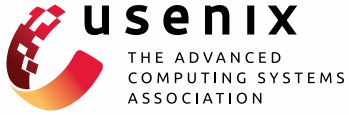

How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime?

Jiangpeng Li, Kai Zhao, and Xuebin Zhang, ***Rensselaer Polytechnic Institute;***
Jun Ma, ***Shanghai Jiao Tong University;*** **Ming Zhao,** ***Florida International University;***
Tong Zhang, ***Rensselaer Polytechnic Institute***

https://www.usenix.org/conference/fast15/technical-sessions/presentation/li

This paper is included in the Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15).

February 16–19, 2015 • Santa Clara, CA, USA

ISBN 978-1-931971-201

Open access to the Proceedings of the 13th USENIX Conference on File and Storage Technologies is sponsored by USENIX

How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime?

Jiangpeng Li*, Kai Zhao*, Xuebin Zhang*, Jun Ma†, Ming Zhao‡and Tong Zhang*
*ECSE department, Rensselaer Polytechnic Institute, USA
†Shanghai Jiao Tong University, China
‡School of Computing and Information Sciences, Florida International University, USA
{lijiangpeng1984@gmail.com, tzhang@ecse.rpi.edu}

Abstract

Although data compression can benefit flash memory lifetime, little work has been done to rigorously study the full potential of exploiting data compressibility to improve memory lifetime. This work attempts to fill this missing link. Motivated by the fact that memory cell damage strongly depends on the data content being stored, we first propose an implicit data compression approach (i.e., compress each data sector but do not increase the number of sectors per flash memory page) as a complement to conventional explicit data compression that aims to increase the number of sectors per flash memory page. Due to the runtime variation of data compressibility, each flash memory page almost always contains some unused storage space left by compressed data sectors. We develop a set of design strategies for exploiting such unused storage space to reduce the overall memory physical damage. We derive a set of mathematical formulations that can quantitatively estimate flash memory physical damage reduction gained by the proposed design strategies for both explicit and implicit data compression. Using 20nm MLC NAND flash memory chips, we carry out extensive experiments to quantify the content dependency of memory cell damage, based upon which we empirically evaluate and compare the effectiveness of the proposed design strategies under a wide spectrum of data compressibility characteristics.

1 Introduction

NAND flash memory cells gradually wear out with program/erase (P/E) cycling due to physical device damage caused by each P/E cycle, and cycling endurance drastically degrades with the technology scaling down. Hence, how to maximize memory lifetime has been widely studied from different aspects, e.g., signal processing and error correction coding (ECC) [1–3], flash translation layer (FTL) [4–10], and system software stack [11–13].

Nevertheless, to our best knowledge, no prior work has thoroughly studied how data compressibility can be leveraged to improve flash memory lifetime. It is actually not surprising, since this question appears to be trivial at first glance: In conventional practice, the sole objective of data compression is to improve storage efficiency (i.e., explicitly increase the number of data sectors that can be stored in one flash memory page). This is referred to as *explicit data compression* in this work. Due to the runtime variation of data compressibility, explicit data compression results in heterogeneity among flash memory pages in terms of the number of sectors per page, which can complicate FTL and/or file system design. As a result, it is not uncommon that commercial flash-based storage devices do not use data compression at all. If sophisticated FTL and/or file systems, which can employ explicit compression to improve storage efficiency, are indeed available, one may simply expect that storing data with an average compression ratio[1] of α can directly improve the flash memory lifetime by $1/\alpha$. Therefore, one may easily draw the following conclusion: If we do not want to complicate the FTL and/or file system, we should simply leave the user data uncompressed, for which the data compressibility is totally irrelevant to flash memory lifetime; If we use complicated FTL and/or file systems to support explicit data compression, the flash memory lifetime improvement solely depends on the average data compression ratio.

This work contends that the above intuitive conclusion is far from revealing the complete potential of how data compressibility can help to improve flash memory lifetime. In essence, it overlooks two factors. First, flash memory experiences *content-dependent memory damage*, i.e., the damage suffered by each memory cell depends on its content (e.g., '11', '10', '00', and '01' in MLC flash memory) being stored. Once data compression leaves some unused storage space within flash

[1] Let S_o and S_c denote the size of the original and compressed data, then we define *compression ratio* as S_c/S_o, which falls into $(0,1]$.

memory pages, we can manipulate their data content in a *damage-friendly* manner to reduce physical damage. Hence, conventional explicit data compression is not necessarily the only option of exploiting data compressibility to improve memory lifetime. We propose *implicit data compression* as an alternative to complement with explicit data compression. With implicit data compression, we compress each data sector but do not increase the number of data sectors per flash memory page. Therefore, implicit compression has no impact on FTL and/or file system but meanwhile does not improve storage efficiency either. Second, for multi-bit per cell (e.g., MLC and TLC) flash memory, physical damage depends on a variety of factors (e.g., distribution characteristics of compressed data size, relative placement or layout of different pages on the same memory wordline), which have not been considered in prior work.

This paper presents a thorough study on exploiting data compressibility to reduce physical damage and hence improve flash memory lifetime. Since random read latency is one of the most important metrics of flash-based storage devices, this work assumes that each compressed data sector must reside entirely in one flash memory page. As a result, each flash memory page almost always contains some unused storage space left by compressed data sectors. Motivated by the content dependency of flash memory cell damage, we present a set of design strategies that can exploit the unused storage space within a flash memory page to reduce the overall memory damage, for both explicit and implicit data compression. Then we derive a set of mathematical formulations for quantitatively estimating flash memory damage reduction gained by the proposed design strategies. These rigorous mathematical formulations build a framework that directly links flash memory lifetime with data compressibility characteristics (e.g., mean and deviation of data compression ratio) and memory cell damage content dependency. Using 20nm MLC NAND flash memory chips, we carried out experiments to quantitatively measure the content-dependent memory cell damage factors, based upon which we empirically evaluated and compared the effectiveness of the proposed design strategies with either explicit or implicit compression. In summary, the main contributions of this work include:

1. We propose an implicit data compression strategy as a viable complement to conventional explicit data compression for exploiting data compressibility to improve flash memory lifetime;

2. A set of design strategies are developed to leverage the unused storage space left by data compression within flash memory pages to reduce the memory cell physical damage;

3. We derive a set of mathematical formulations to accurately estimate the flash memory damage based upon the characteristics of data compressibility and content-dependent memory cell damage;

4. We quantitatively compare explicit data compression and implicit data compression under a wide spectrum of runtime data compressibility characteristics and show that it is important to fully understand the data compressibility characteristics in order to choose the appropriate design strategy.

Finally, we note that, although this work focuses on flash memory, the developed design strategies and mathematical formulations are readily applicable to other emerging memory technologies, e.g., PCM and ReRAM, that experience similar content dependency of memory cell physical damage.

2 Design Strategies

This section presents a set of design strategies that can exploit data compressibility to reduce memory cell damage. We first discuss the content dependency of cycling-induced memory damage that motivates us to propose implicit compression (i.e., compress the data without increasing the number of sectors per flash memory page) in addition to the conventional explicit compression (i.e., compress the data and increase the number of sectors per page as much as possible). We further present different strategies on laying out the compressed data within flash memory pages that aim to leverage the content-dependent memory damage phenomenon for improving flash memory lifetime. We note that this work only focuses on MLC memory, and the discussions could be readily extended to the more complicated TLC case.

2.1 Content-Dependent Damage

NAND flash memory handles data programming and read in page units with a typical size of 4kB or 8kB. For high-density MLC and TLC memory, different bits within each MLC/TLC memory cell belong to different pages. This can be illustrated in Fig. 1 for MLC flash memory, where the two bits within each memory cell belong to lower and upper pages, respectively.

NAND flash memory cells wear out with P/E cycling due to the oxide damage caused by the electrons that pass through the gate oxide during each P/E cycle. Although current practice estimates the memory cell damage solely dependent upon the number of P/E cycles endured by memory cells, actual physical damage further depends on the data content being programmed to memory cells.

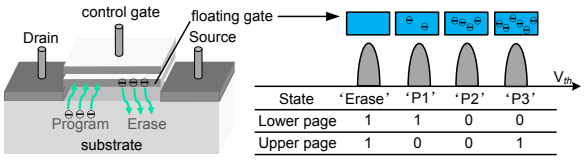

Figure 1: Illustration of MLC NAND flash memory cell.

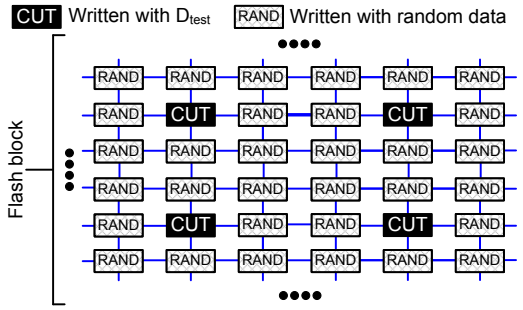

Figure 2: Illustration of method for accurately quantifying the flash memory cell damage caused by each distinct data content.

This can be intuitively explained using Fig. 1: different data content (e.g., '11', '10', '00', and '01') correspond to different number of electrons that pass through the gate oxide, and hence different amount of physical damage [14, 15].

To further demonstrate such content dependency, we carried out experiments using 20nm MLC NAND flash memory chips. To evaluate the effect of writing the content $D_{test} \in \{$'11','10','00','01'$\}$, we program each flash memory block with the pattern as shown in Fig. 2, i.e., to examine the cell content D_{test}, each memory cell written with D_{test} is surrounded by memory cells written with random data. This can incorporate the effect of cell-to-cell interference and program disturb in practice. The memory cells written with D_{test} are called *cells under test* (CUT). Different memory blocks are used for testing different D_{test}, and the locations of all the CUTs are fixed throughout the entire cycling. We capture the raw bit error rate (BER) of all the CUTs every few hundreds cycles by writing random data to all the memory cells. The measurement results are shown in Fig. 3.

2.2 Data Storage Schemes

Since each compressed sector resides entirely in one memory page, each page will have a certain amount of unused storage space. This subsection first discusses how we should determine the content of the unused storage space to minimize the overall damage, then discusses different options of laying out the compressed data within flash memory pages.

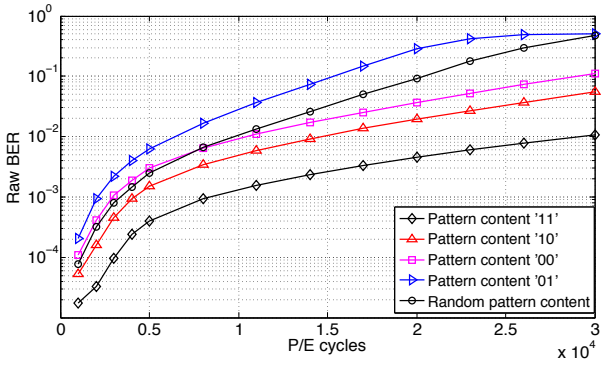

Figure 3: Measured memory raw bit error rate (BER) vs. cycling with different data content.

2.2.1 Content of Unused Storage Space

For MLC NAND flash memory, each pair of lower and upper pages together determine the memory cell content and hence the flash memory damage. To minimize the flash memory damage, we should appropriately determine the data content in the unused storage space left by data compression. Let $S^{(l)}$ and $S^{(u)}$ denote the unused storage space in lower and upper page, and b_l and b_u denote the two bits in the same memory cell and belong to lower and upper page, respectively. Recall that the memory cell damage caused by the content '11', '10', '00', and '01' monotonically increase (where the left bit and right bit resides in lower and upper page, respectively), as illustrated in Fig. 3. Therefore, for each memory cell, we should apply the following rules to minimize flash memory damage:

- If $b_l \in S^{(l)}$ and $b_u \in S^{(u)}$ (i.e., we can freely set the values of both bits), we set $b_l = b_u = 1$ hence the least harmful content '11' is written to the cell;
- If $b_l \in S^{(l)}$ and $b_u \notin S^{(u)}$ (i.e., we can only freely set the value of b_l), we always set b_l as '1' regardless to the value of b_u;
- If $b_l \notin S^{(l)}$ and $b_u \in S^{(u)}$ (i.e., we can only freely set the value of b_u), we always set $b_u = b_l$.

2.2.2 Compressed Data Layout

Since the memory cells covered by $S^{(l)}$ or $S^{(u)}$ experience less damage than the other memory cells, we should keep shifting the location of $S^{(l)}$ and $S^{(u)}$ within flash memory pages in order to equalize the damage among all the memory cells. We define a parameter l_{head} to represent the location from where the compressed data are continuously stored in the lower and upper pages. We should keep changing l_{head} in order to equalize the memory cell damage. Since the storage device FTL module

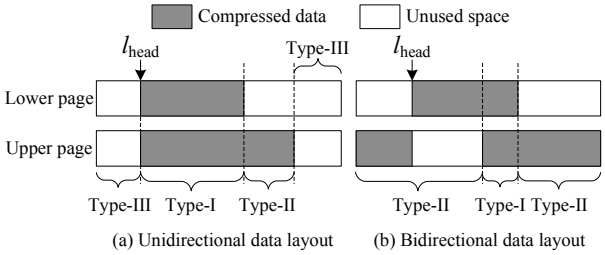

Figure 4: Two different data layout strategies.

always keeps track of the P/E cycles of each memory block, we can fix a relationship between l_{head} and P/E cycle number, e.g., let L denote the memory page size and $N_{P/E}$ denote the P/E cycle number, we can calculate $l_{head} = \lfloor \lfloor t \cdot N_{P/E} \rfloor \bmod L \rfloor$, where t is a fixed constant integer. As a result, the storage device controller does not need to record the value of l_{head} for each memory block. In addition, as decompression is a process that is done serially, the length of the compressed data need not be kept in the FTL. For each compressed memory page, the decompression process can be terminated once the decompressed data length reaches the page length. Therefore, in order to support the proposed design strategy, the only overhead at the FTL layer is to calculate the l_{head} for each memory page.

For MLC NAND flash memory, there are two different options for laying out the compressed data in lower and upper pages. As illustrated in Fig. 4, the first option is to lay out the compressed data towards the same direction from l_{head} in both the lower and upper pages, that we refer is referred to as *unidirectional data layout*. The other option is to lay out the compressed data towards opposite directions in the lower and upper pages, that we refer to as *bidirectional data layout*. As shown in Fig. 4, all the memory cells can be categorized into three types: (1) In each type-I memory cell, both bits belong to the compressed data; (2) In each type-II memory cell, one bit belongs to the compressed data while the other bit belongs to the unused storage space; (3) In each type-III memory cell, both bits belong to the unused storage space. Apparently, the physical damage experienced by type-I, type-II, and type-III memory cells monotonically reduces. Compared with unidirectional data layout, bidirectional data layout leads to more type-II memory cells and less type-I and type-III memory cells.

2.2.3 Conditional Data Exchange

According to the discussion in Section 2.2.1, the content of each type-II memory cell can only belong to {'11', '10'} or {'11', '00'} if the lower or upper page bit belongs to unused storage space. As shown in Fig. 3, '10' causes less damage than '00'. Hence, the memory damage tends to be less if the lower page has more unused storage space (i.e., data being stored in the lower page have better compressibility). This observation directly motivates us to propose *conditional data exchange*: Let $D^{(l)}$ and $D^{(u)}$ denote the compressed data that have been originally arranged by the storage device FTL to store in one pair of lower and upper pages. If the length of $D^{(l)}$ is not larger than that of $D^{(u)}$ (i.e., $|D^{(l)}| \leq |D^{(u)}|$), we directly store $D^{(l)}$ and $D^{(u)}$ to the lower and upper pages, respectively; otherwise we switch their page location, i.e., store $D^{(l)}$ to the upper page and $D^{(u)}$ to the lower page.

Although this design scheme can reduce flash memory damage, it could complicate the FTL design. If the FTL uses the page-level address mapping, we need to update the mapping table once the data exchange operation occurs. This will not introduce any mapping table storage overhead. If the FTL uses block-level or hybrid page/block-level address mapping, we must keep a 1-bit flag for each memory wordline, leading to extra mapping table storage overhead.

3 Mathematical Formulations

This section presents the mathematical formulations that can accurately estimate flash memory physical damage reduction when using the design strategies presented in Section 2, for both explicit and implicit data compression. It is evident that different types of data can have different compressibility characteristics. With the popular LZ77 [16] compression algorithm and sector size of 4kB, Fig. 5 shows the per-sector compression ratio distribution for some common types of data. The results show that the compression ratio tends to approximately follows a Gaussian distribution. We carried out further experiments to verify the accuracy of such distribution approximation. Fig. 5 shows the absolute difference (denoted as "Appr. error" in the figure) between the exact distribution and the approximate distribution for different types of data. The corresponding mean square errors (MSE) for these types of data are all at the magnitude of 10^{-5}. Therefore, we can conclude that such a Gaussian-based approximation is reasonable with almost negligible inaccuracy. Therefore, to facilitate the mathematical derivation, we set that per-sector data compression ratio follows a Gaussian distribution in this work.

3.1 Content-dependent Damage Factor

We first introduce a parameter, called normalized content-dependent damage factor, to quantify the impact of different content on memory cell damage. Let BER_{max} denote the maximum memory raw BER that can be tolerated by the storage device error correction mechanism.

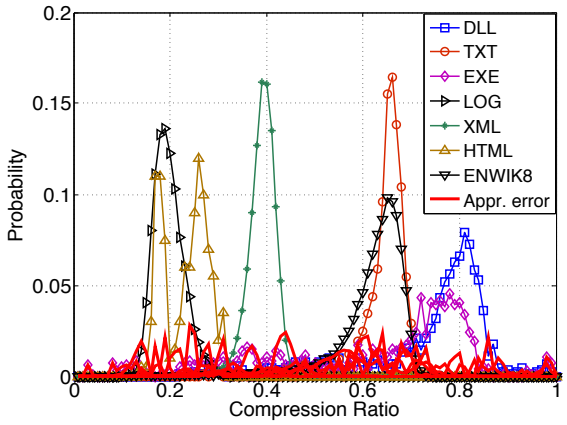

Figure 5: Measured distribution of compression ratio for different types of data.

For l-bit/cell NAND flash memory, let $\Psi^{(i)}(\eta)$ denote the raw BER after we keep programming memory cells with the same content $i \in [0, 2^l - 1]$ for η cycles. Let $\Psi^{(r)}(\eta)$ denote the raw BER after we have programmed memory cells with random content for η cycles. Let $\eta^{(i)}_{max}$ and $\eta^{(r)}_{max}$denote the P/E cycle number under which the raw BER $\Psi^{(i)}(\eta)$ and $\Psi^{(r)}(\eta)$ equal to BER_{max}, respectively. Hence, we can estimate that the physical memory cell damage caused by each programming operation with the content i is proportional to $1/\eta^{(i)}_{max}$. In addition, on average the physical memory cell damage caused by programming random content is proportional to $1/\eta^{(r)}_{max}$. In this work, we define the content-dependent damage factor ρ_i for each content i by normalizing with the average damage caused by random content, i.e.,

$$\rho_i = \frac{\eta^{(r)}_{max}}{\eta^{(i)}_{max}}, \quad \text{where } i \in [0, 2^l - 1], \tag{1}$$

and hence the damage factor ρ_r for random content is 1. Using the measurement results shown in Fig. 3 as an example, assume the BER_{max} is 5×10^{-3}, we calculate the four damage factors as $\rho_{11} = 0.33$, $\rho_{10} = 0.69$, $\rho_{00} = 1.01$, and $\rho_{01} = 1.58$.

3.2 Effect of Compression

We first derive the mathematical formulations for estimating the distribution characteristics of the compressed data and unused storage space size in each page. Let C_s denote the size of each uncompressed data sector (e.g., 4kB), m_s denote the number of uncompressed sectors in each page, and $C_p = m_s \cdot C_s$ denote the size of each flash memory page (e.g., 8kB). As pointed out above, the per-sector compression ratio x approximately follows a Gaussian distribution $N(\mu, \sigma^2)$. Let $m^{(e)}_s$ denote the number of compressed sectors per page when using explicit compression, and $C^{(e)}_s$ denote the length of the compressed data within one page. Due to the variation of the compression ratio x, both $m^{(e)}_s$ and $C^{(e)}_s$ are random variables. Since $x \cdot m^{(e)}_s \cdot C_s$ denotes the length of the compressed data within one page when $m^{(e)}_s$ is determined, $C^{(e)}_s$ can be expressed as

$$C^{(e)}_s = \sum_{m^{(e)}_s = m_s}^{\infty} x \cdot m^{(e)}_s \cdot C_s \cdot P\left(m^{(e)}_s\right), \tag{2}$$

where $P\left(m^{(e)}_s\right)$ is the probability that $m^{(e)}_s$ compressed sectors can fit into one page. We can express $P\left(m^{(e)}_s\right)$ as

$$P\left(m^{(e)}_s\right) = P\left\{x \cdot m^{(e)}_s \le m_s < x \cdot \left(m^{(e)}_s + 1\right)\right\} = P\left\{x \cdot m^{(e)}_s \le m_s\right\} \cdot \left(1 - P\left\{x \cdot \left(m^{(e)}_s + 1\right) \le m_s\right\}\right).$$

Since $x \sim N(\mu, \sigma^2)$, we have that $x \cdot m^{(e)}_s$ and $x \cdot \left(m^{(e)}_s + 1\right)$ follow $N\left(\mu m^{(e)}_s, (\sigma m^{(e)}_s)^2\right)$ and $N\left(\mu(m^{(e)}_s + 1), (\sigma(m^{(e)}_s + 1))^2\right)$, respectively. Hence, $P\left\{x \cdot m^{(e)}_s \le m_s\right\}$ and $P\left\{x \cdot \left(m^{(e)}_s + 1\right) \le m_s\right\}$ is the CDF (cumulative distribution function) for the random variant $x \cdot m^{(e)}_s$ and $x \cdot \left(m^{(e)}_s + 1\right)$. Accordingly, we have that

$$P\left(m^{(e)}_s\right) = \left[1 + erf\left(\frac{m_s - m^{(e)}_s \mu}{\sigma m^{(e)}_s \sqrt{2}}\right)\right] \cdot \left[1 - erf\left(\frac{m_s - (m^{(e)}_s + 1)\mu}{\sigma(m^{(e)}_s + 1)\sqrt{2}}\right)\right] / 4, \tag{3}$$

where $erf(z)$ is the error function for Gaussian distribution, i.e., $erf(z) = \frac{1}{\sqrt{\pi}} \int_{-z}^{z} e^{-t^2} dt$. For each given $m^{(e)}_s$, we can calculate the value of $P\left(m^{(e)}_s\right)$ based upon (3). Hence, each item in (2), i.e., $x \cdot m^{(e)}_s \cdot C_s \cdot P\left(m^{(e)}_s\right)$, is a random variable following a Gaussian distribution. As a result, we have that $C^{(e)}_s \sim N\left(\mu_{c^{(e)}_s}, \sigma^2_{c^{(e)}_s}\right)$, where

$$\begin{cases} \mu_{c^{(e)}_s} &= \mu C_s \cdot \sum\limits_{m^{(e)}_s} m^{(e)}_s P\left(m^{(e)}_s\right), \\ \sigma^2_{c^{(e)}_s} &= (\sigma C_s)^2 \cdot \sum\limits_{m^{(e)}_s} \left(m^{(e)}_s P\left(m^{(e)}_s\right)\right)^2. \end{cases} \tag{4}$$

When using implicit compression, the number of compressed sectors per flash memory page always remains as m_s and the length of compressed data per page is $C^{(i)}_s = x \cdot m_s \cdot C_s$. Therefore, we have that the random variable $C^{(i)}_s \sim N\left(\mu m_s C_s, \sigma^2 m_s^2 C_s^2\right)$.

3.3 Memory Damage Estimation

We further derive the mathematical formulations for calculating average memory cell damage per P/E cycle. Based upon the above discussions, we should consider four different design scenarios: (1) *UD*: unidirectional data layout without conditional data exchange, (2) *BD*: bidirectional data layout without conditional data exchange, (3) *UDC*: unidirectional data layout with conditional data exchange, (4) *BDC*: bidirectional data layout with conditional data exchange. Since the mathematical formulations can be derived with the same principle for all the scenarios, we first show the mathematical derivation in detail for *UD* (i.e., unidirectional data layout without conditional data exchange) and then present the results for the others without detailed derivations.

3.3.1 Derivation for the *UD* Design Scenario

We first define two parameters x_l and x_u as the ratios between the compressed data size and flash memory page size for lower and upper pages, respectively. Recall that both $C_s^{(e)}$ and $C_s^{(i)}$ (i.e., compressed data size within each flash memory page when using explicit and implicit compression, respectively) follow Gaussian distributions as derived in Section 3.2. Hence, x_l and x_u also follow the Gaussian distribution $N(\tilde{\mu},\tilde{\sigma}^2)$, where $\tilde{\mu}=\mu_{c_s^{(e)}}/C_p$ and $\tilde{\sigma}^2=\sigma^2_{c_s^{(e)}}/C_p^2$ for explicit compression, and $\tilde{\mu}=\mu m_s C_s/C_p$ and $\tilde{\sigma}^2=\sigma^2 m_s^2 C_s^2/C_p^2$ for implicit compression. Define $z_l=\min(x_l,x_u)$ and $z_u=\max(x_l,x_u)$ and recall that $\{\rho_{11},\rho_{10},\rho_{00},\rho_{01}\}$ represent the memory damage factors for the four different memory cell content and the damage factor ρ_r for random content is 1. In addition, let $m_s^{(c)}$ denote the average number of sectors per flash memory page and recall that m_s denote the number of sectors per flash memory page without using compression, and define $r=\frac{m_s^{(c)}}{m_s}$. Therefore, we can calculate the average memory cell damage per P/E cycle for the UD design scenario, which is normalized against the case of without using compression, as

$$\begin{aligned}\rho_{UD}=&\frac{1}{r}\left(z_l+|x_l-x_u|\,\frac{\rho_{00}+2\rho_{11}+\rho_{10}}{4}+(1-z_u)\rho_{11}\right)\\ =&\frac{1}{r}\left(1-\frac{\rho_{00}+2\rho_{11}+\rho_{10}}{4}\right)z_l\\ &+\frac{1}{r}\left(\frac{\rho_{00}+\rho_{10}-2\rho_{11}}{4}\right)z_u+\frac{\rho_{11}}{r}\\ =&\frac{\lambda_{UD}^l}{r}\cdot z_l+\frac{\lambda_{UD}^u}{r}\cdot z_u+\frac{\rho_{11}}{r},\end{aligned}\tag{5}$$

where

$$\lambda_{UD}^l=1-\frac{\rho_{00}+2\rho_{11}+\rho_{10}}{4},\ \lambda_{UD}^u=\frac{\rho_{00}+\rho_{10}-2\rho_{11}}{4}.$$

In order to obtain the distribution of ρ_{UD}, we must derive the distributions of z_u and z_l. The CDF of z_u can be written as

$$\begin{aligned}F_{z_u}(z) &= P(x_l\le z,x_u\le z)=P(x_l\le z)\cdot P(x_u\le z)\\ &= F_{x_l}(z)\cdot F_{x_u}(z),\end{aligned}$$

where F_{x_l} and F_{x_u} denote the CDF of x_u and x_l. Since x_u and x_l follow the same Gaussian distribution (denoted as f_N), we have that $F_{x_l}=F_{x_u}$. By taking the derivative of the CDF, we can obtain the PDF of z_u as

$$\begin{aligned}f_{z_u}(z) &= F'_{z_u}(z)=f_N(z)\cdot\left(1+erf\left(\frac{z-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}\right)\right)\\ &\approx f_N(z)\cdot\left(1+\frac{z-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}\right).\end{aligned}\tag{6}$$

Hence, f_{z_u} can be approximately expressed as the product of the PDF of a Gaussian distribution and a straight line with the slope of $\sqrt{2}\tilde{\sigma}$. Since $z\in(0,1]$, we could further approximate f_{z_u} to a PDF of a Gaussian distribution, i.e., $z_u\sim N\left(\mu_{z_u},\sigma^2_{z_u}\right)$ and $f_{z_u}(z)=\frac{1}{\sigma_{z_u}\sqrt{2\pi}}\exp\left(-\frac{(z-\mu_{z_u})^2}{2\sigma^2_{z_u}}\right)$. The value of μ_{z_u} and σ_{z_u} can be obtained by solving

$$\begin{cases}\left.\frac{d(f_{z_u}(z))}{dz}\right|_{z=\mu_{z_u}}\approx\left.\frac{d\left(f_N(z)\cdot\left(1+\frac{z-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}\right)\right)}{dz}\right|_{z=\mu_{z_u}}=0,\\ f_{z_u}(z)|_{z=\mu_{z_u}}\approx\left.f_N(z)\cdot\left(1+\frac{z-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}\right)\right|_{z=\mu_{z_u}}.\end{cases}\tag{7}$$

Accordingly, we have that

$$\begin{cases}\mu_{z_u} &= \tilde{\mu}+\frac{\sqrt{6}-\sqrt{2}}{2}\cdot\tilde{\sigma},\\ \sigma_{z_u} &= \frac{2\sqrt{2}}{\sqrt{6}+\sqrt{2}}\cdot\tilde{\sigma}\cdot\exp\left(\frac{\left(\sqrt{6}-\sqrt{2}\right)^2}{8}\right).\end{cases}\tag{8}$$

We can obtain the PDF of z_l in similar manner. First, we can express the CDF of z_l as

$$\begin{aligned}F_{z_l}(z) &= 1-P(x_l>z,x_u>z)\\ &= 1-(1-P(x_l\le z))\cdot(1-P(x_u\le z))\\ &= 1-(1-F_N(z))^2.\end{aligned}$$

By taking the derivative of F_{z_l}, we obtain the PDF of z_l as

$$f_{z_l}(z)=2\,(1-F_N(z))\cdot f_N(z).\tag{9}$$

Similar to the above derivations for the case of z_u, we can approximate the PDF f_{z_l} as a Gaussian distribution $N\left(\mu_{z_l},\sigma^2_{z_l}\right)$, where

$$\begin{cases}\mu_{z_l} &= \tilde{\mu}-\frac{\sqrt{6}-\sqrt{2}}{2}\cdot\tilde{\sigma},\\ \sigma_{z_l} &= \frac{2\sqrt{2}}{\sqrt{6}+\sqrt{2}}\cdot\tilde{\sigma}\cdot\exp\left(\frac{\left(\sqrt{6}-\sqrt{2}\right)^2}{8}\right).\end{cases}\tag{10}$$

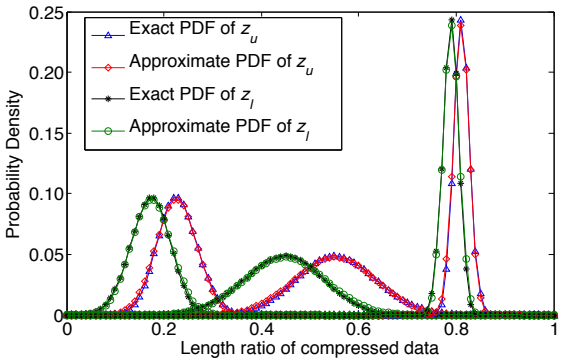

Figure 6: Comparison between the exact PDF of z_l, z_u and their Gaussian approximations with different sets of $\tilde{\mu}$ and $\tilde{\sigma}$.

To justify the Gaussian approximation of $f_{z_u}(z)$ and $f_{z_l}(z)$ (i.e., the PDF of z_u and z_l) in the above derivations, Fig. 6 compares the Gaussian approximation and the exact PDF, where we considered three different sets of $\{\tilde{\mu},\tilde{\sigma}\}$ (i.e., $\{0.2,0.05\}$, $\{0.5,0.1\}$, and $\{0.8,0.02\}$, respectively) to cover a wide range of the compressed data length ratio and deviations. As clearly shown in Fig. 6, the Gaussian approximation of $f_{z_u}(z)$ and $f_{z_l}(z)$ incurs almost negligible inaccuracy.

Since $z_l \sim N\left(\mu_{z_l},\sigma_{z_l}^2\right)$ and $z_u \sim N\left(\mu_{z_u},\sigma_{z_u}^2\right)$, according to (5), the average cell damage ρ_{UD} also follows a Gaussian distribution, i.e., $\rho_{UD} \sim N\left(\mu_{UD},\sigma_{UD}^2\right)$, where

$$\begin{cases} \mu_{UD} &= \frac{1}{r}\left(\lambda_{UD}^{l}\cdot\mu_{z_l}+\lambda_{UD}^{u}\cdot\mu_{z_u}+\rho_{11}\right), \\ \sigma_{UD} &= \frac{1}{r}\sqrt{\left(\lambda_{UD}^{l}\cdot\sigma_{z_l}\right)^2+\left(\lambda_{UD}^{u}\cdot\sigma_{z_u}\right)^2}. \end{cases} \quad (11)$$

Given the same data compressibility, the use of implicit and explicit compression leads to different distribution of x_u and x_l, and different r, leading to different memory cell damage.

3.3.2 More Formulation Results

Using the same principle, we can derive the mathematical formulations that can calculate the normalized average memory cell damage per P/E cycle for the other three design scenarios. Due to the page limit, we will directly present the final mathematical formulations without showing the derivation details.

For the *BD* design scenario that uses bidirectional data layout without conditional data exchange, its average memory cell damage is $\rho_{(BD)} \sim N\left(\mu_{(BD)},\sigma_{(BD)}^2\right)$:

$$\begin{cases} \mu_{(BD)} &= \frac{1}{r}\left(\left(\lambda_{(BD)}^{l}+\lambda_{(BD)}^{u}\right)\cdot\tilde{\mu}+C_{(BD)}\right), \\ \sigma_{(BD)} &= \frac{1}{r}\sqrt{\left(\lambda_{(BD)}^{l}\right)^2+\left(\lambda_{(BD)}^{u}\right)^2}\cdot\tilde{\sigma}. \end{cases}$$

where

$$\begin{cases} \lambda_{(BD)}^{l} &= 1-\frac{\rho_{11}+\rho_{10}}{2}, \quad \lambda_{(BD)}^{u} = 1-\frac{\rho_{11}+\rho_{00}}{2}, \\ C_{(BD)} &= \frac{2\rho_{11}+\rho_{10}+\rho_{00}}{2}-1. \end{cases}$$

For the *UDC* design scenario that uses unidirectional data layout with conditional data exchange, its average memory cell damage is $\rho_{(UDC)} \sim N\left(\mu_{(UDC)},\sigma_{(UDC)}^2\right)$:

$$\begin{cases} \mu_{(UDC)} &= \frac{1}{r}\left(\lambda_{(UDC)}^{l}\cdot\mu_{z_l}+\lambda_{(UDC)}^{u}\cdot\mu_{z_u}+\rho_{11}\right), \\ \sigma_{(UDC)} &= \frac{1}{r}\sqrt{\left(\lambda_{(UDC)}^{l}\cdot\sigma_{z_l}\right)^2+\left(\lambda_{(UDC)}^{u}\cdot\sigma_{z_u}\right)^2}. \end{cases}$$

where

$$\lambda_{(UDC)}^{l} = 1-\frac{\rho_{11}+\rho_{10}}{2}, \quad \lambda_{(UDC)}^{u} = \frac{\rho_{11}+\rho_{10}}{2}-\rho_{11}.$$

For the *BDC* design scenario that uses bidirectional data layout with conditional data exchange, its average memory cell damage is $\rho_{(BDC)} \sim N\left(\mu_{(BDC)},\sigma_{(BDC)}^2\right)$:

$$\begin{cases} \mu_{(BDC)} &= \frac{1}{r}\left(\lambda_{(BD)}^{l}\cdot\mu_{z_l}+\lambda_{(BD)}^{u}\cdot\mu_{z_u}+C_{(BD)}\right), \\ \sigma_{(BDC)} &= \frac{1}{r}\sqrt{\left(\lambda_{(BD)}^{l}\cdot\sigma_{z_l}\right)^2+\left(\lambda_{(BD)}^{u}\cdot\sigma_{z_u}\right)^2}. \end{cases}$$

3.4 Estimation of Memory Lifetime

This subsection discusses how we estimate the flash memory lifetime improvement based upon the average memory cell damage derived in the above section. In this work, we assume ideal wear-leveling, i.e., all the memory blocks always experience the same number of P/E cycles, and quantitatively define memory lifetime as the P/E cycle number that one memory block can survive before reaching the maximum allowable BER. Since it is common practice to use capacity over-provisioning in flash-based storage devices, we define an over-provisioning factor $\tau \geq 1$, i.e., the total physical storage capacity inside the storage device is $\tau\times$ larger than the storage capacity visible to the host. Let η denote the memory block P/E cycling endurance of the baseline scenario without using any data compression. Straightforwardly, the overall memory lifetime of the baseline scenario is $\tau\cdot\eta$ cycles.

Once data compression is used, the average memory cell damage becomes a random variable with a Gaussian distribution due to the Gaussian-like distribution of runtime data compression ratio. As a result, the cycling endurance of each memory block and hence overall memory lifetime also become random variables. Let $P_b^{(t)}$ denote the probability that one memory block can survive (i.e., can ensure the storage integrity even for incompressible data) after t P/E cycles, referred to as memory block survival probability. As the granularity of data erasure is in memory block units in NAND Flash memory,

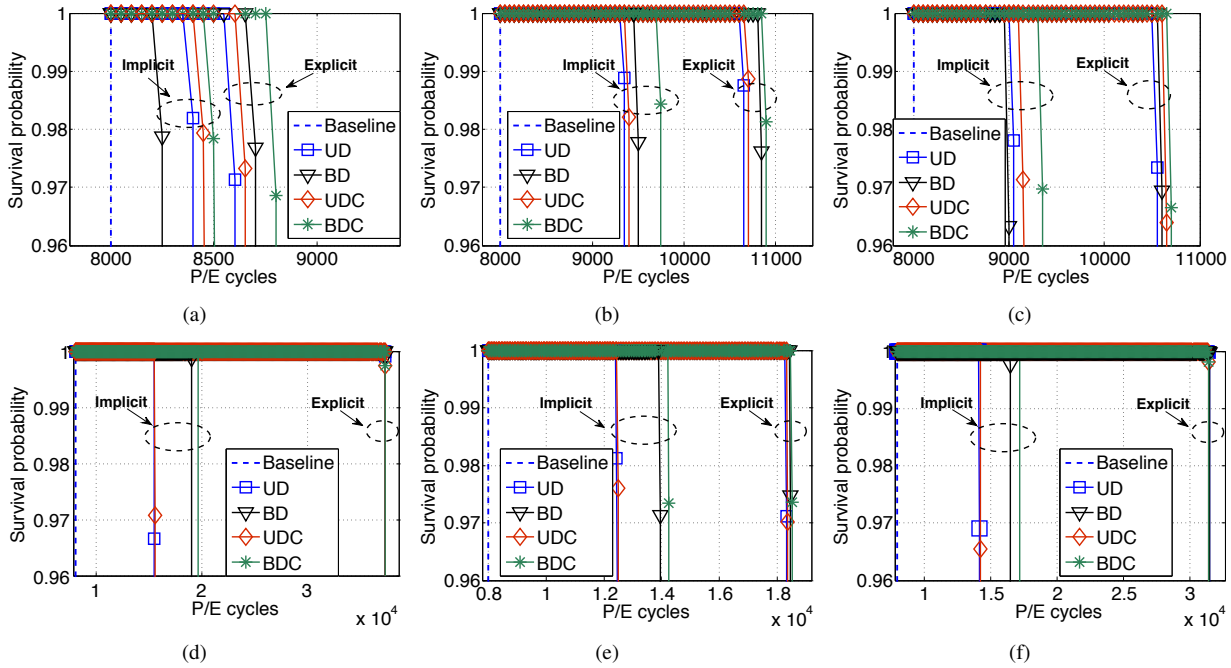

Figure 7: Storage device survival probability when storing (a) DLL, (b) Text, (c) Exe, (d) Log, (e) XML, and (f) HTML data in flash memory. Both explicit compression and implicit compression are considered.

the number of P/E cycles is independent among memory blocks. Hence, the survival of memory blocks is independent. Let N denote the number of memory blocks visible to the host, then the storage device contains $\tau \cdot N$ memory blocks in total. Therefore, once $P_b^{(t)}$ is known, based on the law of total probability, we can calculate the probability that the storage device can survive t cycles as

$$SP^{(t)} = \sum_{k=0}^{(\tau-1)N} \left(\binom{N\tau}{k} \cdot \left(P_b^{(t)}\right)^{N\tau-k} \cdot \left(1-P_b^{(t)}\right)^{k} \right), \tag{12}$$

which is called storage device survival probability. Suppose each memory block contains M wordlines and let $P_{wl}^{(t)}$ denote the survival probability of one wordline, we have that $P_b^{(t)} = \left(P_{wl}^{(t)}\right)^M$, i.e., one memory block survives only when all the wordlines inside this block survive. In the following, we will discuss how we can estimate the memory wordline survival probability $P_{wl}^{(t)}$.

For the baseline scenario without using data compression, the storage device fails to survive once the accumulated average damage of each memory cell reaches $\eta \cdot \rho_r$ (recall that $\rho_r = 1$ is the normalized memory cell damage factor when storing random data). When using data compression, let ρ_w denote the memory cell damage per cycle, where ρ_w could be $\rho_{(UD)}$, $\rho_{(BD)}$, $\rho_{(UDC)}$, or $\rho_{(BDC)}$ dependent upon the design strategies being used. By setting $\eta \cdot \rho_r$ as the maximum tolerable accumulated memory cell damage, we can express the P/E cycling endurance of each wordline as

$$T = \max(t), \;\; t \cdot \rho_w \leq \eta \cdot \rho_r - \rho_r. \tag{13}$$

Since ρ_w follows Gaussian distribution, $t \cdot \rho_w$ also follows Gaussian distribution with mean of $t \cdot \mu_{\rho_w}$ and variance of $t^2 \cdot \sigma_{\rho_w}^2$. Therefore, we can calculate the wordline survival probability $P_{wl}^{(t)}$ at t cycles as

$$P_{wl}^{(t)} = \frac{1}{2}\left(1 + erf\left(\frac{\tau \cdot \eta - 1 - t \cdot \mu_{\rho_w}}{t\sigma_{\rho_w}}\right)\right), \tag{14}$$

where μ_{ρ_w} and σ_{ρ_w} can be obtained using the formulations presented above for the four different design scenarios.

4 Quantitative Studies

With the formulations derived in Section 3, we studied the effectiveness of the design strategies presented in Section 2 for both explicit and implicit data compression. Based upon our measurement results with 20nm MLC NAND flash memory chips, we set the damage factors $\rho_{11} = 0.33$, $\rho_{10} = 0.69$, $\rho_{00} = 1.01$, and $\rho_{01} = 1.58$, as discussed in Section 3.1. As shown in (12), the storage device survival probability $SP^{(t)}$ depends on the over-provisioning factor τ and the total number of memory blocks N visible to the host. Assume the storage capacity of 512GB visible to the host and a block size of 4MB,

we have N equals 128k. Each flash memory page has a size of 8kB, and we set the data sector size as 4kB. We further set the over-provisioning factor τ as 1.2. Based upon the memory chip measurement results and the over-provisioning factor of 1.2, we set the cycling endurance of the baseline scenario (i.e., without using data compression) as 8000.

4.1 Lifetime with Different Data Types

Using the measured compression ratio distribution of different data types as shown in Fig. 5, we evaluated the effectiveness of the developed design strategies on improving memory lifetime over the baseline scenario. Fig. 7 shows the results when using the four different design scenarios. Recall that the design scenario *UD* uses unidirectional data layout without conditional data exchange, *BD* uses bidirectional data layout without conditional data exchange, *UDC* uses unidirectional data layout with conditional data exchange, *BDC* uses bidirectional data layout with conditional data exchange. For the baseline scenario, the storage device lifetime remains 8000 regardless to the data types. When using data compression with different design strategies, the storage device lifetime becomes a random variable, whose CDF (i.e., its survival probability) is calculated according to the formulations derived in Section 3.

As shown in Fig. 7, explicit compression always outperforms implicit compression, which can be intuitively justified because explicit compression always tries to fit as many sectors as possible into each flash page. By comparing the data compressibility shown in Fig. 5 and the results shown in Fig. 7, we can clearly see that the difference between explicit compression and implicit compression strongly relies on the data compressibility. The higher data compressibility is, the larger difference between explicit compression and implicit compression is. In addition, the BDC design scenario always performs the best under both explicit and implicit data compression.

When explicit compression is being used, the difference among different design strategies tends to diminish for data with better compressibility (e.g., LOG and HTML). This can be explained as follows. With highly compressible data, explicit compression can fit more compressed data and hence leave less unused storage space within each flash memory page. As a result, there is a smaller room for these different design strategies to exploit the unused storage space, leading to almost the same storage device lifetime. On the other hand, when implicit compression is being used, unidirectional data layout and bidirectional data layout tend to have noticeable different effect, especially for data with better compressibility. As pointed out in Section 2.2.2, compared with bidirectional data layout, unidirectional data layout leads to more cells with random data content and '11'. Although '11' causes the least memory cell damage, random data tend to cause relatively large damage, as shown in Fig. 3. Based upon the content-dependent damage factors measured from our 20nm MLC flash memory chips, the penalty of having more random data can noticeably off-set the gain of having more '11'. As a result, unidirectional data layout tends to be inferior to bidirectional data layout. In the case of implicit compression, for data with worse compressibility (e.g., DLL and EXE), most memory cells would store random data content in both unidirectional and bidirectional data layout. As a result, bidirectional data layout will be inferior to unidirectional data layout in this scenario, which is shown in Fig. 7(a) and (c). Meanwhile, as shown in the results, the benefit of using conditional data exchange is not significant. Conditional data exchange aims to convert memory cell content from '00' to '10', since '10' causes less damage than '00'. Nevertheless, as shown in Fig. 3, the damage difference between '00' and '10' is not significant, which explains the low effectiveness of conditional data exchange observed in our study.

4.2 Sensitivity to Data Compressibility

The above results are based upon the measured compressibility characteristics of several different types of data. To more thoroughly elaborate on the impact of data compressibility, we carried out further evaluations by considering a much wider range of data compressibility in terms of compression ratio mean and standard deviation.

We first fix the data compression ratio standard deviation as 0.01, and Fig. 8 shows the corresponding storage device survival probability vs. lifetime for a wide range of compression ratio mean from 0.1 to 0.9. Data with better compressibility (i.e., smaller compression ratio mean) lead to larger lifetime improvement in both explicit compression and implicit compression. In addition, the advantage of explicit compression over implicit compression increases as the data have better compressibility. At the compression ratio mean of 0.1 (i.e., the data can be compressed by 10:1 on average), explicit and implicit compression can improve the storage device lifetime by 9.6 and 4.8 times, respectively. As shown in Fig. 8, for less compressible data (e.g., with the compression ratio mean of 0.7 and higher), explicit and implicit compression have almost the same effect. This is because, with low data compressibility, explicit compression can hardly increase the number of compressed data sectors per page. The results more clearly reveal the observations discussed above in Section 4.1: Under explicit compression, the difference between different de-

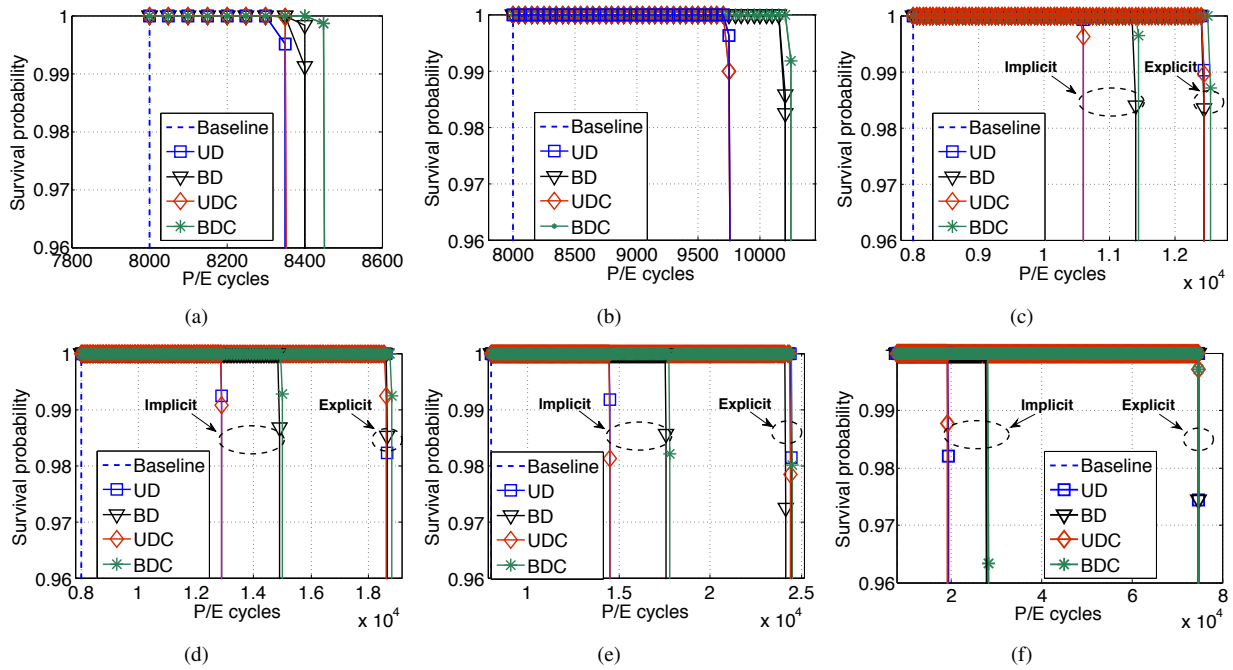

Figure 8: Storage device survival probability when data compression ratio standard deviation is 0.01 and mean is (a) 0.9, (b) 0.7, (c) 0.6, (d) 0.4, (e) 0.3, and (f) 0.1.

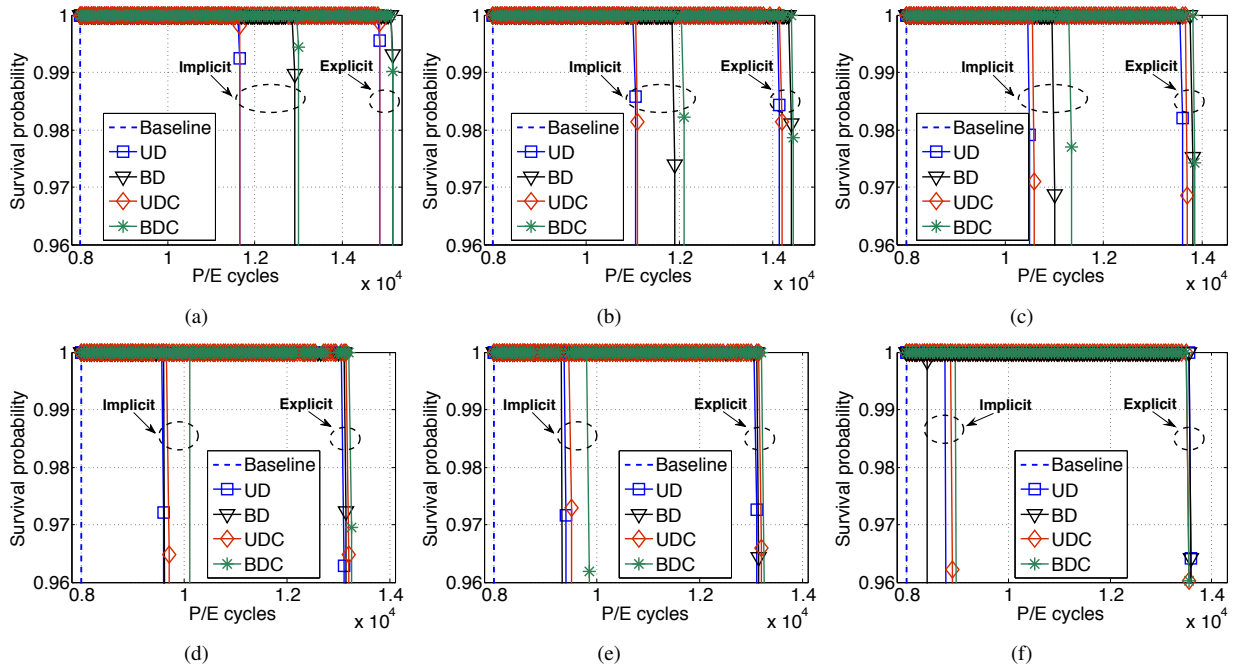

Figure 10: Storage device lifetime survival probability when compression ratio mean is 0.5 and standard deviation is (a) 0.01, (b) 0.03, (c) 0.05, (d) 0.09, (e) 0.1, and (f) 0.14.

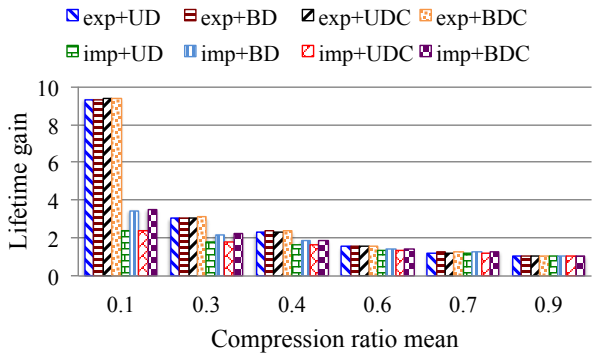

Figure 9: Lifetime gain under different data compression ratio mean.

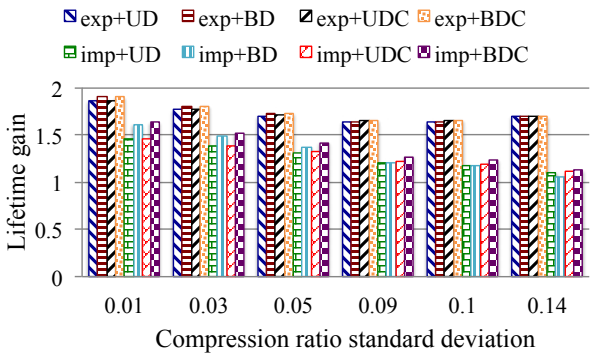

Figure 11: Lifetime gain for different data compression ratio standard deviation.

sign strategies quickly shrinks as we reduce the compression ratio mean; Under implicit compression, bidirectional data layout is always noticeably more beneficial than unidirectional data layout. By setting the storage device lifetime as the P/E cycles corresponding to 99.9% of storage device survival probability, Fig. 9 further plots the storage device lifetime gain over the baseline scenario without using compression under different compression ratio mean.

Next, we examined the impact of data compression ratio standard deviation. With the compression ratio mean of 0.5, Fig. 10 shows the storage device survival probability vs. P/E cycles when the compression ratio standard deviation varies from 0.01 to 0.14. As the data compression ratio standard deviation increases, advantage of explicit compression over implicit compression becomes more significant, and the storage device lifetime improvement generally reduces. In addition, the difference among the four different design scenarios reduces as the compression ratio standard deviation increases, for both explicit and implicit compression. Again, the design scenario of BDC is the most effective for both explicit and implicit compression.

By setting the storage device lifetime as the P/E cycles corresponding to 99.9% of storage device survival probability, Fig. 11 further shows the storage device lifetime gain over the baseline scenario under different compression ratio standard deviation. It shows that the lifetime gain monotonically reduces as we increase the data compression ratio standard deviation for implicit data compression. Nevertheless, for explicit data compression, the storage device lifetime gain first reduces and then saturates and even slightly increases as we increase the compression ratio standard deviation. The figure more clearly reveals the dependency of comparison between explicit and implicit compression on compression ratio standard deviation.

4.3 Discussions

The above quantitative studies show that the proposed implicit data compression is a viable complement to the conventional explicit data compression. Although explicit data compression may noticeably complicate the design of FTL and/or OS, it always outperforms implicit data compression from the storage device lifetime perspective. Nevertheless, the advantage of explicit compression over implicit compression strongly depends on the data compressibility. As shown in the above evaluation results, the advantage of explicit compression over implicit compression reduces as the data compressibility drops, and becomes very small as the data compression ratio mean becomes sufficiently large (e.g., over 0.6∼0.7 in this study), particularly when the data compression ratio has a small standard deviation.

Our studies show that the bidirectional data layout outperforms the unidirectional data layout, especially when using the implicit data compression. Nevertheless, we should emphasize that this conclusion may not be always true. As pointed out above, compared with bidirectional data layout, unidirectional data layout result in more memory cells with random data content and '11'. Hence, which data layout option is better is fundamentally dependent on the exact values of the content-dependent damage factors. In this work, we extracted the content-dependent damage factors based upon measurements with 20nm MLC flash memory chips. However, for further scaled technology nodes such as 16nm or the emerging 3D flash memory, content-dependent damage factors and their relative comparison may (largely) change. This could essentially change the conclusion on the comparison between unidirectional data layout and bidirectional data layout. In addition, the above results suggest that the design strategy of conditional data exchange is not very effective, which is again also essentially due to the content-dependent damage factors being

used in this work. For MLC NAND flash memory, the conditional data exchange will become more effective if the damage factors of '10' and '00' have a larger difference in future memory technology nodes.

Therefore, when applying the developed design framework in practice, one should carry out sufficient measurements and experiments to fully understand the content dependency of NAND flash memory damage and runtime data compressibility characteristics, in order to determine the most appropriate design strategy for leveraging data compressibility to improve device lifetime.

5 Related Work

Prior work [17–19] has studied the practical implementation of data compression in flash-based data storage systems, aiming to improve the storage system I/O speed performance and flash memory lifetime. In [17], a block-level compression engine is devised to support on-line compression for SSD-based cache, which is transparent to the file system. The authors of [18] develop a compression-aware FTL that can support compression-aware address mapping and garbage collection. The authors of [19] implement a caching system with commodity SSD by integrating data compression and data deduplication. All the prior work aimed to explicitly improve the storage efficiency, like the *explicit data compression* scenario being considered in this work. Besides data compression, prior work [20, 21] also investigated the practical implementation of data deduplication in flash-based storage systems.

FTL plays an important role in determining the lifetime of flash-based data storage devices, hence it has been well studied. The wear-leveling function in FTL aims to equalize the physical damage among all the flash memory block by appropriately allocating the memory blocks for erase and programming. A variety of techniques have been proposed to optimize the design of the wear-leveling function (e.g., see [10, 22, 23]). Aiming to reduce the write amplification and hence improve flash memory lifetime, the garbage collection function in FTL has been well studied (e.g., see [24]). The log-structured approach to managing flash memory have been considered through direct management of raw flash memory chips [11] or by facilitating the operation of the FTL inside SSDs [13]. Such log-structured file system level management of memory chips lead to improved flash memory lifetime and storage system performance.

The strength of fault tolerance, in particular ECC, also largely affect the storage device lifetime. Although classical BCH codes are still widely used in commercial flash-based storage devices [25, 26], the more powerful LDPC codes are receiving significant attention from the industry (e.g., see several industrial presentations at recent Flash Summit [2, 3, 27, 28]). A variety of techniques [29–32] have been developed to optimize the implementation of LDPC codes in future flash-based data storage devices.

6 Conclusion

This paper presents a thorough study on exploiting data compressibility to reduce cycling-induced flash memory cell physical damage and hence improve storage device lifetime. This work is essentially motivated by the content dependency of flash memory cell damage. We first present an unconventional implicit data compression strategy as a viable complement to explicit data compression being used in current practice, both of which represent different trade-offs between flash memory lifetime improvement and impact on FTL and system design complexity. In addition, their effectiveness and comparison largely vary with the runtime data compressibility characteristics. We further develop a set of design strategies that can exploit the unused storage space left by data compression within flash memory pages in order to minimize the overall memory physical damage. Furthermore, we derive a set of mathematical formulations that can quantitatively estimate the effectiveness of the proposed design strategies. Using 20nm MLC NAND flash memory chips, we carried out experiments to empirically evaluate the content dependency of flash memory cell damage. Employing these quantized experimental results, we compare the effectiveness of the proposed design strategies when using either explicit or implicit compression. Although this work focuses on flash memory, the proposed design strategies and developed mathematical formulations are readily applicable to other emerging memory technologies, e.g., PCM and ReRAM, that experience similar content dependency of memory cell damage.

Acknowledgements

We would like to thank our shepherd Sam H. Noh and the anonymous reviewers for their insight and suggestions for improvement. This work was supported by the National Science Foundation under Grants No. 1162152 and 1406154, National Science Foundation CAREER award CNS-125394, and the Department of Defense award W911NF-13-1-0157.

References

[1] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in NAND Flash memory," IEEE Transactions on Circuits and

Systems I: Regular Papers, vol. 58, no. 2, pp. 429–439, 2011.

[2] E. Yeo, "An LDPC-enabled flash controller in 40nm CMOS," in Proceedings of Flash Memory Summit, 2012.

[3] X. Hu, "LDPC codes for Flash channel," in Proceedings of Flash Memory Summit, 2012.

[4] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," IEEE Transactions on Consumer Electronics, vol. 48, no. 2, pp. 366–375, 2002.

[5] K. Yim, H. Bahn, and K. Koh, "A flash compression layer for smart media card systems," IEEE Transactions on Consumer Electronics, vol. 50, no. 1, pp. 192–197, 2004.

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

[7] J. Kang, H. Jo, J. Kim, and J. Lee, "A superblock-based flash translation layer for NAND Flash memory," in Proceedings of the 6th ACM & IEEE International conference on Embedded software, pp. 161–170, 2006.

[8] T. Park and J. Kim, "Compression support for flash translation layer," in Proceedings of the International Workshop on Software Support for Portable Storage, pp. 19–24, 2010.

[9] S. Lee, J. Park, K. Fleming, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," IEEE Transactions on Consumer Electronics, vol. 57, no. 4, pp. 1732–1739, 2011.

[10] Y. Pan, G. Dong, and T. Zhang, "Error rate-based wear-leveling for NAND Flash memory at highly scaled technology nodes," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 7, pp. 1350–1354, 2013.

[11] C. Manning, "Introducing yaffs, the first NAND-specific flash file system," http://linuxdevices.com, 2002.

[12] A. Schierl, G. Schellhorn, D. Haneberg, and W. Reif, "Abstract specification of the UBIFS file system for flash memory," in FM 2009: Formal Methods, pp. 190–206. Springer, 2009.

[13] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A New File System for Flash Storage," in Proceedings of the 13th USENIX File and Storage Technologies (FAST), 2015.

[14] Y. Cai, E.F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement characterization and analysis," in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 521–526, 2012.

[15] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, S. Eric, F. Trivedi, E. Goodness, and L.R. Nevill, "Bit error rate in NAND Flash memories," in Proceedings of IEEE International Reliability Physics Symposium, pp. 9–19, 2008.

[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory, vol. IT-23, pp. 337–343, 1977.

[17] T. Makatos, Y. Klonatos, M. Marazakis, M. Flouris, and A. Bilas, "Using transparent compression to improve SSD-based i/o caches," in Proceedings of the European Conference on Computer Systems (EuroSys), pp. 1–14, 2010.

[18] S. Lee, J. Park, K. Fleming, Arvind, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," IEEE Transactions on Consumer Electronics, vol. 57, no. 4, pp. 1732–1739, 2011.

[19] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized SSD cache for primary storage," in Proceedings of USENIX Annual Technical Conference (ATC), pp. 501–512, 2014.

[20] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND Flash-based SSDs," in Proceedings of the 9th USENIX File and Storage Technologies (FAST), 2011.

[21] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of Flash memory based solid state drives," in Proceedings of the 9th USENIX File and Storage Technologies (FAST), 2011.

[22] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in Proceedings of the 11th USENIX File and Storage Technologies (FAST), pp. 257–270, 2013.

[23] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: improving NAND Flash lifetime by balancing page endurance," in Proceedings of the 12th USENIX File and Storage Technologies (FAST), 47–59, pp. 2014.

[24] L. Chang, T. Kuo, and S. 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.

[25] Y. Lee, H. Yoo, I. Yoo, and I. Park, "6.4 gb/s multi-threaded BCH encoder and decoder for multi-channel SSD controllers," in Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), pp. 426–428, 2012.

[26] H. Tsai, C. Yang, and H. Chang, "An efficient BCH decoder with 124-bit correctability for multi-channel SSD applications," in Proceedings of IEEE Asian Solid State Circuits Conference (A-SSCC), pp. 61–64, 2012.

[27] J. Yang, "The efficient LDPC DSP system for SSD," in Proceedings of Flash Memory Summit, 2013.

[28] L. Dolecek, "Non binary LDPC codes: The next frontier in ECC for flash," in Proceedings of Flash Memory Summit, 2014.

[29] J. Wang, T. Courtade, H. Shankar, and R. Wesel, "Soft information for LDPC decoding in flash: mutual-information optimized quantization," in Proceedings of IEEE Global Telecommunications Conference (GLOBECOM), 2011.

[30] S. Tanakamaru, Y. Yanagihara, and K. Takeuchi, "Over-10-extended-lifetime 76%-reduced-error solid-state drives (SSDs) with error-prediction LDPC architecture and error-recovery scheme," in Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), 424–426, pp. 2012.

[31] J. Li, K. Zhao, J. Ma, and T. Zhang, "Realizing unequal error correction for NAND flash memory at minimal read latency overhead," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 61, no. 5, pp. 354–358, 2014.

[32] J. Wang, K. Vakilinia, T. Chen, T. Courtade, G. Dong, T. Zhang, H. Shankar, and R. Wesel, "Enhanced precision through multiple reads for LDPC decoding in flash memories," IEEE Journal on Selected Areas in Communications, vol. 32, no. 5, pp. 880–891, 2014.