Received July 18, 2018, accepted September 16, 2018, date of publication September 28, 2018, date of current version October 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2872721

Dynamic Virtual Page-Based Flash Translation Layer With Novel Hot Data Identification and Adaptive Parallelism Management

QIWU LUO[1], (Member, IEEE), RAY C. C. CHEUNG[2], (Member, IEEE), AND YICHUANG SUN[3], (Senior Member, IEEE)
[1]School of Electrical and Automation Engineering, Hefei University of Technology, Hefei 230009, China
[2]Department of Electronic Engineering, City University of Hong Kong, Hong Kong
[3]School of Engineering and Technology, University of Hertfordshire, Hatfield AL10 9AB, U.K.

Corresponding author: Qiwu Luo (luoqiwu@hfut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 51704089, in part by the Anhui Provincial Natural Science Foundation of China under Grant 1808085QF190, in part by the China Postdoctoral Science Foundation under Grant 2017M621996, in part by the Fundamental Research Funds for the Central Universities of China under Grant JZ2018YYPY0296, and in part by the Ph.D. Special Research Fund of HFUT under Grant JZ2016HGBZ1030.

ABSTRACT Solid-state disks (SSDs) tend to replace traditional motor-driven hard disks in high-end storage devices in past few decades. However, various inherent features, such as out-of-place update [resorting to garbage collection (GC)] and limited endurance (resorting to wear leveling), need to be reduced to a large extent before that day comes. Both the GC and wear leveling fundamentally depend on hot data identification (HDI). In this paper, we propose a hot data-aware flash translation layer architecture based on a dynamic virtual page (DVPFTL) so as to improve the performance and lifetime of nand flash devices. First, we develop a generalized dual layer HDI (DL-HDI) framework, which is composed of a cold data pre-classifier and a hot data post-identifier. Those can efficiently follow the frequency and recency of information access. Then, we design an adaptive parallelism manager (APM) to assign the clustered data chunks to distinct resident blocks in the SSD so as to prolong its endurance. Finally, the experimental results from our realized SSD prototype indicate that the DVPFTL scheme has reliably improved the parallelizability and endurance of nand flash devices with improved GC-costs, compared with related works.

INDEX TERMS Solid-state drive (SSD), nand flash, flash translation layer (FTL), hot data identification, garbage collection.

I. INTRODUCTION

NAND flash-based solid-state drive (SSD) has been trending in various types of computers as well as in portable multimedia products, due to its advantages of smart size, non-volatility, energy-efficient and shockproof structure [1], [2]. Recently, many traditional hard disk drive (HDD) vendors regard the SSD business as the most important economic growth venue. This trend in the storage industry drives technological breakthroughs for both FTL and flash memory chips.

However, NAND flash memory also encounters inherent features that hinder its large-scale applications. The most inconvenient one is that the in-place update is not allowed due to the special organization of blocks and pages [3]. Once a page is occupied, it cannot be programmed until the whole residing block is erased. Such ***out-of-place update*** triggers a large number of invalid pages, which will suppress the performance of NAND flash devices. Another obvious peculiarity is that flash memory has ***limited endurance***. Once the erase counts exceed the maximum level of a block, the dedicated storage unit will become unavailable. Consequently, ‘‘frequent-erase’’ will decrease the device lifetime. Furthermore, the data-intensive multi-level cell (MLC) flash memory draws increasing attention due to its cost-effectiveness. Nevertheless, compared with single-level cell (SLC), MLC has lower endurance. Thus the reliability and lifetime of flash devices will continuously be hot and vital concerns in the storage area [4], [5].

An FTL is commonly deployed to emulate an HDD-like interface between the accessing host and physical flash memories. As for the former ***out-of-place update***, a recycling policy in FTL, namely garbage collection (GC), will be activated to reclaim the blocks when the number of invalid data hits the predefined threshold [6], [7]. Because expensive operations (i.e., copies and erases) are involved, frequent GC will result in significant computation overheads and random access memory (RAM) consumptions. In addition, the GC further challenges ***limited endurance*** of flash devices, then wear leveling (WL) is developed to improve storage lifetime by impartially distributing erases over whole flash memory [6], [8], [9]. Both the GC and WL fundamentally depend on the hot data identification (HDI). Various recent state-of-the-art HDI schemes focus on how to effectively capture the frequency information and recency information of write accesses by using reasonable runtime overheads and memory consumptions [6], [7], [29]–[33]. Traditionally, nearly all of these approaches initially insert ***all*** write accesses into the limited cache and then selectively evict useless ones (with cold LBAs) from the cache afterwards, by adopting special data structures. These accesses of the real-world I/O traces follow a so-called "80/20" rule reliably [33], which means recently accessed data will be more likely to visit again in near future [25], [34]. In other words, only a small fraction of memory space is frequently referenced. Thus, the possession ratio of hot data is always much lower than that of cold data. If the natures of temporal localities and spatial distributions could be exploited fully, then more lightweight HDI scheme would be achieved. This standpoint has been preliminarily proved by the sampling-based method in HotDataTrap [31]. Since mainly based on random under-sampling, this scheme also incurs some degradation on classification accuracy, especially when accesses tend to be decentralized. Consequently, there still remain imagination space on developing more efficient HDI scheme by utilizing the behavior rules of accesses. Inspired by this motivation, a generalized dual layer HDI (DL-HDI) framework is developed in current work, which can assist many HDI algorithms to selectively discard massive purified cold data (i.e., 50% of the whole items) in advance. Also, this paper considers only write accesses like MBF in [30].

Another significant aspect is how to expand the capacity of SSD while improve the parallelizability, which basically resorts to multi-channel and multi-way interleaving [17]. Generally, these approaches exploit parallelism of NAND flash from the external or internal perspective. The architectures in [10], [23], and [24] share a common starting point to magnify the external parallelism via fully utilizing the time slots between independent channels or between different ways on a certain channel. However, the limited bandwidth of the flash bus and the compact integration of consumer electronics turn into the insurmountable ceilings. Further, scholars start to take internal parallelism into consideration, attempting to improve request response speed and wear leveling performance, simultaneously [11], [12].

This paper presents a hot data-aware FTL architecture based on dynamic virtual page (DVPFTL). Besides the capacity and bandwidth of SSDs, the proposed DVPFTL emphasizes GC overheads and WL performance from the perspectives of HDI and internal parallelism. The major contributions are as follows.

- A generalized HDI framework, namely DL-HDI, is proposed to monitor the time-varying access behaviors based on bloom filters. It can improve many other HDI algorithms to discriminatively prevent massive cold data from entering the HDI procedure. Therefore, hot data can be accurately recognized with less runtime overheads and memory consumptions.
- An adaptive parallelism management methodology (APM) is developed to aggressively exploit the two-plane program operations and to distribute hot/cold data to appropriate resident places among flash memories. Hence, the internal parallelism can be fully exploited without ignoring wear leveling, while data migrations are performed imperceptibly during channel selection (external parallelism).
- The architecture of the DVPFTL is implemented on an FPGA-based prototype, which provides a reference case for the related studies on NAND flash devices.

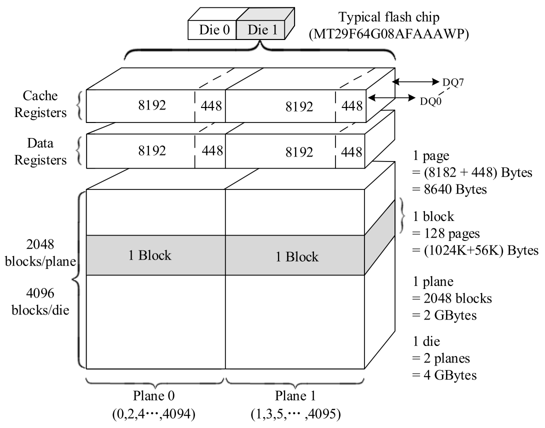
Under a hardware configuration with 8 channels and 2 ways, the realized SSD prototype has achieved an average program rate of 120 MB/s and an average read rate of 150 MB/s, respectively. Extensive experiments prove that the DVPFTL scheme has reliably improved the parallelizability and endurance of NAND flash devices yet with lightweight GC-costs.

The organization of the coming sections is as follows. Section 2 briefly summarizes the typical construction of NAND flash memory, FTL, HDI, and multi-interleaving. Theoretical analysis and detailed design, including the basic ideas and actualization procedures, are demonstrated thoroughly in Section 3. Details of experimental setup and performance evaluation for the DVPFTL are illustrated in Section 4. Finally, Section 5 concludes our research and outlines the future work.

II. BACKGROUNDS AND RELATED WORKS

A. NAND FLASH MEMORY

A typical NAND flash memory chip (package) is constituted of multiple dies, and each die is made up of several couples of planes, each plane contains a set of blocks, and each of them can be partitioned into a fixed number of pages, each page could be further divided into a set of sectors which has a user part and a spare part for storing user data and meta data respectively, among of which meta data is for house-keeping information like status flags, error correcting code (ECC) and logical block addresses (LBAs). The most currently available NAND flash consists of 1, 2, 4, or 8 dies, each die consists of 2 planes, each plane is composed of 2048 blocks, and every block consists of 64 or 128 pages with a size of 4 KB or 8 KB, hence the block size is 512 KB or 2048 KB. Take MT29F64G08AFAAAWP as an example in Fig. 1, it is a kind of 8GB 2-Die-2-Plane SLC NAND flash chip [13]. For better



Die 0
Die 1
Typical flash chip (MT29F64G08AFAAAWP)
DQ7
DQ0
Cache Registers
Data Registers
8192
448
1 page = (8182 + 448) Bytes = 8640 Bytes
1 block = 128 pages = (1024K+56K) Bytes
1 plane = 2048 blocks = 2 GBytes
1 die = 2 planes = 4 GBytes
2048 blocks/plane
4096 blocks/die
1 Block
Plane 0 (0,2,4···,4094)
Plane 1 (1,3,5,··· ,4095)

FIGURE 1. Physical structure of a flash chip.

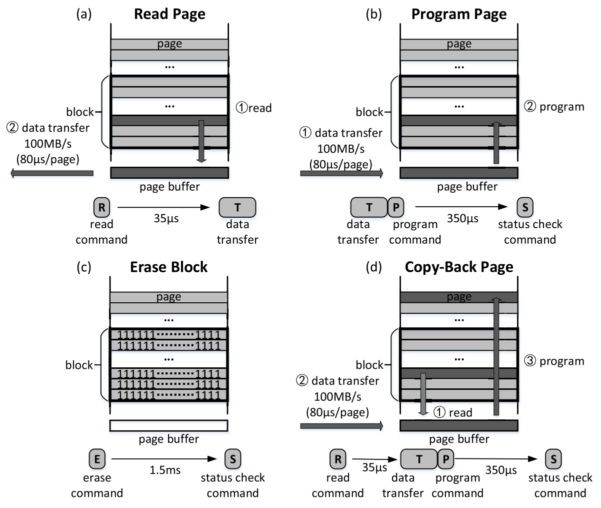


FIGURE 2. Flash operations: (a) page-read, (b) page-program, (c) block-erase, (d) page-copy-back.

speed matching, data is moved from or to the NAND flash memory array sequentially via data registers and cache registers. The program/read unit of NAND flash memory array is page while the erasure unit is block. Read, program and erase operations can be performed independently on each die, even performing that operations simultaneously on two adjacent planes is allowable, but all the two-plane commands at the same time can only be carried out with the same operation on the same die. These characteristics will be taken into account during the current work although the technique is not limited to that.

The four operations of read, program (write), erase and copy-back are performed by flash memory controller, each of them could be divided into three sequential phases: initiation, busy, and completion phase, among of which, the time duration of busy phase indicates the latency of the corresponding operation. For intuitive understanding, Fig. 2 illustrates the work principles of these operations. Besides, two-plane commands are universally supported by NAND flash vendors, operating the paired planes on one die synchronously will be propitious to the parallel response of visiting accesses. It is worth emphasizing that concomitant side-effect, dummy page overheads, should be carefully considered.

B. FLASH TRANSLATION LAYER

The FTL is built to support the downward interface obeying the timing characteristics of typical flash memory, while to provide an upward HDDs-like storage device interface, so as to achieve the goal for tolerating the idiosyncrasies of ***out-of-place update*** [14]. There are three main categories of FTL: page- [15], [16], block- [18], and hybrid-level mapping [19]–[21].

- ***Page-level mapping FTLs*** implement direct address-mapping between the logical and physical page numbers among the flash chip. When a new logical page visits, a free page is allocated under the page-level mapping table to store the supplied data before that table is updated. Due to its one-to-one mapping relationship, the page-mapping FTL is fairly flexible. For the same reason, however, its address mapping table is quite memory-consuming. Taking the above-discussed 8 GB flash chip as an example, the consumption of the mapping table is high up to 4 MB, not to mention the total table consumption of the whole multi-chipped SSD system. Consequently, page-level mapping FTLs are more suitable for the flash memories with smaller capacity.
- ***Block-level mapping FTLs*** describe logical address of file system request with two elements: one is logical block number which will be translated into a physical block address, and the other is page offset acting as unique index of location among the corresponding physical block. It is not difficult to discover the drawbacks under this mechanism that additional block erasure overheads would be incurred when the indexed page referred to its offset has been executed prior updates [19]. Consequently, although the memory-consumption of block-level mapping is far less than that of page-level mapping, it might increase the load of garbage collection with the accumulation of invalid pages.
- ***Hybrid-level mapping FTLs*** are highly flexible but more complex than the page-level and block-level mapping FTLs, which obtain more balanced performance between addressing complexity and memory expense. A consensus had been reached from BAST (Block Associative Sector Translation) in [20], that is, importing discriminative address translating mechanism for block-level and page-level mapping could save resources for keeping mapping information. The FAST (Fully Associative Sector Translation) pushed the mapping performance and memory overheads of log blocks to a new level through adopting fully-associative sector translations [21]. These log-buffer-based schemes can promote the program performance dramatically by importing a small quantity of log blocks. The BAST and FAST subsequently become classics among hybrid-level mapping FTLs. However, the consumptions of valid data page copies of BAST and FAST are inevitably increased, CFTL (Convertible Flash Translation Layer) arrived to further improve both the read and write performance by adopting its convertible mechanism and efficient caching strategy [22], [40], [41].

C. HOT DATA IDENTIFICATION

Hot data identification is an essential procedure to achieve efficient garbage collection and effective wear leaving, which is a crucial measure to compensate the distinct characteristics of out-of-place update and the endurance problem for NAND flash devices.

Considerable efforts have been performed to handle hot/cold separation in an efficient manner. Generally, these HDI approaches can be categorized into direct type [7], [29] and indirect type [6], [30]–[33]. The former HDC [7] and two-level LRU [29] build a counter for each page to record the number of accesses (indexed with LBAs) within a certain time interval. However, these direct approaches consume large memory spaces, since they assign one counter for each page. And then, lots of compact HDI approaches, basically adopting numerical transformation or statistical theory, have arisen subsequently to indirectly detect frequency and recency of accesses. Hsieh *et al.* [6] proposed a Multiple Independent Hash Function framework (MIHF), it adopts multi-hash functions and one bloom filter to acquire both the frequency and recency information. Park and Du [30] proposed a Multiple Bloom Filter framework (MBF) with a Window-based Direct Address Counting algorithm (WDAC), each bloom filter has a discriminative weight for hot degree, thus, the MBF can achieve more fine-grained recency as well as frequency. Liu *et al.* [32] imported a Kernel Density Function (KDF) into HDI for exploring the sparse information from probability distribution of accesses, this novel HDI-KDF scheme has been reported with a better performance than MBF. A new HotDataTrap (HDT) framework [31], which adopts a two-level hierarchical hash indexing mechanism cooperated by a sampling-based strategy, has achieved excellent performance on both false identification rate and memory overheads [35], [40], [41].

To sum up, an excellent HDI mechanism should effectively capture both the frequency and recency information of incoming accesses with lightweight memory consumption and computational overheads.

D. PARALLELISM EXPLOITATION

A great amount of interleaving architectures have been reported in current literatures to promote parallelizability of NAND flash memories [17], [23], [24]. As shown in Fig. 3(a), a chip-level interleaving improves the program performance dramatically through utilizing the inherent read latency effectively, but it has a bandwidth limitation under single bus. Further in Fig. 3(b), a bus-level interleaving can make full use of the misaligned time-slots among different flash buses. One of the most eye-catching findings in previous studies is that designers of SSD devices should pay more attention to the executing command latencies than to the bus frequency.

III. DESIGN AND ANALYSIS

A. DYNAMIC VIRTUAL PAGE FTL (DVPFTL)

In multi-chipped SSD design with multiple channels, two planes of the same chip could be accessed simultaneously,

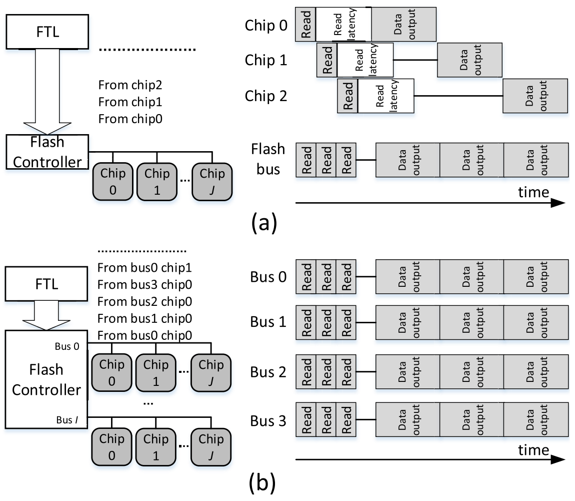


FIGURE 3. Chip-level interleaving (a), and bus-level interleaving (b).

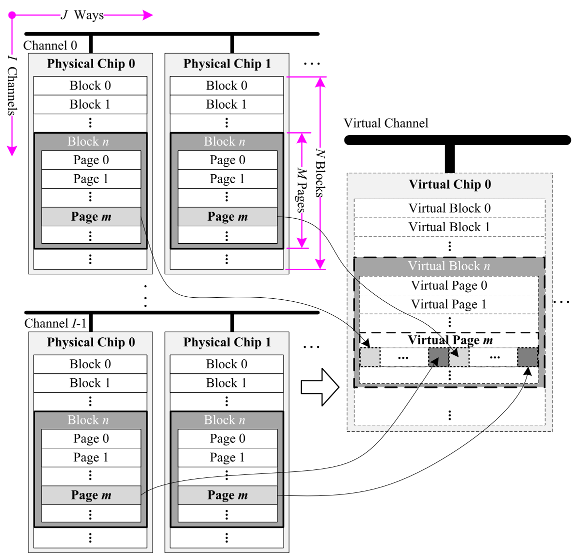


FIGURE 4. Construction procedure of Virtual Pages (VPs).

chips from the same channel could also operate independently, moreover, chips from separate channels could be accessed in parallel. In order to make full use of the parallelism in the multi-interleaving architecture and reduce the address mapping table size of the FTL, we propose a virtual page (VP) architecture demonstrated in Fig. 4 from micro perspective. Assuming that there are I channels and J ways in the designed SSD, each physical flash chip (PFC) is built with N blocks, and each physical block (PB) is made up of M physical pages (PPs). Particularly, it is assumed that each of adjacent 2 ways reconstitute into one virtual flash chip (VFC). Hence, a new VFC contains $2I$ PFCs which are distributed in different channels. For better understanding, the VFC (j) is made up of PFC $(i, 2j)$ and PFC $(i, 2j+1)$, where $i \in [0, I-1]$ and $j \in [0, J/2-1]$, and the PFC (i, j) denotes the j^{th} PFC from the channel i. As the architecture of PFC, each VFC is organized with virtual blocks (VBs),

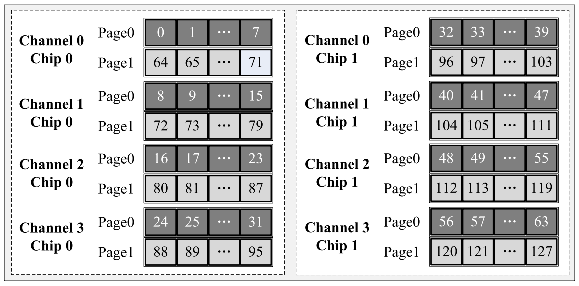


FIGURE 5. Internal constitution of virtual page.

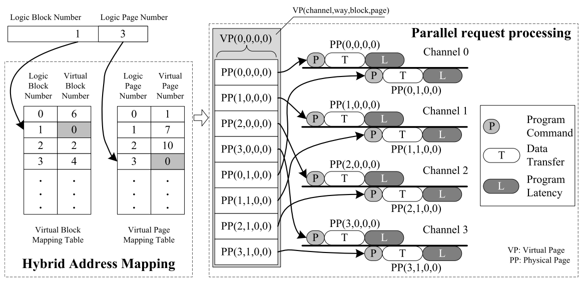


FIGURE 6. Parallel request processing.

and each VB contains a set of VPs, the difference is each of VP is composed with $2I$ PPs. Accordingly, the VP (j, n, m) consists of PP $(i, 2j, n, m)$ and PP $(i, 2j + 1, n, m)$, where $i \in [0, I - 1]$, $j \in [0, J/2 - 1]$, $n \in [0, N - 1]$ and $m \in [0, M - 1]$, and the PP (i, j, n, m) represents the m^{th} PP in the n^{th} PB of the j^{th} PFC from the channel i. It is worth mentioning that the proposed virtual page architecture is not limited by the multi-plane/die features of flash chip. Based on the virtual page architecture, a two-layer address translation mechanism can be turned into reality. The bottom layer is in charge of the address mapping between the actual PFCs and the predefined VFCs with block mapping scheme, while the top layer translates LBAs requested from host file system to the intermediate virtual block addresses (VBAs) under log-based hybrid algorithm, the mapping tables are kept in DDR3 SDRAM, besides, one backup is stored in a private space on flash memory. Consequently, the peculiarities of large RAM consumption caused by page mapping and frequent copy operations caused by block mapping will be hidden to a certain extent. An operation tip is worth mentioning in the address mapping from LBAs to VBAs: an arrived logical block data cannot be distributed as fragments to different VFCs, which is to obey the inherent requirement for issuing in-place copy-back operations.

Two rounds of flash interleaving are operated in the proposed virtual page architecture. The first channel-level interleaving expands the bus bandwidth of flash memories, and the second chip-level interleaving further suppresses bus idle time. As shown in Fig. 5, if suppose the channel-degree $I = 4$, and the way-degree $J = 2$, then the sectors from separated channels are combined in a round-robin modality. Intuitively, in the left VP, sector groups $0 \sim 7$, $8 \sim 15$, $16 \sim 23$, and $24 \sim 31$ are obtained from channel 0, 1, 2, and 3, sequentially. Under such configurations, each VP contains 8 sector sections (i.e. 8 PPs), and each sector section contains 8 distinct sectors.

Now, we make an attempt to demonstrate the multi-level interleaving parallelism from the perspective of request response. As shown in Fig. 6, when the hybrid address mapping layer receives a write request, its LBA will be translated into VBA and VPA. Subsequently, the bottom block mapping layer will allocate this request to the related PPs of the VP under parallel accessing. Compared with conventional FTL works (i.e., Hydra [23]), we only treat such parallel architecture as a basic hardware bench. Balanced performance on GC-overheads and endurance are pursued by developing efficient HDI scheme and exploiting comprehensive internal parallelism, which will be elaborated in the upcoming Section III-B and Section III-C.

B. DUAL LAYER HOT DATA IDENTIFICATION (DL-HDI)

For the NAND flash-based storage system, it is widely accepted that the hot data have a significant impact on its performance as a result of GC activities in the FTL. In this section, a generalized dual layer HDI (DL-HDI) scheme is presented in detail. In particular, we first propose a pre-classifier based on single bloom filter, it enables DL-HDI to early discard massive cold data so as to reduce runtime overheads as well as to avoid a waste of memory spaces. Then, we take one state-of-the-art HDI scheme (i.e., MBF [30]) as its hot data post-identifier to accurately capture the hot items. Finally, an improved HDI scheme, DL-MBF, is generalized from the original MBF.

1) BASIC IDEA AND MOTIVATION

Temporal localities in data access are often observed in plenty of daily applications [25]. Many real-world workloads exhibit a statistical nature that about 80% of accesses are directed to no more than 20% of data and only touch about 1% of the address space of the disk [6], [33]. Consequently, cold data occupy a dominant position in the probability of occurrence. Handling all these accesses indiscriminately through the cache-limited HDI procedure will undoubtedly result in great waste on computational resources and memory spaces. This motivates us to pursue an ultra-simple separation mechanism to accurately discard purified cold data before the main HDI procedure.

Bloom filter (BF) is a space-efficient data structure which specializes in probabilistically testing set membership with allowable errors. Although a given key is not in the set, a BF may provide a false positive, but it never trigger a false negative. In recent studies (i.e., MIHF [6] and MBF [30]), scholars devote themselves to achieve higher true positive rate (TPR) with more lightweight space. However, ***to our best knowledge, the zero false negative nature of BF has rarely been applied in HDI***. With converse thinking, we set up a smart pre-classifier through utilizing this zero false negative feature to capture a large majority of purified cold

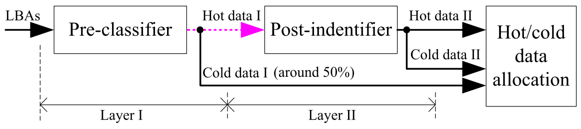


FIGURE 7. Data flow under the proposed DL-HDI framework.

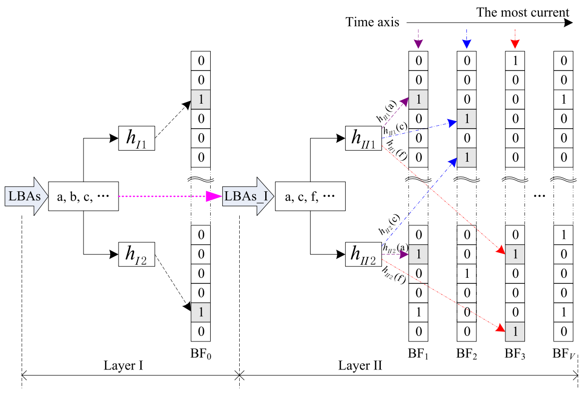


FIGURE 8. Structures of the pre-classifier (Layer I) and post-identifier (Layer II), with an instantiation of DL-MBF.

data with absolute confidence, so as to prevent the captured cold data from entering the cache and HDI procedure. The basic data flow is illustrated in Fig. 7, the pre-classifier separated all the accesses in half: coarse *hot data I* and purified *cold data I*. This process should consume negligible memory space and computational overheads compared with the post-identifier. Then the replaceable post-identifier (i.e., MBF [30], or HDT [31], etc.) executes the HDI operation as usual, outputting the *hot data II* (involving fewer false positive) and *cold data II*.

2) SINGLE BLOOM FILTER-BASED PRE-CLASSIFIER

As shown in Fig. 8, in a certain decay duration (i.e., the window size $= 2^9$ [30]), whenever write requests are issued, the bloom filter (BF_0) is selected and the hash values (LBA) are recorded to it. Different from the traditional usage of bloom filter, the pre-classifier first maintains the unselected LBAs (BF values remain ‘0’) to the category of *cold data I*, then the remainder LBAs constitute the optimal *LBAs_I*, which are treated as the coarse *hot data I* in Fig. 7. This accesses with LBAs_I will be fed to the downstream post-identifier, the MBF is chosen in this instantiation.

3) PARAMETERS SETTING AND ANALYSIS

Applications of a BF are required to configure three parameters: BF size (M_{BF}), the number of hash functions (K_{BF}) and the number of unique elements (N_{BF}), and the N_{BF} corresponds to the very decay duration. These key parameters are closely related with each other and have crucial impacts on the performance of BF. According to [36], an optimal BF size (M_{BF}) can be estimated using

$$M_{BF} = -\frac{N_{BF}\, ln(f_{BF})}{(ln2)^2} \qquad (1)$$

where f_{BF} is a permitted false positive rate (FPR), after M_{BF} is determined, an optimal number of hash functions (K_{BF}) can be estimated from

$$K_{BF} = \frac{M_{BF}(ln2)}{N_{BF}} \qquad (2)$$

As the pre-classifier focuses more on obtaining purified cold data than achieving high TPR, the objective values of f_{BF} and N_{BF} are loosely configured as to be 15% and 2^9, respectively. Through (1) and (2), $M_{BF} = 2022$ (choose 2^{11}), and $K_{BF} = 2.77$ (choose 2), these configurations are consistent with those of MBF [30]. Further, the probability that any one bit in the bitmap of a bloom filter is still ‘0’ (not selected) is

$$P_0 = (1 - \frac{1}{M_{BF}})^{K_{BF}N_{BF}} \approx e^{-\frac{K_{BF}N_{BF}}{M_{BF}}} \qquad (3)$$

where the condition of the “approximation” is $M_{BF} \to +\infty$. The FPR is calculated as

$$\begin{aligned} f_{0\to1} &= (1 - P_0)^{K_{BF}} \\ &\approx (1 - e^{-\frac{K_{BF}N_{BF}}{M_{BF}}})^{K_{BF}} \end{aligned} \qquad (4)$$

According to (3), under our parameter settings, $P_0 = 60.65\%$, it means that about 60% of bits of the BF_0 will remain ‘0’ after a decay duration (512 accesses). Conservatively, suppose we only discard 256 items of the clustered cold accesses, *the FPR ($f_{0\to1}$) of the post-identifier (original MBF) will be improved from the former 15.48% to current 6.82% according to (4)*, obtaining this improvement should afford an extra memory overhead of 0.25 KB (refer to Fig. 8). Another technical route is to *accept the equal performance on FPR, our DL-HDI scheme requires smaller BF size (M_{BF}) for post-identifier, permitting at least a half reduction in memory space for downstream BFs*. In other words, compared to the original MBF, the improved DL-MBF can achieve the equal level of FPR, but consumes only 75% memory space (0.75 KB vs. 1 KB). These results theoretically support our initial motivation, that at least a half of purified cold data can be indeed discarded at the very beginning of HDI procedure, which is benefit to decrease both the memory consumption and computational overheads, our proposed DL-HDI framework is precisely produced for such considerations. Meanwhile, we would like to emphasize the remarkable generalized property of our proposed DL-HDI scheme. The post-identifier is not limited to the original MBF, many other HDI algorithms such as MIHF [6], HDT [31], HDI-KDF [32], *etc.* can obtain improvements under our DL-HDI framework. Then various variants such as DL-MIHF, DL-HDT, DL-HDI-KDF, *etc.* could be established in turn.

C. ADAPTIVE PARALLELISM MANAGEMENT (APM)

The above Section III-B focuses on the identification mechanism of hot and cold data, while in the current section, we concentrate more macroscopically on wear leveling through adaptive parallelism management (APM), which can improve the throughput by handling requests in parallel under the multi-chipped SSD architecture.

Refer to Fig. 4, when using the type of flash chip in Fig. 1, the storage device is consisted of I channels, $C_0, C_1, \ldots C_{I-1}$, each channel C_i includes $2J$ dies, $D_{i,0}, D_{i,1}, \ldots D_{i,2J-1}$, then each die $D_{i,j}$ contains Q planes, $PL_{i,j,0}, PL_{i,j,1}, \ldots PL_{i,j,Q-1}$. Suppose block B_l has experienced e_l times of erasure operations, the total erasure times in the plane $PL_{i,j,q}$ can be defined as

$$e_{i,j,q} = \sum_{B_l \in PL_{i,j,q}} e_l \tag{5}$$

Taking a panoramic view of erasure counts among all flash planes, the maximum minus the minimum is

$$\begin{aligned}\Delta e &= e_{\max} - e_{\min} \\ &= \max\{e_{i,j,q}\} - \min\{e_{i,j,q}\}\end{aligned} \tag{6}$$

where, $i \in [0, I-1], j \in [0, 2J-1]$, and $q \in [0, Q-1]$ are respectively the indexes of channel, die, and plane.

The adaptive arrangement of program requests consists of the following three stages.

- ***Stage 1: External parallelism via multi-interleaving***. Program requests from top layer are dispensed to independent channels one after another through adopting multi-interleaving mechanism. Therefore, the throughput can be improved by overlapping flash operations in multiple independent channels. This external parallelism method is pretty traditional in the SSD design, so we will focus on the later two stages.
- ***Stage 2: Internal parallelism via two-plane operation***. Each program request will be stripped into two sub-requests (Sub_req1, Sub_req2), and then be dispensed to a couple of planes on one die among the selected channel. After two thresholds, Th_{dn} and Th_{up} ($Th_{dn} < Th_{up}$), based on the erasure status are pre-defined, APM will distribute Sub_req1 and Sub_req2 to the related channels. In particular, The Sub_req1 will be allocated to the plane with the minimum number of valid pages in the designated channel when $\Delta e < Th_{dn}$, and will be allocated to the plane with the minimum erasures in the designated channel when $Th_{dn} \leq \Delta e < Th_{up}$. Otherwise, if $Th_{up} \leq \Delta e$, they will be allocated to the plane with the minimum erasures among all planes. Besides, a state flag, *Assign_Status*, is updated in time to record the current status of request assignment. The plane allocation method issued by APM for the Sub_req2 is the same as that of Sub_req1, except that the selected plane must be among the same die of Sub_req1, which is for adapting the essential characteristics of two-plane program commands.
- ***Stage 3: Discriminative strategy for two-plane program***. Generally, two-plane program commands benefit more to the larger and more sequential requests. However, to random and small program requests, it shows less (even no) advantages to normal (non two-plane) program commands, since the program expenses dramatically increase with the triggered dummy pages. In our DVPFTL scheme, two-plane program operations will not be issued if the improvement of response time could not achieve the threshold α ($0 < \alpha < 1$), the quantitative analysis is described as follows.

Suppose the total number of pages to be written is Y_{req} and the time consumption for normal program command per one page is T_{nomal}, the total time required for serving all the program requests without any two-plane operation is $Y_{req} \times T_{nom}$, while the final time spent on handling the program request through executing two-plane operation is $Y_{dum} \times T_{dum} + Y_{req}/2 \times T_{2plane}$, where Y_{dum} and T_{dum} are respectively the counts of triggered dummy pages and the programming time consumption per dummy page, and T_{2plane} denotes programming time consumption per a couple of pages through executing two-plane program operation. Further, our DVPFTL would not gain from two-plane program commands until (7) is matched.

$$Y_{dum} \times T_{dum} + \frac{Y_{req}}{2} \times T_{2plane} \leq \alpha \times Y_{req} \times T_{nom} \tag{7}$$

In other words, APM will be aggressively exploiting two-plane program operations as long as the ratio $\frac{Y_{dum}}{Y_{req}}$ meets

$$\frac{Y_{dum}}{Y_{req}} \leq \frac{\alpha T_{nom}}{T_{dum}} - \frac{T_{2plane}}{2T_{dum}} \tag{8}$$

otherwise, APM will dynamically convert to execute the normal program commands. The greater Δe, or the smaller α, the higher probability of triggering wear leveling, otherwise, the higher probability of issuing two-plane program operation. Tentatively, the value of α is set as 0.65 for the upcoming experiments in current research.

D. PERFORMANCE SPEEDUP ANALYSIS

This section is meant to analyze the performance of parallel speedup. If suppose $\{r_k\}$, $k \in [0, K-1]$ is an incoming sequence with K host requests, involving K_r reads, K_p programs, and will trigger K_e erases, hence there exists $K = K_r + K_p$. Further, suppose H is the maximum timeslot between two requests which could be operated in parallel in a real-world FTL. In particular, r_k is a read or program request, p_{r_p} or p_{p_p} is the respective probability that a request could be operated with r_k in parallel within H requests close following r_k. And t_r, t_p and t_e are the time costs for the read, program and erase operations, respectively. The performance speedup of a parallel flash memory can be estimated according to Amdahl's law

$$P_{speedup} = \frac{T_s + T_p}{T_s + \frac{T_p}{\beta}} = \frac{1}{1 + (\frac{1}{\beta} - 1)\frac{T_p}{T_s + T_p}} \tag{9}$$

where T_s and T_p are the time costs of serial and parallel operations (total time consumption $T = T_s + T_p$), and $\beta (\geq 1)$ is the parallel degree of operations. In a general flash storage system, the total time overhead of all parallel operations can be summed as

$$T_p = T_{r\_p} + T_{p\_p} + T_{e\_p} \tag{10}$$

where $T_{r\_p} = K_r \cdot p_{r\_p} \cdot t_r$, $T_{p\_p} = K_p \cdot p_{p\_p} \cdot t_p$ and $T_{e\_p} = K_e \cdot p_{e\_p} \cdot t_e$, which are the time costs for parallel operations of reads, programs and erases, respectively. Then (9) can be rewritten to

$$P_{speedup} = \frac{1}{1 + (\frac{1}{\beta} - 1)\frac{T_{r\_p}+T_{p\_p}+T_{e\_p}}{T_s+T_{r\_p}+T_{p\_p}+T_{e\_p}}} \tag{11}$$

For a certain request sequence, the values of K_r, K_p, t_r, t_p and t_e are previously determined. Based on the above derivation, the larger T_p, β and H are, the higher is the performance $P_{speedup}$ that can be obtained, among which, however, H is limited to the upper ceiling of processing requests in a definitized duration, while β is in the same situation as that the limited resources can not support an endless increase in parallelism. Consequently, three measures to improve parallel performance are as follows (where '$\uparrow$' or '$\downarrow$' means value increase or decrease).

- "$p_{r\_p} \uparrow \Rightarrow T_{r\_p} \uparrow$", the only measure to increase p_{r_p} is to set H to a relatively large number, which could increase the probability of performing parallel read operations on NAND flash memories.
- "$p_{p\_p} \uparrow \Rightarrow T_{p\_p} \uparrow$", the FTL can help to increase p_{p_p} by effectively determining where and how to program the data on NAND flash memories. Also, a larger H can increase p_{p_p}. Efficient FTLs can increase the probability of performing parallel program operations by using smart algorithms and lightweight data structures.
- "$p_{e\_p} \uparrow \Rightarrow T_{e\_p} \uparrow$", the FTL completely determines p_{e_p} to let there are enough blocks (to be reclaimed) uniformly distributed on each plane, which could increase the probability of performing parallel erase operations according to the pigeonhole principle.

Based on the above analysis, DVPFTL is designed not only to maximize parallelizability ($P_{speedup}$), but also to improve the GC efficiency and decrease the WL degree by taking the following measures: **(1) Omnidirectional parallel architecture** (refer to Section III-A), **(2) Dual layer hot data identification** (refer to Section III-B), and **(3) Adaptive parallelism management**(refer to Section III-C).

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

This section provides diverse experimental results and comparative analyses. First, extensive experiments are carried out to evaluate our proposed DL-HDI framework and APM scheme. Second, the overall performance of DVPFTL-based SSD is demonstrated on our implemented on-chip prototype.

A. PERFORMANCE SIMULATION RESULTS

1) SIMULATION SETUP

We compare our DL-HDI framework (DL-MBF as a case here) with three other state-of-the-art HDI schemes, MIHF [6], MBF [30], and the Window-based Direct Address Counting (WDAC) [30]. The WDAC, which is an ideal baseline scheme proposed in MBF, continues to be used as the baseline for the following experiments. In particular, the DL-MBF is instantiated to a standard version, DL-MBF_s, and a compact version, DL-MBF_c. Table 1 summarizes their detailed system parameters. According to the dual layer structure designed in Section III-B, our schemes (DL-MBF_s and DL-MBF_c) utilize a half size decay duration for their post-identifiers compared with MBF, but the former scheme (DL-MBF_s) consumes an extra memory cost of 0.25 KB. Hence, we also evaluate a more compact one (DL-MBF_c) for clearer understanding. To be fair, we configure the parameters of the three contrastive schemes as same as those in [30]. Also, we select identical hash functions for all the five schemes in Table 1.

TABLE 1. System parameters and values.

System	The state-of-the-art			DL-MBF_s		DL-MBF_c	
Parameters	WDAC	MIHF	MBF	pre-	post-	pre-	post-
M	N/A	2^{12}	2^{11}	2^{11}	2^{11}	2^{11}	2^{10}
V,$W(bit)$	N/A	$1*4$	$4*1$	$1*1$	$4*1$	$1*1$	$4*1$
MC (KB)	N/A	2	1	0.25	1	0.25	0.5
N	2^{12}	2^{12}	2^9	2^9	2^8	2^9	2^8
K	N/A	2	2	2	2	2	2
HDT	4	4	4	N/A	2	N/A	2
RWS	0.000488	N/A	0.5	N/A	0.5	N/A	0.5

M is the BF size, V and W are BF number and width. MC denotes BF memory costs. N is the decay (window size),K is the hash function number, HDT is hot degree threshold, RWS is recency weight stepping. 'pre-' denotes pre-classifier, and 'post-' denotes post-identifier.

TABLE 2. Workload characteristics.

Workloads	Total Requests	Request Ratio (Read:Write)	Inter-arrival Time (Avg.)
Financial1	5,334,987	R:1,235,633(22%) W:4,099,354(78%)	8.19 ms
MSR	1,048,577	R:47,380(4.5%) W:1,005,197(95.5%)	N/A
Distilled	3,142,935	R:1,633,429(52%) W:1,509,506(48%)	32ms
MillSSD	3,581,731	R:53,726(1.5%) W:3,528,005(98.5%)	809.27 ms

Similar to the testing procedure in [30], we adopted four real workloads for objective evaluation. As shown in Table 2, *Financial1* is a write intensive trace file [37]. *Distilled* stands for a general usage patterns in a PC [29] which was also adopted in MIHF [6]. *MSR* represents typical workloads of large enterprise servers [38]. Lastly, we employ a real-world SSD trace file, *MillSSD*, from an industrial personal computer (IPC: Intel X2 7400, 2G DDR3, Windows 7), which provides image backup service for our defect inspection instrument [39] for flat steel surface in a hot-rolling mill. We installed Runcore's RCS-V-T25 SSD (512GB, SATA2) on the IPC to gather traces such as caching images, updating image defect database, storage for daily report information, etc. Because involves lots of storage operations, *MillSSD* is quite write intensive.

2) PERFORMANCE EVALUATION OF DL-MBF

Four performance metrics, including *hot ratio*, *false identification rate*, *runtime overhead*, and *memory consumption*, are observed in this section in Fig. 9, Fig. 10, Fig. 11, and Table 1, respectively.

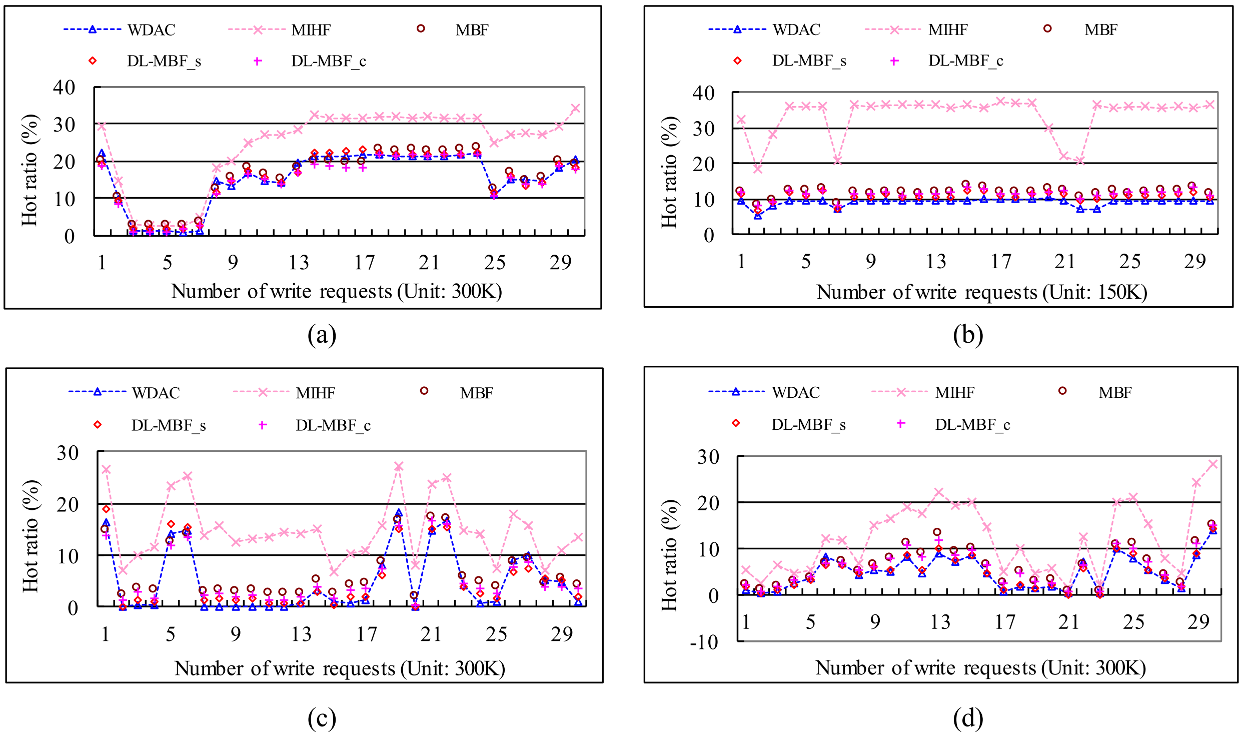


FIGURE 9. **Hot ratios under various workloads (read % : write %). (a) Financial1 (22% : 78%). (b) MSR (4.5% : 95.5%). (c) Distilled (52% : 48%). (d) RealSSD (51% : 49%).**

• ***Hot Ratio:*** As illustrated in Fig. 9, the MBF and its two generalized schemes perform better than MIHF. Intuitively, the MBF's hot ratio curve fluctuates near the upside of the baseline as expected in [30], while our DL-MBF_c scheme draws a considerably similar trajectory, when it comes to our DL-MBF_s scheme, its hot ratio curve even tends to hit the baseline from the above direction most of the time. Although the trajectories under four workloads are not identical, the aforementioned relative tendency exhibits reliably, which proves that our proposed DL-HDI framework has pushed the hot ratio of the MBF to a more objective level.

• ***False Identification Rate:*** For further quantitative evaluation, we analyze false identification rates by comparing the hot ratios of corresponding HDI scheme to the shared baseline. MIHF is omitted in this test since we focus mainly on the performance differences of the same MBF-series of schemes. As illustrated in Fig. 10, our DL-MBF_s scheme performs coherent lower false identification rates than its original MBF, with average performance improvements by 16.97%, 29.41%, 26.94%, and 32.71% under the four workloads successively. Interestingly, even the compact DL-MBF_c scheme exhibits a slightly advantage (about 2.21% average improvement) to MBF. These results are basically consistent with the derivation in Section III-B3, they indicate our simple idea of eliminating most of the pre-classified cold data in advance has worked well, and is indeed benefit to HDI.

• ***Runtime Overhead:*** Computational overhead is an important metric to evaluate HDI scheme. For fair comparison, we exhibit two most representative overheads namely *Checkup* and *Decay* in Fig. 11, where *Checkup* counts the CPU clock cycles spending on the hot data verification process to check whether the data in the LBA are hot, and *Decay* corresponds to the aforementioned aging process. All these statistical results are carried out on an IPC (Intel X2 7400, 2G DDR3, Windows 7), and the exhibited records are the average values through a series of repeated testings fed with many write requests (i.e., 100 K numbers of write requests from *MSR*). As shown in Fig. 11, compared with MIHF, MBF scheme obtains considerable improvements both on *Checkup* and *Decay*. It is noteworthy that the corresponding scores are slightly better than those reported in [30], since our test platform is equipped a CPU with higher performance. Nevertheless, the comparison testing will not be impacted since we concentrate on relative performance. For *Checkup*, the runtime overheads of our two DL-MBF schemes are comparable to that of MBF as a result of their similar BF-based structures and identical hash functions. When it turns to *Decay*, the DL-MBF_s and DL-MBF_c schemes considerably reduce runtime overheads by an average of about 12% and 46% compared to that of MBF, respectively. However, substantially, both the DL-MBF_s and DL-MBF_c scheme trigger more runtime overheads since they need afford extra CPU clock cycles on checkup procedure and aging mechanism of the pre-classifier. Even in this case, the final runtime overheads still decrease uncompromisingly, it is mainly attributed to the following reasons:

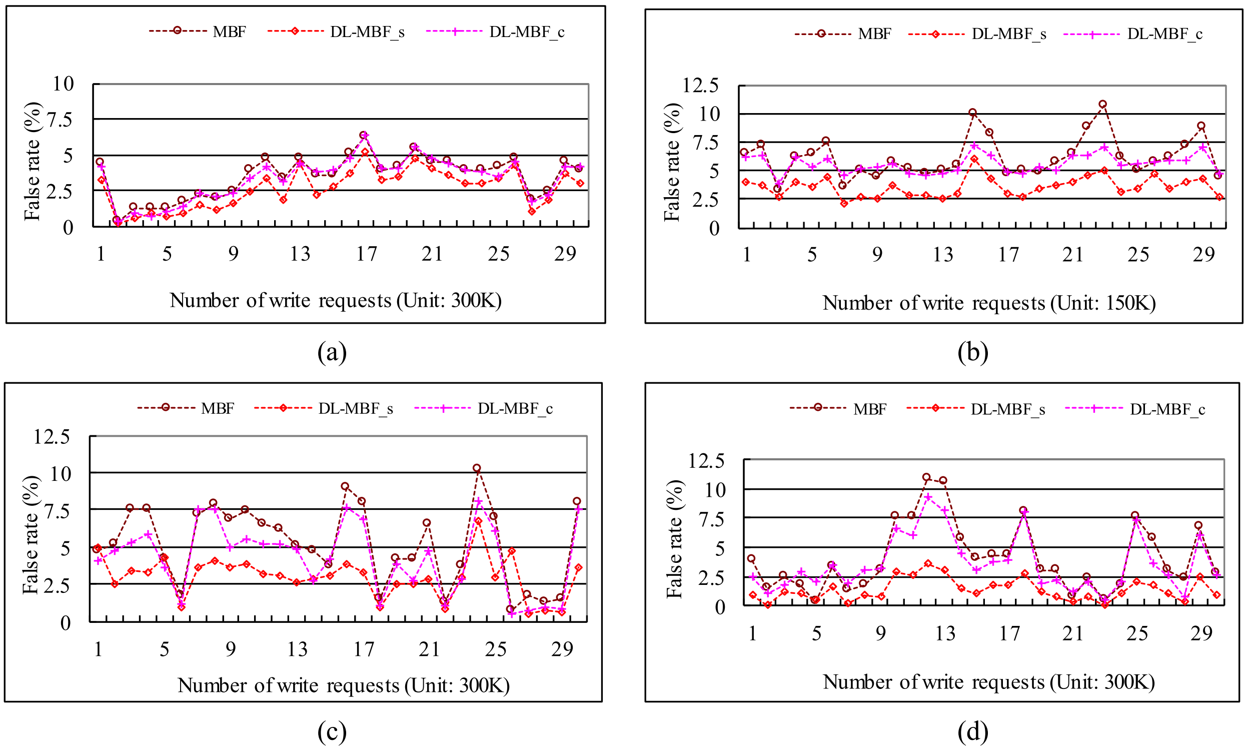


FIGURE 10. False identification rates under various workloads (read %: write %). (a) Financial1 (22% : 78%). (b) MSR (4.5% : 95.5%). (c) Distilled (52% : 48%). (d) RealSSD (51% : 49%).

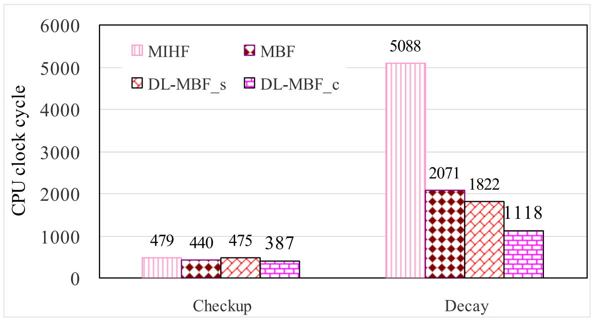


FIGURE 11. Average runtime overhead per operations.

1) The abandon of half of the data in advance greatly counteracts that expense generated by the pre-classifier. 2) The dominant decay operation of the post-identifier completely inherits the advantages of original MBF, while the rhythm of the extra decay operation (of the pre-classifier) is only 1/8 of that of the post-identifier. 3) The half BF size (2^{10} vs. 2^{11}) of DL-MBF_c further reduces the involved data bits for *Checkup* and *Decay* operations.

• ***Memory Consumption:*** In the third row of TABLE 1, we has calculated the basic memory costs of our DL-MBF schemes and the contrastive schemes. It is clear learned that our DL-MBF_s contains 1 number and 4 numbers of 2^{11} sizes of 1-bit array for pre-classifier and post-identifier, respectively. Thus, the total memory cost for maintaining these BFs are 1.25 KB. With a more compact BF size, the DL-MBF_c scheme consumes fewer memory of 0.75 KB. A thorough investigation is presented in [30] regarding the approaches to choose an appropriate size and number of BFs. Here the focus is how to utilize limited memory resource (i.e., around 1 KB) to achieve better performance. As analyzed above, taking the state-of-the-art MBF's performance as a new benchmark, our DL-MBF_s obtains a competitive identification accuracy but requires extra 0.25 KB memory expense. This scheme is highly recommended for applications that focus more on SSD performance than on memory expense (i.e. *MillSSD*). Otherwise, we suggest DL-MBF_c scheme, especially for the mini portable devices. Another advanced variant, DL-MBF_a, is more specialized in handling random requests than DL-MBF_s. It consumes the same quantity of memory, but consists of 8 numbers of 2^{10} sizes of 1-bit array for post-identifier.

3) PERFORMANCE EVALUATION OF APM

It can be learned from Section III-C that the frequency of wear leveling will decrease in DVPFTL for larger values of Th_{dn} and Th_{up} in most cases, while smaller values set to these two thresholds will result in unnecessary block erasures, which will increase the burden of garbage collection. Hence, we had been testing and verifying these two parameters for a long duration, and finally the balanced value combination was confirmed with Th_{dn} and Th_{up} being 100 and 200, respectively.

For the compatibility of testing, we continue to use the aforementioned workload of Distilled (52%:48%) for the

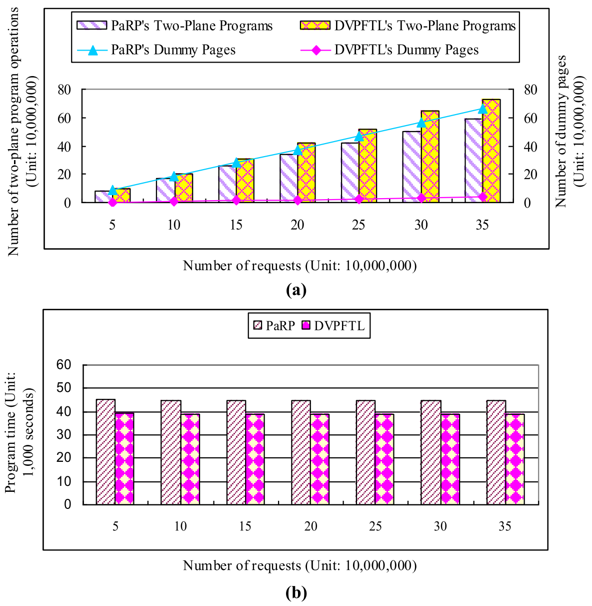


FIGURE 12. Comparative performance tests for APM. (a) Issued two-plane write operations and incurred dummy pages. (b) Accumulated program response time per requests interval.

performance evaluation of APM, and the flash type is the unchanged 2-die-2-plane type (refer to Fig. 1). Here we selected the Parallelism-aware Request Processing (PaRP) scheme [11] as our learning benchmark, since it also considers both external and internal parallelism by exploiting two-plane operations. As shown in Fig.12(a), we demonstrate the contrastive total numbers of the arisen two-plane program operations and the incurred dummy pages, respectively. It can be observed that DVPFTL dispatched more two-plane program operations than that of PaRP, the major reason is that DVPFTL attaches great importance to internal parallelism, utilizing the two-plane command for each program request. When it comes to the PaRP, two-plane program operations would not be issued until two successive program requests are arriving. It can be also learned that the dummy page consumptions incurred from DVPFTL are much less than that of PaRP, the reason is that the pre-defined threshold α possesses discriminative capability to return unnecessary two-plane program operations in time. Furthermore, Fig. 12(b) illustrates the contrastive situation of accumulated program response time per requests interval between DVPFTL and PaRP. Intuitively, our DVPFTL outperforms PaRP by 13.49% on the program latency reliably. The evaluation results in Fig.12 prove that the internal parallelism-aware mechanism can arrange the program request adaptively, which benefits the garbage collection for better wear leveling.

B. OVERALL ON-CHIP PERFORMANCE RESULTS

1) PROTOTYPE IMPLEMENTATION

We have implemented a multi-chipped SSD (MCSSD) prototype for the test-bench, its hardware architecture is shown in Fig. 13. Our proposed DVPFTL is marked in the gray box, which is composed of two key modules: dual layer hot data identifier (DL-MBF for test) and adaptive parallelism manager. The FPGA (XC6VLX240T) on this in-house development board has been pre-configured with an embedded 32-bit software processor, MicroBlaze [26], it has been equipped with 512 MB mobile DDR3 SDRAM for keeping mapping tables (of which only 16 MB is used by our prototype implementation) and software codes. The board is also equipped with interfaces of SATA, G-bit Ethernet and UART for debugging and future extension. In our test-bench, the multi-channel degree I and multi-way degree J are set to 8 and 2, respectively, which means 16 NAND flash memory chips (NFMCs) are mounted through 8 independent flash buses. The selected NFMC model is MT29F64G08AFAAAWP (refer to Fig. 1), its maximum operating frequency of one flash bus is 100 MHz, in order to reserve certain engineering margin, the flash bus is set to 60 MHz, providing a bandwidth of 60 MB/s. The MCSSD controller is made up with the MicroBlaze processor, the dual-buffer and the multiplexer. A SATA 3.0 interface controller arranged between the external host and internal MCSSD controller, the maximal supporting bandwidth of 600 MB/s is competent for the 480 MB/s data rate after 8-bus interleaving. All these embedded peripherals on FPGA chip are managed by the MicroBlaze processor via processor local bus (PLB), and they are implemented integratedly on the Xilinx ISE and EDK tools. The realized MCSSD and its test scene photo are presented in Fig. 14.

Apart from the multi-interleaving, we configure a pair of buffers lying between the DVPFTL and Demux/Mux to improve the interaction efficiency through so-called "ping-pong" operation. The operating frequencies of Microblaze processor and DDR3 SDRAM are set to 150 MHz and 100MHz in the tests, respectively.

2) TEST-BENCH SETUP (PCMark05)

For the performance evaluation of multi-interleaving architecture, although the latest PCMark 10 has been released, we continue to use a more classical version, PCMark05, for fairer comparison especially to some of the previous state-of-the-art works (i.e., Hydra in [23]). Its storage benchmark contains *five* types of workloads: *OS Startup*, *Application Loading*, *General Usage*, *Virus Scan*, and *File Write*, which can well emulate workloads of typical PC environments [23], [27]. *OS Startup* stands for a boot-up behavior of Windows XP, the reading and writing operations occupy about 90% and 10% of all the requests, respectively. *Application Loading* involves launch and termination of application programs, such as IE explorer, Skype, and Adobe softwares, etc., it contains around 83% reads and 17% writes. *General Usage* describes the daily usage of a PC, which consists of about 60% reads and 40% writes. *Virus Scan* represents host requests operated when scanning 600 MB of files for viruses, and nearly all the requests (99.5%) are reads. Finally, *File Write* contains host requests for writing 680 MB of files while without any read requests. In addition, before the

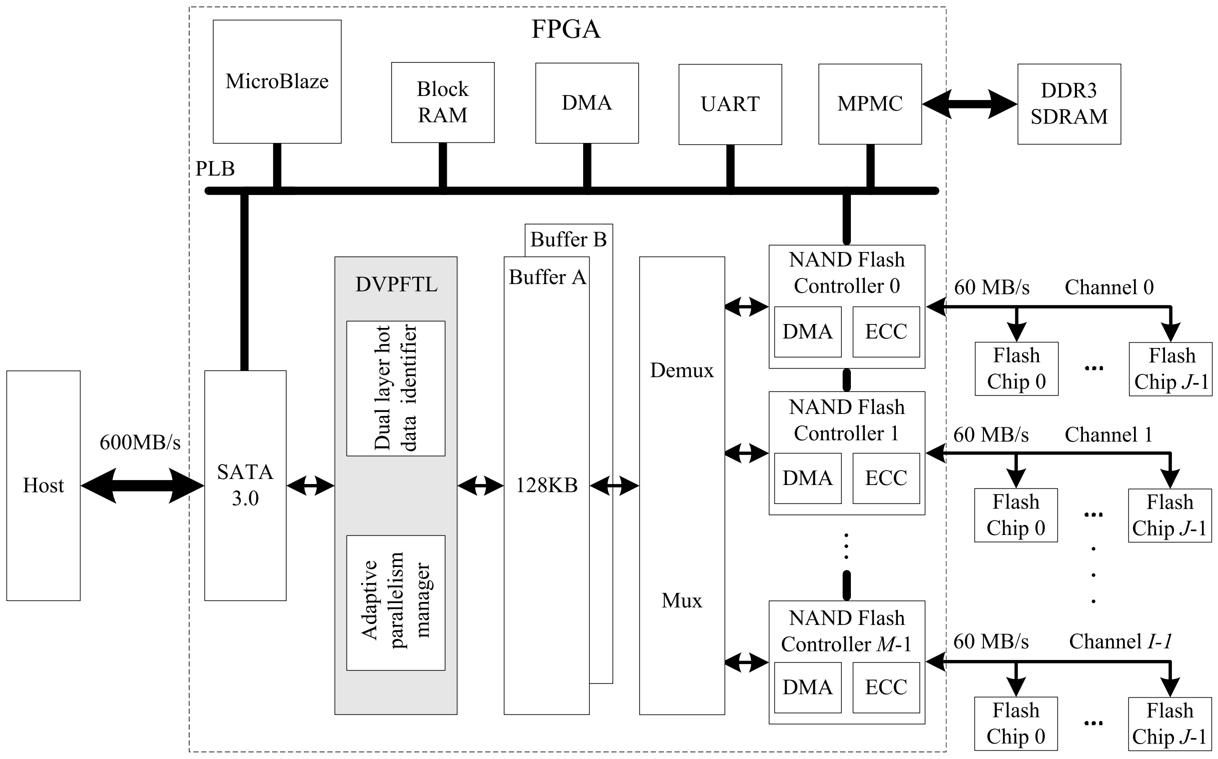


FIGURE 13. On-chip prototype of our DVPFTL-based SSD.

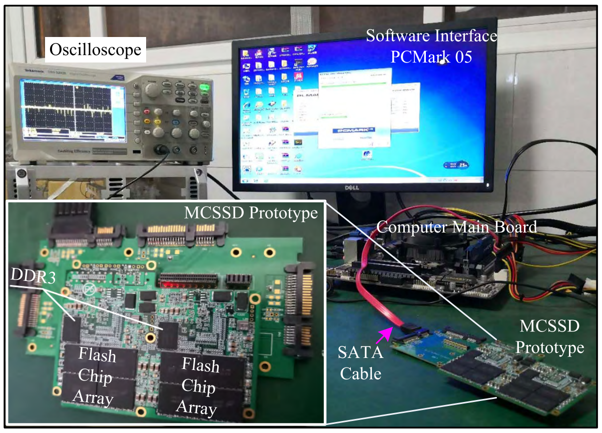


FIGURE 14. Prototype photo and testing scene. Note: FPGA and another 8 NFMCs are welded on the bottom layer.

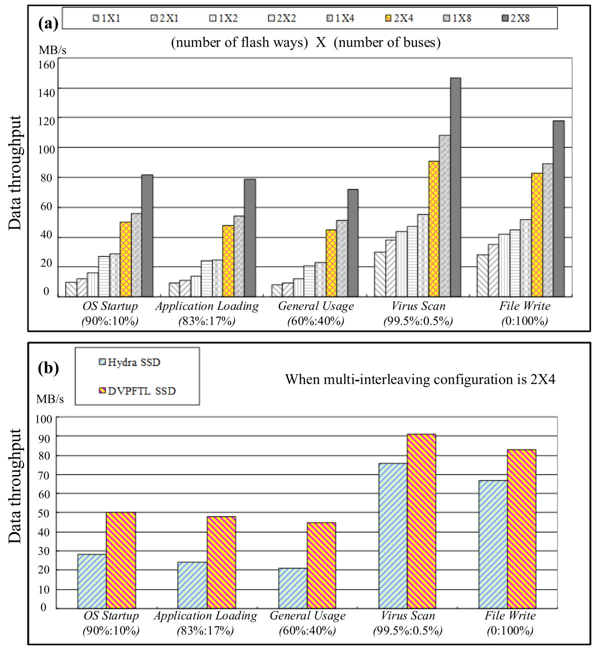


FIGURE 15. On-chip scores with various workloads (read % : write %).

upcoming tests, the DL-HDI is instantiated as the standard DL-MBF_s (refer to TABLE 1), and the two thresholds of Th_{dn} and Th_{up} are set to 100 and 200, respectively (refer to Section IV-A3).

3) OVERALL PERFORMANCE OF DVPFTL SSD

Fig. 15(a) shows the average access speeds of the DVPFTL SSD for various combinations of bus-level and chip-level interleaving. Once a certain interleaving configuration is determined, the performance of *Virus Scan*, involving the most read requests, ranks the first among these five components, followed by that of *File Write* as a result of the few erase and copy back operations. And the average access speed (average value of the five components) of 2-way-8-channel architecture is promoted from the 19.08 MB/s of 1-way-1-channel architecture to 99.80 MB/s. As for the first three components, the improvements on access speeds are suppressed in varying degrees, mainly because frequent write requests trigger more merge and copy operations, and then GC overheads increase. Nevertheless, the growth trend of access speed is reliably promoted as the interleaving-level increases.

To be fair, Hydra [23] is chosen for comparison since it is a typical and state-of-the-art MCSSD architecture. With the identical 2-way-4-channel architecture, Fig. 15(b) shows the contrastive results between Hydra SSD and DVPFTL SSD. Intuitively, except for the slightly unfair advantage of flash bus frequency (40 MHz for Hydra SSD in [23], 60 MHz for our prototype), our DVPFTL has achieved considerable improvements compared with that of Hydra. Notably, the performance ascensions of the former three components are very significant, which prove that it is beneficial to handle discrete and frequent accesses by making the best of temporal and spatial localities. Furthermore, since our DL-MBF scheme is generalized from MBF [30] which plays a key role in the state-of-the-art CFTL [40], [41], the integration of CFTL on our test-bench is in progress, hoping to observe further optimization ideas for DVPFTL. Preliminary tests indicate that, in most cases, our log-structured block level mapping strategy of *"DVP+DL-MBF_s+APM"* performs slightly better than the fundamentally page level mapping strategy of *"Convertible Scheme+MBF+Dual-Caching"*. However, DVPFTL remains to be improved especially on handling workloads interlaced with random reads and writes (i.e., *General Usage*).

V. CONCLUSION

Herein, we have proposed a DVPFTL framework based on dynamic virtual page to improve the throughput and lifetime of SSDs, which combines the following two aspects to achieve a considerable overall performance.

- ***Accurate request identification***: Inspired by a basic idea that hot data always tend to small probability in real-world workloads, we develop a generalized dual layer HDI (DL-HDI) scheme to early discard massive cold items, so that it can reduce runtime overheads as well as a waste of memory spaces. Performance tests of our two DL-HDI cases (DL-MBF_s and DL-MBF_c) prove our proposed scheme achieves considerably lower false identification rate while consumes less resource and runtime consumptions, compared with some recent state-of-the-art HDI schemes.
- ***Effective request allocation***: Apart from the conventional multi-interleaving (external parallelism) measures, we design an adaptive parallelism manager (APM) to aggressively exploit internal parallelism under the proposed DVP structure. The APM is able to effectively issue two-plane program operations while trigger few dummy pages.

Finally, we implement a DVPFTL-based SSD prototype on FPGA for a general test bench. The preliminary results have verified the effectiveness and parallelizability of our *"DVP+DL-MBF+APM"* strategy. In essence, DVPFTL employs classical log-structured block level mapping and emphasizes write accesses more than read accesses. Although it achieves considerable performance when handling write intensive accesses (i.e., *MillSSD*), there still remains much space to improve its versatility to adapt to more variable access behaviors. Future work will concentrate on developing more efficient address mapping mechanism, with the hope of achieving a better flash access performance under more cost-effective overheads.

ACKNOWLEDGMENT

The authors would like to express their gratitude to Dr. Qiyou Xie, a senior engineer from RunCore Co., Ltd., who assisted in verifying the performance of the prototype in this paper.

REFERENCES

[1] Y. Ryu, "A flash translation layer for NAND flash-based multimedia storage devices," *IEEE Trans. Multimedia*, vol. 13, no. 3, pp. 563–572, Jun. 2011.
[2] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen, "FlexLevel NAND flash storage system design to reduce LDPC latency," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 7, pp. 1167–1180, Jul. 2017.
[3] M.-L. Chiao and D.-W. Chang, "ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 753–766, Jun. 2011.
[4] C.-C. Ho, Y.-P. Liu, Y.-H. Chang, and T.-W. Kuo, "Antiwear leveling design for SSDs with hybrid ECC capability," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 488–501, Feb. 2017.
[5] C.-H. Wu, D.-Y. Wu, H.-M. Chou, and C.-A. Cheng, "Rethink the design of flash translation layers in a component-based view," *IEEE Access*, vol. 5, pp. 12895–12912, 2017.
[6] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang, "Efficient identification of hot data for flash memory storage systems," *ACM Trans. Storage*, vol. 2, no. 1, pp. 22–40, Feb. 2006.
[7] M. Lin, S. Chen, G. Wang, and T. Wu, "HDC: An adaptive buffer replacement algorithm for NAND flash memory-based databases," *Optik Int. J. Light Electron Opt.*, vol. 125, no. 3, pp. 1167–1173, Feb. 2014.
[8] J. Hu, M. Xie, C. Pan, C. J. Xue, Q. Zhuge, and E. H.-M. Sha, "Low overhead software wear leveling for hybrid PCM + DRAM main memory on embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 4, pp. 654–663, Apr. 2015.
[9] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 53–65, Jan. 2010.
[10] S. K. Park, Y. Park, G. Shim, and K. H. Park, "CAVE: Channel-aware buffer management scheme for solid state disk," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2011, pp. 346–353.
[11] S.-Y. Park, E. Seo, J.-Y. Shin, S. Maeng, and J. Lee, "Exploiting internal parallelism of flash-based SSDs," *IEEE Comput. Archit. Lett.*, vol. 9, no. 1, pp. 9–12, Jan. 2010.
[12] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1141–1155, Jun. 2013.
[13] *Micron*. Accessed: Jul. 2018. [Online]. Available: https://www.micron.com/products/nand-flash/slc-nand
[14] Y. Wang *et al.*, "A real-time flash translation layer for NAND flash memory storage systems," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 1, pp. 17–29, Jan./Mar. 2016.
[15] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 229–240, Mar. 2009.
[16] Z. Xu, R. Li, and C.-Z. Xu, "CAST: A page-level FTL with compact address mapping and parallel data blocks," in *Proc. IEEE 31st Int. Perform. Comput. Commun. Conf. (IPCCC)*, Austin, TX, USA, Dec. 2012, pp. 142–151.
[17] S.-H. Park, S.-H. Ha, K. Bang, and E.-Y. Chung, "Design and analysis of flash translation layers for multi-channel NAND flash-based storage devices," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1392–1400, Aug. 2009.

[18] R. Chen, Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "On-demand block-level address mapping in large-scale NAND flash storage systems," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1729–1741, Jun. 2015.

[19] Y. Hu *et al.*, "Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–12.

[20] J. Kim, J. M. Kim, S. H. Noh, S. L. 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.

[21] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, Jul. 2007, Art. no. 18.

[22] D. Park, B. Debnath, and D. H. C. Du, "A workload-aware adaptive hybrid flash translation layer with an efficient caching strategy," in *Proc. IEEE 19th Annu. Int. Symp. Modelling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Jul. 2011, pp. 248–255.

[23] Y. J. Seong *et al.*, "Hydra: A block-mapped parallel flash memory solid-state disk architecture," *IEEE Trans. Comput.*, vol. 59, no. 7, pp. 905–921, Jul. 2010.

[24] D. Liu *et al.*, "Durable address translation in PCM-based flash storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 475–490, Feb. 2017.

[25] L.-P. Chang and T.-W. Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation," *ACM Trans. Storage*, vol. 1, no. 4, pp. 381–418, Nov. 2005.

[26] *DS150 (v2.3): Virtex-6 Family Overview*, Datasheet1, Xilinx, San Jose, CA, USA, Jan. 2012.

[27] S. Niemelä. PCMark 05 Whitepaper. Futuremark. Accessed: Feb. 2018. [Online]. Available: http://s3.amazonaws.com/download-aws.futuremark.com/pcmark05-whitepaper.pdf

[28] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *J. Syst. Archit.*, vol. 55, nos. 5–6, pp. 332–343, 2009.

[29] L.-P. Chang and T.-W. Kuo, "An adaptive striping architecture for flash memory storage systems of embedded systems," in *Proc. 8th IEEE Real-Time Embedded Technol. Appl. Symp.*, Sep. 2002, pp. 187–196.

[30] D. Park and D. H. C. Du, "Hot data identification for flash-based storage systems using multiple bloom filters," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, May 2011, pp. 1–11.

[31] D. Park, B. Debnath, Y. J. Nam, D. H.-C. Du, Y. Kim, and Y. Kim, "HotDataTrap: A sampling-based hot data identification scheme for flash memory," in *Proc. 27th Annu. ACM Symp. Appl. Comput. (SAC)*, Trento, Italy, Mar. 2012, pp. 1610–1617.

[32] J. Liu, S. Chen, T. Wu, and H. Zhang, "A novel hot data identification mechanism for NAND flash memory," *IEEE Trans. Consum. Electron.*, vol. 61, no. 4, pp. 463–469, Nov. 2015.

[33] Y. Li, B. Shen, Y. Pan, Y. Xu, Z. Li, and J. C. S. Lui, "Workload-aware elastic striping with hot data identification for SSD RAID arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 5, pp. 815–828, May 2017.

[34] A. Miranda and T. Cortes, "CRAID: Online RAID upgrades using dynamic hot data reorganization," in *Proc. 12th USENIX FAST*, Santa Clara, CA, USA, 2014, pp. 133–146.

[35] C. H. Wu, P. H. Wu, K. L. Chen, W. Y. Chang, and K. C. Lai, "A hotness filter of files for reliable non-volatile memory systems," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 375–386, Jul. 2015.

[36] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using bloom filters," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 397–409, Apr. 2006.

[37] UMASS. (2002). *OLTP Trace From UMass Trace Repository*. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[38] Microsoft. (2007). *SNIA IOTTA Repository: MSR Cambridge Block I/O Traces*. [Online]. Available: http://iotta.snia.org/traces/list/BlockIO

[39] Q. Luo and Y. He, "A cost-effective and automatic surface defect inspection system for hot-rolled flat steel," *Robot. Comput.-Integr. Manuf.*, vol. 38, pp. 16–30, Apr. 2016.

[40] D. Park, B. Debnath, and D. Du, "CFTL: A convertible flash translation layer adaptive to data access patterns," in *Proc. SIGMETRICS*, Jun. 2010, pp. 365–366.

[41] D. Park, B. Debnath, and D. H. C. Du, "A dynamic switching flash translation layer based on page-level mapping," *IEICE Trans. Inf. Syst.*, vol. E99-D, no. 6, pp. 1502–1511, Jun. 2016.

QIWU LUO (M'17) received the B.S. degree in communication engineering from the National University of Defense Technology, Changsha, China, in 2008, and the M.Sc. degree in electronic science and technology and the Ph.D. degree in electrical engineering from Hunan University, Changsha, in 2011 and 2016, respectively.

He was a Senior Engineer of instrumentation with Wasion Group Co., Ltd., Changsha, and a Deputy Technical Director with Hunan RAMON Technology Co., Ltd., Changsha. He is currently a Lecturer with the School of Electrical Engineering and Automation, Hefei University of Technology, Hefei, China. His research interests include the research of real-time information processing, automatic optical inspection, parallel hardware architecture design and reconfigurable computing, and machine learning.

RAY C. C. CHEUNG (M'05) received the B.Eng. degree in computer engineering and the M.Phil. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 1999 and 2001, respectively, and the Ph.D. and D.I.C. degrees in computing from Imperial College London, London, U.K., in 2007.

After completing the Ph.D. work, he received the Hong Kong Croucher Foundation Fellowship for his post-doctoral study at the Electrical Engineering Department, University of California at Los Angeles. In 2009, he was a Visiting Research Fellow with the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA. He is currently an Associate Professor with the Department of Electronic Engineering, City University of Hong Kong (CityU). He has authored over 100 journal and conference papers. His research team, CityU Architecture Lab for Arithmetic and Security, focuses on the research topics, such as reconfigurable trusted computing, applied cryptography, and high-performance biomedical VLSI designs.

YICHUANG SUN (M'90–SM'99) received the B.Sc. and M.Sc. degrees from Dalian Maritime University, Dalian, China, in 1982 and 1985, respectively, and the Ph.D. degree from the University of York, York, U.K., in 1996, all in communications and electronics engineering.

He is currently a Professor and the HoD with the School of Engineering and Technology, University of Hertfordshire, U.K. He has published over 300 papers and contributed 10 chapters in edited books. He has also published four text and research books: Continuous-time Active Filter Design (CRC Press, USA, 1999), Design of High frequency Integrated Analogue Filters (IEE Press, U.K., 2002), Wireless Communication Circuits and Systems (IET Press, 2004), and Test and Diagnosis of Analogue, Mixed-signal and RF Integrated Circuits—the Systems on Chip Approach (IET Press, 2008). His research interests are mainly in the areas of wireless and mobile communications and RF and analogue circuits.

He was a Series Editor of the IEE Circuits, Devices and Systems Book Series from 2003 to 2008. He has been Associate Editor of the IEEE Transactions on Circuits and Systems I: Regular Papers from 2010 to 2011, from 2016 to 2017, and from 2018 to 2019. He is also an Editor of the *ETRI Journal*, *Journal of Semiconductors*, and some others. He was a Guest Editor of eight IEEE and IEE/IET journal special issues: High-frequency Integrated Analogue Filters in IEE Proc. Circuits, Devices and Systems (2000), RF Circuits and Systems for Wireless Communications in IEE Proc. Circuits, Devices and Systems (2002), Analogue and Mixed-Signal Test for Systems on Chip in IEE Proc. Circuits, Devices and Systems (2004), MIMO Wireless and Mobile Communications in IEE Proc. Communications (2006), Advanced Signal Processing for Wireless and Mobile Communications in IET Signal Processing (2009), Cooperative Wireless and Mobile Communications in IET Communications (2013), Software-Defined Radio Transceivers and Circuits for 5G Wireless Communications in IEEE Transactions on Circuits and Systems-II (2016), and IEEE International Symposium on Circuits and Systems in IEEE Transactions on Circuits and Systems-I (2016). He has also been widely involved in various IEEE technical committee and international conference activities.