND WOO HYUN AHN[2]

[1]Division of Computer Engineering, Hankuk University of Foreign Studies, Seoul 02450, South Korea
[2]School of Software, Kwangwoon University, Seoul 01897, South Korea

Corresponding author: Woo Hyun Ahn (whahn@kw.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government [Ministry of Science and ICT (MSIT)] under Grant NRF-2019R1F1A1057503, the Hankuk University of Foreign Studies Research Fund, and the Research Grant of Kwangwoon University in 2020.

ABSTRACT The storage capacity of the NAND flash memory has increased rapidly, and accordingly, the error rate for data writing and reading to the flash memory cell has also escalated. Error-correcting code (ECC) modules, such as low-density parity-check (LDPC), have been applied to flash controllers for error recovery. However, since the error rate increases rapidly, compared to the aging factor and program/erase (P/E) cycle, fixed ECCs and parities are inappropriate methods for resolving this proliferating error, according to the P/E cycle. Therefore, the design of a dynamic ECC scheme and a proper ECC parity management system to increase the lifespan of flash memory storage devices remains in great demand. Herein, an LDPC encoding and decoding scheme is designed to obtain a step-by-step code rate according to the P/E cycle by applying a stepwise rate-compatible LDPC. In addition, an ECC parity management scheme for the increasingly excessive stage-wise ECC is proposed to reduce management and read/write operational overheads. The ECC management scheme also includes the ECC cache system. The proposed LDPC, as well as its management system, will improve the recovery ability of the NAND flash storage device according to the P/E cycle, while it can reduce system read and write overheads due to additional parity data growth.

INDEX TERMS NAND flash memory, flash storage, P/E cycle, RC LDPC, PCHK, parity, ECC cache.

I. INTRODUCTION

The NAND flash memory has been subjected to inter-cell interference due to the rapid increase in its integration. As the number of bits stored per cell increases, the error occurrence rate has remarkably increased due to the interference [1], [3]. Moreover, the longer the time of use, the worse the physical characteristics of the cell become, which leads the error occurrence rate escalates [2], [4]. Program/erase(P/E) cycle is a sequence of events in which data is written, and it is used as a criterion for endurance of flash storage device. Recent advances in 3D stacking technology have enhanced the ease of securing cell-to-cell spacing compared to the conventional 2D approach, resulting in a slight increase in the maximum available P/E cycle. However, despite 3D technology, the P/E cycle remains at approximately thousands, which is a significant disadvantage of flash storage-based computer systems [5], [6].

In general, error correction code(ECC) module is embedded in the flash controller to recover from errors that occur during read and write processes. The ECC module uses parity to correct errors, in which the parity is generated from a specific encoding method. Recently, low density parity check(LDPC) [9], [13] has been applied to flash controller as a ECC module. The LDPC generates a codeword that includes parity and source data in which parity is generated through matrix operation on a parity check matrix called PCHK. In the flash memory, a page consists of a data area and a spare area, and user data is stored in the former, while the parity is stored in the latter. Each time a page-write action occurs, parity is generated by applying LDPC encoding to the source user data, and it is stored together data in the page.

The associate editor coordinating the review of this manuscript and approving it for publication was Tuo-Hung Hou.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

The amount of parity can be determined according to the code rate of ECC module. The code rate is defined as a ratio between data and codeword, so lower code rate represents larger amount of parity. The LDPC is designed and implemented as a hardware module within flash memory controller, so the (LDPC) code rate is generally fixed. However, the fixed code rate could be unsuitable for the flash storage since it raises the error rate as the P/E cycle increases. That is, generating large ECC parities for early P/E cycles, which incurs few errors, result in storage overhead in capacity as well as parity management overheads due to excessive ECC parity. In the latter P/E cycle wherein many errors occur, the error recovery rate decreases due to the relatively small amount of parity. It is required to apply a dynamic LDPC codec scheme, which can generate ECC parity dynamically in accordance with the number of P/E cycles. In order to increase the error correction rate, there has been an existing studies [29], [30] in which a method of generating additional parity is applied. If more parity is created to increase the error correction rate, the additional parity cannot be stored in the page along with the data. The parity should be stored in a separate space separate from the data, which results in additional read/write overheads to the flash device.

We propose the design of an LDPC scheme that has several different code rates according to the P/E cycle, as well as its parity management system. The method designed here is based on the quasi-cyclic(QC) rate-compatible(RC) LDPC, in which it is an LDPC technique that can change code rate step-by-step, according to predefined P/E cycles. Since it is designed to have an RC manner, it is possible to expand parities step by step to existing parity. In the decoding step, by applying the decoding step by step, only the parity required for each step can be read, so read overhead can be significantly reduced. Compared to the existing RC LDPC method, the proposed scheme changes a wider code rate to the base PCHK and reduces the bias of 1's element within the extended PCHK.

In addition to the LDPC scheme, we also designed ECC parity management system. The excessive parities generated from higher step LDPC should be stored elsewhere from the original data if the length of parity exceeds that of the page unit. In our system, parts of parity that exceed the page length are stored and managed separately from the page data written. That is, those excessive parities are aggregated together in another flash blocks. Mapping information for the separated parities is maintain in the FTL. Furthermore, an ECC cache management scheme is applied to excessive parities. At the decoding stage, decoding is applied in the order of low-level to high-level in accordance with RC LDPC. For each level, the exact data and parity are retrieved to decode at the level, so if decoding is successful at low level of LDPC, it is unnecessary to retrieve the excessive parity for the high-level decoding. Thus, it reduces the excessive parity read overhead, while preserving error recoverability with parities for high-level LDPC.

The organization of this paper is as follows: The background and related works are described in Section 2. The proposed stepwise RC LDPC technique and its management system are explained in Sections 3 and 4, while the experimental results of the proposed system are described in Section 5. The conclusions are presented in Section 6.

II. BACKGROUND AND RELATED WORK

A. NAND FLASH MEMORY AND MANAGEMENT SYSTEM

NAND flash memory has two distinct unit called page and block. Page is the unit for read and write, while block is the unit for erase. The typical page size of NAND flash memory is one of that 4KB, 8KB, or 16KB, varying according to the structure of the cell. The block is composed of dozens of pages, i.e., 64 pages or more. The NAND flash memory performs read, write, and erase operations. The actual data transfer occurs on a read or write command, while the erase operation does not cause the actual data transfer. In general, a NAND flash storage device includes a plurality of flash memory chips, as well as a flash memory controller that handles requests from the host computer. The read request extracts data from the flash memory chip to the host computer, whereas the write request enters the data received from the host to the flash memory chip.

In NAND flash memory usage, there are two major issues: Firstly, the read and write units are different from the erase units, and secondly, the write operation requires that the corresponding flash memory cell be in erased status. A special NAND flash memory software, called the flash translation layer (FTL) [14], is used to solve this mismatch, and it is run on the flash controller. The primary role of the FTL is to manage the translation tables that convert logical addresses on the host into physical addresses in the flash memory. In addition, the FTL is responsible for NAND flash memory block wear leveling, garbage collection, and error handling. The main functions of the FTL are address and mapping management between the logical address of the host data and the physical address of the flash memory. Several research works have been performed to develop FTLs and their mapping management [15]–[19].

B. ENDURANCE OF FLASH MEMORY

Increasing the flash memory density narrows the spacing between adjacent cells, thereby heightening the interference between cells, which has led to a sharp increase in data errors that can occur during read and write processes. In general, there is a P/E cycle, which is an indicator of the flash memory life. To write data to flash memory cells, an erase operation should have been performed first for that cells. So, program and erase operation make a sequence in which data is written to flash memory. As the P/E cycle increases, the characteristics of the cell become worse, which leads to a rise in the rate at which errors occur during the read/write operation. Raw Bit Error Rate(RBER) is defined as the fraction of bits that contain incorrect data [1], [23]. As the P/E cycle increases,

the RBER increases rapidly, when P/E cycle is exceeded over some point, error recovery is impossible so it can no longer be used. The available P/E cycle was maintained at approximately 100,000 for the single-level cell (SLC), but at less than 10,000 for the multi-level cell (MLC), as well as 1000 to thousands for the triple-level cell (TLC) and quad-level cell (QLC).

One of the main factors affecting the flash endurance is the P/E cycle, which is due to the rapid increase in RBER with the P/E cycle. However, due to diversification of the manufacturing process of flash memory, factors for RBER variation have also diversified [20]–[23]. Even in the same P/E cycle, the RBER varies depending on the location and state of the flash cell in the block or page. For example, as the retention time of data increases, the RBER of the page increases. In multi-level cell flash memory, RBER varies depending on the location of the bit. Also, RBER is diversified by stacking layer in recent 3D stack flash memory compared to planner 2D flash memory [23].

To recover from data errors in read and write operations, ECC modules, such as LDPC, are embedded in the NAND flash memory controller. When data is received from the host, the NAND flash controller generates a parity through the ECC module, combines the parity and data to make up a codeword, and stores the codeword in the NAND flash memory. The pages of the NAND flash memory are composed of data and spare areas. Generally, ECC parity data are stored in the spare area in conventional flash memory devices. If the ECC module creates an ECC parity larger than the spare region, the excessive ECC must be stored in a different area than the same page. This is referred as *excessive ECC* in this paper.

Some previous studies have been conducted to improve the lifetime of NAND flash memory devices by designing ECC parity management schemes rather than directly dealing with ECC encoding/decoding methods. In [29], as the P/E cycle increases, the code rate is elevated by reducing the data area and increasing the spare area, thereby improving the correction capability in the event of an error. Since these studies do not consider the actual ECC encoding scheme, they may be inefficient in the actual parity generation and restoration methods. Zhou designed the ECC cache management system for the excessive ECC and analyzed its (excessive ECC) performance and the ECC cache [30]. However, the ECC encoding and decoding technique was not applied. Additionally, because it was not an ECC cache management system based on dynamic ECC according to the P/E cycle, the overhead for the excessive ECC was not considered according to the P/E cycle.

C. LDPCs FOR FLASH MEMORY

While there are many error correction algorithm, two typical ECC scheme have been applied to flash memory controllers, that is Bose-Chaudhuri-Hocquenguem (BCH) and LDPC [7], [9], [32]. The BCH code forms a cyclic ECC class constructed using polynomials in finite fields. Any level of error correction is possible and includes efficient codes for uncorrelated error patterns. The open source Linux platform provides the BCH ECC coding driver, and the chip designer develops a chip that implements BCH bit error coding. Recently, however, it may be unsuitable for the error correction of a recent status of flash memory errors.

LDPC is a data encoding method that generates codewords, including parity data from source data, using a parity check matrix (PCHK). In coding theory, the PCHK for a linear code is defined as a matrix describing a linear relationship that components of a codeword must satisfy matrix multiplication. The set of valid codewords for a linear code can be specified by describing a PCHK H having m rows and n columns. The codewords can be the vector x of length n, in which $Hx = 0$. Note that the PHCK for a given linear code is not unique and can be constructed by various methods, which usually involves a random positions of where to put 1 in the PCHK [9]–[11].

For the binary PCHK, most of the elements constituting the matrix are zeros, and only a few elements have a value of one. The size of the PCHK is determined according to a code rate, a ratio of data to codeword. The LDPC code has been widely adopted in flash memory controllers because of its excellent error correction. LDPC has received a lot of attention in the industry, with many presentations made at recent flash conferences such as [13], [26], [33]. For instance, in [28], [34], an LDPC decoding method optimized for the flash memory was proposed, and in [35], Tanakamaru *et al.* showed that the lifespan of the solid-state drive (SSD) could be extended by 10 times using LDPC coding. K. Zhao proposed three techniques to mitigate the response time delay of LDPC decoding [8].

When source data is encoded by LDPC with specific PCHK, the generated parity are usually mixed with source data in resulting codeword. Therefore, if LDPC encoding of the same data is performed on LDPC with different PCHK having different code rate, the resulting codeword has totally different order of bit sequence. On the other hand, rate-compatible(RC) LDPC is possible to extract increasing parity by parity in accordance with code rate changes. Many research works have proposed the RC LDPC encoding technique to improve the real-time error correction of communication channels where noise can change rapidly in real time [36]–[38], as well as memory systems, such as NAND flash memory [26]–[28], [39]–[41]. The RC LDPC basically extends the matrix depending on the base PCHK. Most of the existing methods for extending the base PCHK entail the addition of an identity matrix laterally in the lower-right diagonal direction of the base PCHK, which distributes a value of one only to rows added below the base PCHK. PCHKs extended in this manner may be deteriorated compared to the general PCHK, wherein the error correction rate of the extended PCHK has the same code rate because the value of one is biased. There were some research works for error management of 3D flash memory which has large variation of RBER. [23] designed RBER-aware lifetime prediction scheme for 3D flash memories by applying machine learning

technologies, and in [31], they proposed multi-granularity progressive LDPC in accordance with RBER variations of 3D flash memories.

The proposed LDPC scheme in this paper is based on rate-compatible manner. Although rate-compatible coding schemes already developed and applied to many dynamic error management systems, our approach is different from others at some points. First, compared to the existing RC LDPC method, the proposed scheme adds a wider code rate to the base PCHK. Second, in the structure of the extended PCHKs, element value 1 is not biased to specific columns. As a result, the RC-based LDPC developed in this paper does not suffer much from the performance degradation in comparison with the PCHK without RC. It is suitable for a NAND flash memory that has an increasing error rate according to the P/E cycle and a fixed input/output unit of page. This paper also deals with parity management and caching method, as well as LDPC encoding and decoding scheme.

III. A STEPWISE RATE-COMPATIBLE QC LDPC

A. BASE PCHK AND LDPC

The LDPC generates a codeword including parity and data through matrix multiplication on a PCHK or a PCHK corresponding generator matrix from source data. As described above, the PCHK, especially binary PCHK, is a matrix having a small elemental value of one, and a Tanner graph [10] is applied to LDPC coding based on the PCHK. If the element of one is concentrated in a specific row or column with the PCHK, the characteristic of the Tanner graph may be deteriorated. So the quasi-cyclic (QC) [12] method can be applied to enable the efficient distribution of one's element. Figure 1 describes QC matrix with size of 3 by 3. The PCHK can be made of clustering several QC matrixes in which each QC matrix is different from others by reordering columns. The schematic diagram of the basic PCHK constructed by applying the quasi-cyclic pattern is also shown in Figure 1.

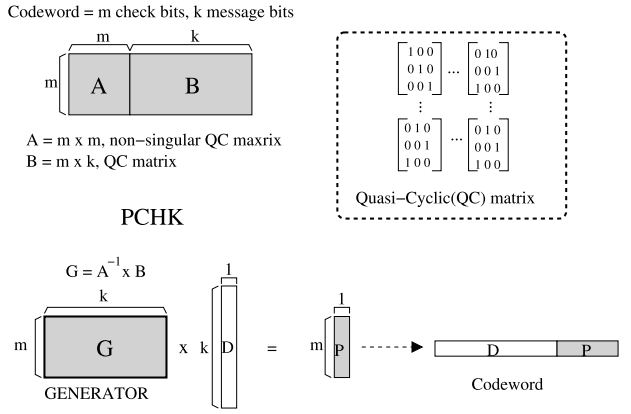

FIGURE 1. Structure of base PCHK with quasi-cyclic(QC) sub-matrix and codeword generation from the corresponding generator matrix.

The PCHK is of a $(m + k) \times m$ matrix structure, where the sizes of the message(data) and parity bits are k bits and m bits, respectively. The small unit of the PCHK has quasi-cycle pattern with different one's position. The PCHK is divided into A and B parts, where A is composed of a $m \times m$ non-singular QC matrix and B is a $m \times k$ QC matrix. The LDPC encoder can create a generator matrix based on the PCHK configured by multiplying A^{-1} and B matrix. It consequentially generates a codeword having a size of $(m + k)$ consisting of a data and parity through a matrix multiplication operation of the generator matrix and message bits. The resulting codeword is divided into D and P, where D is exactly same as data and P is parity, so LDPC can separate message from parity if we construct PCHK described in Figure 1. The basic structure of the PCHK is called *base PCHK* in this paper.

B. EXTENDED PCHK FOR RATE-COMPATIBLE LDPC

To construct an RC LDPC from the base PCHK, we should note the following issues when creating a base PCHK: Firstly, when a codeword is generated through LDPC encoding, data bits and parity bits must be completely separated. Secondly, to have an RC feature, the matrix A must be configured to have a non-singular feature, i.e., it must be configured in a form having an inverse matrix.

After configuring the base PCHK to have this feature, Figure 2 shows how a PCHK can be extended to separate the added parity from the existing one. As shown in the figure, extending the PCHK from the base PCHK entails the addition of specific columns and rows to the base PCHK. That is, to generate additional e parity bits to existing m message and k parity bits, we add the e column and e row to the base PCHK. In the figure, *O* and *I*, denoted by the additional columns, represent the zero matrix and the identity matrix, respectively. The D and E matrix added to the row are composed of a QC matrix, such as A or B. For the D matrix, like the A matrix, a non-singular matrix can be added to further expand it. The structure of the PCHK made by extension from base PCHK is referred as *extended PCHK* in this paper. If LDPC encoding is performed using the extended PCHK, a separate parity is created from the base parity generated by the existing base PCHK.

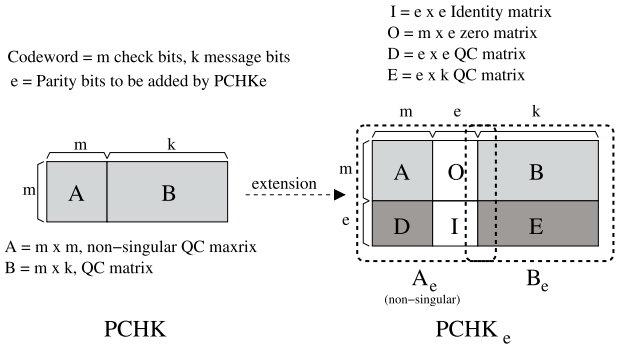

FIGURE 2. The pchk extension method for stepwise rate-compatible parity.

We can also construct next-step RC PCHK by extending the extended PCHK as same manner. Figure 3 shows a

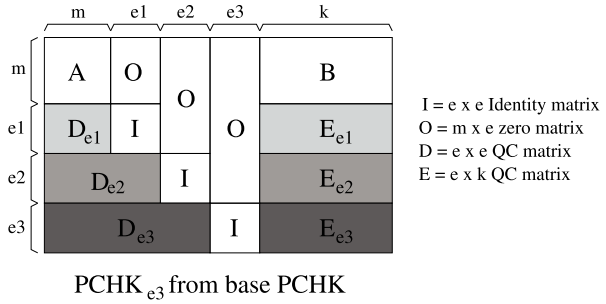

FIGURE 3. Extended PCHK structure using N stepwise rate-compatible QC LDPC technique. It can generate N separate parities ranging from base to N steps.

diagram that extends a three-step RC PCHK from the base PCHK. Although the figure describes only the three-step extension method, the n-step PCHK extension can be applied in the same way. In the figure, e1, e2, and e3 represent the first, second, and third extensions, respectively. As shown in the figure, to extend the RC PCHK for generating e_n parity data separated from the existing codeword from the (n-1)th step PCHK, the O, I, D_{en}, and E_{en} matrices are added to the proper location from (n-1)th step PCHK. The D_{en} and E_{en} should be basically configured to have a QC matrix as well, and D_{en} should be configured to be non-singular matrix. The D_{en} QC matrix added at each step can be configured such that the distribution of elements having a one value is relatively unbiased to some columns. This can minimize the performance reduction for error correction compared to the previous RC LDPC scheme. It is because the values of one are less biased on the columns, making the characteristics of the Tanner graph tolerable.

C. EXTENDED PARITY GENERATION

We can obtain G and G_e generator matrices from PCHK and $PCHK_e$, as well as generate parity data for each step through the matrix operation of source data. Figure 4 illustrates the results for a codeword generated by performing LDPC encoding and the generator matrix using three-step RC PCHK

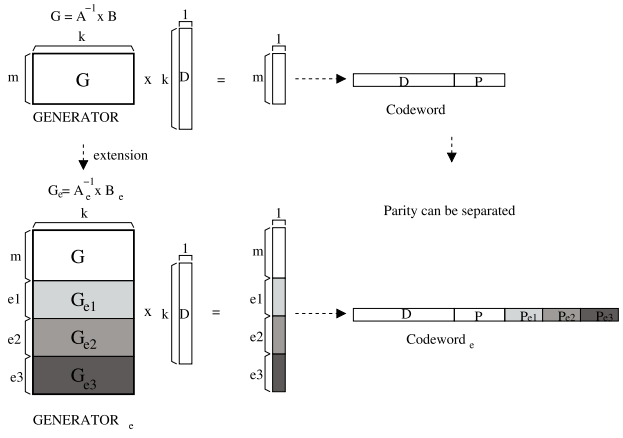

FIGURE 4. Structure to generate N separated parities from generator matrix, which correspond to N step extended PCHK.

extensions. As shown in the figure, when using an RC LDPC with the base PCHK and the extension based on it, the original message data and the parity data can be separated, and the parity can be generated such that it remains unmixed even if the encoding is extended.

In our system, the PCHK extension method is configured to have a large difference in code rate at each step. That is, PCHK extensions are applied to some limited number of P/E cycle changes such as two or three, so we can get large parity enough to correct increasing errors due to the P/E cycle increases. As a result, there are large difference in code rate between each PCHK step. This large code rate difference between each PCHK extension step is appropriate, considering the structure of the NAND flash memory, such as page size.

IV. EXCESSIVE ECC MANAGEMENT SYSTEM

A. BLOCK ALLOCATION AND FTL MAPPING MANAGEMENT

In general, parity generated from LDPC encoding with base PHCK is stored to flash page with its corresponding data, specifically, data is stored in data area and parity is stored in spare area, respectively. However, the parity generated by LDPC encoding with the extended PCHK is larger than base parity, so it cannot be stored in one page together. The parity generated by extended PCHK can be completely divided into several parities according to the extension level. For instance, parity of LDPC encoding with two step PCHK is divided into base parity, extended parity one and two. In this case, base parity can be stored to flash page together its corresponding data, while others cannot be stored together. We refer those parities as *excessive ECC* parity.

In our system, the LDPC encoding level is determined by the predefined P/E cycle threshold values. When each block reaches a certain P/E cycle value, the PCHK of the corresponding level is applied. If the 3-step extended PCHK technique is applied, it has two P/E cycle thresholds as configuration parameters to apply PCHK in each step. FTL manages the P/E cycle value of each block, and FTL increases the P/E cycle value of each block whenever an erase operation of the block occurs. At the beginning, every block uses LDPC coding with base level PCHK. When a block reaches the threshold P/E cycle value, 1-step extended LDPC coding is applied to the block. Likewise, when the block reaches to the next level threshold P/E cycle value, next-level extended LDPC coding is applied.

The excessive ECC parity should be stored in a separate space from data. For this, firstly, we allocate blocks that collect and store the excessive ECC parities separately from the data blocks that store data. Secondly, mapping information about an address where the excessive ECC parity is placed is managed in the mapping table. Finally, the cache management for excessive ECC parity is added to the system to reduce the read/write overhead due to the excessive parity. The block containing excessive ECC parity is referred as ECC

block, and cache for excessive ECC parity is referred as ECC cache, respectively. Those management schemes exist within FTL.

The overall excessive parity management and operation is described in Figure 5, in which one step extended PCHK is applied to LDPC module. When data is written by host, at first, FTL allocates a physical address for the logical address of the data transmitted from a host. Then, a parity is generated through the LDPC module, in which the generated parity is divided into two parts, base parity and excessive parity since one step extended PCHK is applied. As shown in Figure 5, the data and base parity is stored to physical page assigned by FTL, while the excessive parity cannot be stored together. Since read and write operation of flash memory is done in units of pages, the excessive ECC parity must be collected to a buffer until it is filled with other excessive ECC parities, so the excessive parity is written to the flash memory along with other excessive ECC parities of other data. For instance, if an excessive parity of 256 bytes is generated for 4KB data, 16 excessive ECC parities for 16 data pages can be stored in one page together. When the buffer is filled with the excessive parities, it is written to a specific page of the ECC block. The corresponding physical address that stores the excessive parities is managed in the FTL separately from the existing mapping table. As shown in Figure 5, the parity address information is defined with page number and its offset within the page. That information is added to the existing FTL mapping table.

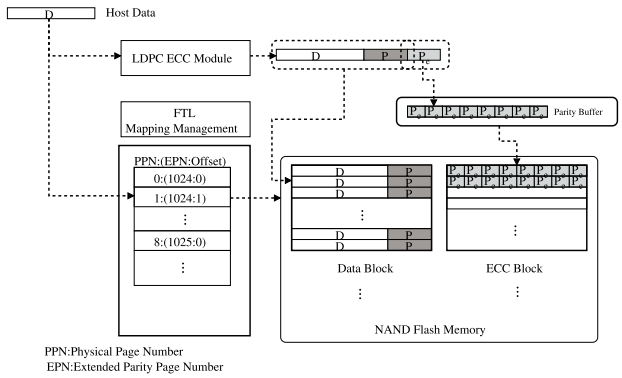

FIGURE 5. The write process of host data in a stepwise rate-compatible LDPC-based flash storage management system.

Although the figure only depicts the extended parity by one level of the extended LDPC, it can be generalized and used in the same way even if next-level expansion is applied. Even when the expansion step for the extended LDPC scheme is increased, the base ECC parity is stored with data together in one page, while other excessive ECC parities are stored in a part of a specific page in the ECC block.

B. CACHE MANAGEMENT

In general, flash storage devices have internal DRAM caches for user data caching to improve IO responsiveness and reduce flash memory read/write operations. In our system, to reduce read overhead for excessive parity caused by extended PCHK, a portion of DRAM is allocated and managed as a cache for ECC, that is, excessive parities. As a result, DRAM is divided into data cache and ECC cache, in which data cache is for caching user data while ECC cache is used for caching excessive parities.

The extended parity generated by the extended LDPC increases the error correction rate of the data, but additional flash read and write overhead occurs because the excessive ECC parity is stored in a separate page from data. The DRAM cache structure, which consists of a data cache region and an ECC cache region, is illustrated in Figure 6. In the ECC cache region, only the excessive ECC parities are cached, while base ECC parities are not cached since those are read from data together as a page unit. The data cache and ECC cache is managed by separate replacement lists and replacement policies. That is, the excessive ECC parity associated with a particular data is not always caching or uncaching altogether. Since the same physical DRAM is shared between the data cache and the ECC cache, the size of each cache can affect each other, so the cache size between the two regions is configured so that it can be changed in the device settings.

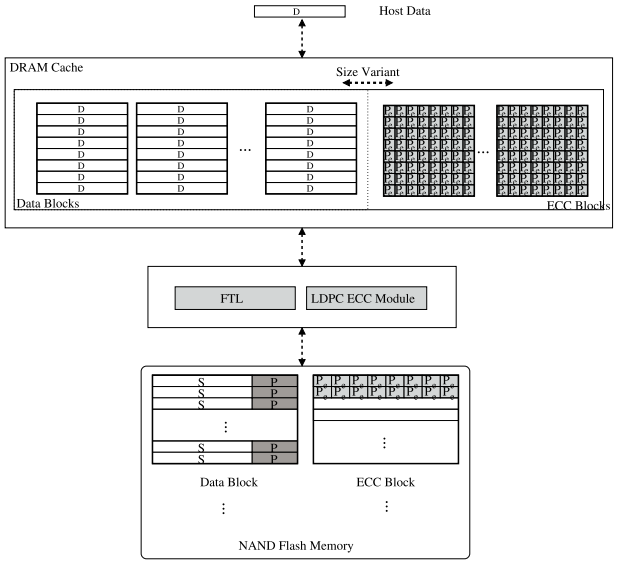

FIGURE 6. The DRAM cache structure, which consists of a data cache region and a parity cache region.

Although the effective replacement policies from many existing studies [24], [25] can be applied, the basic least recently used (LRU) replacement policy is applied in this system. The data cache replacement and its corresponding ECC cache operation is shown in Figure 7. When the requested data is not in the data cache, the victim page is selected from LRU list. and it is flushed to flash if the victim is dirty. After then, we should consider whether its excessive ECC should be generated or not. If the victim's excessive ECC is already exist, which means the victim's data is already in flash and it is not set to be dirty, we do not need to generate excessive ECC. In this case, cache replacement is

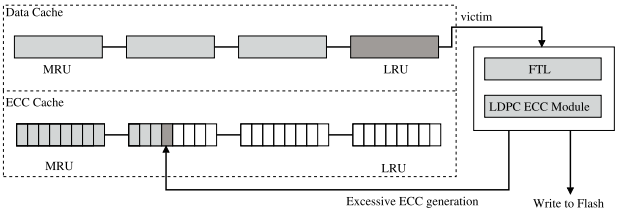

FIGURE 7. The data cache replacement and its corresponding ECC cache operation.

done without generation of excessive ECC. If the victim's excessive ECC is not exist, the excessive ECC is generated by extend LDPC encoding and it is cached in the ECC cache.

The read and write operations are carried out as follows: In the case of data write operations, data are simply cached in the data cache area immediately if it is possible without any eviction of other cached data. In this case, no LDPC encoding is performed since there is no flash write. The ECC parity is actually generated when the cached data are evicted into flash. At this time, the excessive ECC parity is generated by LDPC if extended PCHK is applied. The generated ECC parity is cached into ECC cache. For data read operations, there are several cases according to the cache status of the requested data, as shown in Figure 8. If data exist in the data cache, it can be sent directly from the cache to the host without a flash read. If the data are not in their cache, they should be retrieved from the flash memory. The excessive ECC parity for the corresponding data may be in the ECC cache even though the data is not cached, since the ECC cache is managed by a separate list and replacement policy from the data cache. If LDPC decoding with extended PHCK is required during retrieving data from flash, it can be performed without additional flash reading if the corresponding ECC parity exists in the ECC cache region. It reduces read overhead that is derived from the excessive ECC parity.

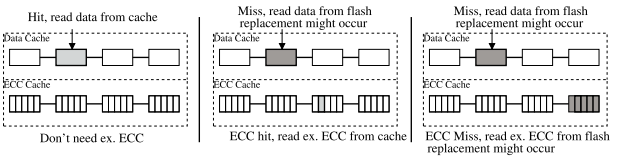

FIGURE 8. Several cases of data read operations with data cache and ECC cache.

C. DATA RETRIEVAL AND LDPC DECODING PROCEDURE

According to the system, the LDPC encoding with n-step extended PCHKs is applied for the specific predefined P/E cycle. For example, if the flash device is set to apply two PCHK extensions for blocks with P/E cycle 2000 and 4000, LDPC encoding with base PCHK is applied to flash blocks having P/E cycles under 2000, and the one-step extended LDPC encoding is applied to blocks having P/E cycles from 2000 to 4000, while the two-step extended LDPC encoding is applied to blocks having P/E cycles over 4000.

On the other hand, when data are read from flash memory, they are restored by performing a step-by-step LDPC decoding from the base PCHK to the maximum step extended PCHK. The procedure of reading and decoding the data is as follow. When a read occurs from the flash memory, firstly, a page in which the data are stored is read and LDPC decoding with base PHCK is performed with the data and the base ECC parity stored in the spare region of the page. If decoding is successful with any inaccuracy, the data can be transmitted to the host normally without any error. In this case, since the data were read without the help of excessive ECC, no additional work associated with the excessive ECC is necessary. If decoding fails with only the base ECC parity, further decoding should be done with next-step PCHK, which entails that the excessive ECC parity is required. The excessive ECC is first checked whether it exists in the ECC cache or not. If it exists therein, LDPC decoding is performed by combining the excessive ECC parity with previously read data and base parity. If decoding is successfully performed by those, the data can be transferred safely without additional flash memory read operation for the excessive ECC parity. If the parity does not exist in the ECC cache, an additional read operation should be performed to read the excessive ECC parity from the NAND flash memory. Then, the ECC parity is combined with the associated data and the base parity, and LDPC decoding is performed using them. In this case, the decoding error is recovered although there is additional read overhead for reading excessive ECC parity.

In summary, through the stepwise RC LDPC encoding and decoding with extended PCHK as well as the excessive ECC management with cache, adaptive data restoration in accordance with P/E cycle can be applied to enhance error recoverability.

V. EVALUATION

A. IMPLEMENTATION AND EVALUATION SETUP

The proposed stepwise RC LDPC-based ECC module and its excessive ECC management system were implemented in FlashSim simulator [42], [43], which is SSD device simulator that models the NAND flash memory chip, flash controller, DRAM, and several FTLs. However, since it does not have the ECC module, we added the ECC module by implementing the LDPC simulator [44]. The added LDPC module can define the desired PCHK and can encode and decode the source data by using the generator matrix derived from the PCHK. For FTL, the page-level mapping management-based DFTL [18] existing in this simulator was used, and it was assumed that all the mapping tables existed in the DRAM. In the FTL, additional mapping information for excessive parities was added to the existing page-mapping table. In addition, caches for data and ECC parity have been implemented in the DRAM buffer. Basically, the cache performs caching and replacement on a block basis, and the LRU replacement policy is applied. Data and ECC cache are independently managed with separate replacement lists. The characteristics of the

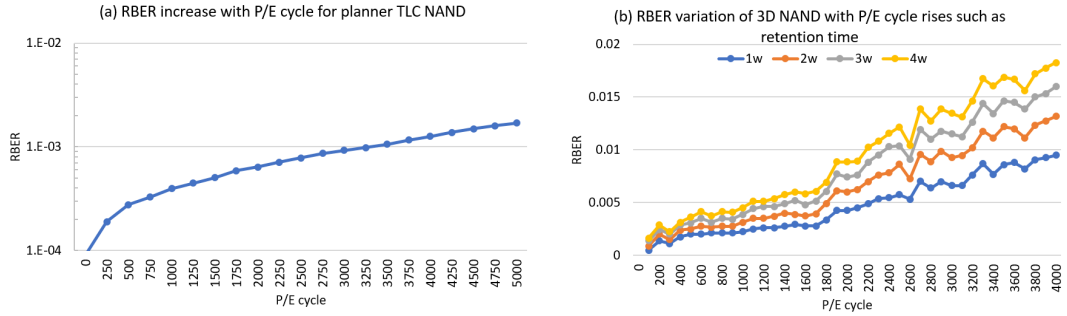

FIGURE 9. Evaluated RBER according to the P/E cycles for planner TLC NAND [3] and RBER variation of 3D TLC NAND according to retention time [23].

flash memory applied to the simulator are as follows: The page size is 4 KB data region with additional 1KB spare region, the block size is 256 KB, and it has a total capacity of 1 GB.

This simulator has a model for the latency for the page read and write, and block erase operations, however, it has no error rate model. It is more essential to measure the error rate that occurs when reading and writing each page, thus, we added a model that generates errors when reading and writing for each page. For this, in the simulator, each time the page is read, the raw bit error rate (RBER) corresponding to the P/E cycle of the page is generated. RBER is the bit error rate before using ECC, which reflects the basic stability state of NAND Flash cell itself. In order to apply the aspect of large RBER variability, the RBERs of both of planner TLC NAND and 3D TLC NAND, which was measured in the previous studies, was used in our experiments. Figure 9(a) shows the RBER according to the P/E cycle measured in the planner TLC NAND flash memory [3], while the RBER variation according to the P/E cycle of the 3D flash memory is plotted in Figure 9(b) [23]. Although Figure 9(b) shows the variability of RBER according to retention in the same P/E cycle, since the RBER variation per each P/E cycle is widely distributed with four kinds of retention period, which can represent the RBER variability in accordance with other cases such as the layers of the 3D stack and the bit position in the flash memory cell.

B. EVALUATION OF THE STEPWISE EXTENDED PCHK

Firstly, to show the feasibility of the stepwise LDPC ECC scheme, uncorrectable bit error rate (UBER) of three-step LDPC with extended PCHK is compared with that of legacy QC LDPC, by changing P/E cycle value. The experiment was conducted on a total of four LDPC configurations, including three legacy LDPCs with code rates of 0.8, 0.67, and 0.57, and the extended PCHK LDPC scheme. The extended PCHK having three stages of LDPC with code rates of 0.8(4 parity-20 data), 0.67(8 parity-24 data), and 0.57(12 parity-28 data) were constructed. The PCHK extensions are applied for P/E cycles with 2000 and 4000. That is, the base PCHK with a code rate of 0.8 is applied to P/E cycles from 0 to 1500, and PCHK_{e1}, a one-step extension, is applied to have a code rate of 0.67 for P/E cycles from 2000 to 3500. For the P/E cycles from 4000 to 5000, PCHK_{e2}, a two-step extension, is applied to have a code rate of 0.57 based on PCHK_{e1}. Since the PCHK for a given code is not unique, we created ten extension cases to evaluate the performance variation of different extended PCHK examples. The stepwise extended LDPC is compared with three legacy LDPCs [12], [44] having code rates of 0.8, 0.67, and 0.57, respectively. These are independent PCHKs. These are created so that they are not related to each other. We also created ten cases for each legacy LDPC and estimated the UBER for each of them.

The experiments are described as follows. While increasing the P/E cycle from 0 to 5000, in steps of 500, hundreds of thousands of pages are written and read for each P/E cycle. For write operations, random data is encoded with each of LDPCs and errors are injected to the encoded data according to the RBERs for the P/E cycle. For read operations, each of LDPCs performs decoding, attempts to recover an error caused by the RBER, and measures the number of times that it cannot be decoded meaning that error occurs. These values are denoted as the uncorrectable bit error rate (UBER) [1].

Figures 10 and 11 plot the experimental results of the decoding experiments for six LDPCs with RBER values of Figure 9(a). Figure 10 shows the UBER value versus P/E cycles for each LDPC configuration. Among ten candidates, three cases of extended PCHK scheme were separately plotted from Figure 10(a) to 10(c) to compare each with the legacy LDPC, and the average UBERs for ten cases of each LDPC configuration are represented in the Figure 10(d). As shown in the figures, in the case of the LDPC with extended PCHK, even though there is some variation, UBER is lowered in the P/E cycle wherein PCHK expansion is applied. This is because the LDPC with extended PCHK improves decoding ability by increasing parities at the specific P/E cycle, so UBER is lowered. For P/E cycles from 0 to 1500, the Ext. PCHK show similar UBER with legacy LDPC with 0.8 code rate, while the UBER of Ext. PCHK is lowered at P/E cycle 2000, wherein one step extension is applied for LDPC encoding. The UBER of LDPCs having extended PCHK is almost same as legacy LDPC having 0.67 code rate during P/E cycles from 2000 to 3500. As some cases,

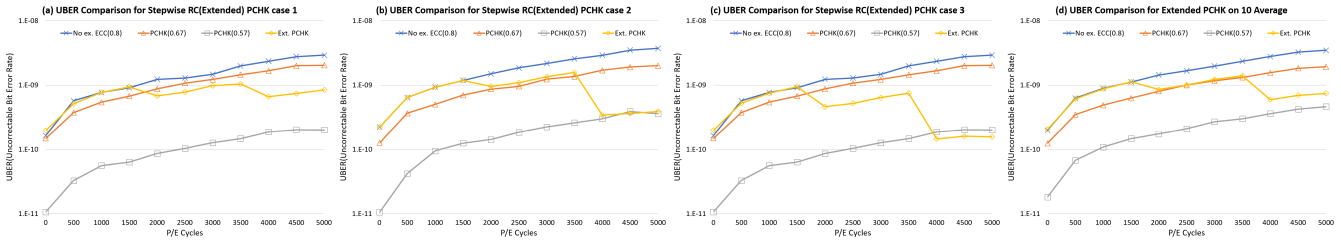

FIGURE 10. Estimated uncorrectable bit error rate(UBER) results for the RBER of LDPC configurations according to changes in the P/E cycle; No ex. ECC(0.8), Legacy LDPC(0.67), Legacy LDPC(0.57), and Stepwise RC(Extended) PCHK cases.

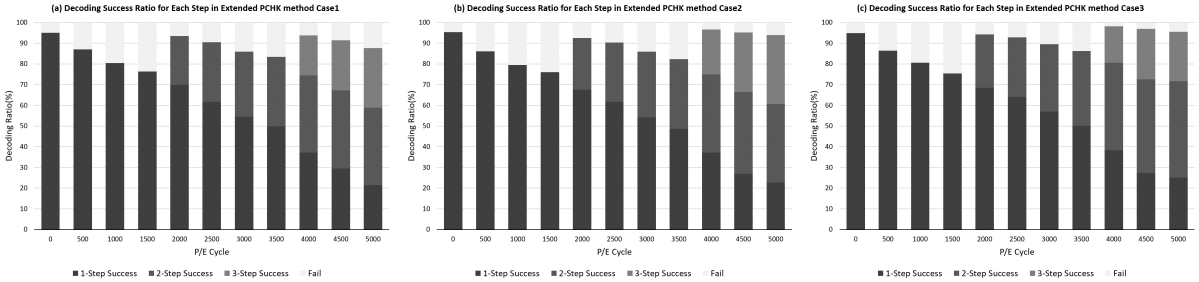

FIGURE 11. The decoding probability distribution according to the P/E cycle for stepwise RC LDPC Case1, 2, and 3. For each P/E cycle, the probability was plotted by measuring which PCHK decoding was a success in each step for stepwise RC LDPC to analyze at which PCHK stage decoding was successful.

the UBER of Ext. PCHK is lower than that of legacy, and some cases the UBER of Ext.is higher than that of legacy. In our experiments, the average UBER of first expansion of PCHK is almost same as that of legacy LDPC, which means that our Ext. PCHK can provide almost same error recoverability for the first expansion. For the second expansion of PCHK, the average UBER of Ext. PCHK is worse than that of legacy LDPC, as shown in the Figure 10(d). It is the limit of rate-compatible manner-based coding scheme. However, the UBER gap is not much to not consider of applying our Ext. PCHK method, and even some case of Ext. PCHK could give better UBER than that of legacy, as shown in the case 3.

Since the decoding procedure of extended LDPC is performed from lowest PCHK to highest PCHK until the decoding is successful, the decoding success level is different from each other for the Ext. PCHK scheme, which is the strength of Ext. PCHK. Thus, to determine which step was successful, the level of PCHK was counted when decoding was successful at each decoding. Figure 11 shows the decoding probability according to the P/E cycle for three extension cases. As shown in the figure, during P/E cycles from 0 to 1500, only the base PCHK exists, so all decoding probability depend on the base PCHK. For P/E cycles from 2000 to 3500, the decoding probability of the base PCHK gradually decreases, while the decoding of one-step extended PCHK compensates for the failure of the base PCHK. The distribution of the decoding probability for the base PCHK, $PCHK_{e1}$, and $PCHK_{e2}$ is plotted for P/E cycles from 4000 to 5000, in which we identify that the decoding probabilities using base PCHK and $PCHK_{e1}$ occupy large portion. From these results, it is identified that even though the RBER rises as the P/E cycle increases, in many cases, decoding is success with LDPC having low PCHK, i.e., the base PCHK or lower step of the PCHK. That is, even if the RBER is high, it is worth using lower level of PCHK for successful decoding in many cases. If the low PCHK fails, a decoding with higher PCHK can be used. Since lower PCHK has less parity and less decoding overhead than higher PCHK, if decoding is successful using a low level PCHK with small amount of parity, the overhead of the flash device can be reduced.

We also performed experiments that applied extended PCHK LDPC to flash memory with RBERs having variation for each P/E cycle as plotted in Figure 9(b). Figure 12 plots the experimental results of the decoding experiments for four LDPCs, that is No. Ex. ECC, legacy LDPCs with code rate of 0.67 and 0.57, and Ext. PCHK. In the case of Ext. PCHK, the extend PCHK method having average error recoverability of ten cases in the previous experiment was applied. To analyze the effect of RBER variation, error rate was applied with evenly distribution of the four RBER variations for each P/E cycle, then LDPC decoding was performed to correct the errors and counted the decoding success of for each RBER. Figure 12(a) plots the number of successful decoding for each step of Ext. PCHK for each of the four RBERs. As shown in the figure, even in the same P/E cycle, the number of successful decoding in the same step of Ext. PCHK varies according to the RBER, and this difference between RBER variations increases as the P/E cycle increases. However, the failed decoding in the lower steps can be decoded at the next step of Extend PCHK, as a result, we identify that

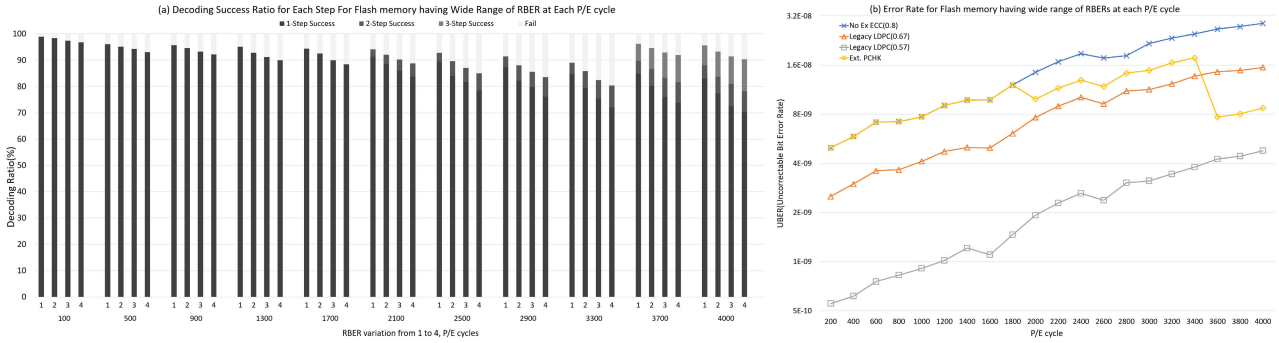

FIGURE 12. It plots the experimental results of the decoding experiments with variable RBERs in each P/E cycle for four LDPCs, No. Ex. ECC, legacy LDPCs with code rate of 0.67 and 0.57, and Extended PCHK.

the difference in the total number of decoding successes according to RBER variation for each P/E cycle decreases. Thus, it can be confirmed that the effect of RBER variation is reduced by applying extend PCHK. That is, it can be identified that the error recoverability using the additional parities of Extend PCHK is effective to cover the variance of errors due to RBER variation. Figure 12(b) shows the their average UBER value versus P/E cycles for each LDPC configuration. As can be seen in the figure, although the error recovery rate is reduced rather than that of the planner flash, the error recoverability of the Ext. PCHK improves as extended step increases.

C. EVALUATION OF FLASH DEVICE PERFORMANCE

1) EVALUATION OF READ

The stepwise LDPC provides increasing encoding according to the P/E cycle, and decoding performs step-by-step decoding. Since parities are additionally generated with the extended LDPC encoding, the parities increased by the LDPC extension adds overhead to read/write and storage. compared to existing system, our extended LDPC and its management system not only lowers errors by using extended parities according to P/E cycle, but also reduces read/write overheads for excessive parity through step-by-step decoding and cache management. We performed a read/write IO experiments to analyze the overhead of the proposed system by comparison with legacy LDPC and existing work [12], [30]. In these experiments, we analyzed the IO overhead according to the IO patterns for the six LDPC configurations applied in the previous experiments. In the LDPC configurations, the legacy represents the basic QC LDPC having code rate 0.8, which does not generate any excessive parities. The SCORE [30] is existing work that creates and manages excessive parities all the times regardless of the P/E cycle. SCORE (0.67) and SCORE (0.57) indicate SCORE with code rates of 0.67 and 0.57, respectively. Ext. PCHK is the method in which stepwise extended PCHK is applied.

In this experiment, the flash simulator was set as same configuration as above experiments except cache configuration. Specifically, for DRAM cache, the size of the DRAM for data cache was set to 64 MB, while the ECC cache size was changed to 0, 8 MB, 16 MB, and 32 MB to analyze the caching effect for the excessive ECC parity. To analyze performance variations according to the request pattern, four IO request patterns were conducted: Random Request 100%, Sequential Request 100%, Random70%-Sequential30%, and Random30%-Sequential70%. In the case of a random request, the requests having page size were randomly generated for the whole logical LBA area, while the requests having page size were continuously generated for the sequential request. These four request patterns were applied to each of the six LDPC configurations, that is, for each LDPC configuration, the four request patterns were performed by increasing the P/E cycles from 0 to 5000. For all configurations and experiments, before the experiment, a write request was performed once in all logical addresses of the flash simulator, and then 50,000 IO operations were performed, while repeating write request and read request according to request pattern.

The additional read overhead generated by the excessive ECC was measured during IO operations for four request patterns, however, among the four request patterns, we plotted the amount of additional ECC page reads only for random request in Figure 13, since random request patterns show worst case read overhead. In the figure, the x-axis represents the P/E cycle value, while the y-axis represents read amplification factor, which means the degree of amplification of the amount of internal read operations compared to data read requests from the host side. Since internal read operations are amplified due to the additional reads for excessive ECC, this metric can show how much the read overhead due to the excessive ECC.

To analyze the overhead in terms of the pure flash memory read operation, an experiment without ECC cache was performed, and the results are shown in Figure 13(a). As shown in the figure, it can be confirmed that read amplification stays at one in the case of the legacy, which is one that does not have any excessive ECC parities. For the two existing LDPC techniques SCORE(0.67) and SCORE(0.57), we can identify that the amount of read has been doubled compared to the legacy one. For random read, the excessive parities

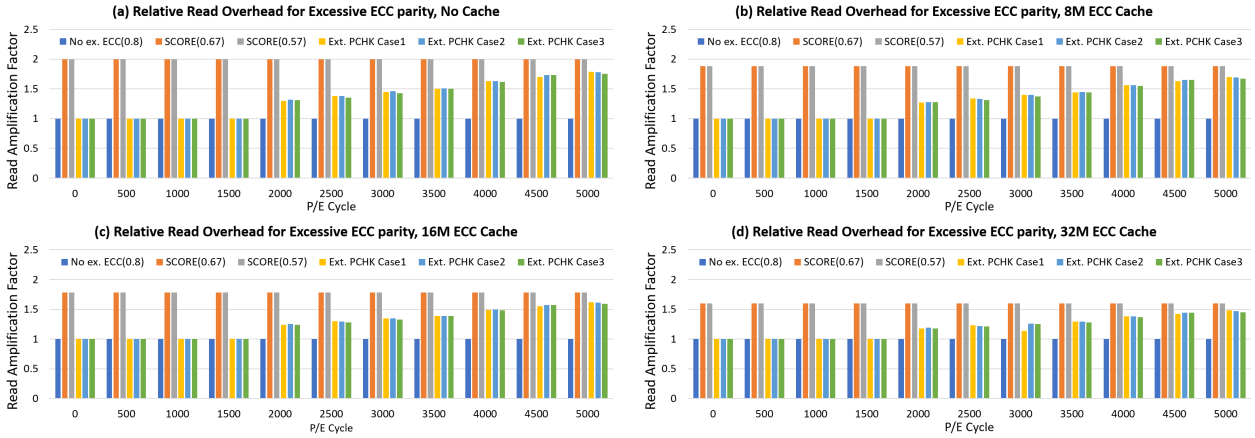

FIGURE 13. It shows the experimental results of additional ECC reading amount compared to data reading as P/E cycle increases for random 100% requests, at cache size 0M, 8M, 16M, and 32M, respectively. The read amplification represents the degree of amplification of the amount of internal read operation cause by excessive ECC read compared to data read of the host side.

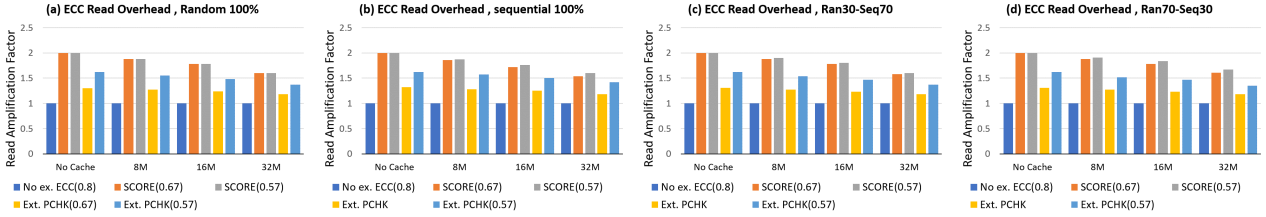

FIGURE 14. It shows the read amplification factors of the five LDPC configurations for each ECC cache 0,8,16, and 32MB size, for request patterns of Random100, Sequential100, Random70-Sequential30, Random30-Sequential70. The Data cache is fixed with 64MB.

must be read along with the corresponding data to decode, which results in double of reads. Since the excessive ECC page associated with a data page is also randomly chosen due to the random request patterns, the read amplification became two at worst case.

On the other hand, for the three cases to which the extended LDPC, is applied, i.e., Ext. PCHK case 1∼3 it can be seen that read amplification increases gradually in each step wherein each extension is applied. Since there is no excessive ECC in the P/E cycles from 0 to 1500, where no extension is applied, each of the data read is made as one page read. In the P/E cycles of the 2000∼3500 section, where the first step extension is applied, the read amplification is ranging from 1.31 to 1.51, which is much lower than that of SCORE(0.67). This is because in the case of the Ext. PCHK, LDPC decoding is firstly attempted by reading only data and parity of the spare with the data, i.e., only one page. If the decoding is successful with only the page, there is no need to read the page that contains excessive ECC. The read amplification increases as the P/E cycle grows because the decoding success with only base parity decreases due to an increase in the RBER, and the read of excessive ECCs correspondingly rises. We identify that the P/E cycle of 4000∼5000 with two-step extension shows a similar trend. Since this section has a low probability of LDPC decoding due to high RBER, the read amplification increases from 1.63 to 1.79 in this section.

We did experiments with increasing cache size from 8M to 32MB to identify the cache effects on read overhead for excessive ECC. The read amplification for cache sizes of 8M, 16M, and 32M are plotted in Figure13(b), Figure13(c), and Figure13(d), respectively. As shown in the figure, when ECC caching is used, the read amplification of the existing LDPC decreases from two as the cache size increases. Similarly, in the case of the proposed extended LDPC, it is observable that read the amplification gradually decreases as the cache size increases. Since the ECC cache area is used for caching the excessive ECC, the number of flash memory read operations on the excessive ECC data can be reduced as ECC cache size increases.

It is noteworthy that even the Ext. PCHK having less ECC cache has less read overhead than the existing method that uses more ECC cache. It means that some area of ECC cache can be altered to data cache to help improve data read/write without any loss of read overhead in comparison with SCORE. To further analyze the cache effect, the read amplification factor was plotted by changing the ECC cache size by 0,8,16, and 32MB for each request pattern, while the cache size for data cache is fixed with 64MB. Figure 14 shows the read amplification factors of the five LDPC configurations for each ECC cache 0,8,16, and 32MB size. In the figures, legacy system, that is, No ex. ECC (0.8) shows no additional reads for the excessive ECC, so there is no

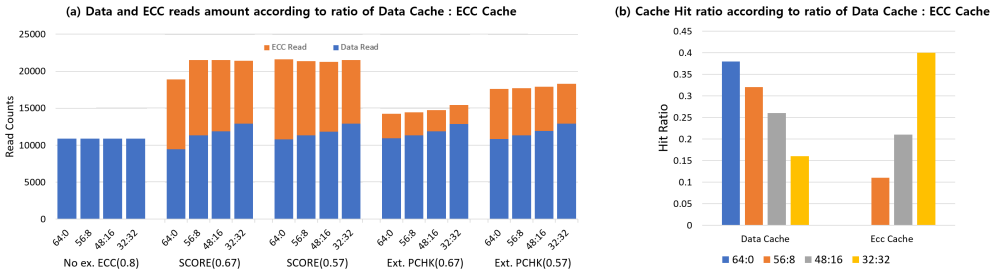

FIGURE 15. It shows the amount of data reads and ECC reads when the random read operations are performed while the ratio of Data Cache:ECC Cache is changed to 64:0, 56:8, 48:16 and 32:32, respectively, with the 64MB total cache size, and the hit ratios of the data cache and ECC cache at this time.

read overhead. In the case of SCORE, as the ECC cache size increases, read overhead decreases, but not much lessen. In numerical value, read amplification of SCORE, which was approximately two when there was no cache, improved the performance by 20% to 1.6 when the cache size was 32M. On the contrary, for the extended LDPC with two step extension, that is Ext. PCHK(0.67), the read amplification is from 1.31-1.32 in the case of no cache, and it decreases to 1.16-1.19 with the 32M cache size, which represents a performance improvement of approximately 25%∼35%. For three step extension, that is Ext. PCHK(0.57), If there is no cache, the read amplification is 1.62-1.63, but if the cache size is 32M, the read amplification is lowered to approximately 1.34-1.38, which presents a performance improvements of approximately 16%∼19% in case of three-step extensions. The three step extensions give more read overhead than two step extensions, however it also gives high error correction capability.

From the results, it is noteworthy that even the Ext. PCHK having no cache is better than that of the original that has 32M ECC cache. It means that even if the ECC cache is not used, the read overhead of Ext. PCHK is lower than legacy system having cache. Certainly, we have known by the previous research [30] that ECC cache can reduce read overhead for excessive ECC, however, the extended LDPC scheme gives more effects on lowering read overhead than using ECC cache.

In addition, we also measured the read performance with fixed total cache size to see the performance impact with different size distribution between data cache and ECC cache. In each of LDPC system, the amount of read was measured by distributing the ratio between data cache and ECC cache from 64:0 to 32:32. Figure 15(a) shows the amount of data reads and ECC reads when the random read operations are performed while the ratio of Data Cache:ECC Cache is changed to 64:0, 56:8, 48:16 and 32:32, respectively, with the 64MB total cache size, and Figure 15(b) is plotted by measuring the hit ratios of the data cache and ECC cache at this time. As shown in Figure 15(a) and Figure 15(b), as the data cache size decreases, the hit ratio for the data cache decreases, so the amount of data read increases, while as the ECC cache increases, the ECC cache hit ratio increases which reduce the amount of ECC reads. It should be noted that the amount ECC reads for Ext. PCHK scheme is much smaller than that of SCORE, so if the ECC cache is reduced in size and altered it to data cache, the overall read performance can be more improved.

In summary, there are two factors influencing the performance improvement for the extended LDPC scheme. Firstly, since many decoding operations can succeed with lower levels of PCHKs, higher-level excessive ECC does not need to additionally read in many cases. This reduces the reading operation itself for the excessive ECC parities. Secondly, the ECC cache activation is relatively low due to the reduced number of reads. By properly reducing the size of ECC cache and increasing it to the data cache, it can contribute to improvements on data caching, in comparison with the existing work, that is, fixed excessive ECC and caching scheme.

2) EVALUATION OF SPACE, WRITE AND GC OPERATION

In general, flash storage is designed to have more physical space than the user's space. The space other than the user's valid data is regarded as an invalid area or a free area. The extended LDPC scheme can be subject to space overhead due to excessive parities generated by the LDPC. The storage overhead can be estimated as displayed in Figure 16, in which storage becomes more demanded as the excessive parities increases. However, in terms of storage usability, user effective data is generally lower than storage capacity, so, more important issue is that the actual amount of physical writes increases compared to the effective amount of writes due to the excessive parities, rather than space overhead. As the

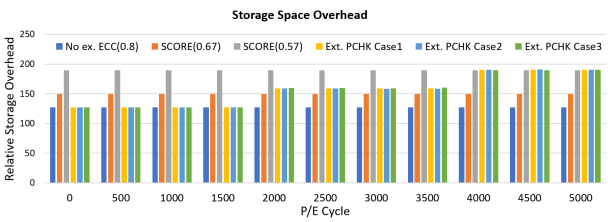

FIGURE 16. It shows storage overheads for LDPC configurations over P/E cycle.

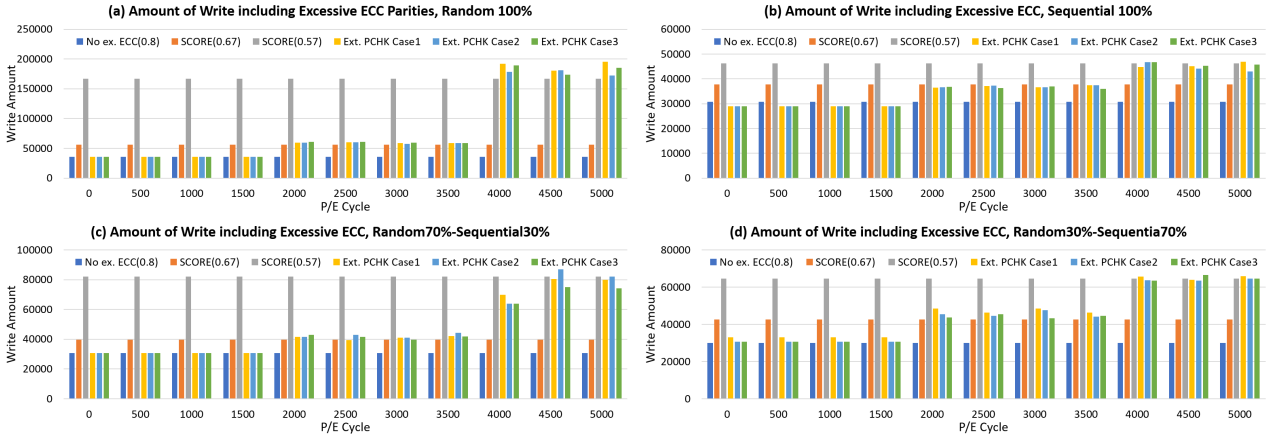

FIGURE 17. Plots of the write count generated in the flash storage device when the host write operation is performed while increasing the P/E cycle from 0 to 5000 by 500, for each request pattern; random100, sequential100, random70-sequential30, random30-sequential70.

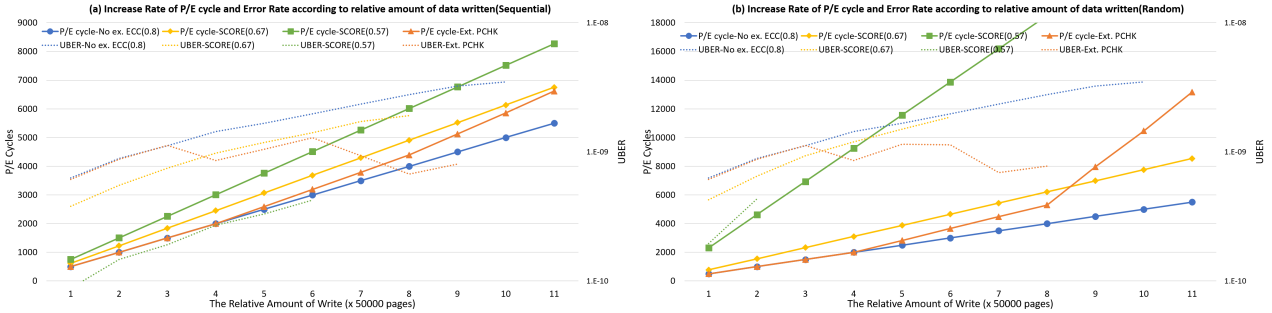

FIGURE 18. It plots the rate of increase of the P/E cycle and relative frame error rate according to the increase in the amount of excessive ECC writes.

amount of writing increases, the space also consumes, which has a negative effect for flash storage usage such as GC, due to the decrease in the free area. So, as another terms of space overhead, we measured how much flash write increased compared to host write, and how GC overhead was, caused by the excessive ECC.

To analyze the write overhead, we have measured the amount of writes in flash device for the host write requests. The measured the amount of writes for four request patterns are depicted in Figure 17. In the figure, the results for No ex. ECC represents base guideline for the amount of writes since it does not any excessive ECC. The LDPCs having excessive ECC that does not provides rate-compatible manner along with P/E cycles, such as SCORE(0.67) and SCORE(0.57), generate write overheads regardless of P/E cycles. Specifically, SCORE(0.57) generates larger amount of writes than SCORE(0.67) due to generating larger excessive parities. On the other hand, it is noticeable that, in case of Ext. PCHK method, the write overhead increases each time the expansion level is raised. For the extended LDPC cases, the write overhead increases at the P/E cycle of 2000 wherein the first-step expansion is applied, and the number of writes further rises in the P/E cycle of 4000 where the second-step expansion is applied. That is, as the P/E cycle of the device increases, the system that has extended LDPC module gradually increases the overhead of the write operation.

The Figures from Figure 17(a) to Figure 17(d) show the amount of writes according to the request patterns;random, sequential, ran70-seq30, and ran30-seq70. In the case of a random request, if the amount of excessive ECC is large, the overall number of writes increases rapidly, as shown in Figure 17(a). Figure 17(b) reveals that in the case of sequential requests, the number of writes does not significantly increase regardless of the amount of excessive ECC. Since random requests have a lower cache hit ratio than sequential requests, more writes occur for the flash storage side. The write amplification factor is much greater for random request because of the huge garbage collection (GC) overhead. Furthermore, the free region is much reduced due to the excessive ECC, which adds more overhead of the write operation.

To see the lifetime change of the Flash device due to the excessive amount of ECC writing, the rate of increase of the P/E cycle according to the increase in the amount of excessive ECC write is estimated and is plotted in Figure 18(a) and Figure 18(b) for sequential and random requests, respectively. In the figure, the x-axis represents the relative amount

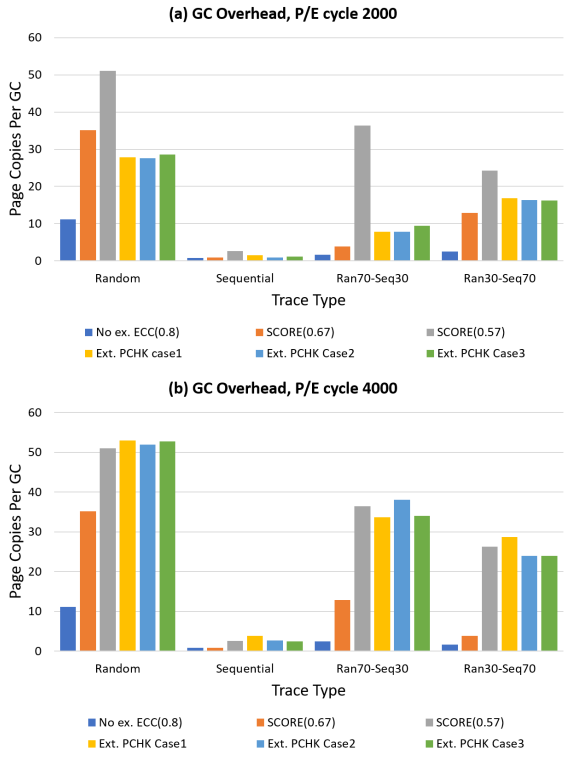

FIGURE 19. For the LDPC settings, the number of valid page copies per GC was plotted for each of the four request patterns; random100, sequential100, random70-sequential30, random30-sequential70.

of written pages performed in the simulator. The left y-axis in the figure shows the increase in P/E cycle as the amount of written pages increases, and the right y-axis plots the UBER for the corresponding P/E cycle. As shown in the figure, the higher the amount of excessive ECC writes, the higher the P/E cycle increase rate as the amount of writes increases. In particular, in the case of the Ext. PCHK, since the amount of excessive excessive ECC is amplified whenever the step of the Ext. PCHK is increased, the increase rate of the P/E cycle is getting higher than that of the no excessive ECC method. This trend can be seen by comparing Figure 18(a) and Figure 18(b) that the random request is more prominent than the sequential request. However, looking at the error rate plotted together in the figure, it can be seen that the error rate of Ext. PCHK is lower than legacy method even though the P/E cycle of the Extend PCHK increases faster than the existing method. It means that the Ext. PCHK method has a lower UBER than legacy method at same amount of writing, and it also imply that Ext. PCHK method could lengthen the lifetime of the flash device than legacy method.

To analyze the GC overhead induced by excessive ECC, we measured the number of valid page copies per GC, which is a metric of the GC overhead. The GC algorithm used in the simulator is the greedy algorithm. At two P/E cycles 2000 and 4000, the number of valid page copies per GC for each of the four request patterns are measured for 6 LDPCs, and the results are plotted in Figure 19. As shown in Figure 19(a), in the case of the P/E cycle of 2000, it is shown that the SCORE(0.57) has the highest GC overhead for all request patterns while the SCORE(0.67) and extended LDPC have the similar amount of excessive ECC. Since There is one-step extended LDPC for P/E cycle of 2000, it gives similar degree of GC overhead with SCORE(0.67). For the P/E cycle of 4000, as shown in Figure 19(b), the extended LDPC also shows a GC overhead similar to that of SCORE(0.57) since it is applied with two-step expansion. Incidentally, the reason that the GC overhead is too large might be a fundamental cause of the high storage consumption due to the excessive ECC, as well as the naive GC policy applied to the excessive ECC area. This poses a need for further research to study the effective GC policy for the excessive ECC region.

VI. CONCLUSION

One of the main problems of the flash storage is the increase in errors due to a rise in integration density. In particular, when the usage time of the storage device lengthens, i.e., the P/E cycle of the flash block increases, the error occurrence rate escalates. Moreover, the error rate increases rapidly compared to the aging factor and P/E cycle. Conventional fixed ECC and parity management schemes are inappropriate methods for the rapidly increasing errors according to the P/E cycle. In the early P/E cycle, excessive ECC parity degrades read/write performances since it has to be stored in a separate space, and each time data are read, the associated ECC must be read, which incurs ample overhead costs. On the other hand, less ECC parity in the later part of the P/E cycle reduces the ability of error recovery and shortens device life.

Herein, we designed and implemented a step-by-step RC LDPC coding technique that can generate appropriate excessive ECC for P/E cycle increase. Also, a flash storage system also proposed to manage the increasing excessive ECC parity. The stepwise RC LDPC increases the error recovery rate through adaptively increased parities according to the P/E cycle. In addition, since stepwise RC LDPC generates excessive ECC by the RC method, LDPC decoding can be firstly attempted by reading only one page of the stored data and the original ECC itself; and if this fails, additional decoding is performed by reading the excessive ECC on a separate page. The excessive ECC can also be managed by FTL and ECC cache. Thus, it is possible to reduce the read overhead much more than the conventional approach. Even if ECC cache is used, the stepwise RC LDPC system shows a much better at read performance than the fixed excessive ECC management system.

In the experiment, we have confirmed that the stepwise RC LDPC and its parity management system adaptively lowered the UBER according to the P/E cycle, and the read/write overhead is reduced in comparison with existing work. From a flash device perspective, read and write operations involve data transfer between the flash controller and the flash memory itself, and this latency has a significant effect on the

overall performance of the flash device. So, we experimented mainly with read and write operations. However, in terms of stepwise RC LDPC encoding and decoding, it is necessary to analyze the increased latency of LDPC encoding/decoding itself. Furthermore, the error analysis and correction method considering the characteristics of data retention and read disturbance of the memory cell should be considered. These will be our further work.

REFERENCES

[1] 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 *Proc. IEEE Int. Rel. Phys. Symp.*, Phoenix, AZ, USA, Apr. 2008, pp. 9–19.

[2] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Asheville, NC, USA, Oct. 2013, pp. 123–130.

[3] E. Yaakobi, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Maui, HI, USA, Jan. 2012, pp. 486–491.

[4] A. Tai, A. Kryczka, O. S. Kanaujia, and K. Jamieson, "Who's afraid of uncorrectable bit errors? Online recovery of flash errors with distributed redundancy," in *Proc. USENIX Annu. Tech. Conf.*, Renton, WA, USA, Jul. 2019, pp. 977–992.

[5] J. H. Yoon, G. Tressler, and H. Hunter, "3D-NAND scaling & 3D-SCM—Implications to enterprise storage," presented at the Flash Memory Summit, Aug. 2017.

[6] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation," in *Proc. Abstr. ACM Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, New York, NY, USA, Jun. 2018, p. 106.

[7] Micron Technology, Inc., "Enabling Software BCH ECC on a Linux platform," Tech. Note TN-29-71, Apr. 2012.

[8] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng, "LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 243–256.

[9] G. R. Gallager, "Low density parity check codes," in *Monograph*. Cambridge, MA, USA: MIT Press, 1963.

[10] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[11] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[12] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic LDPC codes for fast encoding," *IEEE Trans. Inf. Theory*, vol. 51, no. 8, pp. 2894–2901, Aug. 2005.

[13] J. Yang, "Novel ECC architecture enhances storage system reliability," presented at the Flash Memory Summit, Aug. 2012.

[14] Intel Corporation, "Understanding the flash translation layer (FTL) specification," Appl. Note AP-684, Dec. 1998.

[15] J. Kim, J. Min Kim, S. H. Noh, S. Lyul Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.

[16] J. U. Kang, H. Jo, J. S. Kim, and J. Lee, "A superblock-based flash translation layer for NAND Flash memory," in *Proc. 6th ACM & IEEE Int. Conf. Embedded Softw.*, Oct. 2006, pp. 161–170.

[17] S. W. Lee, W. K. Choi, and D. J. Park, "FAST: An efficient flash translation layer for flash memory," in *Proc. Int. Conf. Embedded Ubiquitous Comput.*, Aug. 2006, pp. 879–887.

[18] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. 14th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, Mar. 2009, pp. 229–240.

[19] D. Ma, J. Feng, and G. Li, "LazyFTL: A page-level flash translation layer optimized for NAND flash memory," in *Proc. ACM SIGMOD*, Jun. 2011, pp. 1–12.

[20] F. Wu, Y. Zhu, Q. Xiong, Z. Lu, Y. Zhou, W. Kong, and C. Xie, "Characterizing 3D charge trap NAND flash: Observations, analyses and applications," in *Proc. ICCD*, Oct. 2018, pp. 381–388.

[21] L. Shi, Y. Di, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting process variation for write performance improvement on NAND Flash memory storage systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 334–337, Jan. 2016.

[22] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: Improving NAND Flash lifetime by balancing page endurance," in *Proc. FAST*, 2014, pp. 47–59.

[23] R. Ma, F. Wu, M. Zhang, Z. Lu, J. Wan, and C. Xie, "RBER-aware lifetime prediction scheme for 3D-TLC NAND flash memory," *IEEE Access*, vol. 7, pp. 44696–44708, 2019.

[24] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2003, pp. 115–130.

[25] S.-Y. Park, D. Jung, J. Kang, J.-S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst.*, Oct. 2006, pp. 234–241.

[26] S. Qi, D. Feng, N. Su, W. Liu, and J. Liu, "A new solution based on multi-rate LDPC for flash memory to reduce ECC redundancy," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015, pp. 918–923.

[27] S. Bates, "Using rate-adaptive LDPC codes to maximize the capacity of SSDs," in *Proc. Flash Memory Summit*, Aug. 2013, pp. 1–12.

[28] Y. Zhang, C. Zhang, Z. Yan, S. Chen, and H. Jiang, "A high-throughput multi-rate LDPC decoder for error correction of solid-state drives," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Hangzhou, China, Oct. 2015, pp. 1–6.

[29] S. Wang, F. Wu, Z. Lu, Y. Zhou, Q. Xiong, M. Zhang, and C. Xie, "Lifetime adaptive ECC in NAND flash page management," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Lausanne, Mar. 2017, pp. 1253–1556.

[30] Y. Zhou, F. Wu, Z. Lu, X. He, P. Huang, and C. Xie, "SCORE: A novel scheme to efficiently cache overlong ECCs in NAND flash memory," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 4, Dec. 2018, Art. no. 60.

[31] Y. Du, Y. Zhou, M. Zhang, W. Liu, and S. Xiong, "Adapting layer RBERs variations of 3D flash memories via multi-granularity progressive LDPC reading," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[32] T. K. Moon, *Error Correction Coding, Mathematical Methods and Algorithms*. Hoboken, NJ, USA: Wiley, May 2005.

[33] E. Yeo, "An LDPC-enabled flash controller in 40 nm CMOS," presented at the Flash Memory Summit, Aug. 2012.

[34] R. Motwani and C. Ong, "Robust decoder architecture for multi-level flash memory storage channels," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Maui, HI, USA, Jan. 2012, pp. 492–496.

[35] S. Tanakamaru, Y. Yanagihara, and K. Takeuchi, "Over-10x-extended-lifetime 76%-reduced-error solid-state drives (SSDs) with error-prediction LDPC architecture and error-recovery scheme," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, 2012, pp. 424–426.

[36] T. V. Nguyen, A. Nosratinia, and D. Divsalar, "The design of rate-compatible protograph LDPC codes," *IEEE Trans. Commun.*, vol. 60, no. 10, pp. 2841–2850, Oct. 2012.

[37] Y.-H. Liu, "Rate-compatible QC-LDPC codes based on PEXIT," *Electron. Lett.*, vol. 54, no. 19, pp. 1120–1122, Sep. 2018.

[38] Y. Zhang, K. Peng, Z. Chen, and J. Song, "Construction of rate-compatible raptor-like quasi-cyclic LDPC code with edge classification for IDMA based random access," *IEEE Access*, vol. 7, pp. 30818–30830, Mar. 2019.

[39] H. Sun, W. Zhao, M. Lv, G. Dong, N. Zheng, and T. Zhang, "Exploiting intracell bit-error characteristics to improve min-sum LDPC decoding for MLC NAND flash-based storage in mobile device," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 8, pp. 2654–2664, Aug. 2016.

[40] K. Haymaker and C. A. Kelley, "Structured bit-interleaved LDPC codes for MLC flash memory," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 870–879, May 2014.

[41] P. Chen, K. Cai, and S. Zheng, "Rate-adaptive protograph LDPC codes for multi-level-cell NAND flash memory," *IEEE Commun. Lett.*, vol. 22, no. 6, pp. 1112–1115, Jun. 2018.

[42] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simulation*, Porto, Portugal, 2009, pp. 125–131.

[43] M. Bjorling. *Extended FlashSim*. Accessed: Jan. 15, 2019. [Online]. Available: https://github.com/MatiasBjorling/flashsim

[44] R. M. Neal. *Software for Low Density Parity Check Codes*. Accessed: Jan. 15, 2019. [Online]. Available: https://github.com/radfordneal/LDPC-codess

SEUNG-HO LIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Division of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), in 2001, 2003, and 2008, respectively. He worked in the memory division of Samsung Electronics Company Ltd., from 2008 to 2010, where he was involved in developing a high performance SSD (solid state disk) for server storage systems. He is currently a Professor with the Division of Computer Engineering, Hankuk University of Foreign Studies. His research interests include operating systems, embedded systems, non-volatile memory, flash storage systems, mobile computing, high performance computing, interconnect networks, and parallel computing.

JAE-BIN LEE received the B.S. degree from the Division of Computer Engineering, Hankuk University of Foreign Studies. His research interests include LDPC encoding and decoding, SSD architecture, distributed storage systems, and embedded architecture for AI.

GEON-MYEONG KIM received the B.S. degree from the Division of Computer Engineering, Hankuk University of Foreign Studies. His research interests include LDPC codec, embedded storage system, non-volatile memory systems, and big data processing systems.

WOO HYUN AHN received the B.S. degree from the Division of Electrical Engineering, Kyungbook National University, in 1996, and the M.S. and Ph.D. degrees from the Division of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), in 1998 and 2003, respectively. He worked in the Software Research Center, Samsung Electronics Company Ltd., from 2003 to 2005. He is currently a Full Professor with the School of Software, Kwangwoon University. His research interests include operating systems, embedded systems, and web engineering.