Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance

Yangyang Pan, Guiqiang Dong, and Tong Zhang
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute, USA.

Abstract

This paper advocates a device-aware design strategy to improve various NAND flash memory system performance metrics. It is well known that NAND flash memory program/erase (PE) cycling gradually degrades memory device raw storage reliability, and sufficiently strong error correction codes (ECC) must be used to ensure the PE cycling endurance. Hence, memory manufacturers must fabricate enough number of redundant memory cells geared to the worst-case device reliability at the end of memory lifetime. Given the memory device wear-out dynamics, the existing worst-case oriented ECC redundancy is largely *under-utilized* over the entire memory lifetime, which can be adaptively traded for improving certain NAND flash memory system performance metrics. This paper explores such device-aware adaptive system design space from two perspectives, including (1) how to improve memory program speed, and (2) how to improve memory defect tolerance and hence enable aggressive fabrication technology scaling. To enable quantitative evaluation, we for the first time develop a NAND flash memory device model to capture the effects of PE cycling from the system level. We carry out simulations using the DiskSim-based SSD simulator and a variety of traces, and the results demonstrate up to 32% SSD average response time reduction. We further demonstrate that the potential on achieving very good defect tolerance, and finally show that these two design approaches can be readily combined together to noticeably improve SSD average response time even in the presence of high memory defect rates.

1 Introduction

The steady bit cost reduction over the past decade has enabled NAND flash memory to enter increasingly diverse applications, from consumer electronics to personal and enterprise computing. In particular, it is now economically viable to implement solid-state drives (SSDs) using NAND flash memory, which is expected to fundamentally change the memory and storage hierarchy in future computing systems. As the semiconductor industry is aggressively pushing NAND flash memory technology scaling and the use of multi-bit-per-cell storage scheme, NAND flash memory increasingly relies on error correction codes (ECC) to ensure the data storage integrity. It is well known that NAND flash memory cells gradually wear out with the program/erase (PE) cycling [6], which is reflected as gradually diminishing memory cell storage noise margin (or increasing raw storage bit error rate). To meet a specified PE cycling endurance limit, NAND flash memory manufacturers must fabricate enough number of redundant memory cells that can tolerate the worst-case raw storage reliability at the end of memory lifetime. Clearly, the memory cell wear-out dynamics tend to make the existing worst-case oriented ECC redundancy largely *under-utilized* over the entire lifetime of memory, especially at its early lifetime when PE cycling number is relatively small.

Very intuitively, we may adaptively trade such under-utilized ECC redundancy for improving certain NAND flash memory system performance metrics throughout the memory lifetime. This naturally leads to a PE-cycling-aware adaptive NAND flash memory system design paradigm. Based upon extensive open literature on flash memory devices, we first develop an approximate NAND flash memory device model that quantitatively captures the dynamic PE cycling effects, including random telegraph noise [15, 17] and interface trap recovery and electron detrapping [26, 31, 45], and another major noise source: cell-to-cell interference [25]. Such a device model makes it possible to explores and quantitatively evaluate possible adaptive system design strategies. In particular, this paper explores the adaptive system design space from two perspectives: (1) Since NAND flash

*This material is based upon work supported by the National Science Foundation under Grant No. 0937794

memory program speed also strongly affects the memory cell storage noise margin, we could trade the under-utilized ECC redundancy to adaptively improve NAND flash memory program speed; (2) We could also exploit the under-utilized ECC redundancy to realize stronger memory cell defect tolerance and hence enable more aggressive technology scaling. We elaborate on the underlying rationale and realizations of these two design approaches. In addition, for the latter one, we propose a simple differential wear-leveling strategy in order to minimize its impact on effective PE cycling endurance.

For the purpose of evaluation, using the developed NAND flash memory device model, we first obtain detailed quantitative memory cell characteristics under different PE cycling times and different program speed for a hypothetical 2bit/cell NAND flash memory. Accordingly, with the sector size of 512B user data, we construct a binary BCH code (4798, 4096, 54) with 1.1% coding redundancy that can ensure the data storage integrity at the PE cycling limit of 10K. Using representative workload traces and the SSD model [3] in DiskSim [8], we carry out extensive simulations to evaluate the potential of trading under-utilized ECC redundancy to improve memory program speed while assuming the memory is defect-free. The simulation results show that we could reduce the SSD average response time by up to 32%. Assuming memory defects follow Poisson distributions, we further show that the proposed differential wear-leveling technique can very effectively improve the effectiveness of allocating ECC redundancy for improving memory defect tolerance. Finally, we study the combined effects when we trade the under-utilized ECC redundancy to improve memory program speed and realize defect tolerance at the same time. DiskSim-based simulations show that, even in the presence of high defect rates, we can still achieve noticeable SSD average response time reduction.

2 Background

2.1 Memory Erase and Program Basics

Each NAND flash memory cell is a floating gate transistor whose threshold voltage can be configured (or programmed) by injecting certain amount of charges into the floating gate. Hence, data storage in NAND flash memory is realized by programming the threshold voltage of each memory cell into two or more non-overlapping voltage windows. Before one memory cell can be programmed, it must be erased (i.e., remove the charges in the floating gate, which sets its threshold voltage to the lowest voltage window). NAND flash memory uses Fowler-Nordheim (FN) tunneling to realize both erase and program [7], because FN tunneling requires very low current and hence enables high erase/program parallelism. It is well known that the threshold voltage of erased memory cells tends to have a wide Gaussian-like distribution [41]. Hence, we can approximately model the threshold voltage distribution of erased state as

$$p_e(x) = \frac{1}{\sigma_e\sqrt{2\pi}} e^{-\frac{(x-\mu_e)^2}{2\sigma_e^2}}, \quad (1)$$

where μ_e and σ_e are the mean and standard deviation of the erased state threshold voltage. Regarding memory program, a tight threshold voltage control is typically realized by using incremental step pulse program (ISPP) [6, 39], i.e., all the memory cells on the same word-line are recursively programmed using a program-and-verify approach with a stair case program word-line voltage V_{pp}. Let ΔV_{pp} denote the incremental program step voltage. For the k-th programmed state with the verify voltage $V_p^{(k)}$, ideally ISPP program results in a uniform threshold voltage distribution:

$$p_p^{(k)}(x) = \begin{cases} \frac{1}{\Delta V_{pp}}, & \text{if } V_p^{(k)} \le x \le V_p^{(k)} + \Delta V_{pp} \\ 0, & \text{else} \end{cases}. \quad (2)$$

Unfortunately, the above *ideal* memory cell threshold voltage distribution can be (significantly) distorted in practice, mainly due to PE cycling and cell-to-cell interference, which will be discussed in the remainder of this section.

2.2 Effects of PE Cycling

Flash memory PE cycling causes damage to the tunnel oxide of floating gate transistors in the form of charge trapping in the oxide and interface states [9, 30, 34], which directly results in threshold voltage shift and fluctuation and hence gradually degrades memory device noise margin. Major distortion sources include

1. Electrons capture and emission events at charge trap sites near the interface developed over PE cycling directly result in memory cell threshold voltage fluctuation, which is referred to as random telegraph noise (RTN) [15, 17];
2. Interface trap recovery and electron detrapping [26, 31,45] gradually reduce memory cell threshold voltage, leading to the data retention limitation.

Moreover, electrons trapped in the oxide over PE cycling make it difficult to erase the memory cells, leading to a longer erase time, or equivalently, under the same erase time, those trapped electrons make the threshold voltage of the erased state increase [4, 21, 27, 42]. Most commercial flash chips employ erase-and-verify operation to prevent the increase of erase state threshold voltage at the penalty of gradually longer erase time with PE cycling.

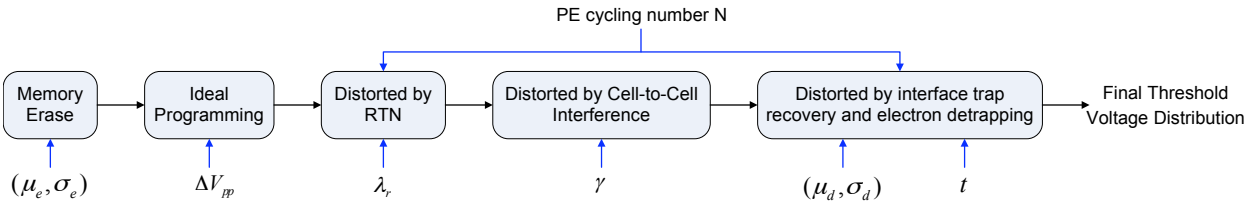


Figure 1: Illustration of the approximate NAND flash memory device model to incorporate major threshold voltage distortion sources.

RTN causes random fluctuation of memory cell threshold voltage, where the fluctuation magnitude is subject to exponential decay. Hence, we can model the probability density function $p_r(x)$ of RTN-induced threshold voltage fluctuation as a symmetric exponential function [15]:

$$p_r(x) = \frac{1}{2\lambda_r} e^{-\frac{|x|}{\lambda_r}}. \tag{3}$$

Let N denote the PE cycling number, λ_r scales with N in an approximate power-law fashion, i.e., λ_r is approximately proportional to N^{α}, where α tends to be less than 1.

Interface trap recovery and electron detrapping processes approximately follow Poisson statistics [30], hence threshold voltage reduction due to interface trap recovery and electron detrapping can be approximately modeled as a Gaussian distribution $\mathcal{N}(\mu_d, \sigma_d^2)$. Both μ_d and σ_d^2 scale with N in an approximate power-law fashion, and scale with the retention time t in a logarithmic fashion. Moreover, the significance of threshold voltage reduction induced by interface trap recovery and electron detrapping is also proportional to the initial threshold voltage magnitude [27], i.e., the higher the initial threshold voltage is, the faster the interface trap recovery and electron detrapping occur and hence the larger threshold voltage reduction will be.

2.3 Cell-to-Cell Interference

In NAND flash memory, the threshold voltage shift of one floating gate transistor can influence the threshold voltage of its neighboring floating gate transistors through parasitic capacitance-coupling effect [25]. This is referred to as cell-to-cell interference, which has been well recognized as the one of major noise sources in NAND flash memory [24,29,36]. Threshold voltage shift of a victim cell caused by cell-to-cell interference can be estimated as [25]

$$F = \sum_k (\Delta V_t^{(k)} \cdot \gamma^{(k)}), \tag{4}$$

where $\Delta V_t^{(k)}$ represents the threshold voltage shift of one interfering cell which is programmed after the victim cell, and the coupling ratio $\gamma^{(k)}$ is defined as

$$\gamma^{(k)} = \frac{C^{(k)}}{C_{total}}, \tag{5}$$

where $C^{(k)}$ is the parasitic capacitance between the interfering cell and the victim cell, and C_{total} is the total capacitance of the victim cell. Cell-to-cell interference significance is affected by NAND flash memory bit-line structure. In current design practice, there are two different bit-line structures, including conventional even/odd bit-line structure [35,40] and emerging all-bit-line structure [10,28]. For write, all-bit-line structure writes all the cells on the same wordline. In even/odd bit-line structure, memory cells on one word-line are alternatively connected to even and odd bit-lines and they are programmed at different time. Therefore, an even cell is mainly interfered by five neighboring cells and an odd cell is interfered by only three neighboring cells. Therefore even cells and odd cells experience largely different amount of cell-to-cell interference. Cells in all-bit-line structure suffers less cell-to-cell inference than even cells in odd/even structure, and the all-bit-line structure can most effectively support high-speed current sensing to improve the memory read and verify speed. Therefore, throughout the remainder of this paper, we mainly consider NAND flash memory with the all-bit-line structure.

2.4 An Approximate NAND Flash Memory Device Model

Based on the above discussions, we can approximately model NAND flash memory device characteristics as shown in Fig. 1. Accordingly, we can simulate memory cell threshold voltage distribution and the corresponding memory cell raw storage reliability. Based upon Eq.(1) and Eq.(2), we can obtain the distortion-less threshold voltage distribution function $p_p(x)$. Recall that $p_{pr}(x)$ denotes the RTN distribution function (see Eq.(3)), and let $p_{ar}(x)$ denote the threshold voltage distribution after incorporating RTN, which is obtained by convoluting $p_p(x)$ and $p_r(x)$:

$$p_{ar}(x) = p_p(x) \bigotimes p_r(x). \tag{6}$$

Cell-to-cell interference is further incorporated based on Eq.(4). To capture the inevitable process variability, we set both the vertical coupling ratio γ_y and diagonal coupling ratio γ_{xy} are random variables with bounded Gaussian distributions:

$$p_c(x) = \begin{cases} \frac{c_c}{\sigma_c\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}, & \text{if } |x-\mu_c| \le w_c \\ 0, & \text{else} \end{cases}, \quad (7)$$

where μ_c and σ_c are the mean and standard deviation, and c_c is chosen to ensure the integration of this bounded Gaussian distribution equals to 1. We set $w_c = 0.1\mu_c$ and $\sigma_c = 0.4\mu_c$ in this work.

Let p_{ac} denote the threshold voltage distribution after incorporating cell-to-cell interference, $p_t(x)$ denote the distribution of threshold voltage fluctuation induced by interface trap recovery and electron detrapping, the final threshold voltage distribution p_f is obtained as

$$p_f(x) = p_{ac}(x) \bigotimes p_t(x). \quad (8)$$

Example 2.1 *Let us consider 2bits/cell NAND flash memory. We set normalized σ_e and μ_e of the erased state as 0.35 and 1.4, respectively. For the three programmed states, we set the normalized program step voltage ΔV_{pp} as 0.3, and the normalized verify voltages V_p as 2.85, 3.55 and 4.25, respectively. For the RTN distribution function $p_r(x)$, we set the parameter $\lambda_r = K_\lambda \cdot N^{0.5}$ where K_λ equals to 4×10^{-4}. Regarding cell-to-cell interference, according to [36, 38], we set the means of γ_y and γ_{xy} as 0.08 and 0.0048, respectively. For the function $\mathcal{N}(\mu_d, \sigma_d^2)$ to capture interface trap recovery and electron detrapping, according to [30, 31], we set that μ_d scale with $N^{0.5}$ and σ_d^2 scales with $N^{0.6}$, and both scale with $\ln(1+t/t_0)$, where t denotes the memory retention time and t_0 is an initial time and can be set as 1 hour. In addition, as pointed out earlier, both μ_d and σ_d^2 also depend on the initial threshold voltage. Hence, we set that both approximately scale with $K_s(x-x_0)$, where x is the initial threshold voltage, and x_0 and K_s are constants. Therefore, we have*

$$\begin{cases} \mu_d = K_s(x-x_0)K_d N^{0.5}\ln(1+t/t_0) \\ \sigma_d^2 = K_s(x-x_0)K_m N^{0.6}\ln(1+t/t_0) \end{cases}, \quad (9)$$

where we set $K_s = 0.333$, $x_0 = 1.4$, $K_d = 4 \times 10^{-4}$, and $K_m = 2 \times 10^{-6}$ by fitting the measurement data presented in [30, 31]. Accordingly, we carry out Monte Carlo computer simulations to obtain the cell threshold voltage distribution as shown in Fig. 2, which illustrates how RTN, cell-to-cell interference, and retention noise affect the threshold voltage distribution.

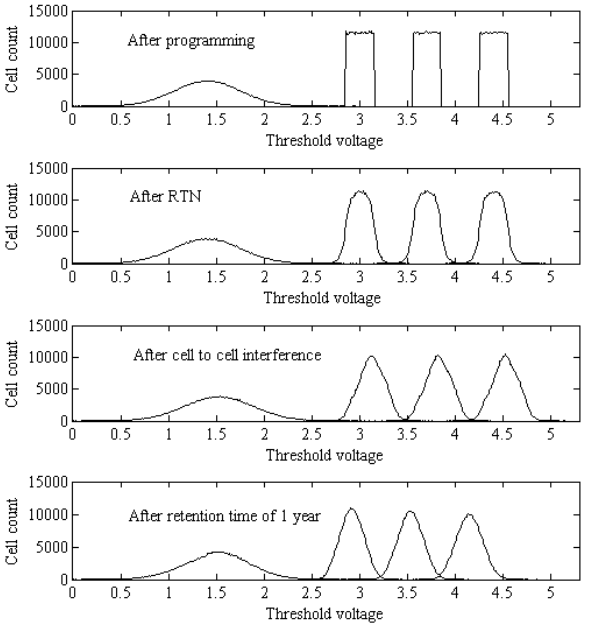


Figure 2: Simulated results to show the effects of RTN, cell-to-cell interference, and retention noise on memory cell threshold voltage distribution.

3 System Design Adaptive to PE Cycling

From the above discussions, it is clear that NAND flash memory cell raw storage reliability gradually degrades with the PE cycling: During the early lifetime of memory cells (i.e., the PE cycling number N is relatively small), the aggregated PE cycling effects are relatively small, which leads to a relatively large memory cell storage noise margin and hence good raw storage reliability (i.e., low raw storage bit error rate); since the aggregated PE cycling effects scale with N in approximate power-law fashions, the memory cell storage noise margin and hence raw storage reliability gradually degrade as the PE cycling number N increases. Given the target PE cycling endurance limit (e.g., 10K PE cycling), each memory word-line must have enough redundant memory cells so that the corresponding ECC can ensure the storage integrity as the PE cycling reaches the endurance limit. Due to the memory cell raw storage reliability dynamics, the redundancy geared to the worst-case scenario will *over-protect* the user data for most time throughout the entire memory lifetime, especially at its early lifetime when memory cell operational noise margin is much larger. This can be illustrated in Fig. 3, which clearly suggests that the redundant memory cells are essentially *under-utilized* at the memory early lifetime.

Very intuitively, we may trade such under-utilized redundancy to improve certain memory system performance metrics, which should be carried out adaptive to

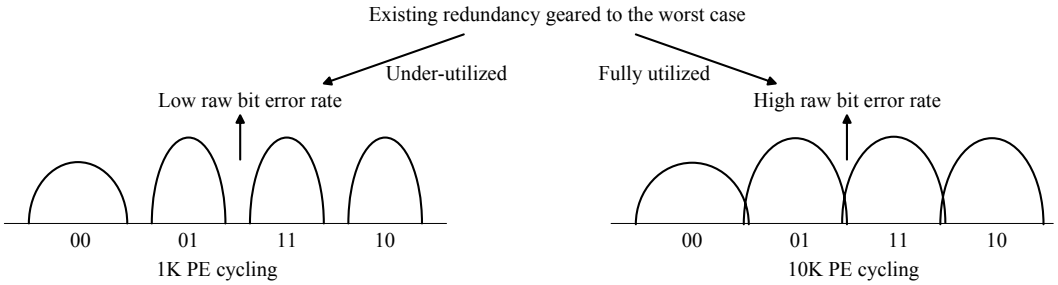


Figure 3: Illustration of the under-utilized ECC redundancy before reaching PE cycling endurance limit.

the memory PE cycling. In this work, we explore this adaptive memory system design space from two perspectives as discussed in the remainder of this section.

3.1 Approach I: Improve Memory Program Speed

In this subsection, we elaborate on the potential of trading the under-utilized ECC redundancy to improve average memory program speed. As discussed in Section 2.1, NAND flash memory program is carried out recursively by sweeping over the entire memory cell threshold voltage range with a program step voltage ΔV_{pp}. As a result, the memory program latency is inversely proportional to ΔV_{pp}, which suggests that we can improve the memory program speed by increasing ΔV_{pp}. However, a larger ΔV_{pp} directly results in a wider threshold voltage distribution of each programmed state, leading to less noise margin between adjacent programmed states and hence worse raw storage bit error rate. Therefore, there is an inherent trade-off between memory program speed vs. memory raw bit error rate, which can be configured by adjusting the program step voltage ΔV_{pp}. Since the memory cell noise margin is further degraded by the PE cycling effects as discussed above, a given ΔV_{pp} will result in different noise margin (hence different raw storage bit error rate) as memory cells undergo different amount of PE cycling.

In current design practice, ΔV_{pp} is fixed and its value is sufficiently small so that the ECC can tolerate the worst-case memory raw storage bit error rate as the PE cycling reaches its endurance limit. As a result, the memory program speed remains largely unchanged while the raw storage bit error rate gradually degrades. Before the PE cycling number reaches its endurance limit, the existing redundancy is under-utilized as pointed out in the above. Clearly, to eliminate such redundancy under-utilization, we could intentionally increase the the program step voltage ΔV_{pp} according to the run-time PE cycling number in such a way that the memory raw storage bit error rate is always close to what can be maximally tolerated by the existing redundancy. Therefore, the existing redundancy is always almost fully utilized, and meanwhile the dynamically increased ΔV_{pp} leads to higher average memory program speed. The above discussion can be further illustrated in Fig. 4.

Although it would be ideal if the program step voltage ΔV_{pp} can be smoothly adjusted with a very fine granularity, the limited reference voltage accuracy in real NAND flash memory chips may only enable the use of a few discrete program step voltages. Assume there are m different program step voltages, i.e., $\Delta V_{pp}^{(1)} > \Delta V_{pp}^{(2)} > \cdots > \Delta V_{pp}^{(m)}$. Given the existing ECC redundancy, we can obtain a sequence of PE cycling thresholds $N_0 = 0 < N_1 < \cdots < N_m$ so that, if the run-time PE cycling number falls into the range of $[N_{i-1}, N_i)$, we can use the program step voltage $V_{pp}^{(i)}$ and still ensure the overall system data storage integrity. If we follow the conventional design practice where the program step voltage is fixed according to the worst-case scenario, the smallest step voltage $\Delta V_{pp}^{(m)}$ will be used throughout the entire memory lifetime. Therefore, we can estimate the average program speed improvement over the entire memory lifetime as

$$s = 1 - \frac{\sum_{i=1}^{m} (N_i - N_{i-1}) \cdot \frac{1}{\Delta V_{pp}^{(i)}}}{N_m \cdot \frac{1}{\Delta V_{pp}^{(m)}}}. \qquad (10)$$

3.2 Approach II: Improve Memory Technology Scalability

In this subsection, we elaborate on the potential of trading the under-utilized ECC redundancy to improve memory defect tolerance. With the help of very sophisticated techniques such as double patterning [20], the decade-long 193nm photolithography has successfully pushed NAND flash memory into the sub-30nm region. However, as the industry is striving to push the NAND flash memory technology scaling into the sub-20nm region by using immersion photolithography or new lithography technologies such as nanoimprint, defects in such extremely dense memory arrays may inevitably increase.

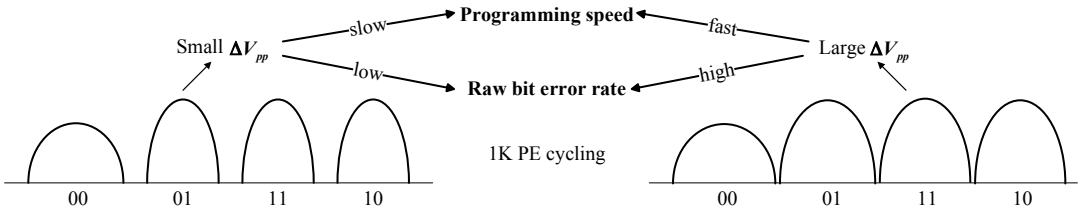


Figure 4: Illustration of the impact of program step voltage ΔV_{pp} on the program speed vs. raw storage bit error rate trade-off.

As a result, conventional spare row/column repair techniques may become inadequate to ensure a sufficiently high yield.

Very intuitively, the existing ECC redundancy can be leveraged to tolerate memory defects, especially random memory cell defects. However, if certain portion of ECC redundancy is used for defect tolerance, it will not be able to ensure the specified PE cycling limit, leading to PE cycling endurance degradation. Since all the pages in each memory block undergo the same number of PE cycling, the worst-case page (i.e., the page contains the most defects) in each block sets the achievable PE cycling endurance for this block. For example, assume the existing ECC redundancy can tolerate up to 50 errors for each page and survive up to 10K PE cycling in the absence of any memory cell defects. If the worst-case page in one block contains 5 defective cells, then it can only use the residual 45-error-correcting capability to tolerate memory operational noises such as PE cycling effects and cell-to-cell interference. Suppose this makes the worst-case page can only survive up to 8K PE cycling, this block can only be erased by 8K times instead of 10K times before risking data loss.

Clearly, if we attempt to reserve certain ECC redundancy for tolerating memory cell defects, we must minimize the impact on overall memory PE cycling endurance. In current design practice, NAND flash memory uses wear-leveling to uniformly spread program/erase operations among all the memory blocks to maximize the overall memory lifetime. Since different memory blocks with different amount of defective memory cells can survive different number of PE cycling, uniform wear-leveling is clearly not an optimal option. Instead, we should make wear-leveling fully aware of the different achievable PE cycling limits among different memory blocks, which is referred to as *differential wear-leveling*. This can be illustrated in Fig. 5: instead of uniformly distributing program/erase operations among all the memory blocks, the differential wear-leveling schedule the program/erase operations among all the memory blocks in proportional to their achievable PE cycling limits. As a result, we may largely improve the overall memory lifetime compared with uniform wear-leveling.

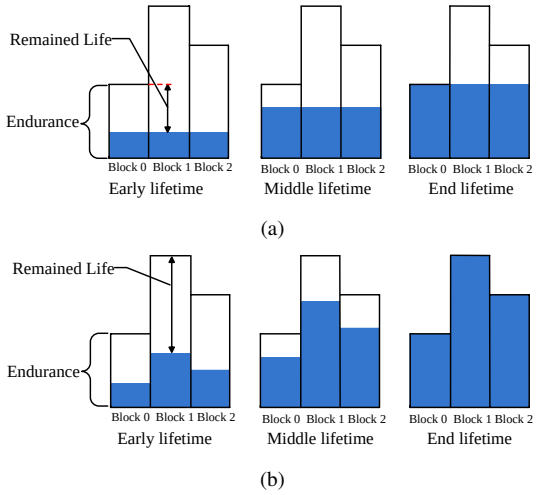


Figure 5: Illustration of (a) conventional uniform wear-leveling, and (b) proposed differential wear-leveling, where the ECC is used to tolerate defective memory cells and hence different blocks may have different achievable PE cycling endurance.

Assume the worst-case page can at most contains M defective memory cells, and let P_d denote the probability that the worst-case page in one block contains $d \in [0, M]$ defective memory cells. Given the number of defective memory cells in the worst-case page d, we can obtain the corresponding achievable PE cycling endurance limit $N^{(d)}$, i.e., the ECC can ensure a PE cycling number up to $N^{(d)}$ while tolerating d defective memory cells. Clearly, we have $N^{(0)} > N^{(1)} > \cdots > N^{(M)}$, where $N^{(0)}$ is the achievable PE cycling limit in the defect-free scenario. Define the *effective PE cycling endurance* as the average PE cycling limits of all the memory blocks. Under the uniform wear-leveling, the memory chip can only sustain PE cycling of $N^{(M)}$. Therefore, compared with the defect-free scenario, the effective PE cycling endurance degrades by $N^{(0)}/N^{(M)}$, which can result in a significant memory lifetime degradation. On the other hand, under the ideal differential wear-leveling, each block can reach its own PE cycling limit as illustrated in Fig. 5, hence the effective PE cycling endurance will be $\sum_{d=0}^{M} P_d \cdot N^{(d)}$,

representing the improvement of

$$\frac{\sum_{d=0}^{M} P_d \cdot N^{(d)}}{N^{(M)}} \tag{11}$$

over the uniform wear-leveling. We note that this design approach can be combined with the one presented in Section 3.1 to improve average memory program speed in the presence of memory cell defects. Given the number of defective memory cells d and the set of m program step voltage $\Delta V_{pp}^{(i)}$ for $1 \leq i \leq m$, we can obtain a set of PE cycling thresholds $N_0^d = 0 < N_1^d < \cdots < N_m^d$, i.e., if present PE cycling number falls into the range of $[N_{i-1}^{(d)}, N_i^{(d)})$, we can use the program step voltage $\Delta V_{pp}^{(i)}$ and meanwhile ensure the tolerance to d defective memory cells. Therefore, for the blocks whose worst-case page contains d defective memory cells, the average program speed improvement is

$$s_d = 1 - \frac{\sum_{i=1}^{m} (N_i^d - N_{i-1}^d) \cdot \frac{1}{\Delta V_{pp}^{(i)}}}{N_m^d \cdot \frac{1}{\Delta V_{pp}^{(m)}}}. \tag{12}$$

The overall average program speed improvement can be further estimated as $\sum_{i=1}^{M} P_i \cdot s_i$.

4 Evaluation Results

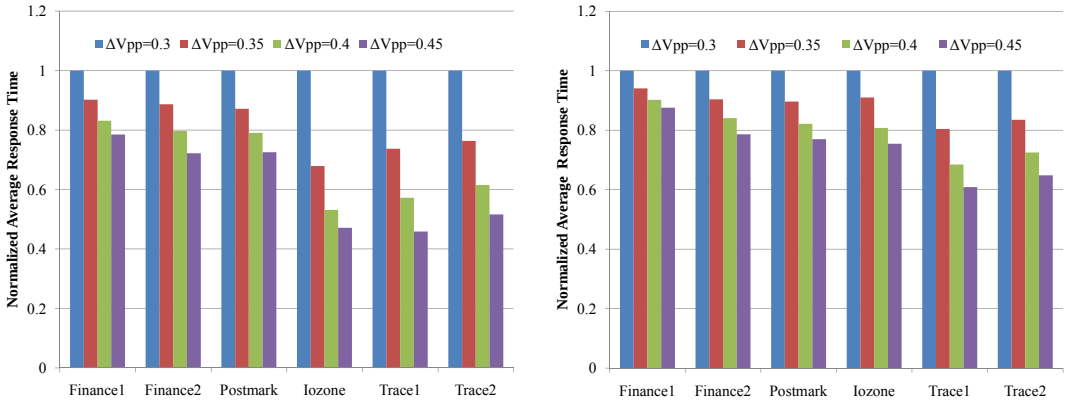
We carried out simulations and analysis to future demonstrate the effectiveness of the above two simple design approaches and their combination. To carry out trace-based simulations, we use the SSD module [3] in DiskSim [8], and use 6 workload traces including Iozone and Postmark [3], Finance1 and Finance2 from [1], and Trace1 and Trace2 from [16]. The simulator can support the use of several parallel packages that can work in parallel to improve the SSD throughput. Each package contains 2 dies that share an 8-bit I/O bus and a number of common control signals, and each die contains 4 planes and each plane contains 2048 blocks. Each block contains 64 4KB pages, each of which consists of 8 512B sectors. Following the version 2.1 of the Open NAND Flash Interface (ONFI) [2], we set the NAND flash chip interface bus frequency as 200MB/s. Regarding the ECC, we assume that binary (n, k, t) BCH codes are being used, where n is the codeword length, k is the user data length (i.e., 512B in this study), and t is the error-correcting capability. We consider the use of 2bit/cell NAND flash memory, and set the baseline 2bit/cell NAND flash memory using the equivalent memory channel model parameters presented in Example 2.1 in Section 2.4, for which a (4798, 4096, 54) BCH code can ensure a PE cycling endurance limit of 10K under the retention time of 1 year. We note that the target NAND flash memory retention time is fixed as 1 year throughout all the studies in this work.

In this section, we first present trace-based simulation results to demonstrate how the first design approach can reduce the overall request response time and hence improve SSD speed performance. Then, we present analysis results to demonstrate the second design approach by assuming memory cell defects follow Poisson distribution. Finally, we demonstrate the effectiveness when these two approaches are combined together to improve SSD speed performance in the presence of memory cell defects.

4.1 Improve SSD Speed Performance

In the baseline scenario with the parameters listed in Example 2.1, the normalized program step voltage ΔV_{pp} is 0.3. As discussed in Section 3.1, we can use larger-than-worst-case ΔV_{pp} over the memory lifetime to improve memory program speed by exploiting the memory device wear-out dynamics. In this work, we assume that memory chip voltage generators can increase ΔV_{pp} with a step of 0.05, hence we consider four different normalized values: $\Delta V_{pp}^{(1)} = 0.45$, $\Delta V_{pp}^{(2)} = 0.4$, $\Delta V_{pp}^{(1)} = 0.35$, and $\Delta V_{pp}^{(4)} = 0.3$. By carrying out Monte Carlo simulations without changing the other memory model parameters, we have that these four different program step voltages can survive up to $N_1 = 2710$, $N_2 = 4820$, $N_3 = 7500$, and $N_4 = 10000$ PE cycling, respectively, under the retention time of 1 year. Therefore, according to Eq.(10), the average NAND flash memory program speed can be improved by 18% compared with the baseline scenario. We further carried out DiskSim-based simulations to investigate how such improved memory program speed can reduce the SSD average response time (incorporating both write and read request response time) for different traces under different system configurations. We set that the 2bit/cell NAND flash memory program latency as 600μs when the normalized program step voltage ΔV_{pp} is 0.3, on-chip memory sensing latency as 30μs, and erase time as 3ms.

In this study, we consider the use of 4 and 8 parallel packages. Fig. 6 compares the normalized SSD average response time when using 4 and 8 parallel packages, respectively, where we set ΔV_{pp} as 0.3. It shows that using more parallel packages can directly improve SSD speed performance, which can be intuitively justified. Fig. 7(a) and Fig. 7(b) show the normalized SSD average response time under the 4 different normalized program step voltage ΔV_{pp} for all the 6 traces when the SSD contains 4 and 8 parallel packages, respectively. We use the first-come first-serve (FCFS) scheduling scheme in the simulations. Compared with the baseline scenario with $\Delta V_{pp} = 0.3$, the average response time can be reduced by up to $\sim$50%



(a) (b)

Figure 7: Simulated normalized average response time when the SSD contains (a) 4 parallel packages, and (b) 8 parallel packages.

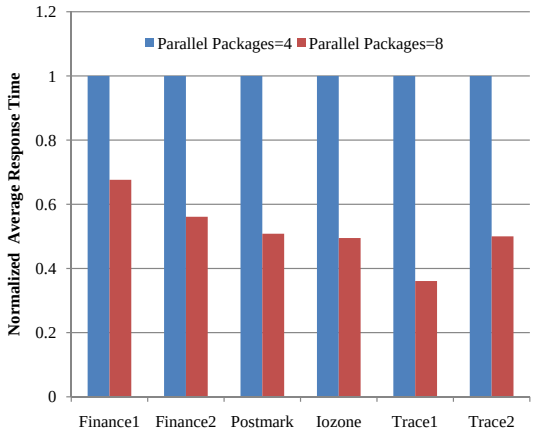


Figure 6: Comparison of normalized SSD average response time with 4 and 8 parallel packages ($\Delta V_{pp} = 0.3$).

with 4 parallel packages and up to ∼40% with 8 parallel packages. The results show that the use of larger program step voltage can consistently improve SSD speed performance under different number of parallel packages.

Given the PE cycling thresholds N_i for $i = 1,2,3,4$ as presented in the above, the NAND flash memory should employ the program step voltage $\Delta V_{pp}^{(i)}$ when the present PE cycling number falls into $[N_{i-1}, N_i)$, where N_0 is set to 0. Therefore, based on the the simulation results shown in Fig. 7, we can obtain the overall SSD average response time reduction compared with the baseline scenario, as shown in Fig. 8. It shows that this proposed design approach can noticeably improve the overall SSD speed performance. Intuitively, those traces with higher write request ratios (e.g., Iozone, Trace1, and Trace2) tend to benefit more from this design approach, as shown in Fig. 8. In addition, as we increase the package parallelism from 4 to 8, the overall response time reduction consistently reduces over all the traces. This can be explained as follows: As the SSD contains more parallel packages, the increased architecture-level parallelism will directly improve SSD speed performance, as illustrated in Fig. 6. As a result, this will make the improvement on the device-level program speed become relatively less significant with respect to the improvement of overall system speed performance.

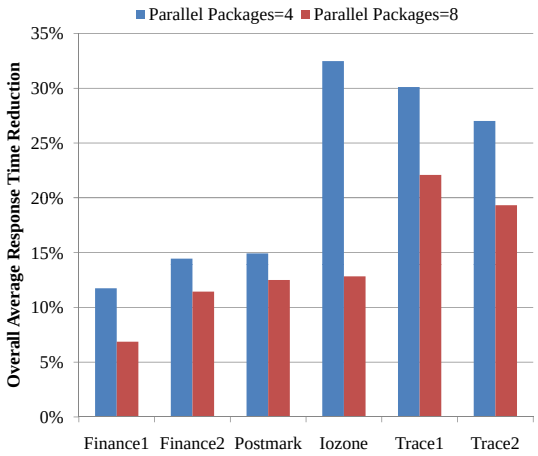


Figure 8: Overall SSD average response time reduction compared with the baseline scenario when using 4 and 8 parallel packages.

In the above simulations, the FCFS scheduling scheme has been used. To study the sensitivity of this design approach to different scheduling schemes, we repeat the above simulations using two other popular scheduling schemes including ELEVATOR and SSTF (shortest seek time first) [43]. Fig. 9 shows the overall SSD average

response time reduction compared with the baseline scenario, where the SSD contains 4 parallel packages. The results show the proposed design approach can consistently improve overall SSD speed performance under different scheduling schemes.

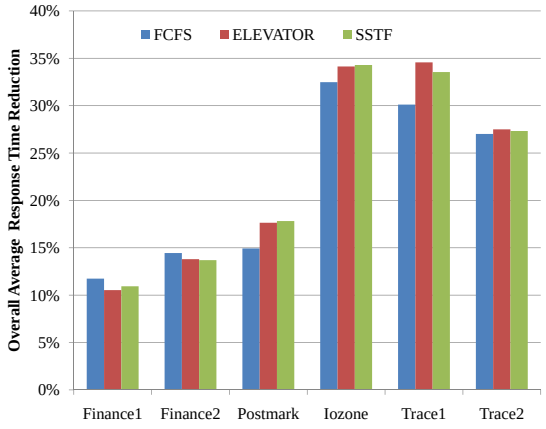


Figure 9: Overall SSD average response time reduction compared with the baseline scenario under different scheduling schemes.

4.2 Improve Defect Tolerance

To demonstrate the proposed design approach for improving memory defect tolerance, we assume that the number of defective memory cells in each worst-case page follows a Poisson distribution that is widely used to model defects in integrated circuits. Therefore, under the Poisson-based distribution model, the probability that the worst-case page in each block contains d defective memory cells is $f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$, where the parameter λ is the mean of the number of defective memory cells in each worst-case page. Given the parameter λ, we find the value M so that $\sum_{i=0}^{M} f(i;\lambda) \geq 0.999$, and assume that any blocks whose worst-case page contains more than M defective memory cells can be replaced by a redundant block. In this work, we consider the mean λ ranging from 1 to 4, and accordingly have that the maximum value of M is 12.

Using the baseline NAND flash memory model parameters as listed in Example 2.1, we can obtain the achievable PE cycling limit $N^{(d)}$ for each d, i.e., we use the (4798, 4096, 54) BCH code to tolerate d defective memory cells and meanwhile use its residual $(54-d)$-error-correcting capability to ensure a PE cycling endurance limit of $N^{(d)}$ under the retention time of 1 year. Fig. 10 shows the achievable PE cycling limit $N^{(d)}$ with d ranging from 0 to 12. Under different value of mean λ, we have different value of M, denoted as $M^{(\lambda)}$. When the uniform wear-leveling is being used, the effective PE

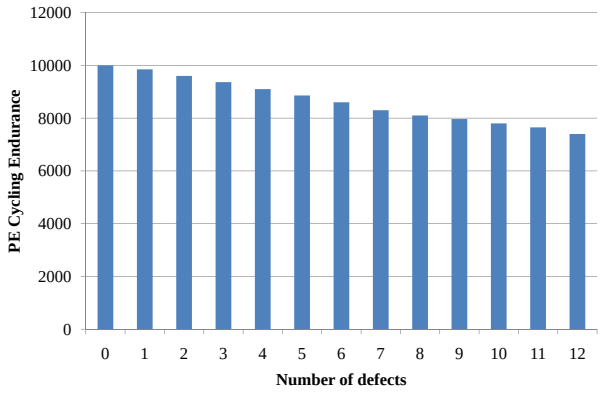


Figure 10: Achievable PE cycling endurance under different value of defective memory cells in the worst-case page.

cycling endurance is simply $N^{(d)}$ when $d = M^{(\lambda)}$. When the proposed differential wear-leveling is being used, the effective PE cycling endurance is

$$\sum_{d=0}^{M^{(\lambda)}} \frac{\lambda^k e^{-\lambda}}{k!} \cdot N^{(d)}, \tag{13}$$

for a given mean λ. Fig. 11 shows the effective PE cycling endurance when these two different wear-leveling schemes are being used under different value of λ. The results show that the proposed differential wear-leveling can noticeably improve the effective PE cycling endurance and hence SSD lifetime compared with uniform wear-leveling. As the defects density increases (i.e., λ

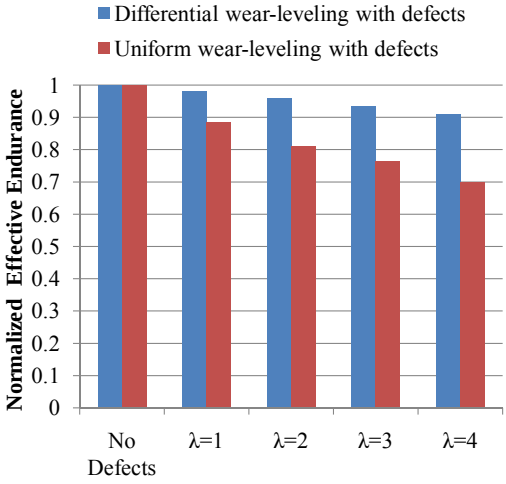


Figure 11: Effective PE cycling endurance when using uniform wear-leveling and differential wear-leveling under different value of λ.

increases), the gain of differential wear-leveling over uniform wear-leveling will accordingly improve (i.e., from about 10% improvement at $\lambda = 1$ to about 30% improvement at $\lambda = 4$).

4.3 Combination of the Two Design Approaches

As discussed earlier, we can combine the proposed two design approaches in order to improve SSD speed performance when ECC is also used to tolerate defective memory cells. Following the discussions in Section 4.2, we assume that the number of defective memory cells in the worst-case page has a Poisson distribution and consider the cases when the mean λ ranges from 1 to 4. Following the discussions in Section 4.1, beyond the normalized program step voltage ΔV_{pp} of 0.3 in the baseline scenario, we consider three larger values of ΔV_{pp}, including 0.35, 0.4, and 0.45. Denote $\Delta V_{pp}^{(1)} = 0.45$, $\Delta V_{pp}^{(2)} = 0.4$, $\Delta V_{pp}^{(3)} = 0.35$, and $\Delta V_{pp}^{(4)} = 0.3$. Given the memory cell defects number d and the (4798, 4096, 54) BCH code being used, we can obtain a set of PE cycling thresholds $N_0^d = 0 < N_1^d < \cdots < N_4^d$ so that, if present PE cycling number falls into the range of $[N_{i-1}^{(d)}, N_i^{(d)})$, we can use the program step voltage $\Delta V_{pp}^{(i)}$ and meanwhile ensure the tolerance of d defective memory cells. Fig. 12 shows the PE cycling thresholds when the defect number increases from 0 to 12. The results can be intuitively justified: as the defect number increases, the residual ECC error-correcting capability degrades, and consequently the larger program step voltage can only be used over a less number of PE cycling.

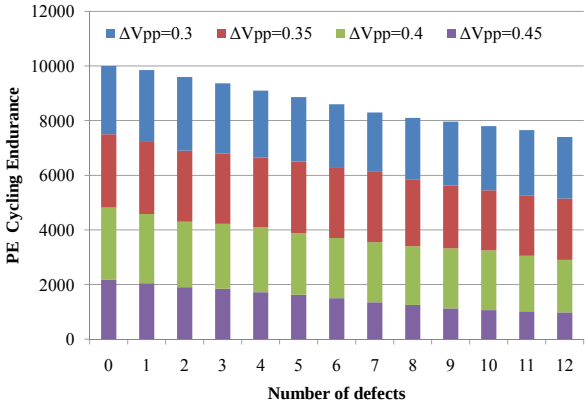


Figure 12: PE cycling thresholds corresponding to different number of defective cells in the worst-case page of one block.

Given each program step voltage $\Delta V_{pp}^{(i)}$, we can obtain the normalized SSD response time τ_i for each specific trace, as shown in Fig. 7. Recall that, when the PE cycling number falls into the range $[N_{i-1}^{(d)}, N_i^{(d)})$, we can use the program step voltage $\Delta V_{pp}^{(i)}$, and the baseline scenario fixes the program step voltage as $\Delta V_{pp}^{(4)} = 0.3$ throughout the entire memory lifetime. Therefore, we can calculate the overall SSD average response time reduction over the baseline scenario for each trace as

$$\sum_{d=0}^{12} f(d;\lambda) \frac{\sum_{i=1}^{4} (N_i^{(d)} - N_{i-1}^{(d)}) \cdot \tau_i}{N_4^{(d)} \cdot \tau_4}, \tag{14}$$

and the results are shown in Fig. 13. The results suggest that we still can maintain a noticeable SSD speed performance improvement when ECC is also used to tolerate defective memory cells.

5 Related Work

NAND flash memory system design has attracted many recent attentions, where most work focused on improving system speed performance and endurance. Dirik and Jacob [16] studied the effect on SSD system speed performance by changing various SSD system parallelism and concurrency at different levels such as the numbers of planes on each channel and the number of channels, and compared various existing disk access scheduling algorithms. Agrawal *et al.* [3] analyzed the effect of page size, striping and interleaving policy on the memory system performance, and proposed a conception of *gang* as a higher-level "superblock" to facilitate SSD system-level parallelism configurations. Min and Nam [32] developed several NAND flash memory performance enhancement techniques such as write request interleaving. Seong *et al.* [37] applied bus-level and chip-level interleaving to exploit the inherent parallelism in multiple flash memory chips to improve the SSD speed performance. The authors of [11,13] applied adaptive bank scheduling policies to achieve an even distribution of write request and load balance to improve system speed performance.

Wear-leveling is used to improve NAND flash memory endurance. Gal and Toledo [18] surveyed many patented and published wear-leveling algorithms and data structures for NAND flash memory. Ben-Aroya and Toledo [5] more quantitatively evaluated different wear-leveling algorithms, including both on-line and off-line algorithms. The combination of wear-leveling and garbage collection and the involved design trade-offs have been investigated by many researchers, e.g., see [12, 14, 22, 23, 44]. In current design practice, defect tolerance has been mainly realized by bad block management that run-time monitors and disables the future use of blocks with defects. Traditional redundant repair can also be used to compensate certain memory defects,

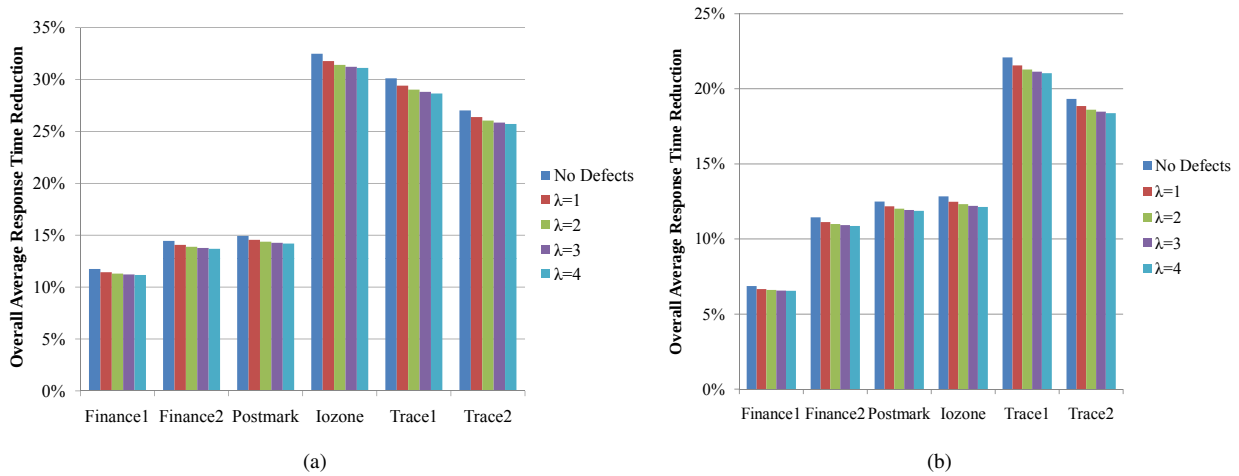


Figure 13: Overall average response time reduction over the baseline scenario under different λ when SSD contains (a) 4 parallel packages and (b) 8 parallel packages.

e.g., see [19]. In addition, a NAND flash memory device model was presented in [33], which nevertheless does not take into account of RTN noise and cell-to-cell interference, and the model was used to show that time-dependent trap recovery can be leveraged to improve memory endurance.

We note that most prior work on improving SSD system speed performance and/or memory endurance are carried out mainly from architecture/system perspective to combat flash memory device issues. To the best of our knowledge, this paper represents the first attempt to adaptively exploit flash memory device characteristics, in particular PE-cycling-dependent device wear-out dynamics, at the system level to improve SSD system speed performance and NAND flash memory scalability. The proposed design approaches are completely orthogonal to prior architecture/system level techniques and can be readily combined together.

6 Conclusion

This paper investigates the potential of adaptively leveraging NAND flash memory cell wear-out dynamics to improve memory system performance. As memory PE cycling increases, NAND flash memory cell storage noise margin and hence raw storage reliability accordingly degrade. Therefore, the specified PE cycling endurance limit determines the worst-case raw memory storage reliability, which further sets the amount of redundant memory cells that must be fabricated. Motivated by the fact that such worst-case oriented redundancy is essentially under-utilized over the entire memory lifetime, especially when the PE cycling number is relatively small, this paper proposes to trade such under-utilized redundancy to improve system speed performance and/or tolerate defective memory cells. We further propose a simple differential wear-leveling scheme to minimize the impact on PE cycling endurance if the redundancy is used to tolerate defective memory cells. To quantitatively evaluate such adaptive NAND flash memory system design strategies, we first develop an approximate NAND flash memory device model that can capture the effects of PE cycling on memory cell storage reliability. To evaluate the effectiveness on improving memory system speed, we carry out extensive simulations over a variety of traces using the DiskSim-based SSD simulator under different system configurations, and the results show up to 32% SSD average response time reduction can be achieved. To evaluate the effectiveness on defect tolerance, with a Poisson-based defect statics model, we show that this design strategy can tolerate relatively high defect rates at small degradation of effective PE cycling endurance. Finally, we show that these two aspects can be combined together so that we could noticeably reduce SSD average response time even in the presence of high memory defect densities. generate the the references with alphatical order.

Acknowledgments

We thank Albert Fazio at Intel for his valuable comments on NAND flash memory device modeling. We also thank the anonymous reviewers and our shepherd Eno Thereska for their feedback.

References

[1] "SPC Trace File Format Specification. http://traces.cs.umass.edu/index.php/Storage/Storage,

Last accessed on June 6, 2010," Storage Performance Council, Tech. Rep. Revision 1.0.1, 2002.

[2] "Open NAND Flash Interface Specification," Hynix Semiconductor and Intel Corporation and Micron Technology, Inc. and Numonyx and Phison Electronics Corp. and Sony Corporation and Spansion, Tech. Rep. Revision 2.1, Jan. 2009.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," in *Proc. of USENIX Annual Technical Conference*, 2008, pp. 57–70.

[4] S. Aritome, R. Shirota, G. Hemink, T. Endoh, and F. Masuoka, "Reliability Issues of Flash Memory Cells," *Proceedings of the IEEE*, vol. 81, no. 5, pp. 776–788, 1993.

[5] A. Ben-Aroya and S. Toledo, "Competitive Analysis of Flash-Memory Algorithms," in *Proc. of the Annual European Symposium*, 2006, pp. 100–111.

[6] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to Flash memory," *Proceedings of the IEEE*, vol. 91, pp. 489–502, April 2003.

[7] R. Bez and P. Cappelletti, "Flash Memory and Beyond," in *Proc. of IEEE VLSI-TSA International Symposium on VLSI Technology*, 2005, pp. 84–87.

[8] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The DiskSim Simulation Environment Version 4.0 Reference Manual," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-08-101, May 2008.

[9] P. Cappelletti, R. Bez, D. Cantarelli, and L. Fratin, "Failure Mechanisms of Flash Cell in Program/erase Cycling," in *Proc. of International Electron Devices Meeting (IEDM)*, 1994, pp. 291–294.

[10] R.-A. Cernea and et al., "A 34 MB/s MLC Write Throughput 16 Gb NAND With All Bit Line Architecture on 56 nm Technology," *IEEE J. Solid-State Circuits*, vol. 44, pp. 186–194, Jan. 2009.

[11] L.-P. Chang and T.-W. Kuo, "An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems," *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 187 – 196, 2002.

[12] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, "Real-time Garbage Collection for Flash-Memory Storage Systems of Real-time Embedded Systems," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 4, pp. 837–863, 2004.

[13] Y.-B. Chang and L.-P. Chang, "A Self-Balancing Striping Scheme for NAND-Flash Storage Systems," in *Proc. of the ACM Symposium on Applied Computing*, 2008, pp. 1715–1719.

[14] M.-L. Chiang and R.-C. Chang, "Cleaning Policies in Mobile Computers Using Flash Memory," *Journal of Systems and Software*, vol. 48, no. 3, pp. 213–231, 1999.

[15] C. Compagnoni, M. Ghidotti, A. Lacaita, A. Spinelli, and A. Visconti, "Random Telegraph Noise Effect on the Programmed Threshold-Voltage Distribution of Flash Memories," *IEEE Electron Device Letters*, vol. 30, no. 9, 2009.

[16] C. Dirik and B. Jacob, "The Performance of PC Solid-state disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 279–289, 2009.

[17] K. Fukuda, Y. Shimizu, K. Amemiya, M. Kamoshida, and C. Hu, "Random Telegraph Noise in Flash Memories - Model and Technology Scaling," in *Proc. of IEEE International Electron Devices Meeting (IEDM)*, 2007, pp. 169–172.

[18] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138–163, 2005.

[19] Y.-Y. Hsiao, C.-H. Chen, and C.-W. Wu, "Built-In Self-Repair Schemes for Flash Memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 8, pp. 1243 –1256, Aug. 2010.

[20] B. Hwang and et al, "Comparison of Double Patterning Technologies in NAND Flash Memory With Sub-30nm Node," in *Proc. of the European Solid State Device Research Conference (ESSDERC)*, 2009, pp. 269 –271.

[21] T. Jung and et al., "A 117-mm 3.3-V only 128-Mb Multilevel NAND Flash Memory for Mass Storage Applications," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1575–1583, Nov. 1996.

[22] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," in *Proc. of the USENIX Technical Conference*, 1995, pp. 13–13.

[23] H. Kim and S. Lee, "An Effective Flash Memory Manager for Reliable Flash Memory Space Management," *IEICE Transactions on Information and Systems*, vol. 85, no. 6, pp. 950–964, 2002.

[24] K. Kim and et.al, "Future Memory Technology: Challenges and Opportunities," in *Proc. of International Symposium on VLSI Technology, Systems and Applications*, Apr. 2008, pp. 5–9.

[25] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of Floating-Gate Interference on NAND Flash Memory Cell Operation," *IEEE Electron Device Letters*, vol. 23, no. 5, pp. 264–266, May. 2002.

[26] J. Lee, J. Choi, D. Park, and K. Kim, "Data Retention Characteristics of sub-100 nm NAND Flash Memory Cells," *IEEE Electron Device Letters*, vol. 24, no. 12, pp. 748–750, 2003.

[27] J. Lee, J. Choi, D. Park, K. Kim, R. Center, S. Co, and S. Gyunggi-Do, "Effects of Interface Trap Generation and Annihilation on the Data Retention Characteristics of Flash Memory Cells," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 1, pp. 110–117, 2004.

[28] Y. Li and et. al, "A 16Gb 3b/Cell NAND Flash Memory in 56nm with 8MB/s Write Rate," in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2008, pp. 506–632.

[29] H. Liu, S. Groothuis, C. Mouli, J. Li, K. Parat, and T. Krishnamohan, " 3D Simulation Study of Cell-Cell Interference in Advanced NAND Flash Memory," in *Proc. of IEEE Workshop on Microelectronics and Electron Devices*, April 2009.

[30] N. Mielke, H. Belgal, I. Kalastirsky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu, "Flash EEPROM Threshold Instabilities Due to Charge Trapping During Program/erase Cycling," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 3, pp. 335–344, 2004.

[31] N. Mielke, H. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery Effects in the Distributed Cycling of Flash Memories," in *Proc. of IEEE International Reliability Physics Symposium*, 2006, pp. 29–35.

[32] S. L. Min and E. H. Nam, "Current Trends in Flash Memory Technology," *Proc. of Asia and South Pacific Conference on Design Automation.*, p. 2., Jan. 2006.

[33] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan, "How I Learned to Stop Worrying and Love Flash Endurance," in *Proc. of the 2nd USENIX conference on Hot topics in storage and file systems*, 2010, pp. 3–3.

[34] P. Olivo, B. Ricco, and E. Sangiorgi, "High Field Induced Voltage Dependent Oxide Charge," *Applied Physics Letter*, vol. 48, pp. 1135–1137, 1986.

[35] K.-T. Park and et al., "A Zeroing Cell-to-Cell Interference Page Architecture With Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories," *IEEE J. Solid-State Circuits*, vol. 40, pp. 919–928, Apr. 2008.

[36] K. Prall, "Scaling Non-Volatile Memory Below 30nm," in *Proc. of IEEE Non-Volatile Semiconductor Memory Workshop*, Aug. 2007, pp. 5–10.

[37] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J. Choi, S. Lee, Y. H. Bae, J. Lee, Y. Cho, and S. L. Min, "Hydra: A Block-Mapped Parallel Flash Memory Solid-State Disk Architecture," *IEEE Transactions on Computers*, vol. 59, no. 7, pp. 905 –921, Jul. 2010.

[38] N. Shibata and et al., "A 70 nm 16 Gb 16-level-cell NAND flash memory," in *Proc. of IEEE Symposium on VLSI Circuits*, 2007, pp. 190–191.

[39] K.-D. Suh and et al., "A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.

[40] K. Takeuchi and et al., "A 56-nm CMOS 99-mm^2 8-Gb Multi-Level NAND Flash Memory With 10-MB/s Program Throughput," *IEEE J. Solid-State Circuits*, vol. 42, pp. 219–232, Jan. 2007.

[41] K. Takeuchi, T. Tanaka, and H. Nakamura, "A Double-level-Vth Select Gate Array Architecture for Multilevel NAND Flash Memories," *IEEE J. Solid-State Circuits*, vol. 31, no. 4, pp. 602–609, Apr. 1996.

[42] D. Wellekens, J. Van Houdt, L. Faraone, G. Groeseneken, and H. Maes, "Write/erase Degradation in Source Side Injection Flash EEPROM's: Characterization Techniques and Wearout Mechanisms," *IEEE Transactions on Electron Devices*, vol. 42, no. 11, pp. 1992–1998, 1995.

[43] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling Algorithms for Modern Disk Drives," in *Proc. of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 1994, pp. 241–251.

[44] M. Wu and W. Zwaenepoel, "eNVy: a NonVolatile Main Memory Storage System," *Proc. of Fourth Workshop on Workstation Operating Systems*, pp. 116 –118, Oct. 1993.

[45] H. Yang and et al., "Reliability Issues and Models of sub-90nm NAND Flash Memory Cells," in *Proc. of International Conference on Solid-State and Integrated Circuit Technology*, 2006, pp. 760–762.