# BER-Based Wear Leveling and Bad Block Management for NAND Flash

Borja Peleato*, Haleh Tabrizi†, Rajiv Agarwal‡, Jeffrey Ferreira†
*Electrical and Computer Engineering, Purdue University
bpeleato@purdue.edu
†DSSD, Inc/EMC
{haleh, jpf}@dssd.com
‡Electrical Engineering, Stanford University
rajivag@stanford.edu

***Abstract*—One of the main challenges keeping flash memories from achieving widespread distribution is their limited endurance. The programming and erasing from re-writes damages the cells, progressively increasing the number of errors until information can no longer be stored reliably. Most manufacturers employ powerful ECC techniques, but there is a limit to the number of errors that these can correct. When the number of errors goes beyond the capability of the ECC, it is necessary to invoke RAID, which introduces significant latency and jeopardizes the speed of the memory if used too often.**

**This paper introduces a method for estimating the BER that a flash page will exhibit after retention and uses this estimate for wear leveling. Instead of leveling out the number of PE cycles in all the blocks, the proposed scheme attempts to wear all the blocks evenly so that they all suffer the same BER. Additionally, the estimate will be used to detect bad blocks, those likely to exhibit a number of errors beyond the ECC correction capability, and retire them from further use.**

## I. Introduction

One of the main challenges of NAND flash memories is their limited endurance. NAND blocks have a limited program-erase (PE) cycle count, or equivalently there is a limit to the number of times information can be re-written. The program and erase operations damage the dielectric barrier in the flash cells, increasing the Bit Error Rate (BER) observed in subsequent reads. Memory controllers generally use Error Correcting Codes (ECC) to recover some of these erroneous bits, but there is a limit to the number of errors that the ECC can correct. The dielectric barrier eventually reaches a state in which the cells can no longer store information reliably. When a certain number of the blocks reach this point, the memory is considered dead. If a subset of blocks are programmed and erased more often than the others they will suffer more damage and die earlier, effectively decreasing the lifetime of the memory. Wear leveling algorithms are therefore used to increase the memories' lifetime by prioritizing the use of undamaged blocks over worn out ones during the write process.

Most wear leveling algorithms take the number of PE cycles as a proxy for the amount of damage that a block has suffered. The number of PE cycles is clearly correlated with the state of the block, but it is not the best measure. Some blocks may be able to withstand a larger number of PE cycles than others before becoming unusable.

This paper presents a wear leveling and bad block detection scheme which classifies the blocks based on the BER that they suffered during previous write operations and on other factors correlated with the damage suffered by the cells. Instead of attempting to even out the number of program and erase cycles in all the blocks, like most of the existing wear leveling schemes, the technique described herein attempts to evenly wear out the blocks so that they all suffer the same raw BER under identical conditions. In most cases, worn blocks will be written less often than their healthier counterparts, but all of them will be active. Occasionally, however, our scheme will detect that a specific block is likely to suffer a BER beyond the ECC correction capability, mark it as bad, and signal the controller that the block (or a subset of its pages) needs to be retired from further use.

There are two reasons why it is desirable that all blocks in a memory suffer the same BER. First, to increase lifetime. Flash memories are designed with some amount of overprovisioning, i.e. extra blocks, for garbage collection and so that bad and worn out blocks can be retired without impacting the user capacity. However, there is a limit to the number of blocks that can be discarded. When the capacity of the remaining blocks is getting close to the nominal capacity, the memory usually signals the user that it is at the end of life. Normally, the user can still recover the information that was stored in the drive, but the controller does not allow additional writes. At this point, having some blocks in very good health is useless. The goal therefore is to spread out the wear among the blocks so that the memory lasts as long as possible.

The second reason why it is desirable that all blocks have the same BER is to help the ECC. At the beginning of the memory's life, when all blocks are fresh, the BER is very low. It is therefore possible to use high rate error correcting codes (ECC), which are more efficient in terms of memory usage and power consumption than low rate ones but they have lower error correction capability [1]. As the blocks wear out and BER increases it becomes necessary to use more powerful codes with lower coding rates. Some existing controllers change the code rate adaptively during the memory's lifetime

[2]. Such controllers choose the code to be used for the data written on a block based on the number of PE cycles that the block has endured. This means that the code chosen for a given number of PE cycles must be able to correct the worst case BER that can arise for that number of PE cycles. However, different blocks can suffer very different BER after the same number of PE cycles. Measuring the BER directly, as this paper proposes, would enable the use of higher code rates, since the ECC scheme can adapt the coding rate to a block's expected BER, instead of to the worst case estimate after a given number of PE cycles.

Unfortunately, the BER observed at read time is not always a good indicator of the amount of damage that a block has suffered. There is another important factor, apart from cell wear, that influences the BER observed at read time: leakage currents. Time and heat cause some of the charges trapped in the cells to escape, effectively decreasing the stored voltages. Leakage increases as cells wear out, but it is also directly related to the amount of time elapsed since the data was written and the temperature at which the memory was kept. A worn out block read shortly after being written might therefore show lower BER than a healthier one if the latter was written long before the time of read.

It is therefore necessary to minimize the disturbance caused by leakage, or at least ensure that the BER is measured in the same conditions for all blocks. Some systems already have background processes, such as data scrubbing and refreshing, that require periodically reading all blocks. If all blocks are read with similar frequency, these reads can be used to evaluate the BER. Alternatively, the controller can read each block right after it is written, or proactively read pages at pre-defined intervals.

This paper takes the BER of a block read 3 months after being written as its measure for damage. Then it studies how different metrics correlate with this damage, and builds a model to predict damage based on those metrics. The proposed wear leveling algorithm will attempt to even out the damage, as estimated by our model, for all the blocks. Additionally, this damage estimate will be used to decide when a block or a subset of its pages must be refreshed or retired.

The paper is organized as follows. Section II will provide some background about NAND flash storage systems and related work that can be found in the literature. Section III introduces the model that will be used throughout the paper. Section IV presents the proposed method for estimating the damage that each page has suffered, which is then used in Section V for wear leveling purposes and in Section VI for detecting bad blocks. Finally, Section VII summarizes and concludes the paper.

## II. Background

### A. NAND flash storage

NAND Flash memories use a log-structured file system with out-of-place write [3], [4]. Instead of erasing and re-programming the entire block whenever data is being updated, the new data is written on a different block and the old page is marked as invalid. From the host perspective the flash is a list of logical blocks which can be overwritten at will, but the physical address in the Flash for each of those blocks keeps changing. The flash controller keeps and frequently updates a table mapping logical addresses to physical addresses, so all the practical complexities associated to the block erasing, garbage collection, wear-leveling, and out-of-place write are hidden from the host and seamlessly addressed by the flash translation layer [5].

Write requests specify the virtual address where the host wants to store the information, but this virtual address is different from the physical address on the flash where the information is written. Instead of erasing and re-writing a whole block just to update a few of its pages, the controller keeps a pool of erased pages available to accommodate new or relocated information. When the controller receives a request to overwrite a given virtual address, it performs three steps. First, it finds the most convenient location to store the new data among the pool of erased pages, and writes the new information there. Then, it finds the physical address previously corresponding to the given virtual address, and marks those physical pages as containing invalid information. Finally, the controller updates the corresponding entry in the virtual to physical mapping to point to the new location. When the host requests information from a specific virtual address, the controller finds the corresponding physical address and returns the data read from that address, after having corrected any bit errors that might have occurred.

Periodically, it is necessary to erase blocks with a large number of invalid pages to refill the pool of available space. The process of selecting these blocks, relocating their valid pages to other blocks and erasing them is known as garbage collection. In practice, it is common for a memory to store both cold data, which is very rarely updated, and hot data, which is updated often. Blocks storing hot data are programmed and erased more often than those storing cold data. Wear leveling algorithms are therefore used to increase the memories lifetime by occasionally moving cold data between blocks. Garbage collection and wear leveling are closely related processes that are sometimes combined into a single process.

As cells wear out, they become more susceptible to perturbations in their programmed voltages, such as inter-cell-interference (ICI) and over-programming. Consequently, data read from worn out blocks generally presents a higher bit error rate than that read from fresh blocks.

Information loss is unacceptable in storage systems, so they usually have several layers of data protection available. Even if the BER on a page is too large for the ECC to correct, it is possible to invoke the RAID protocol which recovers the whole page from information stored on other blocks or drives. However, invoking RAID significantly reduces the read throughput of the memory, so it must be avoided as much as possible.

### B. Related work

There exist several wear leveling and bad block monitoring techniques in the literature. An overview of the most classical techniques can be found in [3], but new variations keep appearing to address specific usage scenarios or system architectures [6], [7], [8]. Most wear leveling techniques attempt to even out the number of PE cycles in all the blocks, but some have attempted to estimate wear using other factors. For example, [9] proposes measuring the time that it takes to erase each block, prioritizing blocks which take longer to erase and retiring blocks which can be erased very fast.

Another approach to wear leveling and bad block management is to rely on a background data refresh process. For example, [10] proposes periodically reading and decoding all the pages in the memory, and reprogramming in place or remapping those whose BER is beyond a pre-fixed threshold. Their goal is to extend the lifetime of the memory by avoiding leakage errors. Before the cells lose their charge, the information is refreshed or transferred elsewhere. This is a feasible solution for systems storing cold data (which is rarely updated) and which are idle during extended periods of time [11], but it could be too time consuming for enterprise scenarios where the memory is used continuously [12].

## III. System Model

The BER is generally measured by comparing the raw data read from the flash with the data at the output of the error correcting block. In NAND flash, each page stores multiple kilobytes of information, so keeping two copies and comparing them can require a large amount of memory. In practice, the model does not require a very precise BER value, so it is possible to compare only a subset of the cells. Similarly, measuring the BER for all the pages in a block can be too time consuming so the proposed scheme only evaluates a few pages chosen to be representative of the block. Some pages are more vulnerable to disturbs than others, depending on their location within the block, so it is important that the same subset of pages is chosen in all blocks.

This paper will consider a MLC flash memory with 256 pages per block and 16 kbytes of information per page. All the pages in a block are programmed at the same time. We measure and store the time that it took to program each of them. This information is later used to classify bad pages. After each block programming operation finishes, the controller reads a pre-fixed set of pages of the newly written block, decodes them, and counts the number of errors corrected. For the sake of simplicity, the paper will assume that the first 10 upper pages of the block are read, but in practice the number and index of the pages read will depend on the specific system and memory manufacturer[1]. Pages are read in sequence, temporarily storing the data read in a buffer while the ECC decoder corrects the errors. The data in the buffer is then compared with the output of the decoder, counting the number of bits that are different. Once all the chosen pages have been processed in this manner, the cumulative number of errors for this block is stored in a table to be used by the wear leveling and bad block management processes.

It will be assumed that the ECC can correct a BER of up to $0.8\%$. Any page having a higher BER will require invoking RAID. The proposed scheme aims to estimate the BER that a block will suffer after 3 months of retention, using factors that are easy to obtain when the block is written. This estimate will be used for two tasks: guiding the wear leveling algorithm and detecting pages which are likely to invoke RAID when read. The simulations presented use floating point values for the write times, BER, etc. but four bits of precision would probably be more than enough precision in practice.

[1]Page indexing and structuring varies widely across NAND flash manufacturers: blocks often combine MLC and SLC pages, bitlines can follow an even/odd or ABL architecture, etc. Consequently, the exact pages to be read should be tailored to each specific memory.

## IV. Damage estimation

Ideally, the controller would read every page periodically and measure the BER observed as proposed in [10], but this is often unpractical. Reading, decoding, and comparing the results for every page after it is written would significantly slow down the memory. Instead, the health of a page is estimated using three factors that can easily be measured. Specifically,

- PE cycles
- Write time
- BER observed in a subset of pages in the same block, measured right after they are written

It is not necessary to store these values for every page. The number of PE cycles is common for the whole block, so one copy is enough for all its pages. The write time and BER at write time are only needed to estimate the current health of the page. As soon as they are compiled into a health value, they can be discarded. In NAND flash, writing takes significantly longer than reading, so evaluating the BER of a few pages right after each block is written should not have a significant impact on the speed of the memory. Measuring the write time can be done with a simple counter of clock cycles, so it does not jeopardize the performance of the memory either.

Figure 1 illustrates the correlation between these factors and the raw BER observed in each upper page after 3 months of retention (simulated according to Arrhenius equation with activation energy of 1.1eV). Lower pages suffer significantly less BER and will not be considered. The top plots show a scatter plot with one point for each page that was measured. The bottom plots show the average and standard deviation of the same data, grouped into 10 equal bins. It can be observed that the RBER after 3 months has significant variance and outliers among pages with the same PE cycles, the same write time, or the same BER after write. However, if all of those metrics are combined in a linear model the BER can be predicted significantly more accurately, as shown in the rightmost plots. The damage estimator for a block $x$ is

$$f(x) = \alpha \cdot PE + \beta \cdot \frac{1}{T_{\text{write}}} + \gamma \cdot (\text{BER after write}), \quad (1)$$

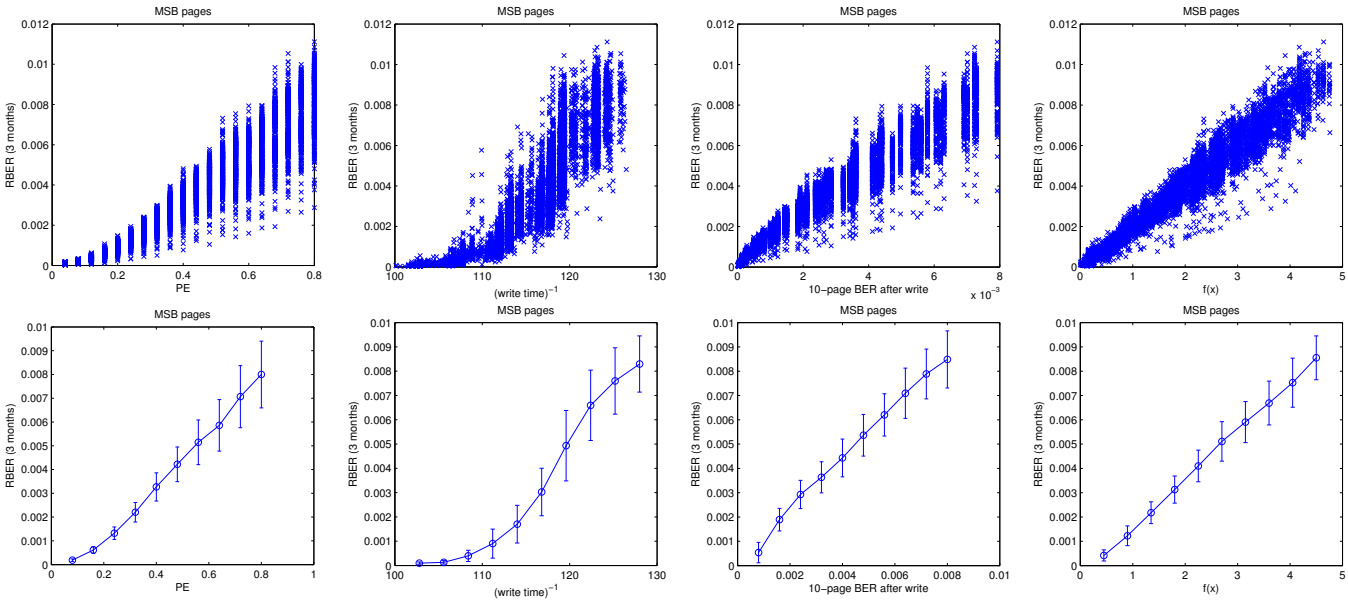


Fig. 1. Top plots show scatterplots with one point for each page that was measured, specifying the RBER observed after a 3 month retention period as a function of PE cycles, programming time, BER at write time, and $f(x)$ from left to right, respectively. Bottom plots show the average and standard deviation of the same data. Values are relative, the actual ones have been obscured to comply with our agreements with NAND manufacturers.

where the model parameters $\alpha$, $\beta$, and $\gamma$ are chosen by least squares approximation to the data. Specifically,

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T \begin{bmatrix} \vdots \\ \mathbf{y} \\ \vdots \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{p} & \mathbf{t}_{\text{write}} & \mathbf{b} \\ \vdots & \vdots & \vdots \end{bmatrix},$$

where $\mathbf{p}$, $\mathbf{t}_{\text{write}}$, and $\mathbf{b}$ denote the (column) vectors of PE cycles, write times, and BER for each page, respectively.

This estimator therefore provides a significantly more accurate estimate for the BER in the blocks after retention than the PE cycles alone. As more parameters are added to the method, the results would be progressively more accurate. It is up to the controller designer to decide what parameters should be included for each specific product. A popular parameter that has been omitted from our simulations is the erase time for each block.

## V. Wear Leveling

There exist many different wear leveling algorithms in the literature. From a high level perspective, most algorithms are based on periodically checking whether some conditions are met, and initiating a wear leveling operation if they are. However, the conditions to initiate and the actions performed in a wear leveling operation can vary significantly between systems. All of the different wear leveling algorithms have their own pros and cons, and the decision on which one should be used depends on the specifics of the system where they need to be implemented. It is beyond the scope of our work to discuss how the above damage estimation model could be combined with each of these algorithms, but the primary directive would be to use the damage estimator described in Section IV instead of the number of PE cycles in all cases.

As an example, we simulate 1000 blocks of a flash memory, with 11% overprovisioning (100 free blocks) storing both cold (never updated) and hot data in equal parts. The wear leveling algorithm has access to both the list of blocks in the erased pool and a list of blocks storing cold data. It compares the damage estimator $f(x)$ of the cold data blocks with that of the erased blocks, and when the lowest $f(x)$ among the cold-data blocks is below 90% of the highest $f(x)$ among the erased blocks, the wear leveling operation is initiated. The healthiest block in the cold data block list is then exchanged with the most damaged block in the erased block pool. The exchange is done by copying the cold data from the former onto the latter block and then exchanging the block number entries in the erased pool and cold data lists. Since cold data is very rarely updated, the healthy block will subsequently be employed more often than the damaged one. As a framework for comparison, we also simulated the same algorithm using the number of PE cycles instead of the damage estimator.

The results are illustrated in Figure 2. The plot shows the standard deviation of the BER that the blocks would experience after 3 months at different points of the simulation (the average BER would be the same in both cases). The lab collected data shown in Fig. 1 was used to map $f(x)$ and the number of PE cycles into a BER value. It can be observed that for the first few samples, leveling the PE cycles creates more

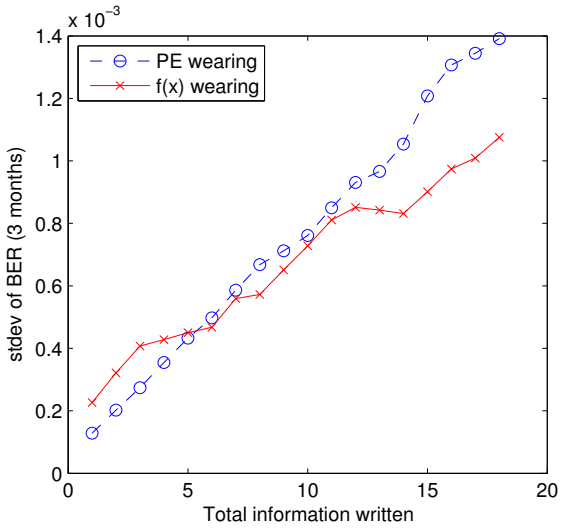


Fig. 2. Evolution of the standard deviation of the BER throughout the memories' lifetime. Wear leveling based on $f(x)$ provides lower standard deviation near the end of life, hence more uniform BER among the blocks.

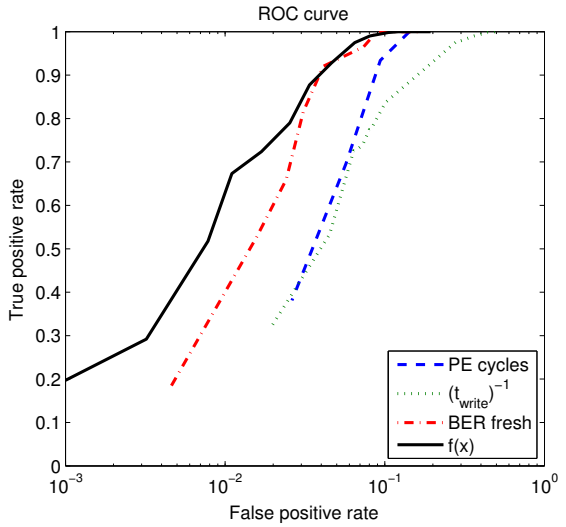


Fig. 3. ROC curves for different bad page detectors at the end of life. A page is considered bad if it has BER $> 0.008$ after 3 months retention.

uniform BER values, but the positions change when the blocks start wearing out. At the end of life, which is the interval that matters in practice, the proposed scheme offers significantly less variation in its BER, which in turn means that RAID will be invoked less often.

If invoking RAID takes 10 times longer than a regular read and the ECC correction capability is two standard deviations above the BER, then wear leveling based on $f(x)$ would provide a 20% increase in the average read throughput at the end of life respect to wear leveling based on PE cycles alone.

## VI. Bad block management

Occasionally, some blocks or sets of pages become so damaged that they need to be permanently retired. Otherwise, they would often present BER values too large for the ECC to correct, requiring the use of RAID and slowing down the memory. One of the most common signals to trigger this retirement are program, erase, and read failures. Whenever one of these happens, the controller usually marks the block as bad. It transfers its valid information to a fresh block and erases the damaged block from the memory map, so that it is never used again.

However, by the time those failures happen, it is usually too late. The damaged block has most likely been suffering high BER for a long time before it triggers a complete program, erase or read failure. The damage estimate found in Section IV provides a method for preemptively retiring blocks which are reaching the point in which they will need to invoke RAID. Figure 1 shows that there is a significant correlation between $f(x)$ and the BER observed after 3 months of retention, so the main idea is to set a pre-fixed threshold $T$ on $f(x)$, so that blocks which have $f(x) \geq T$ in average right after being read will be marked as bad. It is also possible to divide the blocks into sub-blocks, each consisting of a fixed set of pages, and implement the scheme in terms of these sub-blocks. For each sub-block, one or a subset of the pages will be evaluated and, if the value of $f(x)$ observed is beyond $T$, that subset of pages will be marked as bad.

Figure 3 shows the receiver operating characteristic (ROC) curves obtained by moving the threshold in this method, as well as those obtained when using each of the factors independently to detect bad pages. A page is considered bad if it has BER $> 0.008$ after 3 months retention. It can be observed that the damage estimator $f(x)$ performs better than any of the factors independently. It provides fewer false positives for any desired true positive rate.

Once a block (or set of pages) is marked as bad, there are several policies that can be followed. If the speed of the system is critical and there is spare capacity, the controller might retire that block indefinitely. However, some pathological programming patterns or operating conditions can cause unusually large number of errors, as well as program, erase or read failures. This does not necessarily mean that the block has become unusable, so retiring the block indefinitely after a single failure might be too hasty. If the controller wants to give the block a second chance, it can just be refreshed through garbage collection. In any case, this refreshing or early retirement of the blocks will reduce the probability of invoking RAID, therefore increasing the read throughput of the memory.

## VII. Summary

This paper proposes using a combination of three easily measured parameters to estimate the damage that a flash page has suffered. This estimate is then used for wear leveling and bad block detection. Instead of attempting to have the same number of PE cycles in all the blocks, the wear leveling algorithm attempts to have the same damage estimate for all the blocks. It is shown through a combination of measurements

and simulations that the proposed approach can reduce the variance in the BER that the blocks suffer at any point in the memory's lifetime. Additionally this estimate can be used to detect and refresh or retire blocks which are likely to present a large number of errors in the near future. Detecting these blocks and retiring them from further use reduces the number of times that the RAID protocol needs to be activated, therefore helping preserve the read speed of the memory. The algorithm will extend very easily across all types of NAND and new generations.

## REFERENCES


[1] H. Choi, W. Liu, and W. Sung, "Vlsi implementation of bch error correction for multilevel cell nand flash memory," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 5, pp. 843–847, 2010.

[2] T.-H. Chen, Y.-Y. Hsiao, Y.-T. Hsing, and C.-W. Wu, "An adaptive-rate error correction scheme for nand flash memory," in *2009 27th IEEE VLSI Test Symposium*, 2009, pp. 53–58.

[3] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 138–163, 2005.

[4] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 1, pp. 26–52, 1992.

[5] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable ftl (flash translation layer) architecture for nand flash-based applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, p. 38, 2008.

[6] M. Murugan and D. H.-C. Du, "Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead," in *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*. IEEE, 2011, pp. 1–12.

[7] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: improving nand flash lifetime by balancing page endurance." in *FAST*, 2014, pp. 47–59.

[8] Y.-H. Chang, J.-W. Hsieh, T.-W. Kuo, and C.-C. Yang, "Method for performing static wear leveling on flash memory," Apr. 15 2014, uS Patent 8,700,839.

[9] S.-W. Han, "Flash memory wear leveling system and method," Jan. 18 2000, uS Patent 6,016,275.

[10] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 2012, pp. 94–101.

[11] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," *Computers, IEEE Transactions on*, vol. 59, no. 1, pp. 53–65, 2010.

[12] J. S. Kimmel, R. Pafford, and R. Sundaram, "Concurrent content management and wear optimization for a non-volatile solid-state cache," Dec. 31 2013, uS Patent 8,621,145.