UNIVERSITY OF CALIFORNIA, SAN DIEGO

Symbiotic Solid State Drives: Management of Modern NAND Flash Memory

A dissertation submitted in partial satisfaction of the
requirements for the degree of Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Laura Marie Caulfield

Committee in charge:

Steven Swanson, Chair
Chung-Kuan Cheng
John D. Davis
Rajesh Gupta
Paul Siegel
Michael Taylor

2013

The Dissertation of Laura Marie Caulfield is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2013

EPIGRAPH

Nobody ever figures out what life is all about, and it doesn't matter. Explore the world.
Nearly everything is really interesting if you go into it deeply enough.

–Richard P. Feynman

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am grateful for the support of many people during the many years leading to this dissertation.

First, my deepest thanks to my advisers, Steven Swanson and John D. Davis, for their constant optimism and enthusiasm to explore new ideas. Their guidance in research through late nights and endless supplies of pizza produced not only this thesis, but a life-long researcher. I am also grateful for the opportunities afforded to me through their dedication to helping me develop my career.

This thesis is the product of many collaborations with countless researchers and engineers. I would like to thank all of my co-authors, colleagues from the Non-Volatile Systems Lab, officemates and friends in the Computer Science and Engineering Department of the University of California, San Diego.

Finally, I would like to extend my heartfelt gratitude to my family – to my brother for his kindness and wisdom, to my parents for their steadfast love and to my husband for his remarkable combination of patience and wit.

Chapters 2, 6, and 7 contain material from "Characterizing Flash Memory: Anomalies, Observations, and Applications" by Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Eitan Yaakobi, Steven Swanson, Paul H. Siegel, and Jack K. Wolf, which appears in *Micro'09: the 42nd Annual International Symposium on Microarchitecture*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2009 by the Institute of Electrical and Electronics Engineers (IEEE).

Chapters 2 and 5 contain material from "Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications" by Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson which appears in *ASPLOS'09: The 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. The dissertation author was the second investigator and author of this paper. This material is copyright ©2009 by the Association for Computing Machinery, Inc. (ACM).

Chapter 3 contains material from “The Bleak Future of NAND Flash Memory” by Laura M. Grupp, John D. Davis and Steven Swanson, which appears in *FAST ’12: The 10th USENIX Conference on File and Storage Technologies*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2012 by the Association for Computing Machinery, Inc. (ACM).

Chapter 6 contains material from “Reliably Erasing Data From Flash-Based Solid State Drives” by Michael Wei, Laura M. Grupp, Fredrick E. Spada, and Steven Swanson, which will appear in *FAST’11: The 9th USENIX Conference on File and Storage Technologies*. The dissertation author was the second investigator and author of this paper. This material is copyright ©2011 by the Association for Computing Machinery, Inc. (ACM).

Chapters 7 and 4 contain material from “The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs” by Laura M. Grupp, John D. Davis and Steven Swanson, which will appear in *USENIX’13: USENIX Annual Technical Conference*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2013 by the Association for Computing Machinery, Inc. (ACM).

VITA

2007	Bachelor of Electrical Engineering, University of Washington
2007–2013	Research Assistant, University of California, San Diego
2010	Master of Science, University of California, San Diego
2012	Candidate of Philosophy, University of California, San Diego
2013	Doctor of Philosophy, University of California, San Diego

PUBLICATIONS

Laura M. Grupp, John D. Davis and Steven Swanson. “The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs” To Appear In: USENIX Annual Technical Conference, June 2013.

Hung-Wei Tseng, Laura M. Grupp and Steven Swanson “Underpowering NAND Flash: Profits and Perils” 50th Design Automation Conference (DAC 2013), June 2013.

Vidyabhushan Mohan, Trevor Bunker, Laura M. Grupp, Sudhanva Gurumurthi, Mircea R. Stan and Steven Swanson. “Modeling Po wer Consumption of NAND Flash Memories using FlashPower” Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013.

Ryan Gabrys, Eitan Yaakobi, Laura M. Grupp, Steven Swanson and Lara Dolecek. “Tackling Intracell Variability in TLC Flash through Tensor Product Codes” Proceedings of the IEEE International Symposium on Information Theory, 2012.

Laura M. Grupp, John D. Davis and Steven Swanson. “The Bleak Future of NAND Flash Memory” Proceedings of the 10th USENIX Conference on File and Storage Technologies, February 2012.

Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Key Yu, G. Edward Suh, Edwin Kan and Steven Swanson. “Extracting Device Fingerprints from Flash Memory Exploiting Physical Variations” Proceedings of the 4th International Conference on Trust and Trustworthy Computing, 2011.

Hung-Wei Tseng, Laura M. Grupp and Steven Swanson. “Understanding the Impact of Power Loss on Flash Memory” 48th Design Automation Conference (DAC 2011), June 2011.

Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala and Steven Swanson. “NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories” Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011.

Michael Wei, Laura M. Grupp, Frederick E. Spada and Steven Swanson. “Reliably Erasing Data From Flash-Based Solid State Drives” Proceedings of the 9th USENIX Conference on File and Storage Technologies, 2011.

Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, John D. Davis and Steven Swanson. “Beyond the Datasheet: Using Test Beds to Probe Non-volatile Memories’ Dark Secrets” IEEE Global Communications, December 2010.

Eitan Yaakobi, Paul H. Siegel, Steven Swanson, Jack K. Wolf, Laura M. Grupp and Jing Ma. “Error Characterization and Coding Schemes for Flash Memories” IEEE Global Communications, December 2010.

Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siefel and Jack K. Wolf. “Characterizing Flash Memory: Anomalies, Observations, and Applications” MICRO ’09: Proceedings of the 42nd Annual International Symposium on Microarchitecture, IEEE/ACM, 2009.

Adrian M. Caulfield, Laura M. Grupp and Steven Swanson. “An Improved Architecture for Data-Intensive Applications” Micro Top Picks ’09: IEEE Micro Top Picks from Computer Architecture Conferences , ACM, 2009.

Adrian M. Caulfield, Laura M. Grupp and Steven Swanson. “Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications” ASPLOS 2009: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, pp 217–228, March 2009.

ABSTRACT OF THE DISSERTATION

Symbiotic Solid State Drives: Management of Modern NAND Flash Memory

by

Laura Marie Caulfield

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California, San Diego, 2013

Steven Swanson, Chair

As our society becomes more information-driven, we have begun to amass data at an astounding and accelerating rate. At the same time, power concerns have made it difficult to bring the necessary processing power to bear on querying, processing and understanding this data. In light of this, system designers have begun to adopt high density NAND flash memory as the solution for storing data at low power. However, our knowledge about the trade-offs in managing the technology is in its infancy.

In this work, we empirically characterize a representative selection of NAND flash memory technology by directly measuring its performance, power and reliability.

We show these properties vary significantly from publicly available information, that most metrics are failing as density increases and that symbiotic coordination between device and application variations holds the key to designing modern storage systems.

We demonstrate how to improve the following properties of flash-based solid state drives: decreased latency of critical IO requests by 44%, decreased energy consumption by 13%, increased lifetime by up to 5.2x, decreased latency of single-file erasure by more than 95%, increased performance of bursts by 36% and increased steady state performance by 95%.

Chapter 1

Introduction

Our world is overflowing with data – from the handheld to the data center, we are collecting and analyzing ever greater amounts of information. Companies like Google, Facebook and Microsoft routinely process many petabyes of data, and users of desktop search engines routinely pose queries across the terabytes of data stored on their hard drives. There is no reason to expect our appetite for collecting and processing data to stop growing at its current, breakneck rate.

To satiate our appetite for large-scale data processing, current storage technology must overcome two challenges. First, while hard drive capacity continues to grow, the latency and bandwidth that hard drives can deliver do not. Second, power constraints due to cooling, economic and ecological concerns severely limit the range of possible solutions.

For data-centric computing, the most fundamental advancement has been the rise of solid-state storage. Flash memory's performance characteristics enable systems far outside the design space covered by existing technologies such as conventional servers, processor-in-disk and processor-in-memory systems. The highest density flash memories available today (or in the near future) offer 16x the density per package of DRAM at one 16th the power. [1] In the near future, an array of four flash packages will be able to deliver over 7x the write bandwidth of a high-end disk at 1/30th the power and a fraction of the latency. These advantages, combined with the fact that solid state storage arrays comprise many discrete chips, instead of a few large drives, provide vastly more flexibility in the architecture of a combined computing and storage platform.

Despite flash's promise, fully exploiting its advantages requires overcoming flash memory's idiosyncrasies – it has limited durability, suffers from data integrity problems, and its read, program and erase operations function 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.

The community's understanding of these trade-offs is still in its infancy. Flash manufacturers provide conservative and often vague guarantees about flash memory's performance and more detailed information, when it exists, is a tightly held trade secret. For instance, flash devices typically guarantee that their devices can be erased between 5,000 and 1,000,000 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 work empirically characterizes flash memory technology by measuring the performance, power, and reliability of flash chips and then proposes new techniques and applications for more efficient and effective use of NAND Flash Memory. We demonstrate that symbiotic coordination of application and device variations give storage system designers the tools needed to leverage modern flash-based storage systems.

In Chapter 2 we first provide background on the operation of flash memory chips, and then demonstrate that performance varies significantly between vendors, devices, and from the publicly available datasheets. We also demonstrate and quantify some unexpected device characteristics.

In Chapter 3, we then turn our attention to how flash technology is changing. While flash density in terms of $bits/mm^2$ and feature size scaling continues to increase rapidly, all other figures of merit for flash – performance, program/erase endurance, energy efficiency, and data retention time – decline steeply as density rises.

The data and conclusions drawn in Chapters 2 and 3 direct our management improvements for the remainder of the thesis – to provide new application opportunities and to offset the negative trends caused by density scaling.

Chapter 4 describes the basics of flash management techniques (i.e. Flash Translation Layers or *FTLs*) and our FTL simulation infrastructure, followed by an FTL design which leverages the parallelism inherent in SSD hardware in Chapter 5. The new design is the storage system for *Gordon*, a flash-based system architecture for massively parallel, data-centric computing. Our results show that Gordon systems can deliver up to 2.5x the computation per energy of a conventional cluster-based system while increasing performance by a factor of up to 1.5. We also demonstrate that our flash management system can deliver up to 900MB/s of read and write bandwidth.

Our empirical characterization of flash chips leads to a host of new ways of leveraging flash's unique characteristics. Chapter 6 describes two such applications, which leverage our understanding of flash's limited reprogrammability in detail. We first present an encoding scheme which can improve SSDs' lifetime by up to 5.2x and energy consumption by 9.5%. Then (because the layer of indirection in FTLs obscures systems' ability to reliably erase data from the drive) we propose an efficient page-sized erase operation for the limited circumstances of drive sanitization. With this mechanism, the FTL can erase single files 130x more quickly than without, effectively making single-file erasure possible.

Chapter 7 describes mechanisms which allow us to achieve the performance of low-density flash while still using the capacity of high-density flash. First, we design an improved FTL that can reduce flash energy consumption during battery-powered operation and reduce latency for critical write operations by up to 44%. This technique comes with little added complexity. Next, we describe an FTL design that achieves similar results with no added wear, scales well to large numbers of user accesses and ultimately provides up to 100% of the performance of low-density flash.

Finally, Chapter 8 describes the author's ideas for future developments for the management of flash and other storage technologies and Chapter 9 concludes the thesis.

Chapter 2

Flash Chip Organization and Behavior

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, laptops and tablets etc.). Currently, 128Gb flash devices are available [28] with larger sizes on the way. At the beginning of this dissertation work, flash's density continued to scale while the basic performance (read, program, and erase latencies) of flash devices had been roughly constant for over a decade. However, as we show in Chapter 3, these trends have changed as flash faces significant challenges in further scaling [5].

In recent years, the architecture community has investigates flash's role in systems for a range of applications. These include hard disk caches [47, 15], solid-state disks [17], transactionalized SSDs [64], and mobile sensor networks [40]. However, neither the research community nor industry provides detailed information about flash's behavior and the trade-offs available for varied use of the technology. One major contribution of this dissertation is to fill this gap. Our goal is to provide additional insights in flash's behavior to enable further research in these and other directions.

This chapter first provides (in Section 2.1) background on the basic operation and organization of NAND flash memory chips. Section 2.2 describes our experimental setup for characterizing flash chips. Finally, Section 2.3 presents our findings for different flash devices.

Table 2.1. Values Reported in Flash Chip Datasheets This is a summary of publicly available information from flash manufacturers. The ranges include information from datasheets for chips we have not measured.

	Minimum	Maximum
Endurance	500	1,100,000
Rand Read Latency (μs)	12	200
Typ Program Latency (μs)	200	800
Max Program Latency (μs)	500	2,000
Typ Erase Latency (ms)	1.5	2.5
Max Erase Latency (ms)	2	10
Typ Read Power (mW)	30	66
Max Read Power (mW)	60	165
Typ Program Power (mW)	30	66
Max Program Power (mW)	60	165
Typ Erase Power (mW)	30	66
Max Erase Power (mW)	60	165
Typ Idle Power (μW)	30	60
Max Idle Power (μW)	150	300

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 thesis. 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 2.2 and Section 2.3.

2.1.1 Flash Technology

Each flash memory cell is a CMOS transistor with a variable threshold voltage. Flash stores data as charge trapped on a metal layer (called the *floating gate*) embedded in the oxide between the control gate and the channel. The cell's threshold voltage depends on the number of electrons stored on the floating gate. Writing (aka *programming*) and erasing the cell requires high voltages to move electrons through the oxide, on to and off of the floating gate.

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 [30, 31, 73]. Modern SLC and MLC flash devices achieve densities between 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 20 nm [28]), allowing for very high-density flash arrays. Emerging 3D flash technologies [32, 43] promise to push densities even higher.

High-density flash applications use NAND flash devices, which provide less flexible access to data than lower-density, NOR flash. We only consider NAND flash in this thesis.

2.1.2 Organization

Flash memories are organized differently than SRAM or DRAM both logically and physically. A flash device contains a set of *blocks* each made up of 64 (SLC), 128 (MLC) or 384 (TLC) *pages*. Each page contains 2112 to 8448 bytes. This includes a 2048- to 8192-byte primary data area as well as an "out of band" data area used to store bad block information, ECC, and other meta-data.

Devices typically divide blocks among two or four "planes." 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, although they contend for the package pins.

2.1.3 Operations

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. Program operations write entire pages at once and 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. Table 2.1 reports the typical and maximum latencies and power usage for each of these operations. The time and power to transfer data across the pins and into or out of the internal buffer, for program and read operations respectively, are not included in these numbers.

In addition to these primary commands, flash devices also support a variety of other operations, such as copyback-read and copyback-program [9]. These commands increase performance by avoiding unnecessary operations or by skipping bus transfers between the controller and the chip.

2.1.4 Capacity

Current SLC and MLC technology allows for 32 Gb/die and 128 Gb/die, respectively [8]. Manufacturers stack between one and eight chips (or dies) in a single package, allowing for between 4GB (one die SLC) and 128 GB (eight die MLC) per package. While increased density is desirable, it comes at a cost: All chips in a single package contend for a single bus (8 or 16 bits wide, depending on the device) of I/O pins. Additional costs of increased density are detailed in Chapter 3.

2.1.5 Performance

Flash offers significant performance gains over conventional hard drives. Currently, 8-bit flash devices can transfer data on and off chip at 133MB/s and 400MB/s chips are projected [10]. This peak bandwidth is higher than the maximum achievable sustained bandwidth because reading and programming latencies are large (see table 2.1). A single 133MB/s SLC flash chip with a single plane can achieve a maximum of 100MB/s for reads and 17MB/s for writes. Erase bandwidth is not affected by bus speed since it does not require any data transfer. As a result, erase bandwidth (166MB/s for a single plane) far exceeds program bandwidth. Increasing the number of planes per die and/or dies per package can increase performance significantly by increasing bus utilization.

2.1.6 Power

Read, program, and erase operations all require different amounts of power. Datasheets give a maximum current draw of between 20mA and 50mA at 2.7-3.3V for a peak power of 50-165 mW.

2.1.7 Reliability

Flash memories can fail in several ways. Most notoriously, devices wear out with use. After many repetitions, the erase and program process can cause cells to become stuck due to physical damage to the device and charge trapping in the gate oxide. The expected lifetime or *erase budget* of one block in a flash device is 500 for TLC, 30,000 for MLC and 100,000 for SLC. Furthermore, some devices retain data for only one year when programmed at this lifetime.

The limitations of MLC and TLC's reliability and performance arise from their underlying structures. Each flash cell comprises a single transistor with an added layer of metal between the gate and the channel, called the floating gate. To change the value stored in the cell, the program operation applies very high voltages to its terminals which cause electrons to tunnel through the gate oxide to reach the floating gate. To erase a cell, the voltages are reversed, pulling the electrons off the floating gate. Each of these operations strains the gate oxide, until eventually it no longer isolates the floating gate, making it impossible to store charge.

The charge on the floating gate modifies the threshold voltage, V_{TH} of the transistor (i.e., the voltage at which the transistor turns on and off). In a programmed SLC cell, V_{TH} will be in one of two ranges (since programming is not perfectly precise), depending on the value the cell stores. The two ranges have a "guard band" between them. Because the SLC cell only needs two ranges and a single guard band, both ranges and the guard band can be relatively wide. Increasing the number of bits stored from one (SLC) to two (MLC) increases the number of distributions from two to four, and requires two additional guard bands. As a result, the distributions must be tighter and narrower. The necessity of narrow V_{TH} distributions increases programming time, since the chip must make more, finer adjustments to V_{TH} to program the cell correctly (as described below). At the same time, the narrow guard band reduces reliability. TLC cells make this problem even worse: They must accommodate eight V_{TH} levels and seven guard bands.

Flash devices report erase and program failures due to wear out, and manufactures recommend that the entire block be removed from service if an error occurs that cannot be corrected with ECC. To maximize the lifetime of a flash chip, flash systems use wear-leveling [20, 41, 80] to ensure that blocks are erased with equal frequency. Section 4.1 describes these options in more detail.

Bits can also become corrupt while stored in flash memory due to "read disturb," "program disturb," and general data "retention" failures. None of these phenomena 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 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 this, flash systems can refresh data after hundreds of thousands of reads to the same location. To correct these and the other types of errors, flash systems must use ECC.

Whether flash wear-out is a problem depends on the application. With good wear-leveling, it is possible to write 25.6 PB to the storage array in one 256 GB node of the system described in Section 5. The peak program bandwidth for a node is 900MB/s, so wear out could, in theory, occur in one year. However, in practice none of our applications sustain program rates near that level and the average across the applications is much lower, leading to much longer lifetimes for the array. Nonetheless, techniques for carefully managing the write budgets of large flash arrays warrant further study.

2.1.8 Software

Flash storage systems typically include a "flash translation layer" (FTL) that manages flash storage. The FTL serves two primary purposes: The first is to provide wear leveling. FTLs maintain a layer of indirection between the logical block addresses (LBA) that the system uses to address data, and the physical location of the data in flash. This allows FTLs to write data wherever they wish, and in particular, to spread writes out across the available flash storage to improve wear leveling. The second purpose is to provide high-performance access to the array by scheduling accesses across the chips to exploit as much parallelism as possible between and within the flash devices. We discuss flash translation layers in more detail in Chapter 5

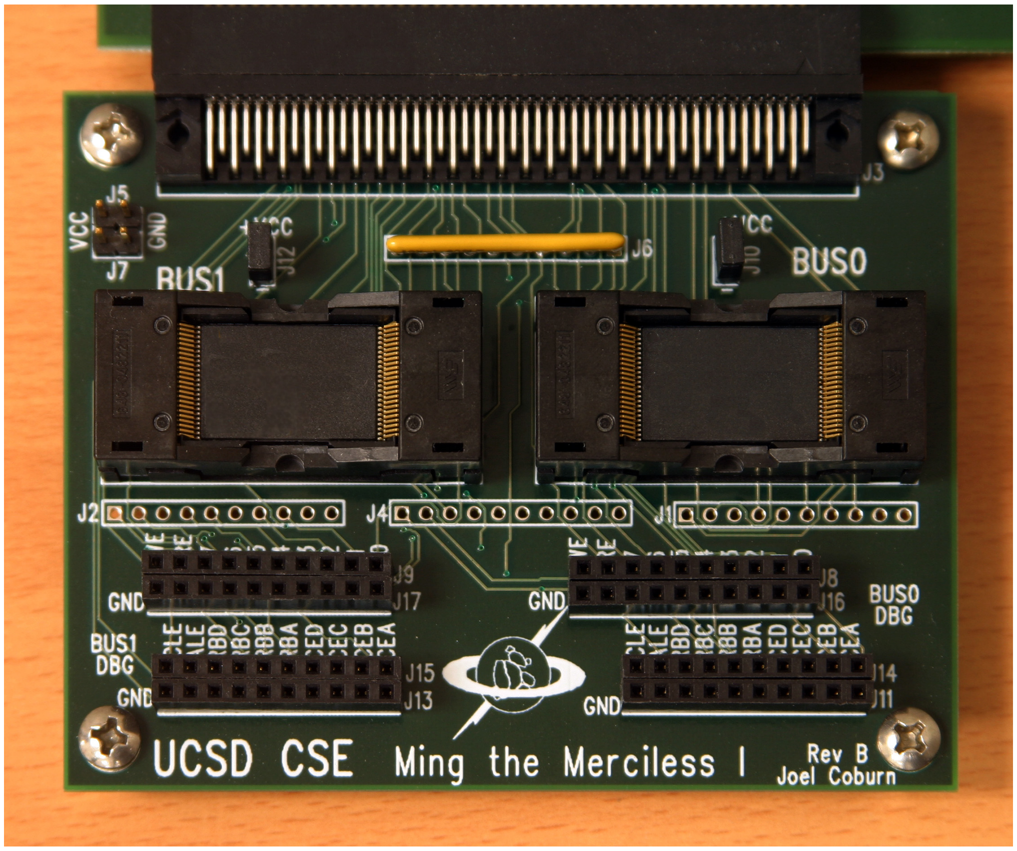


Figure 2.1. Ming the Merciless The flash testing board can test two chips simultaneously and, combined with an FPGA board and a current meter, supports high-resolution timing, power and reliability measurements.

2.2 Characterization Platform

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 latency. This section describes that hardware and the flash devices we used.

2.2.1 Data Collection Hardware

Figure 2.1 shows a photo of our flash characterization board. The custom-built daughter board attaches to a Xilinx XUP Virtex 2 board. The daughter board holds two flash chip test sockets with independent power planes to facilitate 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. A user space application drives the flash testing board and collects the results.

For power measurements we use a high-resolution current probe (Agilent 1147A) attached to a 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.

2.2.2 Flash Devices

Table 2.2 summarizes the flash devices we characterize and study throughout this work. They span 32 models from six manufacturers, one-, two- and three-bit technologies (SLC, MLC and TLC) at process size ranging from 25 nm to 72 nm. Both the existence and our acquisition of this collection of chips evolved over the course of this thesis. Therefore, the earlier studies focus on only a subset of these chips.

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 problems by purchasing flash chips from distributors when possible and removing them from commercially available flash-based USB “thumb drives” and SSDs otherwise. We also built a simple protocol 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.

Table 2.2. Flash Chip Parameters Each name encodes the manufacturer, cell type and die capacity in Gbits. Parameters are drawn from datasheets where available. This is a selection of the 32 models we study. ⋆data unavailable

Chip Name	Max Cycles	Tech Node	Cap. (Gb)	Page Size (B)	Pages /Block	Blocks /Plane	Planes /Die	Dies	Die Cap (Gb)
C-TLC16	⋆	43 nm	16	8192	⋆	8192	⋆	1	16
B-MLC32-4	5,000	34 nm	128	4096	256	2048	2	4	32
B-MLC32-1	5,000	34 nm	32	4096	256	2048	2	1	32
F-MLC16	5,000	41 nm	16	4096	128	2048	2	1	16
A-MLC16	10,000	⋆	16	4096	128	2048	2	1	16
B-MLC16	10,000	50 nm	32	4096	128	2048	2	2	16
C-MLC16	⋆	⋆	32	4096	⋆	⋆	⋆	2	16
D-MLC16	10,000	⋆	32	4096	128	4096	1	2	16
E-MLC16	TBD	⋆	64	4096	128	2048	2	4	16
B-MLC8	10,000	72 nm	8	2048	128	4096	1	1	8
E-MLC4	10,000	⋆	8	4096	128	1024	1	2	4
C-MLC32	10,000	43 nm	64	8192	128	4096	1	2	32
D-MLC16	10,000	50 nm	32	4096	128	4096	1	2	16
E-MLC4	10,000	51 nm	8	4096	128	1024	1	2	4
E-SLC8	100,000	⋆	16	4096	64	2048	2	2	8
A-SLC8	100,000	⋆	8	2048	64	4096	2	1	8
A-SLC4	100,000	⋆	4	2048	64	4096	1	1	4
B-SLC2	100,000	50 nm	2	2048	64	2048	1	1	2
B-SLC4	100,000	72 nm	4	2048	64	2048	2	1	4
E-SLC4	100,000	⋆	8	2048	64	4096	1	2	4
A-SLC2	100,000	⋆	2	2048	64	1024	2	1	2

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.

2.3 Characterization Results

This section presents the results of our chip characterization studies.

2.3.1 Basic Operation Performance

We began by measuring the latency of the basic flash memory operations. Figures 2.2, 2.3 and 2.4 show the latencies for reads, programs, and erases, respectively and grouped by SLC and MLC devices. 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-MLC32), and the measurements are in good agreement with values from publicly available datasheets. MLC chips have, on average, 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 and read times within each MLC block. All of the MLC devices we tested exhibited a regular and predictable variation in 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 for the program operation. 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-MLC32 every other page is fast. Figure 2.5 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.

This effect is well-known to flash chip designers (but often is a surprise to storage

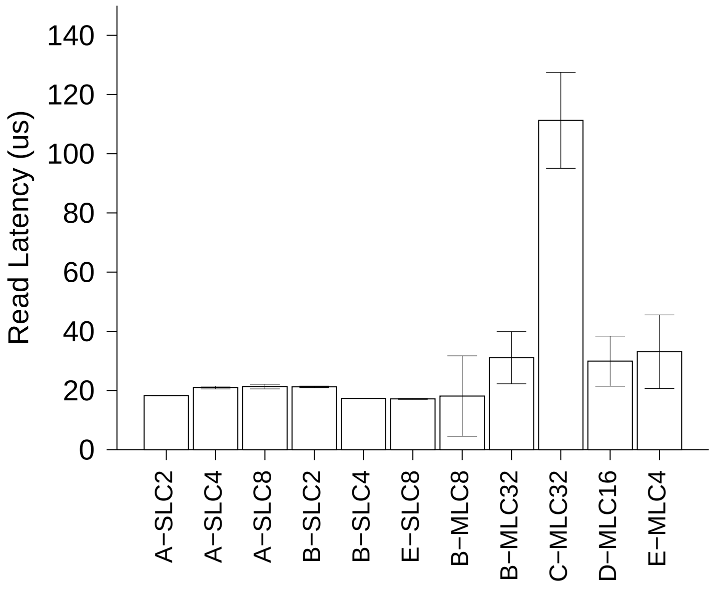


Figure 2.2. Flash Read Latencies Average read latency over ten program/erase cycles for every page in 16 blocks on each chip. Error bars represent one standard deviation.

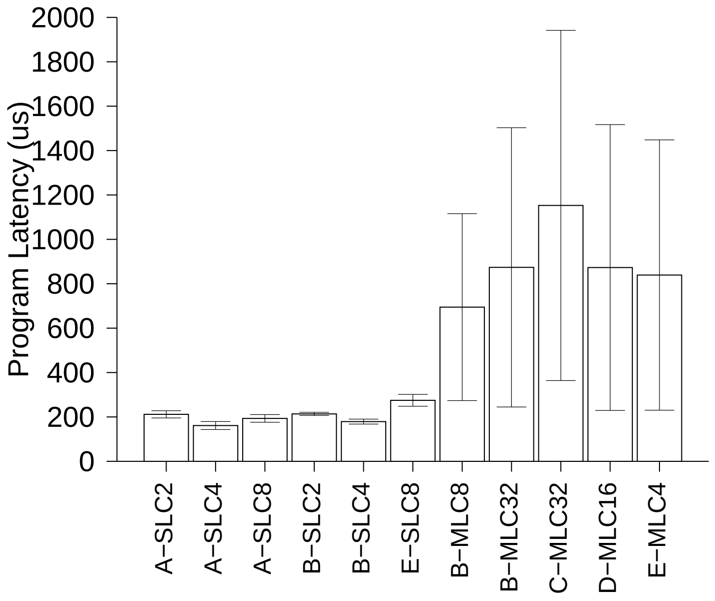


Figure 2.3. Flash Program Latencies Average write (or *program*) latency over ten program/erase cycles for every pages in 16 blocks on each chip. Error bars represent one standard deviation.

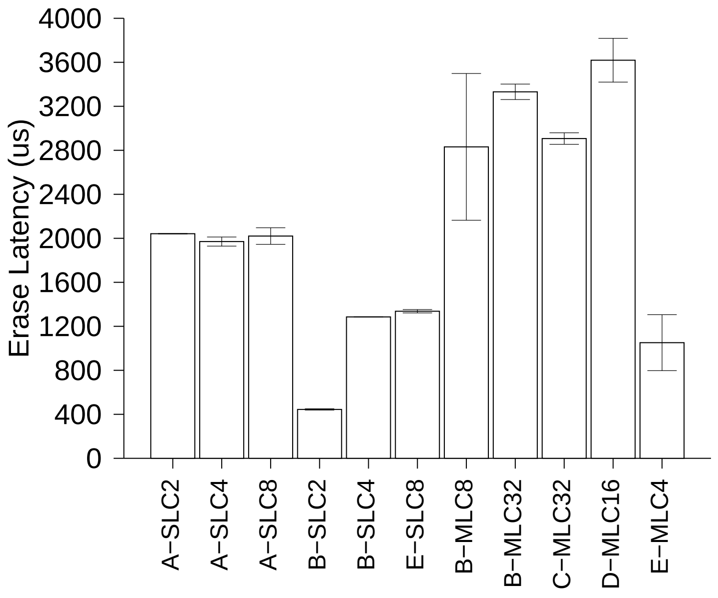


Figure 2.4. Flash Erase Latencies Average erase latency over ten program/erase cycles for 16 blocks on each chip. Error bars represent one standard deviation.

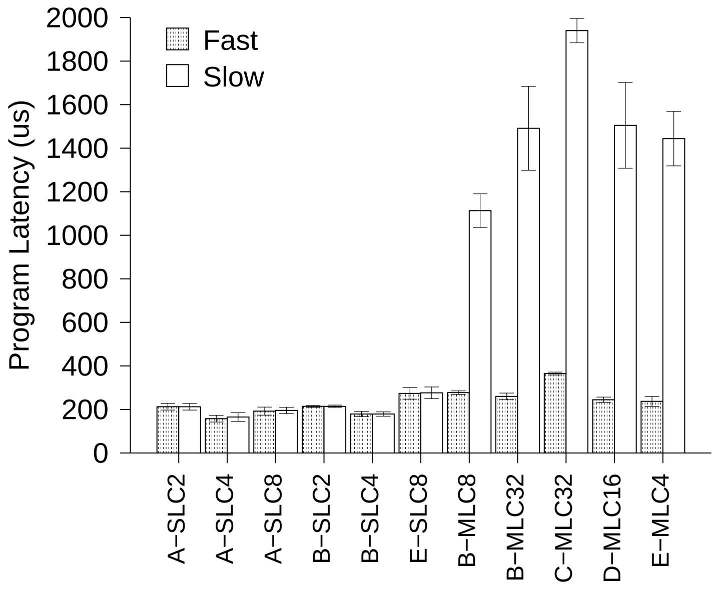


Figure 2.5. The Programing Speed Anomaly Average programming speed over ten program/erase cycles for 16 blocks on each chip. Programming speed varies dramatically between pages in MLC devices in a predictable pattern. SLC chips show no such variation. Error bars represent one standard deviation.

system designers) and arises from the assignment of bits within one MLC cell to separate pages. MLC cells can be in one of four states, depending on how much charge is present on the floating gate, which allows each cell to store two bits. To program either bit, the chip applies a programming pulse, reads the value, applies another pulse, and so on until the cell reaches the desired state (called *Incremental Step Pulse Programming (ISPP)* [71]). Since the high-order bits have a larger target range, the pulses can be relatively powerful and only a few are required. For the low-order bit the pulses are weaker and the program requires more of them, resulting in longer latencies.

Chapter 7 demonstrates how exploiting this difference can significantly improve flash drive responsiveness and efficiency. SLC chips show no corresponding variability.

Although this fast/slow page property was little-known among SSD designers at the time this work was published, papers detailing the design of MLC flash chips explain the development of the phenomenon. Very early flash memory would apply a steady, high voltage to any cell being programed for a fixed amount of time. However, Suh et al. [71] quickly determined that the Incremental Step Pulse Programming (ISPP) would be far more effective in tolerating variation between cells and in environmental conditions. ISPP performs a series of program pulses each followed by a read-verify step. Once the cell is programmed correctly, programming for that cell stops. This algorithm is necessary because programming is a one-way operation: There is no way to "unprogram" a cell short of erasing the entire block, and overshooting the correct voltage results in storing the wrong value. ISPP remains a key algorithm in modern chips and is instrumental in improving the performance and reliability of higher-density cells.

Not long after Samsung proposed MLC for NAND flash [44, 45], Toshiba split the two bits to separate pages so that the chip could program each page more quickly by moving the cell only halfway through the voltage range with each operation [72]. Much later, Samsung provided further performance improvements to pages stored in the least significant bit of each cell [60]. By applying fast, imprecise pulses to program the fast pages, and using fine-grain, precise pulses to program the slow pages. These latter pulses generate the tight V_{TH} distributions that MLC devices require, but they make programming much slower. All the MLC and TLC devices we tested split and program the bits in a cell this way.

The second surprise in our investigation is that performance varies predictably as the devices begin to wear out. Figure 2.6 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%. This effect is due to charge trapping or breakdown of the gate oxide. As the oxide becomes a less effective insulator, it easier to move charge across it onto the gate. The same phenomenon is responsible for the eventual failure of the cell, since it also makes it easier for charge to leak off the floating gating. We discuss a potential application of this phenomenon in Chapter 7.

Figure 2.7 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. A similar argument can be made for increased parallelism through multi-plane operations and multiple chips operating in parallel within an SSD.

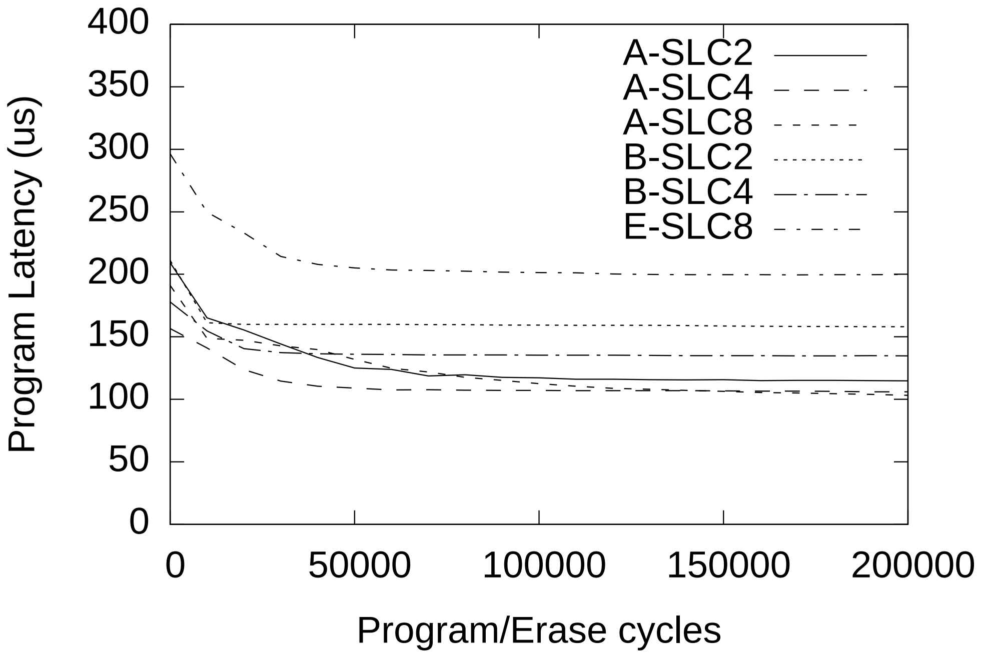


Figure 2.6. Program Performance Over Time For SLC devices, average program time drops by nearly 50% over the 100,000 program/erase cycles of the chip. MLC devices (not shown) show a much smaller, 10-15% decline.

2.3.2 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 accurate 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 2.3 summarizes the results.

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

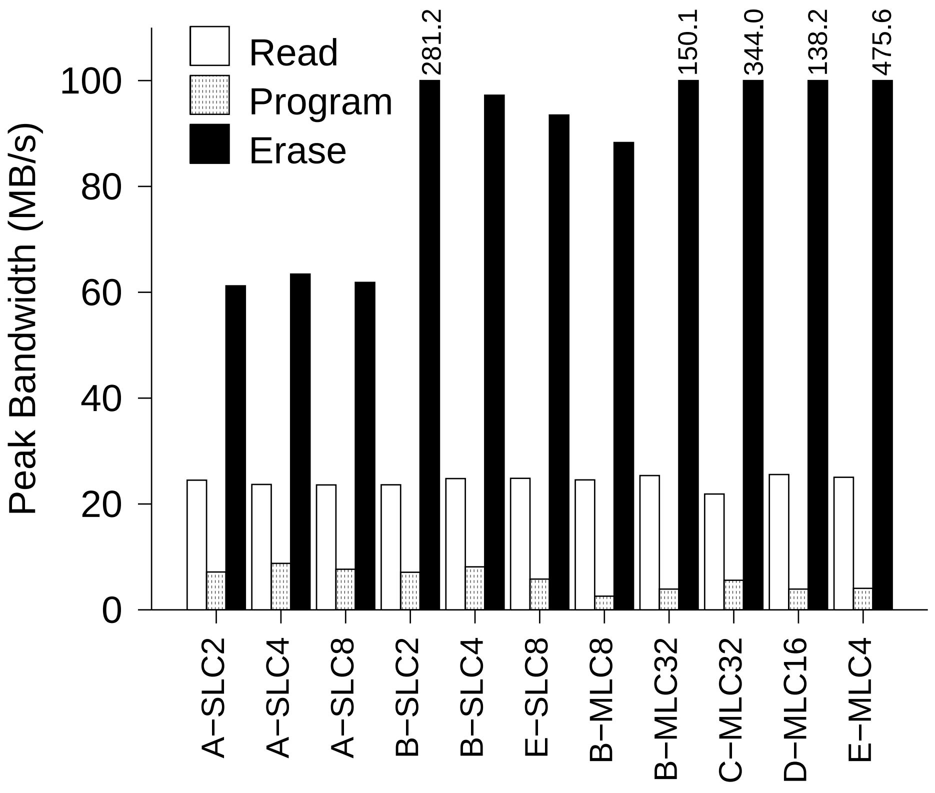


Figure 2.7. Peak Device Bandwidth The peak bandwidth that each device can deliver during single-plane operation.

Table 2.3. Power and Energy Consumption for Flash Operations Peak values are taken from measuring consumption during a single operation with our high-resolution amp meter. The power measurements for operations and the associated data transfer are listed separately. Average values are taken over multiple operations on our test system.

	A-SLC2	A-SLC4	A-SLC8	B-SLC2	B-SLC4	E-SLC8
Peak Read Power in mW (transfer)	35.3 (19.2)	41.1 (18.3)	58.8 (33.1)	27.2 (9.3)	29.9 (8.2)	19.1 (60.8)
Peak Erase Power in mW	30.9	35.5	47.6	25.3	20.0	25.5
Peak Program Power in mW (transfer)	55.2 (43.2)	59.9 (39.2)	78.4 (59.9)	35.0 (13.6)	35.0 (8.4)	56.0 (33.5)
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)	0.052	0.069	0.088	0.046	0.042	0.0056
Program Energy (nJ/bit)	0.72	0.61	0.97	0.47	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-MLC32	D-MLC16	E-MLC4	
Peak Read Power in mW (transfer)	54.0 (29.1)	75.9 (41.1)	112.0 (42.8)	66.3 (31.2)	13.4 (39.9)	
Peak Erase Power in mW	42.4	70.6	111.8	57.0	21.3	
Peak Program Power in mW (transfer)	58.9 (22.4)	94.7 (63.1)	132.2 (65.2)	82.3 (31.7)	118.4 (28.5)	
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	0.093	0.002	
Fast Program Energy (nJ/bit)	1.09	0.96	0.66	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	

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-MLC4, whose remarkably small erase latency provides a correspondingly small erase energy. Excluding the erase energy of E-MLC4, 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.

These measurements provided the means to validate an initial model called FlashPower [58], and later measurements were used to further develop this model [57].

2.3.3 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 fully reversible (see more in the next paragraph). 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, for MLC it is typically 5,000 and for TLC it is typically 500. Recent work promises to extend these lifetimes by 50% by embedding per-memory cell heating elements, much like those used in Phase Change Memory, to partially heal the wear caused by use [29]. This solution comes at the cost of density.

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 random data to each page, and then read the data to check for errors. Then, we performed 990 erase-program operations, of all zeros. We repeated this process until we had reached 1 million erases for SLC chips and 100,000 erases for MLC chips.

Figures 2.8 and 2.9 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 – a shock that prompted us to verify the correct operation of our hardware. 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.

Wear out also has an adverse effect on data "shelf life." Although we have not quantified this effect, flash designers tell us that for recent flash devices the data shelf life can be as low as one year for the blocks that are close to their rated lifetime. Regardless, our data show that for such applications with less stringent shelf life requirements, 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. Figures 2.10, 2.11, 2.12 and 2.13 plot 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.

In practice, the manufacturer-quoted wear-out rates appear to be conservative (as

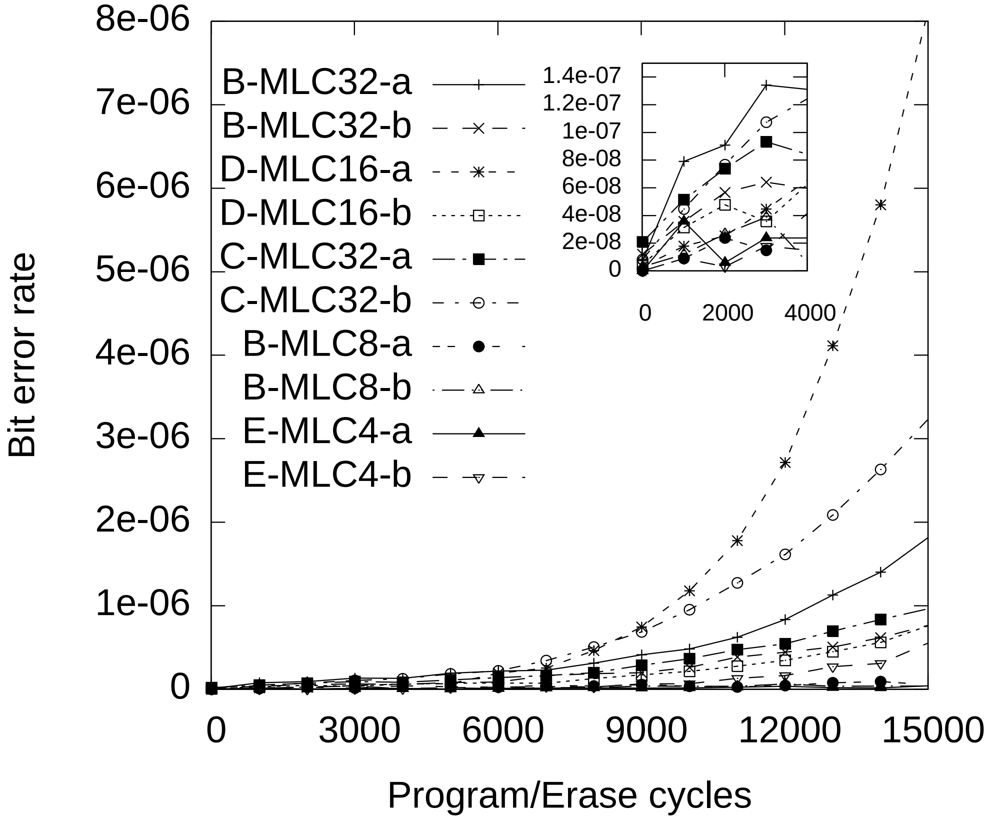


Figure 2.8. Raw Error Rate for MLC Devices The error rate for MLC devices increases dramatically with wear, and is non-zero even for brand-new devices (inset). There is also large variation in the change in error rate between instances of the same chip (notably D-MLC16-a and D-MLC16-b).

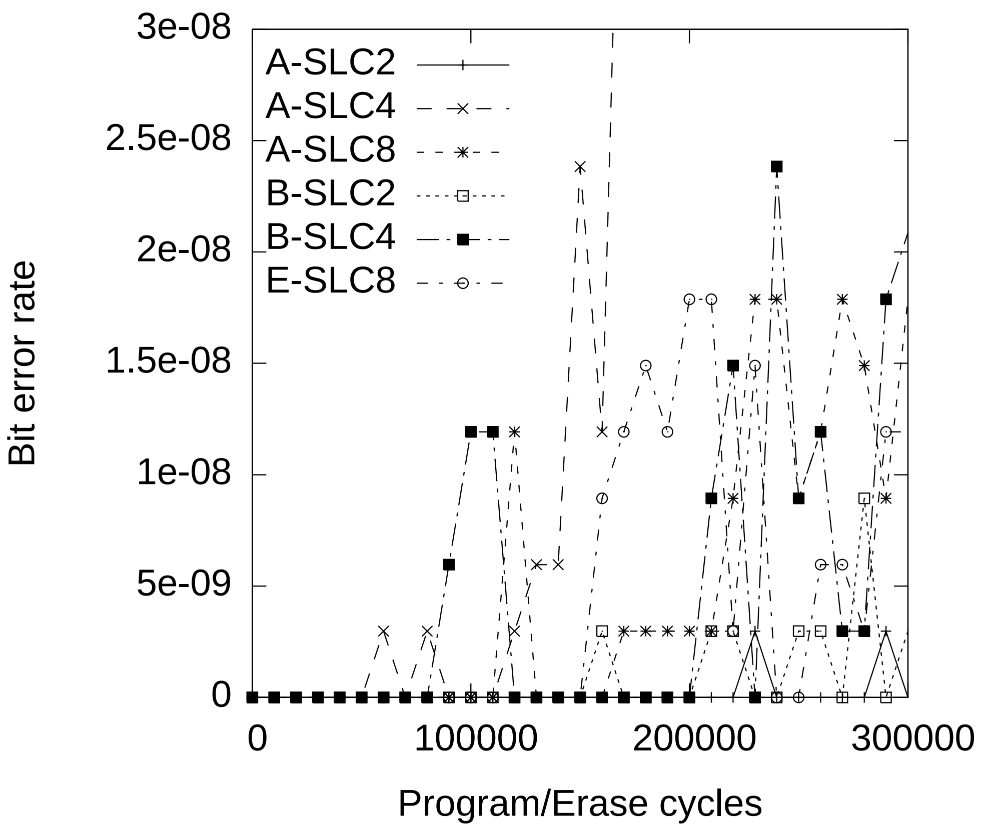


Figure 2.9. Raw Error Rate for SLC Devices For SLC devices, the error rates are several orders of magnitude lower than for MLC even at three times their rated lifetimes. Only A-SLC4 shows a significant increase.

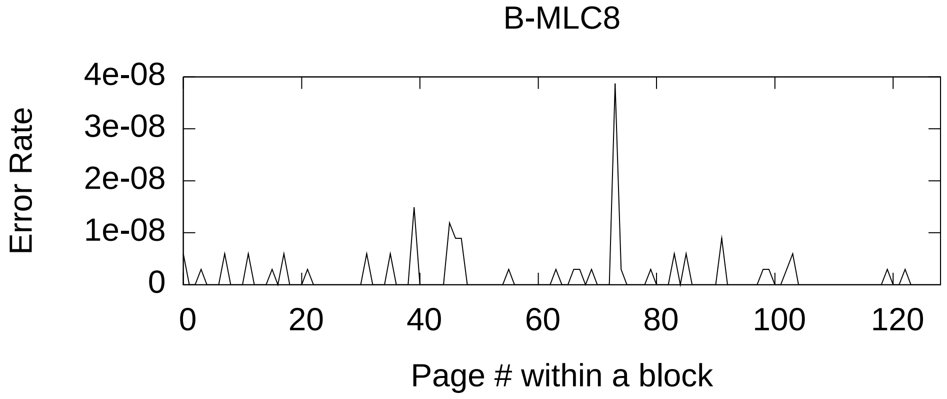


Figure 2.10. Per-Page BER for B-MLC8 This chip exhibits relatively low error rates, but certain pages within the same block appear more error prone than others. If only some pages in a block were decommissioned as error rates grow, the SSD could retain more capacity.

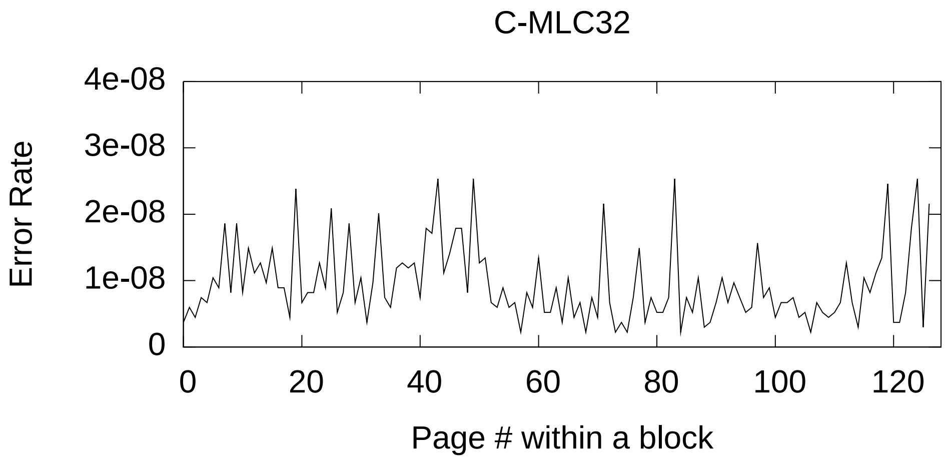


Figure 2.11. Per-Page BER for C-MLC32 MLC chips show large variation in error rates among pages in a single block.

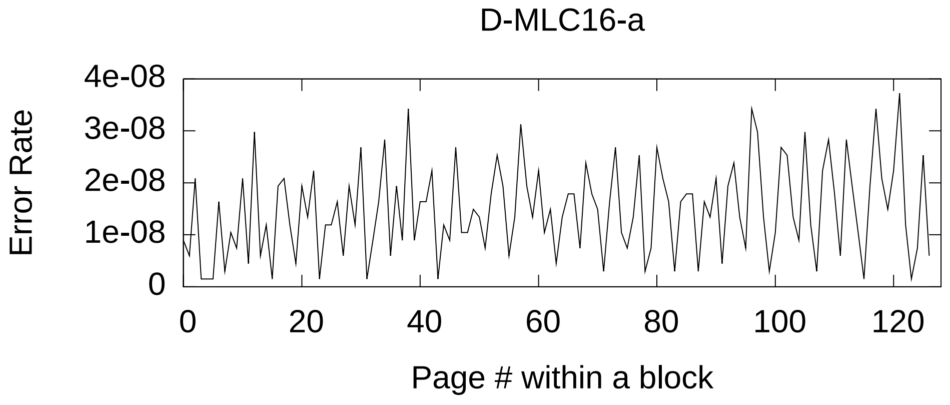


Figure 2.12. Per-Page BER for D-MLC16-a Though this part is the same model as that in Figure 2.13, the error pattern and magnitude is unique.

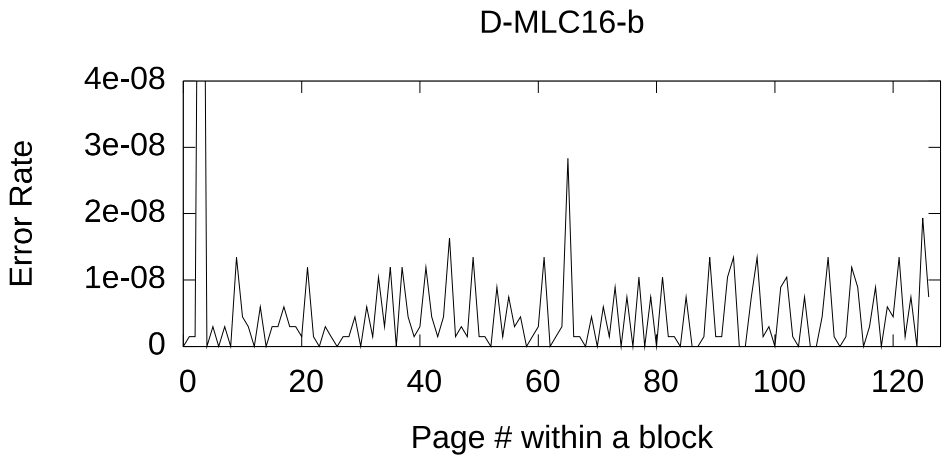


Figure 2.13. Per-page BER for D-MLC16-a This chip and that in Figure 2.12 are supposedly identical parts, but show very different error rates.

expected). We have stress-tested SLC flash chips to measure their failure rate directly. Our data show that 99.9% of blocks will survive to 100,000 erase cycles.

For our MLC devices (two densities from one manufacturer), the average lifetime for a single block is between 27,500 and 116,000, depending on the part. For SLC devices, the average block life span is 868,000 cycles. Other problems potentially emerge at high write counts as well. In particular, data degrades (i.e., its “shelf life” goes down) more quickly as chips age. Depending on the application, this may or may not matter.

Bits can also become corrupt while stored in flash memory due to “read disturb” and “program disturb”. Flash systems must use ECC (stored in the out-of-band data section) to correct these errors, and manufactures suggest (SLC) or require (MLC) that pages be programmed in order within a block to minimize them. Neither of these effects cause permanent damage to the cells.

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

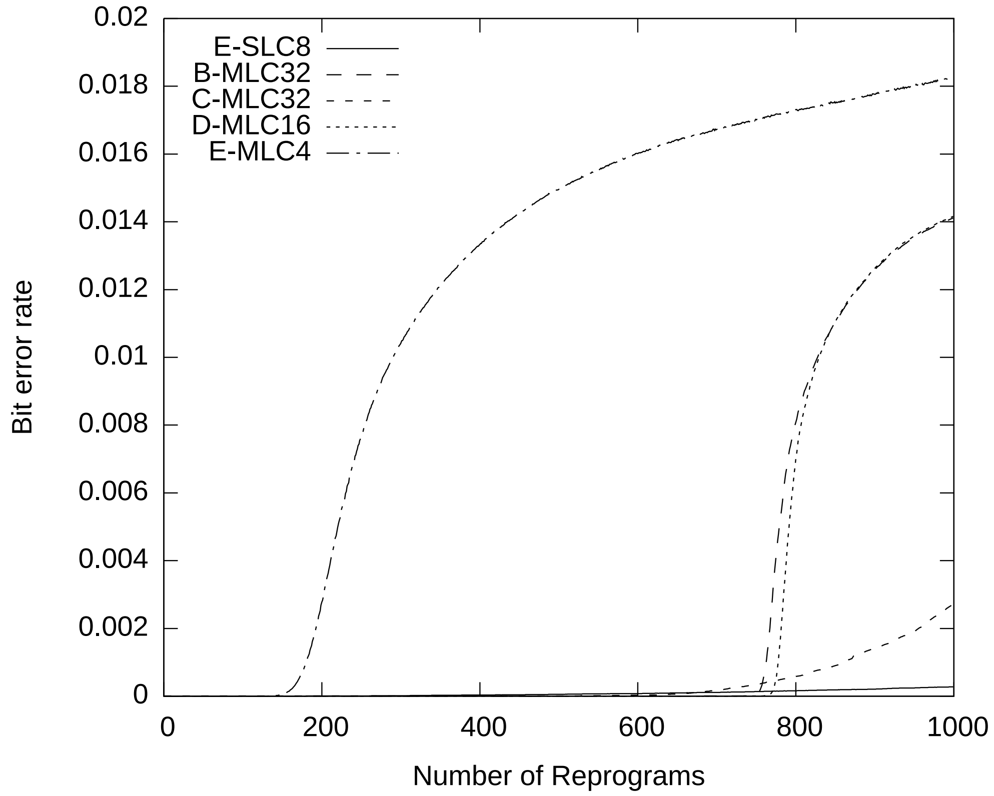


Figure 2.14. Program Disturb Most MLC devices experience a sharp increase in error rates near a certain number of reprograms – a property which we leverage for a new mechanism in Chapter 6.

programmed bits in the block. Figure 2.14 shows how the bit error rate increases with the number of reprograms.

For SLC devices, we observed no program disturb for the first 100 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 2.15 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 of certain pages revealed two distinct patterns of

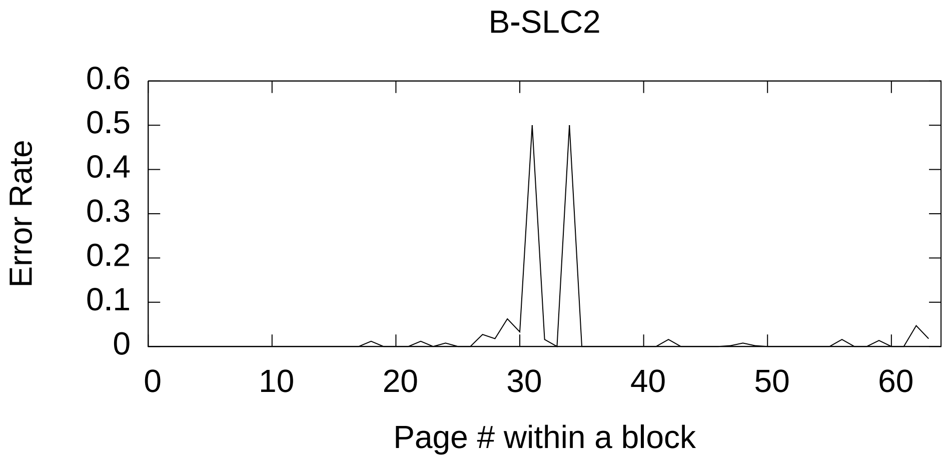


Figure 2.15. Program Disturb for SLC Devices Most SLC devices show no errors even after 10,000 reprograms of page 32. When errors do occur, a variety of patterns emerge. The reprogrammed page consistently shows no errors.

errors. For C-MLC32, reprogramming pages $2n$ would disturb nearly all the bits in page $2n-3$, except for the first two pages (which caused no errors on other pages) and the last two pages (which caused errors on pages 123 and 125). 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 for the first four pages (which cause no errors) and the last four pages (which cause errors on pages 118, 119, 122 and 123).

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. Figures 2.16, 2.17, 2.18 and 2.19 graph the results. Our data show that, in most cases, the effects of read disturb began to appear between 250,000 and 4,800,000 repeated reads.

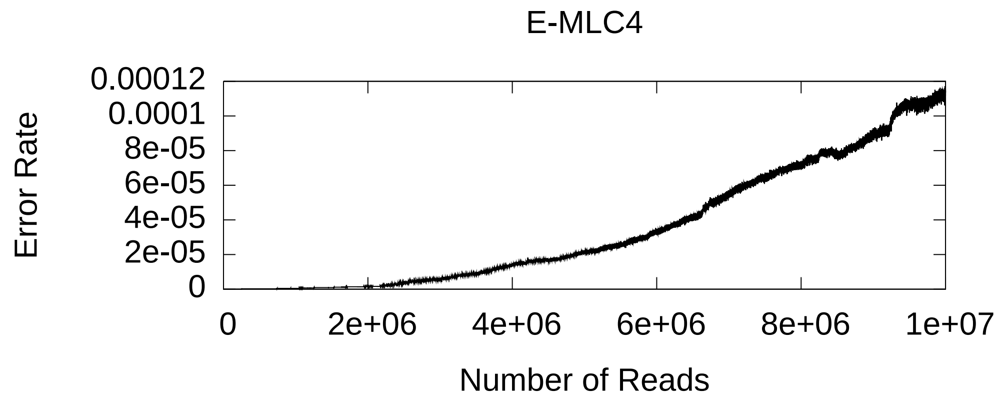


Figure 2.16. Read Disturb, Part E-MLC4 Reading a page 10 Million times without refreshing the data causes steadily increasing error rates.

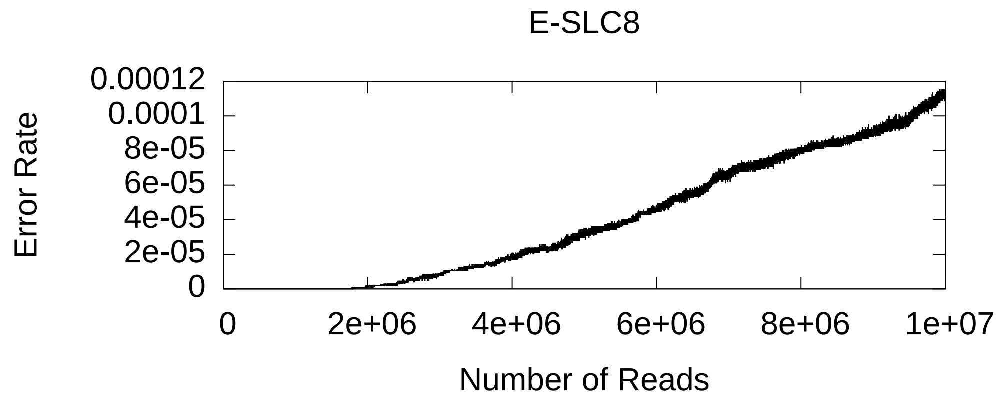


Figure 2.17. Read Disturb, Part E-SLC8 Read disturb errors occur relatively early (after 250 thousand reads) and reach a relatively high rate for both SLC and MLC parts from manufacturer E.

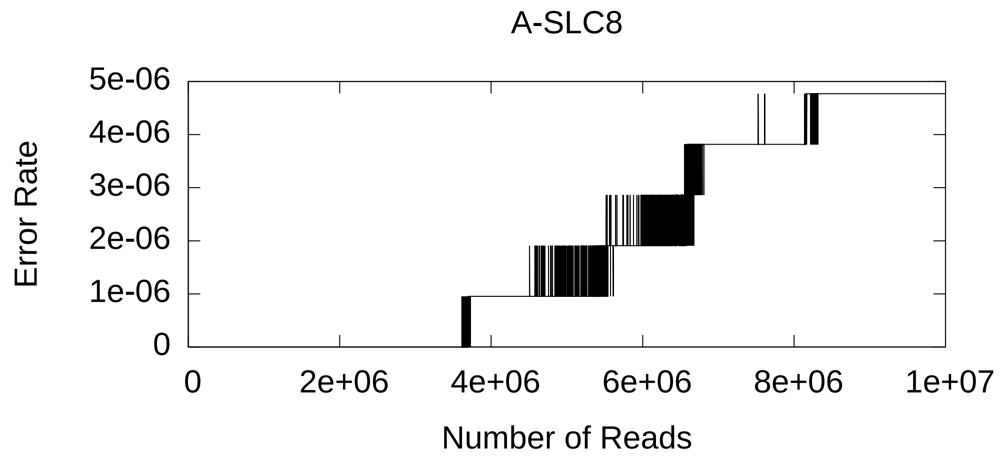


Figure 2.18. Read Disturb, Part A-SLC8 Distinct jumps in the graph show that the number of errors in this part can be counted on one hand.

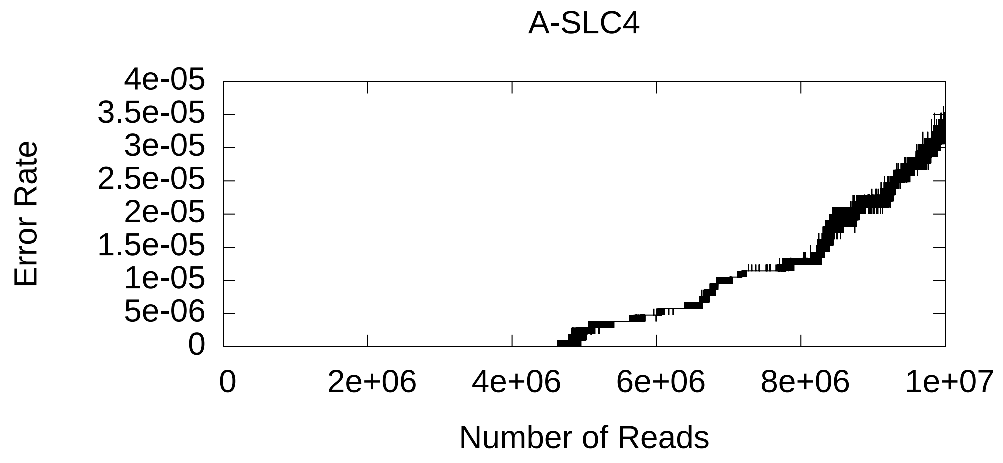


Figure 2.19. Read Disturb, Part A-SLC4 This low-density part shows read disturb errors later than the higher-density part from the same manufacturer, but the error rate increases much more rapidly.

2.4 Conclusion

The devices we characterized in this chapter exhibited variation both within a block and over time in terms of power consumption, latency, and error rates. Our data also show that the values manufacturers provide in publicly available datasheets often tell only part of the story, and that actual performance can be significantly worse and highly variable. By looking beyond the datasheets manufacturers provide, we make significant improvements to flash-based storage devices (described later in this thesis), and enable further research in these directions.

Acknowlegements

This chapter contains material from “Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications” by Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson which appears in *ASPLOS’09: The 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. The dissertation author was the second investigator and author of this paper. This material is copyright ©2009 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1(212)869-0481, or permissions@acm.org.

This chapter contains material from “Characterizing Flash Memory: Anomalies,

Observations, and Applications" by Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Eitan Yaakobi, Steven Swanson, Paul H. Siegel, and Jack K. Wolf, which appears in *Micro'09: the 42nd Annual International Symposium on Microarchitecture*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2009 by the Institute of Electrical and Electronics Engineers (IEEE). If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Chapter 3

Flash Trends

In recent years, flash-based SSDs have grown enormously both in capacity and popularity. In high performance enterprise storage applications, accelerating adoption of SSDs is predicated on the ability of manufacturers to deliver performance that far exceeds disks while closing the gap in cost per gigabyte. However, while flash density continues to improve, other metrics such as reliability, endurance and performance are all declining. As a result, building larger-capacity flash-based SSDs that are reliable enough to be useful in enterprise settings and high-performance enough to justify their cost will become challenging.

In this chapter, we present our empirical data collected from the collection of flash chips described in Chapter 2 – spanning SLC, MLC and TLC densities as well as technology nodes from 72 nm to 25 nm – and examine the performance trends for these raw flash devices as flash scales down. We use this analysis to predict the performance and cost characteristics of future SSDs. We show that future gains in density will come at significant drops in performance and reliability. As a result, SSD manufacturers and users will face a tough choice in trading off between cost, performance, capacity and reliability.

We begin the chapter by describing the current state of NAND Flash memory (Section 3.1), focusing the details presented in Chapter 2. We then describe the details of our baseline SSD design and empirical trends in Section 3.2 before presenting our predictions for the scaling of SSDs in Section 3.3.

3.1 The State of NAND Flash Memory

Flash-based SSDs are evolving rapidly and in complex ways – while manufacturers drive toward higher densities to compete with HDDs, increasing density by using newer, cutting edge flash chips can adversely affect performance, energy efficiency and reliability.

To enable higher densities, manufacturers scale down the manufacturing feature size of these chips while also leveraging the technology's ability to store multiple bits in each cell. Most recently on the market are 25 nm cells which can store three bits each (called Triple Level Cells, or *TLC*). Before TLC came 2-bit, multi-level cells (MLC) and 1-bit single-level cells (SLC). Techniques that enable four or more bits per cell are on the horizon [73], and Macronix has demonstrated 6-bits per cell at small scale [37].

Figure 3.1, collects the trend in price of raw flash memory from a variety of industrial sources, and shows the drop in price per bit for the higher density chips. Historically, flash cost per bit has dropped by between 40 and 50% per year [27]. However, over the course of 2011, the price of flash flattened out. If flash has trouble scaling beyond 12nm (as some predict), the prospects for further cost reductions are uncertain.

We present empirical evidence of worsening lifetime and reliability of flash as it reaches higher densities. We collected this data from 45 flash chips made by six manufacturers spanning feature sizes from 72 nm to 25 nm. Our flash characterization system (described in Chapter 2) allows us to issue requests to a raw flash chip without FTL interference and measure the latency of each of these operations with 10 ns resolution. We repeat this program-erase cycle (P/E cycle) until each measured block reaches the rated lifetime of its chip.

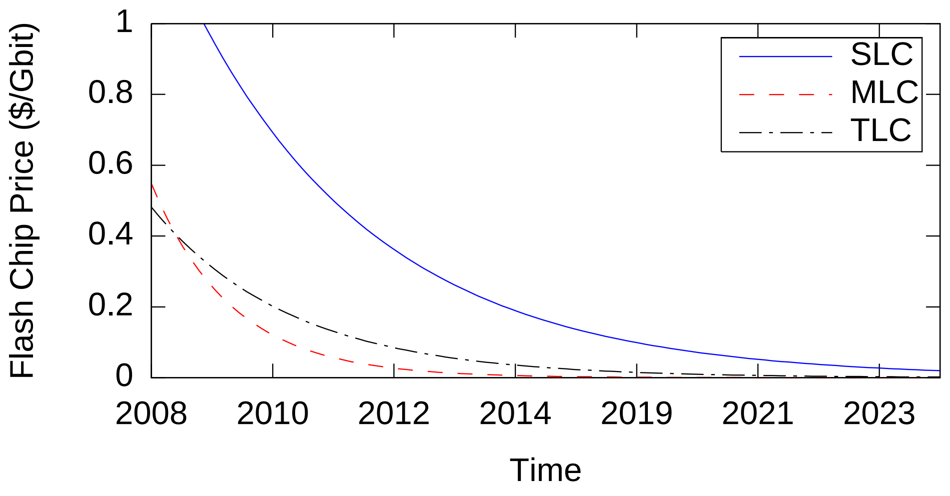


Figure 3.1. Trends in Flash Prices Flash prices reflect the target markets. Low density, SLC, parts target higher-priced markets which require more reliability while high density MLC and TLC are racing to compete with low-cost HDDs. Cameras, iPods and other mobile devices drive the low end.

Figure 3.2 shows the chips' rated lifetime as well as the raw bit error rate (RBER) measured at that lifetime. The chips' lifetimes decrease slowly with feature size, but fall precipitously across SLC, MLC and TLC devices. While the error rates span a broad range, there is a clear upward trend as feature size shrinks and densities increase. Applications that require more reliable or longer-term storage prefer SLC chips and those at larger feature sizes because they experience far fewer errors for many more cycles than denser technology.

For SSD designers, this performance variability between pages leads to an opportunity to easily trade off capacity and performance [35, 68]. The SSD can, for example use only the fast pages in MLC parts, sacrificing half their capacity but making latency comparable to SLC. In this work, we label such a configuration "MLC-1" – an MLC device using just one bit per cell. Samsung and Micron have formalized this trade-off in multi-level flash by providing single and multi-level cell modes [55] in the same chip and we believe FusionIO uses the property in the controller of their SMLC-based drives [68].

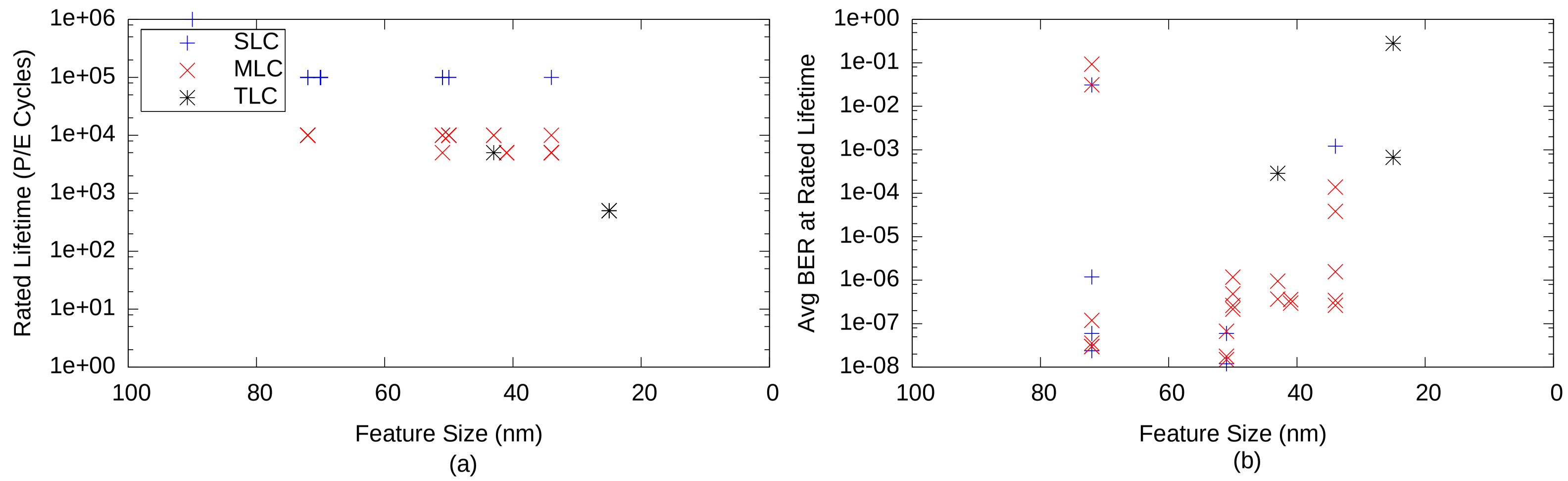


Figure 3.2. Trends in Flash's Reliability Increasing flash's density by adding bits to a cell or by decreasing feature size reduces both (a) lifetime and (b) reliability.

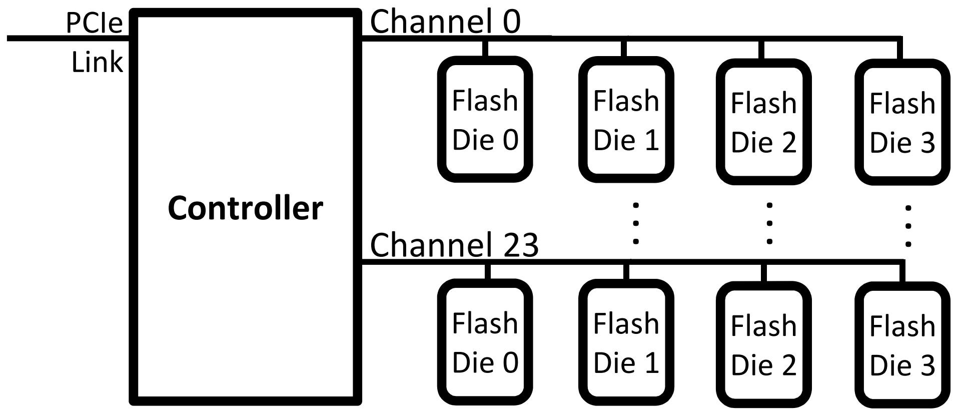


Figure 3.3. Architecture of SSD-CDC The architecture of our baseline SSD. This structure remains constant while we scale the technology used for each flash die.

3.2 A Prototypical SSD

To model the effect of evolving flash characteristics on complete SSDs we combine empirical measurement of flash chips in an SSD architecture with a constant die count called *SSD-CDC*. SSD-CDC's architecture is representative of high-end SSDs from companies such as FusionIO, OCZ and Virident. We model the complexities of FTL design by assuming optimistic constants and overheads that provide upper bounds on the performance characteristics of SSDs built with future generation flash technology.

Section 3.2.1 describes the architecture of SSD-CDC, while Section 3.2.2 describes how we combine this model with our empirical data to estimate the performance of an SSD with fixed die area.

Table 3.1. Architecture and Baseline Configuration of SSD-CDC These parameters define the Enterprise-class, Constant Die Count SSD (SSD-CDC) architecture and starting values for the flash technology it contains.

Architecture Parameter	**Value**
Example Interface	PCIe 1.1x4
FTL Overhead Latency	30 μs
Channels	24
Channel Speed	400 MB/s [3]
Dies per Channel (DPC)	4
Baseline Parameter	**Value**
SSD Price	$7,800
Capacity	320 GB
Feature Size	34 nm
Cell Type	MLC

Table 3.2. Latency Projections We generated these equations by fitting an exponential ($y = Ae^{bf}$) to our empirical data, and they allow us to project the latency of flash as a function of feature size (f) in nm. The percentages represent the increase in latency with 1nm shrinkage. †The trends for TLC are less certain than for SLC or MLC, because our data for TLC devices is more limited.

Config-	**Read Latency (μs)**		**Write Latency (μs)**	
uration	**Equation**	**-1nm**	**Equation**	**-1nm**
SLC-1	max $= 24.0e^{-3.5e-3f}$	0.36%	max $= 287.0e^{-1.1e-2f}$	1.07%
	avg $= 23.4e^{-3.2e-3f}$	0.32%	avg $= 262.6e^{-1.2e-2f}$	1.19%
	min $= 22.8e^{-2.9e-3f}$	0.29%	min $= 239.3e^{-1.3e-2f}$	1.34%
MLC-1	max $= 34.8e^{-6.9e-3f}$	0.69%	max $= 467.3e^{-1.0e-2f}$	1.01%
	avg $= 33.5e^{-6.3e-3f}$	0.63%	avg $= 390.0e^{-8.7e-3f}$	0.87%
	min $= 32.2e^{-5.6e-3f}$	0.57%	min $= 316.5e^{-7.0e-3f}$	0.70%
MLC-2	max $= 52.5e^{-4.5e-3f}$	0.45%	max $= 1778.2e^{-8.3e-3f}$	0.84%
	avg $= 43.3e^{-5.2e-3f}$	0.52%	avg $= 1084.4e^{-8.6e-3f}$	0.86%
	min $= 34.2e^{-6.6e-3f}$	0.66%	min $= 393.7e^{-9.9e-3f}$	1.00%
†TLC-3	max $= 102.5e^{-1.3e-3f}$	0.13%	max $= 4844.8e^{-1.1e-2f}$	1.12%
	avg $= 78.2e^{-4.4e-4f}$	0.04%	avg $= 2286.2e^{-7.1e-3f}$	0.71%
	min $= 54.0e^{9.9e-4f}$	-0.10%	min $= 2620.8e^{-4.6e-2f}$	4.67%

Table 3.3. SSD Scaling Model Equations These equations allow us to scale the metrics of our baseline SSD to future process technologies and other cell densities.

Number	Metric	Value
1	$Capacity_{proj}$	$= Capacity_{base} \times \left(\frac{BitsPerCell_{proj}}{BitsPerCell_{base}}\right) \times \left(\frac{FeatureSize_{base}}{FeatureSize_{proj}}\right)^2$
2	SSD_BW_{proj}	$= ChannelCount \times ChannelBW_{proj}$
3	$ChannelBW_{proj}$	$= \frac{(DiesPerChannel-1)*PageSize}{DieLatency_{proj}}$, when $DieLatency_{proj} \leq BWThreshold$
4	$ChannelBW_{proj}$	$= ChannelSpeed$, when $DieLatency_{proj} > BWThreshold$
5	$TransferTime$	$= \frac{PageSize}{ChannelSpeed}$
6	$BWThreshold$	$= (DiesPerChannel - 1) \times TransferTime$
7	SSD_IOPS_{proj}	$= ChannelCount \times ChannelIOPS_{proj}$
9	$ChannelIOPS_{proj}$	$= \frac{1}{TransferTime}$, when $DieLatency_{proj} \leq IOPS\_Threshold$
8	$ChannelIOPS_{proj}$	$= \frac{(DiesPerChannel-1)}{DieLatency_{proj}}$, when $DieLatency_{proj} > IOPS\_Threshold$
10	$TransferTime$	$= \frac{AccessSize}{ChannelSpeed}$
11	$IOPS\_Threshold$	$= (DiesPerChannel - 1) \times TransferTime$

3.2.1 SSD-CDC

Table 3.1 describes the parameters of SSD-CDC's architecture and Figure 3.3 shows a block representation of its architecture. SSD-CDC manages an array of flash chips and presents a block-based interface. Given current trends in PCIe interface performance, we assume that the PCIe link is not a bottleneck for our design.

The SSD's controller implements the FTL. We estimate that this management layer incurs an overhead of 30 μs for ECC and additional FTL operations. The controller coordinates 24 channels, each of which connects four dies to the controller via a 400 MB/s bus. To fix the cost of SSD-CDC, we assume a constant die count equal to 96 dies.

3.2.2 Projections

We now describe our future projections for seven metrics of SSD-CDC: capacity, read latency, write latency, read bandwidth, write bandwidth, read IOPS and write IOPS. Table 3.1 provides baseline values for SSD-CDC and Table 3.2 summarizes the projections we make for the underlying flash technology. This section describes the formulas we use to compute each metric from the projections (summarized in Table 3.3). Some of the calculations involve making simplifying assumptions about SSD-CDC's behavior. In those cases, we make the assumption that maximizes the SSD's performance.

Capacity

Equation 1 calculates the capacity of SSD-CDC, by scaling the capacity of the baseline by the square of the ratio of the projected feature size to the baseline feature size (34 nm). We also scale capacity depending on the number of bits per cell (BPC) the projected chip stores relative to the baseline BPC (2 – MLC). In some cases, we configure SSD-CDC to store fewer bits per cell than a projected chip allows, as in the case of MLC-1. In these cases, the projected capacity would reflect the *effective* bits per cell.

Latency

To calculate the projected read and write latencies, we fit an exponential function to the empirical data for a given cell type. Figure 3.4 depicts both the raw latency data and the curves fitted to SLC-1, MLC-1, MLC-2 and TLC-3. To generate the data for MLC-1, which ignores the "slow" pages, we calculate the average latency for reads and writes for the "fast" pages only. Other configurations supporting reduced capacity and improved latency, such as TLC-1 and TLC-2, would use a similar method. We do not present these latter configurations, because there is very little TLC data available to create reliable predictions. Figure 3.4 shows each collection of data with the fitted exponentials for average, minimum and maximum, and Table 3.2 reports the equations for these fitted trends. We calculate the projected latency by adding the values generated by these trends to the SSD's overhead reported in Table 3.1.

Bandwidth

To find the bandwidth of our SSD, we must first calculate each channel's bandwidth and then multiply that by the number of channels in the SSD (Equation 2). Each channel's bandwidth requires an understanding of whether channel bandwidth or per-chip bandwidth is the bottleneck. Equation 6 determines the threshold between these two cases by multiplying the transfer time (see Equation 5) by one less than the number of dies on the channel. If the latency of the operation on the die is larger than this number, the die is the bottleneck and we use Equation 3. Otherwise, the channel's bandwidth is simply the speed of its bus (Equation 4).

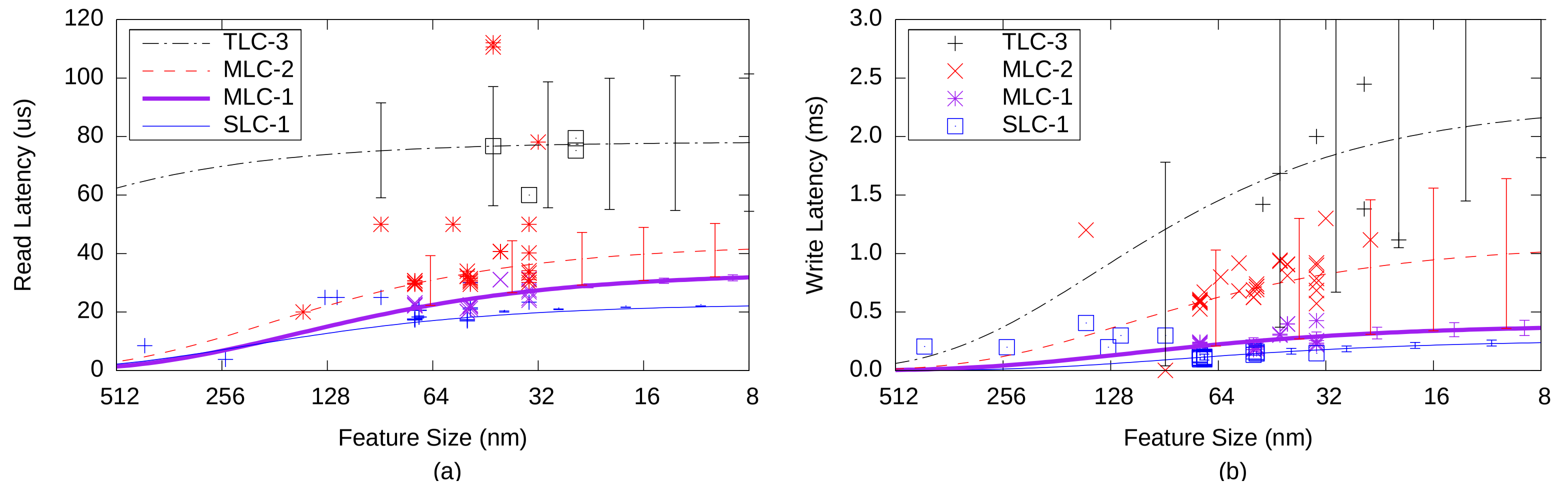


Figure 3.4. Flash Chip Latency Trends Fitting an exponential to the collection of data for each cell technology, SLC-1, MLC-1, MLC-2 and TLC-3, allows us to project the behavior of future feature sizes for (a) read latency and (b) write latency. Doing the same with one standard deviation above and below the average for each chip yields a range of probable behavior, as shown by the error bars.

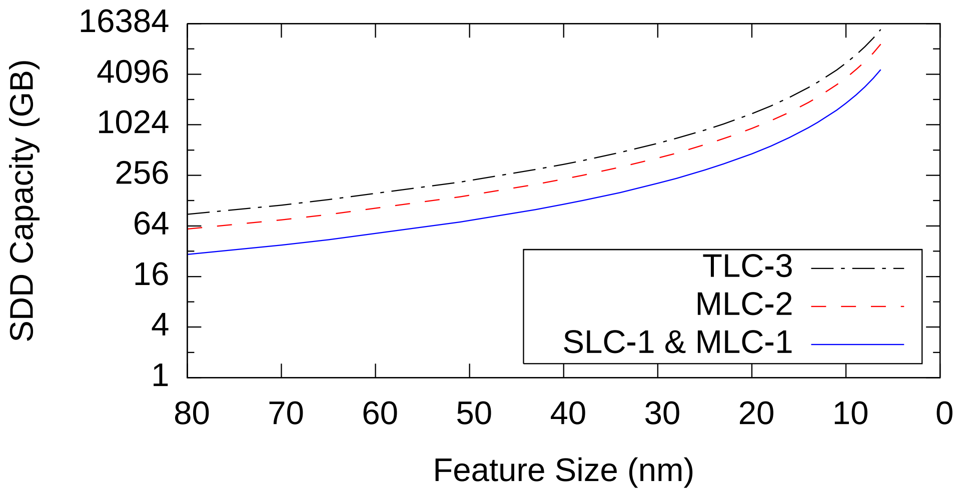


Figure 3.5. Scaling of SSD Capacity Flash manufacturers increase SSDs' capacity through both reducing feature size and storing more bits in each cell.

IOPS

The calculation for IOPS is very similar to bandwidth, except instead of using the flash's page size in all cases, we also account for the access size since it affects the transfer time: If the access size is smaller than one page, the system still incurs the read or write latency of one entire page access. Equations 7-11 describe the calculations.

3.3 Chip Trends Extended to SSDs

This section explores the performance and cost of SSD-CDC in light of the flash feature size scaling trends described above. We explore four different cell technologies (SLC-1, MLC-1, MLC-2, and TLC-3) and feature sizes scaled down from 72 nm to 6.5 nm (the smallest feature size targeted by industry consensus as published in the International Technology Roadmap for Semiconductors (ITRS) [6]), using a fixed silicon budget for flash storage.

3.3.1 Capacity and Cost

Figure 3.5 shows how SSD-CDC's density will increase as the number of bits per cell rises and feature size continues to scale. Even with the optimistic goal of scaling flash cells to 6.5 nm, SSD-CDC can only achieve capacities greater than $\sim$4.6 TB with two or more bits per cell. TLC allows for capacities up to 14 TB – pushing capacity beyond this level will require more dies.

Since capacity is one of the key drivers in SSD design and because it is the only aspect of SSDs that improves consistently over time, we plot the remainder of the characteristics against SSD-CDC's capacity.

3.3.2 Latency

Reduced latency is among the frequently touted advantages of flash-based SSDs over disks, but changes in flash technology will erode the gap between disks and SSDs. Figure 3.6 shows how both read and write latencies increase with SSD-CDC's capacity. Reaching beyond 4.6 TB pushes write latency to 1 ms for MLC-2 and over 2.1 ms for TLC. Read latency rises to least 70 μs for MLC-2 and 100 μs for TLC.

The data also make clear the choices that SSD designers will face. Either SSD-CDC's capacity stops scaling at $\sim$4.6 TB or its read and write latencies increase sharply because increasing drive capacity with fixed die area would necessitate switching cell technology from SLC-1 or MLC-1 to MLC-2 or TLC-3. With current trends, our SSDs could be up to 34x larger, but the latency will be 1.7x worse for reads and 2.6x worse for writes. This will reduce the write latency advantage that SSDs offer relative to disk from 8.3x (vs. a 7 ms disk access) to just 3.2x. Depending on the application, this reduced improvement may not justify the higher cost of SSDs.

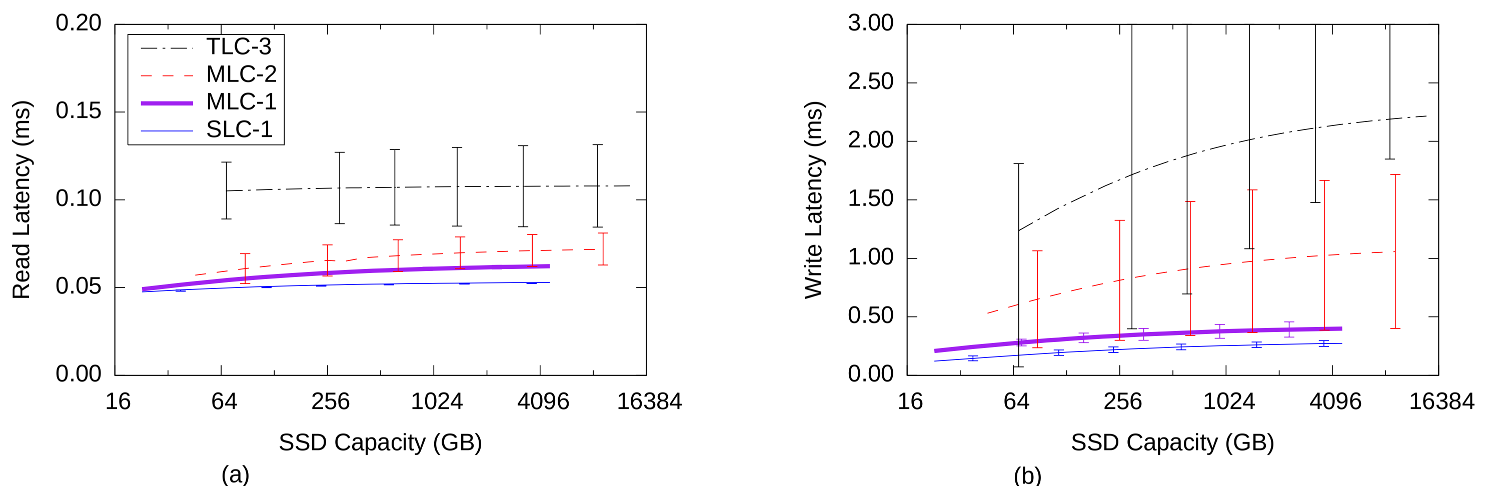
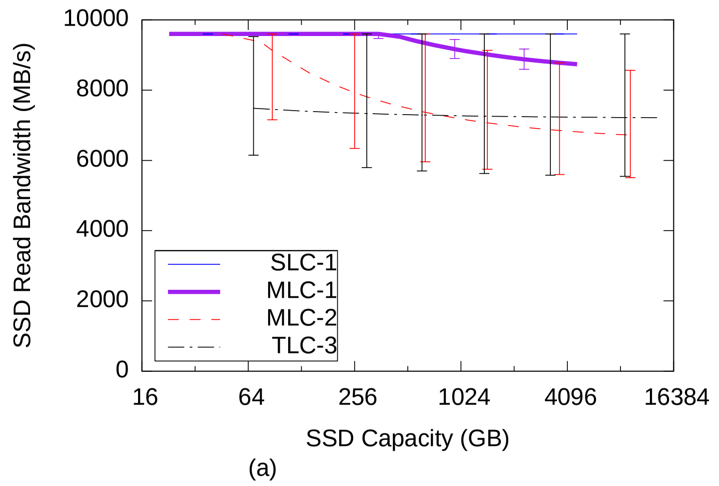


Figure 3.6. SSD Latency In order to achieve higher densities, flash manufacturers must sacrifice (a) read and (b) write latency.



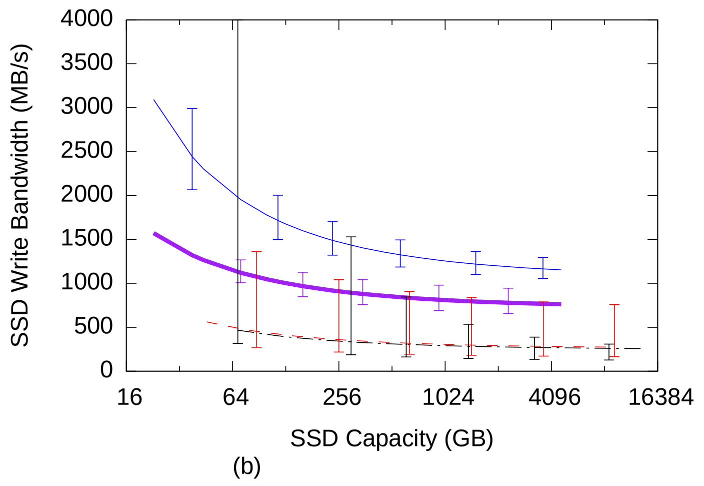


Figure 3.7. SSD Bandwidth SLC will continue to be the high performance option. To obtain higher capacities without additional dies and cost will require a significant performance hit in terms of (a) read and (b) write bandwidth moving from SLC-1 to MLC-2 or TLC-3.

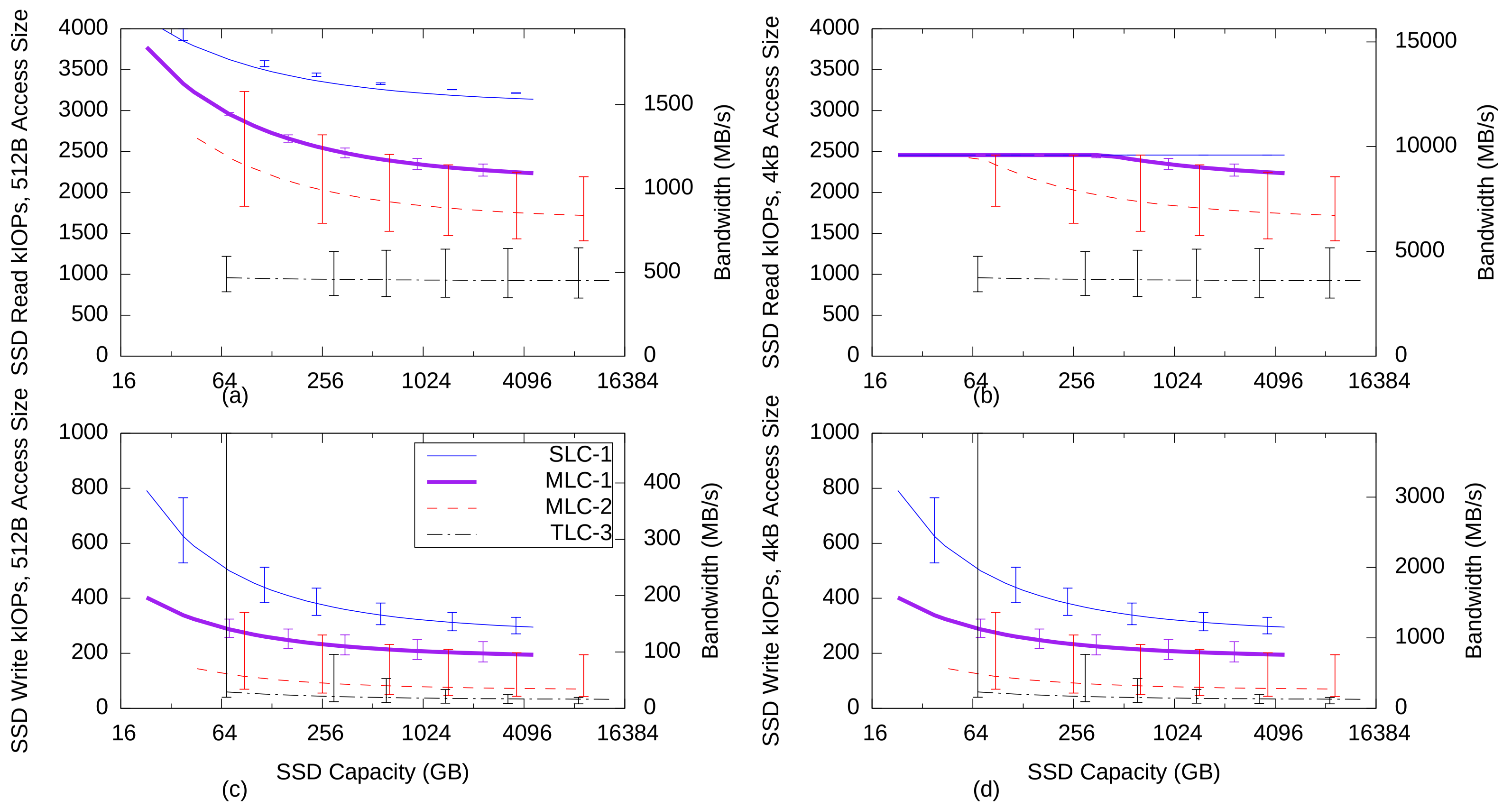


Figure 3.8. SSD IOPS With a fixed die area, higher capacities can only be achieved with low-performing MLC-2 and TLC-3 technologies, for 512B (a) reads and (c) writes and for 4kB (b) reads and (d) writes.

3.3.3 Bandwidth and IOPS

SSDs offer moderate gains in bandwidth relative to disks, but very large improvements in random IOPS performance. However, increases in operation latency will drive down IOPS and bandwidth.

Figure 3.7 illustrates the effect on bandwidth. Read bandwidth drops due to the latency of the operation on the flash die. Operation latency also causes write bandwidth to decrease with capacity.

SSDs provide the largest gains relative to disks for small, random IOPS. We present two access sizes – the historically standard disk block size of 512 B and the most common flash page size and modern disk access size of 4 kB. Figure 3.8 presents the performance in terms of IOPS. When using the smaller, unaligned 512B accesses, SLC and MLC chips must access 4 kB of data and the SSD must discard 88% of the accessed data. For TLC, there is even more wasted bandwidth because page size is 8 kB.

When using 4kB accesses, MLC IOPS drop as density increases, falling by 18% between the 64 and 1024 GB configurations. Despite this drop, the data suggest that SSDs will maintain an enormous (but slowly shrinking) advantage relative to disk in terms of IOPS. Even the fastest hard drives can sustain no more than 200 IOPS, and the slowest SSD configuration we consider achieves over 32,000 IOPS.

Figure 3.9 shows all parameters for an SSD made from MLC-2 flash normalized to SSD-CDC configured with currently available flash. Our projections show that the cost of the flash in SSD-CDC will remain roughly constant and that density will continue to increase (as long as flash scaling continues as projected by the ITRS). However, they also show that access latencies will increase by 26% and that bandwidth (in both MB/s and IOPS) will drop by 21%.

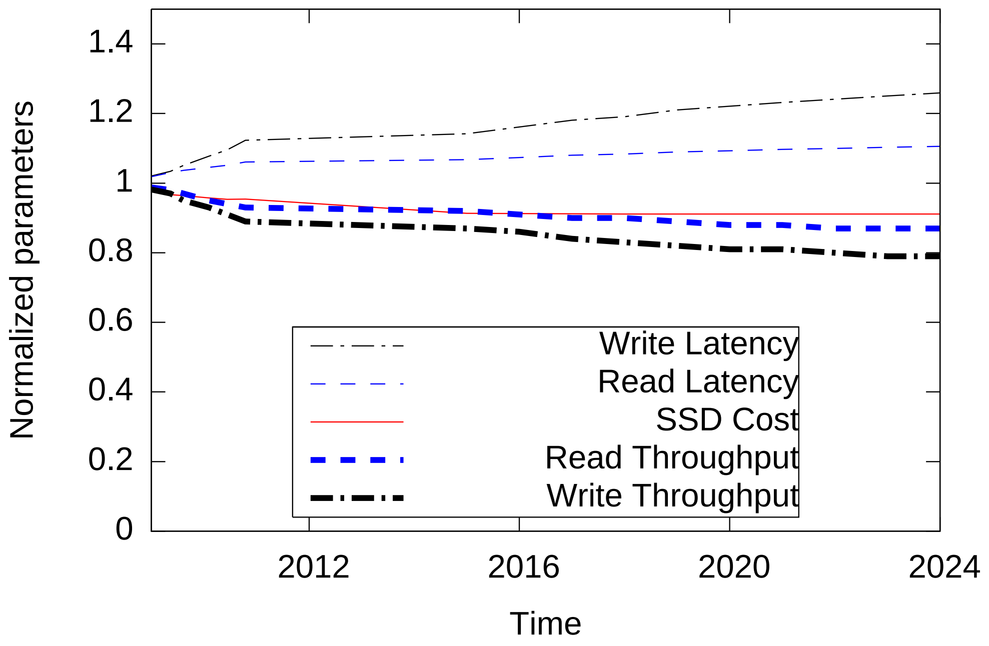


Figure 3.9. Scaling of All Parameters While the cost of an MLC-based SSD remains roughly constant, read and particularly write performance decline.

3.4 Conclusion

The technology trends we have described put SSDs in an unusual position for a cutting-edge technology: SSDs will continue to improve by some metrics (notably density and cost per bit), but everything else about them is poised to get worse. This makes the future of SSDs cloudy: While the growing capacity of SSDs and high IOPS rates will make them attractive in many applications, the reduction in performance that is necessary to increase capacity while keeping costs in check may make it difficult for SSDs to scale as a viable technology for some applications.

Later chapters in this thesis (Chapters 5, 6 and 7) describe our solutions to help mitigate these trends by leveraging flash's unique behaviors. The future work, described in Chapter 8, proposes system-wide changes to expose trade-offs to enable even greater enhancements.

Acknowledgements

This chapter contains material from “The Bleak Future of NAND Flash Memory” by Laura M. Grupp, John D. Davis and Steven Swanson, which appears in *FAST ’12: The 10th USENIX Conference on File and Storage Technologies*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2012 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1(212)869-0481, or permissions@acm.org.

Chapter 4

Flash Management and Experimental Infrastructure

SSDs comprise not only an array of flash memory, but a controller containing algorithms to interface between flash's idiosyncratic access requirements and the disk-like interface presented to the computer system. The previous chapter leveraged a common design and employed a very simple model for management overhead. However, the physical organization of the flash memories and the logical organization that the FTL imposes on them have a large impact on the storage system's performance.

In this chapter we explore the architecture of the flash memory system including the flash controller, the buses that connect flash chips and the firmware layer that manages the array. Section 4.1 describes the basic algorithms needed in all FTLs, and details the development and variety of publicly available designs. Next, Section 4.2 describes our FTL simulation infrastructure, as well as how we isolate important policy decisions.

4.1 Management Schemes for NAND Flash Memory

An SSD's Flash Translation Layer (FTL) resides in the drive's controller and manages the flash while presenting as a block-based storage device. FTLs are mandated by the technology's bulk erase operation, and also enable wear leveling algorithms to help mitigate the effects of flash's limited lifetime.

FTLs achieve these two goals simultaneously by applying accesses to the flash in a log-structured fashion and performing dynamic address reassignment. Most flash management schemes do this by translating between logical blocks addresses (LBAs) and physical addresses and updating the mapping with each update of the data (copy-on-write).

This serves to improve performance by decoupling the latency-critical program operation from the high-latency erase operation into two streams of execution. The fast *programming stream* handles incoming write requests by programming to a previously erased area of flash. Meanwhile, the slow *erasing stream* is responsible for erasing old copies of data in order to provide erased flash for the programming stream.

At the heart of these two streams of execution is the controller's view of flash as a slurry of independent blocks. The programming stream draws blocks from a pool of erased blocks while the erasing stream must replenish this pool through, ideally background, garbage collection routines.

The FTL erases the blocks as part of its garbage collection routine. When the reserved pool of erased blocks fall below a set minimum, the system must block all incoming accesses while it erases. To prevent this, garbage collection may proceed in the background either on idle chips or during memory system idle time.

A number of important design choices accompany this log-structured system which heavily affect performance and storage overheads. Many designers study workloads and carefully choose where to place incoming accesses, which blocks to reclaim and how many free blocks the system should maintain. Well-coordinated access placement and block reclamation has the power to reduce how much valid data is copied within the device, serving to improve both performance and wear on the device. The amount of free space on the drive can also effect the drive's performance and lifetime. Too little free space can even send the drive's garbage collection into deadlock.

FTLs fall into three broad categories based on the granularity of the indirection map. The sections below describe their basic characteristics and are ordered roughly by level of complexity. Each type of FTL provides trade-offs in terms of performance, memory overhead and life expectancy. Researchers have gathered these ideas into several survey papers, written at various stages of the FTL development period. [34, 26]

4.1.1 Block-Level

The block level FTL was the first and simplest FTL design. It uses a map entry for each flash block (see Figure 4.1). When the controller receives a command to update an area of memory smaller than that of a block, the FTL must perform a read-modify-write on the applicable block. The map allows the write to move to a new block, removing the high-latency erase operation from the critical path.

In this design, space overhead is low and algorithms are simple. The flexibility in the physical placement of logical addresses improves on direct access by lowering write latencies, shifting hot addresses around the device and remapping away from bad blocks. However, latency for writes of small areas remains high. Application of block-based FTLs is best when access sizes are large and available DRAM is limited.

A great deal of creativity can be expressed within a top-level block mapping. For example, the Superblock FTL presents a two-level mapping scheme [42, 46]. This scheme first maps a group of blocks in each DRAM map entry and then uses the out-of-band section of flash to store a finer grained map within each group of blocks.

Another variant on the basic block map described above appeared in Ban's NFTL [16]. The system stores a chain of physical locations for each logical block in RAM. On each write, the algorithms walk down the chain until they find the first available page at the correct offset within the block. If the chain does not provide an available

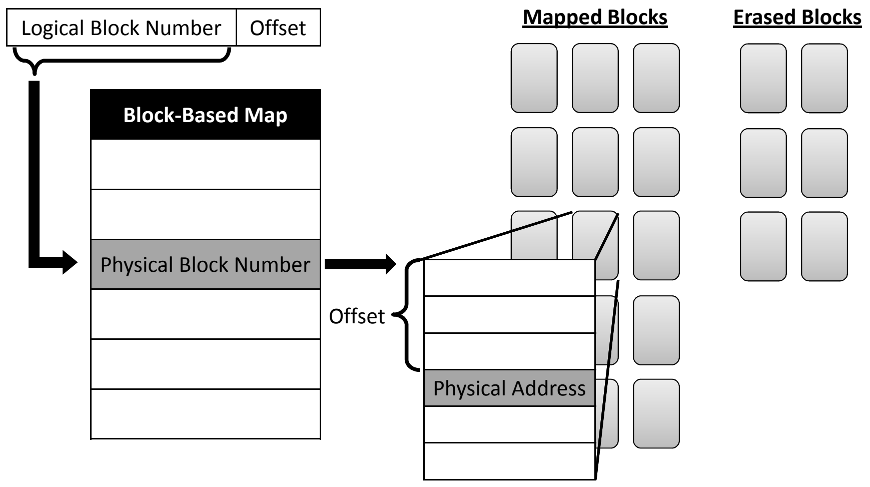


Figure 4.1. Block-Level FTL Scheme This architecture provides a simple mechanism for distributing wear among all blocks in a flash memory array.

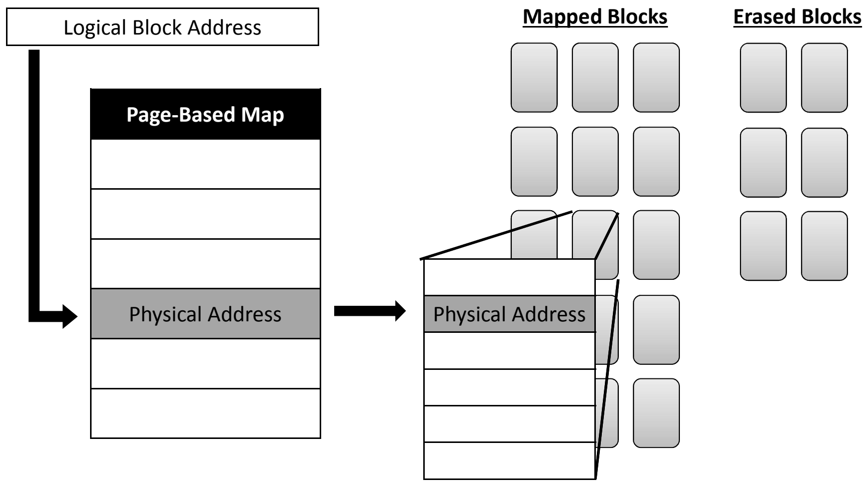


Figure 4.2. Page-Level FTL Scheme This map enables high-performance, fine-grained accesses.

page, a fresh block is tacked to the end of the chain. When new blocks are needed, they are taken from the head of the chains after moving any remaining valid data. In this way, the drive provides a distinct log for each logical block.

Finally, Wells et al. patented a system designed for compressed storage [79]. The logical to physical block map remains static, and garbage collection and compression proceed within each physical block. The design accepts variable sized writes, scheduling them to a log starting at one end of the block. With each of these writes, fixed-sized headers are written in a log beginning at the other side of the block.

4.1.2 Page-Level

As Flash Translation layers became more sophisticated, and systems were able to devote more resources to them, researchers began publishing page-based FTLs [48, 17, 23]. FTLs in this category allow for much higher performance for smaller accesses at the cost of a very large memory footprint (See Figure 4.2). The map maintains an entry for each logical address at the granularity of a flash page.

A read access invokes a simple address lookup before completing the access to flash. Writes invoke a copy-on-write sequence which first writes the incoming data to an erased page and then updates the map. Each new write progresses linearly through the block and once the block fills the algorithms select another erased block.

When the map translates addresses at the page granularity, garbage collection routines must differentiate between *valid* and *invalid* data. Invalid data are stale copies left behind by the copy-on-write policy, whereas valid data are the users' most recently stored information. In order to preserve valid data, the garbage collection algorithms copy it off of the block to a fresh area in flash before erasing the originally targeted block.

Copying valid data from a block can take time and exacerbate write amplification, and so it can be advantageous to choose a block with few or no valid pages. To aid this process further, the system can monitor its workload and schedule writes with similarly hot or cold accesses. Kim et al. explore workloads and advantageous access placement in the context of page-based FTLs [48]. Most hybrid FTLs also exploit this concept.

4.1.3 Hybrid

The third set, hybrid FTLs, combine concepts from both page- and block-level FTLs to achieve high performance for the common case with relatively small memory requirements. One common way to accomplish this, initially presented by Kim et al. [50], is to maintain a map and memory array at the block granularity, while also using "log

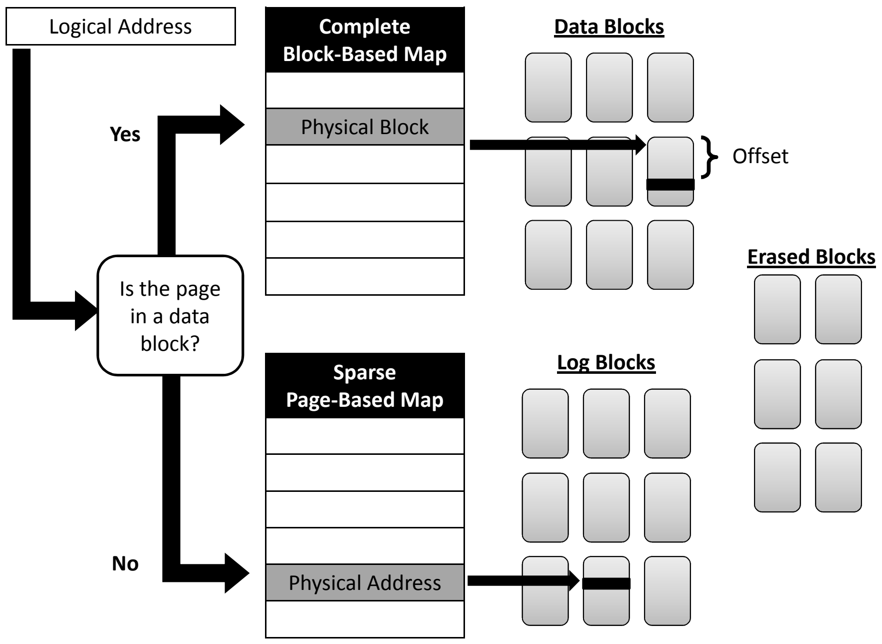


Figure 4.3. Hybrid FTL Scheme Hybrid architectures combine concepts from block- and page-based schemes to achieve high performance with less DRAM overhead.

blocks” to collect incoming accesses (See Figure 4.3). Reading, then, may require an indirect lookup into the log blocks and garbage collection routines must merge valid data from the log block into the main array.

There are three scenarios for merging data from log blocks into the main array. In the simplest case, incoming data was scheduled to a log block in the same order as a data block, and the FTL can perform a *switch* merge. The *partial* merge begins with a log block whose low-numbered pages contain valid data in the appropriate order. The remaining pages are free, and the merge operation copies the data from the data block and finally marks the log block as the data block. The *full* merge encompases all other cases, and requires data to be copied from any number of data and log blocks to a completely erased block.

The efficiency of hybrid designs depends heavily on their ability to recognize and accommodate the incoming access patterns in a way that reduces the frequency and overhead of garbage collection. At the heart of garbage collection in hybrid designs is the merging of the data in log blocks into the data blocks - similar to eviction in a cache. If the FTL directs a perfectly sequential set of accesses to a single log block, the FTL can perform a simple switch merge. Random accesses, however, should be siphoned into a different set of log blocks and the data will need to be copied within the memory array when these log blocks merge into the data blocks. In general, well-designed placement of incoming accesses can cluster valid data onto a near minimum number of blocks, while leaving erasure candidates with few valid pages to copy off-block.

The “AST” suite of FTL designs combine these concepts while often mirroring common cache designs. Block Associative Sector Translation (“BAST” [50]) maintains a block-level map for the primary section of flash storage and a given log block corresponds to one data block. This is similar to direct-mapped caches and performs well with sequential accesses. Fully-Aware Sector Translation (“FAST” [53]) retains a few log blocks to accelerate sequential accesses, and uses the remaining blocks to cache data for any of the data blocks just like a fully-associative cache. Set Associative Sector Translation (“SAST” [59]) limits a set of log blocks to a set of data blocks, and K-associative Sector Translation (“KAST” [25]) limits the associativity of each of its log blocks. Finally, Locality-Aware Sector Translation (“LAST” [54]) employs sequential log blocks for sequential locality as well as random log blocks divided into hot and cold partitions to take advantage of temporal locality.

The authors of KAST also address a challenge prevalent in the services modern SSDs provide: variability in performance. The overwhelming majority of commodity SSDs do not provide consistent access latencies most likely because garbage collection routines interfere with servicing external accesses at unexpected and uncontrollable times. By adjusting the value of K, KAST can limit the worst-cast log block merge time. The community, in recognition of this shortcoming, have set forth a Jedec standard for performance consistency.

4.1.4 Handling Power Cycles

While flash itself retains its data on power cycling, the management techniques often require the use of volatile memory which if lost can corrupt the operation of the entire FTL. While we can easily drop in battery-backed volatile solutions, some FTL designers have devised creative and efficient means of using the flash to store the necessary metadata.

Some designers of page-based FTLs specify a solution for preparing for and recovering from power failure [17]. Their approach is to store a distributed, reverse map in the out-of-band section of each page along with the data. This method does nothing to decrease the performance of accesses. On power up, the controller will then scan the flash to rebuild its volatile data structures. Using the last page of every block to store a summary of this information can decrease the startup time by a factor of 64 to 256. However, this method may still be prohibitively slow for large memory arrays.

The designers of DFTL, a hybrid FTL, propose an alternative approach wherein the metadata is centrally stored in the flash [36]. The FTL provides both a bootable section of flash and some wear leveling by alternately storing the metadata in one of two fixed addresses. Also, certain workloads can cause metadata updates with every access which can induce severe write amplification. One solution is to relax the frequency of metadata updates by only occasionally checkpointing the metadata to flash.

4.2 Mongo, The FTL Simulator

In this work, we study a variety of enhancements to FTL designs using the in-house FTL simulator developed by the author of this thesis. We chose a simple page-based design as the baseline in order to explore changes without the complex interactions of a more complicated design to obscure the effects. In many cases, our proposals extend to other designs as well (for example, to the log block section of hybrid FTLs).

The simulator has undergone several changes over the life of this work. This section first describes the FTL model which underlies all versions of the simulator. Next, it describes the mechanics of the original incarnation of Mongo (used in Chapter 5) followed by a light-weight version called Mango (used in Chapter 6 and Section 7.2). Finally, we present a queue-based version of the Mongo FTL which improves our ability to reason about the flow of operations through the complex algorithms in the FTL (used in Section 7.3).

4.2.1 Mongo's FTL Model

The Mongo Simulator defines the logical organization of the flash array and the interface the processor uses to access the flash storage. The Mongo FTL is an extension of the FTL described in [17]. This FTL allows the application to write to and read from any logical block address (LBA) at random, hiding flash's idiosyncrasies from the rest of the system.

The FTL uses log-structured write operations, filling up one block before moving on to another. To improve bandwidth, we modify the FTL to maintain one log for each chip in the array (see Chapter 5). We refer to the head of each log as a *write point*.

As the FTL writes new data at a write point, the old version of the data for that LBA becomes invalid but remains in the array. In order to remove these stale copies and create room for new data, the FTL performs garbage collection (GC). Ideally in the background, GC algorithms copy valid data from partially-invalid blocks to write points on or off chip, and erase the now fully-invalid blocks to make them ready for new write operations.

When a write point reaches the end of a block, the block is full and the FTL must locate a new, erased block for that write point to continue writing. It keeps a pool of erased blocks on each chip for this purpose. When a chip starts to run short on erased blocks, garbage collection begins to consolidate valid data to create additional erased blocks for that chip. In the best case, garbage collection makes use of idle periods to hide its impact on performance. However, garbage collection latencies are a significant source of performance variability in SSDs.

Our FTL uses two thresholds as parameters for the GC routines. The FTL maintains these thresholds on a per-chip basis, so in the worst case, any single chip can free up resources by taking itself off-line to clean. The first threshold is the *background* threshold. When the FTL finds any chip in the array idle, it performs GC operations on that chip up to the BG threshold. If the number of erased blocks on any chip drops below the second, *emergency* threshold, GC becomes the FTL's top priority for that chip and it will divert all incoming traffic to other chips or block entirely while garbage collection proceeds. In normal operation, the FTL should very rarely enter this "emergency mode."

The LBA table is held in volatile memory, but the FTL must keep a persistent version in case of unexpected power failure. When the FTL writes data to a page, the FTL writes the corresponding LBA to the page's out-of-band section. To accelerate the start-up scan, the FTL stores a summary of this information for the entire block in the block's last page. Since multiple, stale copies of an LBA may exist, the FTL also gives each block a sequence number. The freshest copy of an LBA's data is the last copy written to the block with the highest sequence number.

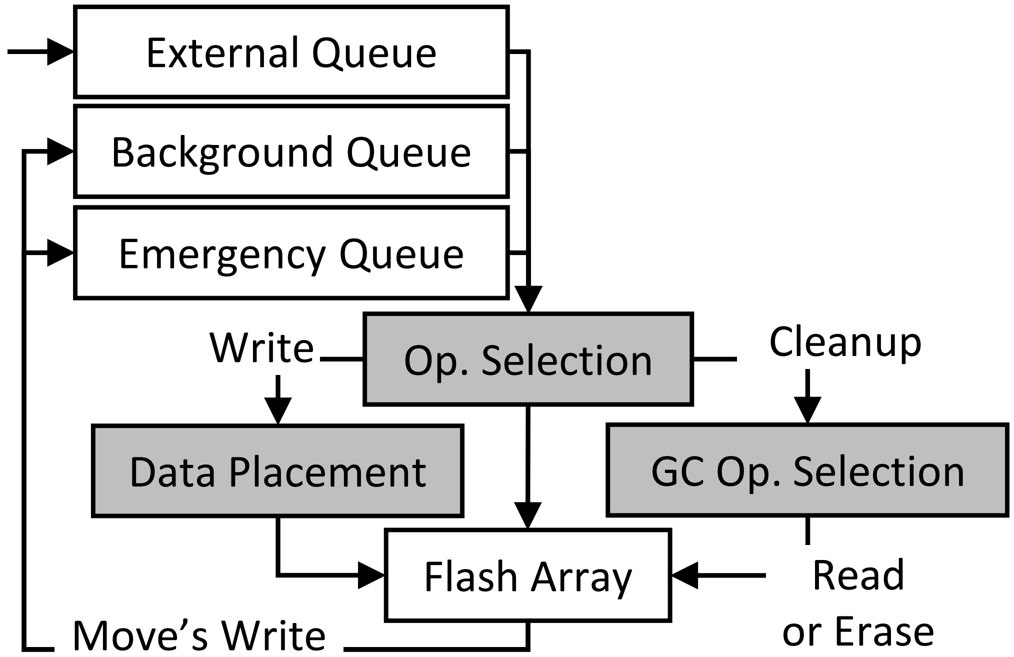


Figure 4.4. Operation Flow Operations move through the FTL's queues and a series of policy decisions (the gray boxes) before executing on a flash chip.

Mongo is the initial design of the simulator, used for the study in Chapter 5. It is a detailed trace-driven flash storage system simulator written in C++. It supports parallel operations between flash devices, models the flash buses and implements our FTL.

Mango is the lightweight version of Mongo sufficient for modeling designs discussed in Chapter 6 and Section 7.2. It is single-threaded and models one chip. Mango is significantly easier to quickly code prototypical designs, runs much faster and with a smaller memory footprint than Mongo.

4.2.2 Queue Structured Mongo

The final study in this thesis (in Section 7.3) required both the power of Mongo, and the clarity in design of Mango. The solution was an FTL structure based on queues of operations. This final revision is optimized for large amounts of parallelism as well as fast simulation of trace idleness.

Figure 4.4 shows the high-level structure of the FTL's operation scheduler. The FTL maintains three queues. The queues hold write, erase, read and *cleanup* operations waiting to execute. External accesses to the SSD enter the *external* queue, background garbage collection operations reside in the *background* queue, and the *emergency* queue holds emergency garbage collection operations. Emergency mode is a rare occurrence, so operations in the background queue are not promoted to the emergency queue.

Operations pass from the queues to the flash array via three distinct policies, marked by the gray boxes in Figure 4.4:

Operation Selection Policy

First, the FTL chooses which operation to execute next. Operations in the emergency queue have the highest priority. If the emergency queue is empty, or contains operations that cannot yet execute (for example, they must access a busy chip or wait for data being read), then an operation is taken from the external queue. Finally, operations are taken from the background queue when the system is idle.

Data Placement Policy

The second policy in the FTL determines where to schedule writes. Because the physical address of an LBA changes with each write, the FTL has the freedom to choose, for example, the fastest page available. The system also uses page sequence numbers for crash recovery (as opposed to the block sequence numbers in Mongo) to add flexibility to the data placement algorithms. In our baseline design, the FTL follows a round-robin approach which avoids busy chips and seeks to maintain an even number of valid LBAs on each chip (more on this second requirement in Section 7.3.3).

GC Operation Selection Policy

The third policy is critical to efficient and flexible operation of garbage collection. One approach to scheduling GC operations in this queued FTL would be to, upon recognizing the pool of erased blocks is too small, queue up all the reads, writes and erases needed to clean the correct number of blocks. However, as time proceeds, and some of these operations begin to execute other operations may change the information used to generate the initial set of queued GC operations. For example, a burst of external activity may invalidate most of the pages in a previously full block, therefore creating a much better choice of block to garbage collect.

Instead of the approach described above, we create an operation called *cleanup* which signals to the FTL that it should perform a single command on the flash array (either a read to begin a move, or an erase of an already invalidated block). When the FTL encounters a cleanup operation, it determines which page's data to move or which block to erase with the most current available information immediately before executing the chosen operation. The "Cleanup Operation Selection" policy in Figure 4.4 makes this determination.

With GC policy reduced to the decision of executing one flash operation at a time, the particular algorithm is simple. Erasing fully invalidated blocks is the best option. When no such blocks are available, we move a page from a block with the least number of valid pages. A move begins with a read operation which, once complete, pushes the paired write operation to the front of the queue from where the cleanup operation originated.

Through the flexible, performance-oriented platform we describe above, we will demonstrate how to more effectively harness the variable performance available in high density flash. Many of these concepts and algorithms will transfer to the more memory-efficient hybrid FTL designs.

Rapid Warm-Up

To ensure steady state behavior, we arrange all of the LBAs randomly throughout the chips in the SSD before starting the simulations. We add enough invalidated pages to fill all blocks to the background threshold. The write points begin on a random page in the write point's assigned block.

4.3 Conclusion

The background and related work presented in this chapter show that improving the FTL is the object of intense work both in industry and academia. In subsequent chapters, we use the in-house simulation tools described here to explore our proposals for informing the FTL with application and device specific improvements.

Acknowledgements

This chapter contains material from "The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs" by Laura M. Grupp, John D. Davis and Steven Swanson, which will appear in *USENIX'13: USENIX Annual Technical Conference*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2013 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a

Chapter 5

Designing an FTL for MapReduce

As our society becomes more information-driven, we have begun to amass data at an astounding and accelerating rate. At the same time, power concerns have made it difficult to bring the necessary processing power to bear on querying, processing, and understanding this data. To accommodate these needs, the author of this thesis helped to design Gordon, a system architecture for data-centric applications that combines low-power processors, flash memory and data centric programming systems to improve performance for data-centric applications while reducing power consumption.

Gordon's flash-based storage system is the key to its power efficiency and performance advantage. This chapter describes the physical organization of the flash storage system as well as the firmware layer that manages it. Although Gordon's storage system targets data-intensive applications, several of the approaches we describe are applicable to more general-purpose storage systems as well.

5.1 The Gordon FTL

A key limitation of the FTLs described thus far is that they allow for only a single write point. As a result, these designs will never allow two operations to proceed in parallel, except in the case of cleaning blocks in the background. For small flash storage systems (like a "USB key"), this is acceptable, but for Gordon it is not.

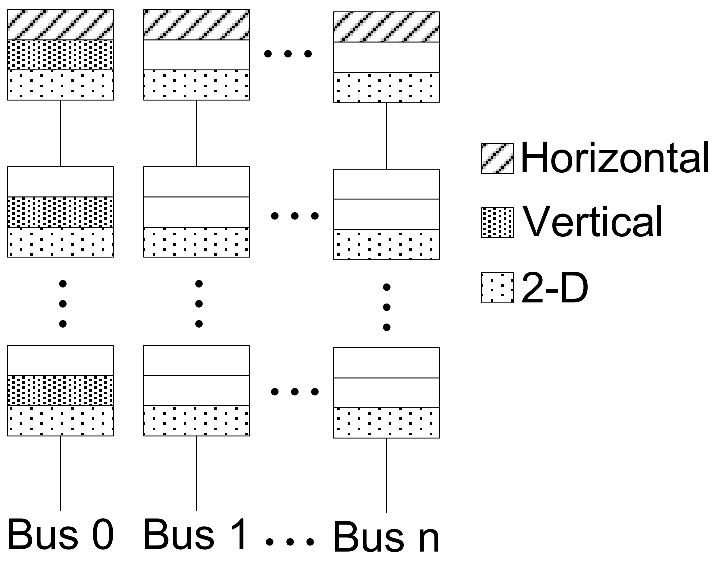


Figure 5.1. Three Approaches to Striping Data Across Flash Arrays Horizontal, vertical, and two-dimensional striping reduce overhead by increasing effective page size.

We use three techniques to solve this problem. The first is to aggressively pursue dynamic parallelism between accesses to the flash array. We have extended our FTL to support multiple write points and spread accesses between them. To maintain the sequence number invariant, each write point has its own sequence number, and once an LBA has been written to a particular write point, future writes must go to the same write point or another write point with a larger sequence number.

The policy for spreading programs across write points selects the write point with the smallest sequence number that can accept data for the LBA while also balancing load across the buses. Using multiple write points does not affect read bandwidth significantly, but it can improve write bandwidth dramatically. Our data show that increasing the number of write points per 133Mhz bus from 1 to 4 increases write bandwidth by 2.8$\times$.

The second approach is to combine physical pages from several dies into "super-pages" (and, therefore, "super-blocks" for erase operations) and manage the flash array at

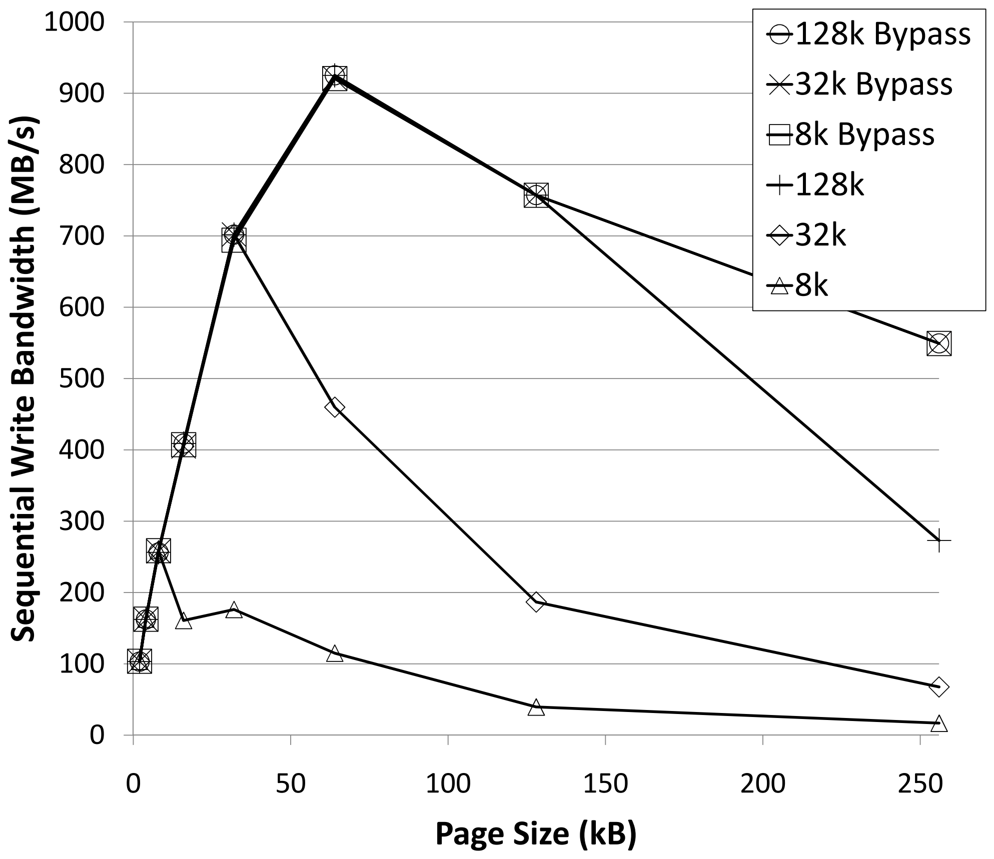


Figure 5.2. Flash Storage Array Performance Without bypassing or write-combining, large super-page sizes lead to decreased performance for small transfers. Adding these features nearly eliminates the effect for sequential accesses.

this larger granularity. We explore three ways to create super-pages: horizontal striping, vertical striping, and 2-dimensional (2D) striping (Figure 5.1).

In horizontal striping each physical page in a super-page is on a separate bus. In this case, access to the physical pages proceeds in parallel. The number of buses in the system limits the size of horizontal super-pages. Horizontal striping is similar to the "ganging" scheme described in [12].

Vertical striping places the physical pages in a super-page on a single bus. The bus transfers each physical page's data in turn, but the program and read operations can occur in parallel across the dies. The number of dies per bus limits the size of vertical super-pages.

The final scheme, 2D striping, combines horizontal and vertical striping to generate even larger super-pages. With 2-D striping, it is possible to divide the array into rectangular sets of chips that are part of the same horizontal and vertical stripes. Our FTL provides one write point for each of these sets.

2D striping trades parallelism between operations for parallelism within a single operation. It also reduces management overhead by reducing the total number of super-pages in the array. This is very important, since the LBA table for a large flash array can be very large. For instance, for 256GB of flash and 2KB pages, the LBA table is 512MB. The same array with 64KB super-pages requires only 16MB. Since our flash controller has 512MB of storage and our storage array is 256GB, super-pages must be at least 4KB in size. In this case the LBA table consumes 256MB, leaving space for other FTL data.

Large super-pages cause two problems. First, they increase the latency of sub-page accesses, since the FTL will need to read or program more data than requested.

The second danger is that wear-out failures will affect a much larger portion of the array. In a striped system, if a single physical block wears out, the entire super-block must be removed from service. We have stress-tested SLC flash chips to measure their failure rate directly, and our data show that for 64KB super-pages, only 98% of super-blocks will last to 100,000 erase cycles, compared to 99.9% for 2KB pages. For 128KB, only 96% will survive this long. In a striped disk-based system, RAID techniques would provide the means to restore this lost reliability. However, the reliability of RAID systems stems from the possibility of *replacing* failed components. Since flash components are soldered to the Gordon nodes, this is not practical.

Vertical and horizontal striping have complementary effects on how the FTL utilizes the buses that connect the flash devices. Horizontal striping effectively creates a wider, high-bandwidth bus, increasing the raw performance of the array for large reads. Vertical striping further enhances throughput by increasing bus utilization. When writing to a vertical stripe, the FTL can queue up all the data for a program operation and stream out the data without interruption, transferring data to each chip in turn. For reads, the FTL can initiate the read operation on all the chips, and then read data from each one in turn. Interleaving [12] accesses between multiple vertical pages can further increase performance.

5.2 The Flash Array Hardware

The flash controller implements Gordon's FTL and provides the hardware interface to the flash storage array. We would like the controller to be able to manage as much storage as possible, but hardware constraints limit its capacity. Flash chips connect to the controller over shared buses. Each bus supports up to four flash packages, each of which contains four dies. We expect that attaching more packages to a 133Mhz (66Mhz DDR) bus would be challenging. Likewise, additional buses would be expensive. Each bus comprises 24 shared pins (eight control and 16 data) and a unique chip enable and ready line for each die, for 56 pins, or a total of 224 pins across four buses. In addition to the flash buses, the controller must also interface with the host processor (150 pins), support

Table 5.1. Flash Characteristics We assume near-future flash memory technology for this study. MLC values are from [51, 62]. SLC numbers are from [38]. Per-die capacity is based on [51]. Bus speed is a projection from [9].

Parameter		**SLC**	**MLC**
Chip Configuration			
Density Per Die (GB)		4	8
Page Size (Bytes)		2048+32	2048+64
Block Size (Pages)		64	128
Bus Width (Bits)		16	16
Operational Latencies (μs)			
Read		25	25
Write		200	800
Erase		2000	2000
Peak Bandwidth (MB/s)			
40MHz Bus	Read	75.8	75.8
	Program	20.1	5.0
133MHz Bus	Read	126.4	126.4
	Program	20.1	5.0
400MHz Bus	Read	161.1	161.1
	Program	20.1	5.0

the system DRAM interface (105 pins), and the narrower, private DRAM interface used to store FTL meta-data (41 pins), for a total of 304 signal pins. Other signals (JTAG, etc.) are needed as well. For comparison, the Atom processor's System Controller Hub (which uses aggressive packaging technology) has 474 I/O pins and a total of 1249 pins [39].

For flash devices, we use the parameters in Table 5.1. The values in the table represent flash technologies projected to be commercially available 1-2 years after this work's initial publication.

5.3 Evaluation of Gordon FTL

To evaluate these alternative organizations, we leverage the simulator described in Section 4.2 and apply both synthetic traces and real workloads. We generate traces of 200,000 random accesses (reads or writes, depending on the trace) of between 2KB and

256KB. use traces from our suite of data-intensive workloads (Table 7.2).

Figure 5.2 demonstrates the impact of increasing super-page size on performance. For super-pages with four or fewer pages, we use horizontal striping, but going beyond four requires vertical striping as well. The data show write bandwidth for varying transfer sizes. The data also show the negative impact on bandwidth for large super-page sizes and small transfers. For instance, bandwidth for 32KB and 128KB transfers is the same until the page size exceeds 32KB. Then, the extra work required to read the rest of the super-page reduces performance.

Our workloads present a mix of large and small transfers. While 88-93% of bytes read and written are part of transfers of 120KB or larger, roughly half of individual transfers are only 8KB. As a result, setting our page size to 128KB would roughly double the amount of data the FTL had to read and write.

Careful examination of traces from our workloads shows a clear pattern in accesses: an 8KB access aligned on a 128KB boundary followed immediately by a 120KB access to the rest of the same 128KB region. To exploit the pattern, we added a simple bypassing mechanism to our FTL that merges incoming read requests with pending requests to the same page and also caches the result of the last read. Our FTL also performs limited write combining by merging write requests to the same page when possible. Figure 5.2 shows that this mechanism nearly eliminates the negative impact of large pages for small sequential reads. The small dip in performance is due to the overhead of scanning the queue of pending reads. The effect is similar for reads. As expected, bypassing and combining do not improve random access performance.

The data also show the necessity of interleaving requests to allow some chips to program or read data while data are flowing over the bus. For 256KB pages, the super-page spans the entire array, so only one request can be active at once. As a result, performance degrades even with write combining. Using 64KB pages gives the best

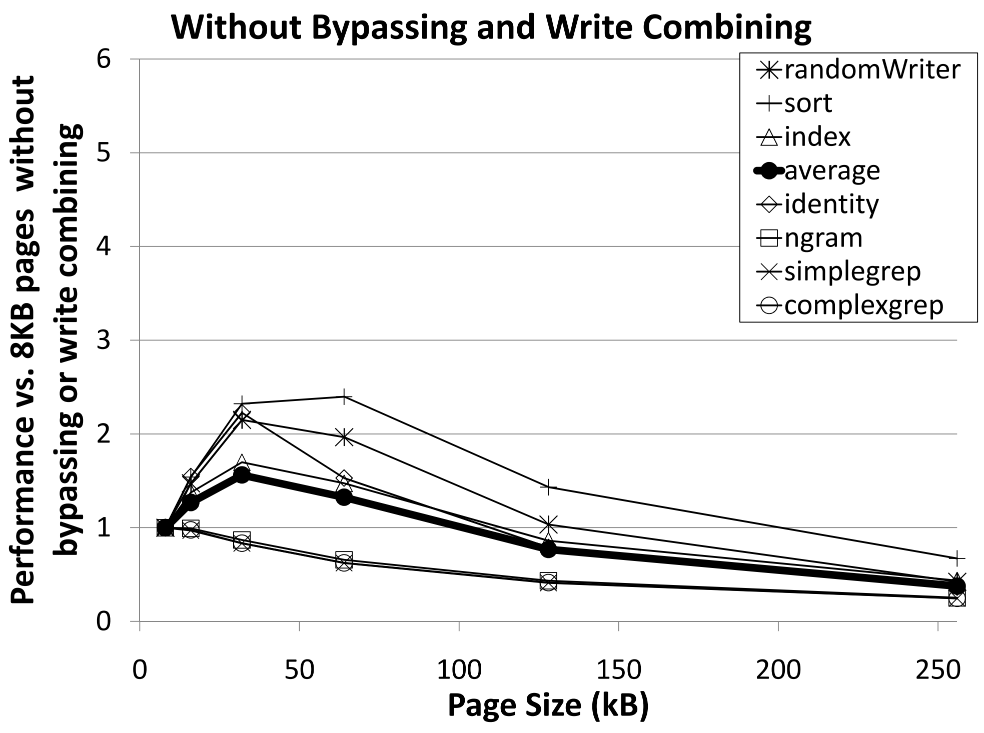


Figure 5.3. Application-Level IO Performance Without Bypassing Without read bypassing and write combining, large pages degrade performance.

performance and allows four operations to proceed in parallel. This is approximately what we would expect: Transferring 64KB of data over four 133Mhz buses takes 58μs, or about 0.29 times the program time for our flash chips. If the transfer time were 1/3 the program time, bus utilization would be nearly 100%.

Figures 5.3 and 5.4 show how page size and bypassing affect overall storage performance of our data-intensive workloads, normalized to 8KB pages. In Figure 5.3, bypassing is disabled, and large pages benefit Index, Identity, RandomWriter, and Sort. Figure 5.4 shows performance with bypassing. There, 64KB pages provide between 1.1$\times$ and 6.3$\times$ speedups for all applications.

For sequential accesses (and random accesses to super-pages of similar size) our storage array delivers 900MB/s of bandwidth for reads and writes. Our design uses

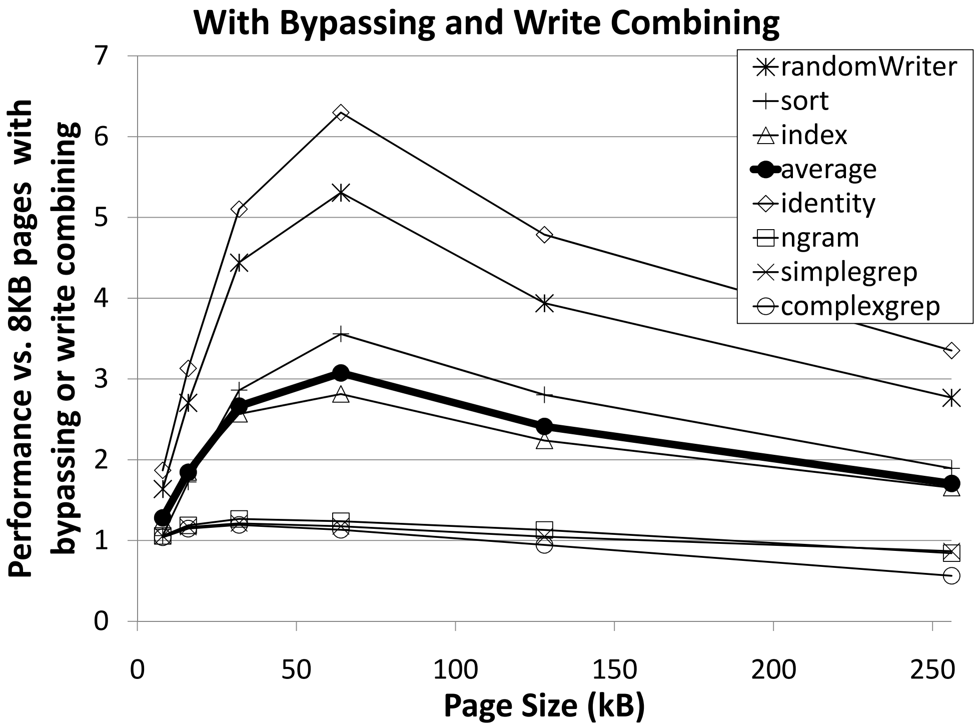


Figure 5.4. Application-level IO Performance With Bypassing Read bypassing and write combining allow all applications to benefit from larger page sizes.

16-bit SLC flash devices and assumes a 133Mhz bus. Currently, 40Mhz devices are commonly available. Moving to these devices would reduce peak bandwidth by 60% and reduce average I/O performance for our applications by the same amount. Conversely, moving to a 400Mhz (200Mhz DDR) bus would increase performance by between 20% for write-intensive applications and 2.5$\times$ for read-intensive applications. It would also push peak read bandwidth to 2.2GB/s. Peak write bandwidth would rise to 1.1GB/s. Write performance is limited by the aggregate program bandwidth of the chips on each bus. Moving to MLC devices also reduces performance because of their long program times, reducing performance by between 2% (read-intensive) and 70% (write-intensive).

5.4 Conclusion

The FTL designed for Gordon relieves the performance bottleneck in previously published FTL designs in order to leverage the flash array's inherent parallelism. We tuned the parameters to accommodate Gordon's data-centric applications, however the mechanisms for increased parallelism (many write points, data striping, bypassing and write combining) apply well to large-scale SSD designs.

Acknowledgements

This chapter contains material from "Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications" by Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson which appears in *ASPLOS'09: The 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. The dissertation author was the second investigator and author of this paper. This material is copyright ©2009 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed

Chapter 6

Device-Informed FTL Modifications

The previous chapter describes changes we made to the systems-level algorithms in an SSD to better leverage the parallelism inherent in an array of flash chips. However, there are many more modifications that can be made by combining knowledge of FTL's algorithms (Chapter 4) with detailed characterization of flash's idiosyncracies (Chapter 2). The work described in this thesis has enabled studies which

- improve the reliability of flash by informing the channel model used in flash-specific ECC [33, 81],
- expose a unique device fingerprint based on the silicon manufacturing variations in flash [65],
- identify relationships between pairs of pages critical for fault tolerance in the face of power failure [74, 75]
- and validate a model for the power consumption of flash memory [58, 57].

This chapter describes two additional works in detail. First, Section 6.1 describes an encoding scheme which potentially increases the lifetime of some flash chips. Next, Section 6.2 proposes a mechanism which removes the performance-crippling aspect of

Table 6.1. 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.

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

single-file erasure from SSDs. Chapter 7 follows with an FTL design which improves the latency and energy consumption of accesses to SSDs.

6.1 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.

The data in Chapter 2 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 [69].

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 to any given bit 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).

Table 6.1 illustrates a simple WOM code that we have implemented. It uses three physical bits to represent two logical bits and allows two logical 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 the first and second generation code words are complements, so that the second generation pattern cannot be programmed. 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 2.3 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-encoding 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 6.1 displays these energy savings for

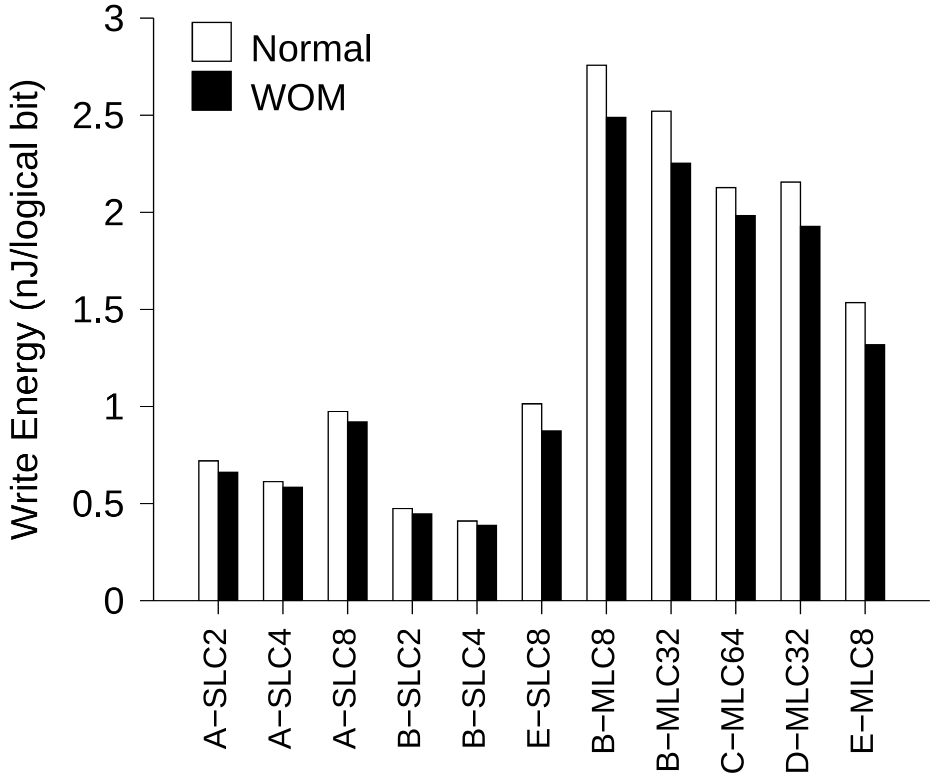


Figure 6.1. Effective Program Energy Because WOM codes decrease the number of erases per logical bit programmed, using them reduces programming energy by 9.5% on average.

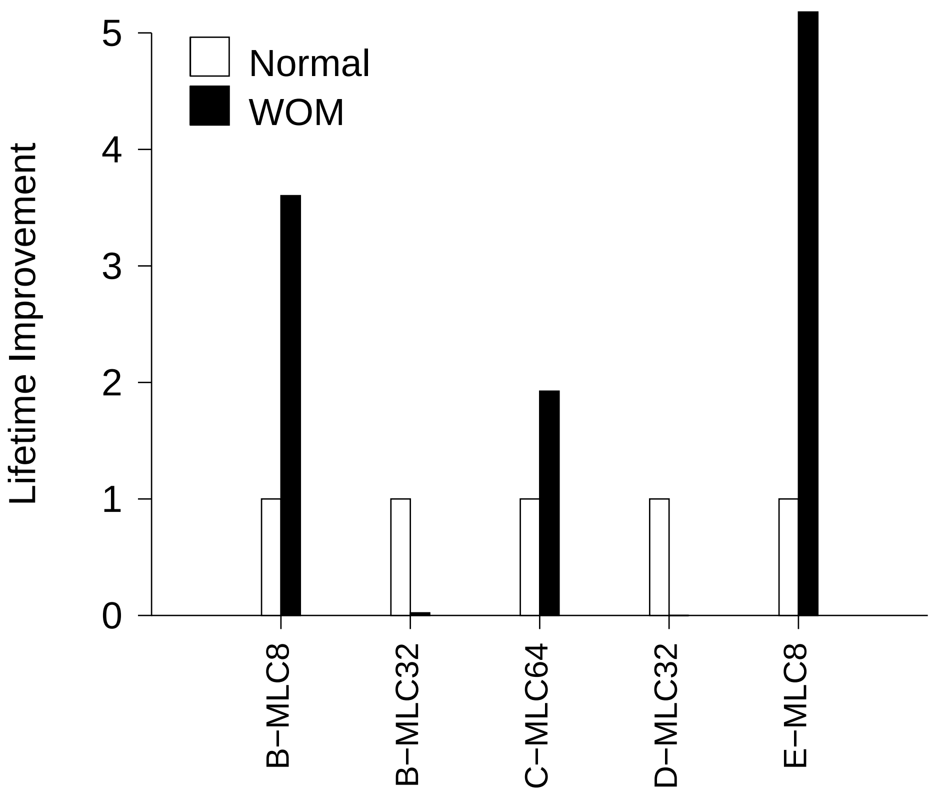


Figure 6.2. Effective Device Lifetime Measured in logical bytes written before the device reaches the fatal error rate, WOM codes allows up to 5.2 times longer lifetime.

each chip.

The second measurable advantage of using WOM codes is a commensurate increase in useful device lifetime. For the purposes of measuring WOM codes' effects, we define the *fatal error rate* to be the error rate at the recommended lifetime of the device under normal operation (without reprogramming or WOM-encoding). We then measure the amount of logical data written to the device before it begins to experience the fatal error rate.

Figure 6.2 shows this increase in effective lifetime. SLC chips are not graphed because the WOM-encoded chips showed no errors even after programming 1.33 times more data than the baseline. 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.

6.2 Erasing Files

Reliably erasing data from storage media is a critical component of secure data management. While sanitizing entire disks and individual files is well-understood for hard drives, flash-based SSDs' address indirection map and copy-on-write algorithm present new challenges. (See Chapter 5)

Wei et. al. [77] find that none of the existing hard drive-oriented techniques for individual file sanitization are effective on SSDs. This is because FTL complexity makes it difficult to reliably access a particular physical storage location. Circumventing this problem requires changes in the FTL. Previous work in this area [52] used encryption to support sanitizing individual files in a file system custom built for flash memory. This approach makes recovery from file system corruption difficult and it does not apply to generic SSDs.

This section describes FTL support for sanitizing arbitrary regions of an SSD's logical block address space. The extensions we describe leverage detailed measurements of flash memory characteristics (validated in Section 6.2.1). We then present (Section 6.2.2) and evaluate (Section 6.2.3) three ways an FTL can support single-file sanitization.

6.2.1 Scrubbing LBAs

Sanitizing an individual LBA is difficult because the flash page it resides in may be part of a block that contains useful data. Since flash only supports erasure at the block level, it is not possible to erase the LBA's contents in isolation without incurring the high cost of copying the entire contents of the block (except the page containing the target LBA) and erasing it.

However, *programming* individual pages is possible, so an alternative would be to re-program the page to turn all the remaining 1s into 0s. We call this *scrubbing* the page. A scrubbing FTL could remove remnant data by scrubbing pages that contain stale copies of data in the flash array, or it could prevent their creation by scrubbing the page that contained the previous version whenever it wrote a new one.

The catch with scrubbing is that manufacturer datasheets require programming the pages within a block in order to reduce the impact of program disturb effects that can increase error rates. Scrubbing would violate this requirement. However, Chapter 2 shows that the impact of reprogramming varies widely between pages and between flash devices, and that, in some cases, reprogramming (or scrubbing) pages would have no effect.

To test this hypothesis, we use our flash testing board to scrub pages on 16 of the chips in Table 2.2 and measure the impact on error rate. The chips in this study span six

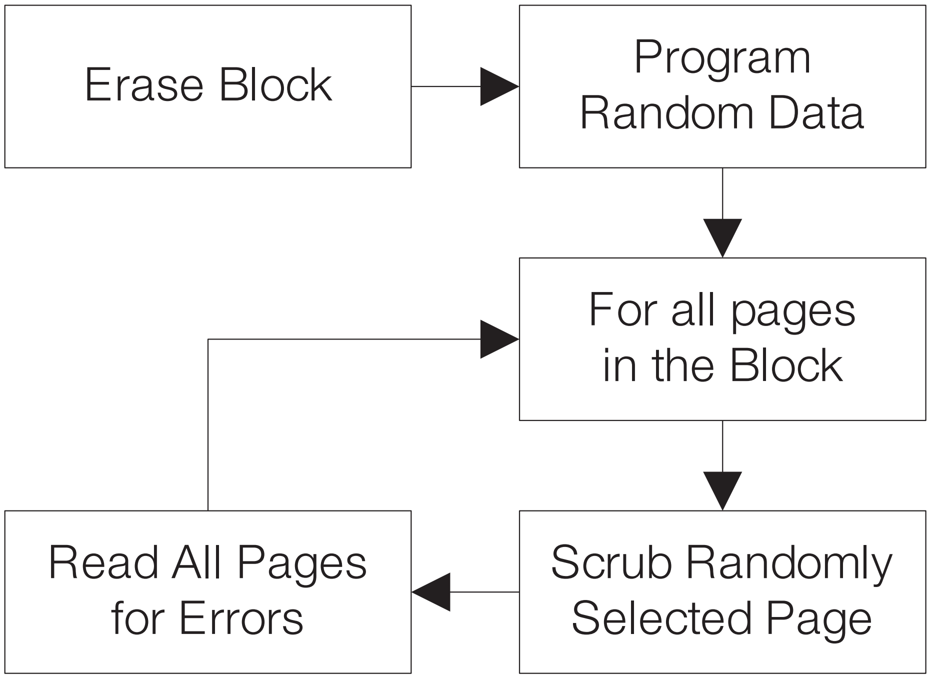

Erase Block
Program Random Data
For all pages in the Block
Scrub Randomly Selected Page
Read All Pages for Errors

Figure 6.3. Testing Data Scrubbing To determine whether flash devices can support scrubbing we programmed them with random data, randomly scrubbed pages one at a time, and then checked for errors.

manufacturers, five technology nodes and include both MLC and SLC chips.

Figure 6.3 describes the test we ran. First, we erase the block and program random data to each of its pages to represent user data. Then, we scrub the pages in random order. After each scrub, we read all pages in the block to check for errors. Flash blocks are independent, so checking for errors only within the block is sufficient. We repeated the test across 16 blocks spread across each chip.

The results showed that, for SLC devices, scrubbing did not cause any errors at all. This means that the number of scrubs that are acceptable – the *scrub budget* – for SLC chips is equal to the number of pages in a block.

For MLC devices, determining the scrub budget is more complicated. First, scrubbing one page invariably caused severe corruption in exactly one other page. This

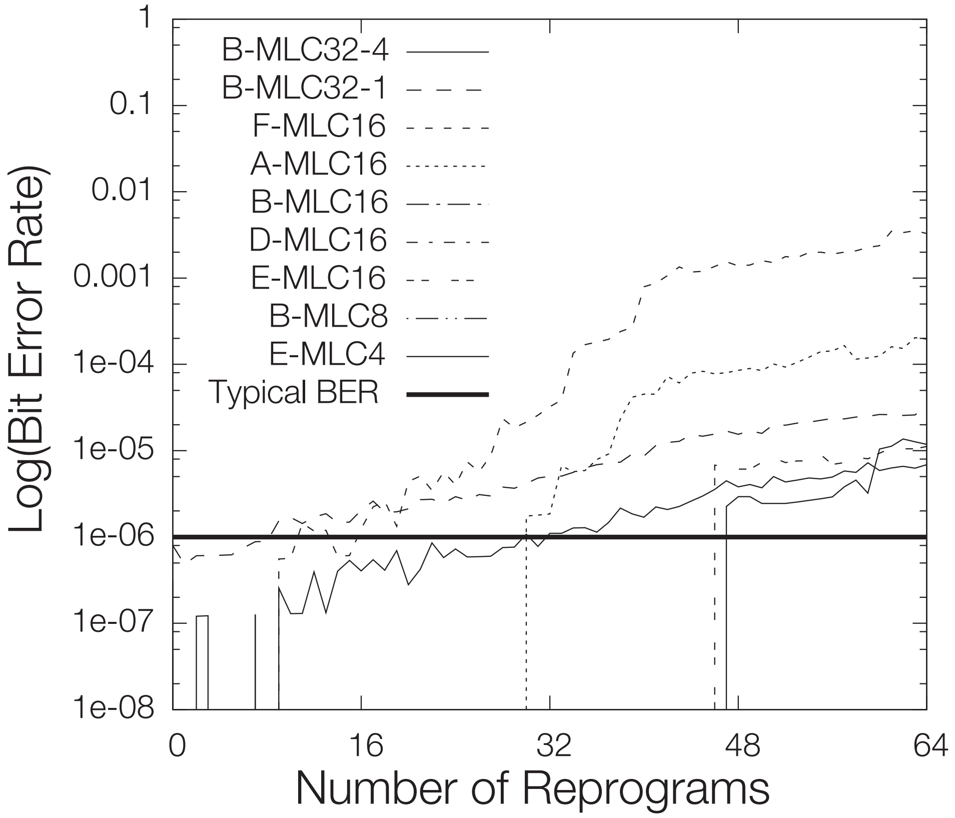


Figure 6.4. Behavior Under Data Scrubbing Scrubbing causes more errors in some chips than others, resulting in wide variation of scrub budgets for MLC devices.

occurred because each transistor in an MLC array holds two bits that belong to different pages, and scrubbing one page reliably corrupts the other. Fortunately, it is easy to determine the paired page layout in all the chips we have tested, and the location of the paired page of a given page is fixed for a particular chip model. The paired page effect means that the FTL must scrub both pages in a pair at the same time, relocating the data in the page that was not the primary target of the scrub.

Figure 6.4 shows bit error rates for MLC devices as a function of scrub count, but excluding errors in paired pages. The data show that for three of the nine chips we tested, scrubbing caused errors in the unscrubbed data in the block. For five of the remaining devices, errors start to appear after between 2 and 46 scrubs. The final chip, B-MLC32-1, showed errors without any scrubbing. For all the chips that showed errors, error rates increase steeply with more scrubbing (the vertical axis is a log scale).

It may be possible to reduce the impact of scrubbing (and, therefore, increase the scrub budget) by carefully measuring the location of errors caused by scrubbing a particular page. Program disturb effects are strongest between physically adjacent cells, so the distribution of scrubs should affect the errors they cause. As a result, whether scrubbing a page is safe would depend on which other pages the FTL has scrubbed in the block, not the number of scrubs.

The data in the figure also show that denser flash devices are less amenable to scrubbing. The chips that showed no errors (B-MLC16, D-MLC16, and B-MLC8) are 50 nm or 70 nm devices, while the chips with the lowest scrub budgets (F-MLC16, B-MLC32-4, and B-MLC32-1) are 34 or 41 nm devices.

6.2.2 Sanitizing Files in the FTL

The next step is to use scrubbing to add file sanitization support to our FTL. We consider three different methods that make different trade-offs between performance and data security – immediate scrubbing, background scrubbing, and scan-based scrubbing.

These methods will eliminate all remnants in the drive's spare area (i.e., that are not reachable via a logical block address). As a result, if a file system does not create remnants on a normal hard drive (e.g., if the file system overwrite a file's LBAs when it performs a delete), it will not create remnants when running on our FTL.

Immediate scrubbing provides the highest level of security: write operations do not complete until the scrubbing is finished – that is, until FTL has scrubbed the page that contained the old version of the LBA's contents. In most cases, the performance impact will be minimal because the FTL can perform the scrub and the program in parallel.

When the FTL exceeds the scrub budget for a block, it must copy the contents

Table 6.2. Benchmark and application traces We use traces from seven benchmarks and workloads to evaluate scrubbing.

Name	Total Accesses	Reads	Description
Patch	64 GB	83%	Applies patches to the Linux kernel from version 2.6.0 to 2.6.29
OLTP	34 GB	80%	Real-time processing of SQL transactions
Berkeley-DB Btree	34 GB	34%	Transactional updates to a B+tree key/value store
Financial	17 GB	15%	Live OLTP trace for financial transactions.
Build	5.5 GB	94%	Compilation of the Linux 2.6 kernel
Software devel.	1.1 GB	65%	24 hour trace of a software development work station.
Swap	800 MB	84%	Virtual memory trace for desktop applications.

of the block's valid pages to a new block and then erase the block before the operation can complete. As a result, small scrub budgets (as we saw for some MLC devices) can degrade performance. We measure this effect below.

Background scrubbing provides better performance by allowing writes to complete and then performing the scrubbing in the background. This results in a brief window when remnant data remains on the drive. Background scrubbing can still degrade performance because the scrub operations will compete with other requests for access to the flash.

Scan-based scrubbing incurs no performance overhead on normal write operations but adds a command to sanitize a range of LBAs by overwriting the current contents of the LBAs with zero and then scrubbing any storage that previously held data for the LBAs. This technique exploits the reverse (physical to logical) address map that the SSD stores to reconstruct the logical-to-physical map.

To execute a scan-based scrubbing command, the FTL reads the summary page from each block and checks if any of the pages in the block hold a copy of an LBA that the scrub command targets. If it does, the FTL scrubs that page. If it exceeds the scrub budget, the FTL will need to relocate the block's contents.

We also considered an SSD command that would apply scrubbing to specific write operations that the operating system or file system marked as "sanitizing." However, immediate and background scrubbing work by guaranteeing that only one valid copy of an LBA exists by always scrubbing the old version when writing the new version. Applying scrubbing to only a subset of writes would violate this invariant and allow the creation of remnants that a single scrub could not remove.

6.2.3 Results

To understand the performance impact of our scrubbing techniques, we implemented them in a trace-based FTL simulator. The simulator implements the baseline FTL described above and includes detailed modeling of command latencies (based on measurements of the chips in Table 2.2) and garbage collection overheads. For these experiments we used E-SLC8 to collect SLC data and F-MLC16 for MLC data. We simulate a small, 16 GB SSD with 15% spare area to ensure that the FTL does frequent garbage collection even on the shorter traces.

Table 6.2 summarizes the seven traces we used in our experiments. They cover a wide range of applications from web-based services to software development to databases. We ran each trace on our simulator and report the latency of each FTL-level page-sized access and trace run time. Since the traces include information about when each of the applications performed each IO, the change in trace run-time corresponds to application-level performance changes.

Immediate and Background Scrubbing

Figure 6.5 compares the write latency for immediate and background scrubbing on SLC and MLC devices. For MLC, we varied the number of scrubs allowed before the FTL must copy out the contents of the block. The figure normalizes the data to the baseline configuration that does not perform scrubbing or provide any protection against remnant data.

For SLC-based SSDs, immediate scrubbing causes no decrease in performance, because scrubs frequently execute in parallel with the normal write access.

In MLC devices, the cost of immediate scrubbing can be very high if the chip can tolerate only a few scrubs before an erase. For 16 scrubs, operation latency increases by 6.4× on average and total runtime increases by up to 11.0×, depending on the application.

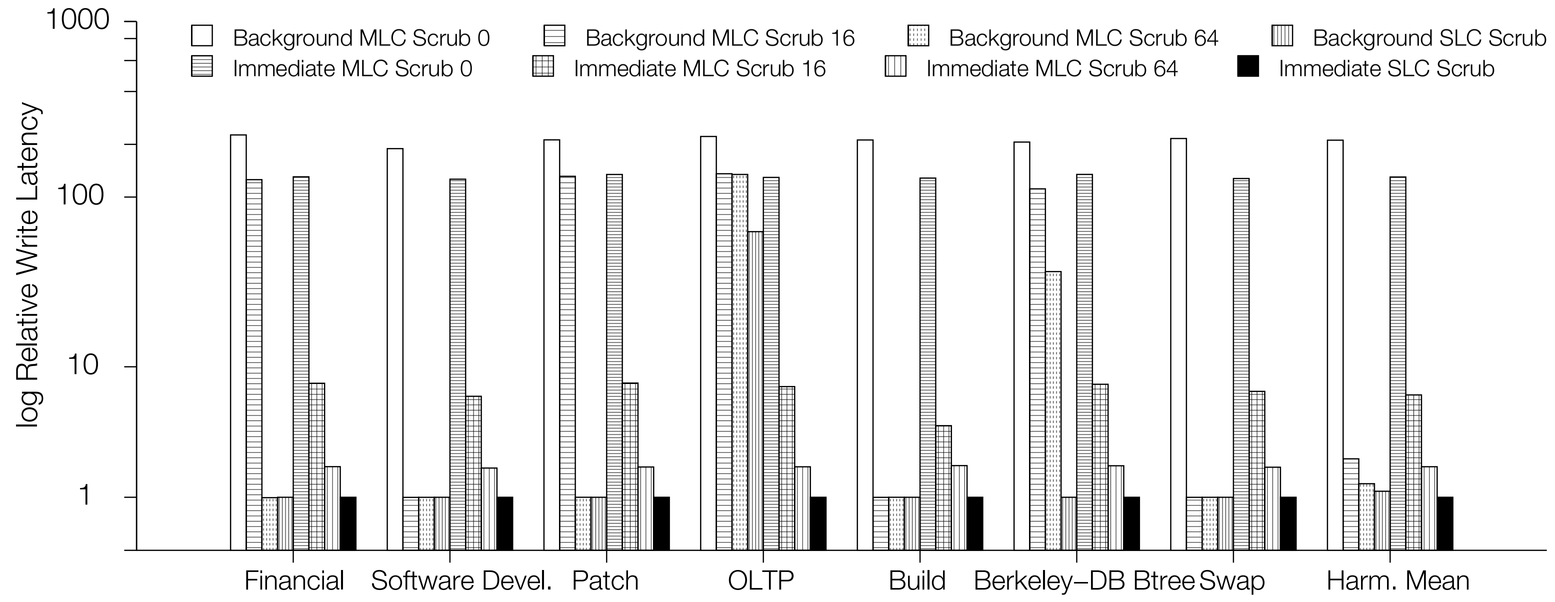


Figure 6.5. Immediate and Background Scrubbing Performance For chips that can withstand at least 64 scrub operations, both background and immediate scrubbing can prevent the creation of data remnants with minimal performance impact. For SLC devices (which can support unlimited scrubbing), background scrubbing has almost no effect and immediate scrubbing increases write latency by about 2×.

For 64 scrubs, the cost drops to 2.0× and 3.2×, respectively.

However, even a small scrub budget reduces latency significantly compared to relying on using erases (and the associated copy operations) to prevent remnants. Implementing immediate sanitization with just erase commands increases operation latency by 130× on average (as shown by the "Scrub 0" data in Figure 6.5).

If the application allows time for background operations (e.g., Build, Swap and Dev), background scrubbing with a scrub budget of 16 or 64 has a negligible effect on performance. However, when the application issues many requests in quick succession (e.g., OLTP and BDB), scrubbing in the background strains the garbage collection system and write latencies increase by 126× for 16 scrubs and 85× for 64 scrubs. In contrast, slowdown for immediate scrubbing range from just 1.9 to 2.0× for a scrub budget of 64 and from 4.1 to 7.9× for 16 scrubs.

Scrubbing also increases the number of erases required and, therefore, speeds up program/erase-induced wear out. Our results for MLC devices show that scrubbing increased wear by 5.1× for 16 scrubs per block and 2.0× with 64 scrubs per block. Depending on the application, the increased wear for chips that can tolerate only a few scrubs may or may not be acceptable. Scrubbing SLC devices does not require additional erase operations.

Finally, scrubbing may impact the long-term integrity of data stored in the SSD in two ways. First, although manufacturers guarantee that data in brand new flash devices will remain intact for at least 10 years, as the chip ages, data retention time drops. As a result, the increase in wear that scrubbing causes will reduce data retention time over the lifetime of the SSD. Second, even when scrubbing does not cause errors immediately, it may affect the analog state of other cells, making it more likely that they give rise to errors later. Figure 6.4 demonstrates the analog nature of the effect: B-MLC32-4 shows errors that come and go for eight scrubs.

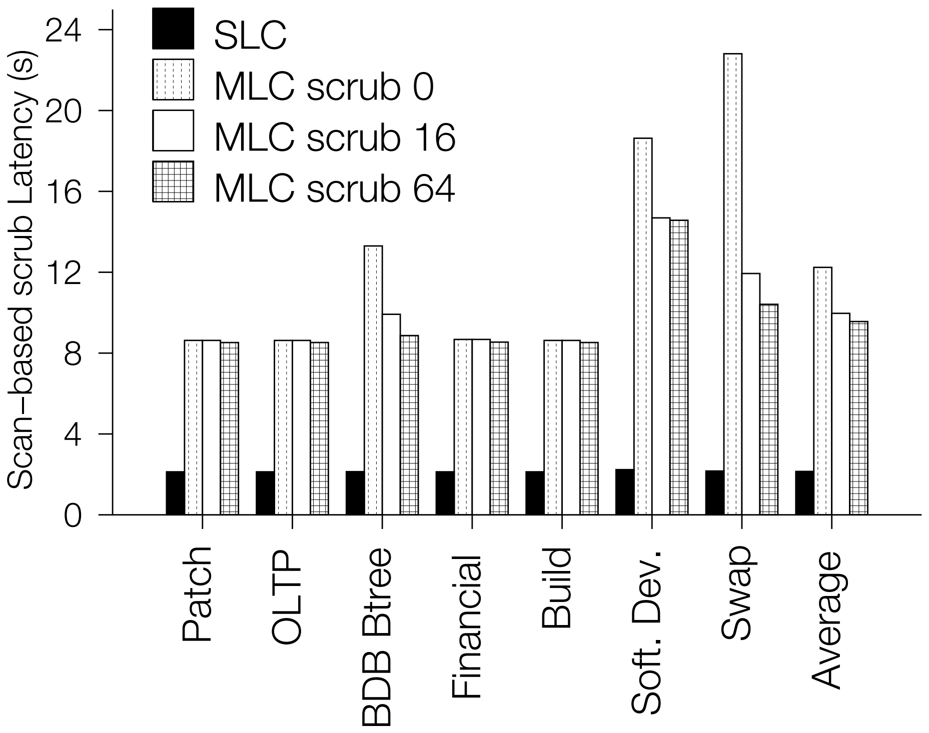


Figure 6.6. Scan-Based Scrubbing Latency The time to scrub 1 GB varies with the number of scrubs each block can withstand, but in all cases the operation takes less than 30 seconds.

Overall, both immediate and background scrubbing are useful options for SLC-based SSDs and for MLC-based drives that can tolerate at least 64 scrubs per block. For smaller scrub budgets, both the increase in wear and the increase in write latency make these techniques costly. Below, we describe another approach to sanitizing files that does not incur these costs.

Scan-Based Scrubbing

Figure 6.6 measures the latency for a scan-based scrubbing operation in our FTL. We ran each trace to completion and then issued a scrub command to 1 GB worth of LBAs from the middle of the device. The amount of scrubbing that the chips can tolerate affects performance here as well: scrubbing can reduce the scan time by as much as 47%.

However, even for the case where we must use only erase commands (MLC-scrub-0), the operation takes a maximum of 22 seconds. This latency breaks down into two parts – the time required to scan the summary pages in each block (0.64 s for our SLC SSD and 1.3 s for MLC) and the time to perform the scrubbing operations and the resulting garbage collection. The summary scan time will scale with SSD size, but the scrubbing and garbage collection time are primarily a function of the size of the target LBA region. As a result, scan-based scrubbing even on large drives will be quick (e.g., $\sim$62 s for a 512 GB drive).

6.3 Conclusion

The FTL modifications we describe represent a class of FTL improvements inspired by a deep understanding of device behavior. Through flash-specific encoding schemes we improve SSD lifetime by up to 5.2x and energy consumption by 9.5%, and by understanding flash's re-programmability we propose a limited but powerful page-size erase mechanism called scrub. The following chapter explores one final set of improvements designed to leverage page-granularity variations for both power and latency savings.

Acknowledgements

This chapter contains material from "Reliably Erasing Data From Flash-Based Solid State Drives" by Michael Wei, Laura M. Grupp, Fredrick E. Spada, and Steven Swanson, which will appear in *FAST'11: The 9th USENIX Conference on File and Storage Technologies*. The dissertation author was the second investigator and author of this paper. This material is copyright ©2011 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not

This chapter contains material from "Characterizing Flash Memory: Anomalies, Observations, and Applications" by Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Eitan Yaakobi, Steven Swanson, Paul H. Siegel, and Jack K. Wolf, which appears in *Micro'09: the 42nd Annual International Symposium on Microarchitecture*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2009 by the Institute of Electrical and Electronics Engineers (IEEE). If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Chapter 7

Mitigating the Negative Properties of High Density Flash

Chapter 3 showed us that as we increase the density of our flash technology, the chips must sacrifice performance, power and reliability. It is the task of the management technology to work around the unfavorable properties to present a usable technology even at high density. Related work tackles flash reliability by informing the channel model for ECC [33, 81]. With respect to performance and power, we must tap in to the SLC-like pages discovered in Chapter 2.

The final contribution of this thesis is an FTL design and set of mechanisms which give the FTL designer access to the differently-behaved regions within multi-bit flash technology for improved performance and energy efficiency. Such control will allow SSD designs to achieve the high performance and lower power of low-density flash technology and the high density of multi-bit flash.

Section 7.1 first describes the behavior and cause of different regions in multi-bit flash. Next, Section 7.2 presents a simple method of tapping in to the fast and energy efficient pages in MLC flash. Finally, Section 7.3 proposes mechanisms and discussion which enable flexible, efficient and large scale use of distinct page types.

7.1 Page Classification

The significant page-level variation in write performance forms the foundation for this chapter's contributions. This section describes the source of these variations, magnitude of variation we have measured in flash chips, the architectural lay-out of fast and slow pages within each flash chip, and how the FTL can non-destructively detect this pattern.

Each of the 30 chip models (from 6 manufacturers) we have characterized show distinct groups of latencies in proportion with the number of bits stored in each memory element. The variation arises because, although MLC devices store multiple bits on a single floating gate, those bits map into different pages. As a result, the programming operation for the first *fast* bit stored on the gate is much faster than the programming operation for the second *slow* bit, and so on for all additional bits stored in the cell. We refer to individual pages as fast or slow depending on which kind of bits they contain, for the 2-bit cell.

Figure 7.1 shows the latency of a representative sample of SLC, MLC and TLC chips. For each chip we measure the time to write random data to each page of 16 blocks. We divide these measurements into fast, slow and (for TLC) medium page latencies. Slow pages from the average MLC chip are are 4.8x slower than fast pages, with D_MLC_50nm exhibiting the largest gap (6x) and F_MLC_41nm the smallest at 3.5x. Our data show that fast page program latency is comparable to SLC program latency in devices from similar technology generations.

Our previous work [35] reveals two common organizations for fast and slow pages within an MLC block. We now extend those observations to TLC parts as well. All except one manufacturer exhibit the structure in Figure 7.2A. In MLC devices, the first four pages are fast, the last four are slow and every pair of pages mid-block alternate

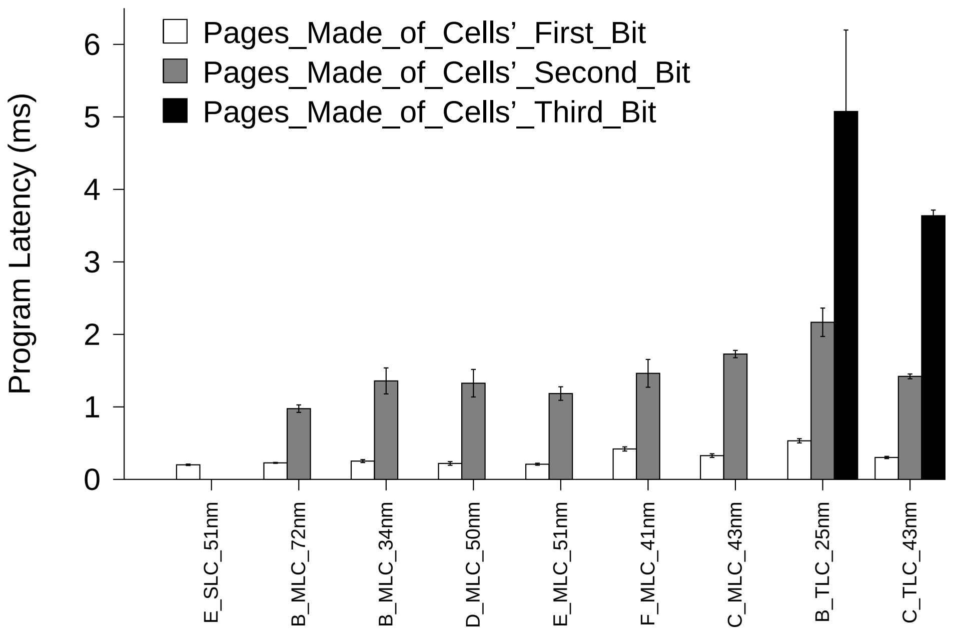


Figure 7.1. Chip Program Latency Multi-bit flash chips retain single-bit performance in their fast pages. The increase in latency is confined to the chips' added capacity.

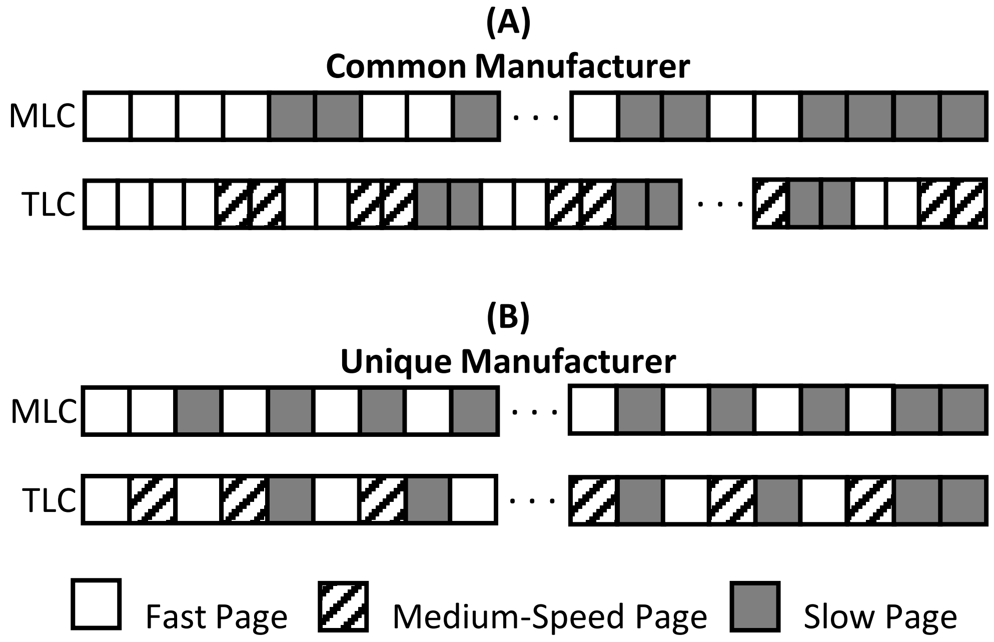


Figure 7.2. Latency Pattern Pages' read and write latencies follow the same pattern within each block of a given chip.

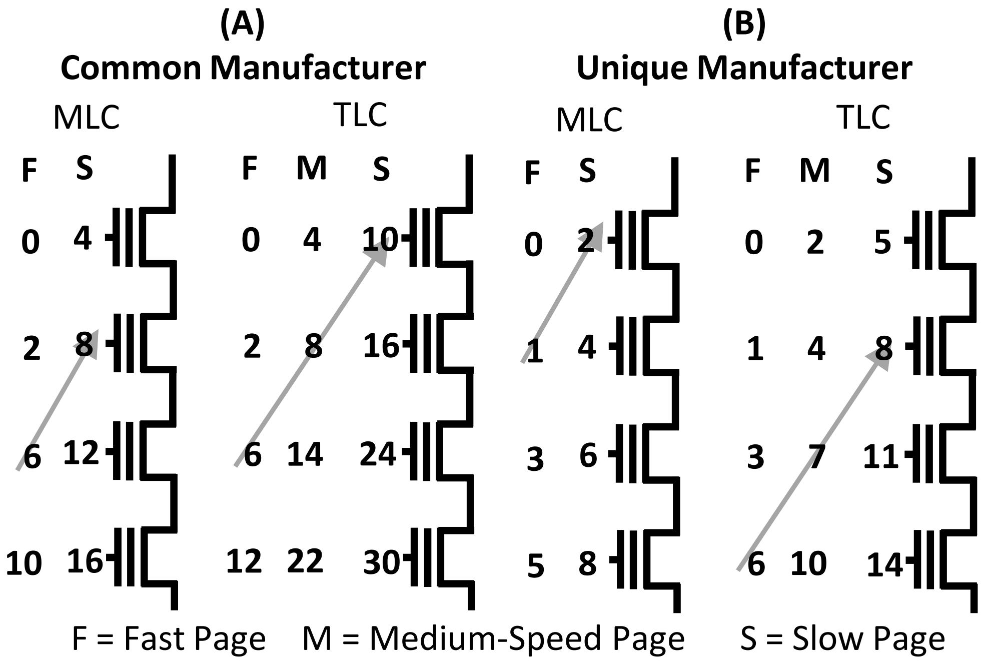


Figure 7.3. Memory Cell Anatomy Fast pages consist of each memory element's first-written bit. In-order programming causes the final bit of a memory cell to be written after most programs to the surrounding cells.

between fast and slow. TLC devices cycle through the three latencies with pairs of pages as well. The unique manufacturer follows the single-page alternating patterns in Figure 7.2B.

Figure 7.3 shows how a single bit from each page maps to the chain of flash memory cells. The numbers correspond to the owning page's location within the block and are in columns corresponding to the speed to program the bit. Figure 7.3B shows the NAND chains from MLC and TLC parts made by the manufacturer with a unique pattern, while Figure 7.3A shows the even chains from the remaining manufacturers (the corresponding odd chain is similar).

Because of the in-order programming constraint, the final program of a cell occurs after most of the program operations to adjacent cells are complete. This reduces program disturb that is a major hindrance to enabling multi-bit technology [60]. Note that the blocks of most manufacturers alternate between page speeds in pairs because they separate pages into even and odd chains, while the unique manufacturer uses only one chain. Also, most of the variation in the latency of slow pages (indicated by the wide error bars in Figure 7.1) comes from the even chain being slower than the odd chain.

The techniques we develop in the following sections depend on the FTL knowing the layout of fast and slow pages within a block. Since the layout is consistent for a given part number and does not vary over time, it is sufficient for the manufacturer to detect this pattern using a single block and configure the FTL accordingly. An FTL could perform the measurement at initialization time by monitoring the programming time of pages in a block, reducing the cost of moving to a new type of flash chip in an existing SSD design. There is also a non-destructive technique for determining page type. Page read latencies exhibit the same variation pattern. Furthermore, because of the small number of possible patterns that are mentioned in the datasheets or derived empirically, only a few page reads are required.

Overall, as shown in Figure 7.1, the dramatic differences in page program latency provide a better opportunity to exploit diversity to improve SSD performance. The following sections describe our extensions to the baseline FTL which leverage these variations in program latency.

7.2 A Variation-Aware FTL

The data in Section 2.3 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 modify the baseline FTL in Mango (See Chapter 5) to exploit the variation in program time between the fast and slow pages by skipping slow pages for improved

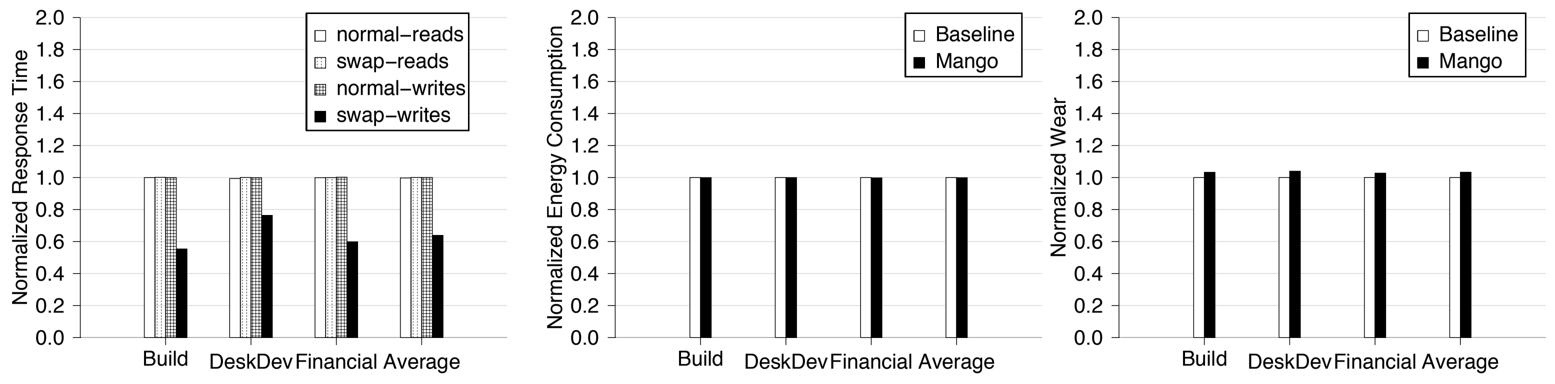


Figure 7.4. Mango Results for the Swap Scenario Mango is able to significantly increase responsiveness for swap requests while only marginally increasing energy consumption and increasing wear by only 3%.

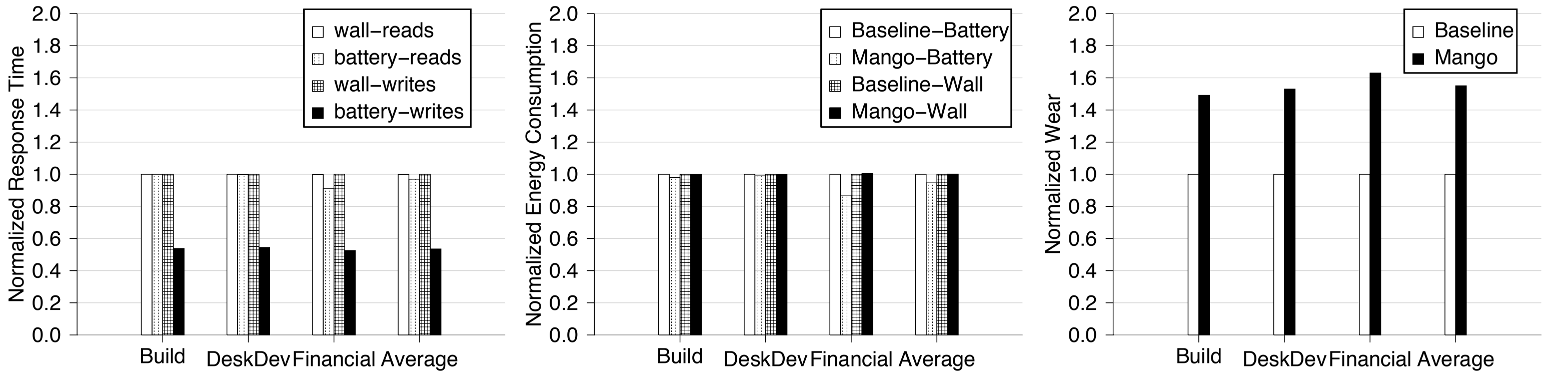


Figure 7.5. Mango Results for the Netbook Scenario Mango is able to significantly reduce the energy drain on the battery without significantly increasing energy while plugged in. It also realizes a slight performance increase, while increasing wear by 55%.

Table 7.1. Workloads for Evaluating Mango The traces cover a wide range of application types.

Name	Description	Mean Req. KB		%
		Read	Write	Reads
Build	Compile Linux kernel	3.7	4.0	12.14
Financial [4]	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

performance or power/energy efficiency for some operations.

We add a priority bit 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 the fast pages are also lower energy, high priority accesses are also low energy.

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 Chapter 5), 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 Table 7.1) 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 7.1. 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.

We implemented Mango in a flash memory simulator using the performance and power measurements from Chapter 2 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 Chapter 2, since the power and performance for the other MLC devices were comparable, we would expect similar performance and energy results.

Figure 7.4 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 7.5 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.

7.3 The Harey Tortoise: Leveraging Variability Without Adding Wear

In this section we describe our mechanisms for scheduling flash operations based on flash page performance variation without skipping over unwanted pages. Our scheduling algorithm improves performance without sacrificing capacity or endurance, providing the speed of the hare (high performance) *and* the endurance of the tortoise (increased capacity and reduced write amplification) We demonstrate how careful, variation-aware scheduling can improve performance under both bursty and sustained workloads. With both mechanisms, we show how increasing the number of write points on each chip increases the FTL's ability to leverage the variability in its flash array.

7.3.1 Many Write Points for More Flexibility

Making good scheduling decisions requires the scheduler to have multiple options available, and without multiple options, no scheduling policy can have much impact on performance. Since each write point is associated with a single block, and the FTL must write to pages in the block in order, a single write point offers limited options: The FTL can either write to the next page (which may not be the type of page it would prefer) or it can skip the page, writing to the page of its choice, but wasting space.

Our baseline FTL maintains one write point per chip, which can only provide multiple options under light load (and some chips are idle). Under heavy load the FTL's only choice is to schedule an access to the most recently idled chip. Even under light load, a large burst of write traffic will use up the fast pages available on each write point. Both of these scenarios force the FTL to choose between the two undesirable options described above.

To provide flexibility, we extend the baseline FTL with multiple write points per chip, ensuring that the FTL will have choices and can make wise scheduling decisions. In the following subsections, we demonstrate how increasing the number of write points in the system and on each chip increases the policies' ability to access its desired page type.

While additional write points provide the flexibility to access fast and slow pages on demand, their number and use constitute a trade-off with over-provisioned capacity and data placement policies the FTL designer wishes to incorporate. Because each write point requires an open block, when the FTL maintains too many write points the over-provisioned space becomes too fractured across open blocks. In particular, the number of blocks between the background and emergency thresholds (for the GC routines described in Section 4.2.2) provide a hard limit for the possible number of write points in our design. The FTL designer will also have to carefully weigh the value of placing data to potentially improve the efficiency of future GC with the effects of using a high or low latency page.

7.3.2 Handling Bursty Workloads

In this section, we present a policy called *Return to Fast* (RTF) that allows the FTL to service bursts of performance-critical operations exclusively with fast pages. The algorithm seamlessly provides nearly the speed of SLC while using all of the MLC pages.

We can apply the RTF policy in a number of situations. With an interface that passes information about the criticality of writes to the device, the system could schedule critical operations to fast pages. Such an interface could, for example, enable fast distributed locking protocols that require persistent writes for ordering via a log.

Even without changes to the interface, we can significantly enhance the performance of bursty workloads by treating user accesses as performance-critical and GC operations as non-critical. In this case, we use fast pages exclusively until we run out, and then return to our baseline policy. We focus on this application in this thesis.

RTF aims to service as many external writes as possible with fast pages. One approach is to skip over slow pages in order to move write points to the fast pages, but that would waste those skipped pages – reducing SSD capacity, invoking GC sooner, and increasing wear and potentially decreasing performance.

RTF avoids skipping pages by returning all write points to fast pages during the idle periods through GC writes. The FTL saves up a reserve of fast pages which it can spend on performance-critical operations. The number of write points in the system controls the size of reserve of fast pages.

The most common pattern of fast and slow pages provides up to two fast pages per write point. The FTL can fully exploit both pages in *Strongly RTF*, which ensures the write points reach the first of the pair of fast pages. The FTL can store an average of

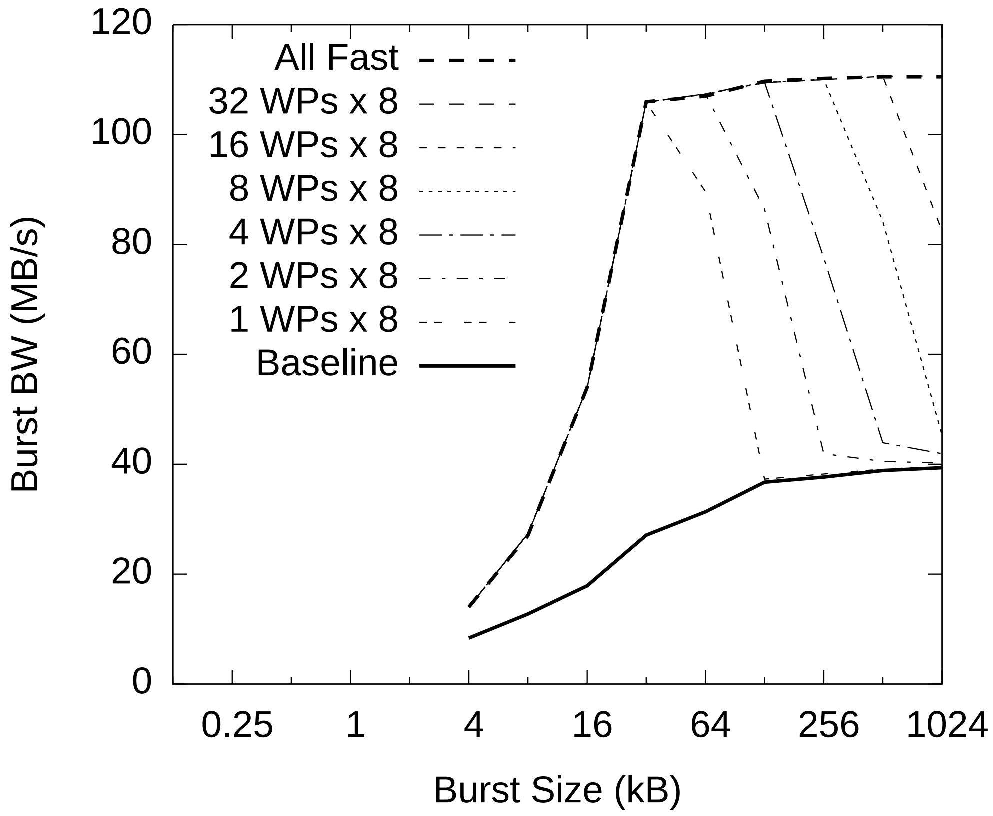


Figure 7.6. Performance of Weakly RTF The weakly RTF policy maintains performance comparable to using only fast pages for burst sizes up to the number of write points before dropping to the performance of using all page speeds.

1.5 writes per write point in *Weakly RTF*, which returns the write points to any fast page. Strongly RTF will give us the largest number of fast pages available after a large enough idle period.

We can further enhance the FTL with *preemptive GC*. During idle periods, the FTL continues to GC until each write point points to a fast page. This runs the risk of increased wear, when external writes or trims invalidate the pre-emptively moved data. However, simulation results show this is not a problem.

Increasing the number of write points in a system increases the performance of the bursts, even when the workload is a complex mix of reads, writes and potentially short idle times. In order for the FTL to direct an external write to a fast page, (1) there must be a write point already pointing at a fast page and (2) this write point must point to a chip which is not busy with another operation. Under a complex workload, the number of write points in the system is directly related to the likelihood of both of these conditions. The more write points there are, the more write points there will be pointing to fast pages. So, even with very little idle time we have increased the number of fast pages for the next burst.

A similar argument holds when you consider the contention over access to chips in the system. Imagine all but one of the chips in the array are blocked with operations. The single available chip is more likely to have a fast page available if there are more write points (and more possible pages available).

Evaluating RTF

We explore the potential of the RTF policy by studying its behavior under a synthetic workload of page-sized accesses to uniformly distributed LBAs, grouped into bursts. The gap between bursts is sufficient to complete all necessary GC and return all the write points to fast pages, when applicable. Each trace uses a different burst size from 4 kB to 4 MB (1 to 1024 pages) and writes a total of 16 MB of data.

Figure 7.6 shows the performance of the Weakly RTF policy for 1-32 write points per chip on an 8 chip array (x8). For burst sizes less than 32 kB, the array is under-used, but as the burst size reaches between one and two pages per chip the performance increases significantly for RTF and the All-Fast configuration (which, as noted in Section 4.2, uses only fast pages). The baseline remains low with a maximum performance of 39.4 MB/s because it uses both fast and slow pages.

At burst sizes greater than 32kB, we observe the positive effect of additional write points in enabling RTF. With one write point, the FTL can manage only short bursts at high speed. Increasing the number of write points per chip provides a larger reserve of fast pages from which to draw and lets the scheduler make better decisions. For weakly (strongly) RTF, the maximum burst size serviced at high speed is equal to (2x) the number of write points in the system times the page size.

7.3.3 Sustained Write-Intensive Workloads

Return to fast provides an effective tool for selective performance enhancement. However, under sustained write traffic, external operations must compete for resources with GC which eclipses the performance benefits of RTF.

In this section, we develop a rate matching technique that allocates fast and slow SSD resources among GC and external operations for the best performance during long periods of sustained load. We begin with a variability-informed analytical model of an FTL, its page scheduling policy, and its GC. The model shows that in most cases the intuitive choice for page variability will lead to performance losses while the counter-intuitive choice improves performance. Finally, we study the potential of the FTL operating with these parameters.

Analyzing FTL Behavior Under Load

In order to maintain the erased block pool during periods of sustained, heavy load, the FTL must match the rate at which it erases pages with its external write rate. The per-chip bandwidths for these two operations remains constant, so the FTL matches these rates by establishing the correct number of chips performing each of the two sets of operations. Equations 7.1 and 7.2 describe the two per-chip bandwidths. For Equation 7.2, we assume 20% over-provisioning and include a parameter (*pgsMvd*) for the number of page moves GC must perform on the average block (which is determined by the workload's locality).

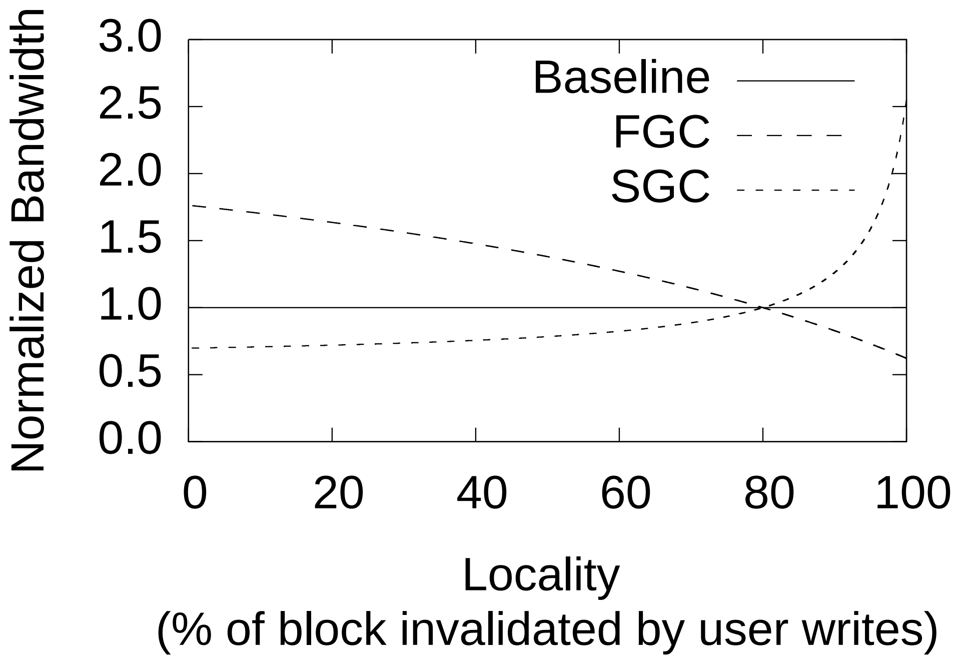


Figure 7.7. Design Space for Rate Matching Which configuration to use under heavy load depends on the workload's locality. If locality is low (less than 80% on this graph), GC must move lots of data and prioritize those writes to fast pages to improve overall performance.

$$ExternalWriteBW = \frac{pageSize}{wLat} \tag{7.1}$$

$$GC\_BW = \frac{0.2 * blockSize}{pgsMvd * (mvLat) + eLat} \tag{7.2}$$

With respect to write latency variability, we consider two choices. The FTL could use slow pages to service GC writes and fast pages to service user writes (SGC), or vice versa (FGC).

Figure 7.7 plots the SSD's bandwidth for these policies and a baseline, latency agnostic configuration over a range of workload localities. Our model assumes the FTL always has access to the preferred page speed without skipping pages. For the FGC configuration, for example, we determine the per-chip user write BW and the cleaning

BW using slow page write latency for Equation 7.1 and fast page write latency for Equation 7.2, respectively. The ratio of the two yields the correct ratio of chips to use for each operation. The chip counts are averaged over time, so they do not need to be integers. Ultimately these values yield the user-visible write bandwidth.

Without the analytical model, our initial choice was to accelerate external operations, corresponding to the SGC configuration. However, as Figure 7.7 shows, the highest performance configuration allocates fast pages to online GC instead (FGC).

Scenarios with average to low page locality will do best under FGC, because GC reclaims relatively few erased pages for many moves. SGC experiences a disadvantage because fast user writes and slow GC writes exacerbate the inherent slowness of GC. FGC, on the other hand, uses the speed of fast pages to help GC to keep pace with the user accesses. Because block erase is necessary, and such a heavy weight process, the FTL does best by completing it quickly.

The crossover point falls exactly at 80% locality because of the particular amount of over-provisioning in our array (20%). The analytical model frees 20% of the pages in a block for the average whole-block GC sequence. With 80% locality, the number of pages erased per block GC equals the number of pages moved, and so external write BW is the same as GC write bandwidth for all configurations. As locality decreases from this crossover point, GC requires more moves and higher-performing writes (FGC).

In order to study FGC and SGC, we make two changes to the baseline FTL. The first does not include knowledge of page variability and is simply to maintain the pool under sustained write traffic. To do this, we modify the operation selection policy. We calculate the ratio of per-chip GC bandwidth to per-chip external write bandwidth, called the *target ratio*. The FTL maintains a *chip use ratio* by monitoring the ratio of time spent on GC and external write operations for the recent history. The FTL then chooses the next operation by attempting to match the chip use ratio to the target ratio.

The second policy change accounts for page variability in the data placement policy by directing pages to match either the SGC or FGC configurations. We implement a page preference policy whereby given the choice between several locations to write, the FTL prefers to direct the previously chosen operation according to the SGC or FGC configuration.

The baseline for studying the FTL under sustained load includes the changes to the operation choice policy, but retains the original round robin baseline for the write point choice policy.

Evaluating FGC and SGC

To study rate matching with page preference under the complex constraints imposed by a real FTL, we apply a write-intensive synthetic load to our simulator. The workload consists of 5 s pulses of infinite load followed by 4 s of idleness. This cycles repeats 80 times, and the load consists purely of writes with evenly distributed LBAs.

Under such a load, all operations reach the Data Placement policy with only one idle chip in the flash array. Because each chip only has one write point, the page preference has no effect, and all operations have an equal probability of being written to fast or slow pages. Skipping pages is not a good option because its negative effect on performance overwhelms any advantage gained from using fast pages, due to the added GC.

Write points again provide the flexibility needed for the FTL to leverage the fast pages in the FTL. With multiple write points on each chip, when the operation arrives with only one idle chip from which to select, it still has multiple options for where it can write.

Figure 7.8 shows how, as the number of write points increases, the FTL can run operations on the desired pages type more frequently. With one write point, both SGC

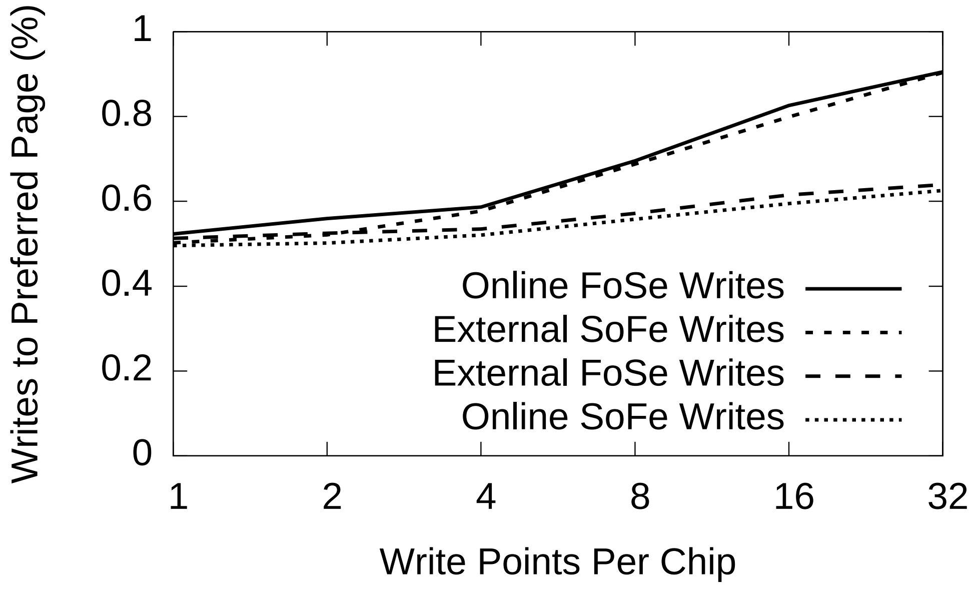


Figure 7.8. Page Preference Improvement Increasing the number of write points per chip increases the availability of the preferred page type when the SSD is under heavy load.

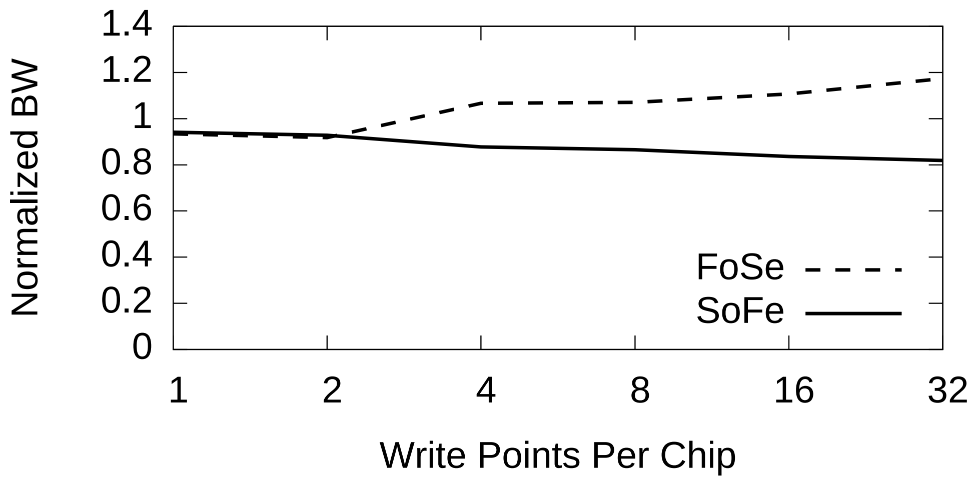


Figure 7.9. Sustained Performance Adding write points allows fast online garbage collection to improve SSD performance by 20%.

Table 7.2. Workload Statistics Characteristics of the burstitness of our tracefiles and the idle times between the bursts.

Trace Name	Min. Δ (Thresh.)	Avg. Burst Size (pgs)	Avg. Idle Time (s)	Description
Build	0.087 s	3.56	1.74	Compilation of the Linux 2.6 kernel.
Financial	18 ms	0.140	0.0620	Live OLTP trace for financial transactions.
WebIndex	48 μs	212	0.000564	Indexing of webpages using Hadoop.
Swap	150 ms	0.0645	0.0218	Virtual memory trace for desktop applications.
DeskDev	0.7 s	4.48	3.82	24 hour trace of a software development work station.

and FGC direct their operations to the two page speeds with equal probability. As the number of write points increases, a larger percentage of operations are scheduled to their preferred page speed. This is especially true when that preferred page speed is fast.

Figure 7.9 shows the performance resulting from the FTL accessing its preferred pages more often, normalized to the baseline of no page preference. As more write points allow the FTL to select its preferences, the performance of FGC improves while the performance of SGC declines.

These results verify that the optimal choice for page preference under heavy write load is to save fast pages for servicing online garbage-collecting moves (FGC), and that increasing the number of write points on each chip better enables the FTL to tap into that supply of fast pages.

7.3.4 Results

In this section we evaluate the effectiveness of our variability aware FTL policies – RTF, FGC and SGC – on a set of five benchmarks.

Simulator

Table 7.3 details the array's dimensions. We model two moderately-sized SSDs – one to quickly simulate results for our microbenchmarks and a larger configuration to run the workloads. We also simulate an *All Fast* configuration, which models a half-capacity SLC-speed array by (1) reducing block size from 128 to 64 pages and (2) using only the fast read and write latencies.

Our SSD manages the array of flash chips and presents a block-based interface. The controller in the SSD coordinates 4 or 8 channels that each connect 2 chips to the controller via a 400 MB/s bus. Larger SSD configurations are possible, but the configurations we choose provide similar performance trends with much shorter simulation

Table 7.3. SSD Configuration Architectural dimensions of the flash array and operation latencies to the flash chips. For speed, microbenchmark studies use fewer channels and fewer dies per channel. All-Fast configuration uses fewer pages per block at the lower read and write latencies.

Parameter	Value
Channels	4 or 8
Dies per channel	2 or 16
Blocks per chip	2048
Pages per block	64 or 128
Bytes per page	4096
Fast Page Read Latency	27 μs
Slow Page Read Latency	40 μs
Fast Page Write Latency	253 μs
Slow Page Write Latency	1359 μs
Erase Latency	2871 μs

times.

Workloads

Table 7.2 describes the five trace files we use to explore our proposed FTL enhancements. Their burst sizes span a range as do the idle time between each burst.

The conventional method of replaying traces does not accurately retain fixed computation time (seen by the SSD as idle time). This runs the risk of mixing the idle and active parts of the workload which could both (1) eat into the idle time needed for RTF and (2) lessen the load FGC and SGC are intended to accommodate.

We pre-process our trace files to alleviate these problems. Instead of each trace line indicating what time it arrives at the SSD, it indicates how much later than the previous trace line it arrives. Then, if the delta is below a particular per-trace threshold, we group that access in the same burst with the previous access by setting the delta to zero. In this way, we ensure the SSD experiences the full brunt of the burst without added idle time.

We assume that a large enough idle period (i.e. that greater than the threshold) indicates the program is executing calculations using the previous burst's data. We also assume that the amount of time before issuing its next burst will remain constant for a given processor architecture. We then enforce the delta time between each burst by issuing the first access of a given burst *delta* seconds after the previous burst completes (i.e. after the completion of the last access).

We set the delta threshold to be the average time between each trace line for a given file. Table 7.2 details the delta threshold for each trace file as well as the average size of the bursts and average amount of idle time between them.

Measuring the performance of an SSD running a trace file that includes idle time requires some care. To factor out the effect of idle time in the trace file, we divide the amount of data written in a given burst by the time it takes to complete that burst (this is the burst's write bandwidth). We then report the average of these bandwidths for each policy normalized to the baseline.

Return To Fast

Figure 7.10 shows the performance of the delta traces running under the Strongly RTF (sRTF) and weakly RTF (wRTF) policies with 1, 8 and 32 write points per chip. The All-Fast configuration shows a potential for 19% to 62% increase in write performance (34% on average) over the baseline and all traces realize at least a portion of these gains. On average, traces realize a 9% performance increase going from 1 to 32 write points per chip and no increase in performance for using strongly RTF rather than weakly RTF.

Financial works well with RTF – it contains a significant amount of idle time between bursts for recovery, and has very few reads which could block and stall the burst. Financial also has very few writes in each burst, so the SSD is able to realize the full potential of the fast pages with very few write points. For other workloads, added performance comes with more write points because a larger pool of fast pages increases

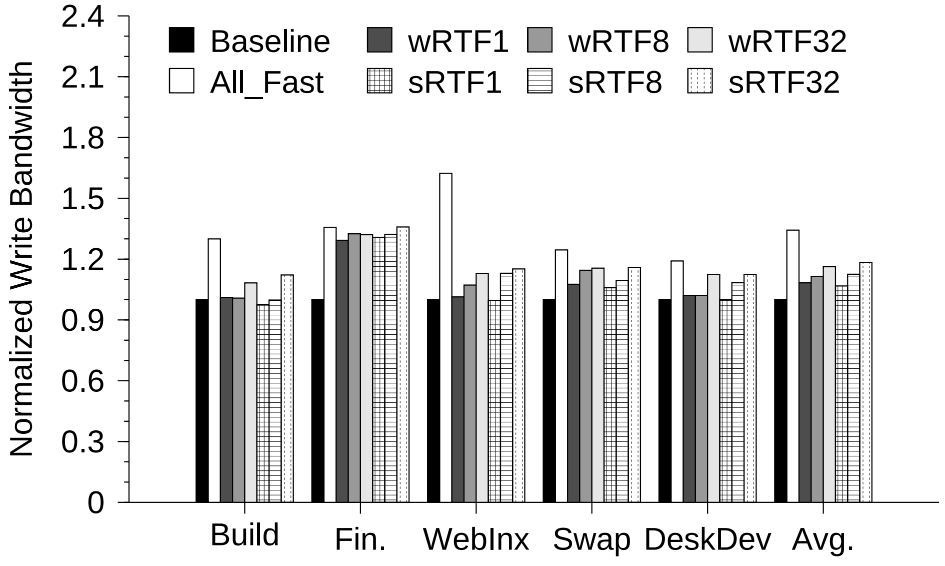


Figure 7.10. Performance of RTF More write points in the flash array increases the reserve of fast pages the FTL can build during idle periods, allowing the FTL to absorb larger burst with only fast pages.

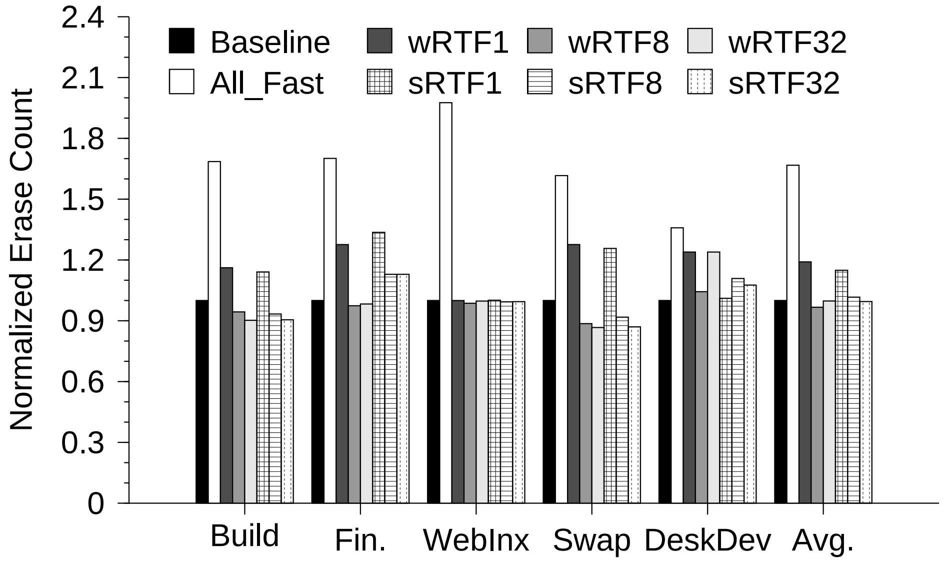


Figure 7.11. Wear of RTF While RTF improves performance, on average its wear is nearly that of the baseline.

the options for where to write, getting around the effect of blocking reads. All workloads on both strongly and weakly RTF achieve more than 24% of the All Fast configuration's gains and most see more than 64%.

While RTF consistently improves the write performance, it has negligible effect on the read performance. On average the RTF configurations gain less than 0.1% in read bandwidth.

Figure 7.11 shows the wear out experienced by our SSD under the different workloads and RTF configurations. Trying to achieve high performance by using only the fast pages significantly increases the wear – up to 2.0x, and 1.7x on average. However, if we instead fill the slow areas with garbage collected data we were planning on moving anyway, our wear increases by 5% relative to the baseline on average, and never more than 34%.

Rate Matching with FGC and SGC

Figure 7.12 shows the performance of the traces running on the FGC and SGC rate matching policies using 1, 8 and 32 write points per chip. The All-Fast configuration is able to realize much larger gains over the baseline, because the FTL makes use of all of the pages during external activity. Even so, the FGC configuration on most workloads achieves a significant portion of these gains while the more intuitive SGC configuration remains at baseline levels. DeskDev reaches the highest performance at 95% above baseline, and the average of all the traces except for WebIndex reaches 65% over baseline.

The spacial locality in the WebIndex's writes set this workload apart – in this case the intuitive choice of directing external operations to fast pages (SGC) provides better performance. WebIndex exhibits an average of 31% fewer moves per erase, placing it in the right-most region of Figure 7.7. The advantage of saving fast pages for online

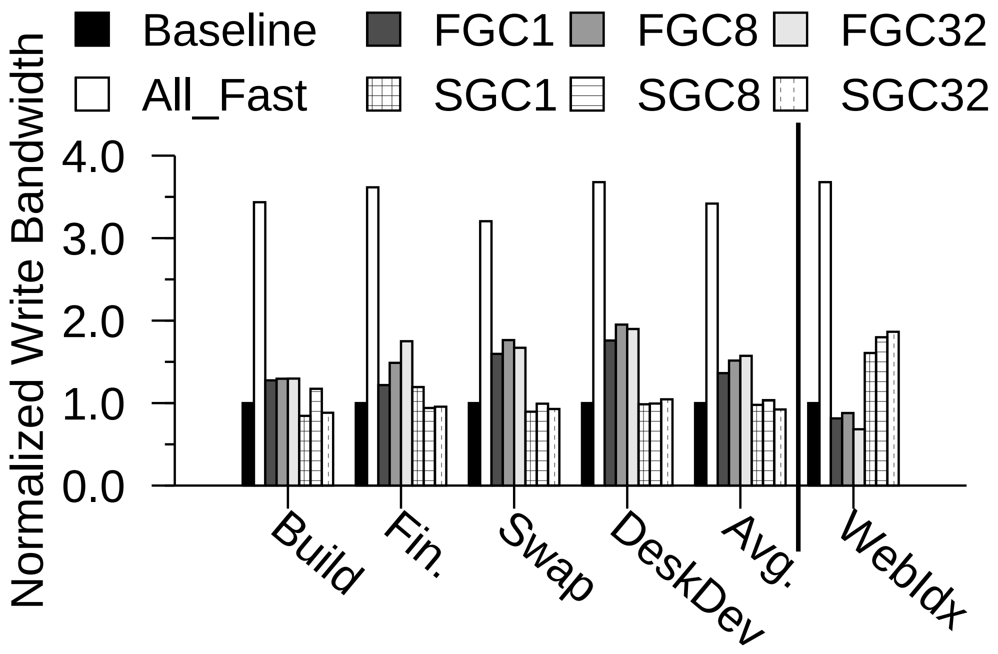


Figure 7.12. Performance of FGC and SGC The counter intuitive choice of servicing online operations with fast pages (FGC) improves the performance, while opposite (SGC) causes performance to decline, when spacial locality is low

operations in FGC is a result of completing GC as fast as possible to match the rate of external writes. However, when the access stream exhibits good spacial locality, the act of writing external operations invalidates pages on a small set of blocks, accelerating GC.

Increasing the number of write points on each chip allows each configuration to approach the predicted behavior. SGC almost always performs on par or worse than the baseline, often declining from baseline as the number of write points decreases. The opposite trend holds for FGC, frequently beginning with a performance better than baseline and increasing as the number of write points increases. This makes sense because increasing the number of write points increases the impact of each policy. Since SGC hurts performance, adding write points makes performance worse.

While FGC and SGC produce performance gains and losses, respectively, in most cases they both perform a number of erases on-par with the baseline (Figure 7.13).

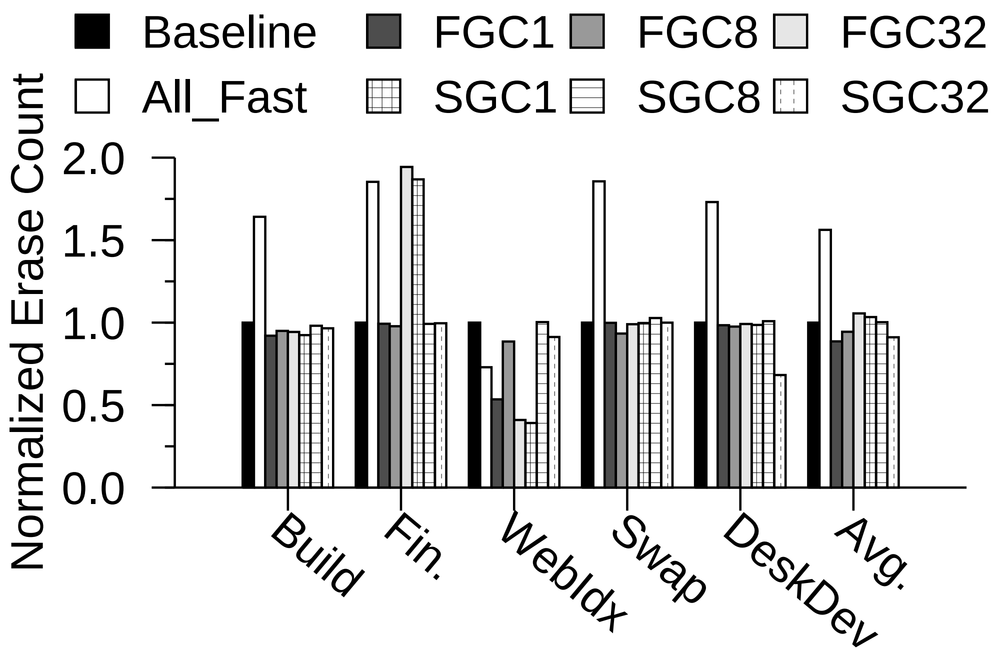


Figure 7.13. Wear of FGC and SGC Using page variability intelligently during heavy load does not effect device wear out in most cases.

Excluding WebIndex, the erase count declines by as much as 32% for the SGC-32 configuration on DeskDev, and increases by no more than 2% (Excluding Financial). On average, FGC and SGC experience a 3% decline in wear while the All-Fast configuration is 56% more wear compared to the baseline.

7.3.5 Application

Although we propose distinct mechanisms for bursts and heavy load, we discuss their coordination with other policies in the system to address a variety of workloads with mixed access patterns. This section describes how this could be done either through coordination with the operating system or by further enhancing the FTL.

OS Support

Coordination with the operating system constitutes one avenue of leveraging the Harey Tortoise techniques. The OS could provide hints with the accesses made to the

SSD. For example, the FTL could use RTF to service latency-critical accesses (marked as high priority), providing the functionality of the variability aware FTL in Section 7.2 without the added wear. Alternately, the OS could signal a course-grained switch between workload style when, for example, a server transitions between workloads or activities that change between peak and off-peak periods. An enhanced interface, such as NVMe would facilitate these implementations.

Dynamic FTL

Without hints from the OS, the FTL could combine the Harey Tortoise's policies to accommodate mixed workloads. It would adjust as a burst of accesses of unknown length progresses – employing RTF early in the burst before transitioning to RM techniques as the "burst" lengthens to a sustained load. This technique would result in RTF accommodating small bursts while the FTL treats long bursts with RM techniques.

For long bursts and sustained load, the FTL would step through several phases combining our techniques proposed in this work. For such a policy, GC during idle period should employ RTF to return as many write points as possible to fast pages. Then, when accesses arrive, the FTL would achieve maximum possible performance from using only fast pages under RTF, before gradually transitioning to RM policies.

During the transition period the FTL would (1) adjust the preference for fast or slow pages of the external and GC writes and (2) tailor the use and cleaning rates to use up the over-provisioned space and create a graceful degradation of performance. The latter could be achieved by relating the target and chip time ratios by some factor which dynamically adjusts to one.

Finally, when the pool of erased blocks reaches a sustainable minimum, the FTL would work exclusively with the RM policies until an idle period allows for additional GC. In this way, the FTL would provide high performance to small bursts and gradually ramp down to a maximum, sustainable performance.

The inversion of preference (for RM) with good write locality suggests another dimension for exploring how to detect and adapt to the correct choice of page preference.

The FTL designer could extend this procedure by leveraging predictive models which anticipate the size of bursts and length of idle periods. With this information the FTL could optimally set the parameters of the phased algorithm just described. For example, with knowledge of the current burst's length, the FTL could use the over-provisioned space such that the FTL reaches the emergency threshold exactly at the end of the burst, thus providing the maximum possible performance.

7.4 Related Works

There is a large body of flash-based storage research spurred on by the promise of high performance, low energy, and the limitations imposed by its idiosyncrasies. The research most closely related to our work falls in four categories: Mode-switching Flash, FTL algorithms, SSD interleaving, and write buffers. All of these topics try to improve the performance, endurance and/or reliability of the SSD, but do not leverage or address the variability inherent in MLC flash. The final section of related work discusses the emerging research that embraces flash page variability.

Mode-Switching Flash: Researchers have proposed changing the cell bit density [47], and SSD vendors have implemented these changes [68, 55] to improve reliability, endurance, and performance. Switching between MLC mode and SLC mode does have the drawback of sacrificing half of the system capacity. In our work, by leveraging write latency asymmetry across the pages, we are able to approximate the performance of SLC without sacrificing device capacity, the best of both worlds. Furthermore, because we use all the pages in the block by not throwing away the slow pages, we reduce the number of erase cycles, improving overall system endurance and reliability.

FTL Algorithms: There is a large body of work focused on FTL optimizations to improve SSD performance, endurance and reduce memory overhead based on access pattern or application behavior. By using an adaptive page- and block-level addressing mapping scheme, KAST [25], ROSE [24] and WAFTL [78] are able to improve performance, reduce garbage collection overhead and reduce FTL address mapping table size. DFTL [36] goes one step further by caching a portion of the page-level address mapping table for reduced size and fast translation. MNFTL [66] reduces the number of valid page copies for garbage collection, explicitly targeting MLC flash. Finally, CAFTL [22], removes unnecessary duplicate writes and increases the lifespan of the SSD. While some of these FTLs address workload variability, none address the variability in the underlying MLC flash.

SSD Interleaving: Intra-SSD parallelism has been explored by many groups [11, 19, 18, 61, 82, 21, 70, 14]. By not only issuing operations in parallel at the package, die, and plane level, others have also shown that rescheduling operations can improve performance [82]. Our work dives deeper into parallel data placement by providing multiple write points for fast pages within the plane that can adsorb burst and sustain high write performance, on par with SLC devices.

Write Buffers: Historically, buffers have been used in HDD to improve read and write performance. Likewise, write buffers have been shown to improve random write performance in SSDs [49]. These write buffers are also sufficient for handling small burst sizes. More recently, research has shown that per package queues and operation reordering provide more opportunities for parallel operations and further improve performance over LRU-based write buffer mechanisms [70]. Write points can be used in conjunction with write buffers, providing the FTL with more flexibility in data placement, in light of the performance asymmetries that exist in MLC flash.

7.5 Conclusion

The design and mechanisms we present enable FTL designers to use modern, high density (MLC) flash while still tapping in to the favorable performance and energy characteristics of high-quality, low-density (SLC) technology. We present a simple design which performs OS-marked writes to MLC at SLC speeds and energies which can improve swap latency by 40% but adds a moderate amount of wear. We also present an FTL which is capable of absorbing large bursts at low latency and mechanisms to appropriately direct accesses under sustained write load based on the application's locality. Without adding wear, the second system improves burst performance by 36% and steady-state performance improvements of up to 95%.

Acknowledgements

This chapter contains material from "Characterizing Flash Memory: Anomalies, Observations, and Applications" by Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Eitan Yaakobi, Steven Swanson, Paul H. Siegel, and Jack K. Wolf, which appears in *Micro'09: the 42nd Annual International Symposium on Microarchitecture*. The dissertation author was the first investigator and author of this paper. This material is copyright ©2009 by the Institute of Electrical and Electronics Engineers (IEEE). If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

This chapter contains material from "The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs" by Laura M. Grupp, John D. Davis and Steven Swanson, which will appear in *USENIX'13: USENIX Annual Technical Conference*. The

dissertation author was the first investigator and author of this paper. This material is copyright ©2013 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part of all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1(212)869-0481, or permissions@acm.org.

Chapter 8

Future Work: The Symbiotic Storage Stack

Throughout her work on this dissertation, the author has observed the convergence of several aspects of flash-based designs, poised to inspire big changes in storage:

Powerful Device-Level Trade-Offs

This thesis demonstrates the opportunities enabled by a detailed understanding of flash chips especially when coupled to application-specific behaviors. With information about flash's reprogrammability we enable a wear-reducing and energy-saving encoding scheme as well as a mechanism for secure data removal. Armed with page-level latency patterns we return the favorable performance of yester-chip to modern devices. Others' work has exploited trade-offs between density and lifetime [29], and lifetime and data retention [76]. The community is building its arsenal of device-level trade-offs, waiting to be paired with applications' needs.

Exhaustion with Heavy-Weight FTLs

Generic, high performance SSDs require page-level FTL and high power to drive and service a fast interconnect. Many are growing weary of these considerable overheads and are looking to the remaining two factors as both contributors to the problem and avenues for a solution.

Duplicate and Contradictory Algorithms

The major component to FTLs' performance and DRAM overheads result from the logical to physical address map (see Chapter 4). While this address remapping allows FTLs to improve performance and implement wear leveling, remapping is also often done in the operating system to improve performance for HDDs. A system such as Nameless Writes [13, 83] which integrates the SSD and OS algorithms alleviates these problems.

Divination of Applications' Needs

Many of the hybrid designs (described in Chapter 4) improve their efficiency by predicting the behavior of the workloads using the SSD. Direct coordination between the application or OS generating the accesses would provide more accurate information with possibly less complexity.

These four factors combine with market forces to produce application-specific SSDs. Proponents of this trend call these "solutions" whereas others worry about a "fracturing of the market." Before somebody wins the standardization battle, we have the opportunity to redefine the storage stack to expose device-level trade-offs and distribute the decisions regarding these trade-offs to the appropriate level of the stack. The remainder of this chapter discusses the aspects of such a redesign.

8.1 A Call for Transparency

The goal is to provide a framework on which storage solutions can evolve for decades to come. We begin with a taxonomy of the trade-offs we have witnessed so far in NAND flash memory coupled with a discussion of the parts of the storage stack which may benefit from controlling these trade-offs.

First, the aspects of NAND flash memory:

- Program/Erase (PE) Cycling (i.e. Lifetime)
- Data Retention Time
- Raw Bit Error Rate (RBER)
- Read and Write Latency
- Erase Latency
- Power Consumption
- Bandwidth
- Density

Currently, the system may trade-off between many combinations of these properties in the FTL. It does so either through easily controllable mechanisms such as physical address (in the case of the fast/slow page property), or through the use of a set of chip-level commands called "test mode" or "modes." Though some SSD manufacturers have access to varying levels of test mode, the full set of commands is rarely exposed to non-chip-manufacturers and none of the commands are public. Several years of negotiating has yielded none to our research lab.

Assuming trade-secret hurdles may be overcome, the remaining questions determine which parts of the storage stack are best suited to making the trade-off. While chip-level designers can apply universal improvements (such as lower power all the time) more complex decisions (such as whether to favor performance or reliability) require application specific information. The storage system may make these decisions at any one or more of the levels whether it be application, file system, host OS, driver, FTL or flash chip.

8.2 Application Realms

We describe several frameworks – one for each broad category of applications. We choose this division because applications in each realm share similar structures, balance of resources and development styles. Our goals are to allow for flexibility in designing for unforeseen applications and for yet-to-be-discovered trade-offs in the technology.

8.2.1 Server Systems

The defining characteristics of servers systems include (1) rich algorithms already built into the storage stack and (2) the ability to switch between a variety of distinctly different applications. Therefore, we choose a design which expands the interface to the SSD. Interfaces such as NVMe (or *NVMHCI*) [2] demonstrate movement toward a richer API which could pass information to the algorithms in the SSD indicating information with each access or marking the division between distinct modes.

There are several styles of trade-off specification the system may use:

- Specify each constraint individually (e.g. $RBER < 5e^{-10}$)
- Specify preferences (prefer high performance, but not critical)
- Specify balance of preferences (favor good reliability over good performance)
- Specify a set of preferences in system-level terms (”caching” means high performance, but relax everything else)

By combine these different styles from several different levels of the storage stack, the system can leverage a spectrum of possible solutions.

8.2.2 Embedded, Special Purpose Designs

In this design space, developers come with a variety of expertise and an interest in rapidly developing an efficient solution for a singular purpose. A symbiotic flash solution in this space must be easy to work with and each design will likely be used for the lifetime of the device.

To accommodate these needs, we foresee a core flash management layer which efficiently implements the basics of flash management while exposing trade-offs with a simple set of commands.

Designers of embedded systems currently have two options. The first is a simple block-based map 4.1.1, which is relatively slow, but they use simple algorithms and can function in an environment with minimal DRAM.

The second option, called managed flash [56], provides NAND flash plus an integrated controller to mitigate its inherent limitations. This solution suggests a new paradigm but, as others have noted [63], prevents the designer from tapping in to the features available from raw flash (such as cached reads and multi-plane operations). The trade-offs suggested in Section 8.1 are certainly also unavailable.

While the second option is limited, a richer version of managed NAND beckons. Enabled by up-and-coming non-volatile technologies, new algorithms could make it easy for designers to hard-code a wide variety of trade-offs to promote lower power for ultra-long battery life, high capacity for data-logging sensor nodes or high PE cycling to enable rugged devices that will last for decades.

8.3 Post-Flash Technologies

Flash is only the first in a series of new non-volatile memory (NVM) technologies, such as Phase Change Memory (PCM, PRAM), Spin Torque Transfer (STT, STRAM) and the Memristor. The author predicts that several of these technologies will co-exist and even complement each other in future designs. Furthermore, these technologies exhibit permanent failures, wearout and read/write asymmetries similar to flash and may also exhibit other device-level trade-offs which the symbiotic storage stack. PCM, although primarily considered byte addressable, already exhibits a write latency benefit from a preceding erase operation [7] and some researchers are predicting the fast/slow page effect present in multi-bit flash will extend to multi-bit PCM [67].

Chapter 9

Conclusion

At the foundation of this thesis, we present empirical information detailing the behavior of flash chips. We characterize a wide variety of devices from six manufacturers and technology nodes spanning from 72nm to 25nm. The results we gather exhibit wide variations within a block, as well as between blocks and over time in terms of power consumption, latency and error rates. Our data also show that the values manufacturers provide in publicly available datasheets often tell only part of the story, and that actual performance can be significantly better or worse.

We leverage these patterns and behaviors throughout the work in a number of ways. First, we present the trends in flash's behavior over time. Then, we present a case study in FTL design and detail several device-informed improvements to the FTL. Finally, we mitigate the negative trends in flash's developing performance by coordinating variations in application and device behavior. Ultimately, we hope this chip-level information and FTL-level techniques inform efforts throughout the academic and industrial community as storage system designs evolve.

The technology trends we have described put SSDs in an unusual position for a cutting edge technology: SSDs will continue to improve by some metrics (notably density and cost per bit), but everything else about them is poised to get worse. This makes the future of SSDs cloudy: while the growing capacity of SSDs and high IOP rates will make them attractive in many applications, the reduction in performance that is necessary to increase capacity while keeping costs in check may make it difficult for SSDs to scale as a viable technology for some applications.

Our case study in FTL design presents a storage system for Gordon, a flash memory-based cluster architecture for large-scale data-intensive applications. We describe a flash-based system that is carefully tuned for data-intensive applications. Gordon systems combine this storage array with a conventional low-power processor and programming abstractions for large-scale distributed programming. The result is a highly-efficient, high-performance, highly-parallel computing system that is easy to use.

Compared to disk-based systems, Gordon systems are 1.5x faster and deliver 2.5x more performance per watt. As flash performance improves, these performance gains will only increase. Gordon demonstrates that flash affords the opportunity to re-engineer many aspects of system design, and, therefore, enables a new class of computing systems.

While our characterization results have inspired many device-informed changes to the FTL, we present two for improvements in lifetime, energy savings and security. 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 measure enabled us to significantly extend the effective lifetimes of chips, and enable algorithms for efficient single-file erasure.

Finally, we developed two FTL designs to significantly improve the latency of accesses to SSDs by leveraging systematic variability in flash memory. The first provides a simple design which enabled us to significantly decrease latency for critical IO requests and the second provide the speed of the hare (SLC) with the capacity of the tortoise (MLC). These designs are integral to achieving favorable performance as flash memory continues to scale.

With society's ever-increasing demand to store and process big data, it has never been more important to fully leverage this decade's dominant new storage technology. This thesis provides the missing link by characterizing a wide variety of NAND flash chips, and demonstrating how a symbiotic coordination of application and device variations give storage system designers the tools needed to leverage modern flash-based storage systems.

Bibliography

[1] DRAMeXchange. http://www.dramexchange.com/.

[2] NVM Express: The Optimized PCI Express® SSD Interface. http://www.nvmexpress.org/.

[3] Open NAND Flash Interface Specification 3.0. http://onfi.org/specifications.

[4] UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[5] International Technology Roadmap for Semiconductors: Emerging Research Devices, 2007.

[6] International Technology Roadmap for Semiconductors: Emerging Research Devices, 2010.

[7] Conversation with Intel, 2013.

[8] Micron, 2013. http://www.micron.com.

[9] Open NAND Flash Interface Specification 2.0, February 2008. http://www.onfi.org/specifications.

[10] Open NAND Flash Interface Specification 3.1, September 2012. http://www.onfi.org/specifications.

[11] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, 2008.

[12] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark S. Manasse, and Rina Panigrahy. Design Tradeoffs for SSD Performance. In *USENIX Technical Conference*, pages 57–70, 2008.

[13] Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Vijayan Prabhakaran. Removing the Costs of Indirection in Flash-based SSDs with Nameless Writes. HotStorage. USENIX Association, 2010.

[14] Shi Bai and Xue-Liang Liao. A Parallel Flash Translation Layer Based on Page Group-Block Hybrid-Mapping Method. *Consumer Electronics, IEEE Transactions on*, may 2012.

[15] Mary Baker, Satoshi Asami, Etienne Deprit, John Ouseterhout, and Margo 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.

[16] A. Ban. Flash File System Optimized for Page-Mode Flash Technologies. *US patent 5,937,425*, Issued August 10, 1999 Filed October 16, 1997.

[17] Andrew Birrell, Michael Isard, Chuck Thacker, and Ted Wobber. A Design for High-Performance Flash Disks. Technical Report MSR-TR-2005-176, Microsoft Research, December 2005.

[18] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: Using Flash Memory to Build Fast, Power-Efficient Clusters for Data-Intensive Applications. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 217–228, New York, NY, USA, 2009. ACM.

[19] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: An Improved Architecture for Data-Intensive Applications. *IEEE Micro Top Picks*, 30(1):121–130, January 2010.

[20] Li-Pin 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.

[21] Feng Chen, Rubao Lee, and Xiaodong Zhang. Essential Roles of Exploiting Internal Parallelism of Flash Memory Based Solid State Drives in High-Speed Data Processing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011.

[22] Feng Chen, Tian Luo, and Xiaodong Zhang. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives. In *USENIX Conference on File and Storage Technologies*, pages 77–90, 2011.

[23] M.-L. Chiang and R.-C. Chang. Cleaning Policies in Mobile Computers Using Flash Memory. *J. Syst. Softw.*, 48(3):213–231, 1999.

[24] Mong-Ling Chiao and Da-Wei Chang. ROSE: A Novel Flash Translation Layer for NAND Flash Memory Based on Hybrid Address Translation. *IEEE Transactions on Computers*, 60:753–766, 2011.

[25] Hyunjin Cho, Dongkun Shin, and Young Ik Eom. KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems. pages 507 –512, apr. 2009.

[26] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. A Survey of Flash Translation Layer. *J. Syst. Archit.*, 55(5-6):332–343, 2009.

[27] Denali. http://www.denali.com/wordpress/index.p-hp/dmr/2009/07/16/nand-forward-prices-rate-of-decline-will.

[28] G. Naso et al. A 128Gb 3b/cell NAND Flash Design Using 20nm Planar-Cell Technology. In *Solid-State Circuits Conference*. IEEE, 2013.

[29] H-T. Lue et. al. Radically Extending The Cycling Endurance of Flash Memory (to $> 100M$ Cycles) by Using Built-In Thermal Annealing to Self-Heal the Stress-Induced Damage. 2012.

[30] S. Chang et al. A 48nm 32Gb 8-Level NAND Flash Memory with 5.5MB/s Program Throughput. In *Solid-State Circuits Conference*. IEEE, 2009.

[31] T. Futatsuyama et al. A 113mm^2 32Gb 3b/cell NAND Flash Memory. In *Solid-State Circuits Conference*. IEEE, 2009.

[32] Y. Fukuzumi, Y. Matsuoka, M. Kito, M. Kido, M. Sato, H. Tanaka, Y. Nagata, Y. Iwata, H. Aochi, and A. Nitayama. Optimal Integration and Characteristics of Vertical Array Devices for Ultra-High Density, Bit-Cost Scalable Flash Memory. *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 449–452, 10-12 Dec. 2007.

[33] R. Gabrys, E. Yaakobi, L. Grupp, S. Swanson, and L. Dolecek. Tackling Intracell Variability in TLC Flash Through Tensor Product Codes. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1000–1004, 2012.

[34] Eran Gal and Sivan Toledo. Algorithms and Data Structures for Flash Memories. *ACM Comput. Surv.*, 37:138–163, June 2005.

[35] Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 24–33, New York, NY, USA, 2009. ACM.

[36] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address

Mappings. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 229–240, New York, NY, USA, 2009. ACM.

[37] Kin-Chu Ho, Po-Chao Fang, Hsiang-Pang Li, C.-Y.M. Wang, and Hsie-Chia Chang. A 45nm 6b/cell Charge-Trapping Flash Memory Using LDPC-Based ECC and Drift-Immune Soft-Sensing Engine. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 222–223, 2013.

[38] Samsung Semiconductor Incorporated. K9F8G08UXM Flash Memory datasheet.

[39] Intel. Intel System Controller Hub Datasheet, 2008. http://download.intel.com/design/chipsets/embedded/datashts/319537.pdf.

[40] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel 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.

[41] Dawoon Jung, Yoon-Hee Chae, Heeseung Jo, Jin-Soo Kim, and Joonwon 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.

[42] Dawoon Jung, Jeong-UK Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. Superblock FTL: A Superblock-Based Flash Translation Layer With a Hybrid Address Translation Scheme. *ACM Trans. Embed. Comput. Syst.*, 9(4):1–41, 2010.

[43] Soon-Moon Jung, Jaehoon Jang, Wonseok Cho, Hoosung Cho, Jaehun Jeong, Youngchul Chang, Jonghyuk Kim, Youngseop Rah, Yangsoo Son, Junbeom Park, Min-Sung Song, Kyoung-Hon Kim, Jin-Soo Lim, and Kinam Kim. Three Dimensionally Stacked NAND Flash Memory Technology Using Stacking Single Crystal Si Layers on ILD and TANOS Structure for Beyond 30nm Node. *Electron Devices Meeting, 2006. IEDM '06. International*, pages 1–4, 11-13 Dec. 2006.

[44] Tae-Sung Jung, Young-Joon Choi, Kang-Deog Suh, Byung-Hoon Suh, Jin-Ki Kim, Young-Ho Lim, Yong-Nam Koh, Jong-Wook Park, Ki-Jong Lee, Jung-Hoon Park, Kee-Tae Park, Jang-Rae Kim, Jeong-Hyong Lee, and Hyung-Kyu Lim. A 3.3 v 128 mb multi-level nand flash memory for mass storage applications. In *Solid-State Circuits Conference, 1996. Digest of Technical Papers. 42nd ISSCC., 1996 IEEE International*, pages 32 –33, 412, feb 1996.

[45] Tae-Sung Jung, Young-Joon Choi, Kang-Deog Suh, Byung-Hoon Suh, Jin-Ki Kim, Young-Ho Lim, Yong-Nam Koh, Jong-Wook Park, Ki-Jong Lee, Jung-Hoon Park,

Kee-Tae Park, Jhang-Rae Kim, Jeong-Hyong Yi, and Hyung-Kyu Lim. A 117-mm2 3.3-v only 128-mb multilevel nand flash memory for mass storage applications. *Solid-State Circuits, IEEE Journal of*, 31(11):1575 –1583, nov 1996.

[46] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. A Superblock-Based Flash Translation Layer for NAND Flash Memory. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 161–170, New York, NY, USA, 2006. ACM.

[47] Taeho Kgil, David Roberts, and Trevor 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.

[48] Han-Joon Kim and Sang-Goo Lee. A New Flash Memory Management for Flash Storage System. pages 284 –289, 1999.

[49] Hyojun Kim and Seongjun Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, 2008.

[50] Jesung Kim, Jong Min Kim, S.H. Noh, Sang Lyul Min, and Yookun Cho. A Space-Efficient Flash Translation Layer for CompactFlash Systems. *Consumer Electronics, IEEE Transactions on*, 48(2):366 –375, May 2002.

[51] Donghwa Kwak, Jaekwan Park, Keonsoo Kim, Yongsik Yim, Soojin Ahn, Yoonmoon Park, Jinho Kim, Woncheol Jeong, Jooyoung Kim, Mincheol Park, Byungkwan Yoo, Sangbin Song, Hyunsuk Kim, Jaehwang Sim, Sunghyun Kwon, Byungjoon Hwang, Hyung kyu Park, Sunghoon Kim, Yunkyoung Lee, Hwagyung Shin, Namsoo Yim, Kwangseok Lee, Minjung Kim, Youngho Lee, Jangho Park, Sangyong Park, Jaesuk Jung, and Kinam Kim. Integration Technology of 30nm Generation Multi-Level NAND Flash for 64Gb NAND Flash Memory. *VLSI Technology, 2007 IEEE Symposium on*, pages 12–13, 12-14 June 2007.

[52] Jaeheung Lee, Junyoung Heo, Yookun Cho, Jiman Hong, and Sung Y. Shin. Secure Deletion for NAND Flash File System. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1710–1714, New York, NY, USA, 2008. ACM.

[53] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.

[54] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. *SIGOPS OSR*, 42(6):36–42, October 2008.

[55] G.G. Marotta, A. Macerola, A. D'Alessandro, A. Torsi, C. Cerafogli, C. Lattaro, C. Musilli, D. Rivers, E. Sirizotti, F. Paolini, G. Imondi, G. Naso, G. Santin, L. Botticchio, L. De Santis, L. Pilolli, M.L. Gallese, M. Incarnati, M. Tiburzi, P. Conenna, S. Perugini, V. Moschiano, W. Di Francesco, M. Goldman, C. Haid, D. Di Cicco, D. Orlandi, F. Rori, M. Rossini, T. Vali, R. Ghodsi, and F. Roohparvar. A 3bit/cell 32Gb NAND Flash Memory at 34nm With 6MB/s Program Throughput and With Dynamic 2b/Cell Blocks Configuration Mode for a Program Throughput Increase up to 13MB/s. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 444 –445, Feb 2010.

[56] Micron. Managed NAND, 2013. http://www.micron.com/products/managed-nand.

[57] Vidyabhushan Mohan, Travor Bunker, Laura Grupp, Sudhanva Gurumurthi, Stan R. Mircea, and Swanson Steven. Modeling Power Consumption of NAND Flash Memories using FlashPower. In *Transactions on Computer Aided Design of Integrated Circuits and Systems*, 2013.

[58] Vidyabhushan Mohan, Sudhanva Gurumurthi, and Mircea R. Stan. FlashPower: A Detailed Power Model for NAND Flash Memory. In *Design, Automation, and Test in Europe*, pages 502–507, 2010.

[59] Chanik Park, Wonmoon Cheon, Jeonguk Kang, Kangho Roh, Wonhee Cho, and Jin-Soo Kim. A Reconfigurable FTL (Flash Translation Layer) Architecture For NAND Flash-Based Applications. *ACM Trans. Embed. Comput. Syst.*, 7(4):1–23, 2008.

[60] Ki-Tae Park, Myounggon Kang, Doogon Kim, Soon-Wook Hwang, Byung Yong Choi, Yeong-Taek Lee, Changhyun Kim, and Kinam Kim. A Zeroing Cell-to-Cell Interference Page Architecture With Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories. *IEEE Journal of Solid-state Circuits*, 43:919–928, 2008.

[61] Sang-Hoon Park, Seung-Hwan Ha, Kwanhu Bang, and Eui-Young Chung. Design and Analysis of Flash Translation Layers for Multi-Channel NAND Flash-Based Storage Devices. *Consumer Electronics, IEEE Transactions on*, august 2009.

[62] Youngwoo Park, Jungdal Choi, Changseok Kang, Changhyun Lee, Yuchoel Shin, Bonghyn Choi, Juhung Kim, Sanghun Jeon, Jongsun Sel, Jintaek Park, Kihwan Choi, Taehwa Yoo, Jaesung Sim, and Kinam Kim. Highly Manufacturable 32Gb Multi-Level NAND Flash Memory with 0.0098 μm2 Cell Size Using TANOS(Si - Oxide - Al2O3 - TaN) Cell Technology. *Electron Devices Meeting, 2006. IEDM '06. International*, pages 1–4, 11-13 Dec. 2006.

[63] Michele Pike. Why Raw NAND Flash with Hardware-based ECC is the Way to Go, November 2012. http://blog.datalight.com/why-raw-nand-flash-with-hardware-based-ecc-is-the-way-to-go.

[64] Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Transactional flash. *USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[65] Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Kei S. Yu, G. Edward Suh, Edwin Kan, and Steven Swanson. Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations. In *Proceedings of the 4th international conference on Trust and trustworthy computing*, TRUST'11, pages 188–201, Berlin, Heidelberg, 2011. Springer-Verlag.

[66] Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. MNFTL: An Efficient Flash Translation Layer for MLC NAND Flash Memory Storage Systems. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, 2011.

[67] Moinuddin K. Qureshi, Michele Franceschini, and Luis Alfonso Lastras-Montand John P. Karidis. Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase Change Memories. In *International Symposium on Computer Architecture*, pages 153–162, 2010.

[68] Dave Raffo. FusionIO Builds SSD Bridge Between SLC, MLC, July 2009.

[69] R.L. Rivest and A. Shamir. How to Reuse a Write-Once Memory. *Information and control*, 55:1–19, December 1982.

[70] Xiaojun Ruan, Ziliang Zong, Mohammed I. Alghamdi, Yun Tian, Xunfei Jiang, and Xiao Qin. Improving Write Performance by Enhancing Internal Parallelism of Solid State Drives. In *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*, dec. 2012.

[71] Kang-Deog Suh, Byung-Hoon Suh, Young-Ho Lim, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Kwon, Byung-Soon Choi, Jin-Sun Yum, Jung-Hyuk Choi, Jang-Rae Kim, and Hyung-Kyu Lim. A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme. *Solid-State Circuits, IEEE Journal of*, 30(11):1149 –1156, nov 1995.

[72] K. Takeuchi, T. Tanaka, and T. Tanzawa. A Multipage Cell Architecture for High-Speed Programming Multilevel NAND Flash Memories. *Solid-State Circuits, IEEE Journal of*, 33(8):1228 –1238, aug 1998.

[73] C. Trinh, N. Shibata, T. Nakano, M. Ogawa, J. Sato, Y. Takeyama, K. Isobe, B. Le, F. Moogat, N. Mokhlesi, K. Kozakai, P. Hong, T. Kamei, K. Iwasa, J. Nakai, T. Shimizu, M. Honma, S. Sakai, T. Kawaai, S. Hoshi, J. Yuh, C. Hsu, T. Tseng, J. Li, J. Hu, M. Liu, S. Khalid, J. Chen, M. Watanabe, H. Lin, J. Yang, K. McKay, K. Nguyen, T. Pham, Y. Matsuda, K. Nakamura, K. Kanebako, S. Yoshikawa, W. Igarashi, A. Inoue, T. Takahashi, Y. Komatsu, C. Suzuki, K. Kanazawa, M. Higashitani, S. Lee, T. Murai, K. Nguyen, J. Lan, S. Huynh, M. Murin, M. Shlick,

M. Lasser, R. Cernea, M. Mofidi, K. Schuegraf, and K. Quader. A 5.6MB/s 64Gb 4b/Cell NAND Flash memory in 43nm CMOS. In *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 246 –247,247a, feb. 2009.

[74] Hung-Wei Tseng, Laura Grupp, and Steven Swanson. Understanding the Impact of Power Loss on Flash Memory. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 35–40, New York, NY, USA, 2011. ACM.

[75] Hung-Wei Tseng, Laura Grupp, and Steven Swanson. Underpowering NAND Flash: Profits and Perils. In *Proceedings of the 50th Design Automation Conference*, DAC '13, New York, NY, USA, 2013. ACM.

[76] Computer Weekly. Enterprise MLC: How Flash Vendors are Boosting MLC Write Endurance, June 2011. http://www.computerweekly.com/feature/Enterprise-MLC-How-flash-vendors-are-boosting-MLC-write-endurance.

[77] Michael Yung Chung Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably Erasing Data from Flash-Based Solid State Drives. In *USENIX Conference on File and Storage Technologies*, pages 105–117, 2011.

[78] Qingsong Wei, Bozhao Gong, Suraj Pathak, Bharadwaj Veeravalli, LingFang Zeng, and Kanzo Okada. WAFTL: A Workload Adaptive Flash Translation Layer with Data Partition. In *Symposium on Mass Storage Systems*, pages 1–12, 2011.

[79] S. Wells, R. N. Hasbun, and K. Robinson. Sector-Based Storage Device Emulator Having Variable-Sized Sector. *US patent 5,822,781*, Issued October 13, 1998 Filed November 1, 1993.

[80] David Woodhouse. JFFS2: The Journalling Flash File System, version 2. http://sources.redhat.com/jffs2/.

[81] Eitan Yaakobi, Jing Ma, Laura Grupp, Paul H. Siegel, Steven Swanson, and Jack K. Wolf. Error Characterization and Coding Schemes for Flash Memories. In *IEEE Globecom Workshops*, 2010.

[82] Seon yeong Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. Exploiting Internal Parallelism of Flash-based SSDs. *Computer Architecture Letters*, jan. 2010.

[83] Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. De-Indirection for Flash-Based SSDs with Nameless Writes. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 1–1, Berkeley, CA, USA, 2012. USENIX Association.