Characterizing Flash Memory: Anomalies, Observations, and Applications[*]

UCSD CSE Tech Report CS2009-0946

Laura Grupp Adrian Caulfield Joel Coburn Steven Swanson
The Department of Computer Science and Engineering
University of California, San Diego
{lgrupp,acaulfie,jdcoburn,swanson}@cs.ucsd.edu

Eitan Yaakobi Paul Siegel
The Center for Magnetic Recording Research
University of California, San Diego
{psiegel,eyaakobi}@ucsd.edu

Abstract

Despite flash memory's promise, it suffers from many idiosyncrasies such as limited durability, data integrity problems, and asymmetry in operation granularity. As architects, we aim to find ways to overcome these idiosyncrasies while exploiting flash memory's useful characteristics. To be successful, we must understand the trade-offs between the performance, cost (in both power and dollars), and reliability of flash memory. In addition, we must understand how different usage patterns affect these characteristics. Flash manufacturers provide only vague guidelines about these metrics, and this lack of detail makes it difficult to design systems that fully exploit flash memory's capabilities. We have empirically characterized flash memory technology from five manufacturers by directly measuring the performance, power, and reliability. We demonstrate that performance varies significantly between vendors, devices, and from the data-sheet. We also demonstrate and quantify some unexpected device characteristics and show how we can use them to improve responsiveness and energy consumption of solid state disks by 44% and 13%, respectively, as well as increase flash device lifetime by 5.2x.

1 Introduction

In recent years, flash memory has begun to make the transition from embedded devices to laptops, desktops, and data centers. It promises enormous performance gains and power savings relative to disk while being much denser and less power hungry than DRAM. However, fully exploiting these advantages requires overcoming flash memory's idiosyncrasies – it has limited durability, suffers from data integrity problems, and its reads, programs, and erases operate at mismatched granularities and have vastly different latencies.

As architects, our goal is to find ways to overcome these idiosyncrasies while exploiting flash memory's useful characteristics. To be successful, we must understand the trade-offs between flash memory's performance, cost (in power and dollars), and reliability. In addition, we must understand how different usage patterns affect these characteristics.

Unfortunately, flash manufacturers provide only vague guarantees about flash memory's performance. For instance, flash devices typically guarantee that their devices can be erased between 10,000 and 100,000 times, but this assumes a ten-year "shelf life" for the data, random access patterns, and a loosely-specified error correction scheme. Applications may require greater or lesser erase counts, different error correction capabilities, and a variety of storage longevity requirements. Likewise, manufacturers provide maximum power consumption numbers, but do not provide details of power consumption on a per-operation basis. This lack of detail complicates the design of systems which fully exploit flash memory's capabilities.

This paper empirically characterizes flash memory technology by measuring the performance, power, and reliability of flash devices from five manufacturers. We demonstrate that performance varies significantly between vendors, devices, and from the data sheet. We also demonstrate and quantify some unexpected device characteristics. Then we provide two

[*]The final version of this work will appear in The 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009 and will be available at http://nvsl.ucsd.edu in mid September 2009.

examples of how to apply this detailed understanding of flash performance. First we design an improved flash translation layer (FTL) that can reduce flash energy consumption by up to 13% during battery-powered operation and reduce latency for critical program operations by up to 44%. Second, we demonstrate how an alternative data encoding scheme effectively increases flash device lifetime by up to 5.2 times.

The remainder of this paper is organized as follows. Section 2 briefly describes flash memory technology. Section 3 describes our experimental setup for characterizing flash devices and presents our findings for different flash devices. Section 4 describes two possible applications using the insight from the data we collected. Section 5 concludes.

2 Flash memory

Flash memory has risen to prominence over the last decade due to the growing popularity of mobile devices with large storage requirements (iPods, digital cameras, etc.). Currently, 64Gb flash devices are available [7] with larger sizes on the way. Despite continued density scaling, the basic performance (read, program, and erase latencies) of flash devices has been roughly constant for over a decade. Density scaling may begin to wane as well, since flash faces significant challenges in further scaling [4].

In recent years, the architecture community has started to investigate flash's role in systems for a range of applications. These include hard disk caches [12, 5], solid-state disks [6], transactionalized SSD [13], mobile sensor networks [10], and data-centric computing [8]. Our goal is to provide additional insights in flash's behavior to enable further research in these and other directions.

2.1 Flash memory overview

Flash memory is a complex technology, and many factors impact its overall performance, reliability, and suitability for a particular application. Below we give a brief description of flash technology and terminology. Then, we describe the aspects of flash memory behavior that we quantify in this paper. These include the performance and power consumption of individual operations, and a range of recoverable and unrecoverable failure modes to which it is susceptible.

In this section, the facts and figures we provide for flash devices are typical values taken from publicly available data sheets. Values for specific devices are in Table 1 and Section 3.

Flash technology Flash memories store data as charge trapped on a floating gate between the control gate and the channel of a CMOS transistor. Each gate can store one or more bits of information depending on whether it is a single-level cell (SLC) or a multi-level cell (MLC). Commercially available devices store between 1 and 4 bits per cell [16, 17, 7]. Modern SLC and MLC flash devices achieve densities of 1 bit per $4F^2$ and 1 bit per $1F^2$ (for 4 bit-per-cell MLC devices) where F is the process feature size (currently 34nm [15]), allowing for very high-density flash arrays.

Internally, a flash chip is broken into one or more *planes* or *banks*. Depending on the device, planes are largely independent of one another, contain local buffering for read and program data, and perform some operations in parallel. Each plane, in turn, contains a set of *blocks*, each made up of 64 (SLC) or 128 (MLC) *pages*. Each page contains between 2112 and 8448 bytes. This includes a 2-8KB primary data area as well as an “out of band” data area used to store bad block information, ECC, and other meta-data.

NAND flash devices support three primary operations: erase, program, and read. Erase operates on entire blocks and sets all the bits in the block to 1. According to manufacturers' datasheets, it takes between 1.5 and 2 milliseconds to erase a block. Program operations write entire pages at once. The time to program a page includes moving the data over the pins and onto the device. Once the data are in an internal buffer, the datasheets claim that “typical” programming latencies are between 200 and 300 microseconds. Program operations can only change 1s to 0s, so an erase operation (of the entire block) is required to arbitrarily modify the page's contents. Read operations read an entire page in parallel. Reading data from the plane into the internal buffer reportedly takes between 25 and 30 microseconds.

In addition to these primary commands, flash devices also support a variety of other operations, such as copyback-read and copyback-program [2]. These commands increase performance by avoiding unnecessary operations or by skipping bus transfers between the controller and the chip.

Performance and power Currently, most flash devices transmit and receive data and commands over an 8- or 16-bit bus and can send and receive a new data word every 25-30ns (33-40Mhz). In theory, this interface could provide 40-80MB/s of bandwidth, but read and program latencies limit performance in practice. Unfortunately, the speed of this interface has not increased since 1995. Industrial efforts [1] are underway to remedy these problems and promise to raise peak bus bandwidth to between 133 and 400MB/s.

Read, program, and erase operations all require different amounts of power in flash devices. Datasheets give a maximum current draw of between 20mA and 30mA at 2.7-3.3V for a peak power of 50-100mW.

Reliability Flash memories can fail in a wide variety of ways. Most notoriously, the devices wear out with use. After many repetitions, the erase and program process can cause cells to become unreliable due to physical damage to the device. The

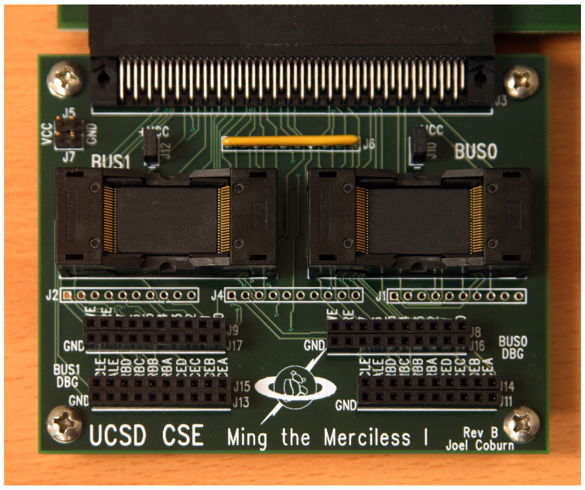

Figure 1: **Our flash testing board** The flash testing board can test two chips simultaneously and, combined with an FPGA board and a current meter, supports high-resolution timing and power measurements.

expected lifetime of one block in a flash device is 10,000 program/erase cycles for MLC and 100,000 for SLC. Furthermore, some devices retain data for only one year when programmed at this lifetime. Flash devices report erase and program failures, and manufactures recommend that the entire block be removed from service if any data within that block experience an error that cannot be corrected with ECC. To maximize the lifetime of a flash chip, flash systems use wear-leveling [9, 11, 18] to ensure that blocks are erased with equal frequency.

Bits can also become corrupt while stored in flash memory due to "read disturb" and "program disturb". Neither phenomenon causes permanent cell damage. Program disturb causes corruption because program operations on one page subject all the pages in the block to weak programming voltages. The effect is especially strong for the cells in adjacent pages. The effect is greatest for the pages immediately adjacent to the cells being programmed. To mitigate program disturb, flash manufacturers require (MLC) or strongly suggest (SLC) that pages within a block be programmed in order. This ensures that once a bit is written, it will only be subjected to one strong program disturbance.

Read disturb occurs because the voltages used to read data from a flash cell also have a weak programming effect on the other pages in the same block. As a result, data near pages that are frequently read can be degraded by millions of reads. To correct these and the other types of errors, flash systems must use ECC (stored in the out-of-band data section of each page).

3 Characterizing flash memory

To directly measure flash chip characteristics, we built a customized flash testing rig that gives us direct control of the devices' pins and provides facilities for measuring power consumption and operation. This section describes that hardware, the flash devices we used, the data we collected.

3.1 Data collection hardware

Figure 1 shows a photo of our flash characterization system. It comprises a custom-built daughter board attached to a Xilinx XUP board. The daughter board supports two flash chip test sockets with independent power planes and facilities for measuring the current that each chip consumes.

The FPGA on the XUP board implements a custom flash controller that provides support for timing measurements with 10ns resolution. The FPGA also hosts a full-blown Linux distribution. We use a user space application to drive the flash test boards and collect the results.

For power measurements we use a high-resolution current probe (Agilent 1147A) attached to an mixed-signal oscilloscope. The probe can measure current changes at up to 50Mhz, and the triggering capabilities of the scope allow us to capture data for individual flash operations.

3.2 Flash devices

Table 1 summarizes the flash devices we characterize in this study. They come from four manufacturers and cover a range of capacities. We suspect, but cannot verify, that they represent a range of implementation technologies.

Flash manufactures are guarded about the details of their devices (many do not publicly release the data sheets for their devices) and some flash devices themselves can be difficult to obtain in the small quantities we needed. We overcame these

Abbrev.	Manufa-cturer	Chips Measured	MLC or SLC	Cap. (GBit)	OOB (Bytes)	Page Size (B)	Pgs/ Blk	Blk/ Plane	Planes/ Die	Dies
A-SLC2	A	3	SLC	2	64	2048	64	1024	2	1
A-SLC4	A	3	SLC	4	64	2048	64	4096	1	1
A-SLC8	A	3	SLC	8	64	2048	64	4096	2	1
B-SLC2	B	3	SLC	2	64	2048	64	2048	1	1
B-SLC4	B	3	SLC	4	64	2048	64	2048	2	1
E-SLC8	E	3	SLC	8	64	2048	64	4096	1	2
B-MLC8	B	3	MLC	8	64	2048	128	4096	1	1
B-MLC32	B	3	MLC	32	128	4096	128	2048	2	2
C-MLC64	C	3	MLC	64	256	8192	128	4096	1	2
D-MLC32	D	3	MLC	32	128	4096	128	4096	1	2
E-MLC8	E	3	MLC	8	128	4096	128	1024	1	2

Table 1: **Parameters for the flash devices used in this study** We characterized nine devices from four manufactures.

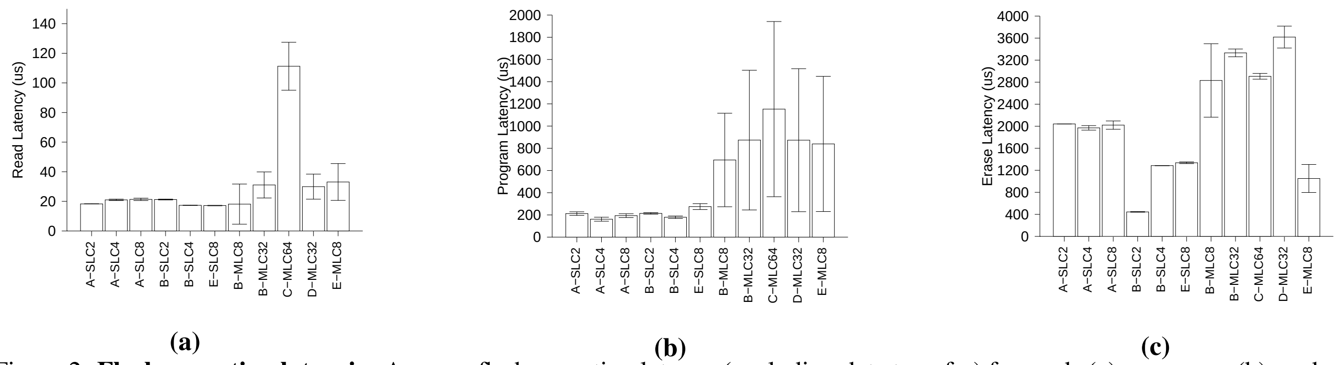

Figure 2: **Flash operation latencies** Average flash operation latency (excluding data transfer) for reads (a), programs (b), and erases (c) over ten program/erase cycles for 16 blocks on each chip. Error bars represent one standard deviation.

problems by purchasing flash chips from distributors when possible and removing them from commercially available flash-based USB "thumb drives" otherwise. We also built a simple protocol analyzer and programmable logic analyzer to reverse engineer the command sets for each flash device, since the command sets vary slightly between manufacturers.

We have elected not to reveal the manufacturers of our devices. We are not interested in calling on manufactures to account for the performance of their products. Rather, our goal is to understand the range of flash behavior so that we (and other researchers) can better understand flash memory's strengths and weaknesses.

3.3 Basic operation performance

We began by measuring the latency of the basic flash memory operations. Figure 2 shows the latencies for reads (a), programs (b), and erases (c). For each operation we measured the latency on 16 blocks on each of two chips for each chip model. The read latency varies little by manufacturer or chip (except for C-MLC64), and are in good agreement with datasheet values. MLC chips on average have longer and enormously variable program latencies, which we discuss in more detail below. Erase latency exhibits a smaller gap, but manufacturer B enjoys an advantage for SLC and E for MLC.

The first anomaly in our data is the variation in program time within each MLC block. All of the MLC devices we tested exhibited a regular and predictable variation in program latency between pages within a block. For instance, for B-MLC32 the first four pages and every other pair of pages in each block are 5.8 times faster on average than the other pages. The performance for these pages matches the "typical" values from the data sheet, but the other pages take well over a millisecond to program. For C-MLC64 every other page is fast. Figure 3 summarizes the results, and shows that the fast MLC programs are nearly as fast as SLC programs, while the slow pages are very slow indeed. In Section 4 we will demonstrate how exploiting this difference can significantly improve flash drive responsiveness and efficiency. SLC chips show no corresponding variability.

The second surprise in our investigation is that performance varies predictably as the devices begin to wear out. Figure 4

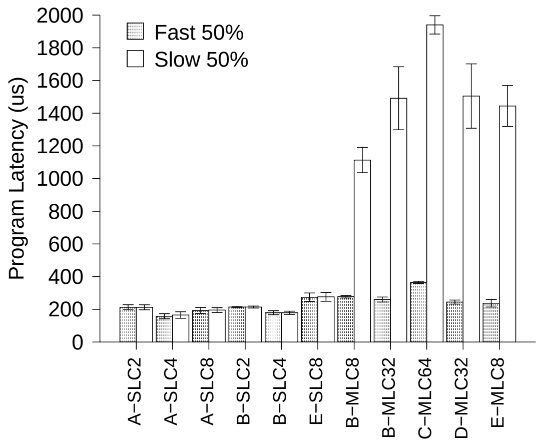

Figure 3: **The programing speed anomaly** Average programming speed for over ten program/erase cycles for 16 blocks on each chip. Programming speed varies dramatically between pages in MLC devices in a predictable pattern. SLC chips show no such variation. Error bars represent one standard deviation.

shows average program latency as a function of erase count for our SLC chips. The data show that program performance increases as the device wears out, resulting in nearly 50% faster program operations over the lifetime of an SLC device. MLC devices show much less variation: their performance increases by only 10-15%. We discuss a potential application of this phenomenon in Section 4.

Figure 5 summarizes the bandwidth that each device can deliver during single-plane operation. This value is a function of the operation latency, the page size, and the bus cycle time. For our experiments we used a 30ns cycle time for sending and receiving data to and from the chip. With a 20ns cycle time (the next faster clock available on our testing rig), none of the chips operated properly, although some are rated to 25ns. The motivation for MLC manufacturers to increase page size is clear: programming more bits in parallel allows them to nearly match SLC's programming bandwidth, despite their greater program latency.

3.3.1 Basic operation power consumption

A key advantage of flash memory over spinning disks is its low power consumption. Unfortunately, datasheet power numbers do not lead to good power modeling. We measured both peak and average power for our chips by using a high-speed amp-meter to measure the current draw while the chips performed various operations. Table 2 summarizes the results. These values represent the beginning of good power modeling.

The table presents peak power, average power, idle power, and per-operation energy for each operation. The average power is measured for a sequence of operations running as quickly as our test setup can drive the chip. We calculate peak power by dividing the energy for a single operation (measured using our amp meter) by its latency.

The table breaks out MLC energy by "fast" and "slow" pages and shows a disparity similar to the one we observed for program time. The pages that are fastest to program also consume dramatically less energy per operation (because program power is constant). Again, SLC chips show no page-to-page variation.

The table also shows that SLC enjoys a large efficiency advantage over MLC for all three operations as well as idle power. The exception is E-MLC8, whose remarkably small erase latency provides an correspondingly small erase energy. Excluding the erase energy of E-MLC8, MLC consumes 2.05, 2.70, and 1.13 times more energy per bit for read, program, and erase operations, respectively. They also consume 1.83 times more idle power, on average.

3.3.2 Reliability

One of the most oft-cited concerns about flash memory is its reliability. Flash memories can corrupt data in three main ways. The most important mechanism, wear-out, causes physical damage to the cells and is not reversible. The two remaining mechanisms, program disturb and read disturb, do not cause physical damage and are fully reversible. Manufacturers recommend that systems use error correction codes and access pattern restrictions to recover from or prevent all three types of errors.

The datasheets for flash chips provide a rating telling how many erase cycles a block can undergo before it is no longer reliable. For SLC chips this is typically 100,000 cycles, and for MLC it is typically 10,000.

Our results show that these ratings tell only part of the story. To measure the effect of wear on reliability, we stress-tested flash chips by performing 10 erase-program-read cycles in which we wrote and then read a random pattern looking for errors. Then, we performed 990 erase-program operations, also with random data. We repeated this process until we had reached 1

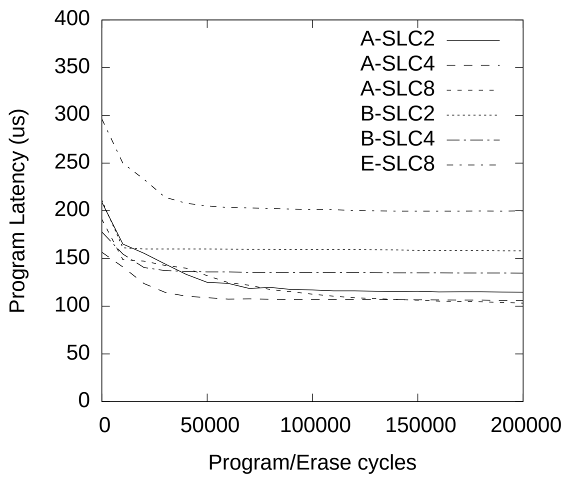

Figure 4: **Program performance variation over time for SLC devices** For SLC devices, average program time drops by nearly 50% over the 100,000 program/erase cycles of the chip. MLC devices (not shown) show a much smaller, 10-15% decline.

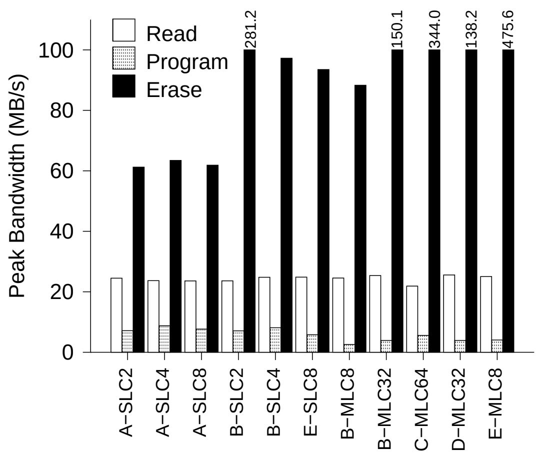

Figure 5: **Peak device bandwidth** The peak bandwidth that each device can deliver during single-plane operation.

	A-SLC2	A-SLC4	A-SLC8	B-SLC2	B-SLC4	E-SLC8
Peak Read Power (mW)	35.3	41.1	58.8	27.2	29.9	19.1
Peak Erase Power (mW)	30.9	35.5	47.6	25.3	20.0	25.5
Peak Program Power (mW)	55.2	59.9	78.4	35.0	35.0	56.0
Ave Read Power (mW)	10.3	14.0	21.0	7.4	11.0	18.8
Ave Erase Power (mW)	27.2	38.4	44.4	27.6	22.9	20.8
Ave Program Power (mW)	27.9	32.4	50.1	19.6	20.8	37.5
Idle Power (mW)	2.7	7.1	17.0	2.9	2.9	13.3
Read Energy (nJ/bit)	0.055	0.067	0.098	0.049	0.043	0.0061
Program Energy (nJ/bit)	0.72	0.61	0.85	0.48	0.41	1.01
Erase Energy (nJ/bit)	0.06	0.067	0.093	0.011	0.025	0.031
	B-MLC8	B-MLC32	C-MLC64	D-MLC32	E-MLC8	
Peak Read Power (mW)	54.0	75.9	112.0	66.3	13.4	
Peak Erase Power (mW)	42.4	70.6	111.8	57.0	21.3	
Peak Program Power (mW)	58.9	94.7	132.2	82.3	118.4	
Ave Read Power (mW)	18.1	31.1	41.5	28.3	21.3	
Ave Erase Power (mW)	45.5	53.0	105.0	56.2	23.5	
Ave Program Power (mW)	46.5	52.5	77.0	55.6	40.9	
Idle Power (mW)	12.7	8.5	27.3	11.2	10.2	
Read Energy (nJ/bit)	0.15	0.11	0.19	0.095	0.0031	
Fast Program Energy (nJ/bit)	1.09	0.96	0.67	0.79	0.46	
Slow Program Energy (nJ/bit)	3.31	3.30	2.86	2.84	2.07	
Erase Energy (nJ/bit)	0.070	0.056	0.038	0.051	0.0057	

Table 2: **Power and energy consumption for flash operations** Peak values are taken from measuring consumption during a single operation with our high-resolution amp meter. Average values are taken over multiple operations on our test system.

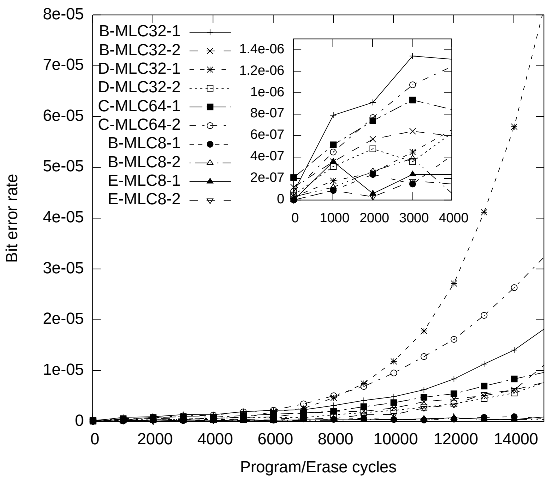

Figure 6: **Wear out and error rate for MLC devices** The error rate for MLC devices increases dramatically with wear, and is non-zero even for brand-new devices (inset). There is also large variation in the change in error rate between instances of the same chip (notably D-MLC32-1 and D-MLC32-2).

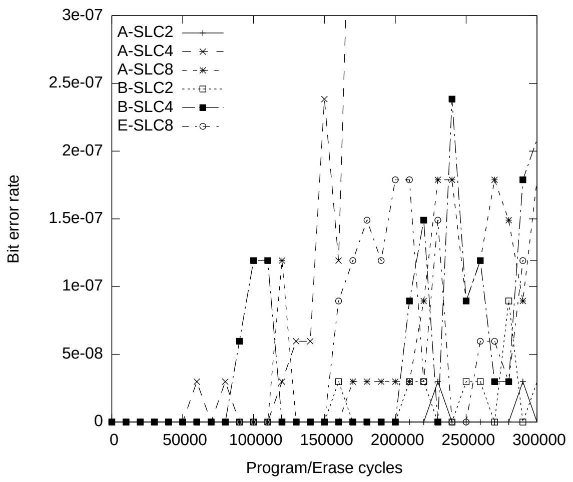

Figure 7: **Wearout and error rate for SLC devices** For SLC devices, the error rates are several orders of magnitude lower than for MLC even at three times their rated lifetimes. Only A-SLC4 shows a significant increase.

million erases for SLC chips and 100,000 erases for MLC chips.

Figures 6 and 7 show the error rate for each chip. The difference between SLC and MLC is stark. MLC devices show significant error rates from the very beginning. For most of the MLC models, the error rate increases sharply shortly after their rated lifetime, and some start to increase sharply even earlier. SLC devices, by contrast, show almost zero errors until they reach their rated lifetime and maintain reasonably low rates for up to six times their rated lifetime.

It is important to realize that these data do not include any effects on the "shelf life" of the data stored in these devices. However, not all applications require the 10 year data retention time that manufacturers aim to provide. Our data show that for such applications, the maximum number of erase/program cycles before SLC chips become unusable may be much higher.

The data also show a marked difference in reliability among pages in MLC devices. Figure 8 plots the total error rate for four chips over their rated lifetime of 10,000 program/erase cycles. Interestingly, although roughly half the pages in a block are significantly more reliable than the others, and there seems to be a pattern within the block, there is not a consistent correlation between program speed and reliability. SLC devices show similar variation, but error rates do not become significant until long past their rated lifetime.

Program disturb To quantify program disturb, we erased a block and repeatedly programmed half of one page to 0. After each program we measured the number of unintentionally programmed bits in the block. For SLC devices, we observed no program disturb for the first few iterations. At this point, several of the chips developed distinctive patterns of errors. B-SLC4, A-SLC2 and A-SLC4 had increasing errors on every other page with no errors on the re-programmed page or the two adjacent pages. E-SLC8 developed errors only in the first few pages. B-SLC2 and A-SLC8 developed no clear pattern. Figure 9 shows a representative sample of these patterns.

For MLC devices, the results were more immediate. For all of the MLC chips, performing just one repeat program revealed two distinct patterns of errors. For C-MLC64, reprogramming pages $2n$ would disturb nearly all the bits in page $2n - 3$, for the midsection of the block. Additionally, 32 bit errors occur on the reprogrammed block itself. For the other chips, reprogramming pages $4n$ or $4n + 1$ disturbs nearly all the bits in pages $4n - 6$ and $4n - 5$ except at the beginning and end of the block.

Read disturb To measure the prevalence of read disturb, we wrote a test pattern to several blocks on the flash chip and then repeatedly read the pattern back, checking to see if any errors had appeared. Figure 10 graphs the results. Our data show that, in most cases, the effects of read disturb began to appear between 250 thousand and 4.8 million repeated reads.

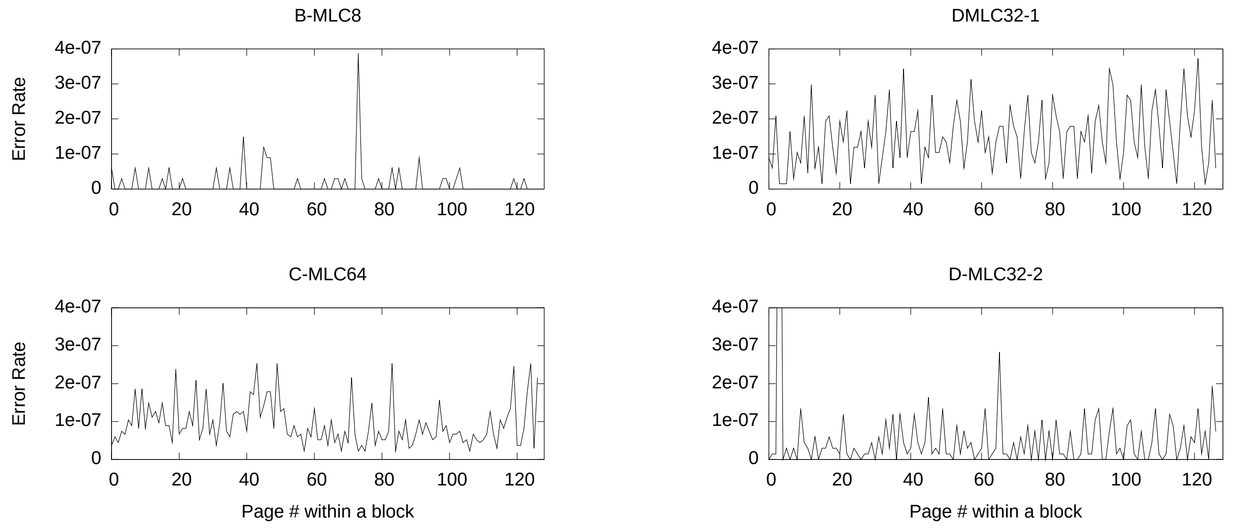

Figure 8: **Per-page error rates for MLC devices** MLC chips show large variation in error rates among pages in a single block. The y-axis measures the total raw error rate over the chips' rated lifetime. The two chips at right are supposedly identical parts, but show very different error rates.

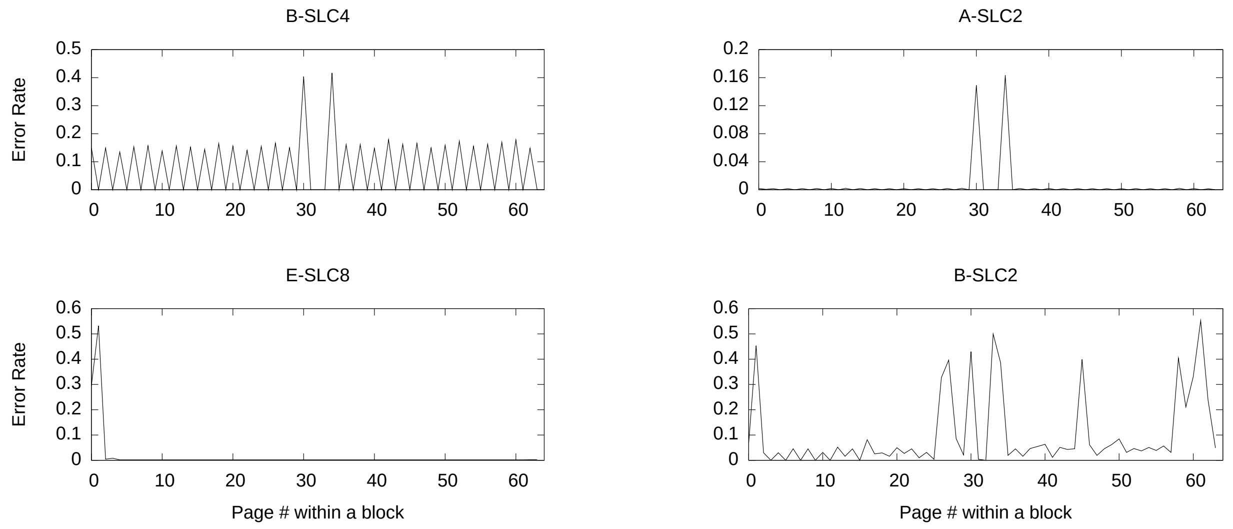

Figure 9: **Program Disturb for SLC devices after 10,000 reprograms of page 32** Varied error patterns emerge when a page is repeatedly reprogrammed. The reprogrammed page consistently shows no errors.

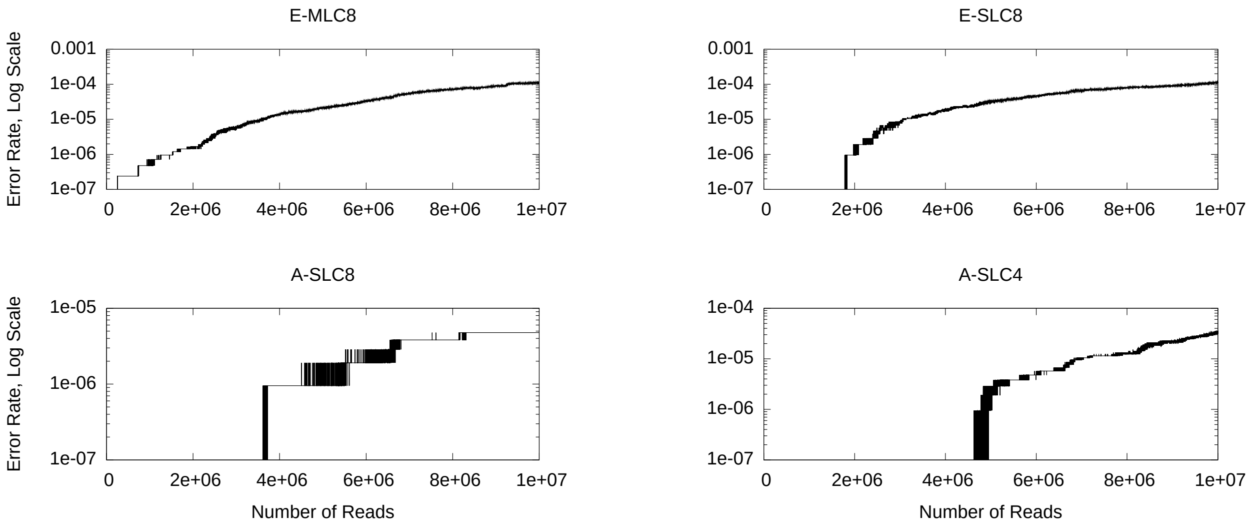

Figure 10: **Read disturb up to 10 Million re-reads** Repeatedly reading a page without refreshing the data causes a steadily increasing number of errors beginning at between 250 thousand and 4.8 million reads.

4 Applications

The ultimate goal of this work is to understand the performance of flash memories so that we can identify new ways to exploit their performance and overcome their limitations. The following sections demonstrate how we can apply the understanding that Section 3 provides to accomplish this.

We explore two applications. The first is a new flash translation layer (FTL) that takes advantage of the fast/slow page phenomenon to increase responsiveness and decrease power consumption. The second is the application of a different data encoding scheme to increase flash's effective lifetime. These applications demonstrate that understanding flash's characteristics in detail can lead to significant system-level improvements.

4.1 A variation-aware FTL

The data in Section 3 demonstrated two sources of variation in program time. The first was the wide variation in program speed and energy consumption between the "fast" and "slow" pages in MLC devices. The second was the change in SLC program latency as the chips aged. We have developed a flash translation layer (FTL) to exploit the first variation to improve performance and/or power efficiency.

We used the FTL described in [6] as the baseline FTL. The FTL provides a disk block-based interface while distributing erase and program operations evenly across the flash memory to ensure uniform wear-out. The FTL maintains a map between logical block addresses (LBAs) and physical flash addresses (PFAs).

The FTL maintains a "write point" at which all program operations occur. With each new write request, the FTL writes the new data at the current write point and updates the map so that the LBA in the write request points to the data that were just written. It then advances the write point.

When the write point reaches the end of a block, the FTL must locate a new, erased block for a new write point. It keeps a pool of erased blocks for this purpose. If the pool falls below a "low water mark," the FTL will perform a garbage collection in which it finds a block with some invalid data (i.e., data for which a newer version exists elsewhere in the array), copies the valid data to the current write point, and erases the block. The FTL repeats this process until the pool reaches a "medium water mark." When the number of empty blocks reaches the "high water mark," the FTL stops background cleaning to avoid unnecessary erases.

To exploit the variation in program time between the fast and slow pages, we developed an FTL called *Mango* that skips slow pages for improved performance or power/energy efficiency.

Mango adds a priority to incoming IO requests. For high-priority writes, the FTL will do its best to use fast pages. The FTL also provides a fast garbage collection mode that uses fast pages for garbage collecting write operations as well. Since

Name	Description	Mean Req. KB		%
		Read	Write	Reads
Build	Compile Linux kernel	3.7	4.0	12.14
Financial [3]	Live OLTP trace	2.3	3.7	15.40
DesktopDev	24 hour software development desktop trace	17.0	4.0	65.81
SwapSpace	Virtual memory for desktop applications	7.8	4.0	84.07

Table 3: **Workloads for evaluating Mango** The traces cover a wide range of application types.

the fast pages are also lower energy, a single priority is sufficient to implement those policies as well.

To find a fast page, Mango uses the next fast page at the current write point for MLC devices. For SLC devices, it could maintain two write points (using techniques described in [8]), one in an old, fast block and one in a young, slow block. We present results here for the MLC case.

There are several dangers in this scheme. The first is increased wear for MLC devices because skipping half of the pages in each block means that we will need to erase blocks more frequently. The second danger is that ignoring pages will increase the frequency of garbage collection and ultimately increase latency for disk-bound applications.

Finally, there is a limitation on how many pages we can skip. Skipped pages appear as invalid pages, and at any time there must be enough valid pages to account for the full, advertised capacity of the SSD. The result is that in some cases, the FTL can be forced to use slow pages during garbage collection.

We evaluate our new FTL in two different scenarios:

- **Swap** This scenario uses high-priority accesses for write requests for paging out virtual memory pages. We interleaved requests from the SwapSpace (see 3) trace with requests from other traces so that swap requests accounted for between 5 and 20% of the total requests. We measure the average latency for swap and non-swap requests separately. Garbage collection occurs in “slow” mode, unless it is required to service a high-priority request.
- **Netbook** This scenario models a mobile device in which energy saving is key. When the device is running from its battery, all operations are tagged as high priority and the FTL is always in fast mode. When the device is running off of wall power, all operations are low priority and garbage collection occurs in slow mode. To model this scenario, we switch from battery to wall power at irregular intervals so that half of the trace is processed in each mode.

In each scenario we use a set of traces summarized in Table 3. The traces are from several sources and represent a wide range of application behaviors. The traces include operation arrival times, and we use that to schedule arrival times at the SSD unless otherwise noted.

We implemented Mango in a flash memory simulator using the performance and power measurements from Section 3.3 to determine the latency for each operation (including garbage collection time if the operation needed to wait for it) and the overall energy consumption for the trace. The simulator collects statistics on the distribution of high and low priority accesses due to external IO requests and internal garbage collection operations. It also measures the fraction of slow pages that the FTL skipped during the trace and the total number of erases performed (to gauge the amount of wear caused by the trace).

The simulator models an SSD comprised of a single flash chip. For this study we used the data for chip C-MLC64 from Section 3, since the power and performance for the other MLC devices were comparable, we would expect similar performance and energy results.

Figure 11 summarizes the results for the swap scenario and compares the responsiveness and power consumption of Mango with the baseline FTL. The data show that Mango achieves its goal of reducing swap latency: On average swap write requests complete 1.5 times more quickly with Mango than with the baseline.

As we expected, the downside of increasing priority on some requests is increased overall wear. In this case, wear increased by an average of 3% across the traces.

Figure 12 shows the results for the netbook scenario. Here, we are most concerned about overall energy consumption, and Mango reduces energy consumption compared to the baseline by 3% on average while increasing wear by 55%. This increase is larger than for the swap scenario because a larger fraction of accesses were high priority.

4.2 Flash-aware data encoding

Flash devices require the use of error correction codes to detect and recover from errors caused by wear, program-disturb, and read-disturb effects. Their performance, as well as their lifetime, can also be improved by using alternative *data encoding* schemes.

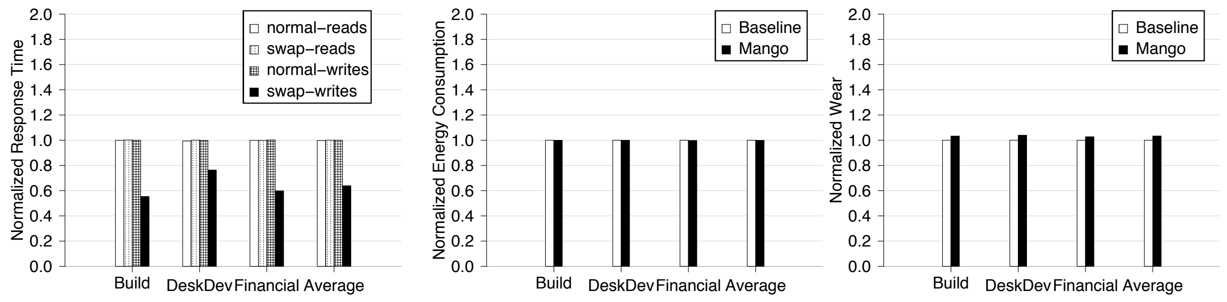

Figure 11: **Mango response time for the swap scenario** Mango is able to significantly increase responsiveness for swap requests while only marginally increasing energy consumption and increasing wear by only 3%.

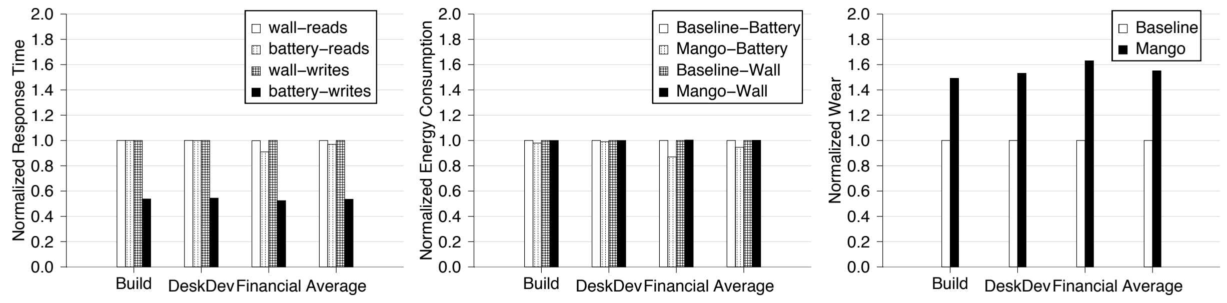

Figure 12: **Mango results for the Netbook scenario** Mango is able to significantly reduce the energy drain on the battery without significantly increasing energy while plugged in. It also realizes a slight performance increase, while increasing wear by 55%.

The data in Section 3 provide us with the means to evaluate the impact of different encoding schemes on flash longevity. To demonstrate this, we have implemented and evaluated the performance of a simple write-once-memory (WOM) coding scheme for flash memory [14].

WOM codes were originally developed for storage devices - punch cards and early digital optical disks, for example - in which a stored '1' could not be changed to a '0'. This property would nominally prevent the user from writing more than once onto any given area of the storage medium. WOM codes provided a method to overcome this limitation, allowing a trade-off between the number of writes and the recorded data density (the number of logical bits stored in a physical bit location on the medium).

Figure 13 illustrates a simple WOM code that we have implemented. It uses 3 physical bits to represent 2 logical bits and allows two sets of bits to be written. Each sequence of two bits has two representations, one for the first program and one for the second. These are the *first and second generation code words*. The key to the code is that, with one exception, the 1s in each second generation code word are a superset of the 1s in all of the first generation code words. As a result, overwriting a first generation word with a second generation word always results in the second generation code word being stored. The exception is that first and second generation code words for the same bit pair are complements, so writing the same logical bits at both generations will result in a '111' physical bit pattern. This leads to ambiguity in decoding, which the code resolves by reading the data before programming it and only reprogramming logical bits that have changed.

In the context of flash memory, this encoding scheme allows us to write data to a block twice before erasing it. The data in Section 3 showed that, for MLC devices, program disturb is only a problem for programing half of the pages in a block. Those pages are unsuitable for WOM codes. The other pages, however, can accept multiple programs with no ill effects. We refer to these pages as "WOM safe." For SLC devices, program disturb is not a problem for the first few iterations on any page, therefore all page are WOM safe.

The writing procedure is as follows. Initially, we program unencoded data into the non-WOM safe pages and first-generation WOM-encoded data into the WOM-safe pages. On the second programming pass, we program second-generation WOM-encoded data into just the WOM-safe pages. WOM-enconding is 66% efficient, so the two writes to the WOM-safe pages combined with a single write to unsafe pages gives a 1.16 times increase in the number of bytes that can be written to a block per erase operation for MLC. For SLC, there is a factor of 1.33 increase.

This leads to two favorable trade-offs. First, WOM codes allow the chip to expend less energy to program a given amount of data. This is because the energy to erase is not required as frequently with respect to writes. Figure 14 displays these

Logical bits	First generation	Second generation
00	111	000
01	110	001
10	101	010
11	011	100

Figure 13: **The write-only memory code** Write-only memory codes (WOM codes) allow multiple logical value to be written even if physical bits can only transition once.

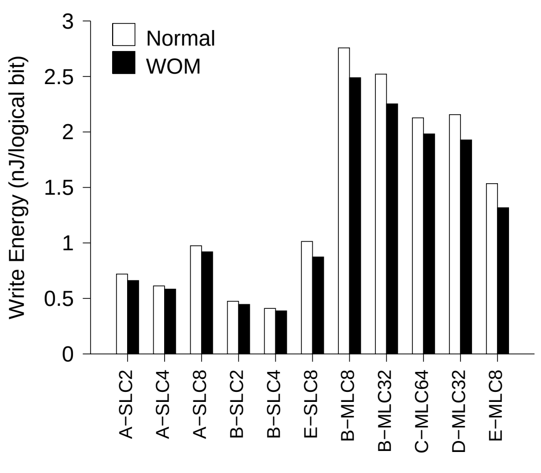

Figure 14: **Effective program energy** Because WOM codes decrease the number of erases per logical bit programmed, using them reduces programming energy by 9.5% on average.

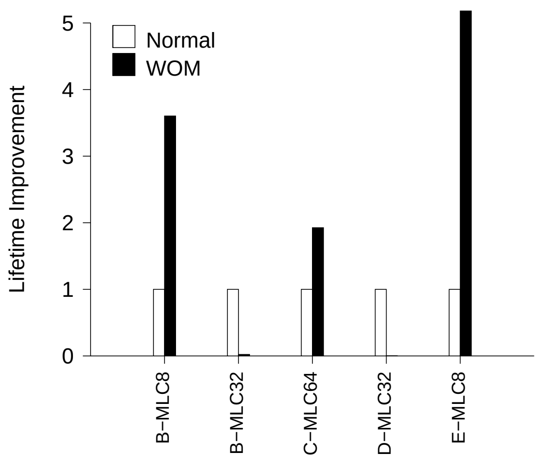

Figure 15: **Effective device lifetime** Measured in logical bytes written before the device reaches the fatal error rate, WOM codes allows up to 5.2 times longer lifetime.

energy savings for each chip.

The second measurable advantage of using WOM codes is a commensurate increase in useful device lifetime. We measure this as the amount of logical data written to the device before begins to experience the *fatal error rate*. This is the error rate at the recommended lifetime of the device under normal operation (without reprogramming or WOM-encoding).

Figure 15 shows this increase in effective lifetime. SLC chips are not graphed because the WOM-encoded chips showed no errors even after programming 13.3 times more data. We would expect a 17% (MLC) or 33% (SLC) increase in the number of bytes programmed for every erase, but several of the chips far exceed this expectation. There are several possible explanations. For example, reprogramming bits to the same value may reinforce the data or the WOM codes may have some other error-reducing properties. These are questions we are still exploring.

5 Conclusion

The devices we characterized in this study exhibited variation both within a block and over time in terms of power consumption, latency, and error rates. Our data also show that the datasheet values that manufacturers provide often tell only part of the story, and that actual performance can be significantly worse and highly variable. Our application case studies demonstrate that by looking beyond the datasheets manufacturers provide, we can make significant improvements to flash-based storage devices. Exploiting two of the effects we measured enabled us to significantly decrease latency for critical IO requests and extend the effective lifetimes of chips.

References

[1] Onfi: Open nand flash interface. http://onfi.org/specifications.

[2] Onfi: Open nand flash interface specification 2.0. http://onfi.org/wp-content/uploads/2009/02/onfi_2_0_gold.pdf.

[3] Umass trace repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[4] International technology roadmap for semiconductors: Emerging research devices, 2007.

[5] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *ASPLOS-V: Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 10–22, New York, NY, USA, 1992. ACM.

[6] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. Technical Report MSR-TR-2005-176, Microsoft Research, December 2005.

[7] e. a. C. Trinh. A 5.6mb/s 64gb 4b/cell nand flash memory in 43nm cmos. In *Solid-State Circuits Conference*. IEEE, 2009.

[8] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. *SIGPLAN Not.*, 44(3):217–228, 2009.

[9] L.-P. Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1126–1130, New York, NY, USA, 2007. ACM.

[10] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107, New York, NY, USA, 2002. ACM.

[11] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In *CASES '07: Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 160–164, New York, NY, USA, 2007. ACM.

[12] T. Kgil, D. Roberts, and T. Mudge. Improving nand flash based disk caches. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 327–338, Washington, DC, USA, 2008. IEEE Computer Society.

[13] V. Prabhakaran, T. L. Rodeheffer, and L. Zhou. Transactional flash. *USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[14] R. Rivest and A. Shamir. How to reuse a write-once memory. *Information and control*, 55:1–19, December 1982.

[15] e. a. R.W. Zeng. A 172mm^2 32gb mlc nand flash memory in 34nm cmos. In *Solid-State Circuits Conference*. IEEE, 2009.

[16] e. a. S. Chang. A 48nm 32gb 8-level nand flash memory with 5.5mb/s program throughput. In *Solid-State Circuits Conference*. IEEE, 2009.

[17] e. a. T. Futatsuyama. A 113mm^2 32gb 3b/cell nand flash memory. In *Solid-State Circuits Conference*. IEEE, 2009.

[18] D. Woodhouse. Jffs2: The journalling flash file system, version 2. http://sources.redhat.com/jffs2/.