LETTER

Error source and latency-aware read performance optimization scheme for aged SSDs*

Shiqiang Nie¹, Chi Zhang¹, Chen Zhang¹, Xuda Zheng¹, and Weiguo Wu^{1, a)}

Abstract LDPC code has been used widely in NAND flash-based storage system due to its high error correction capacity, prolonging the lifetime of multi-bit NAND flash. However, the LDPC decoding latency degrades the read performance of SSD as it induces more read-retry operations. The last RL (Read-Level) recording method has been proposed in recent research works, which achieves better performance improvement by reducing many useless fail reads. However, these schemes reset the RL of these pages to be 1 after these blocks are erased. Using RL 1 to read these pages may induce many fail reads at first read on each page. That because it ignores the different error source issues, i.e., a part of the page error comes from the P/E cycles, while others come from retention time and other sources. Motivated by this observation, in this paper, we propose two schemes to optimize the read procedure of NAND flash-based SSD, especially for aged SSDs. We propose to record RL induced by different error sources separately, so the RL of the page could keep unchanged rather than 1 after the blocks are erased. The scheme could reduce useless fail read after the blocks are read at first time. We also design a latency aware I/O scheduler to reorder the input read requests in batch by prioritizing requests with low latency to reduce the queue latency. Our experiments show that the proposed scheme can reduce the average response time by up to 33% with less storage overhead.

Keywords: NAND flash, read retries, LDPC, retention time, I/O scheduler

Classification: Circuits and modules for storage

1. Introduction

SSDs have been used widely in storage system, such as cloud storage, mobile devices. To reduce the per-bit cost, many techniques, such as shrinking the cell size, storing more than one bit in one cell, and 3D stack has been proposed [1, 2]. These techniques achieve the lower cost with sacrificing of the reliability. The TLC SSD and QLC SSD have only thousands of P/E cycles compared with the 10k of P/E cycles in SLC SSD [3, 4]. To prolong the lifetime of multi-bit SSDs, the LDPC code has been used widely, which close to the Shannon limit [5, 6]. However, the LDPC demands up to 7 times read-retries operation to fully exploit its error correction capability. The larger read-retry operation contains more fine-grain read voltage. To successfully decode, the data

from the aged page may cost about 10 times read latency compared with the junior page. We refer to the number of read-retry operation as RL (Read-Level) below.

Many research works have devoted to the optimization of large read latency in the aged SSD [7, 8, 9]. As the NAND flash adjusts the read voltage to sense the electrons in the cell incrementally, these methods proposed to keep track of the last successful RL for the future reads at different granularity (i.e., page-level, block-level, and layer-level). However, as the P/E cycles, the retention time, the program interference, and the read disturb all contribute to RBER (Raw Bit Error Rate), using one counter to record the last RL maybe not very efficiently. Because the recorded RLs are effective on the future reads only in a limited period, after a long time, the RBER increases due to the leakage of electrons over time. However, the FTL still use the outdated RL to read out the data, and then it fails. More RL needs to be induced to sense the cells, which leads to more read latency. Besides, once the block has been erased, the proposed methods need to reset recording RL to be one, which also causes performance degradation. The issues mentioned above are mainly caused by ignoring the different error sources.

In the preliminary experiment, we observe that the existing RL recording methods can not work efficiently as a result of the increasing RBER caused by different error sources. That results motivate us to design two schemes to mitigate the extra read latency overhead. The main contributions of this paper are as follows:

- we have done an experiment to illustrate the read performance degradation issue induced by different error source. The RBER varies with P/E cycles and retention time and therefore induces the existing RL recording methods working inefficiently.
- we propose error source aware RL recording method. This method records the RL induced by different error sources, calculating the fittest RL for the future reads, which avoid the unnecessary read-retry operation.
- we propose the latency aware I/O scheduler to optimize the queue latency induced by the long requests by prioritizing these requests with less response time in batch.
- we evaluate the proposed schemes with different workloads and shows that it achieves better performance improvement.

In the rest of the paper, Section II discusses the NAND flash background and motivates our design. Section III presents the detailed scheme. Section IV describes the experiment

¹ Dept. of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China

^{a)} wgu@xjtu.edu.cn

* This work is supported in part by the National Key Research and Development Program of China under Grant 2017YFB1001701, in part by the National Science Foundation of China under Grant 61972311.

methodology and analyzes the results. Section V concludes the paper.

2. Background and motivation

2.1 The NAND flash error source

The NAND flash cell stores the data by injecting different amount of electrons inside the floating gate or charge trap [1, 10]. The electron leakage induces to the error by several factors, i.e., P/E cycles, retention time, read disturb, and program interference. The P/E cycling error comes from the weaken the floating gate or charge trap. The intrinsic physical feature of NAND flash restricts page to be erased before rewriting. The cell becomes error-prone after it suffers thousands of erase operation as a result of a high erase voltage performed on the cell to remove all the charges. The retention error comes from the electron leakage over retention time, and also it increases with increasing retention time. Program interference is the result of coupling effect that programming on one flash cell can weak-program its neighbouring cells. Read disturb is caused by lots of read operation because read operation injects few electrons into the cell.

2.2 LDPC-based read

This section introduces the process of read procedure. After the FTL receives the request delivered from the host, FTL splits the request into page-sized sub-request and finds its physical addresses, respectively. The FTL takes the first read voltage to sense the cell and then sensed data are transferred into the LDPC decoder. If the LDPC decoder succeeds in decoding the sensed data, the FTL return the decoded data to host, this phase is called by hard-decision decoding. Otherwise, the FTL uses more read voltages to sense the cell; the operation would cost more time and transfer more data to the LDPC decoder, the sensed data are decoded by LDPC decoder with the help of log-likelihood ratio (LLR) table [11]. The FTL will try up to seven-time to decode data. This phase is called by soft-decision decode. The soft-decision decoding achieves better error correct capacity over the hard-decision decoding; it could correct a very high error rate (e.g., 0.01 for LDPC of 512 B per 4 KB data). However, it leads to more flash sensing and data transferring latency.

2.3 Motivation

Neither planner NAND flash or 3D NAND flash are vulnerable by different error sources (i.e. m P/E cycles error, retention time error, read disturb and program interference). To study the impact of different error sources on RBER, Many research works have been done to explore the error characterize of NAND flash. The main error source, i.e., both P/E cycles and retention time, have been validated in planner NAND flash [12, 13, 14]. Wu et al. measured the RBER variation before and after performing these four operations, and they also found that the P/E cycles and retention time are the main error source for 3D NAND flash [15]. The same result can be found in [16, 17, 18, 19]. According to the research work [20, 21], we plots the RBER induced by different error sources as shown in Fig. 1. In this paper, we

Fig. 1 The RBER induced by different error sources.

Fig. 2 The read latency comparison among LALDPC-FIFO, LALDPC-Amph and ideal.

consider the error mainly comes from the P/E cycles and retention time for simplicity. If read disturb and program interference become the main error source in the future NAND flash-based SSDs, they also can adopt the same design principle of this paper.

Two existing issues motivate our research work in RL recording method and the I/O scheduler. Firstly, As the RBER of each page varies with P/E cycles and retention time, the existing methods ignore the different impact of P/E cycles and retention time on the RBER. If the RL of each page induced by P/E cycles is known in advance, the FTL does not need to reset the RL to be one after the block is erased; if the RL of each page induced by different retention time is also known in advance, the FTL can calculate the fittest RL instead of the recording RL to sense the page after the page suffers a long retention time. Furthermore, these read requests usually have different read latency. Still, the existing methods consider the parallelism, utilization or other factors [22, 23, 24], if we schedule the read request according to their servicing time, so the average read latency could be reduced as a result of the reduced queue latency.

An experiment has been done to illustrate these issues. The read latency comparison result among *LALDPC – FIFO*, *LALDPC – Amph* and the ideal one are presented in Fig. 2. The detailed configuration of SSD and the workload characterization are given in section IV. We observe that optimization potential exists in the existing schemes. In the next section, we present the detailed design of our proposal schemes.

3. Design

3.1 Overview

Motivated by the preliminary experimental result, we design both error source-aware RL recording scheme and latency aware I/O scheduling algorithm, which aims to meet the following objectives.

- Mitigating the read performance degradation caused by mis-estimation. The RBER of one page caused by P/E cycles increases after the block is erased; the RBER of one programmed page also increases over time. We need to distinguish the RBER caused by different error sources to apply the fittest RL for reads.
- Utilizing the different read latency among different requests to reduce the queue latency. The requests in the queue have different RL, a basic idea is to service the request with least RL first to reduce the queue latency of its following requests.

The figure 3 shows two main components proposed in this paper. The first component named as error source-aware RL recording scheme, which is used for tracking the diverse RLs; the second component, namely latency-aware request reordering algorithm, goes through each coming request and reorder them in the request queue according to their latency.

3.2 Error source-aware RL recording scheme

Let's review the read request procedure. Once the read requests are issued into SSD, the FTL obtains its physical addresses and recorded RLs, then the FTL employs corresponding read sensing voltage to read raw data. The recorded RLs contains one RL counter induced by P/E cycles and several RL counters induced by retention time and its corresponding retention time interval. For convenience, we call the RL induced by P/E cycles and the RL induced by retention time as $PE - RL$, $RE - RL$, respectively. The RL to sense the page is the sum of the $PE - RL$ counter and one $RE - RL$ counter. We assume that all the pages in the same block have similar retention time, and each block has its timestamp, which records that time after the first page is programmed. We chose that the difference between the coming time of read request and the time stamp of block as the retention time of the $RE - RL$. Let's take an example, assuming the $PE - RL$ counter of one page is 2, and the $RE - RL$ counters are (1,0) and (30,1). The former

represents the retention time (days), and the latter represents $RE - RL$ counter. For a page, if its retention time is less than 1 days, we use the $RL\ 2 + 0$ to read page. The next section contains two-folds, i.e., the constructing of RL recording table, management of RL recording table.

(1) Constructing RL recording table: It is used for recording RL of each page. The existing methods keep track of RL at page level, layer level and block level. However, due to the process variation existing inside block [25, 26], recording RL at block level may induce to erroneous RL recording, even it saves memory overhead. In our design, we apply the RL recording at the page level for planner NAND flash and layer level for 3D NAND flash, respectively. In the next section, we take the constructing of RL recording table in planner NAND flash as an example. Besides, Two methods could be used for constructing the RL table. The first method extends the address mapping table, attaching the RL recording of each page as the extra information to the entry. The second method is that the FTL maintains an independent table to record this information. However it consumes some extra memory to track the physical address, this method costs more space compared with the first method. We choose the second method for RL recording. The RL entries contain several parts: the $PE - RL$ counter and $RE - RL$ tuples, including $RE - RL$ counters and its retention time interval.

(2) Initiation&Update RL recording table: The $PE - RL$ and $RE - RL$ counters are set to one and zero when the P/E cycle is less and retention time is less than a predefined threshold. It means that the FTL uses RL one to sense page. After the pages have been erased thousands of times and sustain a long retention time. The RBER of some pages increases, so it may fail to decode data with RL one for some requests, then the FTL increases the RL to read and decode data until its success. The update of $PE - RL$ counter and $RE - RL$ counters depicts at the following scenarios. If the pages have been written, the following read request will be issued by FTL to read this page, then the value of $PE - RL$ counter is obtained by using the LDPC decoder. Note that we only utilize the first write request of this page to obtain the $PE - RL$, and update it after the block which this page resides, is erased. After a while, assuming the host delivers some read requests, the FTL will try to use $PE - RL$ counter to read the page. If success, then the data is returned to host, and we update the $RE - RL$ to be zero and the retention time to be the range from the time stamp to now; if it fails, a more fine-granularity voltage is employed until success. We update the $RE - RL$ to be the difference between the successful RL and the $PERL$, also record the corresponding retention time. After one block is erased, the $RE - RL$ and the $PE - RL$ unchanged.

By the management of error source-based RL recording method, the FTL could avoid some useless fail read after block is erased.

3.3 Latency aware I/O scheduling algorithm

As mentioned above, Once the FTL tracks the RL of all the pages, we can utilize the recorded RL to improve the read performance further. The default I/O scheduler employs FIFO (First-In-First-Out) scheme to serve the request, and

Fig. 3 System architecture of the proposed scheme.

prioritizes the read request over the write request, as a result of the high write latency over the read latency. In this section, we propose to reorder the waiting request dynamically and prioritize the request having the low servicing time so that the average queuing time could be reduced.

In order to implement our proposed scheme, we introduces state machine to schedule the read requests. The requests in the queue have three different states: the state *A* waiting for reordering, the state *B* waiting for be serving, the state *C* being serviced. All the read requests go into state *A* first. If no requests in either state *B* or *C*, the FTL groups less than or equal to the predefined number of requests (the batch size are 5) in state *A* and sorts them according to their servicing time. The estimation of servicing time contains two parts, i.e., queue latency and RL latency. We estimate the queue time of request based on [27], which is simple but effective. As a large request may span several channels, the queue time of each request is the max queue time of its sub-requests. The RL latency depends on the scheme mentioned above. The number of the requests in the queue varies with the workload, if we set a static batch size, it cannot be suited for all the workloads, so we adjust the value of the batch size dynamically. We set the default value to be 5 for simplification, and this value may be updated according to the average queue length in the channel during SSD is running.

3.4 Overhead analysis

Storage overhead comes from the introduced RL recording table. The entry in the table contains two parts — *PE* – *RL* counter, several *RE* – *RL* counters and its time interval. It always consumes 32-bits to record the physical page number in mapping table [28]. Assuming we use 3 bits to store RL, which ranges from 1 to 7, and uses 4 bits for retention time. 32-bits is enough to save one *PE* – *RL* counter and four pair of retention time interval and *RE* – *RL* counters. The negligible storage overhead has been validated in [8].

4. Experimental methodology

To evaluate the effectiveness of the proposed scheme, we implemented the proposed scheme, and the existing read optimization scheme (LALDPC [8], Amphibian [29]) based on SSDsim, which has been validated against hardware platform [30]. Table I provides the detailed configuration of the NAND flash-based SSD.

We used the enterprise servers traces from Microsoft research Cambridge [31, 32] to evaluate the proposed scheme. These workloads are widely used in previous studies. The details of these workloads are given in Table II.

In this section, we compare the following schemes.

- LALDPC-FIFO — The scheme uses LALDPC [8] as read-level recording scheme and FIFO (First In First Out) as I/O scheduler.
- LALDPC-Amph — We also combine the I/O scheduler mentioned in amphibian [29] to compare with the proposed I/O scheduler. In this scheme, the read-level recording method employs LALDPC.
- ESRR-FIFO — This scheme implements source error-aware RL recording optimization scheme and use FIFO

Table I SSD parameters.

NAND flash-based SSD parameters	
(Channel, Chip, Die)	(4, 8, 8)
(Plane, Block, Page)	(8, 1888, 1792)
(Page size, Cell density)	(16 KB, TLC)
Latency	write 900 us, Erase 10 ms
Latency	read (90 us, 120 us, 180 us)
Capacity per page	4 KB

Table II Statistics of workloads.

trace	r/w ratio	avg. read	avg. write	days
HM_0	0.25	11.61	11.21	2.2
PRN_0	0.11	26.55	13.93	1.3
PROJ_3	0.90	15.03	30.14	5.0
PRXY_0	0.03	9.7	6.2	0.5
RSRCH	0.10	15.70	12.70	4.9
STG_0	0.23	33.56	12.69	3.16
SRC2_0	0.13	12.64	11.01	1.7

as I/O scheduler.

- LALDPC-LASA — This scheme implements LASA (Latency Aware I/O Scheduling Algorithm) with LALDPC based read-level recording schemes.
- ESRR-LASA — This scheme implements both proposed ESRR and LASA schemes.

4.1 Experimental precondition

The evaluation contains three parts. In the first part, we study the effectiveness of ESRR and LASA with different schemes under different conditions. In the second part, we exploit the distribution of read requests which benefit from our schemes. In order to achieve this goals, we constructs that experimental settings which is consistent with [8]. We assume that the RBER of these pages range from 0.006 to 0.009, and so the read-level of these pages range from 3 to 4. To explore the effectiveness of the proposed schemes, we set the ratio of the erased blocks and in-used blocks to be 7 : 3 during start-up procedure and also we assume the ratio of P/E cycles error and retention time error are 8 : 2, respectively. That means LALDPC record the RL of erased block as 1, while ESRR record the RL of erased block as *PE* – *error*. In the first part, we study the read performance about these schemes by measuring the response time as a performance metric. In the second part, we study the read latency distribution by collecting read requests from the output of SSDsim to plot the CDF (Cumulative Distribution Function). In the third part, we study the effectiveness of the proposed scheme with different configuration on the ratio of *PE* – *error* and *RE* – *error*.

4.1.1 Read latency distribution

We have statistics on the completion time of each read request across all the workloads, Fig. 4 shows the read latency among these schemes. We can find that the proposed schemes reduce the read latency by about 20% on average (range from 33% to 12%) compared with the LALDPC-FIFO. Both ESRR and LASA contribute to the read performance improvement. ESRR reduces the number of failed reads and their associated penalty. LASA reduces the queue latency by pick up the requests with low RL and inserting them in the front of the request queue without sacrificing of other requests. The workloads such as PROJ_3,

Fig. 4 Read latency comparison.

Fig. 5 The CDF of read latency.

Fig. 6 The read latency under different percentage of erased blocks.

PRXY_0, and RSRCH provide high opportunities to improve the read performance. However, we observed that some workloads, such as SRC2_0, achieve little improvement.

4.1.2 Read latency distribution analysis

As shown in Fig. 5, we collect the read latency of these read requests in workloads HM_0 from the output of SSDsim and plot the CDF of read latency. The x-axis represents the read latency (the unit is ns), and the y-axis represents CDF. As we expect, the proposed schemes reduce the read latency of many read requests. About 50% of read requests has read latency of 0.185 ms in *LALDPC – FIFO*, 0.145 ms in *LALDPC – Amph* and 0.135 ms in proposed scheme, it reduces about 27% and 6% read latency compared to *LALDPC – FIFO* and *LALDPC – Amph* schemes. The similar result could be observed for 90% of read requests.

4.1.3 Sensitivity study

In this section, we studied the impact of different combination of error sources on the proposed schemes. Because the read-level of page depends on RBER, the RBER also are decided by the error sources (i.e., P/E cycles and retention time). In this section, we set the ratio of *PE – error* and the *RE – error* to be 1 : 1 and 8 : 2, respectively. The results are shown in Fig. 6. We observed that, compared to the 50% of erased block in Fig. 6b, the read latency in

Fig. 6a are decreased a lot. The result illustrates our proposed scheme- ESRR works effectively and achieves better performance improvement.

5. Conclusion

LDPC-based read not only provides stronger error correction capacity over other ECCs, which extends the lifetime of SSD but also bring the long read latency problem. In this paper, we propose two schemes to alleviate the long read latency issues. The first scheme recognizes different error sources and employs the separate RLs to reduce fail reads. This fine-granularity recording method avoids the performance fluctuations due to the mis-estimated RL. The second scheme depends on the first scheme, reorders the request in the request queue in batch to reduce the queue latency. In the experimental section, we validate the proposed scheme achieves better read performance improvement.

References

- [1] R. Michelsoni: *3D Flash Memories* (Springer Netherlands, 2016).
- [2] H. Maejima, *et al.*: “A 512Gb 3b/cell 3D flash memory on a 96-word-line-layer technology,” *Dig. Tech. Papers IEEE Int. Solid State Circuits Conf.* (2018) 336 (DOI: [10.1109/ISSCC.2018.8310321](https://doi.org/10.1109/ISSCC.2018.8310321)).
- [3] W. Liu, *et al.*: “Characterizing the reliability and threshold voltage shifting of 3D charge trap NAND flash,” *Proc. of DATE* (2019) 312 (DOI: [10.23919/DATE.2019.8714941](https://doi.org/10.23919/DATE.2019.8714941)).
- [4] Y. Takai, *et al.*: “Analysis on heterogeneous SSD configuration with quadruple-level cell (QLC) NAND flash memory,” *IEEE Int. Mem. Workshop* (2019) 4 (DOI: [10.1109/IMW.2019.8739689](https://doi.org/10.1109/IMW.2019.8739689)).
- [5] K. Zhao, *et al.*: “LDPC-in-SSD: making advanced error correction codes work effectively in solid state drives,” *Proc. FAST: USENIX Conf. File Storage Technol.* (2013).
- [6] T. Tokutomi, *et al.*: “Advanced error prediction LDPC for high-speed reliable TLC nand-based SSDs,” *IEEE Int. Mem. Workshop* (2014) 1 (DOI: [10.1109/IMW.2014.6849375](https://doi.org/10.1109/IMW.2014.6849375)).
- [7] J. Cui, *et al.*: “DLV: exploiting device level latency variations for performance improvement on flash memory storage systems,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **37** (2017) 1546 (DOI: [10.1109/TCAD.2017.2766156](https://doi.org/10.1109/TCAD.2017.2766156)).
- [8] Y. Du, *et al.*: “LaLDPC: Latency-aware LDPC for read performance improvement of solid state drives,” *IEEE Symp. Mass Storage Syst. Technol.* (2017) 1.
- [9] Y. Shim, *et al.*: “Exploiting process similarity of 3D flash memory for high performance SSDs,” *Proc. Annu. Int. Symp. Microarchitecture MICRO* (2019) 211 (DOI: [10.1145/3352460.3358311](https://doi.org/10.1145/3352460.3358311)).
- [10] R. Michelsoni: *Inside Solid State Drives (SSDs)*, ed. A. Marelli and K. Gshghi (Springer Netherlands, Berlin, 2010).
- [11] E. Dong, *et al.*: “On the use of soft-decision error-correction codes in NAND flash memory,” *IEEE Trans. Circuits Syst. I, Reg. Papers* **58** (2011) 429 (DOI: [10.1109/TCSI.2010.2071990](https://doi.org/10.1109/TCSI.2010.2071990)).
- [12] Y. Cai, *et al.*: “Error patterns in MLC NAND flash memory: measurement, characterization, and analysis,” *Proc. of DATE* (2012) 521 (DOI: [10.1109/DATE.2012.6176524](https://doi.org/10.1109/DATE.2012.6176524)).
- [13] Y. Cai, *et al.*: “Threshold voltage distribution in MLC NAND flash memory: characterization, analysis, and modeling,” *Proc. of DATE* (2013) 1285 (DOI: [10.7873/DATE.2013.266](https://doi.org/10.7873/DATE.2013.266)).
- [14] Y. Cai, *et al.*: “Experimental characterization, optimization, and recovery of data retention errors in MLC NAND flash memory,” *arXiv preprint* (2018) arXiv:1805.02819.
- [15] F. Wu, *et al.*: “Characterizing 3D charge trap NAND flash: observations, analyses and applications,” *Proc. IEEE Int. Conf. Comput. Des.* (2018) 381 (DOI: [10.1109/ICCD.2018.00064](https://doi.org/10.1109/ICCD.2018.00064)).
- [16] Y. Luo, *et al.*: “Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation,” *Proc. ACM Meas. Anal. Comput. Syst.* **2** (2018) 37 (DOI: [10.1145/3224432](https://doi.org/10.1145/3224432)).
- [17] Y. Luo, *et al.*: “HeatWatch: improving 3D NAND flash memory

- device reliability by exploiting self-recovery and temperature awareness,” *Proc. of HPCA* (2018) 504 (DOI: [10.1109/HPCA.2018.00050](https://doi.org/10.1109/HPCA.2018.00050)).
- [18] M. Fukuchi, *et al.*: “System performance comparison of 3D charge-trap TLC NAND flash and 2D floating-gate MLC NAND flash based SSDs,” *IEICE Trans. Electron.* **E103-C** (2020) 161 (DOI: [10.1587/IRPS.transele.2019CDP0005](https://doi.org/10.1587/IRPS.transele.2019CDP0005)).
 - [19] K. Mizoguchi, *et al.*: “Automatic data repair overwrite pulse for 3D-TLC NAND flash memories with 38x data-retention lifetime extension,” *IEEE Int. Reliab. Phys. Symp. Proc.* (2019) (DOI: [10.1109/IRPS.2019.8720420](https://doi.org/10.1109/IRPS.2019.8720420)).
 - [20] N.R. Mielke, *et al.*: “Reliability of solid-state drives based on NAND flash memory,” *Proc. IEEE* **105** (2017) 1725 (DOI: [10.1109/jproc.2017.2725738](https://doi.org/10.1109/jproc.2017.2725738)).
 - [21] B.S. Kim, *et al.*: “Design tradeoffs for SSD reliability,” 17th USENIX Conference on File and Storage Technologies (FAST 19) (2019).
 - [22] M. Jung and M.T. Kandemir: “Sprinkler: maximizing resource utilization in many-chip solid state disks,” *Proc. of HPCA* (2014) 1 (DOI: [10.1109/HPCA.2014.6835961](https://doi.org/10.1109/HPCA.2014.6835961)).
 - [23] C. Gao, *et al.*: “Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives,” *IEEE Symp. Mass Storage Syst. Technol.* (2014) 11 (DOI: [10.1109/MSST.2014.6855544](https://doi.org/10.1109/MSST.2014.6855544)).
 - [24] C. Gao, *et al.*: “Boosting the performance of SSDs via fully exploiting the plane level parallelism,” *IEEE Trans. Parallel Distrib. Syst.* **31** (2020) 2185 (DOI: [10.1109/TPDS.2020.2987894](https://doi.org/10.1109/TPDS.2020.2987894)).
 - [25] L. Shi, *et al.*: “Exploiting process variation for write performance improvement on NAND flash memory storage systems,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **24** (2016) 334 (DOI: [10.1109/TVLSI.2015.2393299](https://doi.org/10.1109/TVLSI.2015.2393299)).
 - [26] Y. Pan, *et al.*: “Exploiting memory device wear-out dynamics to improve NAND flash memory system performance,” *Proc. FAST: USENIX Conf. File Storage Technol.* (2011) 245.
 - [27] N. Elyasi, *et al.*: “Exploiting intra-request slack to improve SSD performance,” *Int. Conf. Archit. Support Program. Lang. Oper. Syst. ASPLOS* (2017) 375 (DOI: [10.1145/3037697.3037728](https://doi.org/10.1145/3037697.3037728)).
 - [28] A. Gupta, *et al.*: “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” *ACM SIGPLAN Not.* **44** (2009) 229 (DOI: [10.1145/1508284.1508271](https://doi.org/10.1145/1508284.1508271)).
 - [29] M. Bo, *et al.*: “Improving the SSD performance by exploiting request characteristics and internal parallelism,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **37** (2018) 472 (DOI: [10.1109/TCAD.2017.2697961](https://doi.org/10.1109/TCAD.2017.2697961)).
 - [30] Y. Hu, *et al.*: “Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity,” *Proc. of ICS* (2011) 96 (DOI: [10.1145/1995896.1995912](https://doi.org/10.1145/1995896.1995912)).
 - [31] D. Narayanan, *et al.*: “Write off-loading: practical power management for enterprise storage,” *ACM Trans. Storage* **4** (2008) 10 (DOI: [10.1145/1416944.1416949](https://doi.org/10.1145/1416944.1416949)).
 - [32] D. Narayanan, *et al.*: “Migrating server storage to SSDs: analysis of tradeoffs,” *Proc. of EuroSys.* (2009) 145 (DOI: [10.1145/1519065.1519081](https://doi.org/10.1145/1519065.1519081)).