PR-SSD: Maximizing Partial Read Potential by Exploiting Compression and Channel-Level Parallelism

Mincheol Kang, Wonyoung Lee, Jinkwon Kim, and Soontae Kim, *Member, IEEE*

Abstract—Recent NAND flash memories provide a partial read operation that can read a page partially and has lower latency than a normal read operation. In order to maximize the benefit of the partial read operation, compression techniques can be applied to improve performance by generating additional partial page requests via compressing pages into smaller ones. Unfortunately, existing compression-support SSDs suffer from a huge decompression latency that eventually cancels the benefit of the partial read operation. In this paper, we propose Partial Read-aware SSD (PR-SSD) for fully exploiting partial read operations. In order to mitigate the decompression latency, we propose a new compression algorithm, called Dominant Pattern Compression (DPC), which has extremely low decompression latency. Because uncompressed page requests cannot exploit the partial read operation, we propose split Flash Translation Layer (FTL) that can split the requests into smaller ones and allocate them to different channels for exploiting channel-level parallelism in SSD. Experimental results reveal that PR-SSD can reduce the read response time by 18% on average and also the number of writes and write response time by 29% and 24% on average, respectively.

Index Terms—Solid state drive, NAND flash memory, compression, flash translation layer

1 Introduction

NAND flash memory-based Solid State Drives (SSDs) are now considered as fast, silent, low energy-consuming, and shock-resistant storage media. In addition to these original strengths, the recent advanced manufacturing technology has made SSDs overcome their relative disadvantages, such as density, reliability, and cost-per-capacity. Furthermore, manufacturers optimize NAND flash memories for boosting performance by reducing the noise and circuit optimization [1]. As a result, conventional Hard Disk Drives (HDDs) are rapidly being replaced with SSDs to enhance the performance of different kinds of applications from general computer systems to servers.

Although SSDs outperform HDDs, their performance depends on the access patterns of applications. Even though the write latency of NAND flash memories is much longer than the read latency, the effect of different access patterns on write performance is insignificant because the Flash Translation Layer (FTL) remaps different logical addresses to sequential physical addresses for maximizing internal parallelism in SSDs [2]. However, the different access patterns can significantly affect the read performance of SSDs. The performance of the sequential read pattern can be improved by fully exploiting internal parallelisms of SSDs such as multiple channels and chip interleaving [3]. By contrast, the performance of small random read patterns relies on the read latency of NAND flash memories because it is difficult to utilize the internal parallelisms of SSDs completely. Therefore, NAND flash memories must support low read latency to improve the read performance of SSDs.

Fortunately, recent NAND flash memories support *the partial read operation* which enables to read a page partially for providing a lower latency than the normal read operation[1] [1], [4]. Even though the unit of partial read operation depends on the implementation of NAND flash memories, most manufacturers provide a 4 KB partial read for a 16 KB page size. Therefore, the partial read operation can help to improve the performance of OS swap operations or 4 KB random read patterns. Furthermore, the potential for improving performance by exploiting the partial read operation is very promising because prior works show that diverse workloads have many partial page requests [5], [6], [7], [8], [9]. In our experiment, the partial read-enabled SSD can enhance the read performance by up to 22% and 18% on average, if we can ideally exploit partial read operations as shown in Fig. 1.

In order to maximize the potential benefit of the partial read operation, we propose applying a compression technique to SSD. If data is compressed to the smallest unit of the partial read operation, and stored in the NAND flash memory, SSD does not need to perform a normal read

• *The authors are with the School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 34141 Daejeon, Republic of Korea. E-mail: {mincheolkang, wy_lee, coco, kims}@kaist.ac.kr.*

Manuscript received 15 Sept. 2021; revised 25 Apr. 2022; accepted 22 May 2022. Date of publication 27 May 2022; date of current version 10 Feb. 2023. This work was supported in part by the National Research Foundation (NRF) under Grants funded by Korean Government 2022R1A2C200632111 and 2021R1F1A106273011.
(Corresponding author: Mincheol Kang.)
Recommended for acceptance by D. Liu.
Digital Object Identifier no. 10.1109/TC.2022.3178326

1. In the NAND flash memory, the basic unit of a read or write operation is a page. The recent page size is 16 KB for improving bandwidth and capacity.

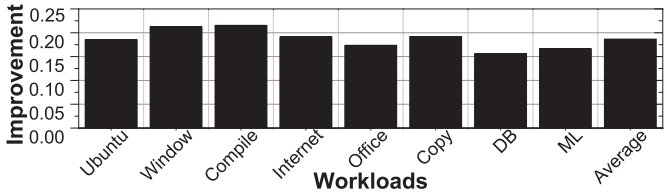


Fig. 1. Read performance improvement of the ideal partial read-enabled SSD.

operation because the required data is partial when a corresponding read request is issued. Therefore, the compression-support SSD can perform more partial read operations than the conventional SSD. In our experiment, the number of partial read operations can be increased by up to 58% and 33% on average after compression. Furthermore, the compression technique can be used to improve the write performance of SSDs by merging several compressed page requests to improve the lifetime of SSDs. Therefore, combining the compression technique and the partial read operation can significantly improve the overall performance of SSDs.

However, in order to realize the compression-support SSD for exploiting partial read operations, we have to overcome two challenges. The first is the decompression latency and the second is the failure of compression.

Previous works, which enabled the SSDs only to compress data, focused on reducing the number of write operations [10], [11], [12]. They overlooked the decompression latency because it was not so significant compared to the read latency of the NAND flash memories. However, as the page size increases, the decompression latency becomes significant in the read performance. Current SSDs use 8 KB or 16 KB page size but previous works assume small page sizes (2 KB or 4 KB) [4]. Since compression algorithms used in storage systems search unique patterns by looking at the entire data sequentially, the decompression latency increases as the data size increase [13]. Furthermore, as the optimization of NAND flash memory leads to a reduced read latency [1], [4], the proportion of the decompression latency in the read response time is further increased. The decompression latency is 13% on average of the read response time in our experiment. As a result, there is no potential benefit in improving the read performance using the partial read operation in the current compression-support SSD. Even worse, the read response time can be increased more than the baseline SSD due to the decompression latency when the compression ratio is low. The next challenge is the failure of compression. Compression can fail depending on the data state [10]. For example, the data in video applications cannot be compressed because they are already compressed. Consequently, there is no way to exploit the partial read operation for uncompressed page requests.

To overcome these limitations, we propose Partial Read-aware SSD (PR-SSD) to fully exploit partial read operations. To the best of our knowledge, this is the first work that combines the partial read operation and compression for improving the performance of SSD. PR-SSD consists of two parts.

The first part is Dominant Pattern Compression(DPC) which has almost zero decompression latency with a reasonable compression ratio, which is defined as *the ratio of reduced size to the original size*. The key insight of DPC comes from memory compression algorithms [14], [15], [16]. Memory compression has very low compression and decompression latency because they perform compression in a cacheline-size (32 or 64 Bytes). However, because these algorithms are designed for CPU cache and DRAM, the compression granularity is not suitable for storage. Thus, we observed data patterns in an SSD and discovered two important characteristics. First, although the number of zero page requests, which have all zero data in the page unit, decreased owing to the large page size, partial zero page requests can be exploited by partitioning the page data into partial read units. Second, we extend the compression granularity from 32 to 1024 Bytes and find dominant patterns in each granularity within a page unit. Inspired by the two aforementioned observations, DPC filters out partial zero data and compresses the rest of non-zero data based on the dominant patterns at each granularity. DPC shows a better compression ratio than memory compression and has extremely low compression and decompression latencies. By using DPC, PR-SSD can increase the potential benefit of partial read operations.

The second part is split Flash Translation Layer (FTL). The key idea of split FTL is to exploit multiple channels that can handle independent requests in parallel in SSD. We split an uncompressed page request into the partial read operation unit and allocate them into different channels. When the corresponding read request is issued, multiple partial read operations are performed in parallel via different channels, thereby PR-SSD can achieve the benefit of the partial read operation even for uncompressed page requests.

Our *main contributions* can be summarized as follows:

- We combine the partial read operation with compression and show how they effectively make a synergy.
- We propose PR-SSD for fully exploiting partial read operations. To solve the decompression latency problem, we propose DPC that has an extremely low decompression latency. Furthermore, in order to fully utilize the partial read operation for uncompressed page requests, we propose split FTL.
- We evaluate PR-SSD using real workloads with actual data. The result shows that our PR-SSD can reduce read response times by 18% on average. In addition, the number of write operations and write response time are reduced by up to 29% and 24% on average, respectively.

2 Background

2.1 Basic SSD Architecture

Fig. 2 shows the basic SSD architecture. The basic I/O unit of NAND flash memory is a page. Read and write operations are performed in a page unit. A block, which is the unit of an erase operation, consists of multiple pages. A plane consists of many physical blocks. Two or four planes make a die, which is an independent unit for executing NAND flash commands. Several dies are built in a single NAND flash memory. Multiple NAND flash memories are connected to a single channel. There are multiple channels for maximizing the bandwidth. This hierarchical structure can provide different level parallelisms from the channel, chip, die, and plane [2].

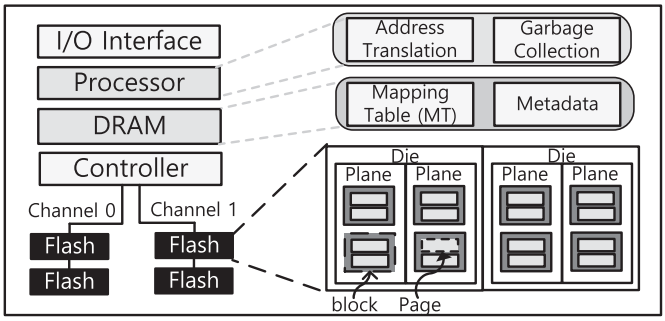


Fig. 2. Basic SSD architecture.

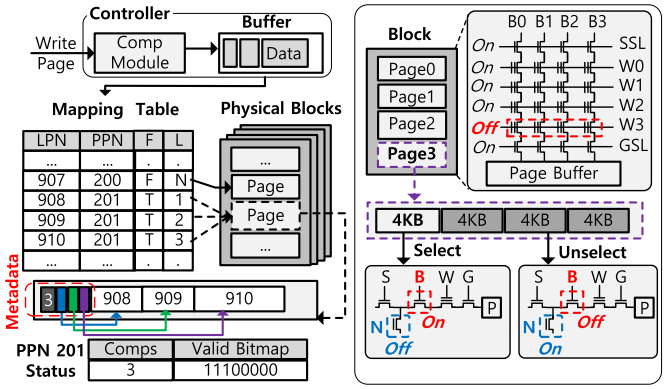


Fig. 3. Compression-support SSD (a) and partial read-enabled NAND flash memory (b).

In SSD, there are several hardware components for handling I/O requests and managing NAND flash memories. The I/O interface is responsible for communicating with the host system through SATA or PCIe interfaces. The processor in SSD executes several important functions such as address translation and garbage collection. Because the NAND flash memory does not allow in-place updates, physical addresses should be changed whenever update requests are issued. Thus, address translation module, i.e., Flash Translation Layer (FTL), which changes the Logical Page Number (LPN) to the Physical Page Number (PPN), exploits a Mapping Table (MT) that is stored in the embedded DRAM in SSD. After update requests are performed, previous pages should be invalidated. These invalid pages are erased by the garbage collection. Before an erase operation is performed in the block unit, garbage collection moves valid pages to a free block and erases a target block, which contains several invalid pages. Metadata, including the validity of pages and the number of invalid pages in each block, is stored in the embedded DRAM in SSD.

2.2 Compression-Support SSD

Fig. 3a shows a compression-support SSD and its mapping table. When host write requests are issued, they are divided into multiple page requests. Hardware-based compression is normally implemented in the controller [13], [17]. When the controller received write page requests, the SSD attempts to compress their data. If the compression succeeds, the data will be stored in an internal buffer, whose size is a page. When the internal buffer is full, the write page requests in the internal buffer are merged and written to the flash memories. In contrast, write page requests that fail the compression are directly written to the flash memories. The compressed pages can be easily decompressed by using the corresponding decompression module. Since multiple compressed pages are stored in a physical page, the mapping table should include additional information [10], [11].

The mapping table of the compression-support SSD includes the compression flag (F) and location (L), which points to the location of the compressed logical page within the physical page. An uncompressed page request with a false compression flag is managed as a normal page request. Therefore, it is not necessary to present the location information, which is NULL (N), such as for example, LPN 907. Conversely, compressed page requests are merged and stored on the same physical page as LPN 908, 909, and 910. The location information indicates the position of each logical page within a physical page. For example, the location of LPN 908 within PPN 201 is the first location. The total number of stored compressed pages and the length of each compressed page, which constitute metadata information, is stored on a physical page. If a read request is issued, the indicated physical page is read, and by comparing the total number of stored compressed pages within the metadata and the location information in the mapping table, the exact start position and length of the compressed data within the metadata can be obtained. Finally, the corresponding compressed data can be decompressed.

In order to provide valid information regarding each compressed page for garbage collection, the physical page status is required for the mapping table. The physical page status includes the number of compressed pages and the valid status of each compressed page. The valid status is represented by a valid bitmap that indicates whether the corresponding compressed page is valid. If an update request to a compressed page is issued, the corresponding bit in the bitmap becomes zero, indicating an invalidated page. Using this information, garbage collection can be performed to identify pages to be copied.

2.3 Partial Read Operation

Fig. 3b shows the internal hardware structure of partial read-enabled NAND flash memory. The physical block is warped between the Source Selection Line (SSL) and Ground Selection Line (GSL). In the block, flash cells are connected to bit-lines (B0, B1, B2, and B3) serially, and each word-line (W0, W1, W2, and W3) indicates a single page. By measuring the current between the source and the ground, the cell state can be identified. This is a read operation in the NAND flash memory [6]. For example, to read page 3, all word-lines except page 3 are turned on as well as the SSL (S) and GSL (G). As all bit-lines are turned on, the current of the cells charges the page buffer (P) depending on the cell state. Therefore, all the cells of the corresponding word-line are read simultaneously [6].

Since the detailed implementations of the partial read operation are different among manufacturers [4] or not publicly disclosed [1], we explain it using the latest NAND flash memory [4]. To enable a partial read operation, two transistors are added, i.e., BLC transistor (Red B) for selecting the

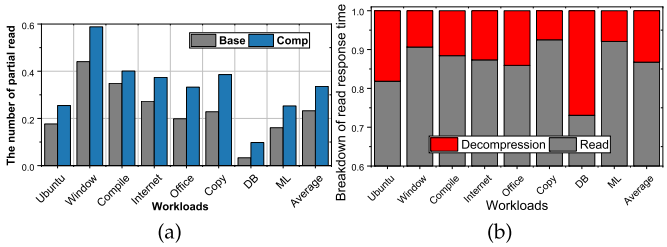


Fig. 4. The normalized number of partial read operations (a) and breakdown of read response time (b) in storage compression.

bit-line and NLO transistor (Blue N) for discharging the bit-line. For a selected partial unit, the BLC transistor is turned on so that the current in the cell can charge the page buffer. In unselected partial units, the BLC transistor is turned off while the NLO transistor is turned on and the current is discharged. Therefore, the current cannot charge the page buffer. As a result, only a selected partial unit will be read. The partial read operation can reduce the latency by 20% compared to the normal read operation and reduce the current consumption by 40% because the number of activated bit-lines and cells are reduced [4].

3 Motivation and Challenges

In this section, we explain the potential of partial read operation combined with compression techniques and discuss what challenges the conventional techniques face by showing their limitations.

3.1 Potential of Partial Read Operation

As the page size continuously grows up, read requests smaller than the page unit become dominant requests in various workloads [5], [6]. A partial read operation, designed for improving the performance of small reads, can improve the read performance by up to 20% in these types of workloads. However, this operation is not applicable to normal read requests whose size is the page unit.

To overcome this challenge, compression techniques can be considered. The compression technique can benefit both normal and small requests whose size is larger than the partial read unit but smaller than the page unit. If data are compressed to the smallest unit of the partial read operation and stored in the NAND flash memory, the SSD can perform the partial read operation to obtain the corresponding page data. Fig. 4a shows that the number of partial read operations can be increased by up to 58% and 33% on average after compression.

3.2 Limitations of Existing Compression

There are two fundamental limitations in conventional compression techniques for realizing partial read potential. The first is decompression latency and the second is incompressible data. In this subsection, we introduce two possible compression solutions for partial read and their limitations.

3.2.1 Storage Compression

Conventional compression-support SSDs use Lempel-Ziv-Welch (LZW) algorithms such as LZ77, LZ78, and LZ4 [17], [18]. These algorithms exploit a dictionary table for storing unique data. They identify the longest redundant data among the entire data and store unique data in the dictionary table. Furthermore, to increase the compression ratio, Huffman coding, which transforms data into binary codes sorted by the value frequency, can be performed after dictionary-based compression (e.g., Zlib, Zstandard, and Zipline) [19], [20], [21]. Even though they show a better compression ratio than dictionary-based compression, it is hard to implement inside SSDs because of hardware cost and complexity. The required hardware area and decompression latency of Zlib are approximately two times higher than those of LZ4 [22]. These algorithms have high decompression latency and are typically implemented outside the SSD and inserted into the PCI Express interface [21]. Therefore, we used the LZ4 algorithm among the LZW algorithms because it has the lowest decompression latency; herein, we refer to this algorithm as storage compression.

However, as the technology of the NAND flash memory is improved, the overhead of decompression latency of storage compression is also increased. The storage compression algorithm searches matching data in the dictionary that should be generated during compression and decompression. If the data size increases, the time for searching and managing the dictionary also increases during compression and decompression. Consequently, the compression and decompression latency also increase [13]. This problem has further deteriorated because manufacturers prefer a larger page size of NAND flash memory for bandwidth and capacity [7]. Furthermore, the latency optimization of the NAND flash memory is another main hurdle. Recent commercial NAND flash memories can achieve low read latency by reducing noise or improving sensing mechanisms [1], [4].

To illustrate this overhead, we performed simulation experiments using various workloads. The details of the workloads, NAND flash memory configuration, and decompression latency are explained in Section 5. Fig. 4b shows the breakdown of the read response time in storage compression. The maximum decompression overhead is 26% and 13% on average. Furthermore, the read latency can be increased to more than the baseline in storage compression if the compression ratio is low. Typically, the partial read operation does not provide any benefit owing to the decompression overhead.

There are two ways to reduce the decompression overhead of the storage compression. The first way is to perform compression at smaller granularity than the page unit and decompress them in parallel. However, it decreases the overall compression ratio. In our experiment, the compression ratio can be decreased by 20% when the granularity of storage compression is 1 KB compared to 16 KB page compression. The second way is to use multiple decompression modules and allocate them to each channel. However, the decompression overhead of this structure is 7% on average, which is still high. Moreover, it is hard to implement multiple decompression modules because of many constraints such as the hardware area, cost, and power [13]. Furthermore, this problem can be more serious when we use a fast NAND flash memory such as Single-Level Cell (SLC) [23] or Multi-Level Cell (MLC) [24]. Thus, the actual overhead of decompression can be larger than demonstrated in this experiment.

3.2.2 Memory Compression

Memory compression can be considered to achieve the potential of partial read operations because its decompression latency is almost zero. As memory compression is performed in the CPU cache or DRAM, several design constraints exist, such as power, hardware area, and latency [25], [26], [27]. These algorithms are simple and performed in a small unit, such as a CPU cacheline (32 or 64 Bytes). Hence, their compression and decompression latencies are extremely low. The popular algorithms in memory compression include Frequent Pattern Compression (FPC) [14], Base-Delta-Immediate compression (BDI) [15], and COMPression using Frequent words (COMF) [16]. Because the data pattern in memory is significantly closer to the application at the code level than that in storage, algorithms leverage value type information such as the integer or pointer value and the relationship between words (2 or 4 Byte) in a CPU cacheline [28], [29]. For example, several zero values exist because of variable initialization, and many integer values can be narrow-width values, which use only lower bits area, and the remaining bit areas are not used.

In order to leverage the memory compression to SSD, a single compression unit can be used. Since the page size is significantly larger than the cacheline size, data should be separated into cacheline size and compressed serially. Even though memory compression is performed several times, the total compression latency is still significantly low because the latency of memory compression is nanosecond scale [15]. Furthermore, multiple units can be used for reducing compression latency because each compression unit can be operated independently in parallel.

However, a major limitation of memory compression is a considerably lower compression ratio than storage compression because the data patterns in storage are significantly more complex than those in memory.

3.2.3 Incompressible Data

Both compression techniques exhibit a fundamental disadvantage, i.e., incompressible data. Compression can fail when the data entropy is high, which implies that the data pattern is complex or random, such as video data [10], [30]. In addition, requests with extremely low compression ratios cannot benefit from compression because they must be stored in the page unit; hence the number of write page operations cannot be reduced. Furthermore, the read response time can be increased because of decompression latency. Consequently, there is no way to exploit the partial read operation for uncompressed page requests.

4 Partial Read-Aware SSD

Conventional compression-support SSDs cannot realize the potential of the partial read operation because of the long decompression latency of storage compression. Although memory compression can be a possible solution, its compression ratio is low. In this section, we introduce Partial Read-aware-SSD (PR-SSD). PR-SSD adopts a new compression algorithm, which is called Dominant Pattern Compression (DPC), which has an extremely low decompression latency and a relatively high compression ratio. Next, we introduce split FTL, which allows uncompressed page requests to use partial read operations. Finally, we will discuss the overhead of PR-SSD.

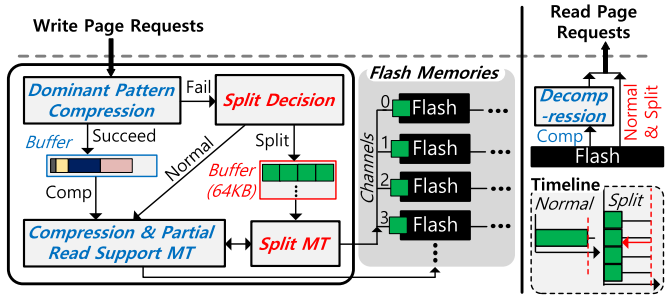


Fig. 5. Overview of partial read-aware SSD.

4.1 Overview

Fig. 5 shows an overview of PR-SSD. When write page requests are issued, PR-SSD attempts to compress the data via DPC. If the compression is successful, the compressed requests are stored and merged into the internal buffer. Since the existing compression-support mapping table cannot fully exploit partial read operations, we modify the mapping table to support both partial read and compression. The compression requests are managed using this mapping table. If the compression fails, PR-SSD attempts to split uncompressed page requests. The requests should be carefully split because this process can affect performance, and thus we add a split decision module. If the uncompressed page requests do not need to be split, they serve as normal page requests. In order to split the uncompressed page requests, PR-SSD buffers four-page requests. Because current NAND flash memories do not allow partial page writes, a single write page request cannot be split. The split requests are managed by an additional mapping table, i.e., the split mapping table. Handling of read requests is similar to that of conventional SSDs except for compressed read requests. They should decompress for restoring original data. Since each split request is served via different channels, the latency of such requests is shorter than that of normal read requests.

4.2 Dominant Pattern Compression

4.2.1 Observations

Data Pattern in the Page. Storage compression algorithms exhibit high compression ratios and high decompression latencies because they attempt to identify redundant patterns of minimum 1 B to n bytes to remove *non-contiguous* redundant data. On the other hand, memory compression algorithms have low decompression latencies because they use a fixed pattern table and compress small granularities for the CPU cache and DRAM to identify contiguous redundant data such as narrow-width values. However, their compression ratios are low because of too small compression granularity. In order to obtain the benefits of both storage and memory compression, PR-SSD must compress larger data than memory compression and find non-continuous data. Hence, we analyze the page data by partitioning

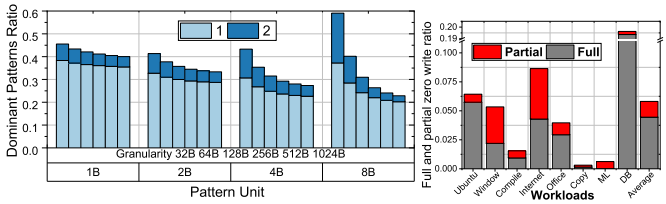


Fig. 6. Dominant patterns ratio at different granularities (32∼1024 Bytes) in the page (a) and partial zero requests ratio (b).

the granularities that are larger than memory compression to find non-continuous data of particular pattern sizes.

Fig. 6a shows the two most dominant patterns ratio (first and second) at different granularities (32∼1024 Bytes) in pages. Pattern sizes vary from 1 B to 2, 4, and 8 bytes. For instance, as shown in the right side of Fig. 7, we divide page data into small granularity and make a pattern table ordered by value (pattern unit) frequency. In this example, data "13" is the first dominant pattern and "FC" is a second dominant pattern. As the pattern size increases, the portions of the dominant patterns decrease because the possibility of the same patterns is reduced. Furthermore, as the granularity increases, the ratios of the dominant patterns also decrease because the probability of diverse patterns increases. The ratio of the most dominant pattern is almost 45% for the 1-byte pattern size.

Partial Zero Requests. When zero page requests, which contain all zero data, they need not be stored to the NAND flash memories; instead, we just mark zero information to the corresponding LPN on the mapping table [31]. However, as the page size increases, the number of zero page requests decreases because the page size is larger than that of conventional OS I/O units, such as sector (512B) or the swap operation (4 KB). Nonetheless, if we divide the page unit into the partial page unit, we have still an opportunity to exploit zero page requests. Fig. 6b shows that the ratio of the partial zero page requests is 6% on average and by up to 20% in various workloads. If the partial zero page requests are filtered out before compression, the overall compression ratio can be further improved because metadata for restoring zero data are not necessary.

4.2.2 Dominant Pattern Compression

Procedure. Fig. 7 shows the overall procedure of DPC, which comprises three steps. First, the page data is divided into partial page units (4 KB) to obtain partial zero data via the partial zero filter. If partial zero data are found, they are represented as partial zero bitmaps in the mapping table. Each bit represents whether the partial data is zero or non-zero. After filtering out partial zeros, only non-zero data are compressed. Second, non-zero data are segmented into small granularity units, and granularity data are compressed independently. Third, the granularity data are counted by each pattern unit to generate a pattern table. For example, for a 1-byte pattern unit, the input granularity data are partitioned into 1-byte sub-data, and each pattern is counted. After finding the most dominant pattern (0 x 13), the pattern in the granularity data is eliminated, and only the remaining (non-compressed) data are copied to a compressed output.

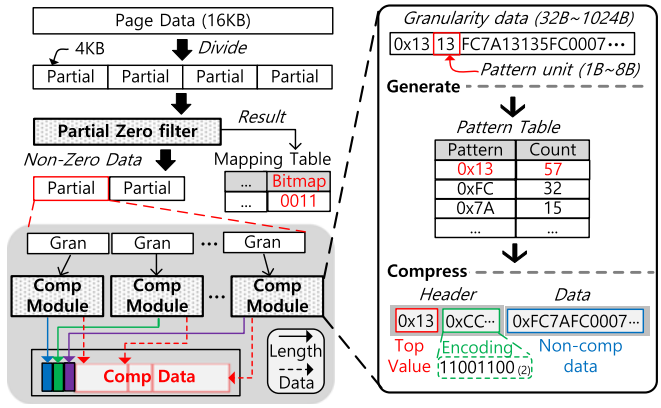


Fig. 7. Dominant pattern compression.

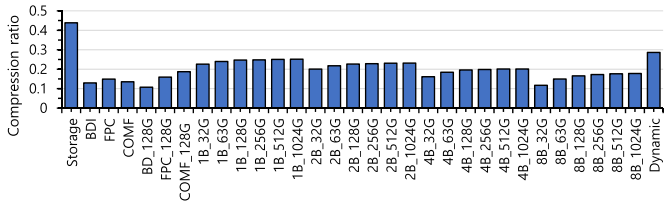


Fig. 8. Compression ratios of different configurations of DPC as well as storage and memory compression. Higher is better.

The compressed output comprises the non-compressed data and a *header* that includes the dominant patterns used as well as the encoding bitmap for identifying whether the location data are compressed. An encoding bit of 1 means that the corresponding location data is compressed, whereas zero means uncompressed data. After each granularity of the data is compressed, the compression ratios of each data can be different. Hence, the compressed length of each compressed data is stored in the final non-zero compressed data.

Implementation. Fig. 8 shows the compression ratios for different configurations of DPC. Although dominant patterns ratios are high at small granularity as shown in Fig. 6a, their compression ratios are lower than larger granularities because of the header information. For example, when the configuration has two dominant patterns and 8 bytes pattern size in 32-byte granularity, it has the header overhead of 16 bytes (8 bytes*2) for two dominant patterns and 1 B (4*2 bits) for encoding that should identify three states (two dominant patterns and uncompressed case; thus 2 bits are required for each pattern size). The total header information is more than 50% of 32-byte granularity. Therefore, the compression ratio is not proportional to the ratio of the dominant pattern. In order to implement DPC in HW, we select only the most dominant pattern, and 1-byte pattern size, and 128-byte compression granularity because this configuration shows the best compression ratio and makes it easy to build the pattern table. This configuration shows 25% compression ratio which is higher than conventional (BDI 12%, FPC 14%, and COMF 13%), and extended (BDI_128 G 10%, FPC_128 G 15%, and COMF_128 G 18%) memory compression algorithms. To measure the latency of DPC, we built the DPC circuit in Verilog HDL and synthesized it using Synopses Design Compiler. The compression latency is 340 ns including building a pattern table and

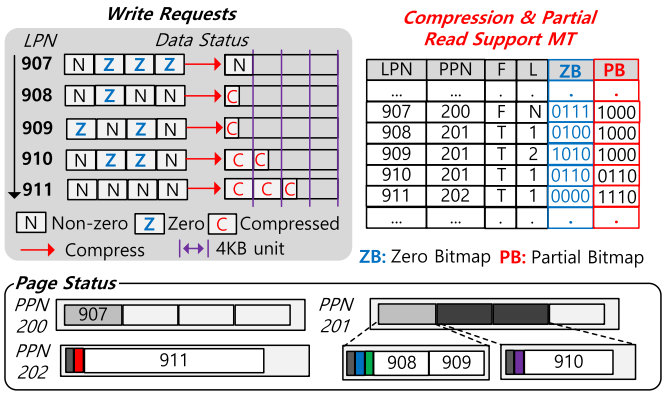


Fig. 9. Example of handling compressed and normal page requests in the PR-SSD and its mapping table.

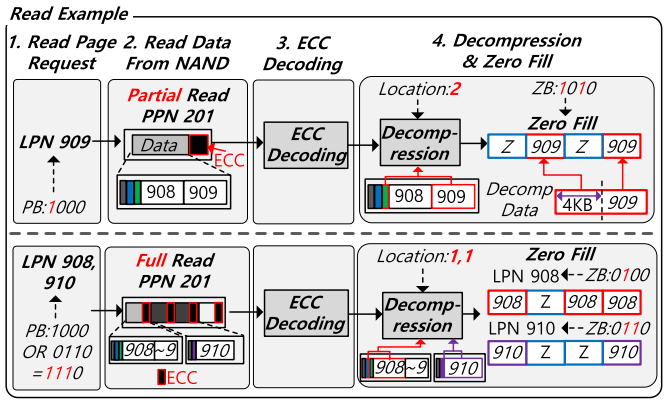


Fig. 10. Read examples in PR-SSD.

decompression latency is 38 ns when the flash controller operates at 400 MHz [13]. Even though this latency is higher than memory compression latency, which is almost 1 ns [15], it is much lower than the storage compression latency. Furthermore, DPC can be parallelized whereas storage compression has to be performed serially in the page unit. DPC decompression latency is negligible compared to the current NAND flash memory latency.

To improve the compression ratio, dynamic DPC, which performs different configurations and selects the best one, can be considered. However, dynamic DPC increases HW cost and the overhead of compression and decompression. Even though DPC requires only low HW resources, multiple HW resources, as well as high compression and decompression latencies, are required to identify the best compression ratio among different configurations in DPC. Furthermore, the compression ratio of the dynamic DPC is improved by only 2∼3% compared with the static DPC used in our experiment. Hence, we choose static DPC in this paper.

4.2.3 Mapping Table for Partial-Read With Compression

Fig. 9 shows the process to manage compressed and normal page requests in PR-SSD. To leverage the partial read operation in the conventional compression-support mapping table, we add zero and partial bitmap information to the mapping table. The zero bitmap indicates the location of the partial zero data in incoming requests. For example, the write page request of LPN 907 contains three partial zero data and is represented as 0111. The partial bitmap informs the location of the partial page index and its size on the physical page. For instance, LPN 908 and 909 are compressed and stored in the first partial location PPN 201. Although LPN 907 contains three partial zero data, the non-zero data in LPN 907 cannot be compressed. Hence, the compression flag of LPN 907 is false. All compressed page requests are stored temporarily in the buffer for merging several page requests. However, if the remaining size of the buffer is smaller than the incoming page request, the current buffer is flushed, and the incoming page request will be stored in the buffer. For example, the current buffer stores LPN 908, 909, and 910 and the incoming page request is LPN 911. Because the compressed size of LPN 911 is larger than the current remaining buffer size, the current buffer flushes LPN 908, 909, and 910; subsequently, they are stored in PPN 201. LPN 910 will be stored in the buffer after flushing; eventually, it will be stored in PPN 202.

Fig. 10 shows the method to read compressed data in PR-SSD. When read page requests are issued, PR-SSD lookups the partial bitmap of the corresponding LPN and performs NAND read operations. If the partial bitmap of the incoming read page request includes a single bit, a partial read operation can be applied. For example, since the partial bitmap of LPN 909 is 1,000, it can be served by a partial read operation. ECC decoding is required to check errors after reading data from NAND flash memories, Since each partial page unit within a physical page has a corresponding ECC area [6], PR-SSD does not need to read the entire page for ECC decoding. After decoding ECC, the location information of LPN 909 allows the exact position of the merged data, and which can be decompressed, to be obtained. The original data of LPN 909 will be restored using a zero bitmap after decompressing. Since the different compressed LPNs can be merged into the same PPN, they can be read simultaneously. For example, when the read page requests of LPN 908 and 910 are issued simultaneously, PR-SSD can serve that data via a single page read. If the partial bitmap of incoming read page request includes multiple 1 s, the request is served by a full-page read operation rather than multiple partial read operations. Since the result of the OR operation of two partial bitmaps (LPN 908 and 910) is 1110, PPN 201 will be read fully. After ECC decoding, the location and zero bitmap of each LPN allow the original data to be restored.

The procedure of garbage collection is similar to that of the existing compression-support SSD. The PR-SSD stores multiple compressed pages in a physical location; and uses valid bitmap information. If invalidation occurs, PR-SSD invalidates the corresponding page using the valid bitmap. If the valid bitmap indicates all zero, it is implied that the entire page is invalidated. If a single valid bit exists in the merged physical page, it should be copied to the free block.

4.3 Split Flash Translation Layer

Fig. 11 shows the process to split write requests. When the compression fails, PR-SSD attempts to split write requests to enable a partial read operation. However, splitting all uncompressed requests can affect the performance

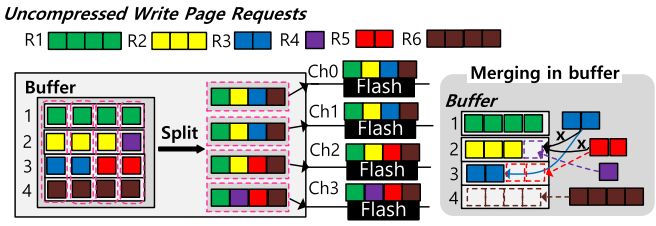


Fig. 11. Split uncompressed requests and merging mechanism in the buffer.

depending on the read access pattern. We will discuss this in more detail in the next subsection. After the write requests undergo the split decision, PR-SSD stores them in a buffer of size 64 KB (four pages). The reason why PR-SSD exploits this internal buffer is that current NAND flash memories do not allow for partial data writing. If we use a single page buffer as the conventional compression-support SSD, a write request must be stored partially in different physical NAND flash locations, and the remaining space will not be utilized.

In this buffer, partial write requests can be merged as shown in upper and right Fig. 11. The requests (R1, R2, R3, R4, R5, and R6) are stored in the internal buffers in order. Thus, requests 1 and 2 are stored in the first and second buffer respectively. Request 3 is stored in the third buffer because it cannot be inserted into the second buffer because of its size. If the size of an incoming request is smaller than the remaining size in the buffer, the request can be merged in the buffer. For example, as request 4 cannot be stored in the second buffer, it is inserted into the third buffer, and requests 3 and 4 can be merged. Similarly, requests 2 and 5 are merged in the second buffer.

When the internal buffer is full, split FTL reconstructs the requests by creating multiple sets of vertical packages of the partial page unit (4 KB) in the buffers; subsequently, each set is allocated to different channels. Thus, when a read request is issued, four partial read operations are performed instead of one normal read operation. The read latency of NAND flash memory varies depending on the page type [32]. In TLC (Triple-Level-Cell), there are three page types: LSB (Least-Significant-Bit), CSB (Center-Significant-Bit), and MSB (Most-Significant-Bit). The latency differences among these types can be two or three times. Even though read requests are performed in multiple partial read operations, no latency improvement is afforded when the page types are different because the latency of read requests will be decided by the longest latency. Therefore, we split the requests when the page types of the four partial read requests are the same. The latency of splitting uncompressed page requests is insignificant compared with the write latency of the NAND flash.

4.3.1 Split Decision

Uncompressed page requests can be split through multiple partial read operations via different channels, and its latency is shorter than that of the normal read page request. However, this does not always guarantee performance improvement. Fig. 12 shows this circumstance. Assume that an SSD has four channels and each channel has four chips. If we split all the uncompressed page requests, each chip

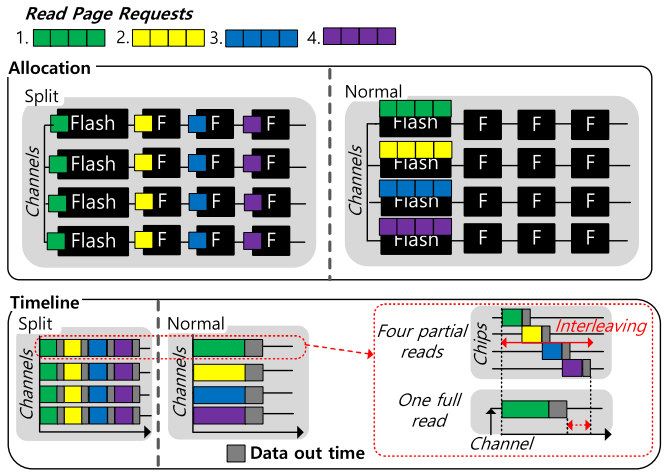


Fig. 12. Split conflict.

stores a single partial page request. However, if we do not split uncompressed page requests, one of the chips in a single channel stores a full-page request. When a bulk read request is issued, PR-SSD has to read multiple split partial pages among the different chips in each channel. Even though these split partial requests can exploit chip-level parallelism, which can overlap the data out time, their latency can be longer than the latency of the full-page reads.

To prevent this circumstance, we add a split decision module before splitting requests in PR-SSD. This module considers host read and write request sizes. If a bulk host write request is issued, the split decision compares the number of partial pages of the host write request when splitting with the number of channels that can allocate the maximum parallelism in the current SSD. For example, since the maximum number of partial read operations that can be parallel is four, as shown in Fig. 12, the host size that can be allowed to be split is a single page unit. If the host request size is larger than the size of a page unit, PR-SSD does not split the requests. Therefore, the number of channels in the SSD can affect PR-SSD; a detailed analysis will be described in Section 4.4.2. This module considers the host read size of each LPN except for the first host write requests that do not contain the history of the host read sizes. The host read size information can be managed using two approaches. The first approach is to mark this information as a single bit in the mapping table as the exact host read size is not necessary, that is only a threshold is required. The second approach is to store the information partially and allow it to be managed by an LRU. Both implementations involve a small overhead and their performance results are similar.

4.3.2 Split Mapping Table

Fig. 13 shows a split mapping table. We used a two-level mapping table to reduce the total mapping table size. We add a split flag to the mapping table. If the split flag is true, the corresponding LPN is split into four partial units and the corresponding mapping information is stored in the split mapping table. Each logical partial index indicates the different PPNs, which are mapped to different channels. In addition, the partial index of each physical page is added to the mapping table. For example, LPN 915 is split and its first

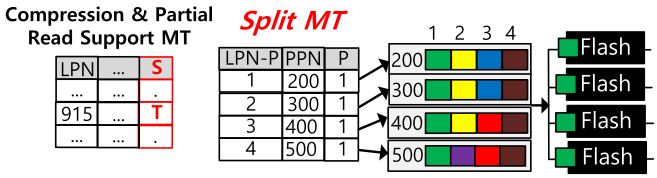


Fig. 13. Split mapping table.

partial index directs to PPN 200, with a partial index is 1. Since the split mapping table can identify the partial index of each physical page, multiple partial read operations can be performed simultaneously. If multiple partial reads indicate the same physical address, PR-SSD performs a single full-page read instead of multiple partial reads, to avoid split conflicts. Also, the ECC process is similar to handling the compressed page requests. If an update request is issued, different physical pages are invalidated partially. To do so, PR-SSD exploits the existing valid bitmap. Thus, the garbage collection or wear-leveling process is the same as the process of managing compressed requests. If a valid page includes split data, PR-SSD copies the corresponding split data. The other split data, which are stored in different channels, will be maintained. Hence, PR-SSD does not need to perform multiple partial read operations. As a result, PR-SSD does not affect the garbage collection or wear-leveling performance.

4.4 Discussion

4.4.1 Mapping Table Size

In the conventional compression-support SSDs, multiple write requests are merged into a single page. Theoretically, the maximum number of merged requests is over a few hundred, but the current implementation limits it to 20 because of the growing size of the mapping table when we use a 16 KB page size [10]. Furthermore, a partially valid status of the merged requests is also required. Since garbage collection has to find a mapped LPN using the PPN, the reversed mapping information should be preserved. This reversed information is proportional to the maximum number of merged requests. In order to store metadata and ECC, each physical page has a special area, called Out-Of-Bound (OOB), which has the 640-byte size for an 8 KB page [33]. The basic mapping table uses this area for storing metadata [11]. Even though OOB size is sufficient to store the metadata of the compression-support SSD, the size of the mapping table of the compression-support SSD can be overhead because OOB has to store ECC and other metadata [11].

By contrast, since PR-SSD applies DPC, which has a lower compression ratio than the storage compression, the maximum number of merged requests is set to 12. Therefore, the size of reversed mapping information decreases. In addition, the partially valid status of the merged requests decreases. Even if the mapping table of PR-SSD includes a partial index, partial zero bitmap, and split flag, the mapping table of PR-SSD is smaller than the mapping table of the conventional compression-support SSD because additional information can be represented in only a few bits. Consequently, the mapping table size of PR-SSD can be reduced by 37% as compared with that of conventional compression-support SSD.

However, PR-SSD has to consider the size of the split mapping table. This mapping table is generated when uncompressed requests are split. In the worst case, all LPNs can have this mapping table. However, we can exploit the temporal locality of read access to manage the number of split mapping tables. Assuming that we have the hotness information of each LPN, whenever the corresponding LPN tries to split, we can decide whether this request should be split or not. If the corresponding LPN is a cold page, PR-SSD does not perform split and writes it normally. Furthermore, PR-SSD can update the split mapping table during garbage collection and read refresh for retention of NAND flash memory [32]. Two-level mapping is not a new idea and has been used in different kinds of SSDs [34]. Similar to previous works, we can utilize a cached mapping table such as DFTL [35] to reduce the overhead of the mapping table. Furthermore, the size of the split mapping table is sufficient to $2 \sim 6$MB in our experiment.

4.4.2 Channel Constraint

Since the split FTL exploits channel-level parallelism, the number of channels can affect the performance of PR-SSD. Recent SSDs use more than four channels for improving bandwidth [3], [23], [24]. Thus, PR-SSD uses a minimum of four channels. In addition, we evaluate PR-SSD with different channel configurations and these results are shown in Section 5.

4.4.3 Energy Consumption

The number of total read operations can be increased because our PR-SSD generates more partial page operations than normal read operations As a result, the energy consumption of read requests can be increased. However, since the partial read operation can reduce power consumption by reducing the number of activated bit-lines and cells by approximately 40% [4], the increased energy consumption is insignificant. Furthermore, our PR-SSD can reduce the total number of read operations because the number of garbage collections is reduced. In other words, the number of valid pages to be copied is reduced. As a result, total read energy consumption can be reduced. In Section 5, we will show the number of read operations and the energy consumption results.

5 Evaluation

5.1 Experiment Setup

5.1.1 SSD Configuration

We evaluated PR-SSD using simpleSSD, which simulates every hardware component inside an SSD, such as the microprocessor, DRAM, and parallelism structure of NAND flash memories [3]. Therefore, our experiment involved the DRAM access time and firmware execution time. Furthermore, since the bus transfer time of channels and the ECC decoding time are critical paths in the read latency, we considered them by referring to real commercial NAND flash memory [36]. We used TLC NAND flash memory [4]; the SSD configurations are shown in Table 1. The partial read operation is performed in a 4 KB unit, which is a quarter of

TABLE 1
SSD Configuration

SSD configuration	
Partial read size	4 KB
Partial read latency reduction	20%
Page size	16 KB
Blocks, pages	1024,512
Dies,planes	2,2
Channels,chips	8,2
Read latency	$56 \sim 180$ us
Program latency	$700 \sim 1400$ us
ECC Decoding	20 us
Bus Transfer	833 MB/s

TABLE 2
Workload Characteristics

Workloads	Read ratio	Read size (KB)	Write size (KB)	Read random ratio	Write random ratio	Compression ratio
Ubuntu	0.24	42.83	179.47	75%	49%	54%
Window	0.42	14.19	56.93	36%	55%	45%
Compile	0.69	25.45	262.87	42%	61%	54 %
Internet	0.24	28.08	42.65	72%	78%	34%
Office	0.21	35.35	55.27	70%	60%	43%
Copy	0.60	32.32	115.24	63%	14%	9%
DB	0.10	114.60	432.75	26%	64%	63%
ML	0.70	36.83	18.60	32%	62%	56%

the page size, and its latency reduction is 20% as compared with that of normal read operation [4]. We used the LZ4 algorithm as the storage compression algorithm as it affords the best decompression latency [37]. Even though the compression and decompression performances can differ depending on the implementation [17], [38], we assume the best compression and decompression performance of 11Gbit/s [38] for a conservative comparison. Therefore, the latency of storage compression(decompression) is 12us and memory compression(decompression) is 1 ns [15].

5.1.2 Workloads

The available block I/O workloads include the address, size, operation, and timing of each request; however, they do not contain real data or they contain only have hashed data for privacy reasons [39], [40], [41]. These public I/O workloads cannot be used for compressing real data. Another option is to use synthesis workloads that can generate various I/O patterns and configure the compression ratio of data such as Oracle VdBench [42] and OLTP (On-Line Transaction Processing) benchmark [43]. However, these workloads randomly fill the data to satisfy the configured compression ratio. Therefore, we created real I/O workloads that included both access patterns and real data using QEMU-KVM. We modified QEMU (Quick EMUlator) source files to generate I/O requests using real data and verified the access patterns using blktrace, which is the Linux tool for capturing I/O traces. Therefore, we created eight workloads with different scenarios. Ubuntu workload is to install ubuntu 16.04. Windows workload is to install Windows 7. Compile workload is to compile Linux kernel 4.9.13 on Ubuntu. Internet workload is web browsing in six hours on Windows. Office workload is to perform office applications such as Microsoft Word, Excel, and PowerPoint in six hours on Windows. Copy workload is to copy a movie file on Windows. DB workload is a database application that creates several tables and imports IMDb (The Internet Movie Database) data set and executes SQL queries on MySql. ML workload is a machine learning application that classifies real images from ImageNet by using Darknet [44]. The important I/O characteristics including the storage compression ratio are shown in Table 2.

5.1.3 Schemes

We evaluated PR-SSD compared to different compression algorithms by enabling partial read operation or not. We implement six schemes and detailed descriptions are shown in Table 3. The others except for S and M schemes are introduced in this paper. The memory compression algorithm we used is FPC because it shows a better compression ratio than BDI and COMF. If the compression ratio is extremely low, it cannot obtain the benefit of compression because compressed size is similar to the original. DPC considers that compression is failed when the compression ratio is lower than 10%.

TABLE 3
Detail Description of Evaluated Schemes

Schemes	Description
S	Compression-support SSD that uses storage algorithm
M	Compression-support SSD that uses memory compression
D	Compression-support SSD that uses DPC
P	Partial read-enabled SSD without compression
SP	Storage compression with partial read-enabled SSD
MP	Memory compression with partial read-enabled SSD
DP	DPC with partial read-enable SSD without split FTL
PR	Proposed PR-SSD including DPC and split FTL

5.2 Performance Results

5.2.1 Read

Fig. 14 shows the read response times for different schemes that are normalized to the baseline (no partial read and no compression). The storage compression (S) shows that the read response time is increased by up to 36% and 11% on average. This is because of the decompression latency. Although storage compression enables partial read operations (SP), it does not reduce the latency. The read response time is increased by up to 33% and 5% on average in SP scheme. Since memory compression (M) has extremely low decompression latency, it does not increase the read response time. Since several read requests can be merged into a single page, different logical addresses can indicate the same physical address. As a result, the read operations can be reduced if the physical addresses are the same. Therefore, memory compression can reduce the read response time compared to the baseline. However, this improvement is insignificant as compared with enabling a partial read operation. Depending on the number of partial

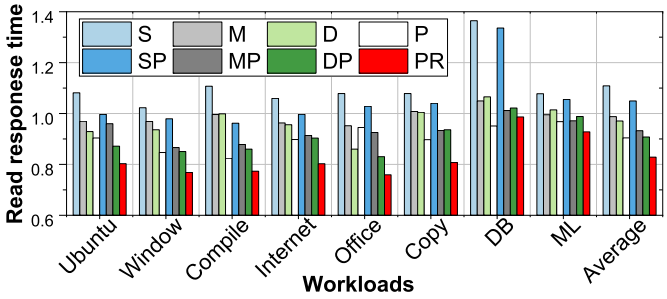


Fig. 14. Normalized read response time.

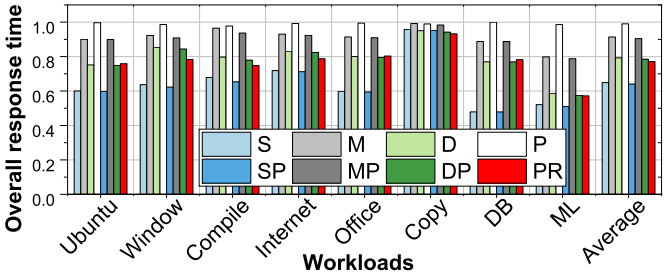


Fig. 16. Normalized overall performance.

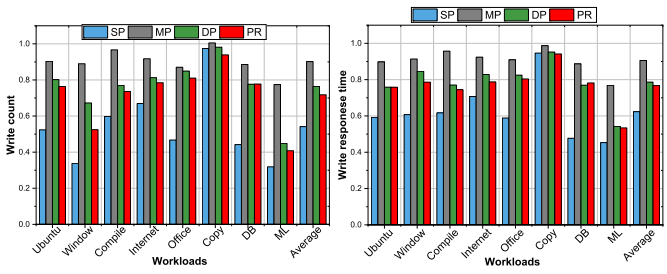


Fig. 15. Normalized numbers of write operations (a) and normalized write response time (b).

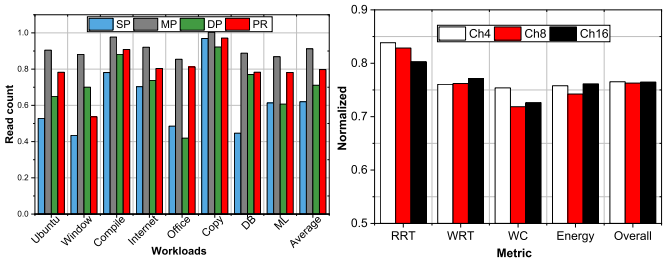


Fig. 17. The normalized number of read operations (a) and different channel configurations results (b).

read requests, the partial read-enabled SSD (P) can reduce the read response time. More improvements can be achieved when it is combined with memory compression (MP). However, the improvement in the read performance is insignificant because of the low compression ratio of memory compression. DPC (DP) outperforms the memory compression due to higher compression ratio. Furthermore, since it stores partial zero data information in the mapping table, it does not need to serve read requests that have all zero data. As a result, the read response time in the office workload can be further reduced. However, this is a special case, and it cannot fully utilize partial read operations. Meanwhile, PR-SSD (PR) can reduce the response time more significantly than memory compression by splitting uncompressed page requests. Since PR-SSD can manage uncompressed requests by splitting read requests, it is more effective when the compression ratio is low such as the copy workload. PR-SSD can reduce the read response time by up to 25% and 18% on average.

5.2.2 Write

Fig. 15 shows the normalized number of write operations and normalized write response time. Since the partial read operation does not affect the number of write operations or write response time, we show only the results of two different compression schemes (SP and MP) and our schemes (DP and PR). Compression can reduce the number of write operations by merging several write requests; therefore, the average write response time is also reduced. In other words, the compression requires fewer blocks, which means that the number of garbage collections is reduced and hence, the number of erase operations is reduced. Since storage compression has a higher compression ratio than memory compression and DPC, it can significantly reduce the number of write operations and the write response time. However, when the compression ratio is low, the number of write operations cannot be reduced as shown in the copy workload. Meanwhile, because PR-SSD can merge several uncompressed requests in the internal buffer, it can slightly reduce the number of write operations even in the copy workload. Although PR-SSD does not achieve the storage compression performance in terms of the write counts and write response time, the performance gap between the two schemes (SP and PR) is 17% for the number of write operations and 13% for the write response time on average. PR-SSD can reduce the number of write operations by up to 60% and 29%, and the write response time by up to 47% and 24% on average.

5.2.3 Overall

Fig. 16 shows the overall response time. The trend of the overall performance is similar to the write response time. This is because the write latency of NAND flash is much larger than the read latency of that and the write performance includes the latency of the garbage collection. The partial read operation can slightly reduce the overall response time. However, in the case of a read-intensive workload such as ML, the partial read operation can help to improve the overall performance. Thus, we believe that the overall performance can be significantly improved for read-intensive workloads. PR-SSD can reduce the overall response time by up to 43% and 23% on average.

5.3 Detailed Results

5.3.1 Read Operations

Fig. 17a shows the normalized number of read operations. Storage and memory compression can reduce the number of read operations depending on the compression ratio. This is because several read operations can indicate the

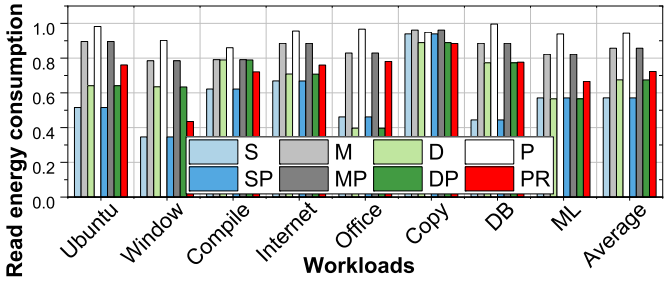


Fig. 18. Read energy consumption.

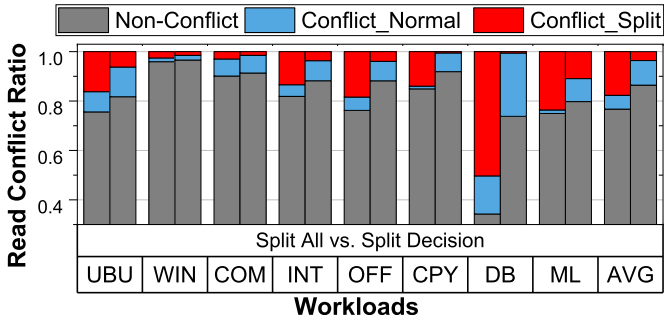


Fig. 19. Breakdown read operations conflict ratio among channels. UBU to ML indicate each workloads we used. AVG means average.

same page owing to merged compressed pages, and then they are read simultaneously. Furthermore, since the number of garbage collections is reduced due to merging compressed pages, the number of valid pages to be copied is reduced. As a result, the high compression ratio can reduce the number of read operations. However, when the compression ratio is low, this advantage is not observed such as the copy workload. The advantage of merging compressed pages is reflected in PR-SSD as well. Even though PR-SSD performs multiple partial read operations instead of normal read operations when the requests are split, the number of read operations can be reduced. Since PR-SSD carefully splits uncompressed page requests by considering the host request sizes, the increase in the number of read operations is insignificant. If the number of total read operations can be increased by splitting requests, the increased read operations will be primarily partial read operations. Therefore, the overhead of increased read operations, such as the energy consumption, can be mitigated because the partial read operation consumes less energy than the normal read operation.

5.3.2 Read Energy Consumption

Fig. 18 shows the normalized read energy consumption. Although the partial read operation reduced read energy consumption by 40%, the partial read enabled-SSD (P) without compression reduced the read energy consumption by 6% on average. This is because valid pages in the garbage collection should be read as whole pages because the valid pages are not partially invalided. Energy consumption can be further reduced if a compression technique is applied such as storage, memory compression, or DPC (SP, MP, or DP). This is because the compression can reduce the number of read operations. PR-SSD can reduce energy consumption, as compared with the baseline, because of this advantage. Even though PR-SSD increases the number of read operations because of splitting requests, the increased read operations are partial read operations, which have lower energy consumption. Hence, the read energy consumption of PR-SSD can be reduced. If the number of read operations is higher than the baseline, total read energy consumption is much lower than the write energy consumption [45]; therefore, the read energy overhead is insignificant. PR-SSD can reduce the read energy consumption by up to 57% and 26% on average.

5.3.3 Split Conflict

Fig. 19 shows the conflict ratio among multiple channels when read page requests are issued. The Split All scheme performs to split all uncompressed page requests. The Split Decision is PR-SSD. Non-Conflict means that all read page requests utilize all different channel resources without chip interleaving. Conflict Normal means that some of the non-split page requests should be performed by chip interleaving. In other words, at least two or more read page requests access the same channel address. Conflict Split means that some of the split page requests should be performed by chip interleaving. The Split All scheme transforms uncompressed page requests into multiple split page requests without considering the host read access pattern and its size. As a result, this scheme can increase the conflict ratio because of splitting page requests as compared with PR-SSD. Even if the Split All scheme increases the conflict ratio, the read performance does not reduce significantly because most read page requests are non-conflict page requests. However, it can increase the read response time by up to 10% in DB trace. This is because DB trace reads bulk data; as such the split-read page requests interfere with each other. Since PR-SSD attempts to split only small-sized requests, it almost eliminates conflict split requests in DB traces, thereby PR-SSD can reduce the overhead of split conflicts. Although the read performance improvement of DB workload is smaller than other workloads owing to large read/write host request sizes and low read random access ratio, PR-SSD outperforms previous works (S or M). PR-SSD can reduce the conflict ratio of split requests by 3% on average.

5.4 Sensitivity Result

Since PR-SSD exploits multiple channels to reduce the read latency, it can be affected by the number of channels. Fig. 17b shows the average results of changing the number of channels from 4 to 16. RTT is the read response time, WRT is the write response time, and WC is the number of write operations. Even if PR-SSD uses four channels, which is the minimum number of channels for enabling split FTL, the read response time can be reduced. This is because PR-SSD handles multiple read page operations simultaneously when the corresponding pages are merged. PR-SSD can reduce the read response time by 17% on average in a four-channel SSD. Since recent SSDs prefer to increase channels for improving bandwidth, PR-SSD can be implemented on current SSDs. The number of write operations and the write response time can differ depending on the number of channels because of garbage collection. However, this difference is insignificant. Although the number of channels is

increased to 16, the read response time is reduced slightly as compared with that of eight channels. This is because the maximum number of requests to be split is increased. As a result, the read response time is reduced, including for the 16-channel SSD.

6 Related Work

6.1 Compression in SSD

Many studies that exploit compression have been proposed for the improving lifetime of NAND flash memory [10], [11], [12]. They can achieve data reduction using storage compression algorithms. The compression technique can be also used for a different purpose, such as recovering data from a power failure [11]. Li *et al.* [12] proposed a selective compression scheme for improving read performance by reducing the amount of data that LDPC needs to check an error. However, since previous works use storage compression which has a decompression overhead or heavier compression algorithm that is extremely slow such as zlib or gzip [37], the read performance is inevitably decreased compared to the baseline or it is difficult to achieve the desired purpose what they want.

6.2 Partial Read

Even though the industry has already proposed the partial read operation a few years ago, since the detailed implementations are not open for the public, few studies enable the partial read operation by modifying the HW in NAND flash memories [5], [6]. Park *et al.* [5] first proposed partial read enabled NAND flash memory. They modified the HW in the NAND flash memory and added a new command for realizing partial read operation. Furthermore, it can provide the different granularities of the partial read operation from 2 KB to 14 KB with 16 KB page size. However, there are too many assumptions related to the granularity of partial read operation and its latency reduction. Liu *et al.* [6] proposed the partial read operation in 3D NAND flash memory. It performs multiple partial read operations that have different page addresses simultaneously in the same chip to read a page. However, the partial read operation that they built cannot reduce the latency because it activates all bit lines for reading the entire page; actually, it can increase the read latency slightly. Furthermore, it can increase performance when workloads have many small-size read requests.

7 Conclusion

In this paper, we propose PR-SSD that provides DPC for compressing pages with the minimum decompression overhead and split FTL that allows uncompressed pages to exploit partial read operations by using channel-level parallelism. PR-SSD further improves performance by supporting partial zero and compaction of compressed pages. Experimental results show that our PR-SSD can reduce read response time by 18% on average as well as the number of writes and write response time 29% and 24% on average, respectively. Reduced write operations help improve the endurance of SSD.

References

[1] C. Kim *et al.*, "A 512-gb 3-b/cell 64-stacked WL 3-D-NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 124–133, Jan. 2018.

[2] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Architecture*, 2014, pp. 524–535.

[3] D. Gouk *et al.*, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *Proc. IEEE/ACM 51st Annu. Int. Symp. Microarchitecture*, 2018, pp. 469–481.

[4] C. Siau *et al.*, "13.5 A 512gb 3-bit/cell 3D flash memory on 128-wordline-layer with 132MB/s write performance featuring circuit-under-array technology," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 218–220.

[5] J. Park, M. Kim, S. Lee, and J. Kim, "Improving I/O performance of large-page flash storage systems using subpage-parallel reads," in *Proc. IEEE 7th Non-Volatile Memory Syst. Appl. Symp.*, 2018, pp. 25–30.

[6] C.-Y. Liu *et al.*, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 955–969.

[7] M. Kang, W. Lee, and S. Kim, "Subpage-aware solid state drive for improving lifetime and performance," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1492–1505, Oct. 2018.

[8] I. Fareed, M. Kang, W. Lee, and S. Kim, "Update frequency-directed subpage management for mitigating garbage collection and dram overheads," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2467–2480, Dec. 2021.

[9] I. Fareed *et al.*, "Leveraging intra-page update diversity for mitigating write amplification in SSDs," in *Proc. 34th ACM Int. Conf. Supercomputing*, 2020, pp. 1–12.

[10] Y. Park and J.-S. Kim, "zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices," *IEEE Trans. Consum. Electron.*, vol. 57, no. 3, pp. 1148–1156, Aug. 2011.

[11] D. Kim, Y. Won, J. Cha, S. Yoon, J. Choi, and S. Kang, "Exploiting compression-induced internal fragmentation for power-off recovery in SSD," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1720–1733, Jun. 2016.

[12] Q. Li, L. Shi, R. Pan, C. Ji, X. Li, and C. J. Xue, "Selective compression scheme for read performance improvement on flash devices," in *Proc. IEEE 36th Int. Conf. Comput. Des.*, 2018, pp. 43–50.

[13] S. Lee, J. Park, K. Fleming, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Trans. Consum. Electron.*, vol. 57, no. 4, pp. 1732–1739, Nov. 2011.

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

[15] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Architectures Compilation Techn.*, 2012, pp. 377–388.

[16] A. Nath and H. K. Kapoor, "WELCOMF: Wear leveling assisted compression using frequent words in non-volatile main memories," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2020, pp. 157–162.

[17] W. Liu, F. Mei, C. Wang, M. O'Neill, and E. E. Swartzlander, "Data compression device based on modified LZ4 algorithm," *IEEE Trans. Consum. Electron.*, vol. 64, no. 1, pp. 110–117, Feb. 2018.

[18] J. Li, K. Zhao, X. Zhang, J. Ma, M. Zhao, and T. Zhang, "How much can data compressibility help to improve NAND flash memory lifetime?," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 227–240.

[19] J.L. Gailly and M. Adler, "Zlib," 1995. [Online]. Available: https://zlib.net/

[20] Facebook, "Zstandard," 2016. [Online]. Available: https://facebook.github.io/zstd/

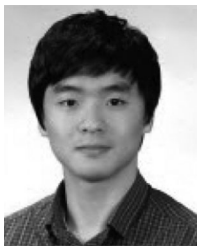
[21] C. Eric, D. Chiou, and S. Carrie, "Acceleration at Microsoft," *Hotchips*, 2019.

[22] Xilinx, "Vitis data compression library," 2020. [Online]. Available: https://xilinx.github.io/Vitis_Libraries/data_compression/2020.1/source/results.html

[23] W. Cheong *et al.*, "A flash memory controller for 15 μs ultra-low-latency SSD using high-speed 3D NAND flash with 3 μs read time," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 338–340.

[24] Intel, "Intel SSD 750 specification," 2015. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-750-spec.pdf
[25] S. Kim, J. Kim, J. Lee, and S. Hong, "Residue cache: A low-energy low-area L2 cache architecture via compression and partial hits," in *Proc. IEEE/ACM 44th Annu. Int. Symp. Microarchitecture*, 2011, pp. 420–429.
[26] J. Hong, H. Kim, and S. Kim, "EAR: ECC-aided refresh reduction through 2-D zero compression," in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn.*, 2018, pp. 1–11.
[27] J. Kim, S. Hong, J. Hong, and S. Kim, "CID: Co-architecting instruction cache and decompression system for embedded systems," *IEEE Trans. Comput.*, vol. 70, no. 7, pp. 1132–1145, Jul. 2021.
[28] J. Kim, M. Kang, J. Hong, and S. Kim, "Exploiting inter-block entropy to enhance the compressibility of blocks with diverse data," in *Proc. IEEE 28th Int. Symp. High Perform. Comput. Architecture*, 2022, pp. 1100–1114.
[29] M. Jang, J. Kim, J. Kim, and S. Kim, "ENCORE compression: Exploiting narrow-width values for quantized deep neural networks," in *Proc. Des., Autom., Test Europe Conf.*, 2022, pp. 1503–1508.
[30] J. Park, Y. Jung, J. Won, M. Kang, S. Lee, and J. Kim, "RansomBlocker: A low-overhead ransomware-proof SSD," in *Proc. IEEE/ACM 56th Des. Automat. Conf.*, 2019, pp. 1–6.
[31] D. Kim and S. Kang, "zf-FTL: A zero-free flash translation layer," in *Proc. 31st Annu. ACM Symp. Appl. Comput.*, 2016, pp. 1893–1896.
[32] W. Choi, M. Jung, and M. Kandemir, "Invalid data-aware coding to enhance the read performance of high-density flash memories," in *Proc. IEEE/ACM 51st Annu. Int. Symp. Microarchitecture*, 2018, pp. 482–493.
[33] Samsung, "K9gbg08u0a 32gb a-die NAND flash multi-level-cell (2 bit/cell)," 2009. [Online]. Available: https://datasheetspdf.com/pdf-file/704185/Samsung/K9GBG08U0A/1
[34] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 77–90.
[35] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, 2009.
[36] Micron, "Product flyer: Micron 3D NAND flash memory," 2016. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/product-flyer/3d_nand_flyer.pdf?la=en
[37] LZ4, "LZ4 - Extremely fast compression," 2011. [Online]. Available: https://github.com/lz4/lz4
[38] J. Kim and J. Cho, "Hardware-accelerated fast lossless compression based on LZ4 algorithm," in *Proc. 3rd Int. Conf. Digit. Signal Process.*, 2019, pp. 65–68.
[39] S. repository, 2011. [Online]. Available: http://iotta.snia.org
[40] Umass trace repository, 2009. [Online]. Available: http://traces.cs.umass.edu/
[41] M. Kwon *et al.*, "TraceTracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction," in *Proc. IEEE Int. Symp. Workload Characterization*, 2017, pp. 87–96.
[42] Oracle, "Vdbench," 2000. [Online]. Available: https://www.oracle.com/downloads/server-storage/vdbench-downloads.html
[43] D. E. Difallah *et al.*, "OLTP-Bench: An extensible testbed for benchmarking relational databases," *Proc. VLDB Endowment*, vol. 7, no. 4, pp. 277–288, 2013.
[44] J. Redmon, "Darknet: Open source neural networks in C," 2013-2016. [Online]. Available: https://pjreddie.com/
[45] J. Zhang, M. Shihab, and M. Jung, "Power, energy, and thermal considerations in SSD-based I/O acceleration," in *Proc. 6th USENIX Workshop Hot Topics Storage File Syst.*, 2014, Art. no. 15.

Mincheol Kang received the BS degree in computer science and engineering from Dongguk University, Seoul, Korea, in 2014, and the MS degree from the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2016, where he is currently working toward the PhD degree with the School of Computing.



Wonyoung Lee received the BS degree from the Department of Computer Engineering, Ajou University, Suwon, Korea, in 2014, and the MS degree from the School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2016. He is currently working toward the PhD degree with the School of Computing, KAIST, Daejeon, Korea.

Jinkwon Kim received the BS degrees in industrial engineering and computer science and engineering from Hanyang University, Seoul, South Korea, in 2016, he is currently working toward the integrated master's and PhD degrees with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.

Soontae Kim (Member, IEEE) received the PhD degree in computer science and engineering from Pennsylvania State University, State College, Pennsylvania, in 2003. He was an assistant professor with the Department of Computer Science and Engineering, University of South Florida, Tampa, Florida, from 2004 to 2007. He has been with the School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Korea, since 2007, as an associate professor.