IEICE Electronics Express, Vol.16, No.21, 1–6

LETTER

An implementation of low latency address-mapping logic for SSD controllers

Yuchan Song[1], Hyunjoo So[1], Yongjae Chun[1], Hyun-Seok Kim[1], and Youpyo Hong[1a)]

Abstract Solid-state drives (SSDs) are replacing hard-disk drives (HDDs) because of their advantages of light weight, low power, and high speed. A flash translation layer (FTL) is a key to achieving a high efficiency in accessing an SSD. This letter presents an architecture to implement the mapping between the logical address and the physical address as hard-wired to reduce the workload of the FTL inside an SSD.
Keywords: SSD, FTL
Classification: Integrated circuits

1. Introduction

In the recent years, solid-state devices (SSDs) have become the main storage owing to their higher bandwidth and lower power consumption compared to hard disk drives (HDDs) [1, 2]. Flash translation layer (FTL) is an intermediate function in between host interface and flash memory controller to handle various tasks including address mapping [3, 4, 5]. That is, a file name is translated to a logical address (LA) in its file system, and a FTL [6, 7] maps the logical address to physical address (PA) in the flash memory in computer systems with an SSD. The basic read or write unit in a SSD is called a page, and the logical address of a file whose size is smaller than or equal to that of a page is mapped to a logical page number (LPN). If a file is bigger than a single page, multiple LPNs are allocated to a single file, and the information about the list of LPNs is stored by the FTL. Each LPN is mapped to a Physical Page Number (PPN), which specifies a physical location in the flash memories of the SSD.

Address-mapping schemes are classified as page-level mapping, block-level mapping, and hybrid mapping [8, 9]. In page-level mapping, a logical address is mapped to a physical address at a page granularity, which is the most straightforward scheme at the expense of large storage requirement. To reduce the space requirement, extensive research has been conducted on page-level address mapping algorithms such as DFTL [6], OAFTL [10], K-CPM [11], DAC [12], DVPFTL [13] and MVFTL [14]. Block-level mapping requires much less space for the mapping table but suffers from significant performance overhead [5, 8, 15]. In hybrid mapping, a logic block is mapped to a physical block using block level mapping algorithm, and the page level algorithm is used to locate a page within a block. Examples of hybrid mapping algorithms are FAST [5], Superblock [16], LAST [17] and MAST [18]. Most hybrid mapping algorithms suffer inefficient garbage collection which seriously degrades the read-write performance and increases the wear of the SSD [9].

Many algorithms have been proposed to efficiently perform address mapping exploiting various storage types. An SSD controller typically consists of three layers of memory hierarchy: SRAM cache, DRAM, and flash memory. The contents of the table are moved partially or fully from flash memory to faster memory before the access by the FTL to improve system performance. The original address-mapping tables are stored in the flash memory because the table size was huge and the information in the tables must be maintained even when the system is off, and most early SSD controllers used a two-level memory hierarchy with the SRAM cache and flash memory. Based on such a two-level memory hierarchy, a demand-based Flash Translation Layer (DFTL) [6] and CAST FTL [19] that selectively caches page-level address mappings to the SRAM cache were proposed. Unfortunately, the capacity of an SRAM is inherently limited because of its size, power, and cost. In modern SSD systems, DRAMs have been actively used to store the address-mapping table, because they have shorter latency than does a NAND flash memory, and bigger capacity than that of a SRAM. One of the FTLs using DRAM is HP-FTL [20]. Their main focus is to reduce the table size by encoding PPN, which allows more entries to be stored in DRAM instead of flash memory. Details on the table referencing method, however, is not described in their work.

Another major trend in research on SSDs is to exploit the spatial locality of the workloads. For example, S-FTL [21] exploits spatial locality for an intelligent caching strategy, and ZFTL [22] grouped the entire flash space into multiple zones and cached only the information for a zone based on locality analysis to DRAM. CDFTL [23] is an on-demand page-level mapping FTL similar to DFTL. All these approaches are too sensitive to file access patterns and their performance degrade significantly if the workloads show weak locality. Many practical cases such as massive databases or data processing applications, unfortunately, show random access patterns [24]. Therefore, few researches such as [25] and [26] transform random write requests to sequential ones in hardware and virtualization layer, respectively, to improve the random write latency based on page-level mapping.

[1]Dept. of Electrical and Electronics Engineering, Dongguk University, 26 3-Ga, Pil-Dong, Jung-Gu, Seoul, Rep. of Korea
a) yhong@dgu.edu

DOI: 10.1587/elex.16.20190521
Received August 16, 2019
Accepted September 9, 2019
Publicized October 15, 2019
Copyedited November 12, 2019

The goal of this paper is to implement the mapping function between LA and PPN optimized for short latency in DRAM at cycle level. As summarized before, there have been extensive research on algorithms and logics to efficiently perform address mapping, but no work has addressed cycle-level optimization for DRAM-based L2P table lookup. A key of the proposed architecture is to interleave multiple banks of DRAM with a dynamic scheduling algorithm to access DRAM.

2. Proposed address mapping algorithm

2.1 Overall configuration of address mapping tables

In our approach, the address-mapping information is stored in two tables, the LA-PN Mapping Table (LPMT) and the Block Table (BT). The LPMT stores entries consisting of a logical address and the first page number of a file. LPMT, again, consists of two tables; a primary LPMT and a secondary LPMT as shown in Fig. 1. Each LPMT is configured in a set-associative way, and each set in LPMT is associated with a hash-key value. That is, a hash-key value is computed for an LA, and the information on the LA-PN pair is stored in the set associated with the hash-key value. If the space for the set with a hash key is full, then additional space for another set in a secondary table is allocated for the same hash key, and the location of the additional space is registered in the set of the primary table. That is, the storage in the secondary table is used by the sets that need extra spaces to accommodate exceptionally many LA-PN entries. The maximum capacity of the LPMT is twice the size of the total of LA-PN entries, because the worst case is that all LA-PN entries are associated with only one hash key.

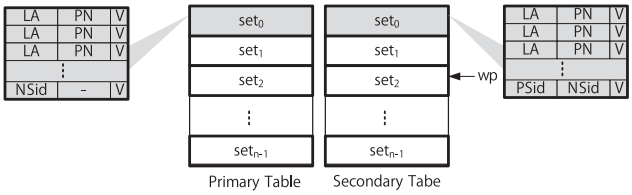


Fig. 1. LPMT configuration (NSid: ID of Next Set with the same hash key value, V: Valid, PSid: ID of Previous Set with the same hash key value)

Each entry in LPMT consists of an LA-PN pair and two other flag bits that indicate if the entry is valid and if the LA is a single-page file. The latter flag is used to avoid the unnecessary access by BT looking up the list of PNs for the LA. If a file is not bigger than a single page, FTL can access the physical location of the file simply by referring to the LA-PN information in LPMT. Otherwise, FTL needs to look up BT to obtain the list of PNs for the file.

BT stores the pages pertaining to a particular file in a doubly linked list fashion. BT stores the information on a list of PNs associated with a file using a linked-list type of PN information to enable read, modify, and delete. For each PN, PPid and NPid denotes previous page id and next-page id sharing the same hash key value, respectively. In addition to the valid flag, two extra flag bits are appended to indicate if the PN is either a first page or a last page. Each block of flash memory has its own BT. A physical address consists of a block number and a page number. For example, a file with the logical address 23d1 is mapped to two pages: a physical address 01 000 for the first page, 10 000 and 11 000 for the second and third pages. Note that the three pages are virtually connected, as shown in the BTs in Fig. 2 using the doubly linked list fashion. Note that PPid of the first page for a file indicates the location of the entry in a LPMT. For example, PPid 0 01 00 of the first entry in BT00 indicates the LA-PN pair information is in the primary LPMT by the first 0, in the set01 by the following 01, and it is the first entry in the LPMT by the last 00.

Note that, to maintain uniformity, even if a single page is assigned to a file, the corresponding entry in BT is marked as the first page, last page, and valid page. PPid, in this case, points to the location of the corresponding entry in LPMT.

The access time to LPMT is not predictable, because it involves the hash function to locate the target set and the secondary LPMT must be looked up in the worst case. Another important factor in determining the access time to LPMT is that the table resides in a DRAM whose access time is determined various factors. Therefore, the reset of the paper is focused on the scheme to improve the access time to LPMT.

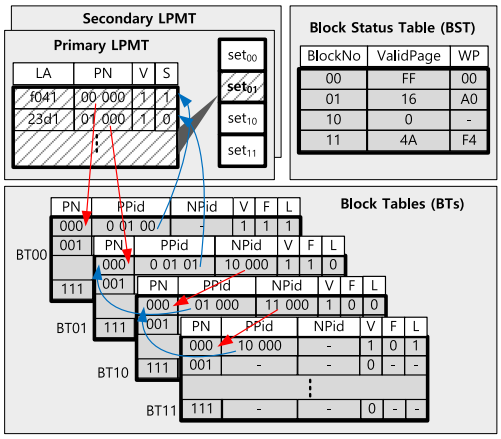


Fig. 2. Complete address mapping tables configuration (V: Valid, S: Single-page, WP: Write Pointer, PPid: Previous Page ID, NPid: Next Page ID, F: First-page, L: Last-page)

2.2 DRAM configuration for LPMT

Most DRAMs consist of multiple banks of memory cells, row and column decoders, and a sense amplifier. For read operations, the basic three steps are:

(1) a row-activation command is applied, and the row data from a bank are copied to the sense amplifier;
(2) a read command with a column address is applied, and the corresponding data are sent to the output bus; and
(3) the sense amplifier is reset by a precharge command to prepare for the next access.

Once a row-active strobe (RAS) signal becomes active, the DRAM is in an active row status and the inputted row address is decoded. The time taken to decode the row address is the RAS-to-CAS delay (RCD). After the delay,

the column-active strobe (CAS) and the column address are applied simultaneously. The delay from the CAS command to the valid data output is the CAS latency. The following figure shows the case where the same bank is accessed for two different rows, with the worst-case latency between the access of the last data from the previous transaction and the access of the first data from the next transaction, which is 18 cycles in Fig. 3.

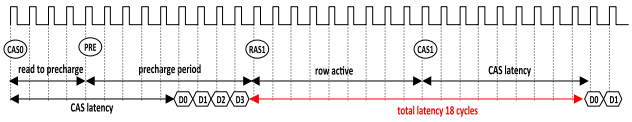


Fig. 3. Latency without bank interleaving

If two consecutive transactions access addresses in different rows, a precharge operation must be executed before the execution of the row-activation command for the second transaction. Commands to different banks can be overlapped to save the cycles needed to set up a bank or a row. Because RAS and CAS commands can be applied to different banks independently, commands for multiple transactions can be overlapped, which is called bank interleaving as shown in Fig. 4.

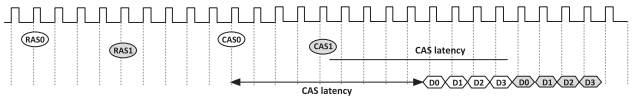


Fig. 4. Latency with bank interleaving

The exact latency of DRAM access with multiple banks is determined by various factors, such as bank and rows of previous DRAM accesses. Fig. 5 shows a case where previous DRAM access affects the latency of the last DRAM access for a commercial four-banks DRAM [27]. Suppose that three different banks—denoted bank [t-1], bank [t-2] and bank [t-3]—are accessed consecutively, but the last accessed bank, bank [t], is the same as bank [t-3]; i.e., bank 3, bank 1, bank 2, and bank 3 are accessed in series. In such case, Fig. 5 shows that it is possible to issue row active before the data transfer is finished. So CAS1 and CAS2 come out immediately after CAS0 comes out. However, since RAS3 uses the same bank as the first bank, it can be row active after the data corresponding to the first bank is transferred. As a result, the data for the third bank transfer continuously, but the data for the fourth bank can transfer after 15 cycles, which number was derived as follows.

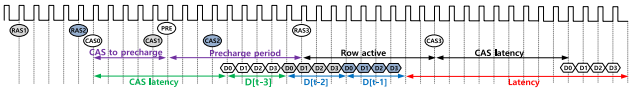


Fig. 5. Read latency when bank[t] = bank[t-3], r[t] ≠ r[t-3]

Table I shows the DRAM parameters used and Fig. 6 shows the latencies for various bank combinations when different rows are accessed.

In order to minimize table-lookup latency while maximizing throughput, we propose two schemes as follows. First, we distribute each set over the four banks in the DRAM in order to use bank interleaving. Second, the access order of a set over the four banks is dynamically rearranged so that the bank interleaving effect is maximized. For example, if bank 0 was recently accessed, then the banks other than bank 0 better be accessed in order to increase the probability of bank interleaving. Fig. 7 shows the algorithm to record the order of banks recently accessed. That is, $access_{t-3}$, $access_{t-2}$, $access_{t-1}$ and $access_t$ stores the bank numbers.

Table I. DRAM parameters

Parameter	Cycles
ROW active delay (RCD)	9
CAS latency (CL)	9
CAS to Precharge time (RTP)	5
CAS to CAS delay (CCD)	4
Precharge period (RP)	9
Data latency (DL)	4

$$\begin{cases} RTP + RP + RCD + CL + DL - (CL[t-1] + DL[t-1]), & if\ b[t] = b[t-1] \\ RTP + RP + RCD + CL + DL - (CL[t-2] + \sum_{n=1}^{2} DL[t-n]), & else\ if\ b[t] = b[t-2] \\ RTP + RP + RCD + CL + DL - (CL[t-3] + \sum_{n=1}^{3} DL[t-n]), & else\ if\ b[t] = b[t-3] \\ RTP + RP + RCD + CL + DL - (CL[t-4] + \sum_{n=1}^{4} DL[t-n]), & else\ if\ b[t] = b[t-4] \\ DL & else \end{cases}$$

Fig. 6. Read latency for $r[t] \neq r[t\text{-}n]$ where $n = \min\{1, 2, 3, 4\}$ such that $b[t] = b[t\text{-}n]$

```
for each ACT command
    if bank_number_curr != access_t
        if bank_number_curr = access_t-1
            then access_t ← access_t-1,
                 access_t-1 ← access_t
        else if bank_number_curr = access_t-2
            then access_t ← access_t-2,
                 access_t-1 ← access_t,
                 access_t-2 ← access_t-1
        else access_t ← access_t-3,
             access_t-1 ← access_t,
             access_t-2 ← access_t-1,
             access_t-3 ← access_t-2
```

Fig. 7. Bank reordering algorithm

Then, when a new set is to be accessed, the four banks are accessed in the order of $access_{t-3}$, $access_{t-2}$, $access_{t-1}$ and $access_t$.

3. Results and discussion

We modelled the proposed algorithm using C code with parameters from a commercial DRAM [27]. In our implementation, the tables are sized to accommodate the workload financial1 that includes the largest number of files. For example, financial1 includes 700,873 files; so the number of entries in LPMT is set to 2^{20}, each set contains eight entries, and there are 2^{17} sets in LPMT.

We assumed that a page size is 2 KB, one block contains 256 pages, and the SSD contains 4,096 blocks ($256 * 4{,}096 = 1{,}048{,}576 = 2^{20}$ pages) which amounts 2 GB in total capacity. Notice that the 2 GB SSD can hold financial1 as $710{,}875 < 2^{20}$ (= 1,048,576), assuming all file sizes are under 2 KB. Table II show the various dimensions of LPMT.

Table II. LPMT dimensions

Parameter	Size	Comment
Row numbers	2^{20}	256 (2^8) Pages per Block
Row width	42 bits	20(LA) + 20(PA) + 2(VS)
LA width	20 bits	
PA width	20 bits	Block Table ID(12 bits) + # BT Rows (8)

We first evaluated the utilization of the primary LPMT table using four hashing functions: the shifting method [28], the division method [29], the folding method [29], and the radix transformation method [30]. Table III shows the characteristics of the SPC workload and Table IV shows the occupancy rates of Primary LPT for various hashing functions. Three hash-key functions combined with the proposed algorithm led the primary LPMT table to be occupied more than 90%, except for Websearch 2 and Websearch 3, which have a monotonic logical addresses pattern, i.e., the addresses in the two workloads are all multiples of four.

Table III. Workload characteristics

Workload	Req. Size (KB)	Read (%)	Sequential Read (%)	Sequential Write (%)
Financial 1	3.38	23.16	1.61	0.01
Financial 2	2.39	82.35	0.94	0.22
MSR	29.812	34	0	0
Websearch 1	15.15	99.98	0	0
Websearch 2	15.07	99.98	0	0
Websearch 3	15.41	99.97	0	0

Table IV. Primary LPMT occupancy statistics

Workload	Shifting (%)	Folding (%)	Radix (%)	Division (%)
Financial 1	99.13	78.23	99.05	98.89
Financial 2	92.45	72.01	92.39	92.05
MSR	94.09	79.23	96.98	20.52
Websearch 1	100	100	100	78.77
Websearch 2	71.92	71.92	100	67.78
Websearch 3	71.37	70.64	100	61.30

Table V shows latencies for various combinations of DRAM access in terms of banks and row address changes for LPMT lookup. For example, b[t] = b[t-1] indicates that a subsequent dram access occurred to the same banks. Notice that our hash-key assignment policy led to a distribution that is more favorable than when bank interleaving is not considered.

Table V. DDR read access latencies statistics

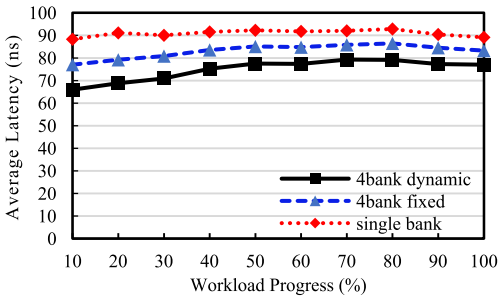
Row	r[t] ≠ r[t-n] where $n = \min\{1,2,3,4\}$ such that $b[t] = b[t-n]$					r[t] = r[t-n]
Bank		b[t] ≠ b[t-1]				
			b[t] ≠ b[t-2]			
				b[t] ≠ b[t-3]		
	b[t] = b[t-1]	b[t] = b[t-2]	b[t] = b[t-3]	b[t] = b[t-4]	b[t] ≠ b[t-4]	b[t] = b[t-n]
Latency	23	$23-\sum_{n=1}^{1} DL_{[t-n]}$	$23-\sum_{n=1}^{2} DL_{[t-n]}$	$23-\sum_{n=1}^{3} DL_{[t-n]}$	4	4
Single Bank (%)	8.86	0	0	0	4.84	84.31
4 Banks fixed order (%)	4.28	0	0	0	8.53	87.18
4 Banks dynamic order (%)	0	0	0	8.68	3.83	87.49

Table VI shows the average latencies for three addressing algorithms: only one bank is used, each set is mapped to one of the four banks in turn, and the proposed dynamic re-ordering algorithm. Among the four hashing functions, the folding function with the worst performance in terms of set-distribution is used. The four-bank fixed-order algorithm shows a speed gain solely because of using bank interleaving, which amounts to 11.91% on average, and the proposed algorithm shows an additional speed gain of 4.37% on average.

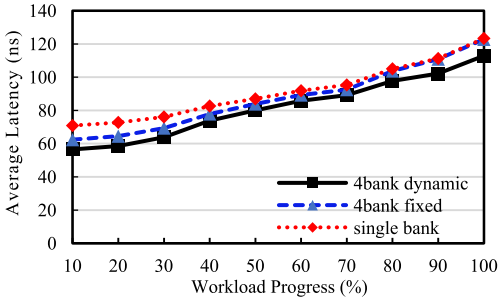
Table VI. Average latency (ns) & average improvement ratio over single bank (%)

	Single Bank	4 Banks Fixed Order	4 Banks Dynamic Order
Financial1	90.95	83.07	74.87
Financial2	91.59	87.72	82.05
MSR	98.62	89.56	86.84
Websearch1	69.47	57.20	55.16
Websearch2	71.14	59.71	57.80
Websearch3	70.64	59.58	57.73
Δ (%)	-	11.91	16.28

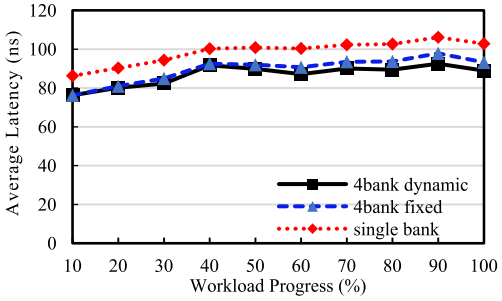
Fig. 8 shows the average latencies for each tenth of the progress of a workload. In general, the latencies increase as more files are accumulated in the flash memory because it takes longer to look up the table as address-mapping tables are filled up. Financial2 and MSR workloads show that the improvement factors of the proposed algorithm increase as the table fills up.



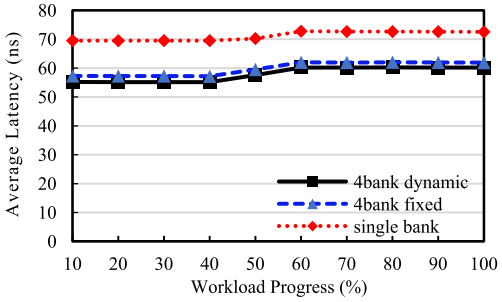
(a) Financial1



(b) Financial2



(c) MSR



(d) Websearch2

Fig. 8. Latency trends over workload progress

4. Conclusion

We presented a hardware-based address-mapping logic in this paper. A bank-interleaving scheme is applied to reduce the access time for the address-mapping table stored in DRAM. Our main contribution is to maximally utilize bank interleaving by dynamically rearranging bank access orders. This scheme is very effective when DRAM access pattern is irregular, as is the case for address mapping logic for SSD controllers.

Acknowledgments

This research was supported by 'Research on NOC application for power-efficient backbone in SDC' funded by Samsung Electronics and the Dongguk University Research Fund of 2018.

References

[1] N. Agrawal, *et al.*: "Design tradeoffs for SSD performance," USENIX Annual Technical Conference (ATC) (2008) 57.
[2] C. Dirik and B. Jacob: "The performance of PC solid state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," ACM/IEEE International Symposium on Computer Architecture (ISCA) (2009) 279 (DOI: 10.1145/1555815.1555790).
[3] R. Micheloni, *et al.*: *Inside Solid State Drives (SSDs)* (Springer Science & Business Media, New York, 2012).
[4] J.-U. Kang, *et al.*: "A superblock-based flash translation layer for NAND flash memory," ACM International Conference on Embedded Software (EMSOFT) (2006) 161 (DOI: 10.1145/1176887.1176911).
[5] S.-W. Lee, *et al.*: "A log buffer-based flash translation layer using fully-associative sector translation," ACM Trans. Embed. Comput. Syst. **6** (2007) 18 (DOI: 10.1145/1275986.1275990).
[6] A. Gupta, *et al.*: "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2009) 229 (DOI: 10.1145/1508244.1508271).
[7] J. Kim, *et al.*: "A space-efficient flash translation layer for compact flash systems," IEEE Trans. Consum. Electron. **48** (2002) 366 (DOI: 10.1109/TCE.2002.1010143).
[8] T.-S. Chung, *et al.*: "System software for flash memory: A survey," International Conference on Embedded and Ubiquitous Computing (EUC) (2006) 394 (DOI: 10.1007/11802167_41).
[9] Y. Jin and B. Lee: in *Advances in Computers*, vol. 114, ed. A. R. Hurson (Academic Press, Cambridge, Massachusetts, 2019) 1 (DOI: 10.1016/bs.adcom.2019.02.001).
[10] X. Y. Qi, *et al.*: "OAFTL: An efficient flash translation layer for enterprise application," J. Comput. Res. Develop. **48** (2011) 1918.
[11] H. Kim and D. Shin: "Clustered page-level mapping for flash memory-based storage devices," IEEE Trans. Consum. Electron. **61** (2015) 47 (DOI: 10.1109/TCE.2015.7064110).
[12] R. Chen, *et al.*: "On-demand block-level address mapping in large-scale NAND flash storage systems," IEEE Trans. Comput. **64** (2015) 1729 (DOI: 10.1109/TC.2014.2329680).
[13] Q. Luo, *et al.*: "Dynamic virtual page-based flash translation layer with novel hot data identification and adaptive parallelism management," IEEE Access **6** (2018) 56200 (DOI: 10.1109/ACCESS.2018.2872721).
[14] D. Lee, *et al.*: "MV-FTL: An FTL that provides page-level multi-version management," IEEE Trans. Knowl. Data Eng. **30** (2018) 87 (DOI: 10.1109/TKDE.2017.2757016).
[15] T. Shinohara: U.S. Patent 5905993A (1999).
[16] D. Jung, *et al.*: "Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme," ACM Trans. Embed. Comput. Syst. **9** (2010) 1 (DOI: 10.1145/1721695.1721706).
[17] S. Lee, *et al.*: "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," ACM SIGOPS Oper. Syst. Rev. **42** (2008) 36 (DOI: 10.1145/1453775.1453783).
[18] G. Shim, *et al.*: "A hybrid flash translation layer with adaptive merge for SSDs," ACM Trans. Storage (TOS) **6** (2011) 15 (DOI: 10.1145/1970338.1970339).
[19] Z. Xu, *et al.*: "CAST: A page-level FTL with compact address mapping and parallel data blocks," IEEE International Performance, Computing and Communications Conference (IPCCC) (2012) 142 (DOI: 10.1109/PCCC.2012.6407747).
[20] F. Ni, *et al.*: "A hash-based space-efficient page-level FTL for large-capacity SSDs," International Conference on Networking,

Architecture and Storage (NAS) (2017) (DOI: 10.1109/NAS.2017.8026838).
[21] S. Jiang, *et al.*: "S-FTL: An efficient address translation for flash memory by exploiting spatial locality," IEEE Conference on Mass Storage Systems and Technologies (MSST) (2011) 1 (DOI: 10.1109/MSST.2011.5937215).
[22] M. Wang, *et al.*: "ZFTL: A zone-based flash translation layer with a two-tier selective caching mechanism," IEEE International Conference on Communication Technology (ICCT) (2011) 578 (DOI: 10.1109/ICCT.2012.6511426).
[23] Z. Qin, *et al.*: "A two-level caching mechanism for demand-based page-level address mapping in NAND flash memory storage systems," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2011) 157 (DOI: 10.1109/RTAS.2011.23).
[24] T. Harter, *et al.*: "A file is not a file: Understanding the i/o behavior of apple desktop applications," ACM Trans. Comput. Syst. **30** (2012) 10 (DOI: 10.1145/2324876.2324878).
[25] H. Kim, *et al.*: "SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device," USENIX Conference on File and Storage Technologies (FAST) (2017) 271.
[26] S. Kim, *et al.*: "Improving spatial locality in virtual machine for flash storage," IEEE Access **7** (2018) 1668 (DOI: 10.1109/ACCESS.2018.2886473).
[27] Micron DDR4 SDRAM Datasheet: MT40A512M8: https://www.micron.com/products/dram/ddr4-sdram.
[28] X. Gou, *et al.*: "Single hash: Use one hash function to build faster hash based data structures," IEEE International Conference on Big Data and Smart Computing (BigComp) (2018) 280 (DOI: 10.1109/BigComp.2018.00048).
[29] M. Singh and D. Garg: "Choosing best hashing strategies and hash functions," IEEE International Advance Computing Conference (IACC) (2009) 51 (DOI: 10.1109/IADCC.2009.4808979).
[30] A. D. Lin: "Key addressing of random access memories by radix transformation," AFIPS Spring Joint Computer Conference (SJCC) (1963) 355 (DOI: 10.1145/1461551.1461594).