ULTRA-RELIABLE FLASH MEMORY SYSTEMS FOR EMBEDDED APPLICATIONS

A Dissertation Presented

by

Thomas McCormick

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

in the field of

Computer Engineering

Northeastern University

Boston, Massachusetts

March 2016

Abstract

Based on its ruggedness, solid-state flash memory has been accepted as the basis of code and data storage in embedded systems applications for several decades. In more recent years, widespread mainstream acceptance of flash memory in consumer and enterprise applications has created tremendous downward cost pressures on flash memory manufacturers. The flash memory manufacturers have responded to these pressures by compromising on parameters that are most critical for flash memory's continued suitability for code storage in embedded computer applications. In particular, data retention specifications have been decreased from ten years to as low as one year. This is unacceptable for embedded systems applications that depend on flash memory systems to provide reliable code storage for many years of service.

Enabling flash memory systems to continue to reliably support code storage in embedded computer applications requires that wear induced by write requests be minimized. One means of reducing write requests is to reduce the impact of the overhead writes performed by the flash translation layer (FTL) that is used to manage the flash memory while presenting the overall flash memory system as a non-volatile rewritable block device. These overhead write activities may be represented with a measure of Write Amplification Factor (WAF) which is the amount of flash write requests scaled by the amount of write requests performed by the system hosting the

flash memory system.

In this dissertation, we present FSAware, a novel algorithmic approach that enhances existing FTL designs. Specifically, FSAware reduces overall WAF by separately supporting the write requests associated with the file data and file system overhead produced by host file system write activities. FSAware distinguishes file data write requests from file system overhead write requests by characterizing the file system installed on the flash memory system by the host system. We consider the File Allocation Table (FAT) format, which is specifically selected for its ubiquity in embedded computer applications. FSAware is applicable to both block-mode and page-mode style FTLs.

In this work, we develop a novel instrumentation technique called FTLProbe to develop empirical (gray box) models of commercially available drives. Our empirical models are then used to develop WAF equations for file system operations. Simulations of FSAware on commercially available drives are validated with extensions of these WAF equations. Our simulations results show that FSAware can produce a 97% reduction of WAF for a block-mode FTL and a 36% reduction of WAF for a page-mode FTL. Further extension of the WAF equations show that an enhanced FSAware that consolidates meta-data into a single flash allocation unit can theoretically produce a 99% reduction of WAF for a block-mode FTL and a 64% reduction of WAF for a page-mode FTL. With these reductions in overall WAF for file system operations associated with embedded systems, FSAware can form the basis of an ultra-reliable flash memory system for embedded computer applications.

Contents

Chapter 1

Introduction

Flash memory has been accepted as the basis of code and data storage in embedded systems applications for several decades. In more recent years, widespread mainstream acceptance of flash memory in consumer and enterprise applications has created tremendous downward cost pressures on flash memory manufacturers. The flash memory manufacturers have responded to these pressures by compromising on parameters that are most critical for flash memory's continued suitability for code storage in embedded computer applications.

Most significantly, data *retention* specifications have been decreased from a fixed value of ten years to a value coupled with the amount of utilized flash lifetime (*program/erase cycles* or *P/E cycles*). Retention specifications are now commonly expressed as 10 years at up to 10% of P/E cycles and 1 year at 100% of P/E cycles. Retention between these P/E cycle values degrades exponentially.

Moreover, as semiconductor process lithographies have decreased and the number of data bits per cell has increased, the specified number of P/E cycles available per flash lifetime (*endurance*) has also decreased. Endurance specifications were once

100K cycles for *single-level cell (SLC)* flash memory. This specification has decreased to 60K cycles for SLC flash memory and 3K cycles for *multi-level cell (MLC)* flash memories.

The impact of these specification changes on flash memory system lifetime is significant. Previous generations of embedded systems could depend upon flash memory to support operations of up to 100K P/E cycles and to continuously provide 10 years of data retention through this operational lifetime. Current flash memory systems can only support a few percent of the operational lifetime during which time data retention will decrease to as low as 1 year. This is not acceptable for embedded systems, which may be intended to have a service life of many years.

Enabling flash memory systems to continue to reliably support code storage in embedded computer applications requires that data retention be extended as much as possible. Given the coupling of retention and endurance, this requires that wear induced by write requests be minimized as much as possible. One means of minimizing write requests is to reduce the amount of the overhead writes performed by the *flash translation layer (FTL)*, a layer that is used to manage the flash memory, while presenting the overall flash memory system as a non-volatile rewritable block device. These overhead write activities may be represented with a measure of *Write Amplification Factor (WAF)*, which is the amount of data written to flash scaled by the amount of data written by the system hosting the flash memory system:

$$Write\ Amplication\ Factor\ (WAF) = \frac{Data\ Written\ to\ Flash}{Data\ Written\ by\ Host}$$

The intrinsic nature of flash memory makes it well suited to support *sequential write* requests. When supporting sequential write operations, WAF values are very low. Specifically, the WAF value approaches unity with only a minor overhead used

for the FTL's own internal data management.

However, the intrinsic nature of flash memory makes it poorly suited to support *random write* requests. When supporting random write operations, WAF is significantly higher as the FTL needs to perform considerable amounts of copy operations to support the flash memory system's abstraction of appearing as a non-volatile rewritable block device such as a hard disk.

The challenge of managing and reducing WAF values is compounded by a disconnect between academic literature and industry practice. Specifically, FTLs employed by industry in commercially available flash memory systems are regarded as trade secrets and very few details regarding their operation are ever released. Academic researchers have been frustrated by this disconnect [45, 11, 4, 19, 12, 102] and have resorted to idealized models of FTL implementations [53, 13].

To remove this disconnect and provide the research community with realistic models that represent industrial FTLs, this work develops and demonstrates the power of empirical (gray box) FTL models of commercially available flash memory systems. These models are developed using a specialized instrumentation technique identified as FTLProbe developed in conjunction with this work that measures the performance of a flash memory system in response to a specifically designed series of write operations. Consistent performance impacts occurring in both temporal magnitude of impact and frequency of occurrence are interpreted in the context of FTL management activities. Leveraging this experimental approach, empirical models of the first-order effects of user data management as well as second-order effects of FTL data can be constructed.

Our empirical models enable a user to estimate WAF values for host write requests. Compound extensions of these empirical models allow estimates of WAF values for

host file system activities. Analysis of file system activities suggests that write activities are not as random as may be perceived by the flash memory system based upon the scattered, discrete, write requests observed. Instead, file system activity may be generally expressed as a series of sequential write operations of file data coupled with a series of *repeated* write operations of file system meta-data, such as pointer tables and directory structures.

FSAware is an enhancement to existing FTL designs that reduces overall WAF values by separately supporting the sequential write requests associated with the file data and repeated write requests associated with file system overhead data. FSAware distinguishes file data write requests from file system overhead write requests by characterizing the file system installed on the flash memory system by the host system. File Allocation Table (FAT) format is specifically selected for its ubiquity in embedded computer applications. FSAware is valid for both block-mode and page-mode style FTLs. By reducing WAF values, FSAware reduces utilized P/E cycles for a given workload thereby extending retention. As such, FSAware can form the basis of an ultra-reliable flash memory system for embedded computer applications.

1.1 File System Aware (FSAware) FTL Extension

FSAware is a new design that enhances existing FTL designs. Specifically, FSAware effectively increases retention by reducing the P/E cycles required to support a given workload. FSAware achieves this by reducing the WAF associated with file system operations. This reduced WAF is due to FSAware being able to recognize file system operations as a collection of sequential write requests, interspered with a collection

of repeated write requests.

FSAware contrasts with existing FTLs which may perceive the write requests associated with these same series of file system operations as a collection of random write requests. This mis-classification is the result of the FTL's inability to recognize the purpose of the sequence of write operations that it receives. This is particularly true if we are observing requests from multiple processes or threads. This operation is highlighted below in Figure 1.1 for a standard FTL.

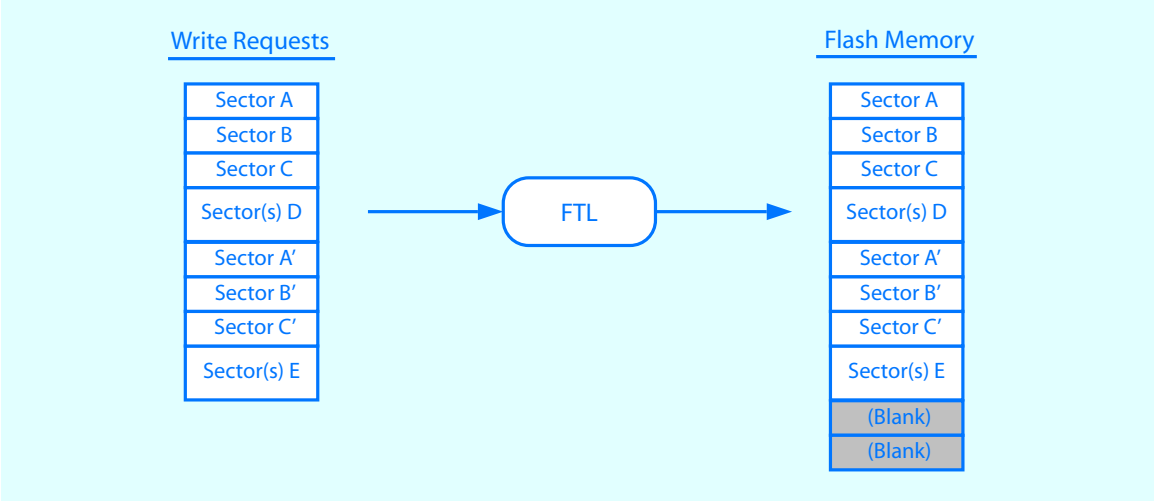

Figure 1.1: FTL Processing of file operations.

As Figure 1.1 shows, all writes are transferred into spare flash blocks exactly as received. When garbage collection is performed on these flash blocks, WAF will be high due to the amount of valid data in the blocks that will need to be copied.

FSAware is designed to be specifically aware of the file system format utilized with the flash memory system. With this ability, we can segregate write requests based on logical block address. File data writes, which are normally sequential in nature, are processed with one handler and file meta-data, such as the FAT tables and directory entries utilized in a FAT file system, are supported by additional handlers.

The operations of these handlers are shown below in Figure 1.2 for an FTL with our FSAware extensions.

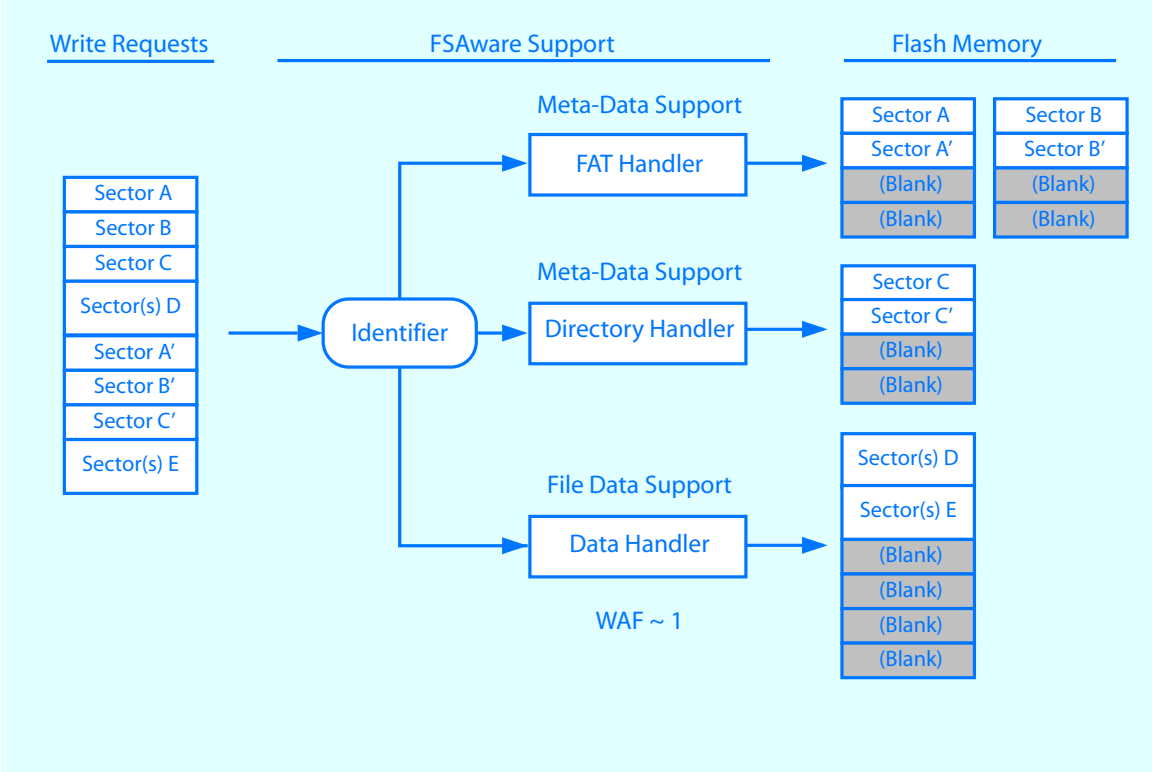

Figure 1.2: FSAware FTL extension processing of file operations.

By minimizing the amount of write requests that are mis-classified as random, FSAware is better able to leverage the inherent nature of flash memory to efficiently support sequential write operations with low overhead (WAF). As Figure 1.2 suggests, FSAware can support file data with a WAF readily approaching unity. Moreover, as the meta-data writes are repeated, much of the data in the meta-data blocks is invalidated and the need to copy data from the meta-data support blocks is greatly reduced. As such, the overall WAF is expected to be greatly reduced with FSAware.

FTL models and WAF equations are needed to demonstrate the improvements

produced with FSAware. However, as discussed above, FTLs utilized in commercially available products are commonly regarded as trade-secrets and insufficient details are available to develop even rudimentary WAF estimations for compound write requests such as file operations. As will be discussed in Chapter 3, researchers have been frustrated by this lack of disclosure [45, 11, 4, 19, 12, 102] and have developed idealized models of FTLs for WAF estimations and other modeling purposes [53, 13].

Rather than focus on idealized models, this work aspires to demonstrate the effectiveness of FSAware for commercially available FTLs. As such, this work is built upon empirical models of commercially available flash memory systems. These models were used to develop WAF equations for single-mode transfers. These single-mode WAF equations were then extended to develop WAF equations for file system activities. Finally, the models were adjusted to include the impact of the FSAware FTL extension and its effectiveness is demonstrated.

1.2 Contributions

This work provides the following contributions to the state-of-the-art in flash memory system analysis and design:

- FTLProbe Measurement Technique: We have developed a novel instrumentation methodology FTLProbe to study commerically available flash memory systems. This technique purposely saturates the flash memory system with specific sequences of write requests and provides performance measurements of the individual transfers.

- Empirical (Gray Box) Models of FTLs: The arrays of the fine-grained performance measurements produced by FTLProbe can be analyzed to identify

periodic performance impacts outside of the norm. Both the magnitude and frequency of these periodic performance impacts are interpreted in the context of flash management activities. Empirical (gray box) FTL models, based upon these flash management activities, are constructed. These empirical models are novel in the sense that they enable analytic study of commercially available products. Previous analytic work could only study theoretical models as FTLs in commercially available products that are regarded as trade secrets, where operational details are not commonly disclosed.

- WAF Measurements and Equations: The FTLProbe measurement technique facilitates measurement of Write Amplification Factor (WAF) in commercially available flash memory systems. Only a single WAF measurement of a simplified flash memory system has been previously published [59]. This work presents an array of WAF measurements for more complex flash memory systems. These measurements also served to validate WAF equations developed using the empirical (gray box) FTL models.

- The recognition of "repeated" as a Significant Transfer Mode: The pervasive perspective of considering host write requests as either sequential or random, borrowed from prior hard disk drive (HDD) characterization work [98], has been expanded to include the notion of "repeated" write requests. Repeated writes are writes conducted at specific logical addresses, such as commonly occur for file system meta-data. While repeated requests can appear as sequential to hard disk drives, they appear as random to flash memory systems. This transfer mode is the least efficient in terms of WAF and should not be the default for commonly encountered write operations, such as those for meta-data.

- FSAware: A file system aware (FSAware) FTL extension was developed. FSAware is a multiple-pool FTL that specifically separates data and meta-data by understanding the file system format that is installed on the flash memory system by the host computer platform to organize its files. Specifically, one pool is used for file system data, and additional pools are used for meta-data types such as FAT and directory structures. The FSAware extension works nicely with existing FTL designs, suggesting that no new changes in existing FTL designs are needed, other than the addition of multiple active blocks (pools). Expansion of the empirical models and WAF equations, as well as validation using measurements of the commercially available flash memory systems that were used to develop the empirical models, indicate that FSAware can reduce WAF by 97% for block-mode FTLs and by 38% for page-mode FTLs for 4 KB files.

- FSAware Enhanced: FSAware Enhanced has also been developed. FSAware Enhanced is a multiple-pool FTL extension similar to FSAware. However, FSAware Enhanced recognizes that multiple meta-data writes occur proximate in time with each other. FSAaware Enhanced proposes grouping meta-data into a single flash memory system allocation unit. This sector-mode FTL is a more significant FTL development activity. This technique could not be simulated with the commercially available flash memory systems used in this study. However, further extension of the empirical models and WAF equations indicates that FSAware Enhanced can reduce WAF by 99% for block-mode FTLs and by 64% for page-mode FTLs for 4 KB files.

1.3 Organization of This Thesis

The remainder of this document is organized as follows. In Chapter 2, background material relevant to this thesis is presented. This includes material related to the economic pressures impacting public disclosure of the FTL in the various market segments, recognizing that the present state of the flash memory market desperately needs capabilities enabled by FSAware development.

In Chapter 3 we review related work to FSAware. This work includes prior studies of Flash Translation Layers (FTLs) and Write Amplification Factor (WAF). As there are two decades of research into FTLs, the topic is rather expansive. However, research into the impact of WAF is fairly new, starting only in the last five years (2009) [53], and much of the existing FTL work predates it. Moreover much of the academic study and resulting literature has shifted away from embedded systems towards enterprise applications around 2007 [11]. As such, FSAware is believed to be a novel contribution to the field.

In Chapter 4 we introduce the FTLProbe measurement methodology. FTLProbe is used to develop and validate the empirical (gray box) models that are ultimately used to validate FSAware and FSAware Enhanced. Measurements of single-point write requests and multiple-point write requests associated with file system operations collected from FTLProbe are presented. Observations of these measurements that provide the foundation of our empirical models and, ultimately, the WAF equations, are discussed.

In Chapter 5, simulations that demonstrate the effectiveness of FSAware are presented and validated using adaptations of the empirical models. Finally, the empirical models are further extended to demonstrate the theoretical effectiveness of FSAware Enhanced.

In Chapter 6, directions for future work inspired by this thesis are presented. This thesis concludes with a summary and list of contributions in Chapter 7.

Chapter 2

Background

2.1 Flash Memory

Flash memory is based upon the floating gate transistor developed at Bell Labs by Kahng and Sze [60]. As Figure 2.1 shows, it is similar to a common MOSFET transistor except that it has an electrically isolated planar region near the gate called a floating gate. Because of its electrical isolation, charge within the floating gate persists without any external supply and the floating gate transistor is classified as non-volatile. Charge storage times (*retention*) vary based upon semiconductor design specifics, but have historically been on the order of 10 years.

Charge is added to or removed from the floating gate using a quantum process known as tunneling. The process of tunneling requires relatively large voltage potentials (typically 5-12 V) to provide electrons with sufficient energy to enter the silicon oxide insulation layer and have the potential to tunnel through. The applied voltages vary depending upon whether charge is being added to the floating gate or removed from it. These different voltage requirements imply that the process of programming

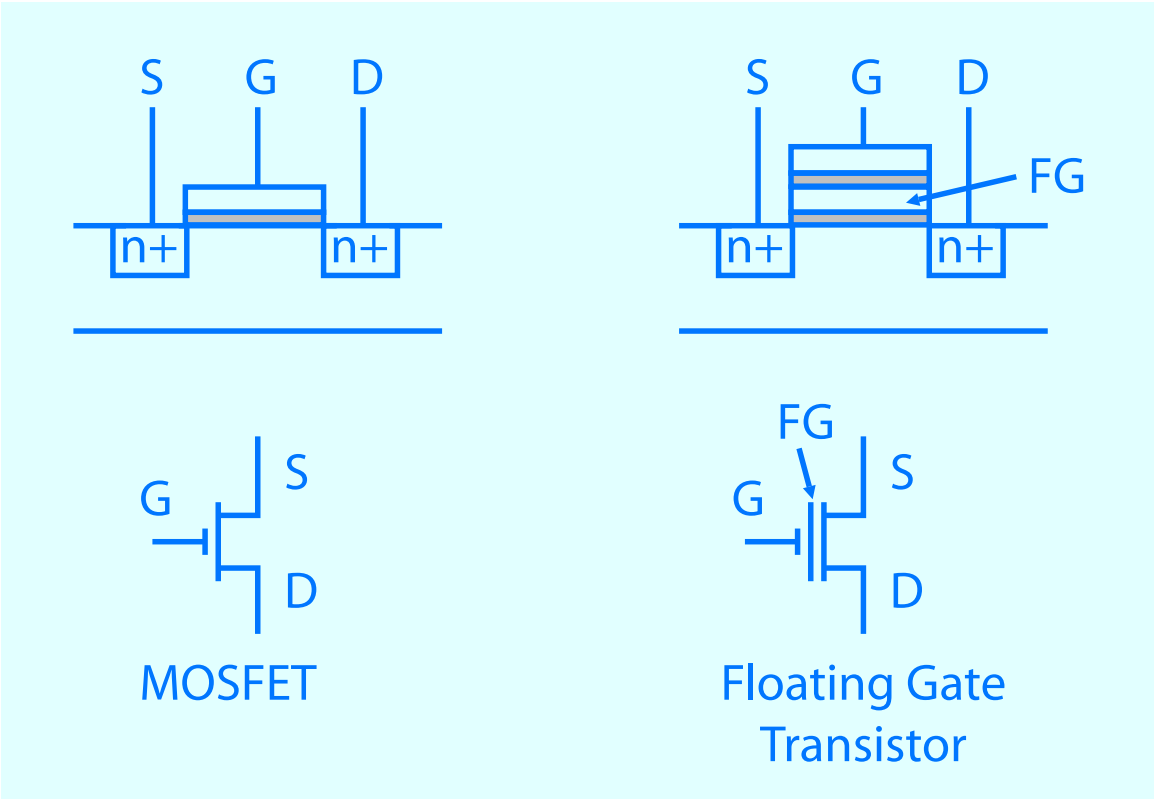

Figure 2.1: MOSFET and Floating Gate Transistor

the transistor (adding charge to the floating gate to that it represents a '0' state) and the process of erasing the transistor (removing charge from the floating gate so that it represents a '1' state) are different. These processes are illustrated in Figure 2.2 As such, a floating gate transistor is considered programmable and erasable, but not rewritable in the sense that is commonly associated with memory technologies such as DRAM and SRAM. As will be discussed, modern forms of memory based on the floating gate transistor continue to expose the distinct program and erase operations through its external interface.

The stress induced by the passage of electrons through the oxide layer between the gate and source-drain during programming and erasing degrades the oxide layer. In

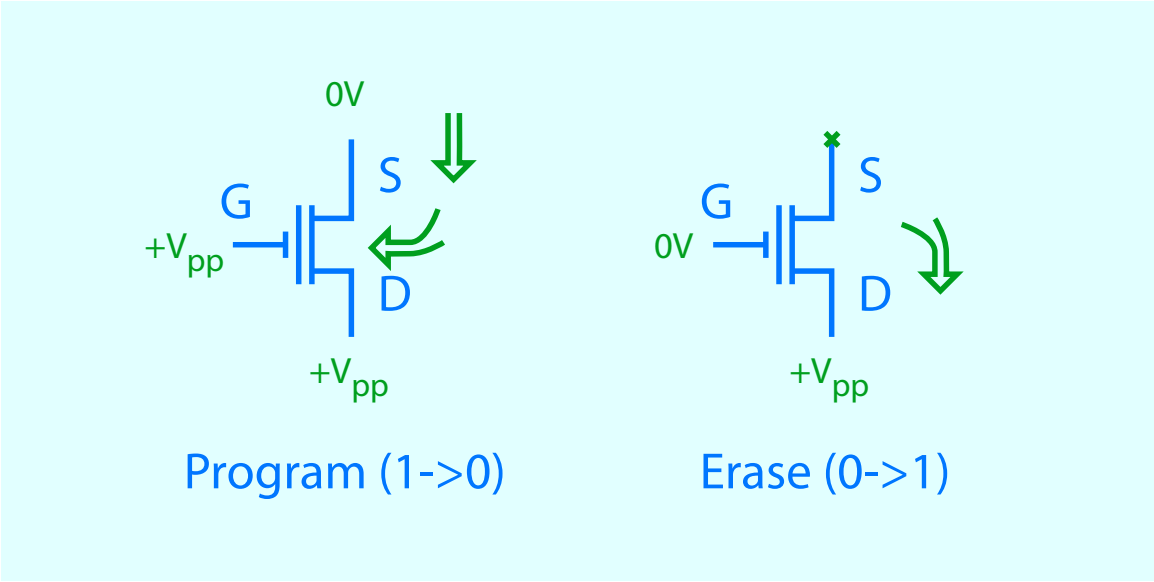

Figure 2.2: Floating Gate Transistor: Program and Erase

its degraded state, the oxide layer can trap electrons (see Figure 2.3). When sufficient electrons are trapped, the cell is falsely read as programmed even though the floating gate may be completely devoid of electrons. In this state, the floating gate transistor is considered to have failed through fatigue. All forms of memory based on the floating gate transistor continue to have this failure mechanism and, as a consequence, have limited *program/erase cycles* (*P/E cycles*). P/E cycle ratings can vary significantly based upon semiconductor design specifics, but have historically been on the order of 100K (10^5) cycles.

Kahng and Sze were discouraged by their management from pursuing development of the floating gate transistor into a commercial product [41] and the technology went unused for a number of years. Eventually, the floating gate transistor saw its first commercial application in the EPROM technology developed by Dov Frohmann-Bentchkowsky at Intel and released in 1971 [93, 43]. The EPROM only supported electrical byte programming and required a UV light to impart energy to the electrons

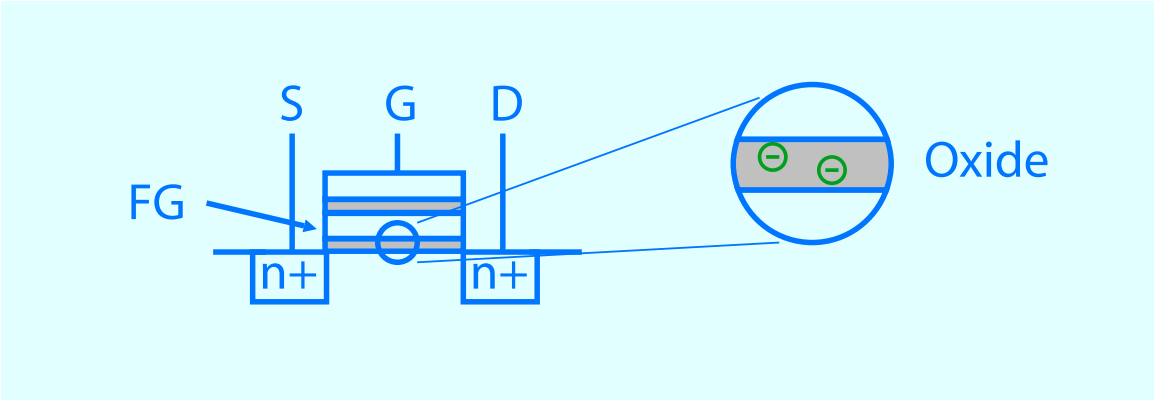

Figure 2.3: Floating Gate Transistor: Failure, Trapped Electrons

in the floating gate for bulk erasure of the device. Although it was an electrically incomplete implementation, it was notable as it was the first commercially available memory technology that was both nonvolatile and changeable.

The floating gate transistor next saw commercial application in the EEPROM technology developed by George Perlegos at Intel and released in 1978 [93]. The EEPROM allowed programming and erasing of individual bytes. This allowed the EEPROM to be an electrically complete implementation that no longer required a UV light for changes to stored data.

The EEPROM is a desirable device for computer architecture designs as it offers randomly accessible, byte-addressable memory. However, it has considerable limitations. Most significantly, the line density within the semiconductor process to support the byte-addressability is high. This limits process scaling and its inherent cost benefits. Moreover, floating gate programming and erasure are slow processes. Changing arrays of bytes is a rather time-consuming operation. While data in the EEPROM is indeed changeable, the performance limitation of the operations prevents the EEPROM from being used as a practical randomly accessible memory. It remains in use as of this writing as small data caches which do not need to change frequently such

as MAC addresses in Ethernet cards.

The next two advancements in floating gate based memory would both come from Fujio Masuoka [44] who was a line manager at Toshiba. Masuoka envisioned arrays of floating gate transistors organized in groups that would become solid-state block storage devices similar in function to sectors in mechanical hard drives; rather than byte-addressable memory. With this vision, Masuoka first developed NOR flash memory in which bytes were clustered into groups called erase blocks. This reduced line density and subsequently reduced the cost of the semiconductor device. Moreover, by performing multiple floating gate erasures in parallel, they were erased "in a flash" and the net performance of the erase operation was improved. The design was identified as NOR flash memory as the collection of floating gates appears similar to a logical NOR gate implemented in MOSFETs (see Figure 2.4). The read and program operations in NOR flash memory remained byte-addressable.

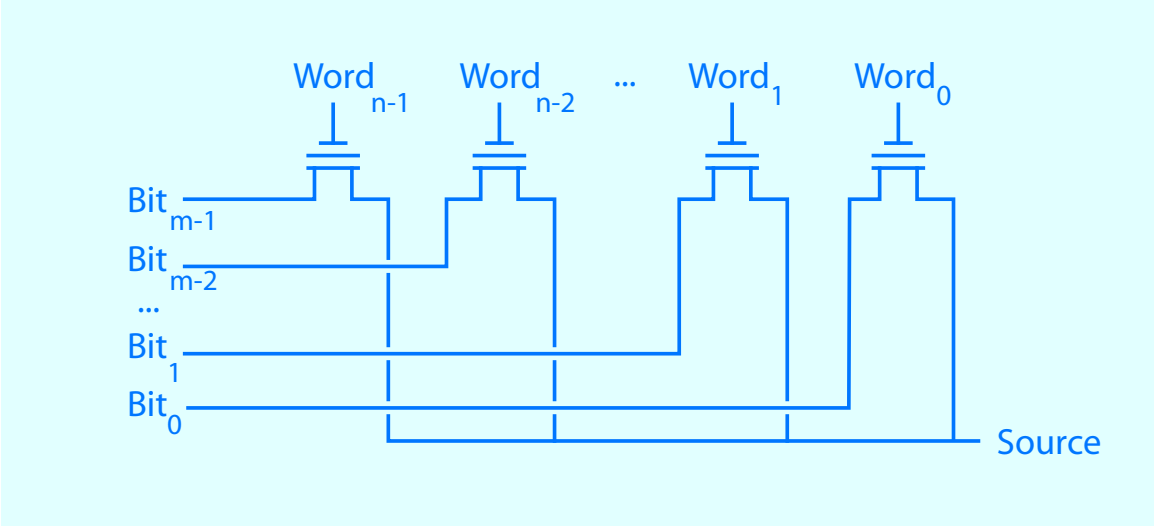

Figure 2.4: NOR Flash Memory

Masuoka next developed NAND flash memory. In this implementation, floating gates remain clustered into groups called erase blocks, but they are also grouped

into units called pages for reading and writing. This design fully abandons byte-addressability, but comes closest to the solid-state block storage device that Masuoka originally envisioned. This design further reduces line density which again reduced the cost of the semiconductor device. Moreover, by performing multiple floating gate programming operations in parallel, the net performance of the program operation was improved. The design wass identified as NAND flash memory as the collection of floating gates appears similar to a logical NAND gate implemented in MOSFETs (see Figure 2.5).

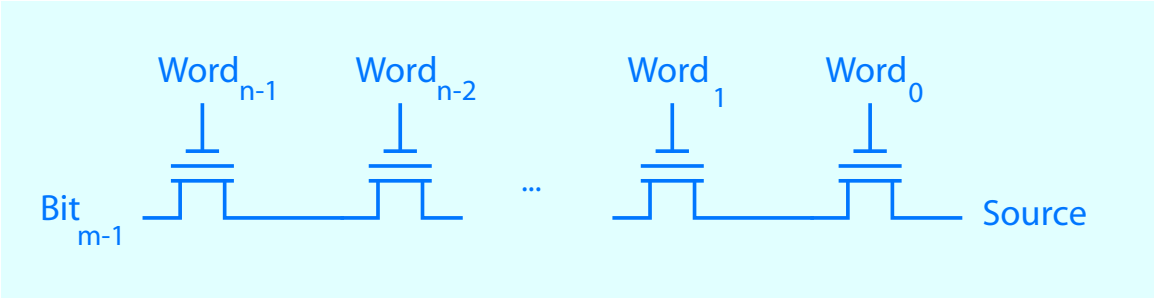

Figure 2.5: NAND Flash Memory

As might be expected, NAND flash memory is a significant departure from NOR flash memory in that noise and errors are inherent in the design as all reads are performed though a string of floating gate transistors, rather than individually as was the case with previous memories. Because of this noise, error checking and correction (ECC) is always required in the design of NAND based flash memory systems. Whereas all of the previous designs of memory could be directly connected to a host processor in a parallel fashion similar to RAM, NAND flash must be used as a secondary block device much like the traditional hard drives that Masuoka intended to replace. However, because the lines required for random access were removed, NAND flash memory cells could achieve $4F^2$ spacing whereas NOR cells could only

achieve $10F^2$ spacing [26]. This density improvement gave NAND flash memory a distinct cost advantage.

It is an interesting aside to note that Masuoka developed both NOR and NAND flash on his own time in the Toshiba factory starting around the year 1980. Masuoka eventually published his NOR design at IEEE IEDM in 1984 [75] and his NAND design at IEEE IEDM in 1987 [76]. Toshiba did not approve of his independent development work and did not support development of a commercial product based his designs until many years after other companies introduced them to the marketplace.

Modern flash memory, as it is known at the time of this writing, is based upon Masuoka's design for NAND flash memory. Specifically, NAND flash memory (or simply flash memory) continues to consist of an array of floating gate transistors grouped in units of erase blocks which are further subdivided into pages. No other advances have occurred through design at the semiconductor level to achieve a density higher than the $4F^2$ offered by NAND flash memory.

Flash memory does however continue to evolve in terms of semiconductor process scaling and multiple bits stored per cell. With regards to semiconductor process scaling, flash memory has continually evolved from 1500 nm (1.5 um) in 1986 to 19 nm in 2014. The 19 nm flash that is available as of this writing is referred to by the flash memory manufactures as the "1y" process node [25]. For several years now [15, 90, 116] the "1y" process node has been regarded as the second to last planar semiconductor process shrink for flash memory with "1z" being the last process node. This process node is expected to be on the order of 15 nm and made available in 2016 [100]. Of particular concern, cell-to-cell interference is having an overwhelming effect and further planar scaling beyond "1z" is not being pursued. Subsequent density improvements are expected to come as 3-D semiconductor technologies [90, 33].

Beyond semiconductor process scaling, density and cost improvements in NAND flash have been enabled by storing multiple bits of data per cell. This requires additional logic for reading and programming as multiple charge levels are required. (Mis)usage of the terminology in industry has defined MLC as storing two bits per cell and TLC as storing three bits per cell. The charge levels in these styles of memory are shown with traditional SLC memory in Figure 2.6.

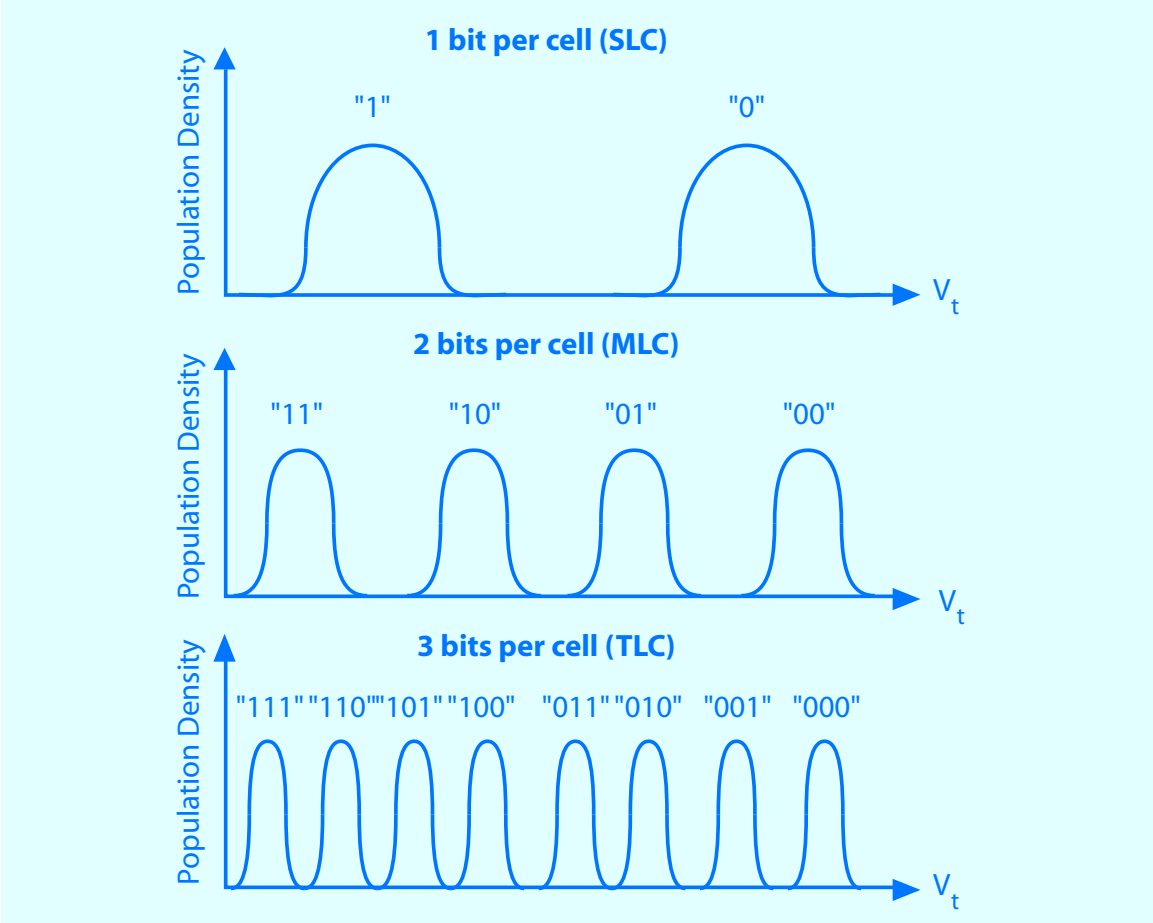

Figure 2.6: SLC, MLC, and TLC charge levels.

With the advances in flash memory design, scaling, and increased bit density, the cost of flash memory has steadily decreased. As costs have decreased, marketplace

acceptance has increased. This has created a positive dynamic which has enabled NAND flash memory to become the cheapest memory available when it surpassed DRAM in 2003 [49]. NAND flash memory has also been the leading semiconductor process technology since 2003 surpassing DRAM and even logic (microprocessors) [32].

Flash memory has revolutionized, and continues to revolutionize, many aspects of computer architecture. Several consumer electronic devices such as cellular/smart phones and digital cameras owe their existence to flash memory. At a Lifetime Achievement Award ceremony honoring Sze in 2014, it was remarked that the floating gate transistor exists in one form or another in nearly 100% of all currently shipping electronics devices and an estimated 10^{21} floating gate transistors have shipped to date [41].

2.2 Application Segments

Over the years, three highly generalized segments of computer applications have evolved: consumer, enterprise, and embedded. Each of these segments is distinctly different in its intended audience of end-users and their resulting usage of and demands on flash memory. Understanding these segments and their respective requirements for flash memory systems is important for understanding the motivation and applicability of the present work.

The consumer application segment is the category of general-purpose computers that is most familiar to the general public. This category includes laptops, PCs, tablets, and any other computing platform which becomes specifically suited to its end-user's needs by virtue of the software that is ultimately loaded upon it. This segment of computer user is most significantly cost sensitive.

The enterprise application segment is the category of general-purpose computers that are most familiar to those in the Information Technology (IT) industry. As with consumer applications, the enterprise computers are general-purpose and are enabled to suit their end-user's needs by virtue of the software that is loaded upon them. This segment of computer user is usually a business for which return on investment (ROI) is most critical. As such, this segment can tolerate cost so long as it can be justified on a cost basis. Higher performance and longer lifetimes are commonly desired attributes.

In contrast to general-purpose computer applications, embedded applications are fixed-function computer systems. They are designed as a system with both hardware and software (firmware) to perform a fixed set of tasks; perhaps as few as one. The array of embedded computer designs is enormous and exceptionally diverse. Examples include medical equipment, communications equipment, industrial computers, and automotive subsystems just to name only a few. Embedded systems generally utilize flash memory for code and data storage. Embedded systems may be expensive, difficult to service, and have very expensive downtime costs. As such, reliability is most critical to this segment of computer user.

Because of their need of reliability and reduced cost sensitivity, embedded systems were early adopters of flash memory. Coupled with the relatively low storage density requirements, it is fair to conclude that embedded systems have generally always used solid-state storage for code and data whereas general purpose computers have relied upon mechanical hard drives for much of their existence. In fact, it is somewhat interesting to note that the first EEPROM [43] was released in the same year (1971) as the 4004, Intel's first microprocessor, and the two technologies were effectively intertwined in the market place for a number of years.

In 1992, the size of the flash memory industry was $270M [80]. At this time, the cost of flash memory was around $22/MB which generally prohibited its usage in consumer and enterprise applications. It could be inferred that 100% of the flash memory industry at that time was due to embedded systems applications. With each passing year, the positive dynamic of decreasing cost and increasing demand pressured flash memory manufacturers to continue their cost reductions.

In 2010, the price of flash memory was $1.00/GB [114] and the size of the flash memory industry had increased to $18.7B [55]. A marketing study of the flash memory industry conducted by Gartner in that year found that around 80% of that market was devoted to consumer applications and 10% was devoted to each enterprise and embedded applications [108].

The flash memory industry is expected to grown to an estimated $28B in 2014 [31] and the price of flash memory has dropped to $0.45/GB [30]. In the time since the 2010 Gartner study, consumer applications and enterprise applications, in particular, have continued to grow while the embedded applications, being more mature, have not. Extrapolating a constant embedded market size from the 2010 data suggests that usage in embedded systems now accounts for less than 5% of the total flash memory industry. These market size segments are highlighted in Figure 2.7.

As Figure 2.7 suggests, embedded systems were effectively the only users of flash in its early days. As such, flash manufacturers were well motivated to ensure that flash memory met the needs of the segment. However, as the sizes of the other segments, particularly consumer, grew, flash memory manufacturers were compelled to redistribute their attentions. As mentioned above, the driving demand of consumer application is cost reductions. To meet this demand, flash memory manufacturers have needed to compromise on parameters significant to enterprise and embedded

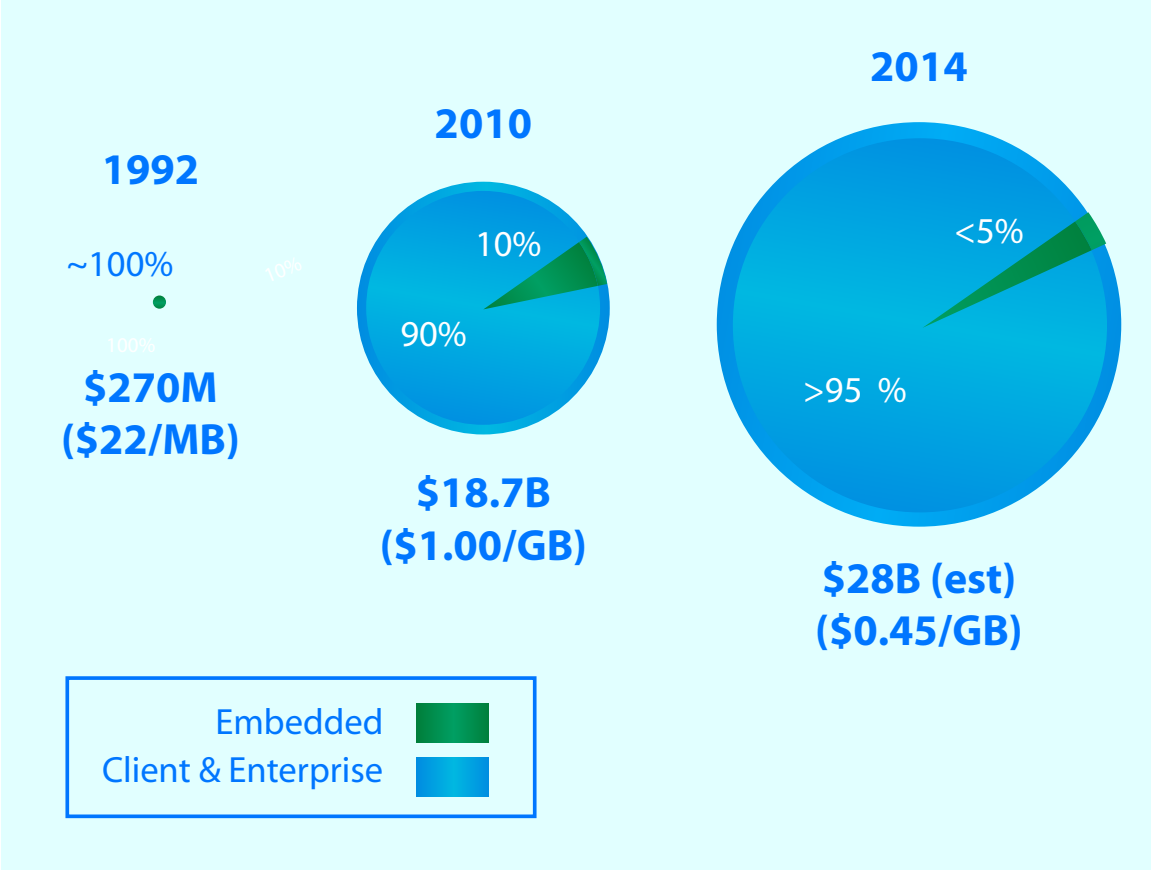

Figure 2.7: Market & Segment Sizes

applications.

2.3 Specification Degradation

Flash memory has many parameters of significance. Three of the more importance parameters relative to this work are:

- Cost ($/GB)
- Endurance (P/E cycles)

- Retention (Years)

As presented in the previous section, cost reductions from the consumer segment have created pressure on flash memory manufactures. Flash memory manufactures have responded in ways that have reduced endurance (significant to enterprise applications) and retention (significant to embedded applications).

The changes in endurance in recent years are illustrated in Figure 2.8. As mentioned above, SLC floating gate endurance has been on the order of 100K P/E cycles for much of its existence. To maintain compatibility with EEPROM designs, Intel had always mandated endurance specification of flash to maintain this level. However, as semiconductor processes have been reduced to 32 nm and below in more recent years, endurance specifications needed to be relaxed to 60K P/E. More significantly, MLC floating gate endurance was on the order of 10K P/E cycles at its introduction in 2005. Lithography reductions have rapidly reduced this specification to 3K. Endurance for TLC floating gate memory has been well below 1K P/E cycles for all of its existence. See [2, 90, 116, 50, 1] for details.

Similar to its mandate with endurance, Intel mandated that its flash memory match its EEPROM and have a ten year retention specification. For many years, flash did indeed have a ten year data retention specification. However, as with endurance, the effects of semiconductor process scaling eventually forced relaxation of the retention specification.

The compelled relaxation of retention specifications occurred at 60 nm. Unlike endurance, which had a basic decrease, retention became coupled with life-cycle in terms of utilized P/E cycles. Specifically, whereas flash memory has previously had a retention specification of 10 years regardless of much of it was cycled, at 60 nm and below, flash memory was rated to have a retention specification of 10 years at start of

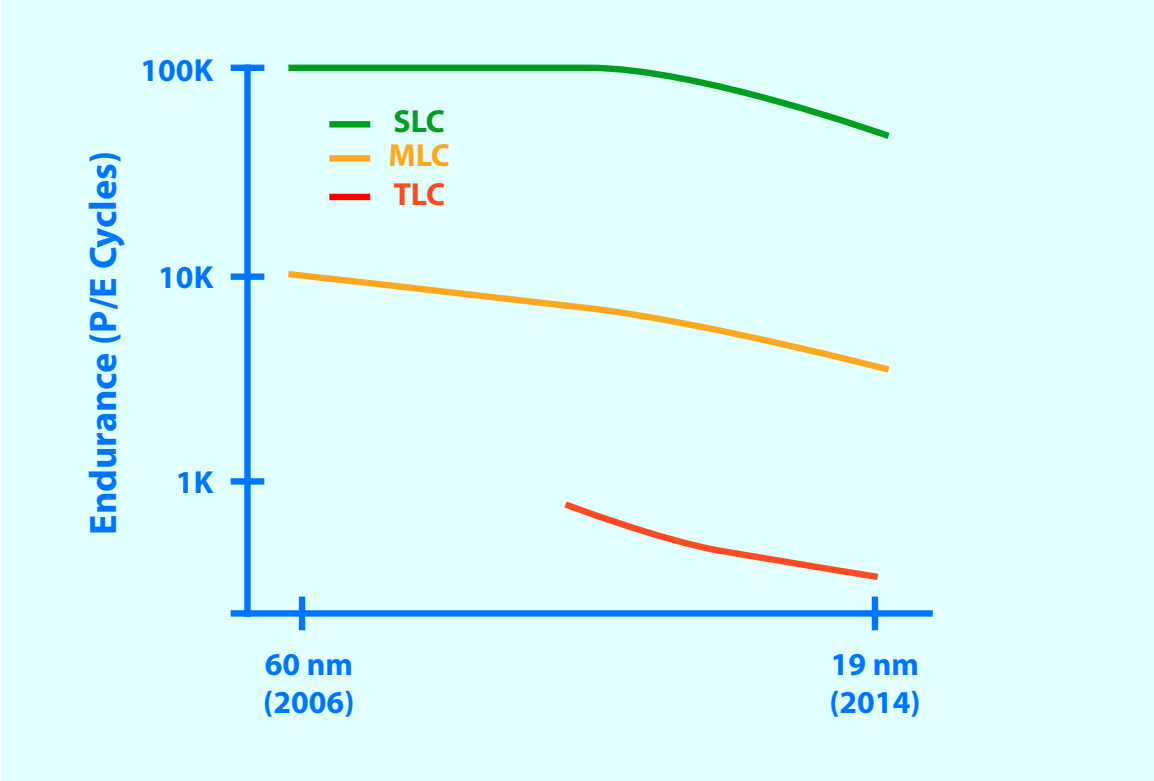

Figure 2.8: Endurance Specifications for Various Flash Memories

life, 10 years at 10% of utilized P/E cycles, and 1 year at 100% of utilized P/E cycles. MLC has a similar retention specification. However, as its endurance specification is much lower, the time to reach 100% P/E cycles is far shorter than SLC memory for a consistent workload. The endurance specification of TLC memory is 1 year at start of life, 1 year at 10% of utilized P/E cycles, and 3 months at 100% of utilized P/E cycles. These specifications are highlighted in Figure 2.9. See [2, 90, 116, 50, 1] for details.

In 2016, the endurance and retention specifications are believed to make TLC unsuitable for embedded applications. However, it is noted that MLC memory was once regarded as inadequate for either enterprise or embedded applications. Despite this historical perspective, cost pressures have enabled MLC memory to become

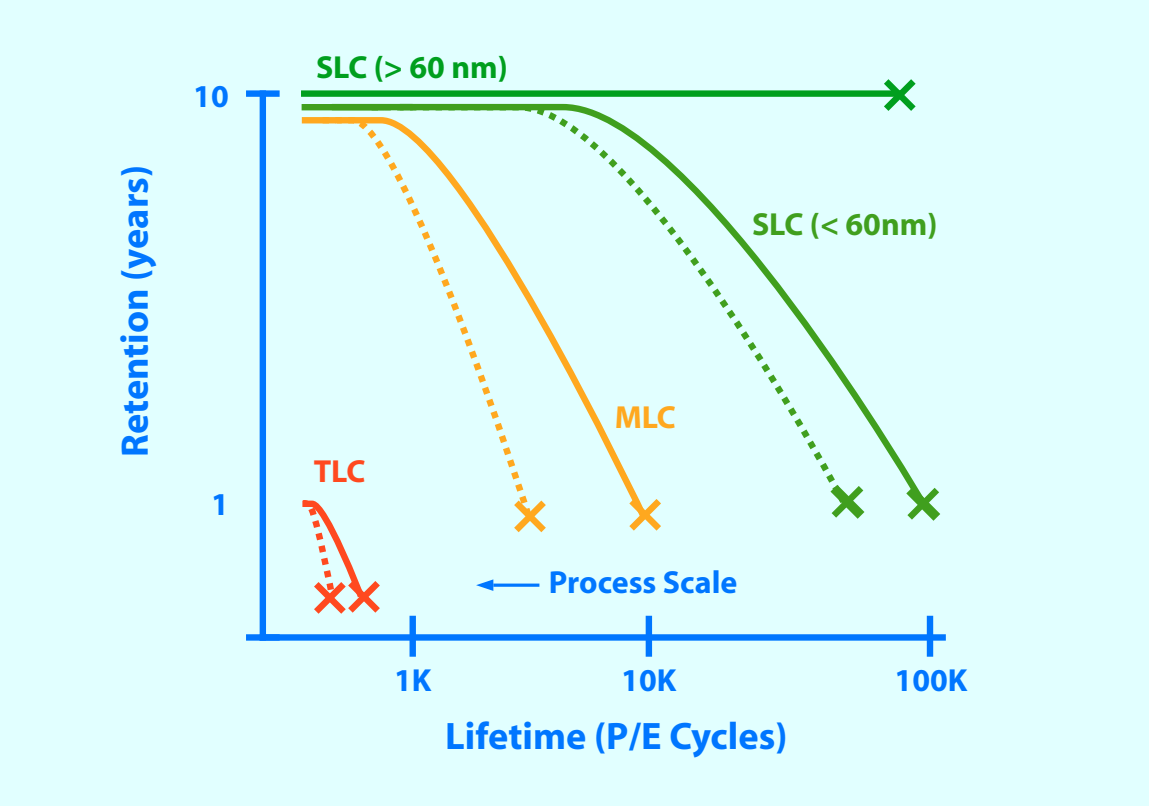

Figure 2.9: Retention Specifications for Various Flash Memories

mainstream in enterprise applications and is rapidly becoming adopted in embedded applications.

The coupling of endurance and retention specifications has created a secondary stress on flash memory manufacturers. Specifically, how do they balance specifications between the needs of the enterprise (endurance) and the needs of embedded systems (retention)? As recently as 2010, the markets were roughly of equal size [108]. However, in the time since, enterprise usage of flash has increased considerably whereas embedded usage of flash has stayed relatively consistent (recall Figure 2.7). Attention of the flash memory manufacturers has understandably shifted to enterprise applications.

For example, flash memory manufactures have responded to the needs of the enterprise with a specialized form of MLC memory called eMLC. This flash memory has an increased endurance specification (30K P/E cycles), but a lower retention specification (5 years start of life, 5 years 10% of utilized P/E cycles, 4 months at 100% of utilized P/E cycles).

The shift of attention from embedded systems to enterprise applications is also noted in the research literature. As will be discussed in Section 3.1, academic publications targeted embedded systems [17] up until around 2007 [11]. Since that time, academic publications have largely targeted SSDs in enterprise applications[87].

As the enterprise segment continues to evolve and increase its usage of flash memory, its demands are increasingly at odds with the needs of embedded systems applications. In 2013 at an industry conference, Facebook publicly requested even cheaper flash memory with only 30 days of retention [105]. The flash memory manufactures have not yet responded to the Facebook request, in part because they are "already making the cheapest flash that they know how" [50]. However the direction of the flash memory manufacturers is clear; they will create flash memory to suit the needs of their largest segments. Embedded applications represent a dwindling portion of market share and will be most challenged to use these newer flash memory devices while continuing to meet the historically established needs of embedded systems [77].

2.4 Flash Memory Systems

Modern flash memory systems generally use collection of flash memory devices to emulate non-volatile rewritable block devices such as hard disks. As discussed previously (recall Section 2.1), commercially available flash memory devices have three

peculiarities that complicate their usage in flash memory systems: a.) erase-then-program architecture, b.) coarse-grained erase blocks and program pages, and c.) finite program and erase endurance.

Flash memory systems support the emulation of a block device and address the peculiarities of flash memory devices using a system of management known as a Flash Translation Layer (FTL). FTLs manage the erase-then-program architecture using a system of indirection between host sectors and discrete units of flash using a mapping table to track the indirection. These concepts are illustrated in Figure 2.10. As this figure suggests, sectors are stored incrementally within the flash units.

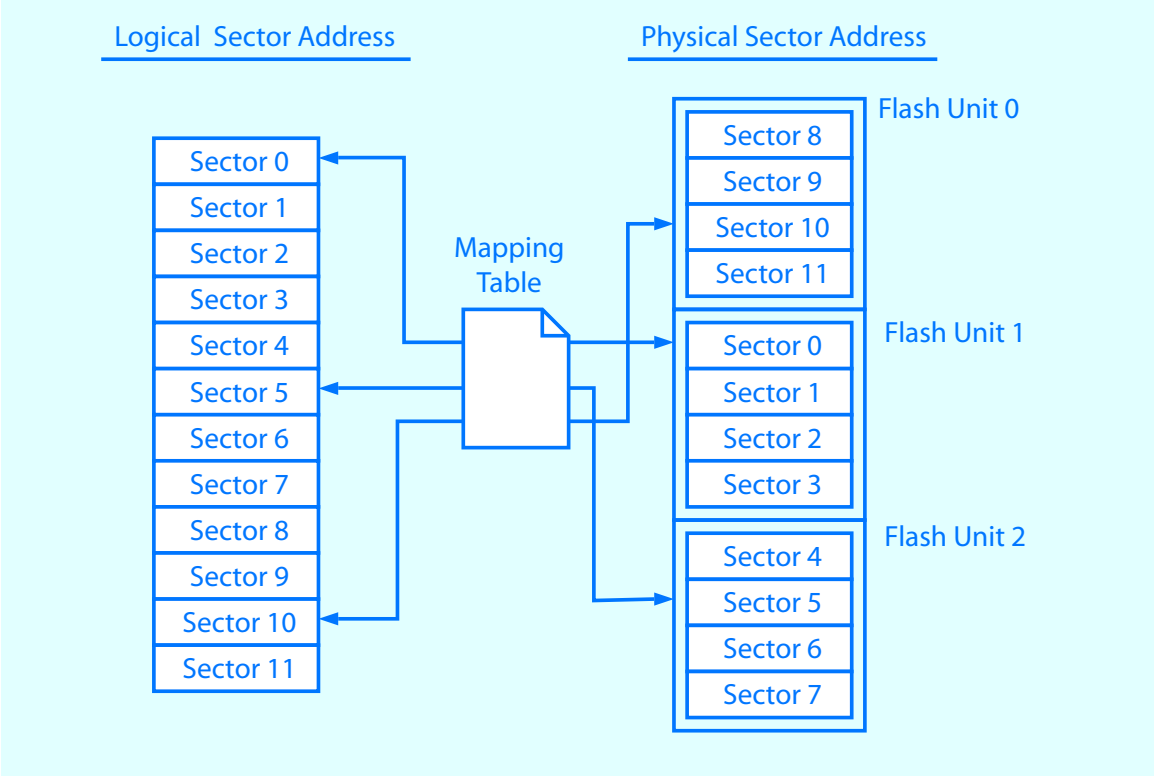

Figure 2.10: FTL Indirection

There are three general types of FTLs:

1. Block-Mode

2. Page-Mode

3. Hybrid

In block-mode FTLs, flash erase blocks are used as flash units and in page-mode FTLs, flash pages are used as flash units. As flash erase blocks are generally much larger than flash pages, block-mode FTLs may be considered to be more practical in terms of computation capabilities and RAM requirements. However, as will be discussed (see Section 3.1), block-mode FTLs may require much more flash copying to support the hard drive abstraction. This increased overhead accelerates flash wear and reduces performance. As such, page-mode FTLs may be considered more desirable. However, for much of the history of flash memory systems, page-mode FTLs were impractical due to computational requirements and, more specifically, RAM cost.

Given the performance limitations of the block-mode FTL and the historically impractical costs associated with page-mode FTLs, block-mode and page-mode FTLs may be viewed as two extremes of available options. As will be explored in depth in Section 3.1, most FTL designs employ a blending of these techniques and are collectively identified as hybrid FTLs. In summary, the hybrid FTLs are generally block-based FTLs for most of the storage space, but use a small space managed with a page-based FTL for caching to aid performance.

Read requests by a host system are readily supported with any of the FTL designs. The controller supporting the FTL simply uses the mapping table to locate the requested sector(s) and transfers them to the host system. Page-mode FTLs require more translations, but a design with a capable processor and the mapping table in fast RAM minimizes the impact. At the flash level, read operations are performed at page granularities regardless of the size of transfer request made by the host system.

As read operations are well managed, they are not considered further in this work. In contrast, write operations are far more challenging to support with flash memory.

The erase-then-program architecture of flash memory makes flash memory systems relatively well suited to supporting sequential write requests. Specifically, the FTL streams incoming write requests to spare flash units; either block or page depending upon the FTL design. When the flash unit is filled, the FTL updates the mapping table pointer to indicate mapping with the new flash unit and the previous flash unit is considered to be invalid. If a block-mode FTL is used, the flash unit is an erase block and the invalid block can be immediately erased. If a page-mode FTL is used, the flash unit is a page and the FTL needs to wait until an entire erase block of invalid pages is accumulated before it can be erased. These concepts are illustrated in Figure 2.11.

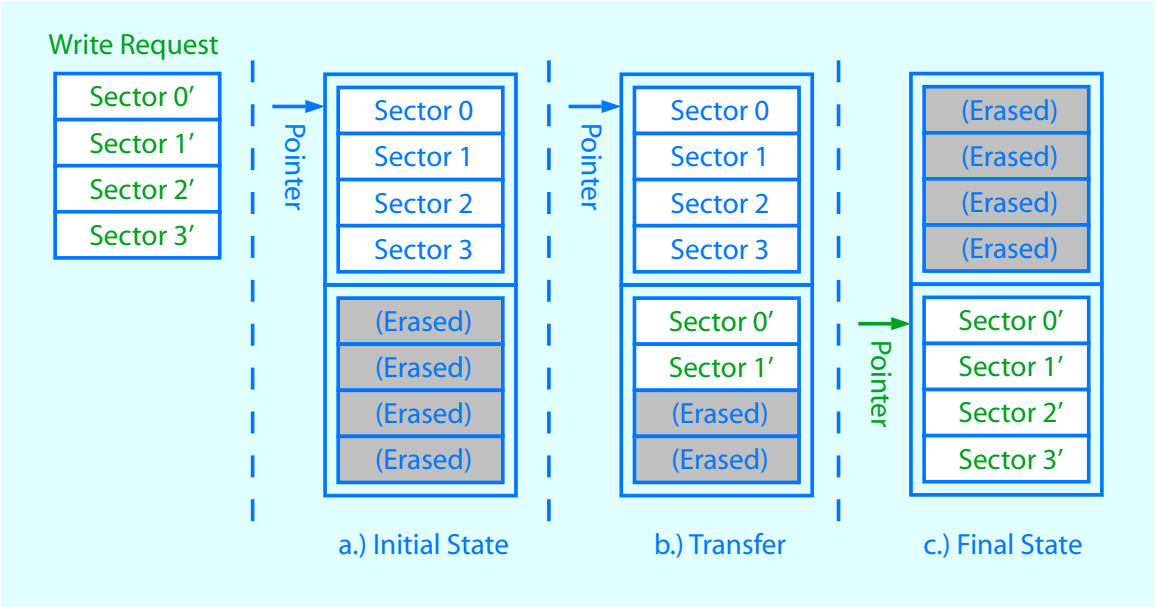

Figure 2.11: FTL Support of Sequential Write Requests

It is important to note that no copy operations occur among the flash units for

sequential write requests. Rather, the write request is directly committed to a spare erase block. Only the slight overhead of updating the flash unit pointer in the mapping table needs to be committed to flash beyond the data of the write request. This operation is highly efficient.

The true challenge to FTL design is encountered in supporting random write requests. The erase-then-program architecture of flash memory makes flash memory systems poorly suited to supporting random write requests. Beyond the flash, pure block-mode FTLs are also poorly suited to handling random write requests. When these operations are encountered, the pure block-mode FTL needs to copy all pages in the block to maintain the incremental integrity of the block. The overhead to support this extraneous copying has a considerable penalty in terms of increased flash wear and performance. These operations are illustrated in Figure 2.12.

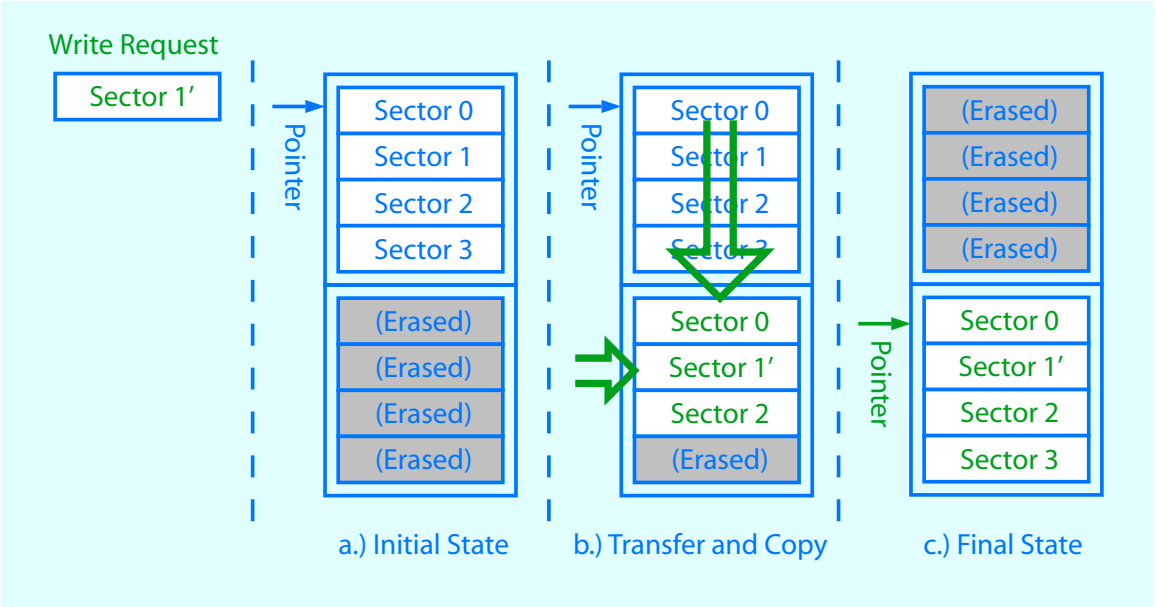

Figure 2.12: Block-Mode FTL Support of Random Write Requests

Because of the granularity of its design, page-mode FTLs can potentially alleviate

the need for copy operations among flash blocks. Specifically, as the flash pages within a flash erase block are each tracked independently, there is no need to maintain the incremental integrity within the flash erase blocks. With this constraint eliminated, write requests can be transferred to flash pages as received. The order of the write requests, sequential or random, is irrelevant. As each flash page is committed, the mapping table pointer is updated. As individual pages cannot be erased, the page with the previous data is marked as invalid. The overhead of updating the flash unit pointers in the mapping table is greater than for block-mode FTLs as more pointers are involved, but this operation is still highly efficient. These details are highlighted in Figure 2.13.

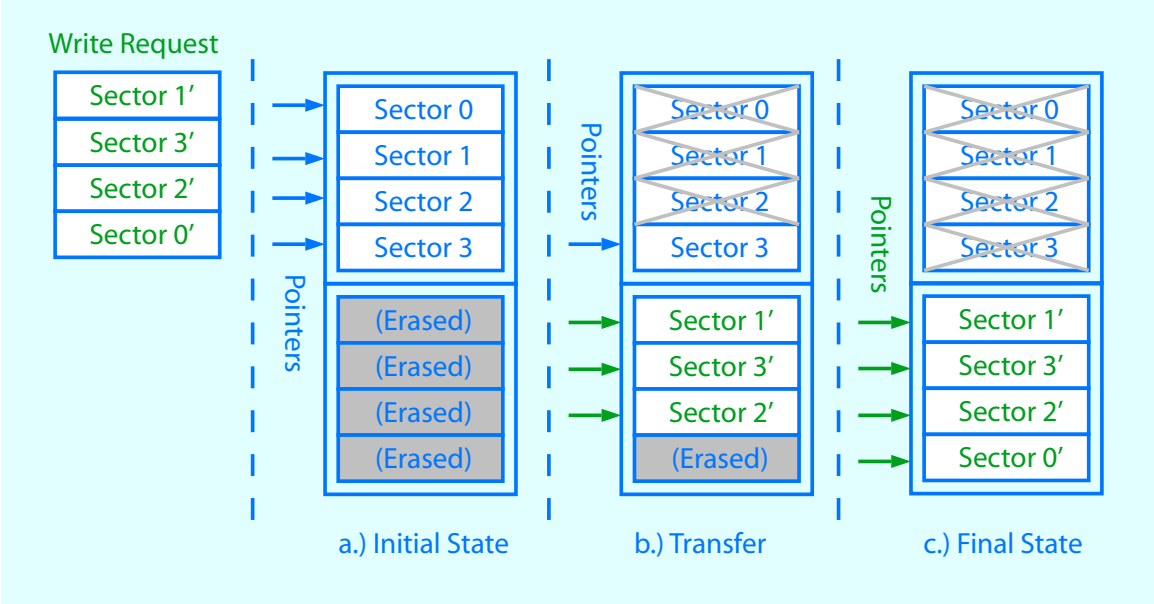

Figure 2.13: Page-Mode FTL Support of Random Write Requests

For simplicity, the flash page size shown in Figure 2.13 is exactly one sector. While this geometry of flash memory was commercially available in the past, modern flash has page sizes which are multiples of sector size. Using this flash memory for single

sector writes does require copy operations to maintain the incremental integrity of sectors within flash pages. However, considering that flash pages are much smaller than erase blocks (commonly on the order of 64 times smaller), the copying associated with a page-mode FTL to support random write requests is greatly reduced [64].

The process of recovering invalid flash units is called garbage collection. The simplest form of garbage collection is when an entire block is invalidated as shown in Figure 2.11. The invalid block can simply be erased. The data pointer(s) have already been updated at this point in time. This operation may also be called a *switch* in the literature [64].

Pure block-mode FTLs always store sectors incrementally within a block, so no other forms of garbage collection are required. In effect, the cost of overhead of the copying is paid at the time of the write operation.

Garbage collection gets more complicated for pure page-mode FTLs and hybrid FTLs leveraging an area managed by page-mode FTLs. With these FTLs, spare blocks are used to cache incoming writes. However, there is only a very limited number of spare blocks. When the spare blocks become full, the valid data within selected blocks must be consolidated (merged) so that block erases may occur and new spare blocks can be generated. This occurs by copying the valid data from blocks targeted for garbage collection into the remaining available spare block(s). This process is shown in Figure 2.14. As part b.) of this figure suggests, there is no change in the data during the merge operation. All of the copy operations are being performed solely to isolate a block selected for erasure. This process is very inefficient.

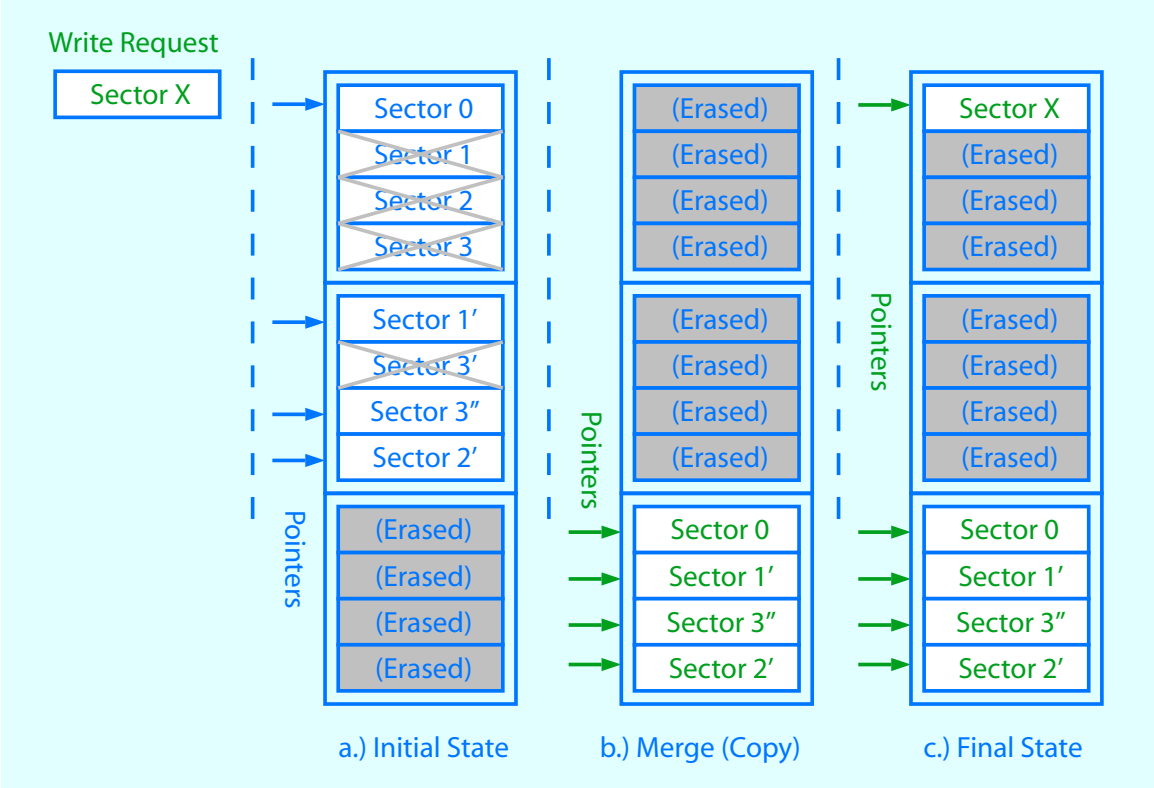

Figure 2.14: Garbage Collection

2.5 Write Amplification

As discussed in the previous section, the most significant function of the FTL is to provide a system of indirection that allows the flash memory to appear as a rewritable array of host addressable blocks. Because of the inherent nature of flash memory (erase-then-program architecture and relatively large-grained structures such as flash erase blocks and flash pages), the FTL needs to perform copy operations to support this abstraction. These copy operations are either in the form of block copy operations to support random operations in a block-mode FTL or page copy operations to support garbage collection in a page-mode FTL.

Recalling that flash memory has a finite endurance (P/E cycles), the design of

the FTL is considered to be critical as it affects the amount of data written to the flash memory beyond those used to support host write requests. This is considered especially true in embedded systems applications for which the critical retention parameter degrades with increasing P/E cycles.

To provide a basis of FTL efficiency evaluation, a term identified as *Write Amplification* was proposed [53]. The equation of *Write Amplification Factor (WAF)* is the amount of data written to the flash memory scaled by the amount of data written by the host system:

$$Write\ Amplication\ Factor\ (WAF) = \frac{Data\ Written\ to\ Flash}{Data\ Written\ by\ Host}$$

It is noted that FTL designs require non-volatile memory for their own usage. For example, the mapping table shown in Figure 2.10 is normally in RAM for performance reasons. However, this is only a working copy. The mapping table needs to be maintained across power cycles so FTLs will utilize portions of the flash memory for this purpose. As will be presented (see Chapter 4), this overhead is minimal (later measurements will demonstrate approximately 2%) with the bulk of overhead being due to the FTL copying of data for management purposes.

Because flash memory is well suited to sequential write operations, even the most basic FTLs are inherently well suited to efficiently support sequential write operations [64, 70]. Specifically, incoming write requests from the host system are programmed as whole pages incrementally into a single spare erase block without the need of copy operations (recall Figure 2.10). Even considering the small amount of FTL management data written to flash, this mode of operation is generally quite efficient and capable of offering the highest possible data rates with WAF values, as will be shown, approaching unity (1.0).

Unfortunately, flash memory systems are poorly suited to supporting random write operations. There are significant challenges to FTLs when used to support random write operations [64, 70]. As will be discussed, many types of FTLs have been used over the years to handle these operations (see Section 3.1). In the extreme, an FTL using pure block mapping would use an entire spare erase block to support a single write request by merging it with current data from its existing block location. This mode of operation would have very low performance and WAF values of 100s or even 1000s depending upon the block size and minimum allowed host transfer size. As such, pure block-mode FTLs were only used in very early flash memory systems and have not been widely used since.

FTLs using a mixture of block mapping and page mapping or pure page mapping have an array of spare blocks to support the writes of page mapped data. When these blocks are completely consumed for incoming writes, a process of garbage collection is used to consolidate data and free up some of the used blocks for reuse as spare blocks. The process of garbage collection is time consuming and involves a significant performance decrease and WAF value increase. Overall, the design of the FTL is intended to ensure that average performance and write amplification are improvements over the levels offered by block-mode FTLs. As might be expected, different FTLs can cause widely varying performance and, as will be shown, WAF values. Moreover, these values can be expected to vary depending upon actual usage scenarios. See [64, 70] for details.

2.6 FAT File System

Hard drives have traditionally utilized a file system to organize their data in a fashion recognizable to a host computer system. As flash memory systems have emulated the sector based interface of hard drives (recall Section 2.4), they have also commonly utilized file systems. Numerous file systems are available including FAT [38] for legacy Windows and DOS, NTFS [86] for more recent releases of Windows, and ext for Linux [35, 36, 37].

FAT was originally developed in 1977 [38] by Microsoft. Because of the relative simplicity of computers in existence at this time, FAT was a very minimalistic file system with very low computational and memory resource requirements for the hosting computer systems. Because embedded computer systems have traditionally also been designed with minimal computational and memory resources, the FAT file system was well suited towards these computer systems. Numerous documentation efforts and open-source implementations have helped to ensure that FAT remains ubiquitous in embedded computer applications as of this writing [38, 39, 40]; even as available computational and memory resources have increased in these computer applications. Because of its ubiquity as well as the general concern for backwards compatibility in flash memory system design, the FAT file system is selected for consideration in the present work.

Summarizing from the Microsoft FAT specification [38], a FAT file system consists of a group of sectors linearly organized as clusters. Clusters are a fixed number of sectors. This value is recorded in a leading data structure called a BIOS Parameter Block (BPB) which is stored in the first sector of the disk partition. This sector is identified as the Boot Sector. Clusters are tracked with an index data structure called a File Allocation Table (FAT). For legacy concerns regarding the potential for hard

drive data reliability, two identical FATs are utilized. These concepts are illustrated below in Figure 2.15

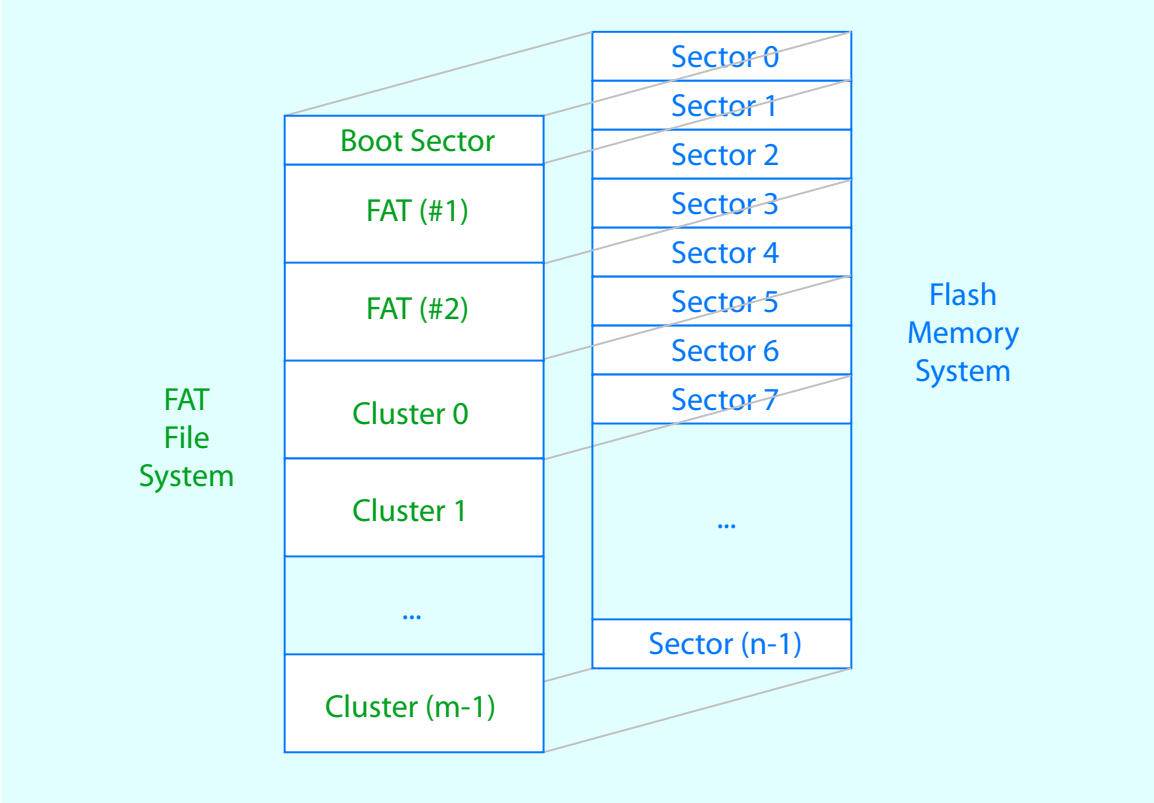

Figure 2.15: FAT Structures

The first clusters are reserved for the root directory. The root directory contains a list of files and directories stored in the root directory. File and directories are recorded in DirectoryEntry structures. DirectoryEntry structures include a starting cluster index which contains the initial contents of the file or directory. By looking this index up in the FAT structure, the next cluster index can be determined. By repeating this process, the entire chain of cluster indexes of the file or directory can be determined. The file data is contained incrementally at these cluster indexes. Sub-directories under the root directory are similar to files except that they include

a flag that identifies them as directories in the DirectoryEntry data structure and their contents are also in the form of DirectoryEntry structures. These concepts are illustrated below in Figure 2.16.

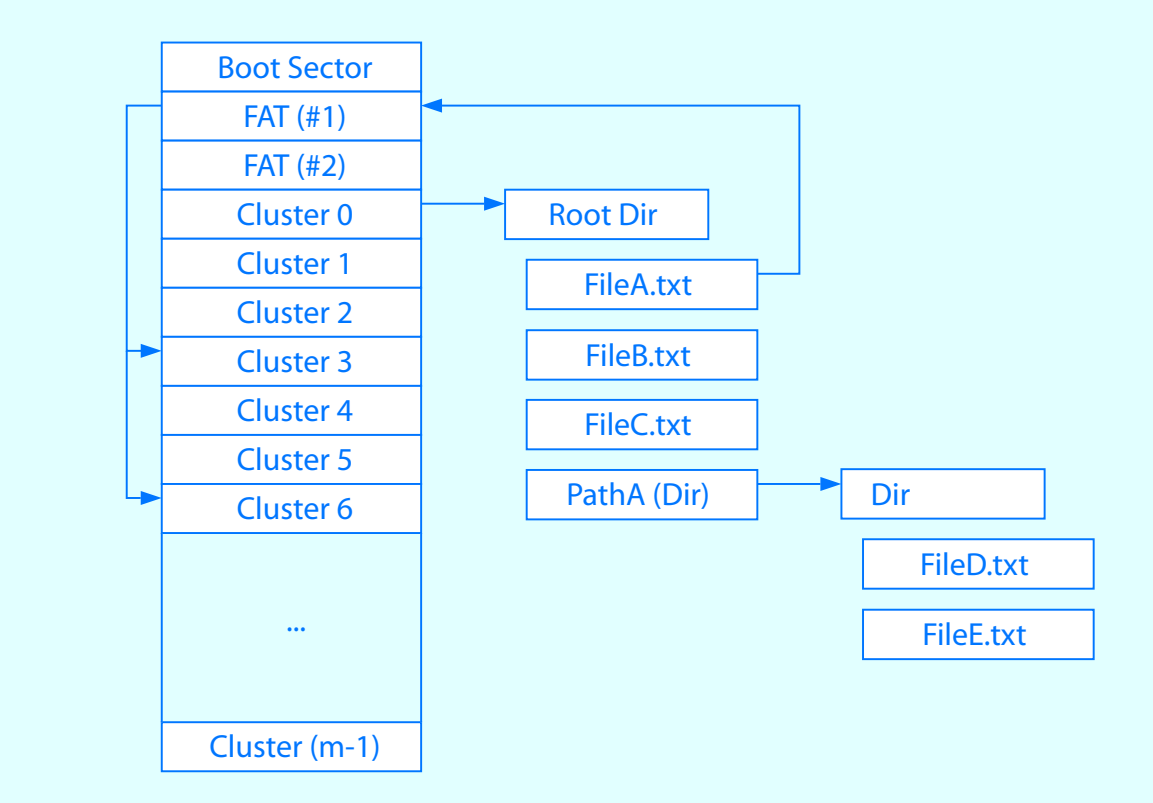

Figure 2.16: FAT File System

The design of FAT file system includes two types of meta-data which describe the file data: DirectoryEntry and FAT structures. Because there are two identical FAT structures, each file operation that involves writing a cluster of user data involves writing the cluster as well as three sectors of meta-data.

2.6.1 Write Request Types

For mechanical hard disk drives, read/write accesses commonly included consideration of sequential and random access types [98]. More specifically, mechanical hard disk drives could readily stream data between the host system and the spinning magnetic media with minimal read/write head motion. As such, hard disk drives were considered to be well suited for sequential accesses. However, mechanical hard drives had more difficulty supporting random accesses. Specifically, as these transfers potentially required significant read/write head motion, there was potentially a much longer transfer time required for this access type.

These two models of data access potentially formed the boundaries of hard drive operation. Sequential transfers provided best case performance and random transfers provided worst case performance.

Again owning to the legacy support considerations of flash memory systems (recall Section 2.4), the conventional models of accesses were again considered to be sequential and random. This was especially reasonable given that the inherent nature of flash memory readily supports sequential write accesses and is less well-suited to random write accesses.

However, this perspective is rather restrictive given the FTL's system of indirection. This is especially true with respect to file systems. Specifically, whereas hard drives were traditionally write-in-place and could incur performance penalties as the read/write head displaced to support the accesses of file data and meta-data, flash memory systems are not. Because of the system of indirection, data and meta-data have no specific separation and, more importantly, no inherent performance impacts when written. Recall Figure 2.16.

Moreover, when considering the FAT file system (recall Figure 2.16), observe that

there is nothing "random" about the meta-data of the DirectoryEntry and FAT data structures. All of these data structures are stored in fixed sector locations that, owing to the simplicity of the FAT file system, are easily discoverable with minimal computational resources. As such, the potential of knowing about the file format installed on the flash memory system is well within the capabilities of the flash memory management controller. By learning these details and leveraging our observation that meta-data is more properly considered as "repeated" (not random) write requests, it is possible to create a file-system aware (FSAware) drive that is optimized for these operations.

Chapter 3

Related Work

3.1 Flash Translation Layers (FTLs)

As mentioned previously (recall Section 2.4), there are three types of FTLs:

1. Block-Mode
2. Page-Mode
3. Hybrid

Block-mode FTLs manage logical sectors incrementally within flash erase blocks and page-mode FTLs manage logical sectors incrementally within flash pages. While flash erase block and flash page sizes have varied over the generations of commercially available flash devices, flash erase blocks have always been significantly larger than flash pages. The resulting storage and computational resource requirements of block-mode FTLs are significantly lower than page-mode FTLs.

Because of the reduced storage and computational requirements, it is not surprising that the earliest FTLs utilized block-mode FTLs. Two notable exceptions to this

trend were very early work on FTLs [112, 62]. Wu's work described a page-mode FTL for which NOR flash was used for a main memory; not a storage device. Kawaguchi presented a pure page-mode FTL for storage. As discussed below, resource costs would prohibit commercial adoption of this design until around 2012.

A 1995 US patent [7] described the first block-mode FTL. Ban's work was later adopted by the PCMCIA industry association in May 1996 [92]. A frequently cited Intel white-paper [56] explains the specification and Ban's original work. Collectively, Ban's patent, the PCMCIA specification, and the Intel white-paper document the first pure-block mode FTL.

Ban's 1995 US patent was originally for an FTL to manage NOR flash memory. A second US patent filed in 1999 was for an FTL to manage NAND flash memory [8]. The FTL was called NAND Flash Translation Layer (NFTL). This FTL was also a pure-block mode FTL.

Aside from the challenges with specification degradation (recall Section 2.3) associated with flash memory, MLC flash memory has additional restrictions on how data is written to flash. Specifically, MLC flash memory prohibits random page access within blocks. Dan and Singer [27] and Qin et al. [95] recognizes that the block mode FTLs (NFTL in particular) require random access and proposes Sequential Access FTL (SAFTL) to utilize MLC memory with a block-mode FTL.

Block-mode FTLs are more efficient than page-mode FTLs in terms of management resource requirements. However, the main weakness of the block-mode FTLs is the magnitude of copy operations that need to occur within the flash blocks to support random write operations can be substantial. While this document recognizes this overhead as needless write amplification, previous FTL research observed that this overhead was undesirable solely because of the performance impact [23].

Researchers [23] have long recognized that true page-mode FTLs would reduce the amount of internal copying operations. Earlier research [112, 62] presented pure page-mode FTLs. However, the cost of resources, RAM in particular, made the cost of such designs impractical for many years of flash memory system utilization. Only more recently (starting in 2012) has the cost of computational resources decreased to the point for which flash memory systems with pure page-mode FTLs been commercially viable [94, 103]. Page-mode FTLs may also be denoted log-structured following the file-system concept proposed by Rosenblum [98] in which only file changes are recorded.

The design of block-mode FTLs has been augmented to include groupings of flash erase blocks called *superblocks* [61, 71]. Given the efficiency of page-mode FTLs, this development is somewhat counter-intuitive. However, super-block architectures support erasure of multiple flash erase blocks in parallel which improves performance. As will be discussed, superblocks are utilized in commercially available FTLs.

In the years following the descriptions of block-mode FTLs and leading up to feasible page-mode FTLs, a considerable number of developments occurred in which block-mode FTLs were augmented to include a subset of blocks that were managed as page-mode to act as caches to support random write performance. These intermediate FTLs are identified as hybrid FTLs in this document. The first hybrid FTL was presented by Kim et al. in 2002 [64]. This paper was later identified as Block Associative Sector Translation (BAST) [70]. Later work [24, 89], FAST [70], EAST [66], Reconfigurable FTL [88], JFTL [22] and Self-Tuning Hybrid FTL [83] presented similar approaches. Because the page-mode blocks are limited in a hybrid FTL, write demands may cause "log-block thrashing" [107] which can negate the performance and reduced wear benefits intended by the hybrid FTL.

Internally, FTLs may also utilize log-structured journaling approaches for flash management data structures [58]. Birrell [11] proposes using RAM for flash management data structures to support fast random writes. The authors concede that data loss is a risk with this design; although they have extended their design to minimize this loss.

Gupta et al. present a page-mode based FTL identified as Demand-based FTL (DFTL) [47]. While pure page-mode in terms of mapping, DFTL uses paging for flash memory management structures to limit overall RAM requirements. Later work [54, 74, 115, 113] follows similar approaches.

As mentioned before, a pure-page mode FTL was previously described [62]. As of this writing, no further papers describing pure page-mode FTLs have been identified. Several extensive literature reviews of FTLs have been published [45, 24, 67, 10]. The work of Gal and Toledo [45] and Ben-Aroya and Toledo [10] are notable in that their efforts to include patent searches. These literature reviews cover the details of the FTLs mentioned above, as well as other, less significant, FTL designs. The interested reader is referred to these documents for additional information.

3.2 FTLs: Segregated Data

FSAware proposes to reduce write amplification in FTLs by separately supporting file data as sequential writes and file meta-data as repeated writes. While FSAware is novel in its approach of segregating data within a flash memory system based upon its purpose in the file system, the concept of segregating data has been explored considerably.

The original description of a log-structured file system [98] identifies the notion of

hot data and *cold* data in which hot data is written frequently and cold data is not. By itself, the log-structured file system was intended for hard drives. Interestingly, it was developed to convert random write requests into sequential write requests as hard drives are better suited at handling sequential accesses.

This notion of a log-structured file system leveraging sequential access efficiency is effectively similar to flash memory systems where log-structured file systems are utilized in hybrid and page-mode FTLs. In this regard, the notions of hot and cold data remain applicable. Specifically, hot data will be in recently written log blocks and will actively be copied during garbage collection whereas cold data will tend to remain static. In this regard, a segregation of data based upon access frequency will naturally occur. This effect was noted in the early work by Wu and Zwaenepoel [112] and Kawaguchi et al. [62], although these works utilize a greedy garbage collection policy in which blocks are selected for garbage collection solely on the basis of invalid pages, not *temperature* (activity).

In addition to these early works on page-mode FTLs, it is noted that all FTLs have an inherent tendency to segregate data on the basis of temperature. Recall the sequential write support shown in Figure 2.11. The distribution of writes across the available spare blocks is identified as *dynamic wear-leveling*. In the extreme, a host writing a single block would prematurely wear out the spare blocks while the static data blocks would be unaffected.

Chiang et al. [21] proposes Cost Age Time (CAT) garbage collection on both the basis of data temperature and erase counts. This work purposely identifies data dynamically and clusters it based upon its temperature. The proposed design is intended to reduce erase operations and more evenly wear flash memory by avoiding concentrations of hot data.

Later work [9, 18, 85] describes *static* wear leveling in which writes are specific distributed across the drive with the express goal of evenly distributing wear and prolonging drive life. Unlike the work of Chiang et al. [21], these works avoid clustering of data and specifically seek to undo the effects of access frequency. Wear is slightly accelerated in these systems as each static wear leveling events require one additional block copy and erase operation.

The notion of segregating file data from meta-data for performance has been explored for hybrid disks developed using flash and non-volatile memory such as MRAM [82, 29]. This design can be highly effective in reducing writes in file system meta-data as the selected non-volatile memory technologies are byte-addressable and have considerably higher endurance specifications than flash memory. However, even as of this writing, non-volatile memories are very expensive ($65/MB for FRAM and $45/MB MRAM) and remain available only in very small densities (1 MB for FRAM and 512 KB for MRAM, maximum) [97, 34]. Commercial designs leveraging these technologies remain largely impractical.

The concept of improving performance by segregating data based upon temperature is explored by numerous works. Wang and Hu [110] propose to reduce the work of garbage collection in a general log-structured file system, not necessarily used by flash, by dynamically identifying writes on the basis of temperature and streaming them into different buffers.

The work of Chang and Kuo [16] proposes a striping algorithm in which data is written to multiple flash devices in parallel. This work is adaptive and significant with regards to data segregation in that it specifically directs hotter data to the flash block with the least erase count to better distribute wear.

Later work by Baek et al. [6] specifically suggests that garbage collection in flash

drives has become analogous to seek time in hard drives and proposes a design of page allocation that improves performance by clustering data uniformly, based upon the frequency that the data is updated at.

Chang [16] specifically proposes that a dual-pool algorithm be used with one pool for cold data and one pool for hot data. The pools are adaptive in that they dynamically resize and that cold data migrates towards the cold pool over time (write activity).

The idea of two pools is further developed by Lee et al. [68]. Their FAST FTL uses two pools, but the data is segregated between these two pools on the basis of the sequential or random nature of the write request. A limitation of this design is a lack of a clear heuristic to identify data as sequential. This is especially true when writes are initiated by multiple host threads. As noted before (see Section 2.4, (mis)-identification of write requests as random results in larger amounts of wear.

Similar approaches are presented by Lee et al. with their LAST FTL [69], Ryu with his SAT design [99], Wei et al. with their WAFTL design [111], and Wang and Wong [109] with their ADAPT design. From an FTL perspective, these designs are interesting in that they are hybrid FTLs which dynamically manages blocks as either block-mode (sequential) or page-mode (random), depending upon the data pattern.

Another development of an FTL leveraging two pools is presented by Lee et al. [68]. This work is notable in that it identifies file system write operations as either sequential or repeated write requests and proposes two pools to address these. While similar to the focus of FSAware, the proposed design supports identification of data by frequency at the time of garbage collection and subsequent grouping (migration).

The concept of using deduplication to reduce flash activity was explored by Gupta et al. [48] and Chen et al. [20]. The later work is notable as it is "content-aware".

However, in this case, context-aware refers to the deduplication engine that checks for redundant data. It is not aware of the context of the data at the file system level as FSAware is.

3.3 Write Amplification

Write Amplification has always been present in flash memory systems that implement some degree of persistence of translation layer between the logical block addresses (LBAs) exposed to a host system and physical flash addresses (PBAs) used internally within the storage device due to the added overhead of the translation layer. However, usage of the term "Write Amplification" to describe the phenomena is a much more recent development.

The term Write Amplification was first used by Intel at the Intel Developer Forum in 2007 [46]. The first mention of Write Amplification in the literature appeared in Moshayedi et al. in 2008s [84]. As presented above, work prior to this time appears to focus on other aspects of flash memory systems such as performance [4].

The work of Hu [53] presents an equation defining Write Amplification Factor (WAF). The work uses a probabilistic model of SSDs for evaluation. Their model is intuitively developed using basic knowledge of flash operations and does not include any empirical drive data. Hu's [53] paper cites work by Rosenblum [98] that mentions the concept of extraneous write in general log-structured file systems. The connection between performance and lifetime (i.e. the higher Write Amplification, the slower the performance) is presented. The authors use their simulator to show improvements in WAF by segregating static and dynamic data. This is expected as static data decreases the active size of the storage space.

A slightly later work [13] develops an analytical model of an SSD for Write Amplification evaluation. Similar to [53], this model is developed using basic knowledge of flash operations based on the work of Menon et al. [81]. It is not based upon any empirical data. The model presented uses a greedy algorithm for erased block garbage collection and is cited by their later works. WAF values are presented as being between 3 and 5 for their page-mode FTL model.

Rajimwale et al. [96] mentions Write Amplification as a new manifestation of SSDs and a violation of the "unwritten contract" of block based storage devices. The authors mention Write Amplification of parity data in RAID applications. The authors estimate relative Write Amplification as a correlation between bandwidth and write data size. Limited data is provided, and there was no discussion of SSD modeling from measurements, as this appears to be out of scope with the author's goal of evaluating the "unwritten contract" with regards to SSDs in place of mechanical drives.

The work of Agrawal et al. [3] presents an equation of Write Amplification using an analytical model of a page-mode FTL using the greedy algorithm presented in [13]. No empirical measurements are made. Write Amplification is found to be solely a function of spare blocks set aside during the flash memory system manufacturing (*overprovisioning*), and not a function of page size nor other factors. Similar to Bux [13], WAF values are presented as being between 3 and 5.

Later work by Hu et al. [52] suggests that the limit of write performance is internal flash copying and propose an equation for a theoretical limit of Write Amplification. The authors demonstrate that Write Amplification is minimized for a greedy garbage collection and indicate that SSD performance is commonly expected to degrade over time. Again, WAF values are presented as being between 3 and 5. The authors suggest

that controlling Write Amplification may be accomplished with overprovisioning.

The work of Kim et al. [63] presents the effect of the ATA Trim command on Write Amplification. This command provides a means for a host to inform a storage device which sectors no longer contain useful data and no longer need to be actively managed, nor even persisted. The associated reduction in Write Amplification is expected as static data decreases the active size of the storage space. Later work of Frankie et al. [42] provides an analytic expression of Write Amplification considering the ATA Trim command.

The work of Park and Kim [91] presents the effect of data compression on Write Amplification. Their work considers a model of an FTL (zFTL) developed on a Linux and Windows platform using the NANDSim simulator. WAF estimations are extracted from the internals of the simulator.

The work of Soundararajan et al. [104] builds on the concept of Write Amplification and presents the concept of write-lifetime. The authors specifically state that Write Amplification "cannot be measured", but inferred indirectly using performance differences between workloads and a pure sequential workload in a fashion similar to prior work by Rajimwale et al. [96]. The author's specific work presents the concept of using an HDD as a cache for an SSD.

The work of Luojie and Kurkoski [73] provides an improved analytic expression of Write Amplification. It builds upon the work of [3]. Their work continues to find that Write Amplification is a function of overprovisioning only. WAF values from 1 to 4 are reported.

The work of Boboila and Desnoyers [12] focuses on flash memory system performance while considering actual workloads. The work uses a flash simulator (FlashSim) and analytic FTL models. Performance measurements are provided, but no

results of Write Amplification are given. Later work [28] presents an analytic expression of Write Amplification. Validation uses the FlashSim simulation model. WAF values from 3 to 13 are reported (depending upon Spare Factor i.e. overprovisioning).

The work of Templeman and Kapadia [106] provides an attack vector that seeks to wear flash storage systems by exploiting maximum Write Amplification. The work uses a basic lifetime model and extrapolates overall Write Amplification from data rates and rated flash lifetime. The work of Houdt [51] provides a model for predicting Write Amplification using simulation. New FTLs are proposed. WAF values from 2 to 20 are considered, depending upon pages per blocks and spare factor for these.

The work of Lu et al. [72] proposes an object-based file system that works with flash FTLs. The effect is to minimize Write Amplification. Their work uses a device driver to simulate the FTL and presents Write Amplification results based upon counts within the driver.

3.3.1 Write Amplification Measurements

As presented in the last section, the body of the work to date on Write Amplification is based exclusively upon analytical models. Estimations of Write Amplification values are based upon simulations using these idealized models. With a single exception noted below, no empirical measurements of Write Amplification has been reported in the literature. One author [104] even suggests that Write Amplification "cannot be measured".

The single presentation of empirical Write Amplification measurements is made by Jurenka [59] using an apparatus to monitor a single flash device in an MMC card subjected to discrete operations. This is indeed a valid, and novel, approach. However, the author's choice of technique and card (MMC) is not readily applicable

to multi-channel devices such as SSDs. Moreover, the author does not develop the results of their work into an empirical model or WAF equation.

3.4 FTLs: Trade-Secrets

While numerous models of flash translation layers (FTLs) have appeared in the literature over the years (recall Section 3.1), flash controller manufacturers typically do not disclose management details. This lack of disclosure has frustrated numerous researchers [45, 11, 4, 19, 12, 102] who have attempted to gain insight into flash system operations and their impact on host system performance. This situation has required analytical models of FTLs to be developed for Write Amplification simulations [53, 13]. While effective for their scope, this compromise limits the applicability of analysis to commercial applications.

3.5 Segments

Similar to the flash memory manufacturers shift towards addressing the largest customer segments, research papers have also shifted their focus. Early work by Chang et al. and Baek et al. [17, 6] specifically study flash memory systems in embedded applications. Later work shift focus towards SSDs in general applications [11] and enterprise applications [87]. The work of Pan et al. [87] is notable in that it reflects the enterprise propensity to value endurance at the expense of retention (recall Section 2.2) which is diametrically opposed to this work. Consideration of flash memory systems in embedded applications in general, and specifically retention, even in light of the trends in the flash memory highlighted (recall Section 2.3), has diminished in more recent years.

Chapter 4

Measurements and Modeling

FSAware is an enhancement to existing FTL designs that reduces overall flash memory system WAF by separately supporting the write requests associated with the file data and the file system overhead produced by host file system write activities. Unlike other FTLs in the literature that support a manner of data segregation, FSAware distinguishes between file data and file system overhead by understanding the file system installed on the flash memory system by the host system.

Demonstrating the effectiveness of FSAware requires models of FTLs that provide estimates of WAF. Unfortunately, FTLs in commercially available flash memory systems are regarded as trade secrets and even simple models are unavailable. Rather than leverage academically developed FTLs, this work aspires to maintain alignment with FTLs in commercially available flash memory systems. As such, the intermediate step of developing empirical models of commercially available FTLs is undertaken.

In this thesis, developing empirical models of FTLs required development of a novel instrumentation technique. We have developed FTLProbe, a strategy and toolset that studies the performance of a flash memory system in response to an

array of fine-grained measurements of individual transfer requests, while controlling the transfer address and size. Consistent performance impacts occurring in both time and frequency were interpreted in the context of FTL management activities. Leveraging this experimental technique, empirical (gray box) models of the first order effects of user data management and the second-order effects of the FTL's own data were constructed.

Development of the measurement and modeling techniques was a sizable undertaking. While necessary to ensure the applicability of our work with commercially available flash memory systems, the resulting activities were numerous and, at times, lengthy. These activities and their results are substantially presented in this work to act as a guide for future studies which adopt an empirical approach for commercial alignment. The below table is presented in each major section as a guide to the overall process and progress.

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

4.1 Glossary

In this Chapter, we utilize an array of terms used to identify the unique features significant to our measurements and modeling efforts. These terms are summarized below in Table 4.1

Term	Description
Fx	A periodic feature present in the performance measurements used to infer flash management activities
F1S	The most prominent periodic feature present in an array of sequential writes
F2S	The second most prominent feature present in an array of sequential writes
F3S	The third most prominent feature present in an array of sequential writes
F1R	The most prominent periodic feature present in an array of random writes
F2R	The second most prominent feature present in an array of random writes
F1OP	The first most prominent feature present in an array of sequential writes for over-provisioning measurements
Rx	A repeated write requests used for metadata associated with file system operations
R1	A write request continuously written to simulate metadata for a single sector
R1	A mode of write requests used to simulate metadata for a single sector, repeatedly written
R2	A mode of write requests used to simulate metadata for a two sectors, repeatedly written
R3	A mode of write requests used to simulate metadata for a three sectors, repeatedly written. Three sectors of metadata is equivalent to the metadata written for a file operation

Table 4.1: Measurement and Modeling Terms

4.2 Measurement Technique

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Flash memory system characterization may be performed with a commercially available disk, benchmarking a suite of software applications. Two popular choices include CrystalDiskMark [14] or IOMeter [57]. These tools interact with the target flash memory system using a large array of operations that are designed to measure average overall performance under induced steady state conditions and commonly provide only bulk performance measurements. Aside from being limited to coarse-grained measurements, these tools may work through the file system that is installed over the flash memory system. This approach incurs an extra layer of indirection which reduces the accuracy of the application's measurements. Lastly, these applications are host-bus agnostic and, as such, do not provide any support for either bus specific measurements or management functionality.

Instead of using available course-grained software applications, the FTLProbe software application was specifically developed to achieve the extended level of control

and the fine-grained measurements required for this study. This software application was designed to directly interact with the flash memory system under analysis with a minimum of overhead. Most significantly, this software framework can generate transfers of specifically controlled size and logical block address to and from the flash memory system being analyzed. Moreover, as the application was designed for a specific host bus, additional functionality beyond read and write accesses can be supported. This includes support of measurements beyond performance. This approach also facilitates support of any desired management functionality provided by the specific bus specification or any vendor commands provided by the flash memory system provider.

There are three general architectural designs that can be utilized to develop such an application. These three approaches are shown in Figure 4.1 and are:

1. User-Mode Service
2. Kernel-Mode Service
3. Direct I/O Access

The *User-Mode Service* architectural design takes advantage of services provided by the storage driver stack at the user-mode level. When these services are available, this approach can be desirable as it only requires development of the software application. The *Kernel-Mode Service* architectural design takes advantage of services provided by the storage driver stack that are only available at the kernel-mode level. This approach is more complex as it requires development of a support driver to export these services to the user-mode software application.

The *Direct I/O Access* architectural design requires the most development work as it requires development of a support driver that interacts with the host bus chipset

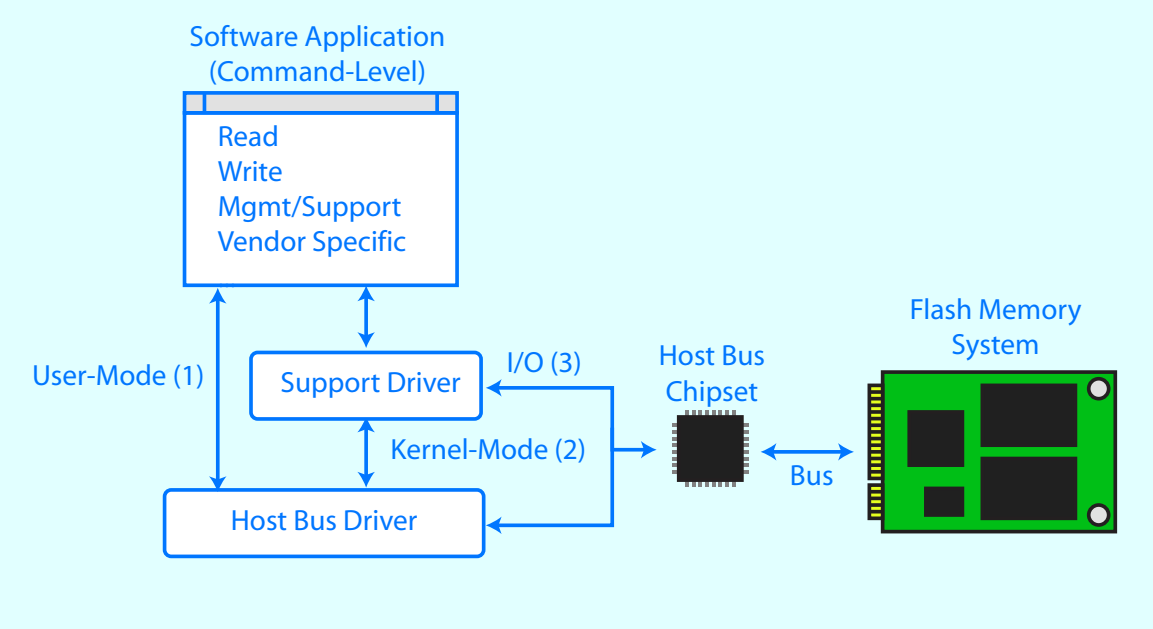

Figure 4.1: Software Application Designs

and exports this interaction to the user-mode software application. However, this approach is the most powerful as it is capable of utilizing all of the functionality supported by the host bus chipset. It is specifically not restricted to the services that the developers of the storage driver stack chose to export. When the storage driver stack is specifically restrictive about the functionality that it supports, this architectural design may be the only possible approach.

At the time of this study, flash memory systems using the SATA interface [101] are commonly used in many computer applications. As such, this style of interface was selected. Because of its wide array of SATA host bus adapters and development tools for both the software application and kernel-mode driver development, the Windows platform was additionally selected to host the characterization efforts of the selected SATA interface flash memory systems. These choices necessitated adopting the Direct I/O Access architectural design.

As mentioned, the developed FTLProbe software application facilitates direct execution of data transfers to and from the flash memory system under test. This direct execution includes specific control of data, transfer size, and logical block address of data transfers. As with benchmarking applications, large sequences of data transfers are utilized to induce conditions of steady-state. However, unlike benchmarking applications, the FTLProbe can track the performance of every transfer request. The single-command granularity provides fine-grained resolution in our performance measurements, which are required for this work.

Using the Direct I/O Access architectural design also minimizes overhead of the storage driver stack. Specifically, only a single user-mode to kernel-mode request is made for each transfer request. The support driver performs I/O accesses directly using x86 instructions. Round-trip single sector read transfers were measured to be an average of 458.8 us with a standard deviation of 7.01 us. Outliers due to operating software preemption of the measurement process were effectively eliminated by elevating the process and thread priority class to the maximum allowable. The absolute maximum measurement for a sample one hour test was found to be 623 us.

Because the developed application is bus aware, it can support functionality specific to the SATA interface bus. This functionality includes both measurement and management features. One specific area of measurement is provided by Self-Monitoring, Analysis and Reporting Technology (SMART), which is part of the ATA/ATAPI standard [5]. SMART was originally intended to provide disk health monitoring and warn of pending failures, but it has been expanded to include an array of monitoring activities including usage and operating conditions such as temperature and power cycles. Of particular importance to this thesis, SMART data

from some commercially available drives includes measures of internal flash write activity in a vendor-defined SMART attribute. Scaled by host write activity, which is often available in the standard SMART attribute 0xF1, direct measures of WAF can be attained while performance measurements are being performed.

The direct measurement of WAF serves as a means of validating the characterization techniques and the empirical FTL models developed in this thesis. In particular, this thesis develops techniques for constructing models of FTLs used in commercially-available flash memory systems using performance measurements. Among other uses, these models can be used to develop estimates of WAF. These estimates may then be compared with the WAF measurements obtained from SMART data. Alignment of these results helps to validate the FTL models and the techniques used for their development. These concepts are highlighted in Figure 4.2. Once validated, the techniques presented in this work may be used for empirical FTL model development for drives without supporting WAF measurements from SMART data.

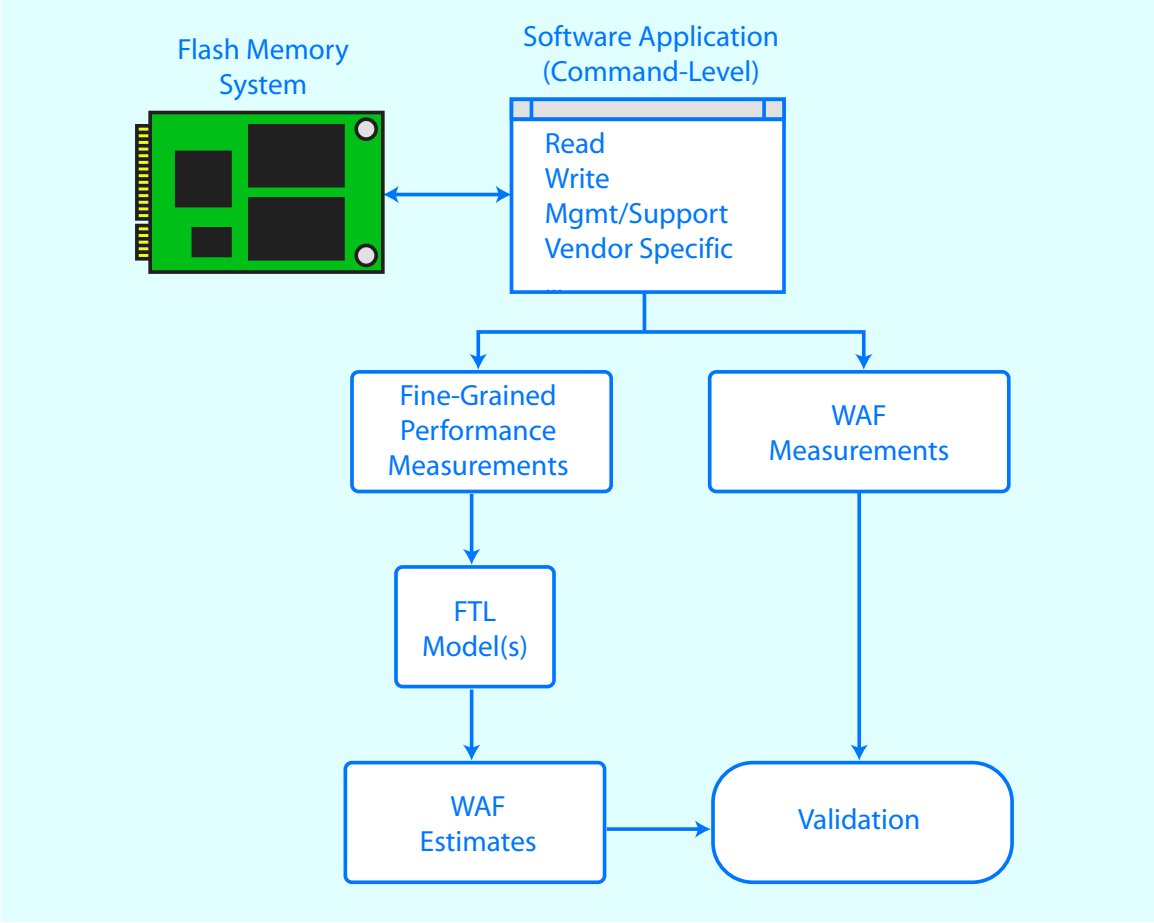

Figure 4.2: Characterization and Modeling Validation Process

4.3 Measurements (Block-Mode FTL, Single-Point)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Flash memory system characterization is performed using the FTLProbe software application presented in Section 4.2. Characterization consists of collecting an array of individual test points. Each test point consists of an extended battery of fixed-sized transfer requests. Standard testing was conducted for one hour. Extended tested, when required for improved resolution, was conducted for eight hours. Transfer size was varied from 512B (one sector) to 128KB (256 sectors - the limit of 28-bit ATA addressing), incremented in powers of 2. Both sequential and random tests for all transfer sizes were also conducted, independently. For random writes, sector addresses were randomly selected from the entire range of available sectors. Alignment of 4KB (eight sectors) was used for transfers of 4KB or larger and alignment of 512B (one sector) was used for transfers smaller than 4KB.

A commercially available 128GB drive using Toshiba MLC (TH58TEG8D2HBA) flash memory using a block-mode FTL was selected for characterization. Overall

performance for the array of test points used in this characterization was recorded using the developed software and is shown in Figure 4.3. WAF measurements based on SMART data were also made for these test points. These measurements are presented in Table 4.2.

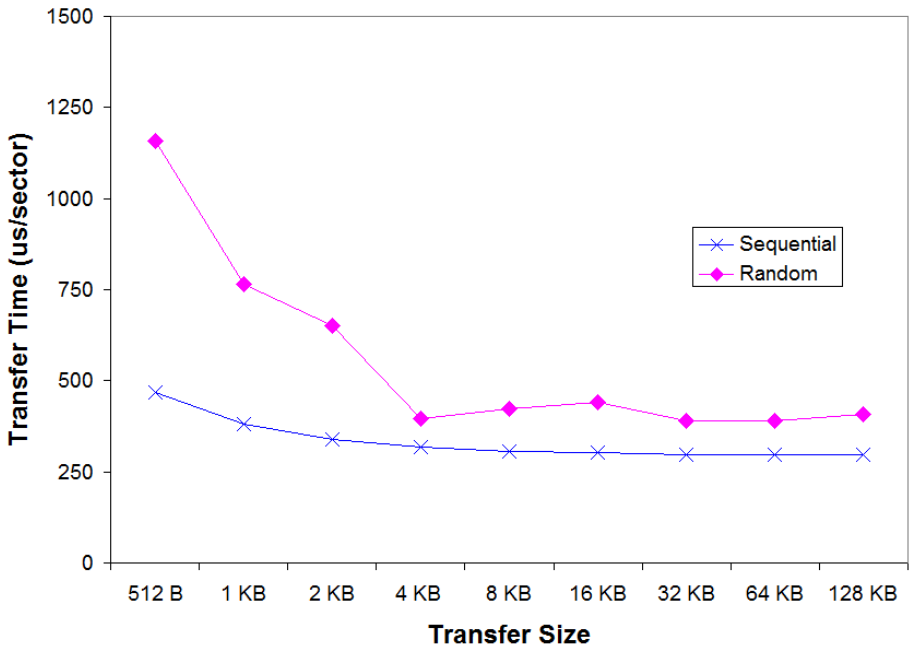

Figure 4.3: Overall Performance Measurements (Block Mode FTL)

Comparing the performance measurements shown in Figure 4.3 with the WAF measurements listed in Table 4.2 suggests that performance alone is not a good predictor of WAF. For example, the write performance difference for the 512B transfers is a factor of 3 between sequential and random transfer modes. However, the difference in WAF is around a factor of 160. Furthermore, the performance data shown in Figure 4.3 is misleading as it suggests overall trends that are not necessarily reflected in the WAF measurements. For example, the performance times for sequential transfers are observed to increase with decreasing transfer sizes. The WAF measurements

Transfer Size	WAF (Measured, Sequential)	WAF (Measured, Random)
512 B	1.0036	162.6
1 KB	1.0032	83.16
2 KB	1.0034	65.08
4 KB	1.0074	16.77
8 KB	1.0040	23.45
16 KB	1.0140	28.42
32 KB	1.0104	18.46
64 KB	1.0079	19.77
128 KB	1.0130	24.44
Average	1.0074	-
Average (4 KB)	-	21.89

Table 4.2: WAF Measurements (Block-Mode FTL)

in Table 4.2 confirm that no such trend in WAF is present. It is concluded that WAF estimates based upon performance scaling, a technique previously suggested by Rajimwale et al. [96], is not valid.

4.4 Measurements (Block-Mode FTL, Single-Point, Sequential)

Plots of performance measurements for all sequential test points (512B to 128KB) are shown in Figure 4.4 to Figure 4.12, respectively. As these figures illustrate, there is a baseline of nominal performance for all test points. There are however, notable instances of periodic performance impacts apparent in many of the plots. Considering the 32KB transfers shown in Figure 4.10 for example, there is a impact of around 10 ms every several thousand samples.

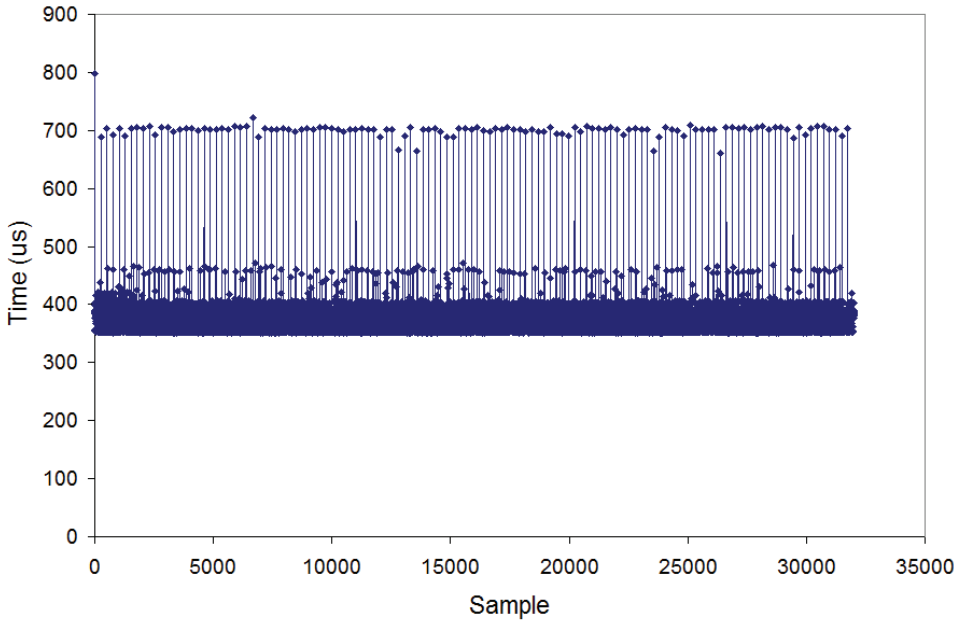

Figure 4.4: 512 B Performance Data (Block-Mode FTL, Sequential)

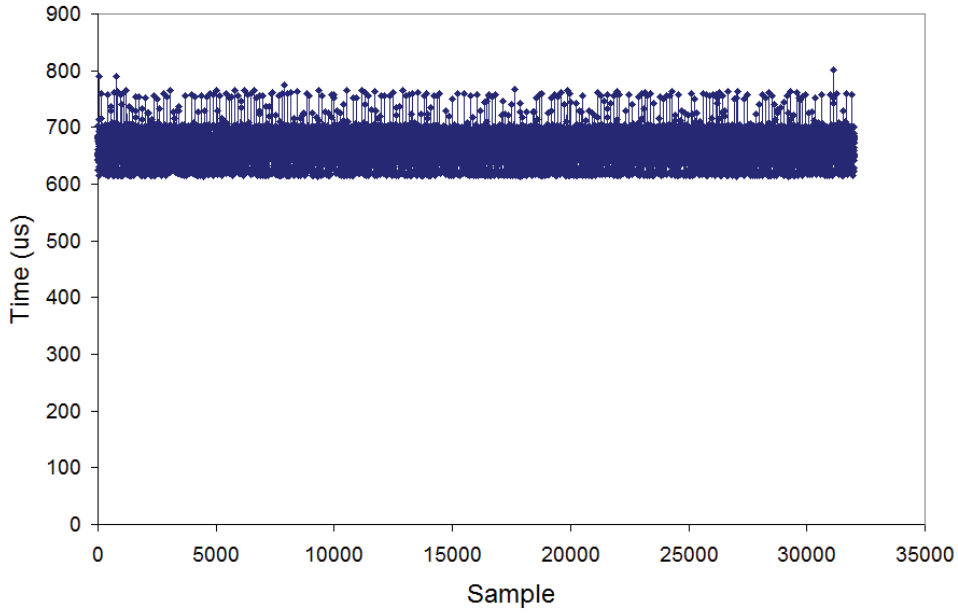

Figure 4.5: 1 KB Performance Data (Block-Mode FTL, Sequential)

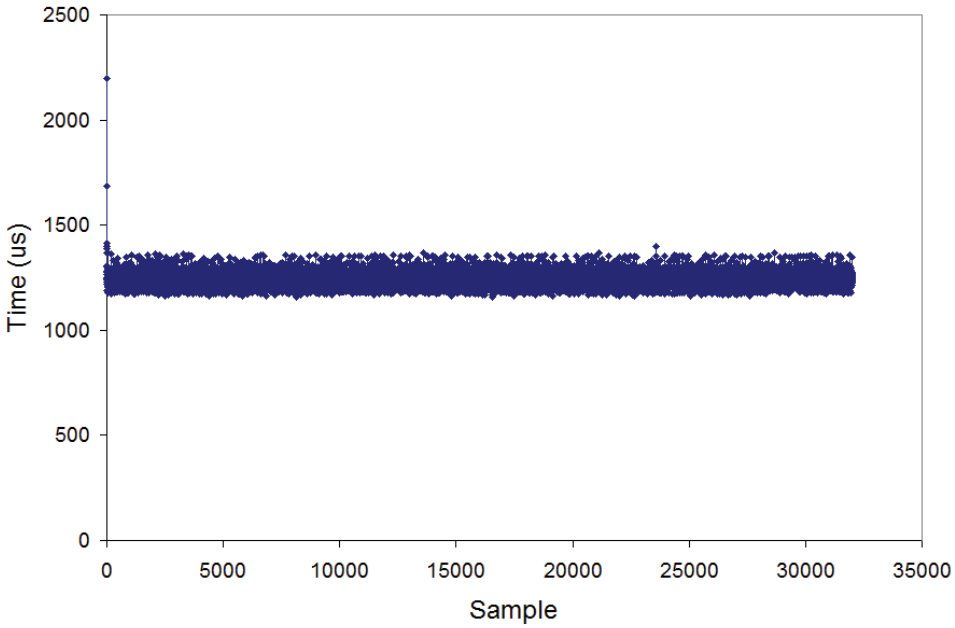

Figure 4.6: 2 KB Performance Data (Block-Mode FTL, Sequential)

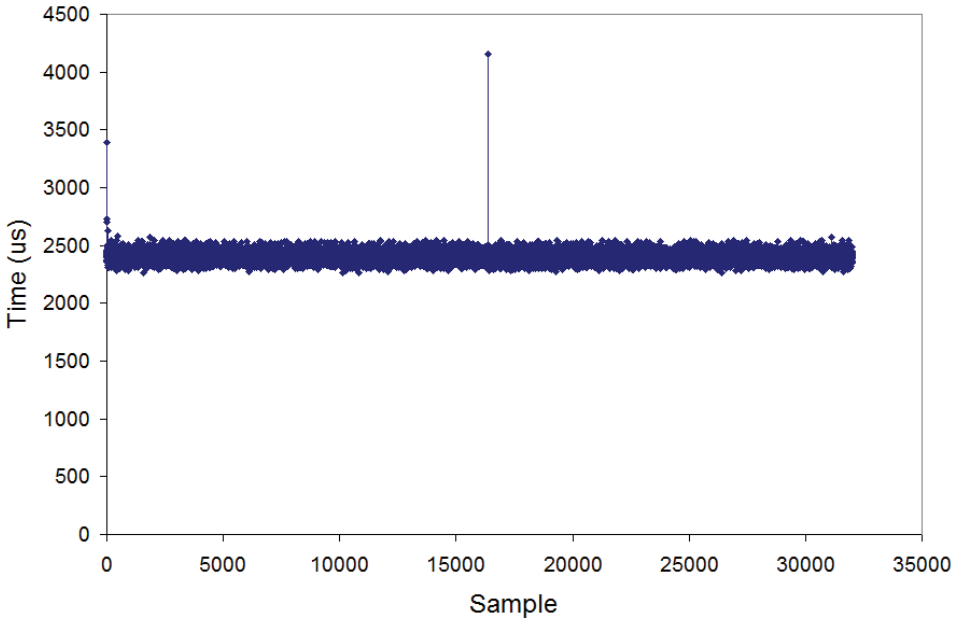

Figure 4.7: 4 KB Performance Data (Block-Mode FTL, Sequential)

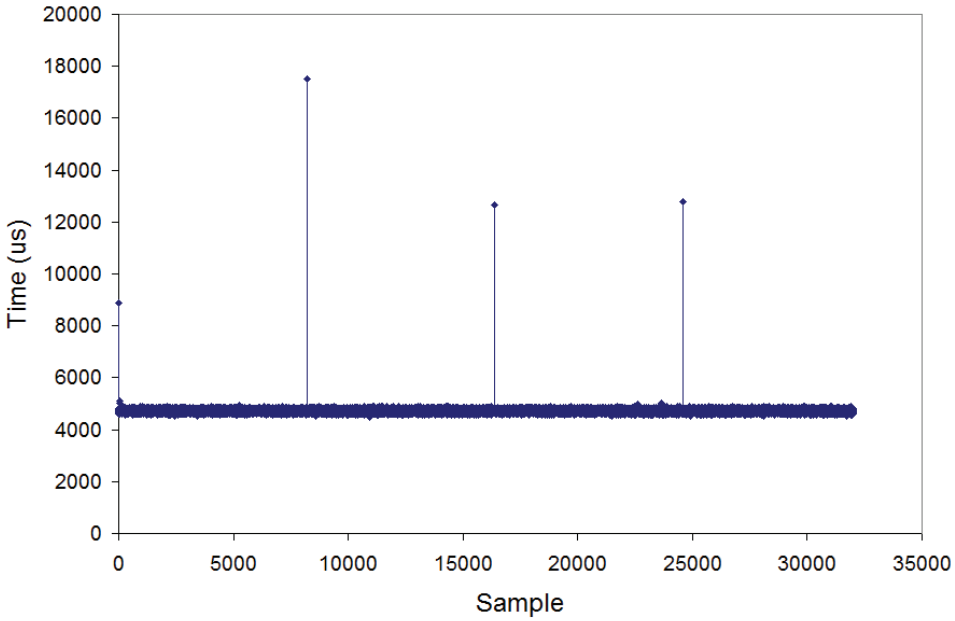

Figure 4.8: 8 KB Performance Data (Block-Mode FTL, Sequential)

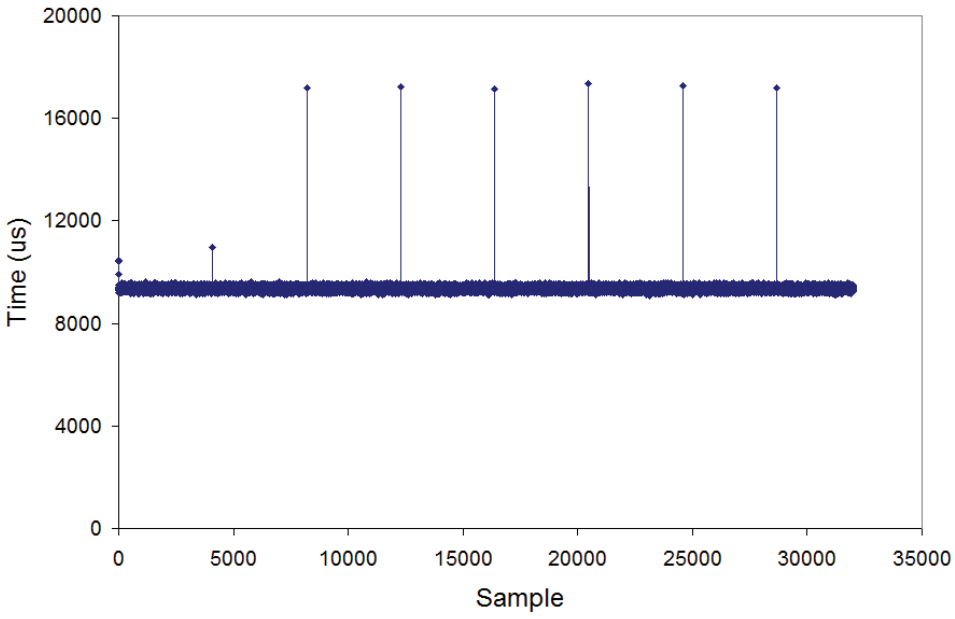

Figure 4.9: 16 KB Performance Data (Block-Mode FTL, Sequential)

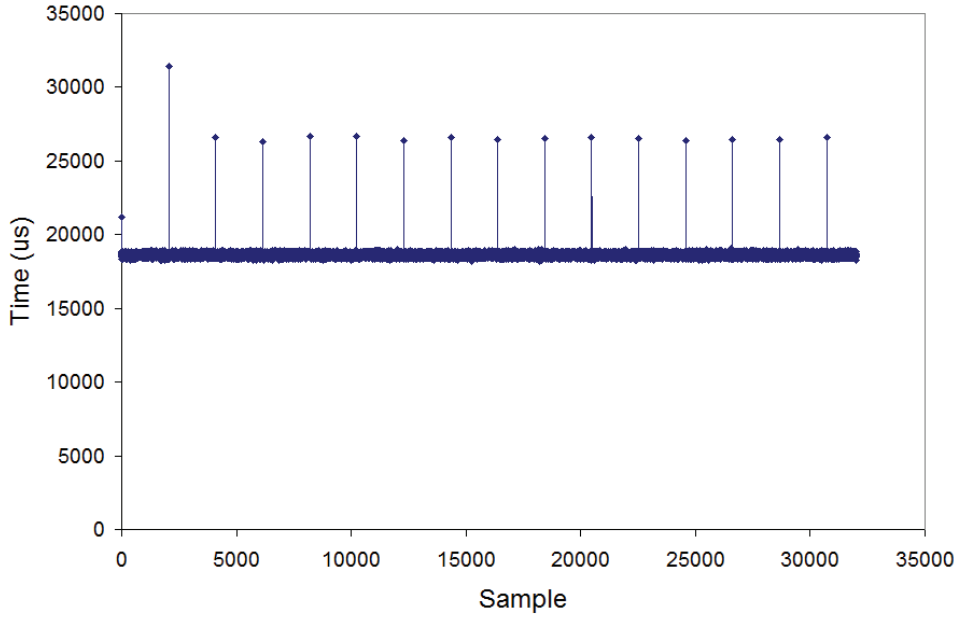

Figure 4.10: 32 KB Performance Data (Block-Mode FTL, Sequential)

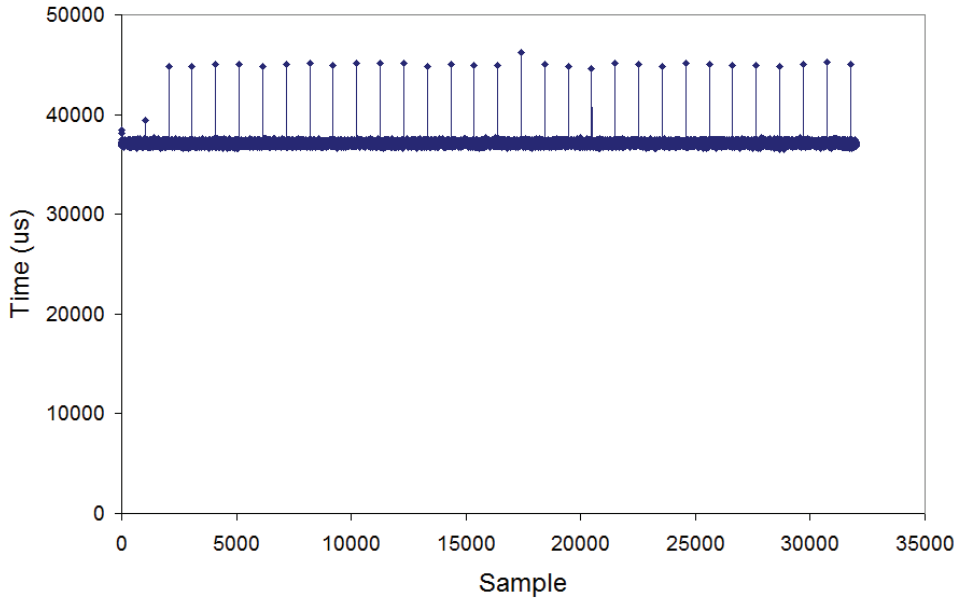

Figure 4.11: 64 KB Performance Data (Block-Mode FTL, Sequential)

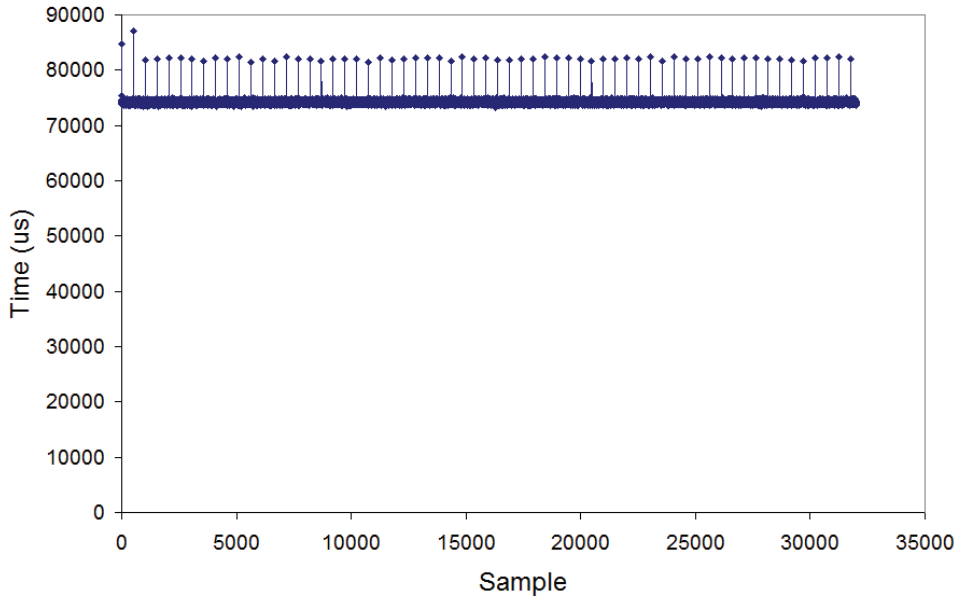

Figure 4.12: 128 KB Performance Data (Block-Mode FTL, Sequential)

4.4.1 F1S

The repeating 10 ms impact on performance can be clearly seen in Figure 4.10. This periodic impact is readily apparent in all of the plots for larger transfer sizes. The same periodic performance impact is present for smaller transfer sizes. However, the frequency of occurrence decreases with decreasing transfer size. Extended tests (8 hours) are required to recover the periodic pattern. This periodic performance impact is denoted as spectral feature *F1S*.

By filtering out collected data that dips below 5 ms of the average performance, the F1S occurrences can be effectively isolated. We plot filtered data collected during the sequential test points in Figures 4.13 to Figure 4.15. As these figures show, the frequency of F15 and the magnitude of its performance impact is seen over the one

hour of testing for all sequential test points.

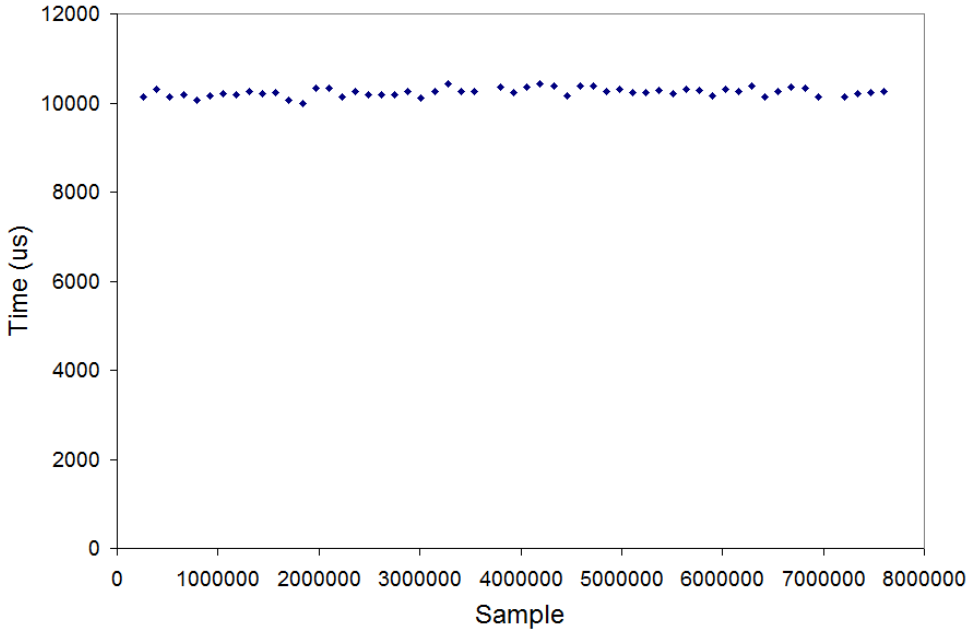

Figure 4.13: 512B Performance Data (Block-Mode FTL, Sequential, F1S)

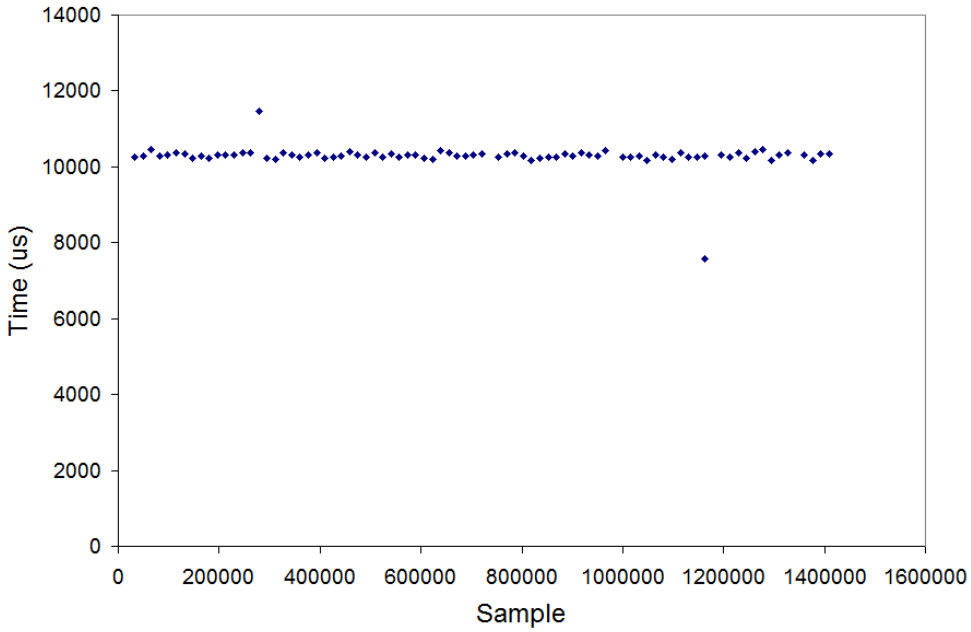

Figure 4.14: 4 KB Performance Data (Block-Mode FTL, Sequential, F1S)

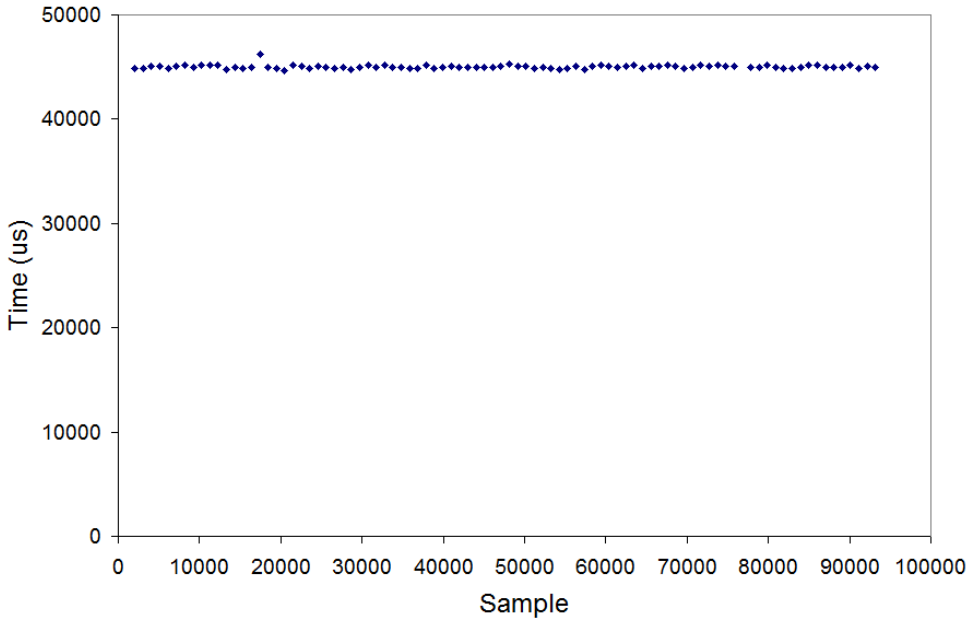

Figure 4.15: 64 KB Performance Data (Block-Mode FTL, Sequential, F1S)

F1S features were isolated for all of the sequential test points. The performance impact and frequency of occurrence of the F1S feature was captureed. This data is provided in Table 4.3. It should be noted that the while the performance impact of the F1S data points showed some degree of variation in magnitude, the frequency of the F1S occurrence was exactly as shown in Table 4.3. Only in rare cases was the frequency of occurrence observed to vary. Even for these cases, the frequency of the next F1S occurrence was such that the overall periodicity was maintained.

As Table 4.3 indicates, the product of the transfer size and the frequency of F1S events is 64 MB for all transfer sizes. Since the flash devices used in the characterized drive have 2 MB erase blocks and a value of 10000 us is on the order of two nominal erase block times (5 ms x 2), it is concluded that the drive is using a superblock

Transfer Size	Average Overall Measured Transfer Time	Average F1S Transfer Time	F1S Transfer Difference	Frequency of Occurrence	Total Transfer Size
512 B	468.2 us	10240 us	9770 us	131072	64 MB
1 KB	763.4 us	10160 us	9400 us	65536	64 MB
2 KB	1356 us	10200 us	8840 us	32768	64 MB
4 KB	2535 us	10280 us	7750 us	16384	64 MB
8 KB	4907 us	12600 us	7690 us	8192	64 MB
16 KB	9649 us	17260 us	7711 us	4096	64 MB
32 KB	19099 us	26540 us	7441 us	2048	64 MB
64 KB	37973 us	45020 us	7047 us	1024	64 MB
128 KB	75844 us	82100 us	6256 us	512	64 MB

Table 4.3: F1S Features

architecture for sequential write management that consists of 64 MB superblocks which internally consist of pairs of erase blocks. With this conclusion, it follows that the flash memory devices are organized as 16 parallel channels with 2 channels in series.

As Table 4.3 also shows, the difference between the F1S transfer times and nominal transfer times starts at around 10000 us and decreases for larger transfer sizes. This decrease is not believed to be due to any change in the flash management algorithms, but is instead attributed to the drive performing block erases in parallel with incoming data. As larger transfers require more time for data transfer, the impact of flash erases, when performed in parallel, is reduced.

4.4.2 F2S And F3S

Continued analysis of the fine-grained performance data suggests periodic features other than F1S are also present. An example of representative data collected for a sequential test point using 512B transfers is shown Figure 4.16

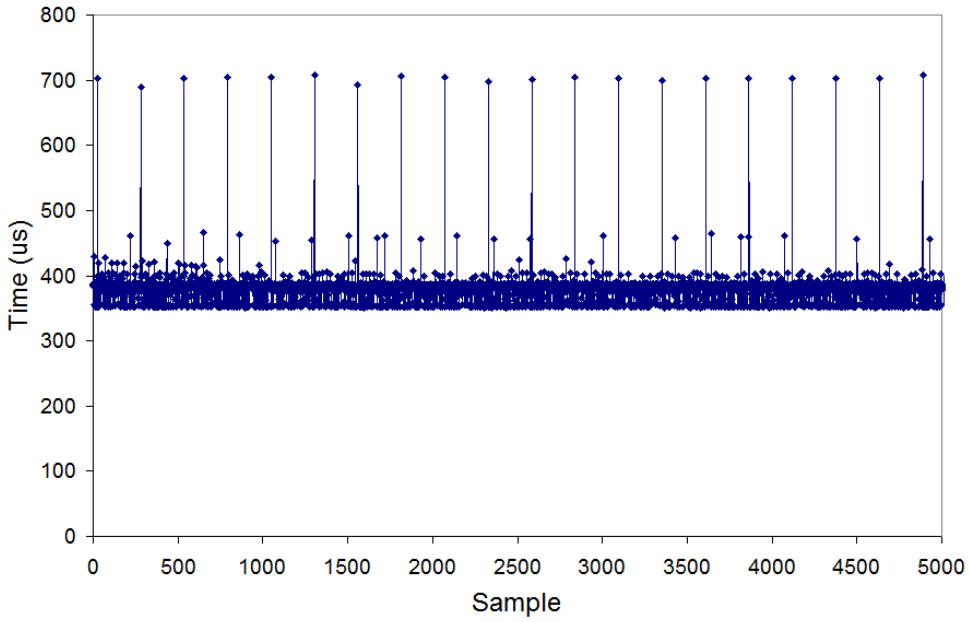

Figure 4.16: 512B Performance Data (Block-Mode FTL, Sequential, F2S And F3S)

Overall, two periodic features (in addition to F1S) can be observed. These spectral features are denoted as *F2S* and *F3S* respectively. The features of F2S are presented in Table 4.4 and the features of F3S are presented in Table 4.5. Because of the lower magnitude of F2S and F3S performance impacts, these features are not observable for larger transfer sizes. Since the flash controller in the drive has been observed to conduct management operations in parallel, it is expected that the operations associated with these features are present, but it is suspected that the performance impact of these operations is unobservable for the larger transfer sizes.

Transfer Size	Average F2S Transfer Time	Frequency of Occurrence	Total Transfer Size
512 B	1249 us	131072	64 MB
1 KB	1266 us	65536	64 MB

Table 4.4: F2S Features

Transfer Size	Average F3S Transfer Time	Frequency of Occurrence	Total Transfer Size
512 B	700 us	256	128 KB
1 KB	750 us (approx.)	128 (approx.)	128 KB

Table 4.5: F3S Features

The transfer time of F2S is on the order of 1400 us which is the specified time to program a single flash page (page program time) for the flash used in the sample drive. This suggests that at least one page of data (8KB) is being written for every 64MB of incoming data. Since user data transfer is entirely supported with transfers of superblocks, the F2S feature is attributed to overhead from the FTL. Since the flash is organized in 64MB blocks using 32 erase blocks, we expect that 32 pages (256KB) are being used as persistent storage for FTL overhead.

The transfer time of F3S is below a page program time. Since the flash in our study is configured as 8KB pages organized into 16 parallel channels (128KB), we deduce that F3S is the result of the overhead from transferring 128KB of data to the flash prior to page write operations. Building on this association, it also follows that

the flash drive uses a RAM buffer of 128KB to accumulate writes.

4.5 Empirical Model and WAF Equations (Block-Mode FTL, Single-Point, Sequential)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Based upon the measurements and observations in Section 4.4, an empirical model of the block-mode FTL that support sequential writes in the characterized drive is developed. This model is presented in Figure 4.17.

As Figure 4.17 illustrates, the characterized 128GB drive with a block-mode FTL is organized into 64MB superblocks. Each superblock consists of 16 erase blocks in parallel and 2 groups of parallel blocks in series. Incoming sequential write requests are cached in 128MB RAM. When the RAM is full, the flash controller transfers the data from RAM to flash pages of a specific buffer superblock. When the buffer flash superblock becomes full, the invalidated flash superblock assigned to the particular sector address range is erased and reassigned as an available spare superblock.

This mode of operation is well suited to supporting the previously mentioned

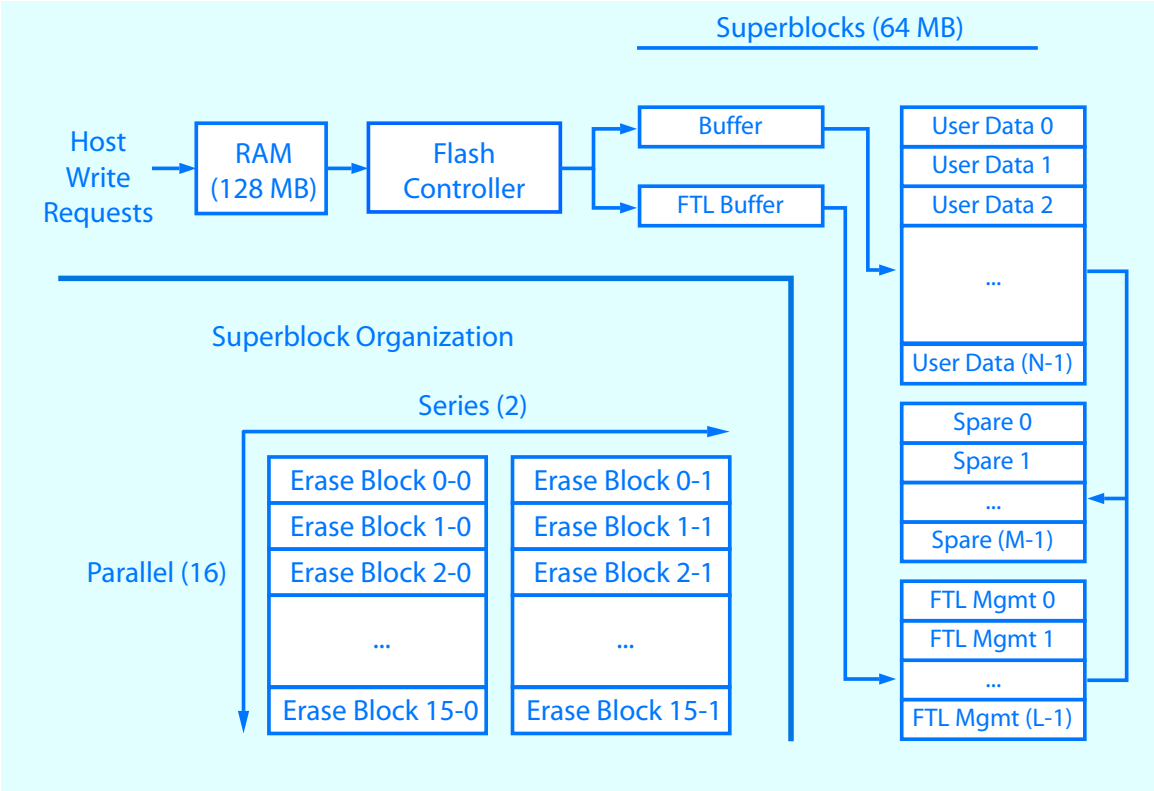

Figure 4.17: Block-Mode Sequential FTL Model

peculiarities of flash. In particular, flash is committed in whole pages and, ultimately, into whole erase block groups. Because the write requests are sequential, data can be logically aligned in each superblock. In this manner, only one superblock is invalidated for each newly written buffer superblock. No data transfer (copy) operations are required within the flash to support these operations. Only a switch garbage collection needs to take place when the superblock becomes full (recall Figure 2.11). This mode of operation can be supported with as little as one spare superblock. As no overhead is required for data support, the WAF for data is exactly 1.0 for the sequential mode of operation.

Figure 4.17 also shows the secondary effect of writing management data used for

the FTL to FTL Mgmt superblocks. As measured, the amount of data is 32 pages (256KB) per superblock (64KB). In a fashion similar to data superblocks, we can assume that the filled FTL Mgmt superblocks becomes part of a rotation of FTL data blocks in which old FTL blocks are eventually released to become new spare superblocks.

The WAF of the characterized FTL when supporting sequential operations is modeled as shown in Figure 4.18. As this equation shows, WAF is calculated to be 1.0040. As this equation has no dependence transfer size, this value is expected to be valid for all sequential transfers.

$$WAF_{Sequential} = \frac{Superblock\ Size\ +\ Management\ Data}{Superblock\ Size} \tag{4.1}$$

$$WAF_{Sequential} = \frac{65536\ KB\ +\ 256\ KB}{65536\ KB} = 1.0040 \tag{4.2}$$

Figure 4.18: WAF Model (Sequential)

The WAF measurements from Table 4.2 are plotted with the WAF estimations from the model shown in Figure 4.18 in Figure 4.19. As this figure suggests, we find good agreement between measurements and modeled valuess. Alignment of measured and modeled results using the method presented in Figure 4.2 is achieved and the technique of using periodic performance impacts to develop an empirical model of an FTL is considered valid.

It is noted that the measured WAF values shown in Figure 4.19 have an average of 1.0078. This is exceptionally well aligned with the value of 1.0075 which would be obtained from the WAF equation shown in Figure 4.18 if the management data were doubled to 512KB. Based upon this observation, we believe it is likely that the management data is indeed 512KB per superblock of user data (64MB). However,

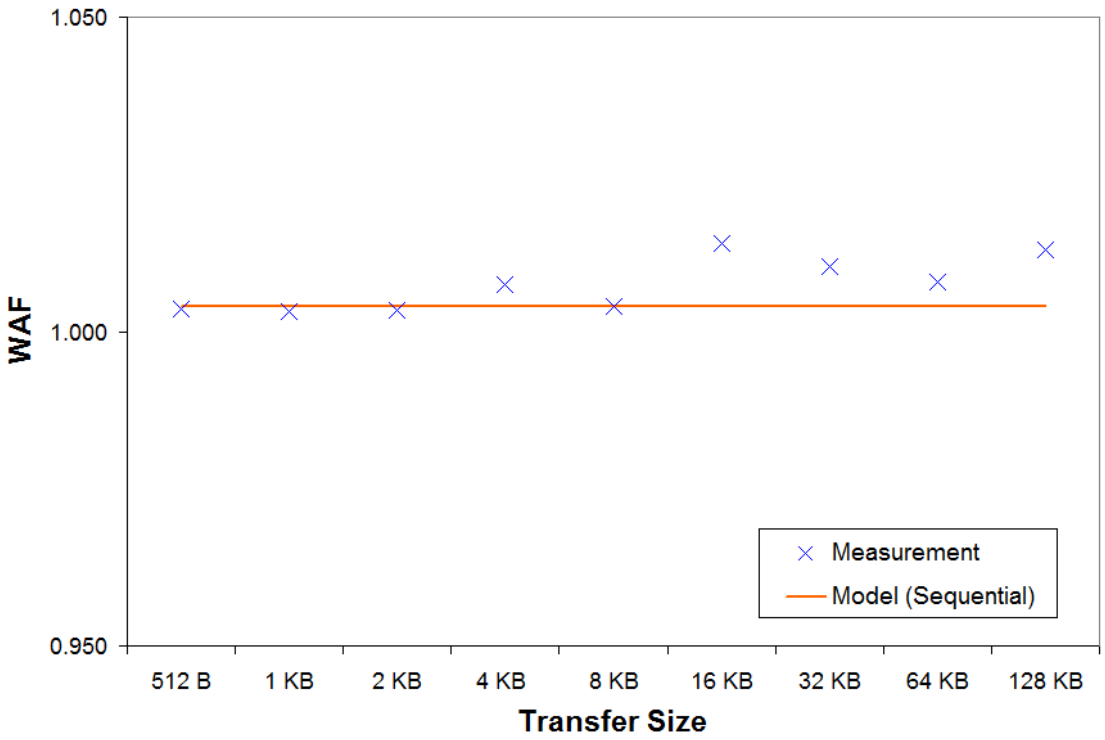

Figure 4.19: Block-Mode WAF (Sequential)

since on of the main objectives of this thesis is to determine a means for measuring WAF in the absence of SMART data, and experimental data alone provides no basis for doubling the amount of management data, the modeled values of WAF are left as above. We note here that operations conducted in parallel are more challenging to characterize using the technique presented in this work.

4.6 Measurements (Block-Mode FTL, Single-Point, Random)

For each random test point (recall Figure 4.3 and Table 4.2), the performance of every transfer over the entire one hour test was also collected. Representative data collected for the random test points is shown below in Figures 4.20 to 4.23.

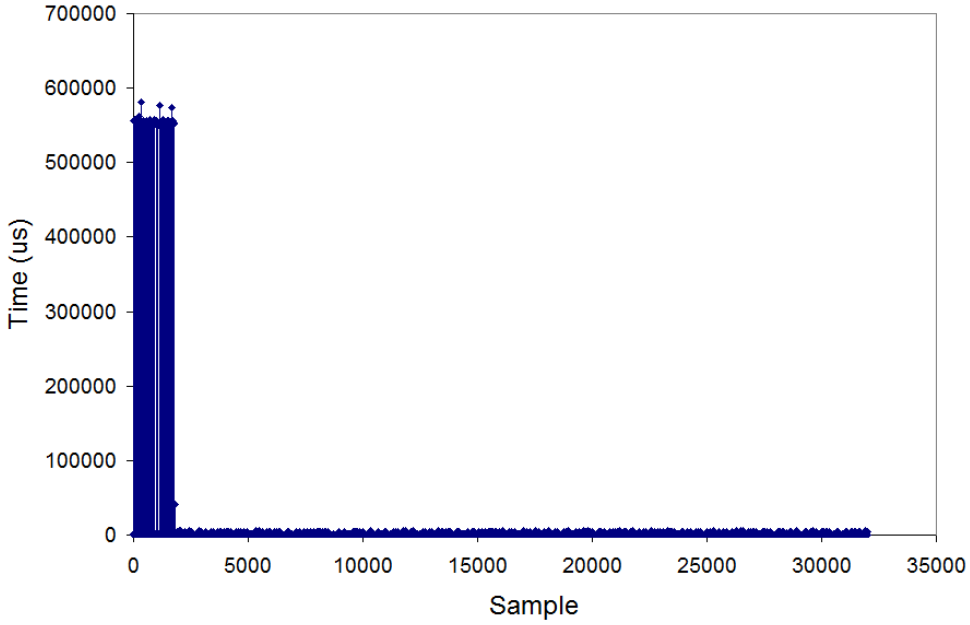

Figure 4.20: 1 KB Performance Data (Block-Mode FTL, Random)

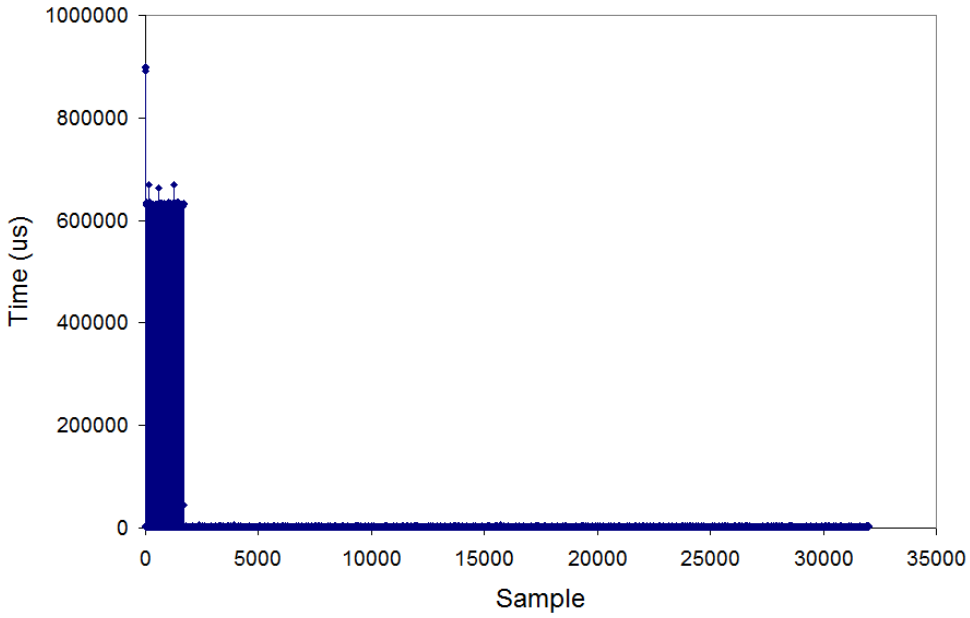

Figure 4.21: 4 KB Performance Data (Block-Mode FTL, Random)

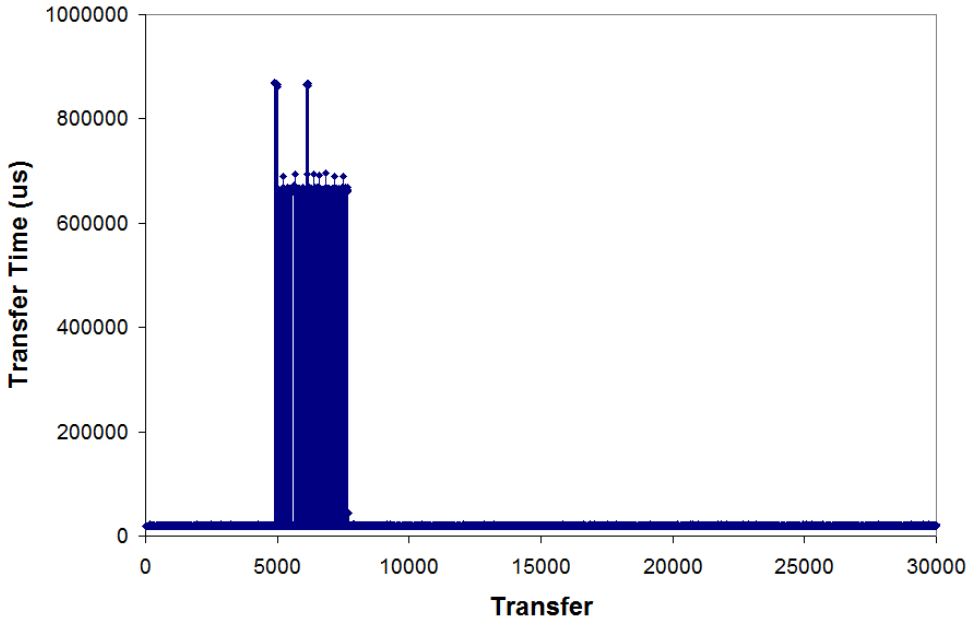

Figure 4.22: 32 KB Performance Data (Block-Mode FTL, Random)

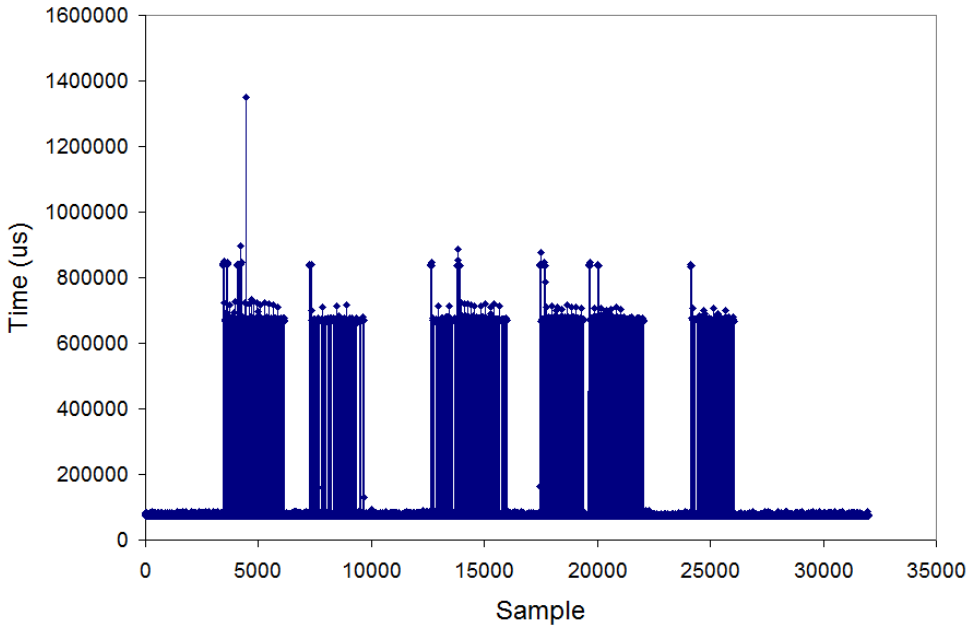

Figure 4.23: 128 KB Performance Data (Block-Mode FTL, Random)

4.6.1 F1R

As Figures 4.20 to 4.23 indicate, there are occasionally bursts of periodic events for which the data transfer takes longer than the baseline. In this work, these events are identified as spectral features. Spectral features may be conspicuous, as they are here, or they may be less apparent, as later presentation will be illustrate. These specific spectral features apparent above are denoted as *F1R* features. Inspection of data collected from other random test points confirms that bursts of F1R features are present in all random test points.

By filtering out the collected data to include only transfer times above 250 ms, the F1R occurrences can be effectively isolated. An example plot of all filtered data collected during the one hour random test point using 32KB transfers is shown in

Figure 4.24. As this figure shows, the magnitude of the performance impact of F1R features is consistently observed over the one hour of testing. However, unlike the previous F1S features, there is no apparent periodicity of the features.

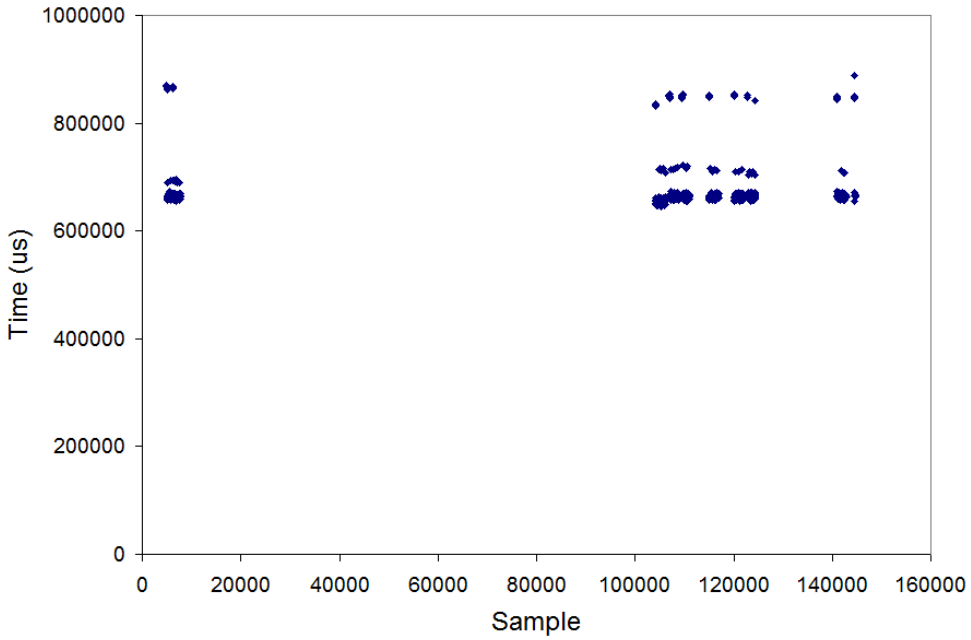

Figure 4.24: 32 KB Performance Data (Block-Mode FTL, Random, Filtered)

The Toshiba TH58TEG8D2HBA8C flash used in the characterized drive has 2MB erase blocks, with an expected erase time of 4ms, and 8KB pages with an expected programming time of 1.4 ms. For the 256 pages per block, a block copy (including the erase time) is expected to take 362.4ms. Two block copy operations would be expected to take 724.8ms. Since we have established that the flash drive being characterized uses erase blocks in pairs (recall Section 4.5), it is expected that block copy operations would also be performed in pairs. As such, the F1R features are assumed to use two block copy operations. As block copy may involve reads from multiple areas in multiple chips, it is not expected that operations can be supported in parallel as was

the case with sequential operations.

It is expected that block copy operations of the F1R bursts are associated with garbage collection events. Analysis of the data collected from all random test points shows that some bursts of F1R features have more events than others, but no burst has less than 119 events. Therefore, it is speculated that the bursts of more than 119 events are overlapping garbage collection events.

Because of the apparent lack of periodicity in the data from the one hour test points, an extended eight hour run of the 4KB random test point was conducted. Data from this point was also filtered to a minimum of 250000 us to extract the F1R features. A plot of all F1R features for the sampled data is shown in Figure 4.25.

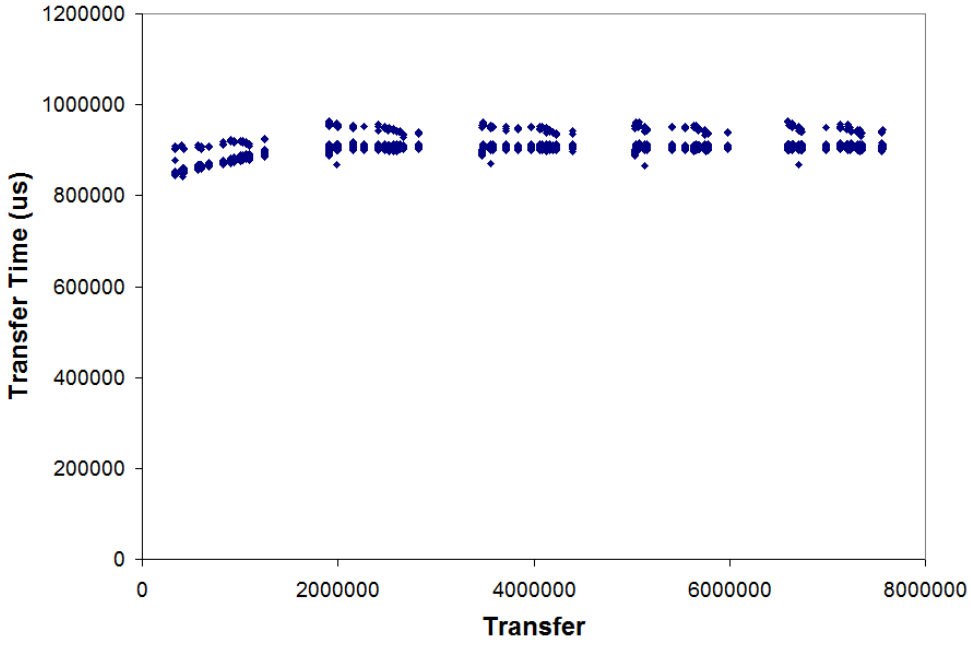

Figure 4.25: 4 KB Performance Data (Block-Mode FTL, Random, Filtered)

As Table 4.6 also shows, the number of F1R events in a region is consistently 1908. Scaled by the 119 F1R peaks per garbage collection event, we can conclude that there

F1R Region	F1R Region Start (Transfer)	F1R Events	F1R Region Size (Transfer)	Sectors Per Transfer	F1R Region Size (GB)
1	2,019,711	1908	1,530,800	8	5.840
2	3,550,511	1908	1,476,058	8	5.630
3	5,026,569	1908	1,509,494	8	5.758
4	6,536,063	-	-	-	-
Average					5.742

Table 4.6: F1R Region Size

are 16 garbage collections occurring in each F1R region. The value is equivalent to the number of parallel flash channels. As the individual garbage collections do not have any apparent periodicity within an F1R region, we can conclude that the garbage events collections are independent. Based on this logic, we conclude that the drive is divided into 16 independent management regions and that each garbage collection is for one of these regions.

User space of the tested drive is 250,069,680 sectors or 119.2GB. If the drive is assumed to be divided equally into 16 independent management regions, each region consists of 7.45GB. If it is further assumed that each group receives 1/16 of the 5.7GB transferred per F1R region, then we can conclude that 364.8MB is written to each group for each garbage collection. For 64MB superblocks, this implies that there are 5.7 buffer superblocks, on average, among the 16 groups to support management of random writes. Considering that 364.8MB of data causes 119 F1R peaks in which two block copy operations are performed, the WAF can be calculated as shown in Figure 4.26.

From analysis of sequential transfers, a superblock is known to be 64MB. The

$$WAF_{Random,User} = \frac{Flash\ Data\ Writes\ +\ Host\ Data\ Written}{Host\ Data\ Written} \tag{4.3}$$

$$WAF_{Random,User} = \frac{119\ \times\ 64\,MB\ +\ 364.8\,MB}{364.8\,MB} = 21.88 \tag{4.4}$$

Figure 4.26: WAF Equation (Random)

Transfer Size	Transferred Sectors	Overhead Sectors	Write Amplification (Calculated)
512 B	471,859,200	3,110,329	151.7
1 KB	363,462,656	4,713,074	77.12
2 KB	447,510,400	5,522,004	61.12
4 KB	140,115,968	9,057,288	15.47
8 KB	187,564,032	8,517,264	22.02
16 KB	218,890,240	8,142,752	26.88
32 KB	158,597,120	9,258,944	17.13
64 KB	169,607,168	9,206,016	18.42
128 KB	203,030,528	8,842,240	22.96
Average (4 KB - 128 KB)			21.89

Table 4.7: WAF Calculations Based Upon F1R Counts

count of F1R features were measured and the estimates of overhead from block copy operations were be determined. WAF calculations can also be determined by scaling this value by the amount of sectors written. These values are shown in Table 4.7.

As Table 4.7 shows, the WAF is on average 21.89 for transfer sizes of 4KB and larger. This aligns quite well with the value predicted by Figure 4.26. From these observations, we can conclude that the FTL is indeed managing random write operations as presented above. Moreover, based upon the 4KB threshold, we conclude that the FTL is managing using granularities of 4KB.

For transfer sizes smaller than 4 KB, the calculated WAF is observed to increase by 2x for each 2x reduction is transfer size. Based upon the 4 KB FTL granularity, it is expected that this impact in WAF is incurred by copying data internally to ensure that the 4 KB allocation units are 4 KB aligned.

4.6.2 F2R

Further analysis of our fine-grained performance data indicates that one other periodic feature is present. An example of representative data collected for a random test point using 512B transfers with F1R features filtered out is shown in Figure 4.27.

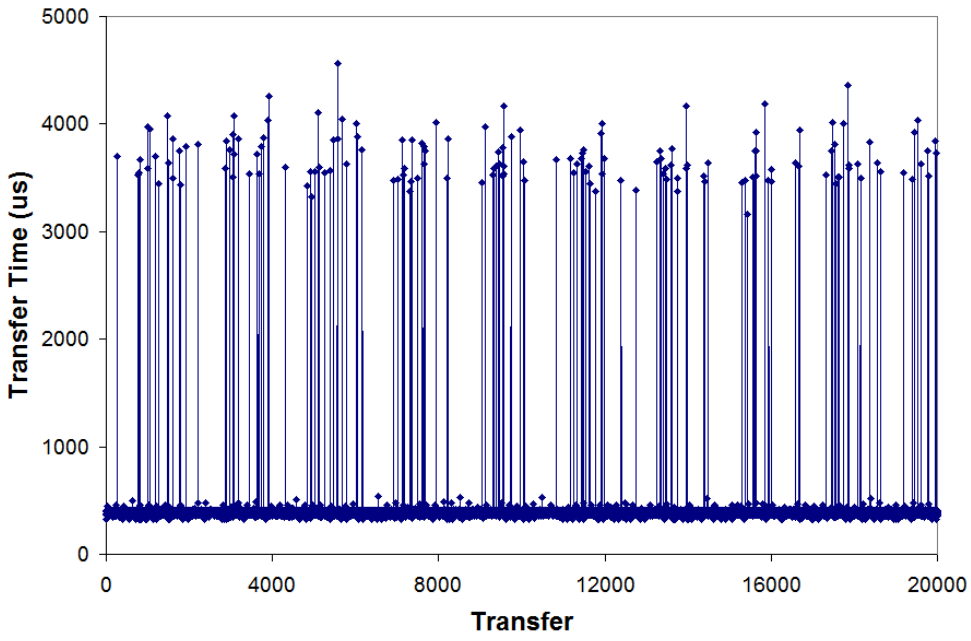

Figure 4.27: 512 B Performance Data (Block-Mode FTL, Random, Filtered)

The features beyond the nominal baseline performance shown in Figure 4.27 are denoted F2R. Analysis of the F2R peaks is overlapped by events that are apparently associated with garbage collections. An example of this overlap is shown in

Figure 4.28.

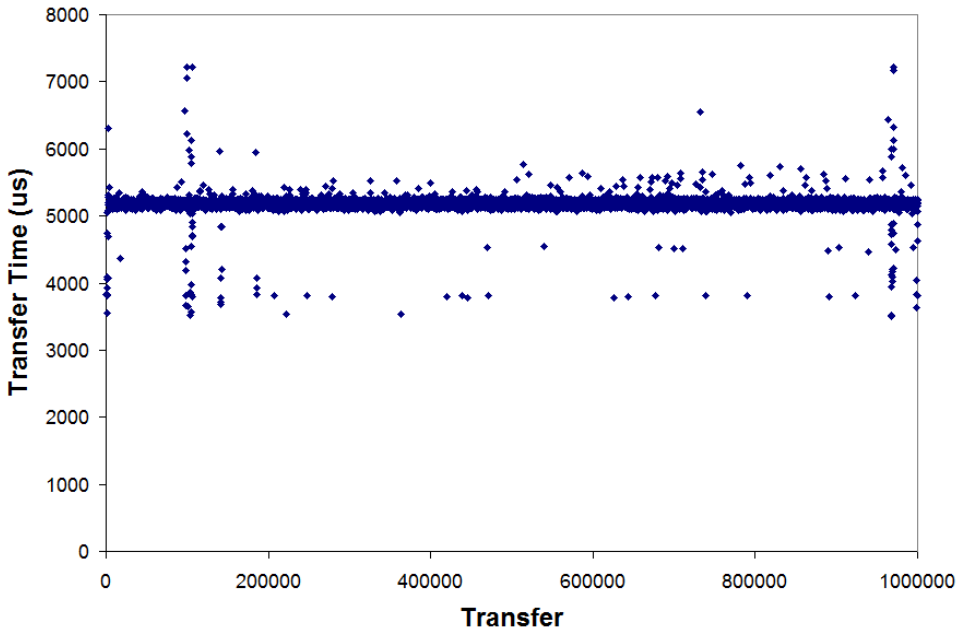

Figure 4.28: 4 KB Performance Data (Block-Mode FTL, Random, Filtered)

As Figure 4.28 highlights, the F2R features form a consistent horizontal trend around a transfer time of 5200us. However, a few spectral features are also apparent. These features align in time with the F1R events and are attributed to lesser significant events during garbage collection such as individual page writes and block erases. These effects can be isolated by choosing a narrow window of transfer time for events to assign as F2R features.

It is also noted that the F2R features do not exhibit periodicity. This is expected since there are 16 independent management regions used to support random write requests. Because of the independence of these regions, it is expected that the F2R features align with the periodicity of these feature. However, as these relationships

Transfer Size	Average F2R Transfer Time	Total Transfers	F2R Counts	Frequency of Occurrence (User Data)
512 B	3650 us	3,110,329	24,251	64.1 KB
1 KB	4280 us	2,356,537	20,671	114 KB
2 KB	4580 us	1,380,501	14,715	187 KB
4 KB	2790 us	1,132,161	8,825	525 KB
8 KB	2870 us	532,329	8,117	524 KB
16 KB	3100 us	254,461	7,764	488 KB
Average	3650 us			

Table 4.8: F2R Features

may not be deduced using performance measurements alone, we also consider overall averages. Using this approach for the narrow transfer time windows described above, the F2R features are identified and summarized in Table 4.8. Similar to the effect observed with the F2S features, the F2R features appear to be obfuscated by background processing for transfer sizes larger than 16KB and are not presented.

The transfer time of F2R features is on the order of two page program times (2800us). For transfers sizes 4KB or larger, this suggests that 16KB of data is being written for every 512KB of incoming data for transfer sizes. As might be expected from the 4KB granularity, the overhead scales by a factor of two for each level of sub-4K transfers.

4.7 Measurements (Block-Mode FTL, Over-Provisioning)

Given the expected exclusive dependence of Write Amplification [4] on over-provisioning, the values of over-provisioning used for the drives in this work are need if we want to compare and contrast the predicted values of Write Amplification with the measured ones. Over-provisioning is established by a low-level formatting performed during the flash memory system's manufacturing. It is a measure of the amount of User Data Space that is directly accessible and the Total Data Space available, as defined in Figure 4.29.

$$Over - Provisioning(OP) = \frac{Total\ Data\ Space\ -\ User\ Data\ Space}{User\ Data\ Space} \tag{4.5}$$

Figure 4.29: Over-Provisioning (OP)

User Data Space is the size of the flash memory system that is available to the user. This can be determined directly from the drive itself. For SATA drives, this value can be readily determined using the ATA Identify Drive command.

In concept, the size of the Total Data Space can be determined by performing random write transfers to a clean drive and observing the point at which performance initially decreases. More specifically, when the drive is clean, such as following an ATA Secure Erase operation, all of the data space (Total Data Space) is available to directly receive data to support write requests. However, when all of the data space becomes full, the flash memory system will need to perform periodic garbage collection to support additional writes. The FTLProbe technique of measuring fine-grained write performance can readily highlight this transition point.

4.7.1 Random Writes

Using the conceptual technique presented in the last section, a 128GB drive with block-mode FTL is completely cleaned using an ATA Secure Erase operation. As the drive is fully erased (clean), all of the data space (Total Data Space) is ideally available as spare blocks. As such, random 4KB transfers can be readily accepted with no performance impacts beyond typical transfer times of 2.5ms (see Figure 4.30).

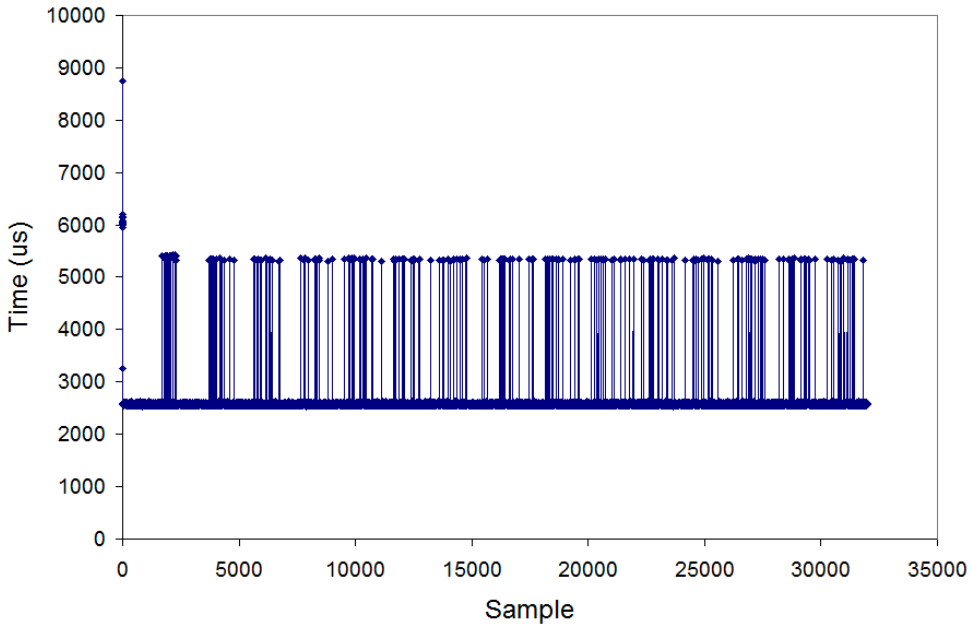

Figure 4.30: 4 KB Transfers (Block-Mode FTL, Random, Clean)

The performance impact of 3ms above baseline performance shown in Figure 4.30 are associated with page writes and considered to be typical overhead to support random write requests. The 1855 transfers that occur without any delay are associated with transfers being cached in the DRAM used in the flash memory system. For a 4KB transfer size, the 1855 transfers are associated with 7.24MB.

Ideally, once all of the spare blocks are consumed, the drive will need to perform garbage collection to sustain additional transfers. At this point, performance impacts will become apparent (see Figure 4.31).

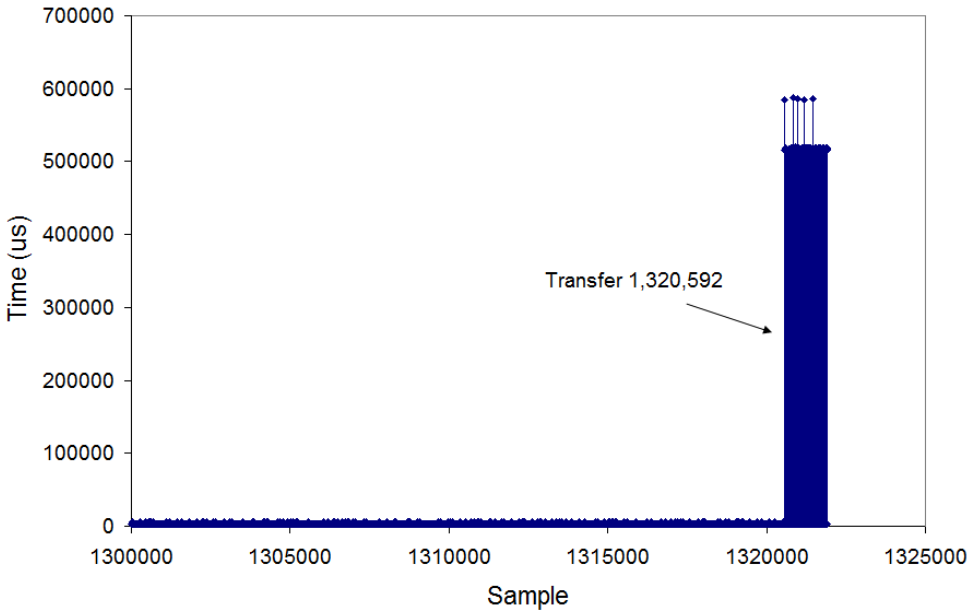

Figure 4.31: 4 KB Transfers (Block-Mode FTL, Random, Initial Saturation)

As Figure 4.31 shows, performance impacts become apparent at transfer 1,320,592. For a 4KB transfer size, the 1,320,592 transfers are associated with 5.034GB. This is a small portion of the overall 128GB drive capacity.

Instead of observing the Total User Space, the random transfers have shown the size of the data space that the block-mode FTL has allocated for caching writes as a page-mode FTL (recall Section 2.4). This measurement is useful as it provides a secondary measurement of the FTL's sixteen management areas. It aligns with the value of 5.742GB previously computed using our measurement technique when using a larger number of samples (recall Section 4.6 and Table 4.6 in particular).

4.7.2 BAST and FAST

Recall from Section 3.1 that hybrid block-mode FTLs represent a transition between pure block-mode FTLs and pure page-mode FTLs. Specifically, the hybrid block-mode FTLs could offer some of the benefits of a page-mode FTL without all of the computational and memory resource costs which were simply not practical until recently.

The original hybrid block-mode FTL presented in the literature was by Kim et al. [64] and was later identified as *BAST*. As described, the BAST FTL is a block-mode FTL which includes a small region of page-mapped blocks for caching random write requests. This is similar to the block-mode FTL on the commercially available flash memory system used in this study. Specifically, small regions for caching random writes as page-mode FTL can be clearly seen in Figure 4.30 as no performance impacts occur during the initial transfers. As such, it is concluded that BAST is a conceptual basis for the block-mode FTL used in this study.

However, the flash memory system used in this thesis utilizes distinct management regions. The description of management regions is not present in the work of Kim et al. [64], nor is it discussed in prior work discussing hybrid block-mode FTLs.

The manufacturers of the flash memory system with the block-mode FTL used in this thesis invariably concluded that dividing the flash into sixteen management units was a good design choice for their constraints and/or expected usage models. This observation again highlights the choice of this work to focus on empirical models of commercially available flash memory systems as being viable.

The block consolidation events shown in Figure 4.31 were observed to increase in frequency as the flash memory system transitioned from sequential organization of data into random organization of data. This thrashing of log blocks during the

transition is highlighted by later work [107] as a disadvantage of the BAST FTL.

Later hybrid block-mode FTLs, notably FAST [70], proposed to avoid this thrashing by using separate pools for random and sequential data. Ideally, these pools of segregated data would be better suited to handle the different write models without performance impacts from the transition between them. As the flash memory system in this work shows a distinct thrashing from even a simple transition from sequential to random organizations, it is concluded that it is not based upon these later designs.

4.7.3 Sequential Writes

Instead of using 4KB random transfers, attempts are made to estimate the size of the the Total Data Space using 128KB sequential transfers. This strategy is the same as before, except that the drive is written using sequential transfers after the ATA Secure Erase operation. Using this technique, the sequential 128KB transfers can be readily accepted with no performance impacts beyond the typical observed transfer times of 46ms (see Figure 4.32). In particular, no consolidation events are observed to occur.

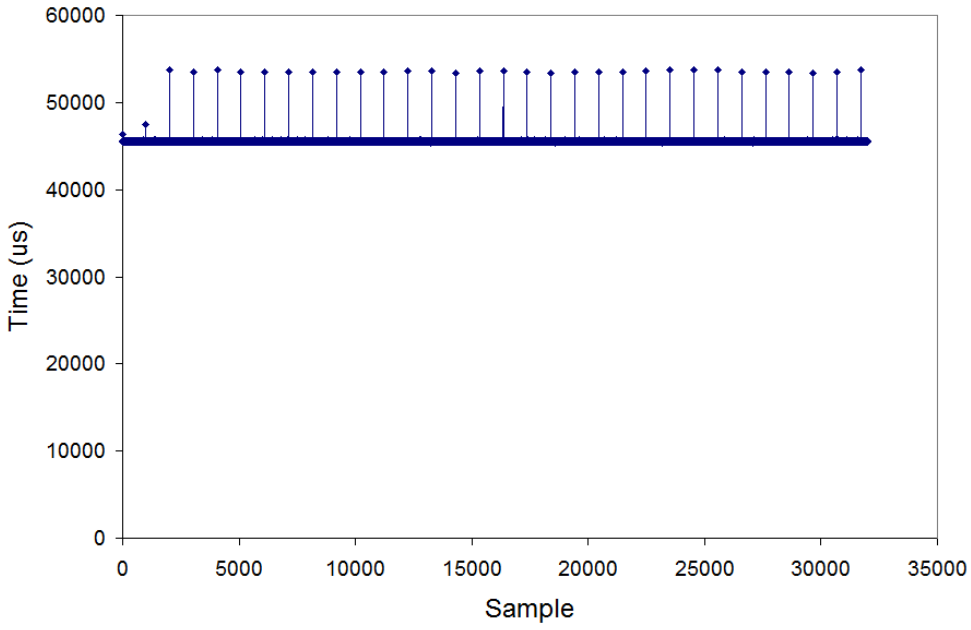

Figure 4.32: 128KB Transfers (Block-Mode FTL, Sequential, Clean)

The performance impact of 8ms above baseline performance that is shown in Figure 4.32 are periodic at 1024 write requests (128MB) and are associated with two parallel block erases that are required to support normal sequential write transfers. Analysis of write performance over multiple drive writes finds that this behavior continues indefinitely. As no inflection in performance impacts occurs over time, the technique is concluded to be unable to determine over-provisioning directly. However, further analysis of the performance data identifies a non-frequent periodic feature. This feature occurs around 30ms above nominal. One such feature, called *F1OP*, is shown in Figure 4.33.

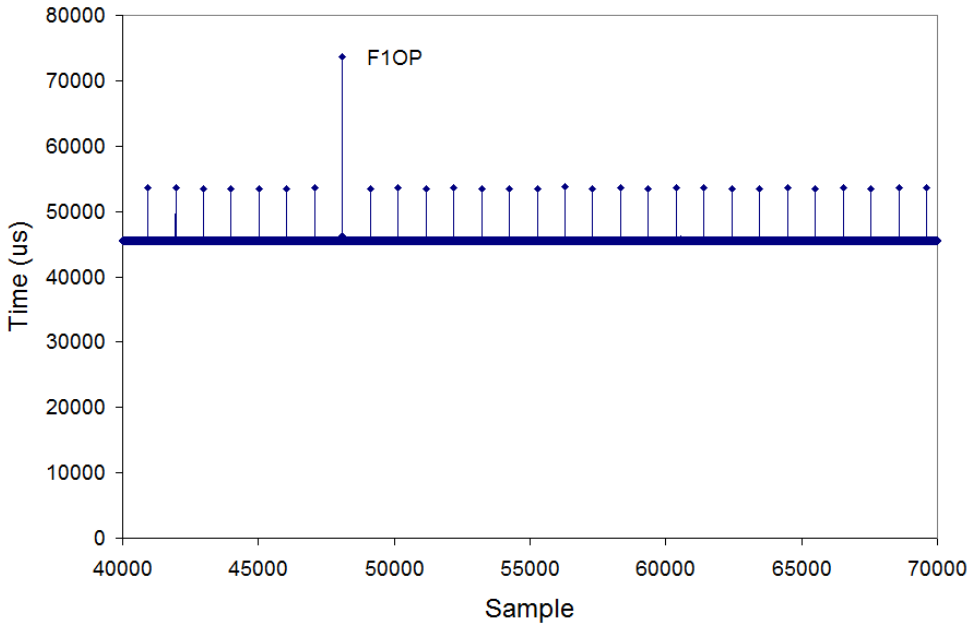

Figure 4.33: 128 KB Transfers (Block-Mode FTL, Sequential, F1OP)

Overall, sixteen F1OP features are observed per drive write. As the block-mode FTL has been determined to use sixteen management units (recall Section 4.6), we can conclude that each of these features is associated with the overhead for a full management unit.

Considering that the drive has a data area of 250,069,680 sectors, there is data space for 976,835 128KB transfers. The average periodicity of F1OP features is 1,008,922 transfers. Considering WAF is measured to be 1.009 for these transfers, it is inferred that 1,018,228 transfers of size 128 KB are written to flash. Over-provisioning is estimated to be 4.24% (see Figure 4.34). This value compares well to the value of over-provisioning of 7% commonly reported for commercially available drives [65]. This makes sense given that bad blocks are commonly identified during manufacturing and marked as unavailable so that true over-provisioning is typically

lower than the ideal.

$$OP = \frac{1,018,228 \ - \ 976,835}{976,835} = 4.24\% \tag{4.6}$$

Figure 4.34: Over-Provisioning (Block-Mode FTL)

4.8 Empirical Model and WAF Equations (Block-Mode FTL, Single-Point, Random)

Based upon the measurements and observations in Section 4.6, an empirical model of the block-mode FTL that supports random write requests in the characterized drive is developed. Our model is presented in Figure 4.35.

As Figure 4.35 illustrates, the 128GB drive is organized into management regions. All of the available user data superblocks are divided among the management regions. Each management region includes a collection of buffer blocks for incoming write requests. For the characterized drive, there are 16 management zones, 119 user data superblocks per management zone, and an average of 5.7 buffer blocks per management zone. As with the sequential management model, superblocks are 64MB. The random management FTL supports writes in whole and aligned 4KB units. For write requests smaller than 4KB, data is copied within the flash to ensure that the 4KB units remain whole and aligned.

As discussed in Section 2.4, the peculiarities of flash are not well suited to supporting write requests. In particular, a garbage collection for the entire management region must occur each time a comparably few buffer superblocks become filled. The occurrence of the F1R feature suggests that the garbage collection is performed as a

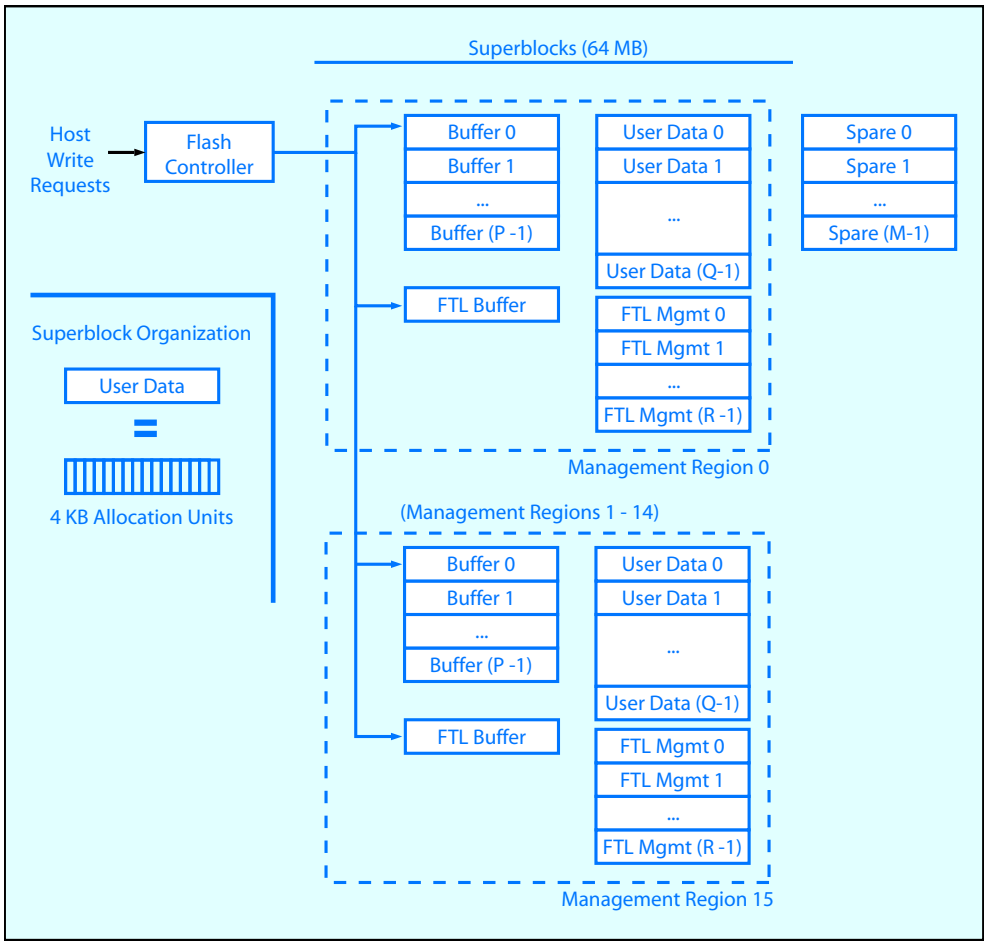

Figure 4.35: Block-Mode Random FTL Model)

single bulk operation. Delaying management operations, which occurs with a page-mode FTL, is not observed. As comparably few buffer superblocks exist, compared to the number of user data superblocks, a considerable amount of copying occurs.

The model in Figure 4.35 also shows the secondary effect of writing management data used for the FTL to FTL Mgmt superblocks. As measured, the amount of data is on the order of 2 pages (16KB) per 512KB of user data for transfer sizes of 4KB or larger. As with the data, this rate scales at 2x per 2x size reduction for transfer sizes smaller than 4KB. It is again is assumed that the filled FTL Mgmt

superblocks become part of a rotation of the FTL data blocks in which old FTL blocks are eventually released to become new spare superblocks (recall Figure 4.17).

The general equation for WAF for random transfers is shown in Figure 4.36. As presented above, the characterized drive is observed to have 64MB superblocks with 119 user superblocks and an average of 5.7 buffer superblocks. As shown, the amount of overall management data is the amount of written buffer data (5.7 x 64MB) scaled by the rate of management data (16KB per 512KB).

$$WAF_{Random} = \frac{User\ Data\ +\ Buffer\ Data\ +\ Management\ Data}{Buffer\ Data} \tag{4.7}$$

where,

$$Management\ Data\ =\ (5.7\ \times\ 64\,MB)\frac{16\,KB}{512\,KB}\ =\ 11.4\,MB \tag{4.8}$$

$$WAF_{Random} = \frac{(119\ \times\ 64\ )\ +\ (5.7\ \times\ 64\,MB)\ +\ (11.4\,MB)}{5.7\ \times\ 64\,MB} = 21.91 \tag{4.9}$$

Figure 4.36: WAF (Random)

The general equation of WAF for random transfers is shown in Figure 4.37. This equation is similar to Figure 4.36 except that the overall result is scaled by 2x per 2x reduction in transfer sizes below 4KB.

The random WAF measurements from Table 4.2 are plotted with the WAF estimations from the model shown in Figure 4.36 and Figure 4.37 in Figure 4.38. As this figure suggests, there is very good agreement between measurements and calculated values.

The ability to directly measure WAF, as well as gain insight into an FTL's internal operations that is sufficient to develop empirical models of FTL operation and

$$WAF_{Random,Sub-4KB} = (WAF_{Random}) \left(2^{\frac{2\,KB}{Transfer\ Size}} \right) \tag{4.10}$$

Example (512 B):

$$WAF_{Random,Sub-4KB} = (21.91) \left(2^{\frac{2\,KB}{512\,B}} \right) \;=\; 175.3 \tag{4.11}$$

Figure 4.37: WAF (Random, Sub-4KB)

produce equations to predict WAF, is considered to be an effective technique characterizing commercially available flash memory systems. This approach complements prior theoretical approaches presented in Section 3.4. The empirical instrumentation technique, as well as the model and WAF results of the block-mode FTL in developed in this thesis, have been presented [78, 79].

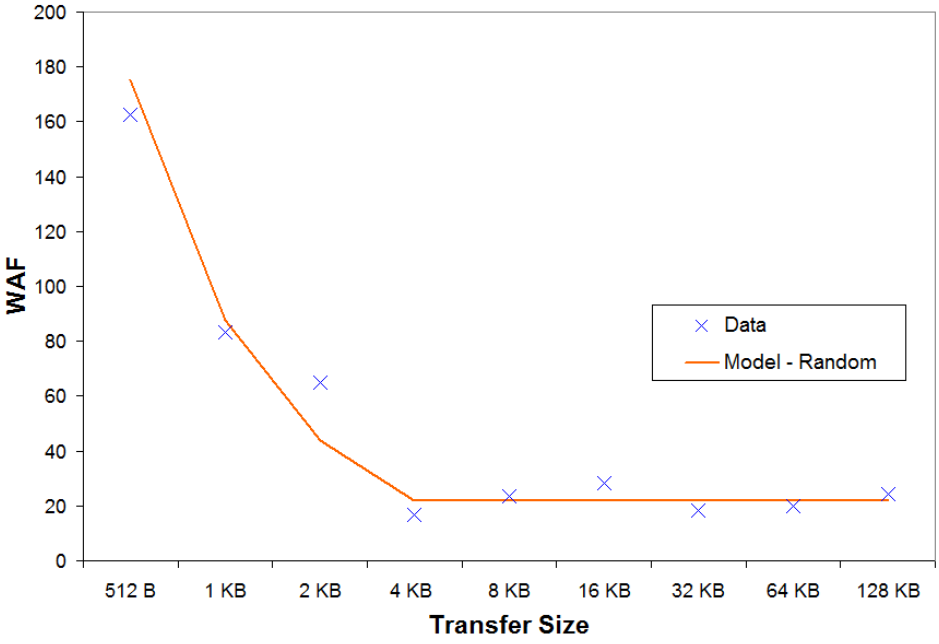

Figure 4.38: Block-Mode WAF (Random)

4.9 Measurements (Page-Mode FTL, Single-Point)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Characterization of a flash memory system using a page-mode FTL is similar to the technique employed with the flash memory systems using a block-mode FTL (recall Section 4.3). Specifically, characterization consists of an array of individual test points with testing at each point conducted for one hour. Extended tested, when required for improved resolution, is conducted for eight hours. However, when characterizing a page-mode FTL, the initial state of the data within the drive is significant. Of particular concern is the amount of *fragmentation* that has occurred among the specific flash pages within the flash erase blocks.

The amount of fragmentation present in the flash erase blocks is significant as it represents sizable inertia that needs to be overcome before meaningful measurements can be conducted. For example, if the flash memory system is subjected to an extended array of random write requests, the data distributed among the flash pages will eventually become 100% fragmented. Alternatively stated, the drive will have

become *dirty*. In this state, WAF will be at its maximum.

If, at that point in time, the random write requests are halted and an array of sequential write requests is initiated, the flash memory system will still behave as if random write requests were being issued as the flash memory system must conduct garbage collection that involves consolidation of the data in the flash pages from disparate flash erase blocks. This process (recall Figure 2.14) involves relatively high amounts of copy operations within the flash blocks and the WAF measurements will be observed to be rather high; even though sequential write requests are occurring. If the sequential write requests are continued, eventually the fragmentation of the flash memory units will be eliminated. Alternatively stated, the drive will have become *clean*. In this state, WAF will be at its minimum. As with block-mode FTLs, this minimum will approach unity (1.0).

These concepts are illustrated below in Figure 4.39. As this figure suggests, a clean drive subjected to random write requests will eventually become dirty and its WAF will approach the maximum. Conversely, a dirty drive subjected to sequential write requests will eventually become clean and its WAF will approach the minimum.

The consistent WAF performance of a clean flash memory system is subjected to random writes as a result of the spare blocks being used as "cache" blocks to directly support the incoming write requests without any specific management of data. Eventually, these blocks become filled and the flash memory system begins to become fragmented (dirty).

Given the behavior illustrated in Figure 4.39, it is clear that knowing the initial state of the flash memory system is critical for characterizing a flash memory system. As such, the flash memory system being characterized is cycled to ensure that it is fully clean for sequential characterization and fully dirty for random characterization.

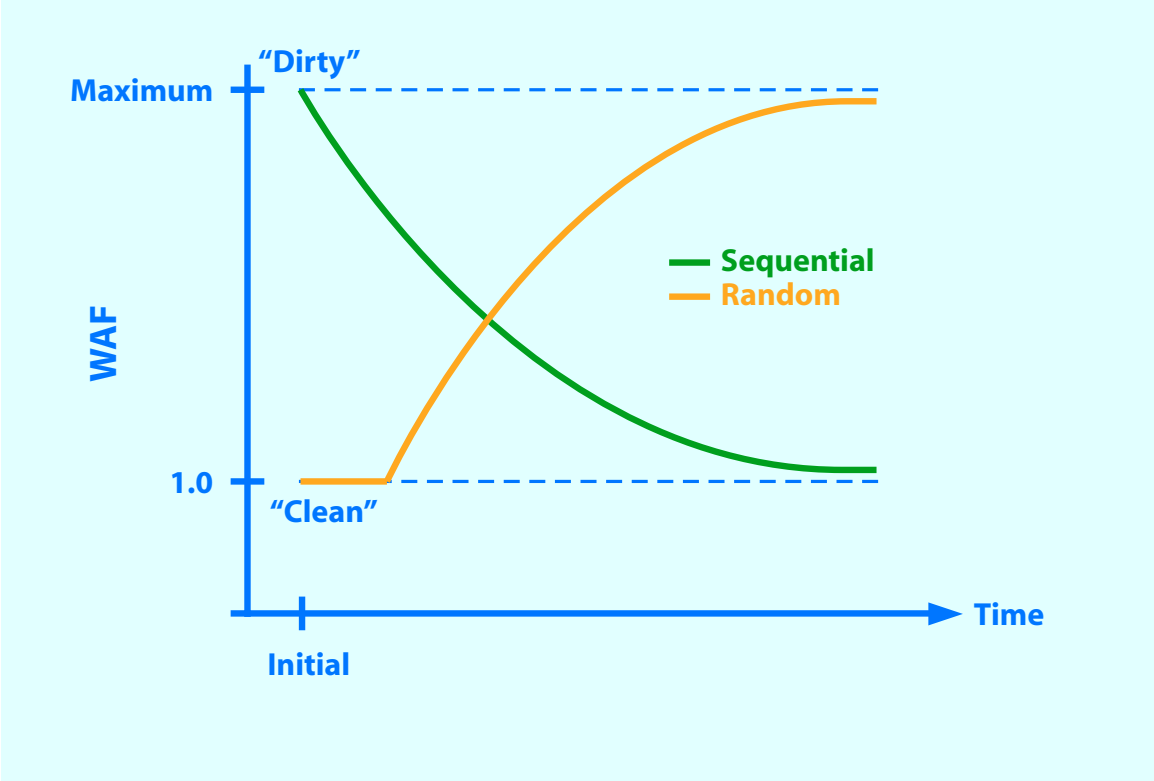

Figure 4.39: WAF Over Time

A commercially available 128GB drive using a Toshiba MLC (TH58TEG8D2HBA) flash memory using a page-mode FTL was selected for characterization. WAF measurements based on SMART data were made for the test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.9.

Comparing the overall WAF measurements listed in Table 4.9 with those for the block-mode FTL (recall Table 4.2) suggests that sequential WAF is around unity for for both FTL designs. However, the tables also suggest that a page-mode FTL can yield a lower WAF (12.70) for random write requests compared to a block-mode FTL (21.89). All else being equal, the lower WAF afforded by the page-mode FTL would lead to a longer flash memory system lifetime.

Transfer Size	WAF (Measured, Sequential)	WAF (Measured, Random)
512B	1.3355	87.02
1KB	1.3355	56.18
2KB	1.3355	34.34
4KB	1.0019	12.53
8KB	1.0019	12.60
16KB	1.0020	12.52
32KB	1.0020	12.60
64KB	1.0020	12.88
128KB	1.0020	12.65
Average (4KB)	1.0020	12.70

Table 4.9: WAF Measurements (Page-Mode FTL)

4.10 Measurements (Page-Mode FTL, Single-Point, Sequential)

Representative plots of performance measurements for sequential test points (512B to 128KB) are shown in Figure 4.40 to Figure 4.43, respectively. As these figures illustrate, there is a baseline of nominal performance for all test points. Unlike the plots produced from the analysis of block-mode FTLs, there are no periodic artifacts apparent.

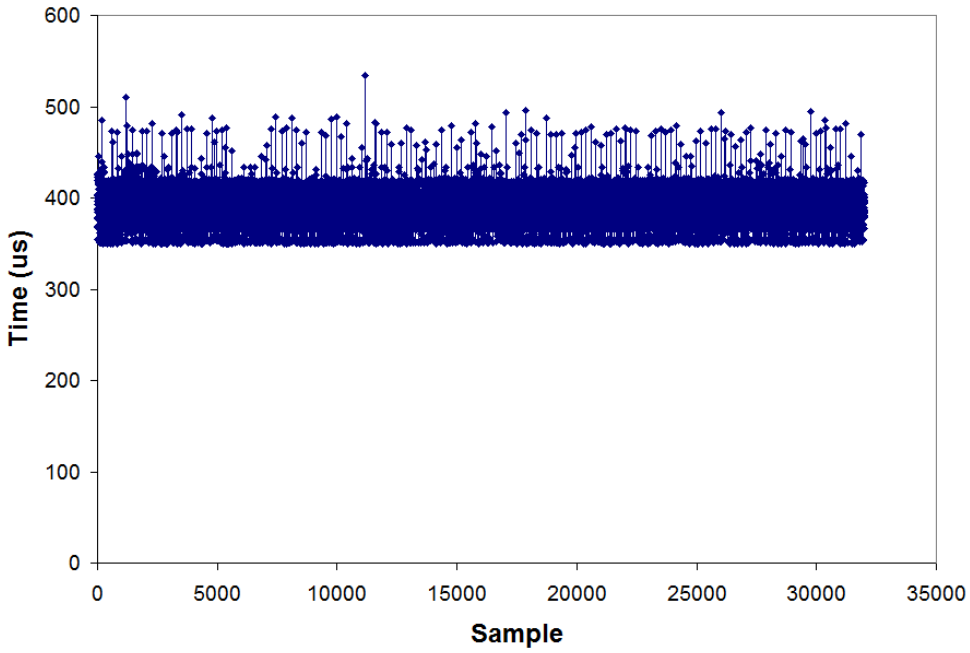

Figure 4.40: 512B Performance Data (Page-Mode FTL, Sequential)

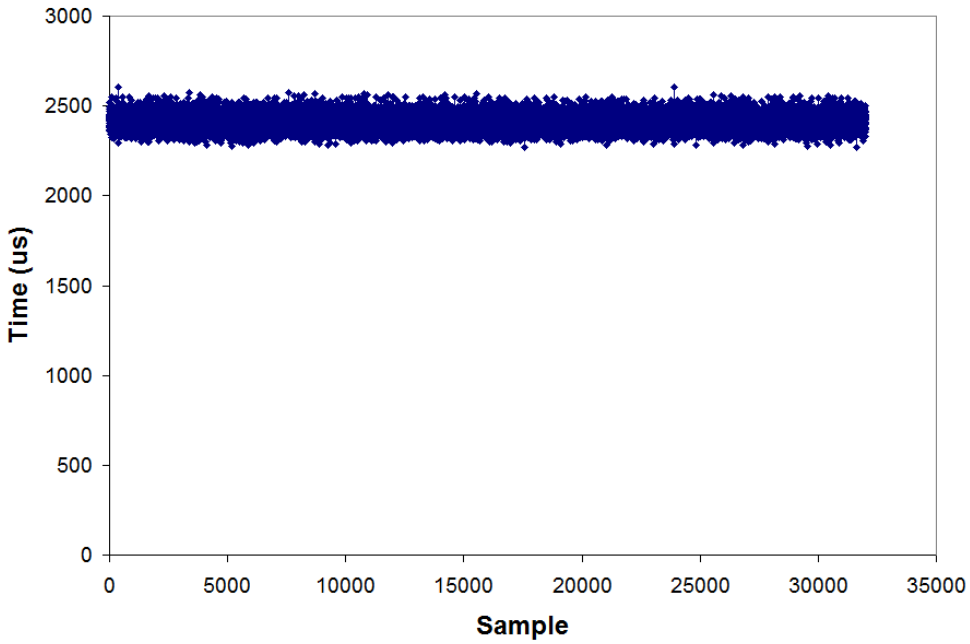

Figure 4.41: 4 KB Performance Data (Page-Mode FTL, Sequential)

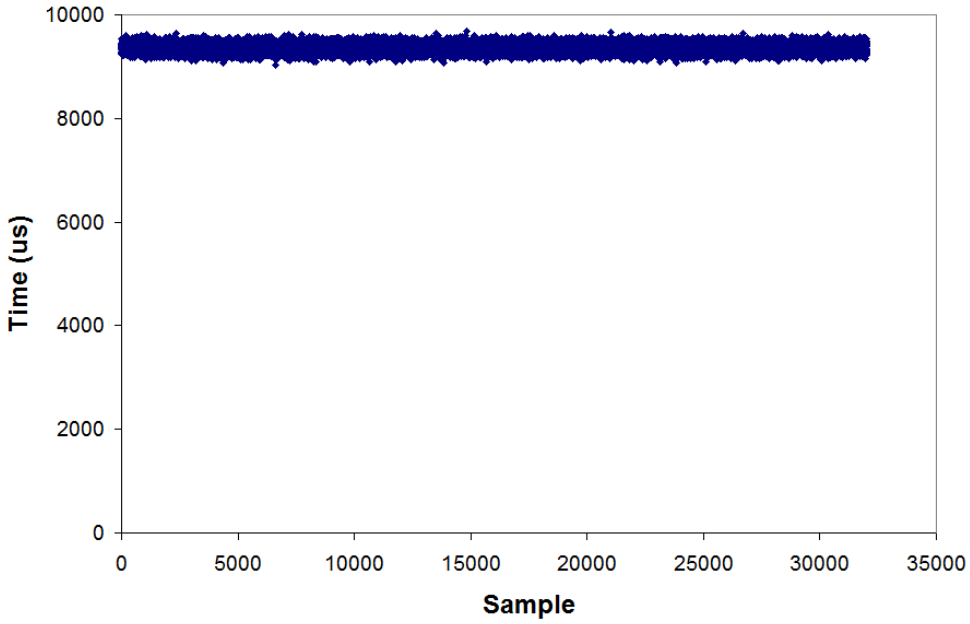

Figure 4.42: 16 KB Performance Data (Page-Mode FTL, Sequential)

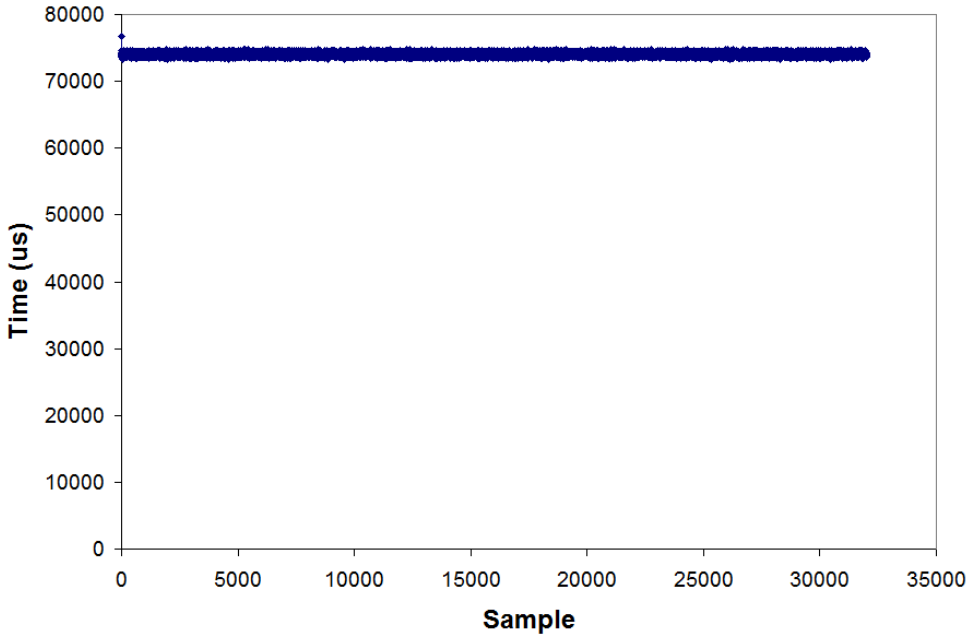

Figure 4.43: 128 KB Performance Data (Page-Mode FTL, Sequential)

Analysis of the collected data indicates that all points exhibited transfers times to within an average of 500 us. This sharply contrasts with the behavior observed for the flash memory system with the block-mode FTL (recall Figure 4.4 to Figure 4.12).

Since a flash block erase takes on the order of 5 ms, the lack of any periodic performance degradation, particularly for smaller transfer sizes where transfer times are much smaller than erase times (recall Figure 4.40), suggests that at least two channels of flash are being used in parallel. One channel has data being committed, while the other channel may be erased. As noted before with block-mode FTLs, parallel operations may potentially be obscured with regards to a signature of performance impact.

It is also noted that the controller used for the page-mode FTL was developed more recently. As such, we can conclude that its processor is more advanced. One performance enhancement is its ability to perform the necessary block erases in parallel with receipt of incoming data.

4.11 Measurements (Page-Mode FTL, Single-Point, Random)

For each random test point (recall Table 4.9), the performance of every transfer over the entire one hour test was also collected. Representative data collected for the random test points is shown below in Figures 4.45 to 4.52.

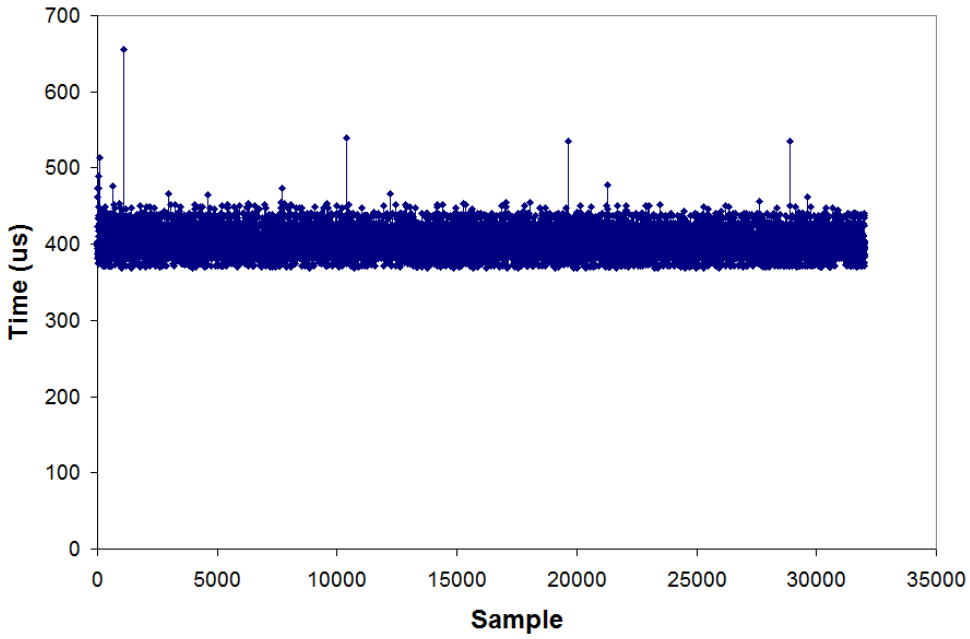

Figure 4.44: 512 B Performance Data (Page-Mode FTL, Random)

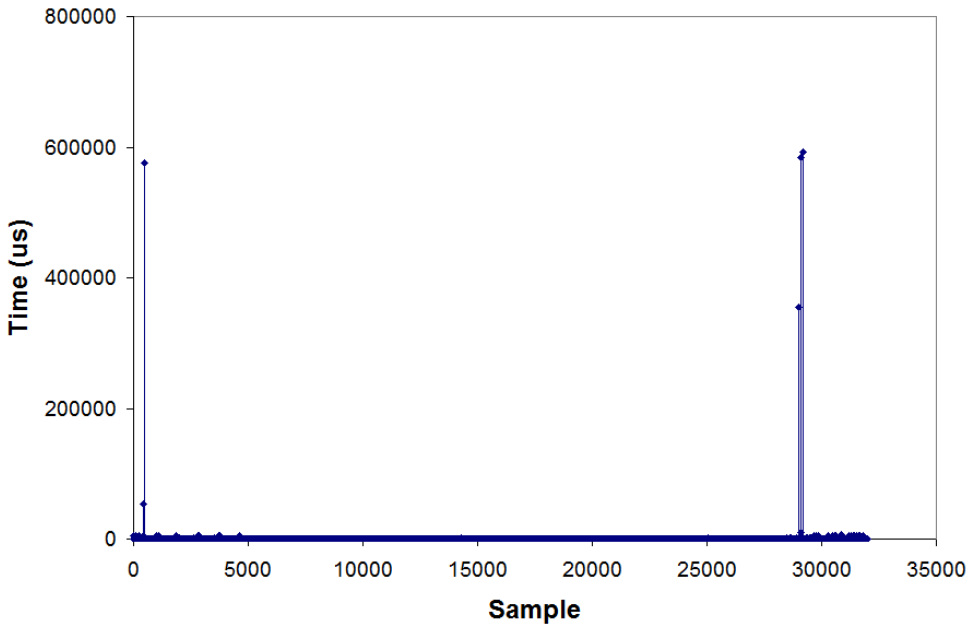

Figure 4.45: 1 KB Performance Data (Page-Mode FTL, Random)

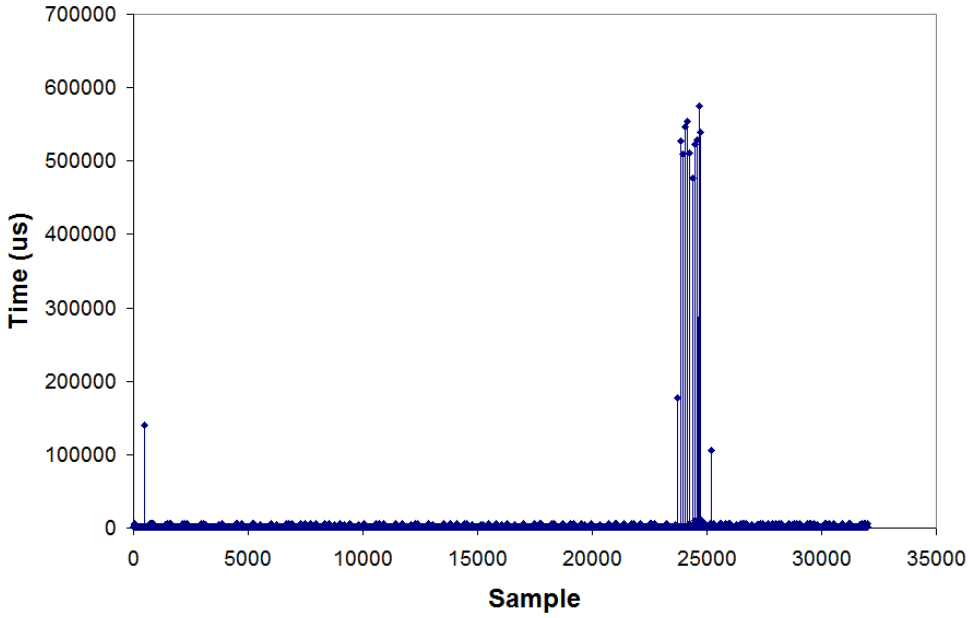

Figure 4.46: 2 KB Performance Data (Page-Mode FTL, Random)

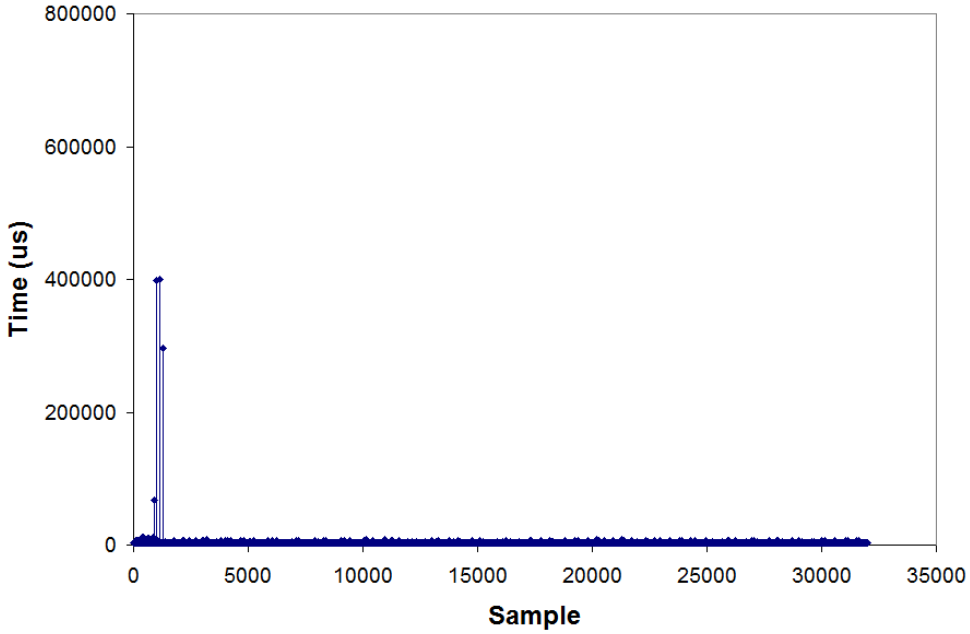

Figure 4.47: 4 KB Performance Data (Page-Mode FTL, Random)

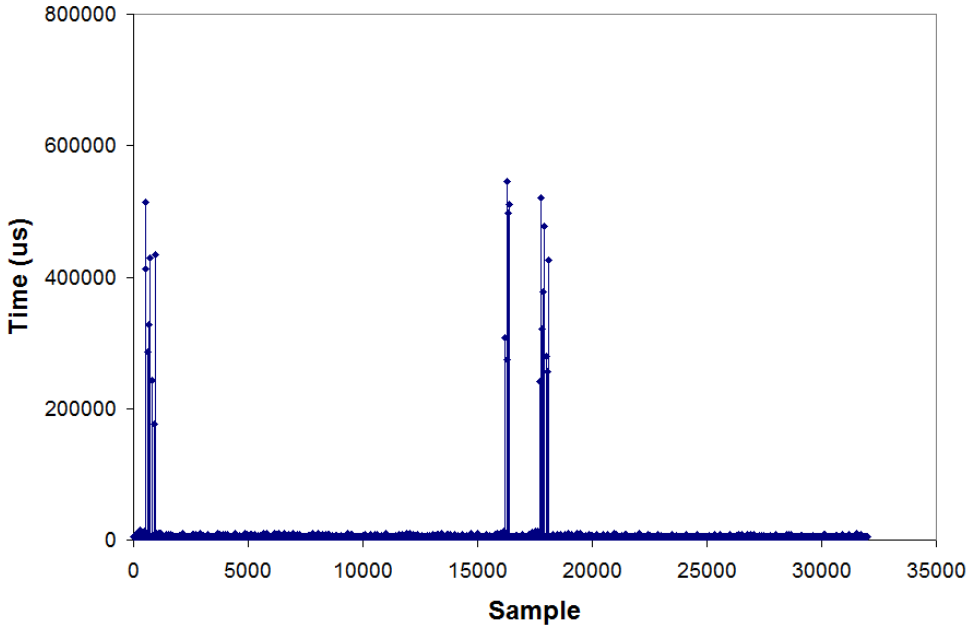

Figure 4.48: 8 KB Performance Data (Page-Mode FTL, Random)

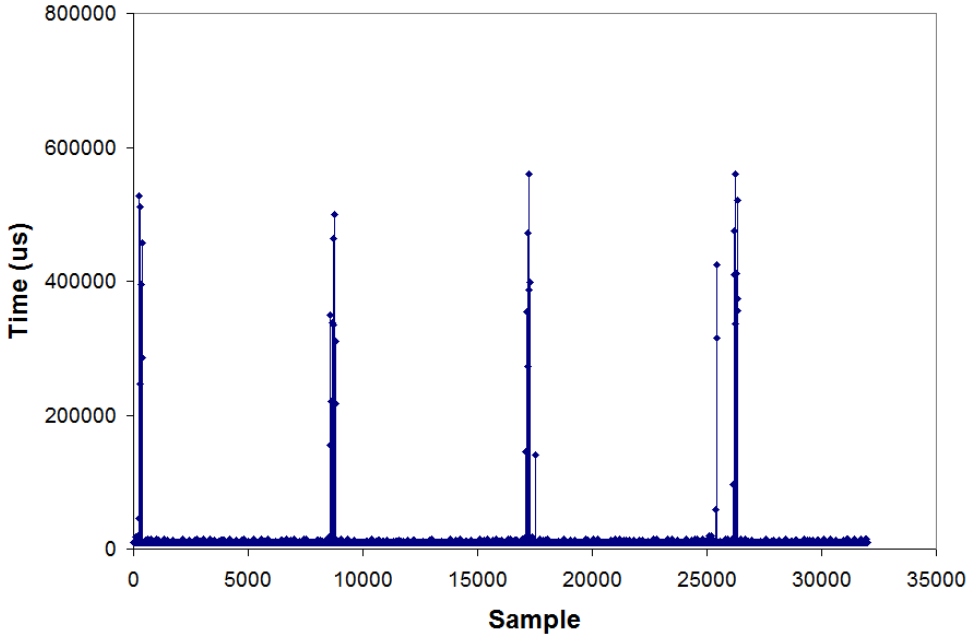

Figure 4.49: 16 KB Performance Data (Page-Mode FTL, Random)

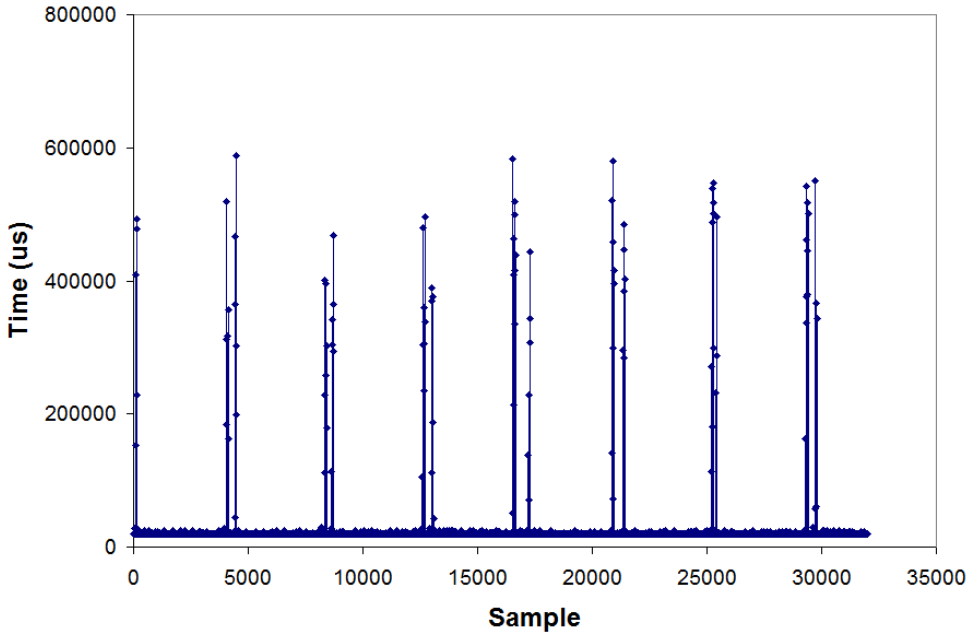

Figure 4.50: 32 KB Performance Data (Page-Mode FTL, Random)

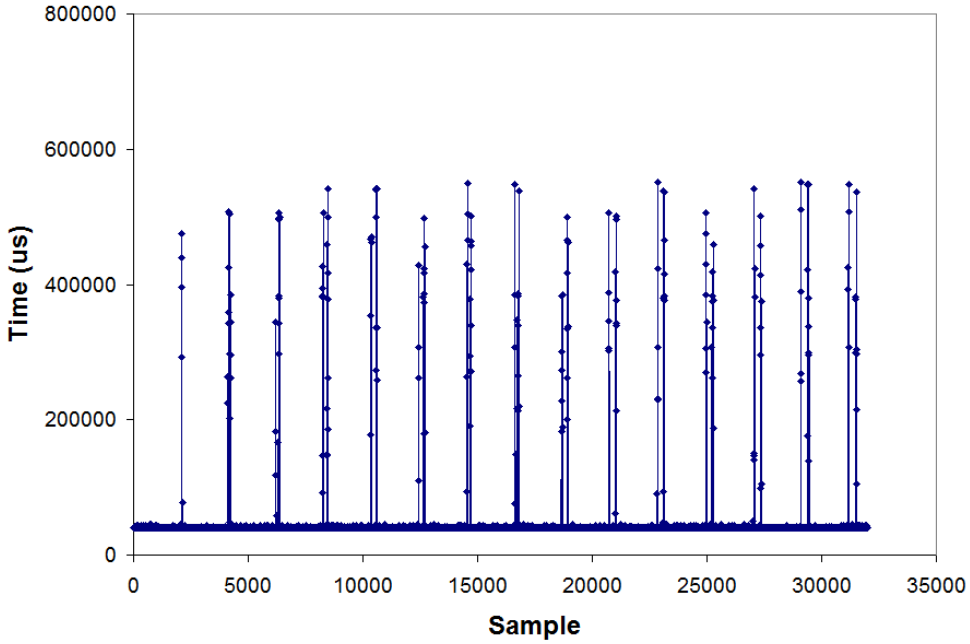

Figure 4.51: 64 KB Performance Data (Page-Mode FTL, Random)

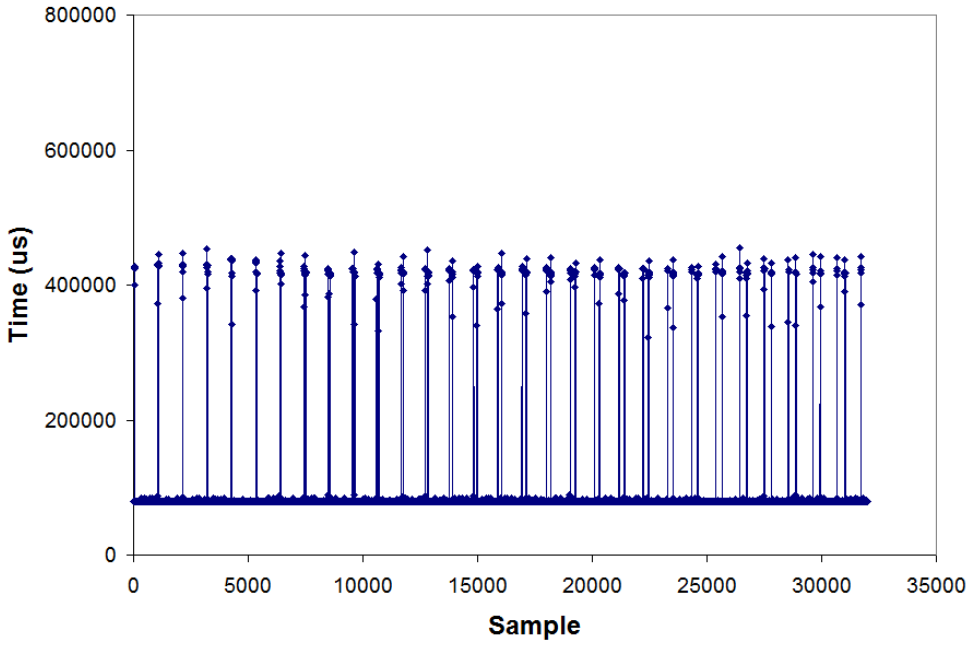

Figure 4.52: 128 KB Performance Data (Page-Mode FTL, Random)

4.11.1 F1R

As Figures 4.44-4.52 illustrate, there are transfers with considerable delays beyond the nominal. These delays are on the order of 400 ms. This value is well aligned with the previously calculated block copy time of 362.4 ms. These spectral features are identified as F1R.

Some scatter is present in the magnitude of the F1R features. Of particular interest are the values below the block copy time that are observed. This may be due to the lack of a need to copy a full flash erase block. Garbage collection invariably involves invalid pages that do not need to be copied.

The reduced block copy times can also be explained by the multiple channels

used to avoid the impact of block erases noted for sequential operations (recall Section 4.10). The presence of these channels is suggested by the paired clusters of F1R features shown in many of the above figures. These features are especially conspicuous in the plot of data for 32 KB transfers (Figure 4.50). The irregular distance between the clusters of F1R features confirms that the operations are independent, and that the flash memory system is using two parallel channels.

Representative plots of data filtered to show only F1R features are shown below in Figures 4.53-4.56. The range of transfers (x-axis) has been adjusted from the above range to highlight a small number of F1R events.

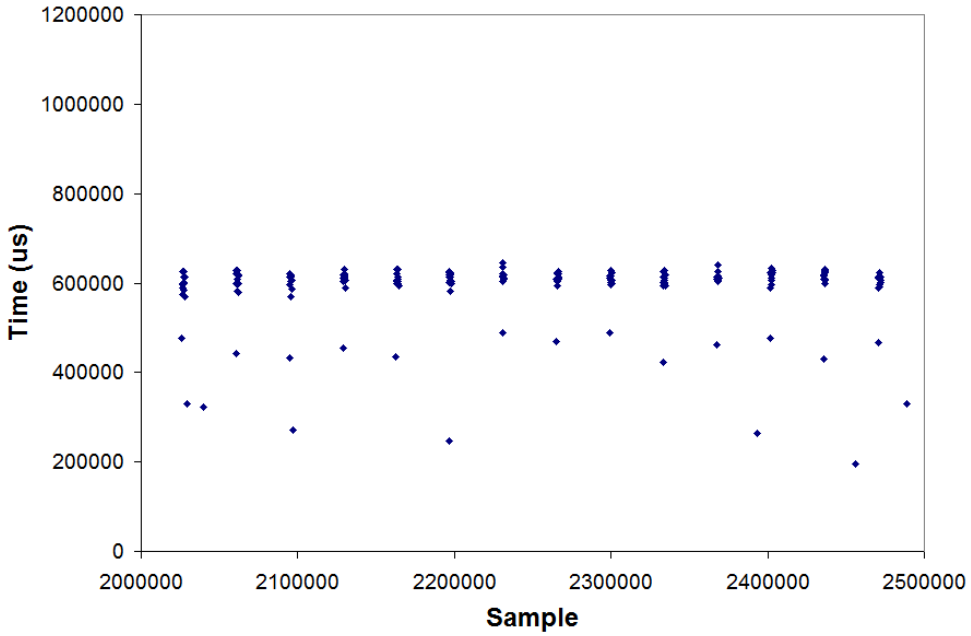

Figure 4.53: 512 B F1R Features (Page-Mode FTL, Random)

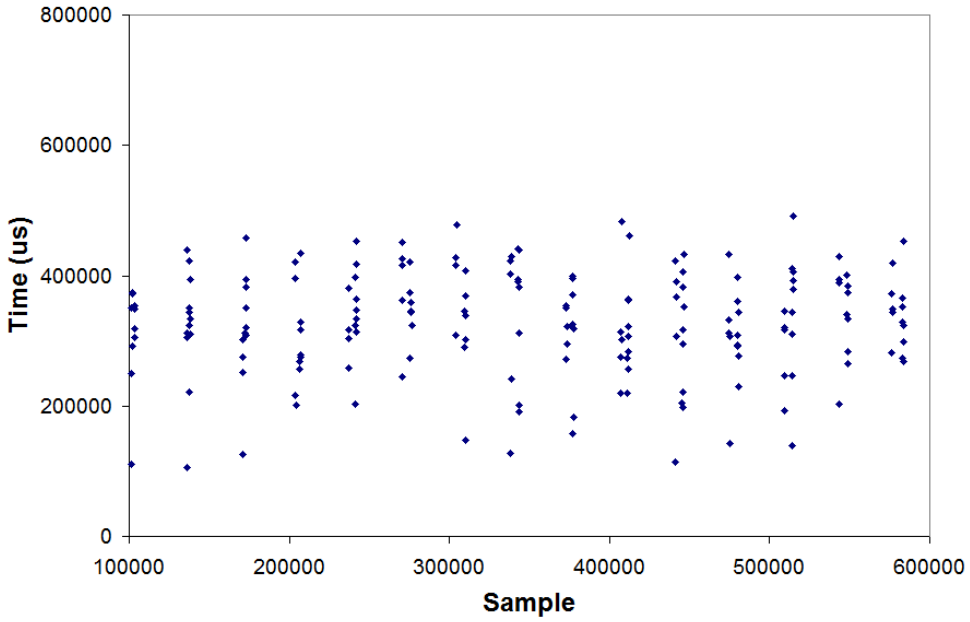

Figure 4.54: 4 KB F1R Features (Page-Mode FTL, Random)

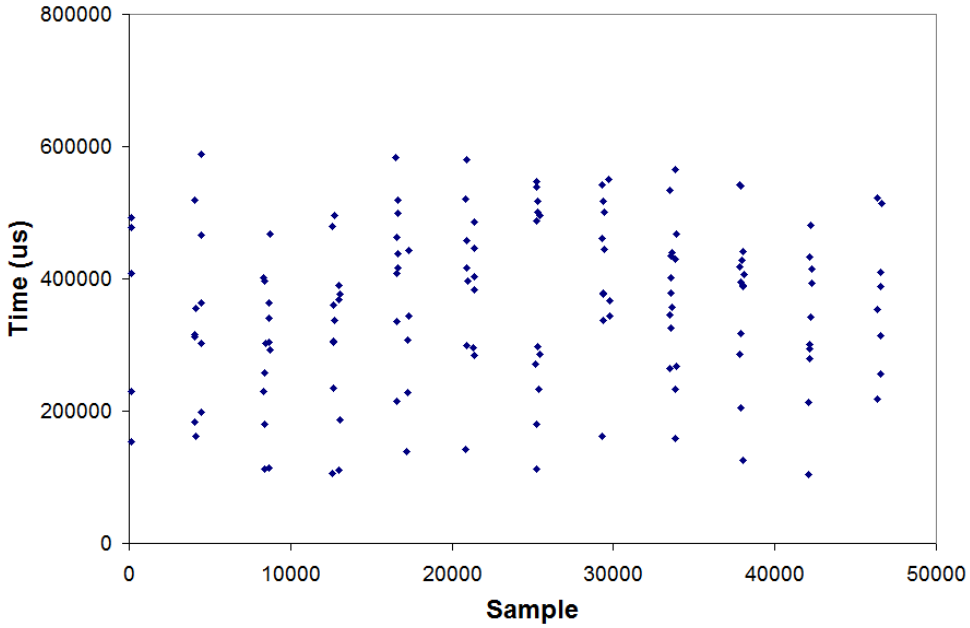

Figure 4.55: 32 KB F1R Features (Page-Mode FTL, Random)

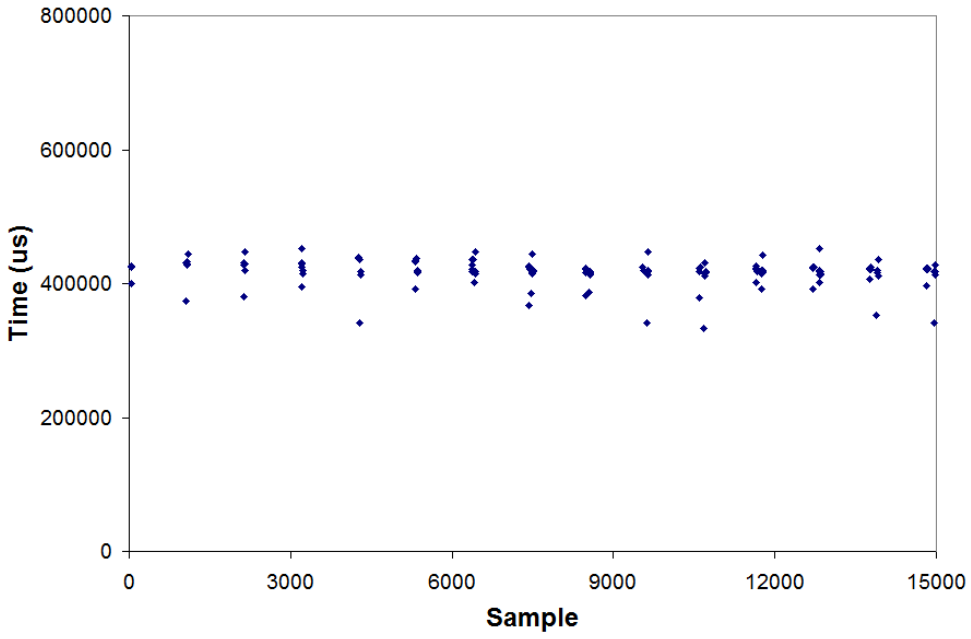

Figure 4.56: 128 KB F1R Features (Page-Mode FTL, Random)

As these figures show, F1R events occur periodically in collective bursts. For each random test point, the number of F1R events per region and the number of host transfers for each F1R region are determined. Averages of these values are calculated for each test point and are tabulated below in Table 4.10. The amount of user data copied for each average F1R Region is also included.

The average value of host data per F1R region indicates that 105.7 MB of data is transferred for each F1R region. Alternatively stated, 105.7 MB of data is transferred for each garbage collection event. As garbage collections involve whole erase blocks, and erase blocks are 2 MB, we can conclude that each F1R region involves 128 MB of erase blocks. As two channels are observed to be used, we further conclude that the 128 MB F1R region consists of two 64 MB superblocks.

Knowing the size of the F1R management region used, the total number of F1R

Transfer Size	F1R Events Per Region	Transfers Per F1R Region	Host Data Per F1R Region (MB)
512 B	13.85	30190	14.74
1 KB	18.10	23220	22.68
2 KB	17.95	16450	32.13
4 KB	11.60	27880	108.9
8 KB	12.15	12540	97.97
16 KB	12.65	6342	99.09
32 KB	11.25	3859	120.6
64 KB	12.65	1594	99.64
128 KB	12.35	861.8	107.7
Average (4 KB)	12.11		105.7

Table 4.10: F1R Features

events are considered in the context of the data transfer for all of the data for each test point and WAF can be estimated. Specifically, the amount of host data written is calculated for each test point based upon the number of transfers and the amount of flash data written is calculated for each test point based upon the size of the F1R management region. The ratio of these two values is then used to estimate WAF. The results are shown below in Table 4.11.

4.12 Measurements (Page-Mode FTL, Over-Provisioning)

As mentioned in Section 4.7, knowing the amount of over-provisioning for a drive is desirable when characterizing Write Amplification measurements and comparning them with our analytical models.

Transfer Size	Transfers	Host Data (GB)	F1R Events	Flash Data (GB)	WAF
512 B	5401131	2.575	1844	230.5	89.50
1 KB	3237389	3.087	1986	248.3	80.41
2 KB	2031898	3.876	1479	184.9	47.70
4 KB	1298458	4.953	471	58.88	11.89
8 KB	662665	5.056	504	63.00	12.46
16 KB	335079	5.113	518	64.75	12.66
32 KB	169017	5.158	484	60.50	11.73
64 KB	84577	5.162	539	67.38	13.05
128 KB	42485	5.186	496	62.00	11.95
Average (4 KB)		5.105		63.75	

Table 4.11: WAF Calculations Based Upon F1R Counts

Using the technique initially suggested in Section 4.7, the 128GB drive with page-mode FTL is completely cleaned using an ATA Secure Erase. As the drive is fully erased (clean), all of the data space (Total Data Space) is available as spare blocks. As such, random 4KB transfers can be readily accepted with no performance impacts beyond the average transfer times (see Figure 4.57).

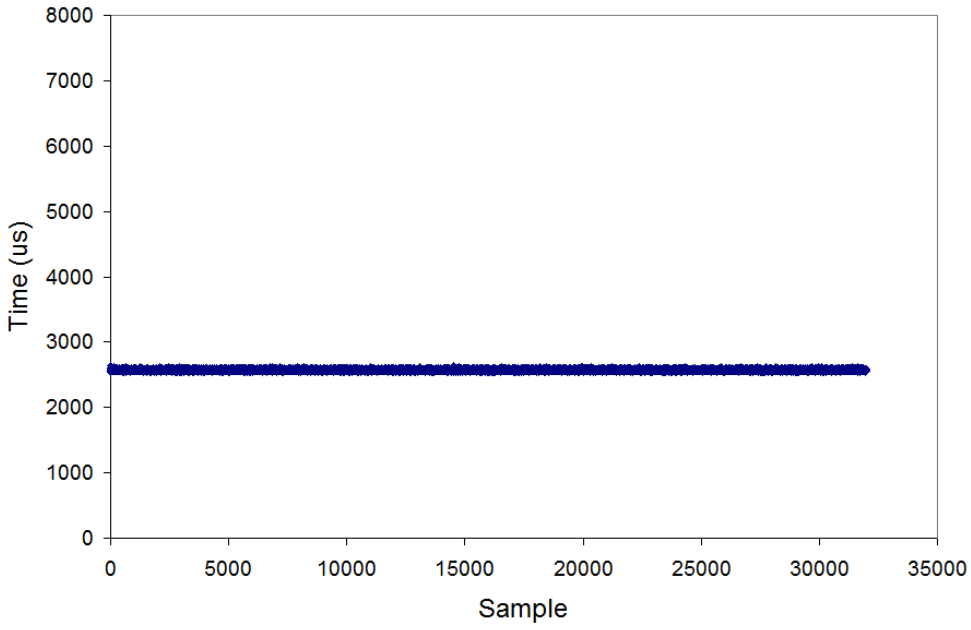

Figure 4.57: 4 KB Transfers (Page-Mode FTL, Random, Clean)

As all of the spare blocks are consumed, the drive will need to perform garbage collection to sustain additional transfers. At this point, performance impacts will become apparent (see Figure 4.58).

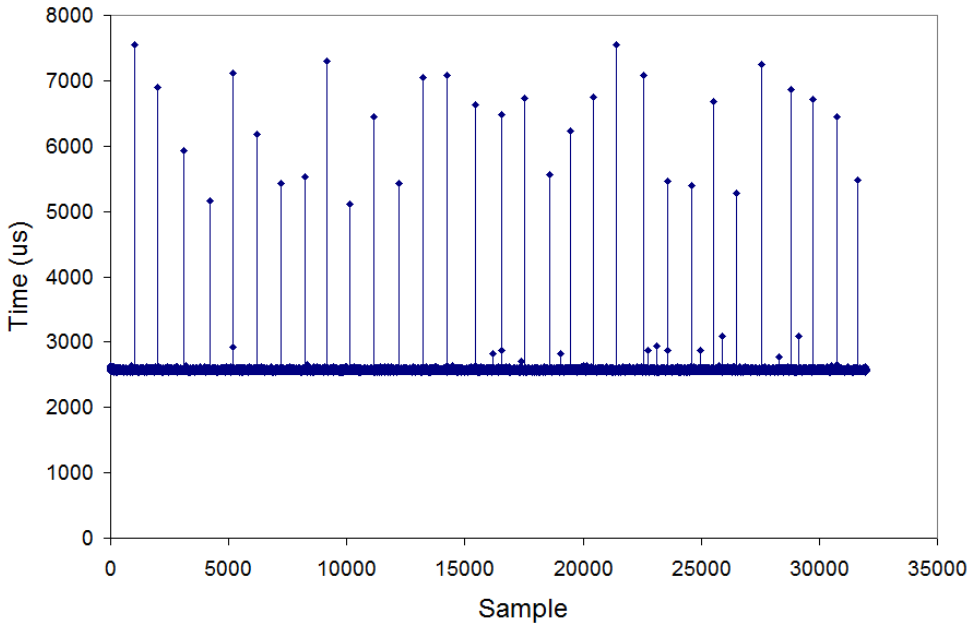

Figure 4.58: 4 KB Transfers (Page-Mode FTL, Random, Sustained)

The first performance impact represents the point at which all of the Total Data Space has been consumed (see Figure 4.59). A total of 33,130,626 4KB transfers have occurred up until this point. Considering that the User Data Space of the drive is 31,258,710 4 KB transfers (250,069,680 sectors), the over-provision of this drive is calculated to be 5.988% (see Figure 4.60). This value again compares well to the value of over-provisioning of 7% commonly reported utilized in commercially available drives [65]. This is especially so given that bad blocks are commonly identified during manufacturing and marked as unavailable so that true over-provisioning is typically lower than the ideal.

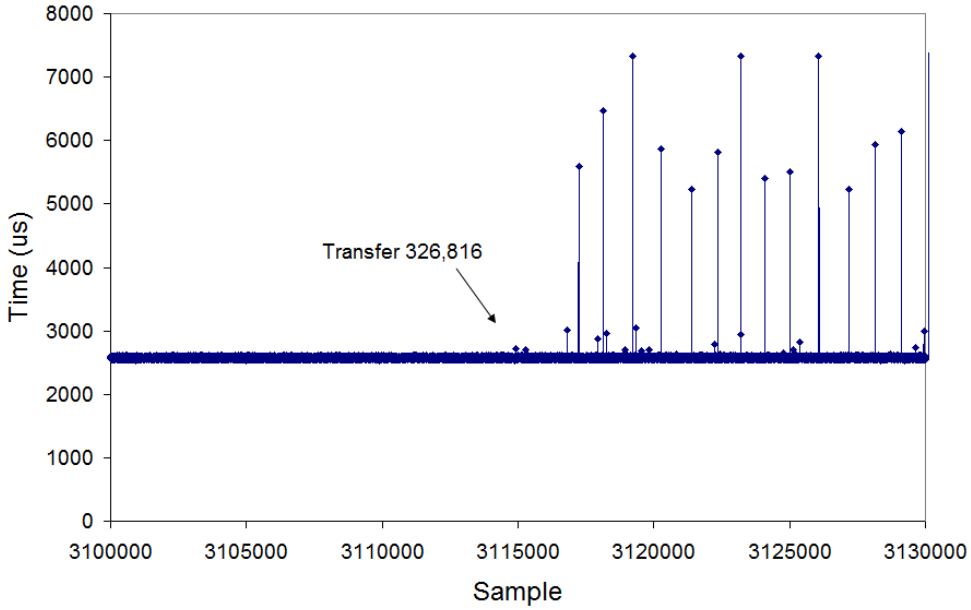

Figure 4.59: 4 KB Transfers (Page-Mode FTL, Random, Initial Saturation)

$$OP = \frac{33,130,626 \; - \; 31,258,710}{31,258,710} = 5.988\% \tag{4.12}$$

Figure 4.60: Over-Provisioning (Page-Mode FTL)

Knowing over-provisioning, the values of WAF (4KB, random) measured in this thesis for the flash memory system with the page-mode FTL (recall Table 4.9) are contrasted with the values predicted by the three analytic WAF equations in the literature. These results are listed in Table 4.12. As this table indicates, all analytic models under-predict WAF.

Source	WAF	Difference (Measured)
Measured	12.7	-
Agrawal2010 [3]	8.85	-30.3%
Luojie2012 [73]	9.02	-29.0%
Desnoyers2014 [28]	7.95	-37.4%

Table 4.12: WAF Values (Page-Mode FTL, 4 KB)

4.13 Empirical Model and WAF Equations (Page-Mode FTL, Single-Point, Sequential)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Based upon the measurements and observations in Section 4.10, and the confirmation of two flash channels presented in Section 4.11, an empirical model of the page-mode FTL that supports sequential writes in the characterized drive is developed. This model is presented in Figure 4.61.

As Figure 4.61 illustrates, the characterized 128GB drive with a page-mode FTL is organized into two channels composed of 64MB superblocks. Each superblock consists of 32 erase blocks. The flash controller transfers the data to the flash pages of a specific buffer superblock. When the buffer flash superblock becomes full, the invalidated flash superblock assigned to the particular sector address range is erased and reassigned as an available spare superblock. The next buffer flash superblock will be in the other channel so that the erase operation in the first channel can occur

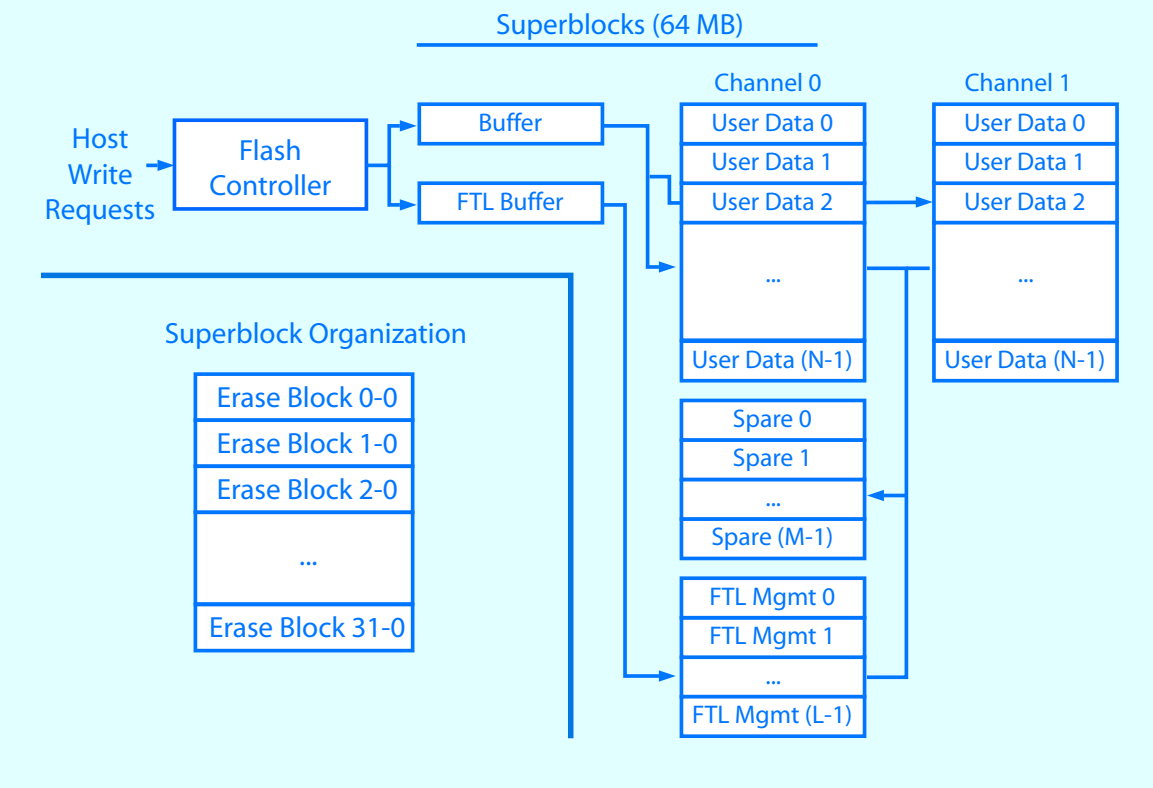

Figure 4.61: Page-Mode Sequential FTL Model

without host detectable delay.

As with the block-mode FTL, this mode of operation is well-suited to supporting the previously mentioned peculiarities of flash. In particular, flash is committed in whole pages and, ultimately, into whole erase block groups. Because the write requests are sequential, data may be logically aligned in each superblock; although the page-mode FTL architecture does not require this. In this manner, only one superblock is invalidated for each newly written buffer superblock. No data transfer (copy) operations are required within the flash to support these operations. Only a switch garbage collection needs to take place when the superblock becomes full (recall Figure 2.11). This mode of operation can be supported with as little as one spare

superblock. As no overhead is required for data support, the WAF for data is exactly 1.0 for the sequential mode of operation.

Figure 4.61 also shows the secondary effects of writing management data used for the FTL-to-FTL Mgmt superblocks. Inclusion of these effects is assumed since our measurement technique is unable to detect any performance impacts that could be associated with this management overhead.

As measured, the amount of data is observed to be 128KB per superblock (65536 KB). In a fashion similar to data superblocks, it is assumed that the filled FTL Mgmt superblocks become part of a rotation of FTL data blocks in which old FTL blocks are eventually released to become new spare superblocks.

The WAF of the characterized FTL when supporting sequential operations is modeled as shown in Figure 4.62. As this equation shows, WAF is calculated to be 1.002 and 1.336 for transfers smaller than 4KB. As this equation has no dependence on transfer size, aside from the slight management overhead, these values are expected to be valid for all sequential transfers.

$$WAF_{Sequential} = \frac{Superblock\ Size\ +\ Management\ Data}{Superblock\ Size} \tag{4.13}$$

$$WAF_{Sequential} = \frac{65536\ KB\ +\ 128\ KB}{65536\ KB} = 1.002 \tag{4.14}$$

$$WAF_{Sequential,Sub-4KB} = \frac{65536\ KB\ +\ 21.5\ KB}{65536\ KB} = 1.336 \tag{4.15}$$

Figure 4.62: WAF Model (Sequential)

The WAF measurements from Table 4.9 are plotted with the WAF estimations from the model shown in Figure 4.62 in Figure 4.63. We observe good agreement between measurements and modeled values.

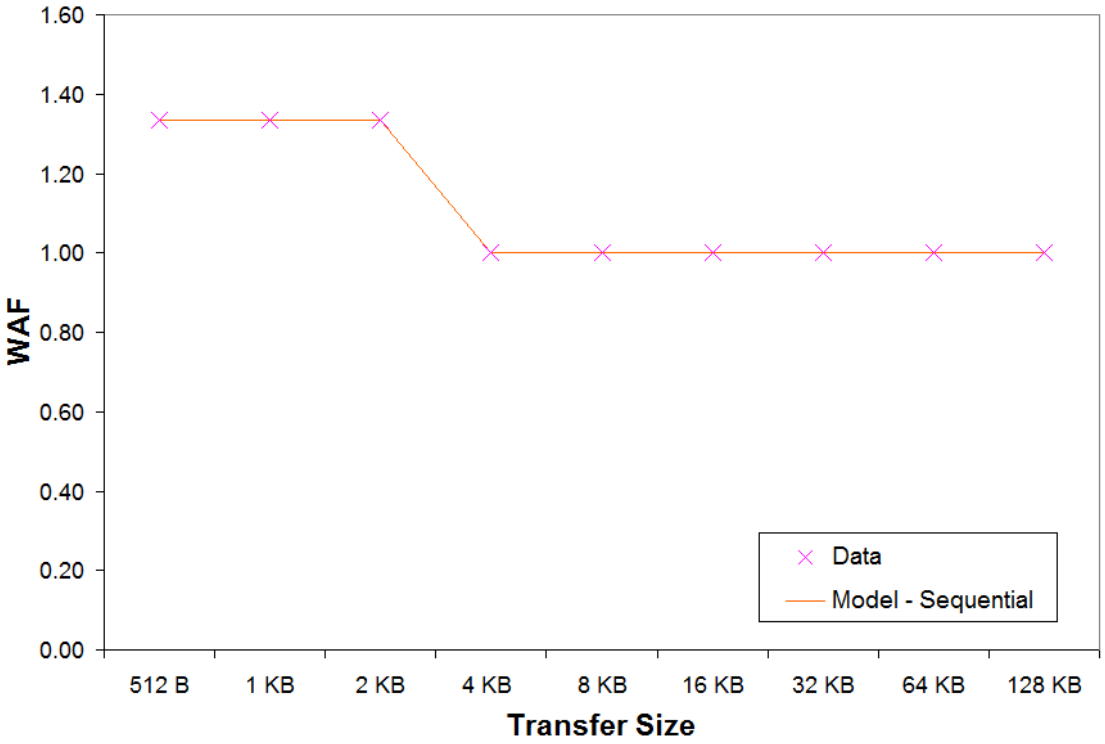

Figure 4.63: Page-Mode WAF (Sequential)

4.14 Empirical Model and WAF Equations (Page-Mode FTL, Single-Point, Random)

Based upon the measurements and observations in Section 4.11, an empirical model of the block-mode FTL that supports random write requests in the characterized drive may be developed. This model is presented in Figure 4.64.

As Figure 4.64 illustrates, the 128GB drive is organized into two channels. These are similar to the management regions shown for the block-mode FTL in Figure 4.35. All of the available user data superblocks are divided among the management regions. Each channel includes a single buffer block for incoming write requests. This contrasts

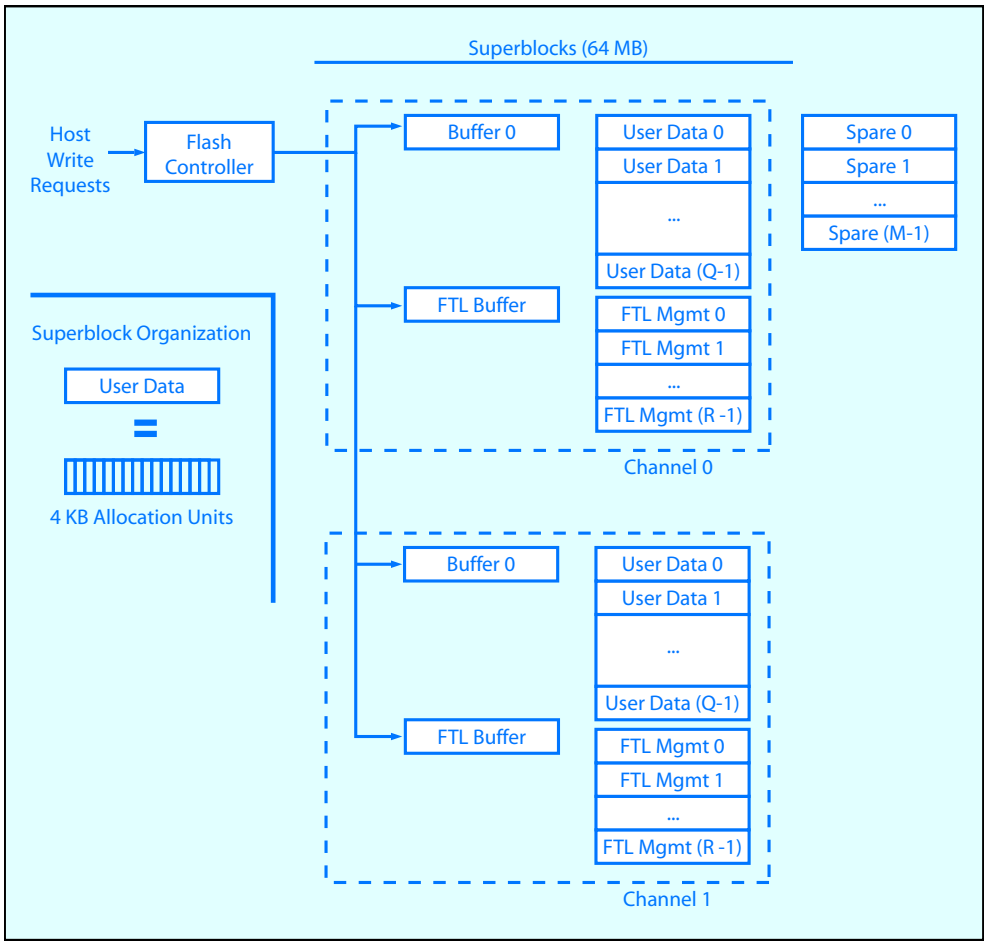

Figure 4.64: Page-Mode Random FTL Model

with the block-mode FTL which required an array of buffer blocks. For write requests smaller than 4KB, data is copied within the flash to ensure that the 4KB units remain whole and aligned.

As discussed in Section 2.4, the peculiarities of flash are not well-suited to support write requests. In particular, a garbage collection for the entire management region must occur each time the buffer superblock becomes filled when the flash pages are fully fragmented (dirty). Occurrences of the F1R features suggest that the garbage collection is performed as a single bulk operation.

The model in Figure 4.64 also shows the secondary effects of writing management data used for the FTL-to-FTL Mgmt superblocks. As with the page-mode sequential write measurements, there are no observable performance impacts. The overhead is estimated based upon the differences between the measured and calculated WAF values. As measured, the amount of data appears to be on the order of 4KB per 128KB of user data for transfer sizes of 4KB or larger. It is again is assumed that the filled FTL Mgmt superblocks becomes part of a rotation of FTL data blocks in which old FTL blocks are eventually released to become new spare superblocks (recall Figure 4.17).

As presented above, the characterized drive is observed to have two channels each of which has an array of 64MB superblocks with one buffer superblock. As shown in Table 4.11, the amount of overall flash data is 62.75GB and the overall user data is 5.105GB. Scaled to one superblock, this amounts to 5.206MB of user-data per 64MB superblock. With these measurements, the general equation of WAF for random transfers is shown in Figure 4.65.

$$WAF_{Random} = \frac{Superblock \ + \ Management\ Data}{User\ Data} \tag{4.16}$$

where,

$$Management\ Data \ = \ (64\,MB)\frac{4\,KB}{128\,KB} \ = \ 2.0\,MB \tag{4.17}$$

$$WAF_{Random} = \frac{(64\,MB) \ + \ (2.0\,MB)}{5.206\,MB} = 12.66 \tag{4.18}$$

Figure 4.65: WAF (Random)

The general equation of WAF for random transfers is shown in Figure 4.66. This equation is similar to Figure 4.65 except that the overall result is scaled by 2x per 2x

reduction in transfer sizes below 4KB.

$$WAF_{Random,Sub-4KB} = (WAF_{Random}) \left(2^{\frac{2\,KB}{Transfer\ Size}} \right) \tag{4.19}$$

Example (512 B):

$$WAF_{Random,Sub-4KB} = (12.66) \left(2^{\frac{2\,KB}{512\,B}} \right) \ = \ 101.3 \tag{4.20}$$

Figure 4.66: WAF (Random, Sub-4KB)

The random WAF measurements from Table 4.9 are plotted with the WAF estimations from the model shown in Figure 4.65 and Figure 4.66 in Figure 4.67. As this figure indicates, we see fairly good agreement between measurements and calculated values.

The WAF of 12.66 for the page-mode FTL is lower than the WAF of 21.91 observed for the block-mode FTL (recall Figure 4.36). This confirms that that page-mode FTLs can be more effective for handling random write operations than block-mode FTLs.

It is interesting to note that that the WAF for the page-mode FTL is higher than values predicted by the three equations based analytic models for page-mode FTLs (recall Table 4.12). Additionally, the value of WAF predicted by the analytic equations is especially conservative as it does not account for the internal copy operations used to support transfers smaller than 4KB. For comparison, data and models from Figure 4.37 and Figure 4.66 are combined with the values predicted by the three analytic models and shown below in Figure 4.68. As this figure highlights, the models developed in this work are more accurate predictors of WAF over all transfer sizes; especially for transfer sizes below 4KB.

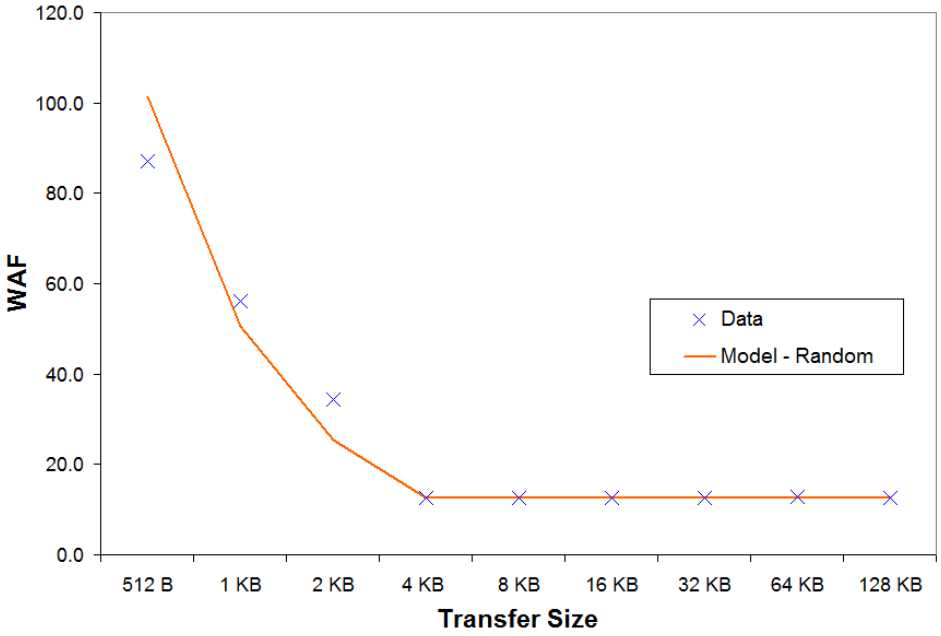

Figure 4.67: Page-Mode WAF (Random)

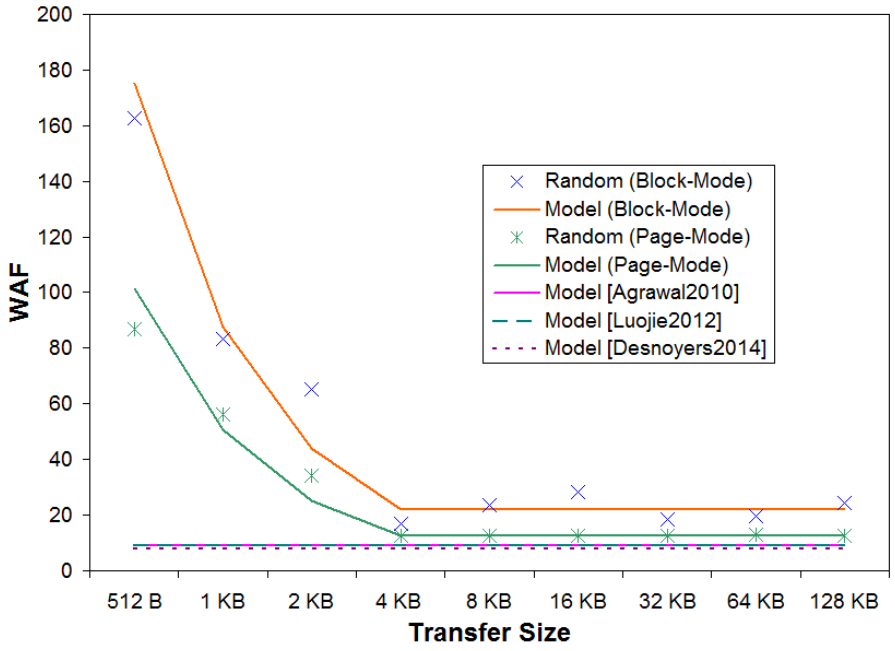

Figure 4.68: WAF Models (Random)

4.15 Measurements (Block-Mode FTL, Repeated)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Recognizing that repeated transfers are a more accurate representation of file-transfer activities (recall Section 2.6), the flash memory system with the block-mode FTL is characterized using the previously developed technique for repeated writes. Specifically, the drive is subjected to an array of tests consisting of continuous repeated write operations using a fixed transfer size for each test. Repeated transfers of a single fixed location (R1), two fixed locations (R2), and three fixed locations (R3) were studied.

As in the previous sections, characterization consists of an array of individual test points with testing at each point conducted for one hour. Extended testing, when required for improved resolution, is conduced for eight hours. Overall, tests using 512B, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, and 128KB transfer sizes have been conducted.

4.15.1 Repeated 1 (R1)

WAF measurements based on SMART data were made for the single fixed location (R1) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.13.

Transfer Size	WAF (Measured)
512 B	505.1
1 KB	252.5
2 KB	126.6
4 KB	63.32
8 KB	32.48
16 KB	16.91
32 KB	8.876
64 KB	5.006
128 KB	3.253

Table 4.13: WAF Measurements (Block-Mode FTL, R1)

Representative plots of performance measurements for the repeated (R1) test points (512B to 128KB) are shown in Figures 4.69-4.72, respectively.

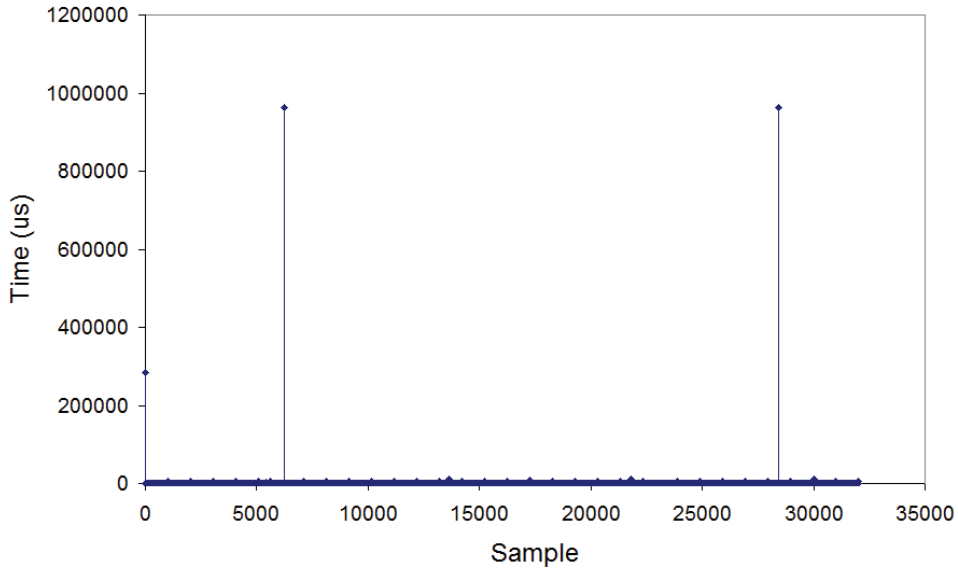

Figure 4.69: 512 B Performance Data (Block-Mode FTL, R1)

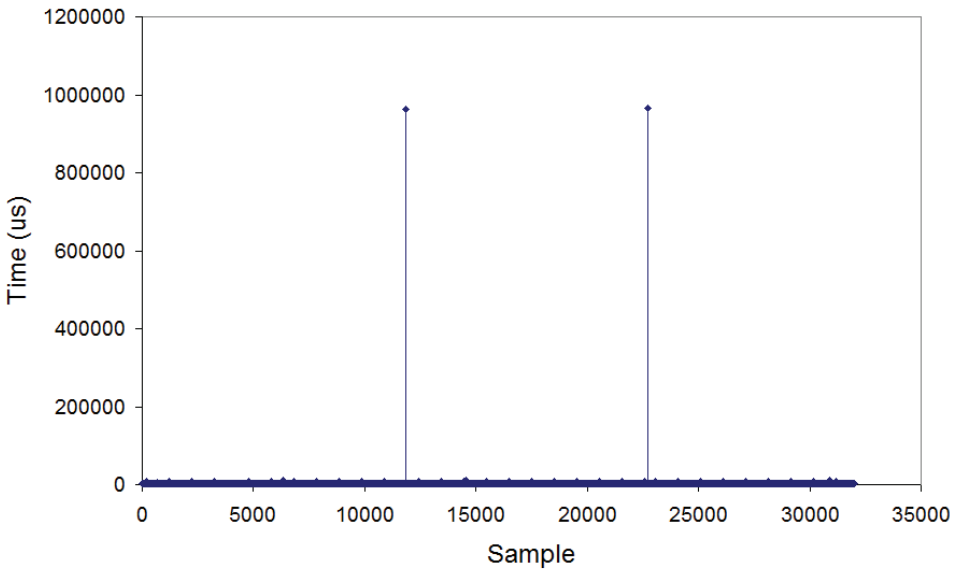

Figure 4.70: 4 KB Performance Data (Block-Mode FTL, R1)

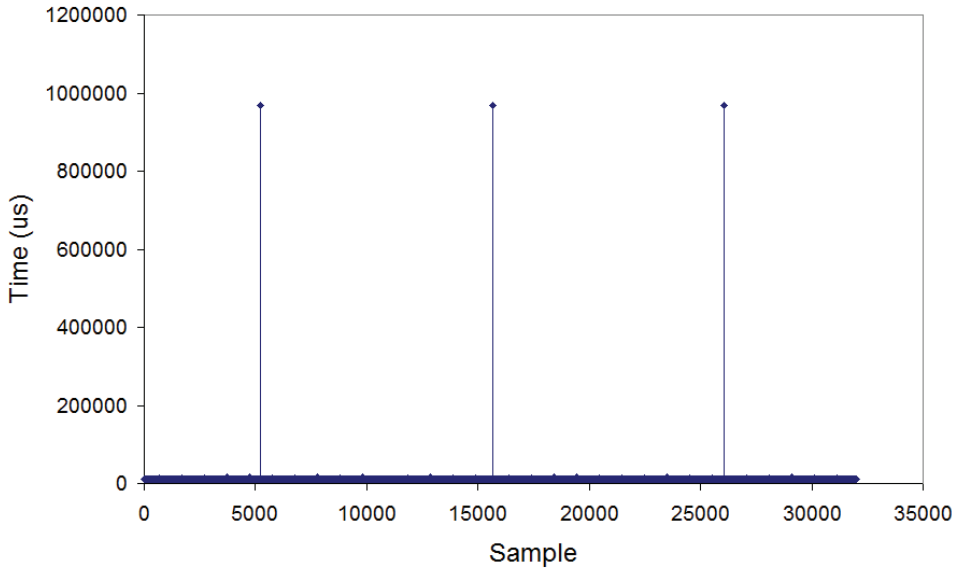

Figure 4.71: 16 KB Performance Data (Block-Mode FTL, R1)

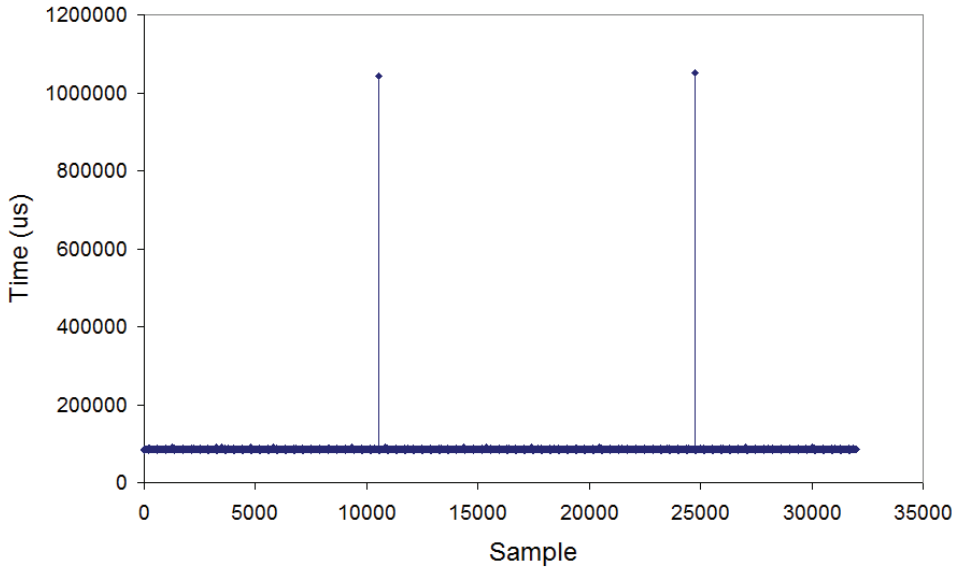

Figure 4.72: 128 KB Performance Data (Block-Mode FTL, R1)

4.15.2 F1 and F2 (R1)

As Figure 4.69 to Figure 4.72 illustrate, there are transfers with considerable delays beyond nominal. These delays are on the order of 1 s. These spectral features are identified as F1. Representative plots of data filtered to show only F1 features are shown below in Figure 4.73 to Figure 4.76. The transfer range (x-axis) has been adjusted to shown a small number of F1 events.

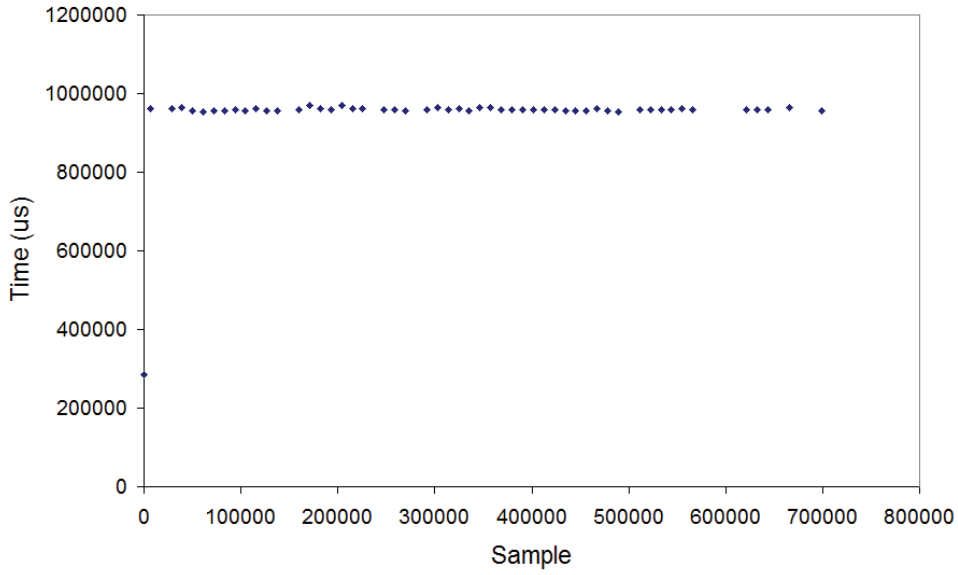

Figure 4.73: 512 B F1 Features (Block-Mode FTL, R1)

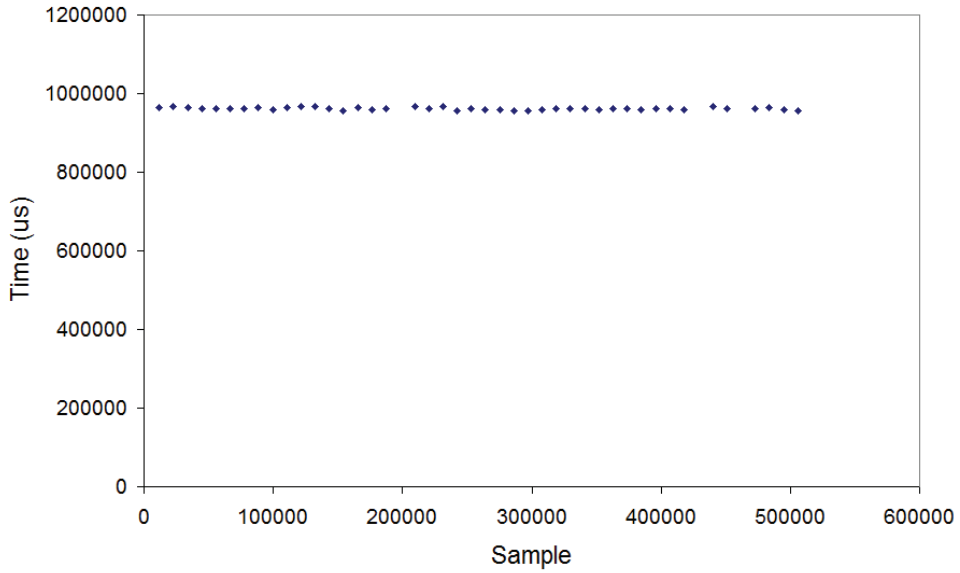

Figure 4.74: 4 KB F1 Features (Block-Mode FTL, R1)

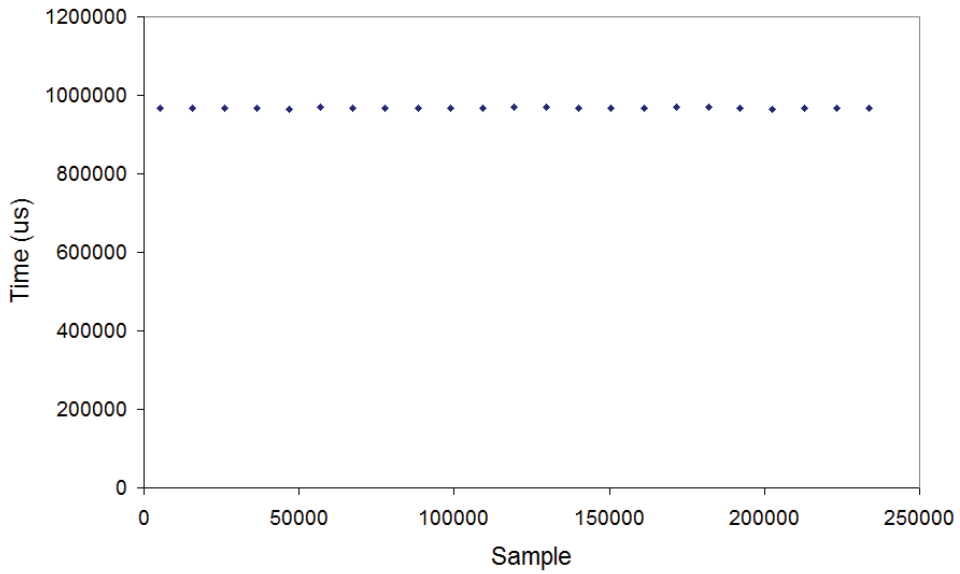

Figure 4.75: 16 KB F1 Features (Block-Mode FTL, R1)

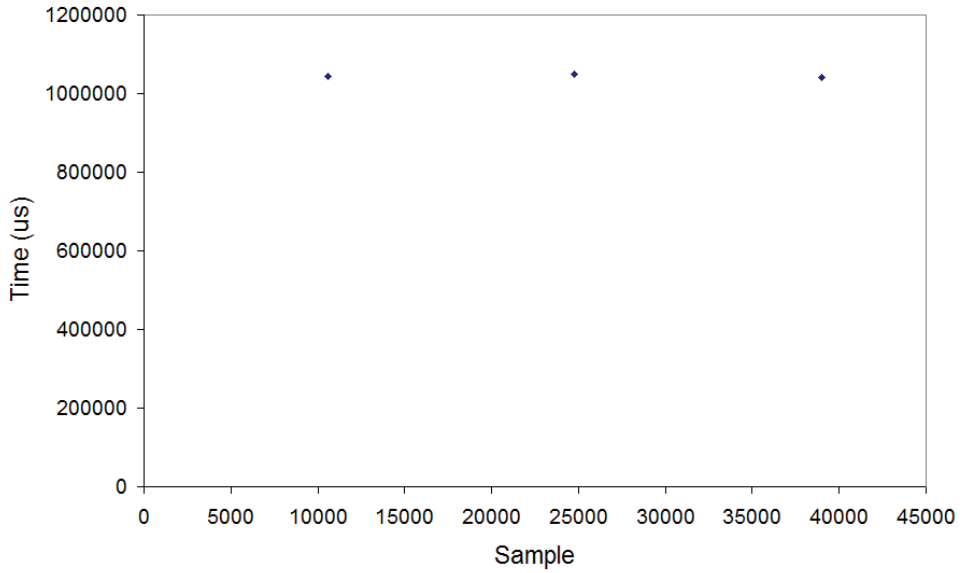

Figure 4.76: 128 KB F1 Features (Block-Mode FTL, R1)

Although some random data values are present, the plots show a highly consistent and periodic behavior (denoted F1), with a performance penalty of 1,000,000 us (1 s) above the nominal. This feature largely appears as a horizontal line across the plots.

We recall from our earlier discussion, a block copy operation takes 362.4 ms. Three block copy operations take 1,092,600 us. This value is highly similar to the 1,000,000 us (1 s) observed for the F1 peaks. As such, F1 is believed to be three block copy operations. As block copies may involve multiple areas in multiple chips, it is not expected that operations can be supported in parallel. The number of F1 features observed is provided in Table 4.14.

Continued analysis of the fine-grained performance data suggests a second periodic feature is also present. These delays are on the order of 1 s. These spectral features are identified as F2. Representative plots of data filtered to show only F2 features are shown below in Figures 4.77-4.80. The transfer range (x-axis) has been adjusted

Transfer Size	F1 Frequency
512 B	10,950
1 KB	10,980
2 KB	10,980
4 KB	10,950
8 KB	10,690
16 KB	10,390
32 KB	9,905
64 KB	8,989
128 KB	7,104

Table 4.14: F1 Frequencies (Block-Mode FTL, R1)

to show a small number of F2 events.

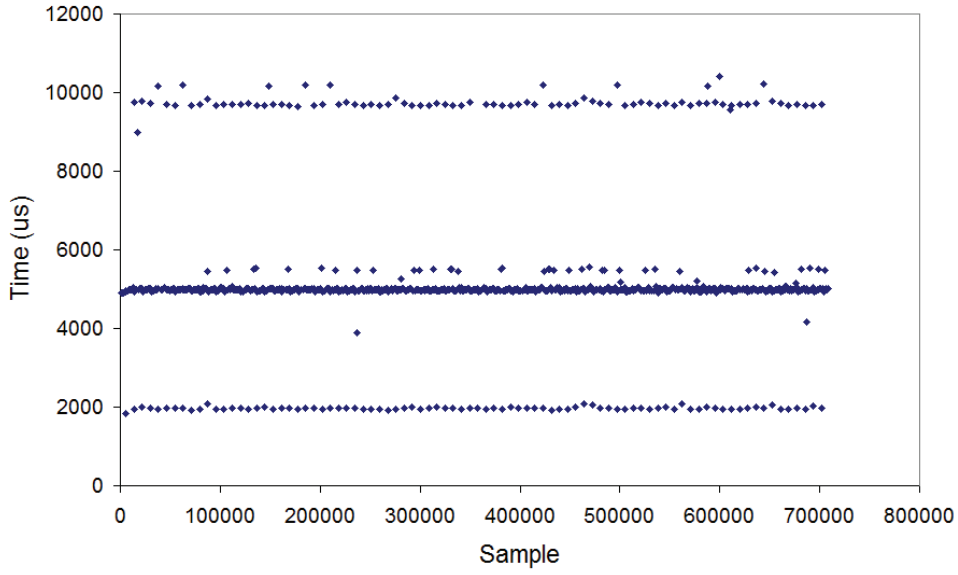

Figure 4.77: 512 B F2 Features (Block-Mode FTL, R1)

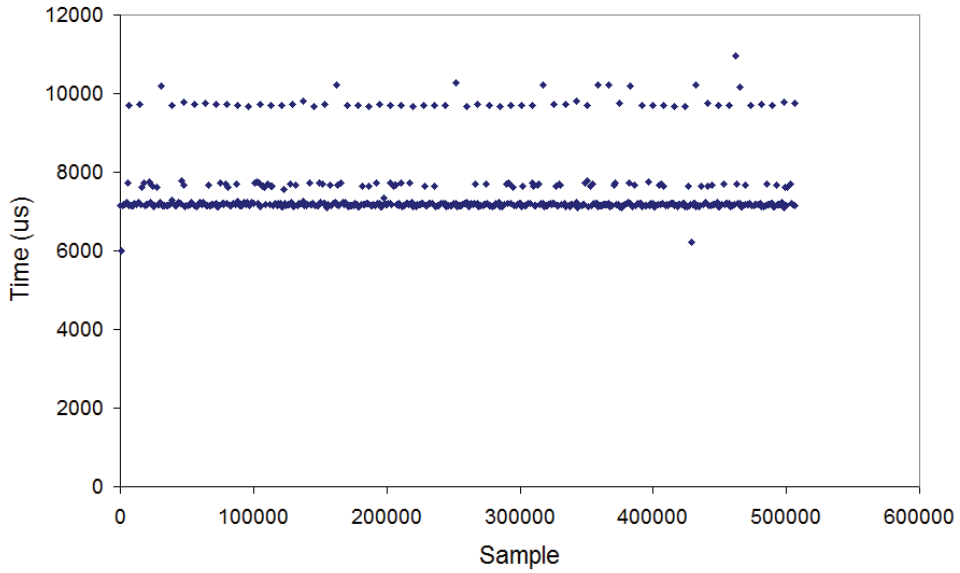

Figure 4.78: 4 KB F2 Features (Block-Mode FTL, R1)

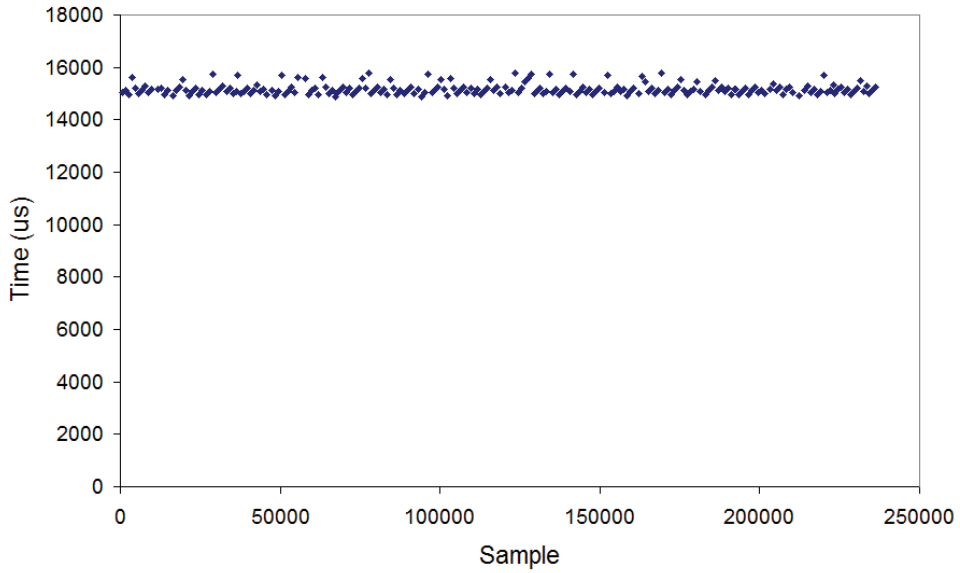

Figure 4.79: 16 KB F2 Features (Block-Mode FTL, R1)

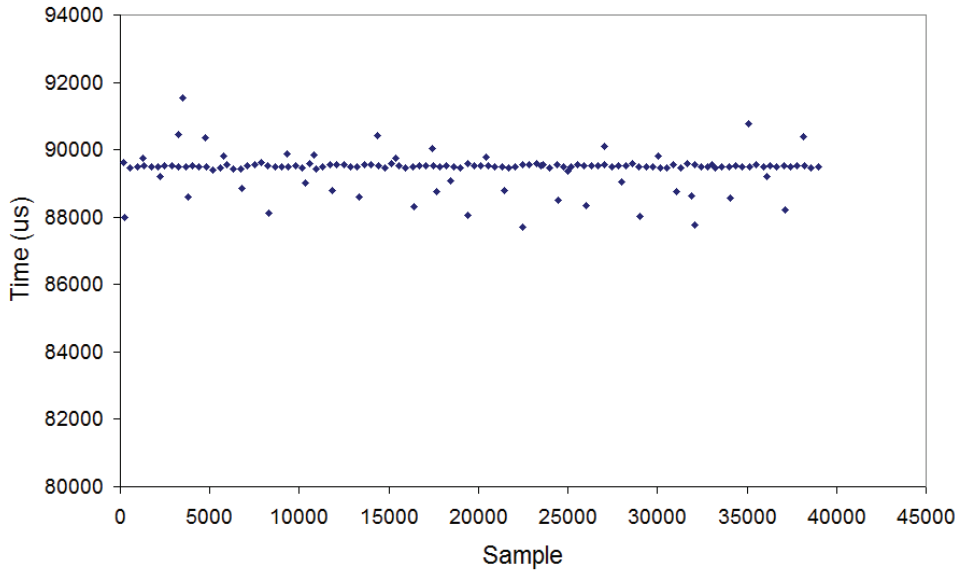

Figure 4.80: 128 KB F2 Features (Block-Mode FTL, R1)

Although some random data values are present, the plots show a highly consistent, periodic behavior (denoted F2), with a performance penalty of 5,000 us (5 ms) above the nominal. This feature largely appears as a horizontal line across the plots. As the given flash has a nominal block erase time of 5 ms (5000 us), these peaks are associated with a block erase. The number of F2 features is provided in Table 4.15.

It appears that data is being streamed in using a handler that appears sequential in nature. Specifically, it is observed that one block erase occurs every 32KB x 1024 = 32MB of host data.

While the sequential aspects of the handler is encouraging, the use of 32KB allocation units (instead of the previously discerned 4KB allocation units) is less so. This requires a WAF of 64 (at a minimum) for 512B blocks.

The FTL overhead data is being stored with the host data. This is presumed since F2 peaks occur at every 1016 transfers, instead of the 1024 transfers that we would

Transfer Size	F1 Frequency
512 B	1,016
1 KB	1,016
2 KB	1,016
4 KB	1,016
8 KB	1,016
16 KB	1,016
32 KB	1,016
64 KB	482
128 KB	384

Table 4.15: F2 Frequencies (Block-Mode FTL, R1)

expect for 32KB transfers. This 1% overhead is plausible for FTL management data.

However, the apparent use of a single handler for host data and FTL management data appears to be a poor choice. For every 10,000 transfers, the effect of three block copy events needs to occur for a delay of 1s (F1 events).

4.15.3 Repeated 2 (R2)

WAF measurements based on SMART data were made for the two fixed locations (R2) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.16.

Representative plots of performance measurements for the repeated (R2) test points (512B to 128KB) are shown in Figure 4.81 to Figure 4.84, respectively.

Transfer Size	WAF (Measured)
512 B	441.3
1 KB	220.6
2 KB	110.1
4 KB	55.28
8 KB	28.50
16 KB	14.87
32 KB	7.92
64 KB	4.46
128 KB	3.01

Table 4.16: WAF Measurements (Block-Mode FTL, R2)

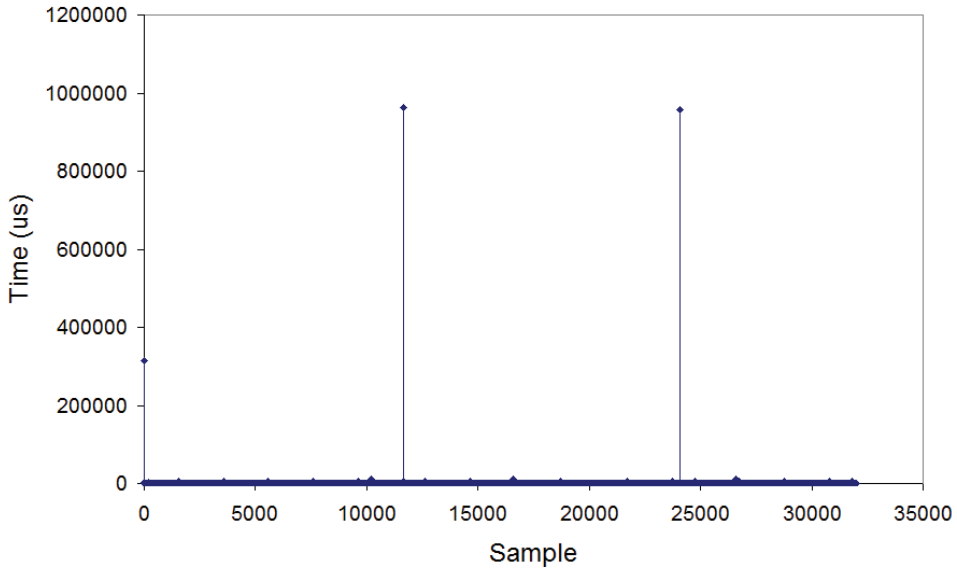

Figure 4.81: 512 B Performance Data (Block-Mode FTL, R2)

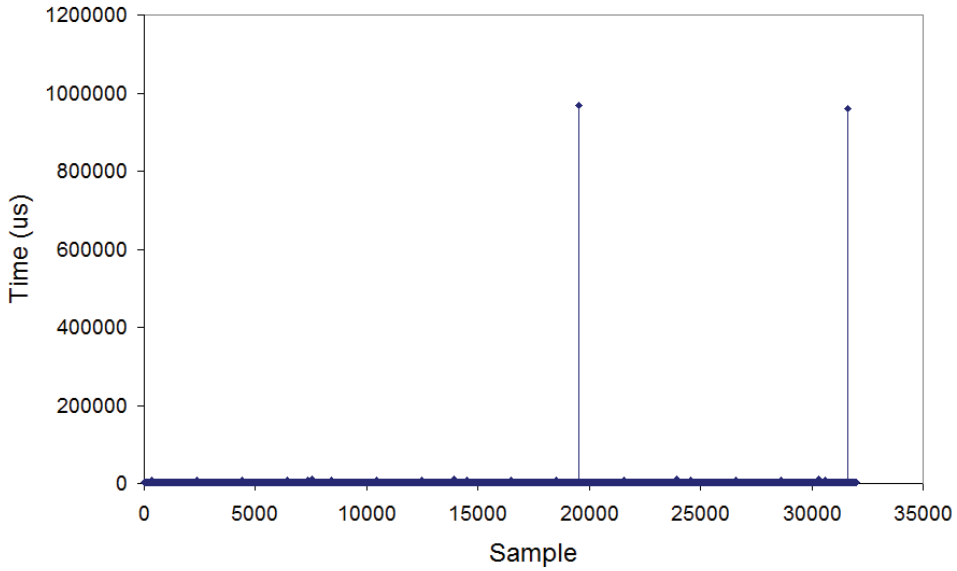

Figure 4.82: 4 KB Performance Data (Block-Mode FTL, R2)

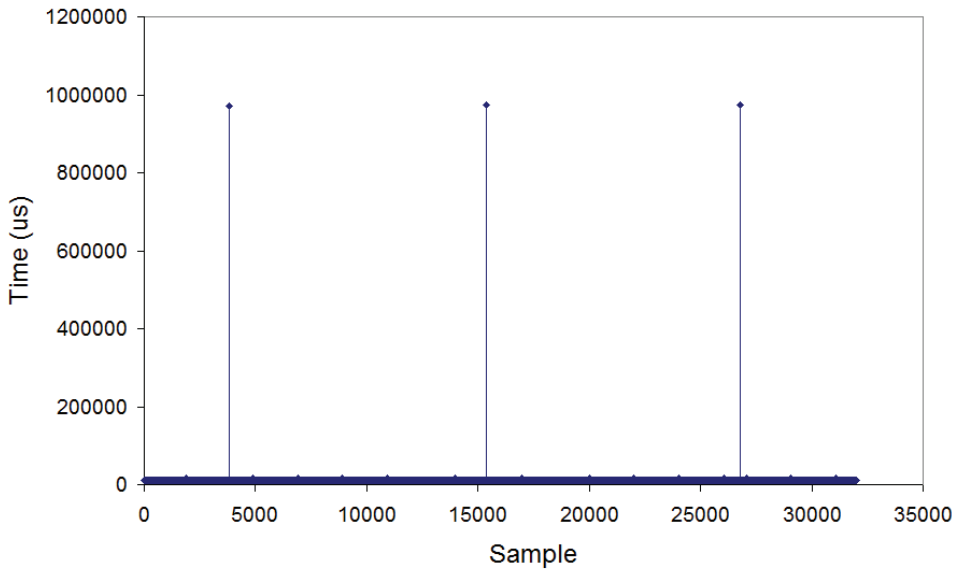

Figure 4.83: 16 KB Performance Data (Block-Mode FTL, R2)

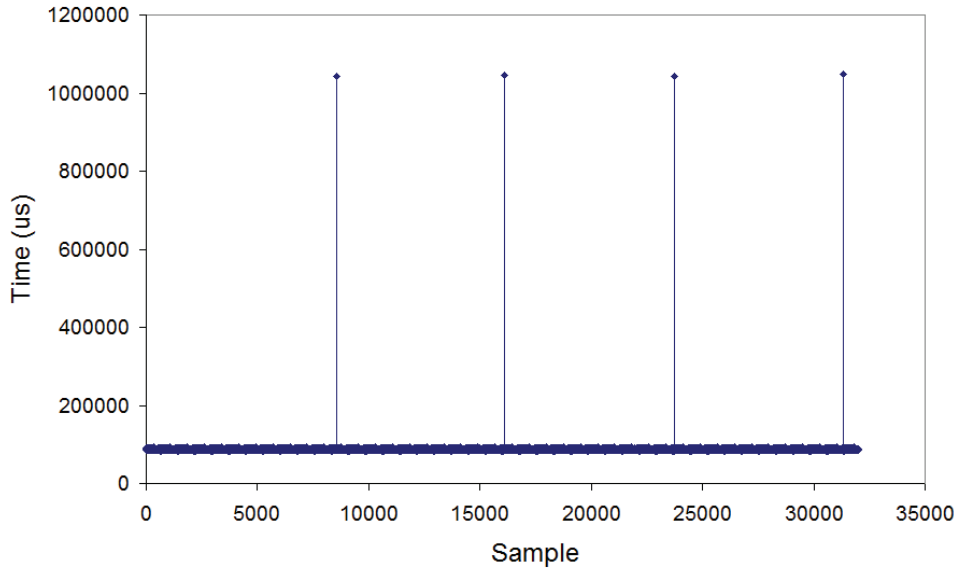

Figure 4.84: 128 KB Performance Data (Block-Mode FTL, R2)

4.15.4 F1 and F2 (R2)

As Figures 4.81-4.84 illustrate, there are transfers with considerable delays beyond nominal. These delays are on the order of 1 s. These spectral features are identified as F1. Representative plots of data filtered to show only F1 features are shown below in Figures 4.85-4.88. The transfer range (x-axis) has been adjusted to shown a small number of F1 events.

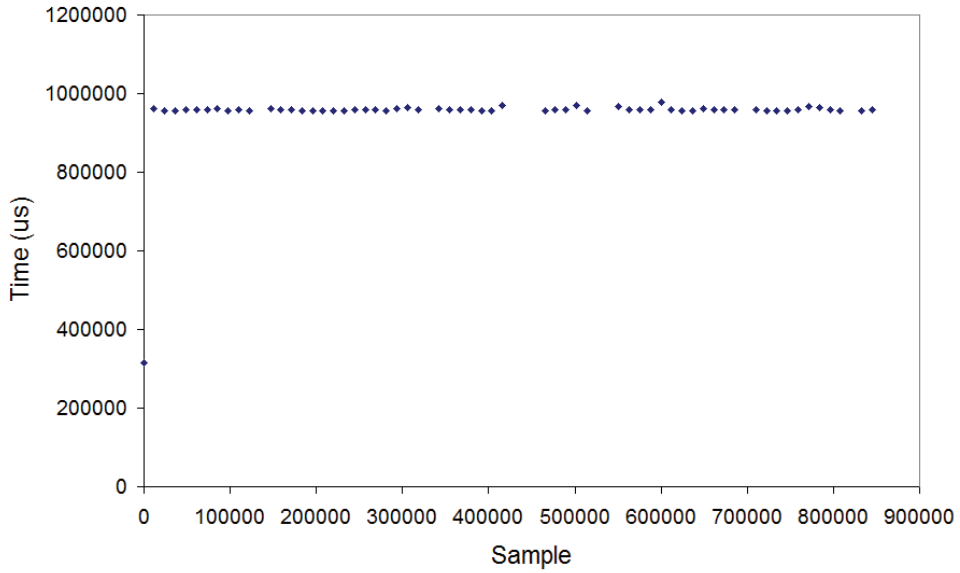

Figure 4.85: 512 B F1 Features (Block-Mode FTL, R2)

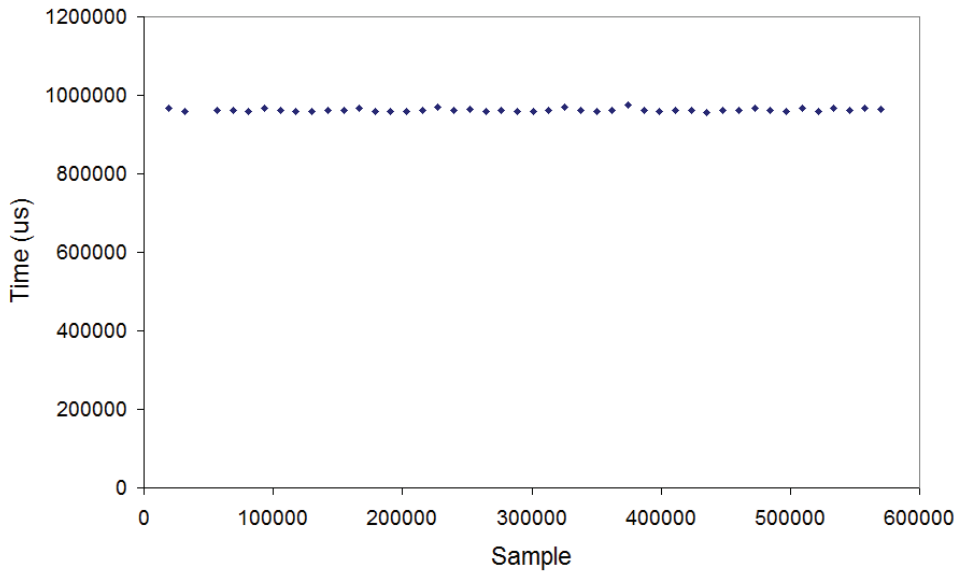

Figure 4.86: 4 KB F1 Features (Block-Mode FTL, R2)

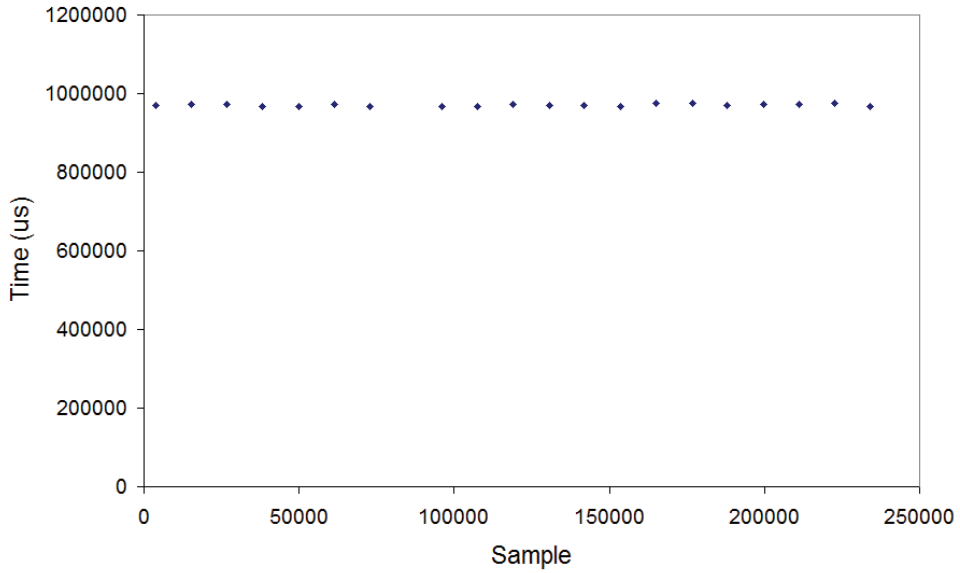

Figure 4.87: 16 KB F1 Features (Block-Mode FTL, R2)

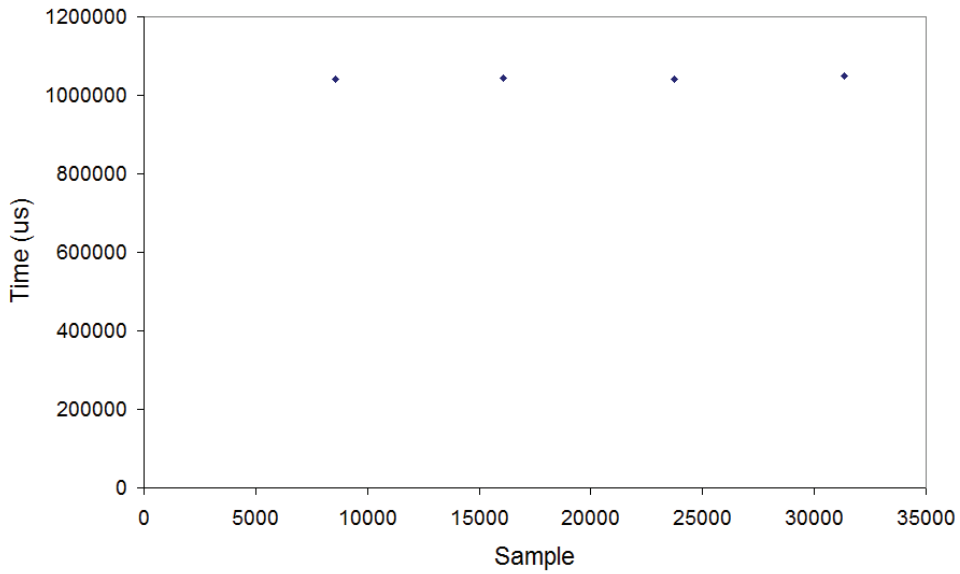

Figure 4.88: 128 KB F1 Features (Block-Mode FTL, R2)

The plots show a highly consistent, periodic feature (denoted F1) with a performance penalty of 1,000,000 us (1 s) above the nominal. This feature largely appears as a horizontal line across the plots.

From before, a block copy operation takes 362.4 ms. Three block copy operations take 1,092,600 us. This value is highly similar to the 1,000,000 us (1 s) observed for the F1 peaks. As such F1 is believed to be three block copy operations. As block copy may involve multiple areas in multiple chips, it is not expected that operations can be supported in parallel. The counts of F1 features appear below in Table 4.17.

Transfer Size	F1 Frequency
512 B	12,220
1 KB	12,240
2 KB	12,220
4 KB	12,230
8 KB	11,890
16 KB	11,510
32 KB	10,930
64 KB	9,833
128 KB	7,585

Table 4.17: F1 Frequencies (Block-Mode FTL, R2)

Continued analysis of the fine-grained performance data suggests a second periodic feature is also present. These delays are on the order of 1 s. These spectral features are identified as F2. Representative plots of data filtered to show only F2 features are shown below in Figure 4.89 to Figure 4.92. Transfer range (x-axis) has been adjusted to shown a small number of F2 events.

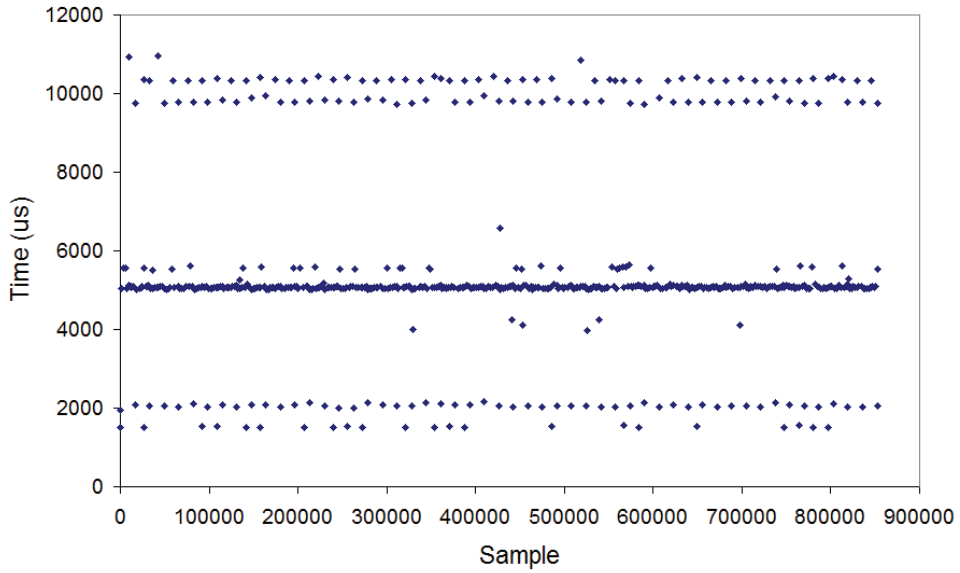

Figure 4.89: 512 B F2 Features (Block-Mode FTL, R2)

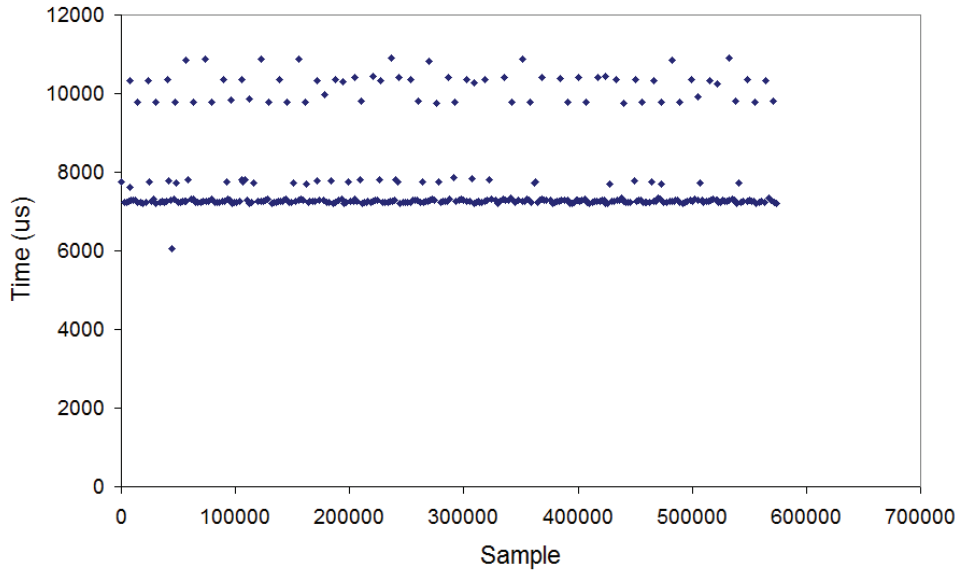

Figure 4.90: 4 KB F2 Features (Block-Mode FTL, R2)

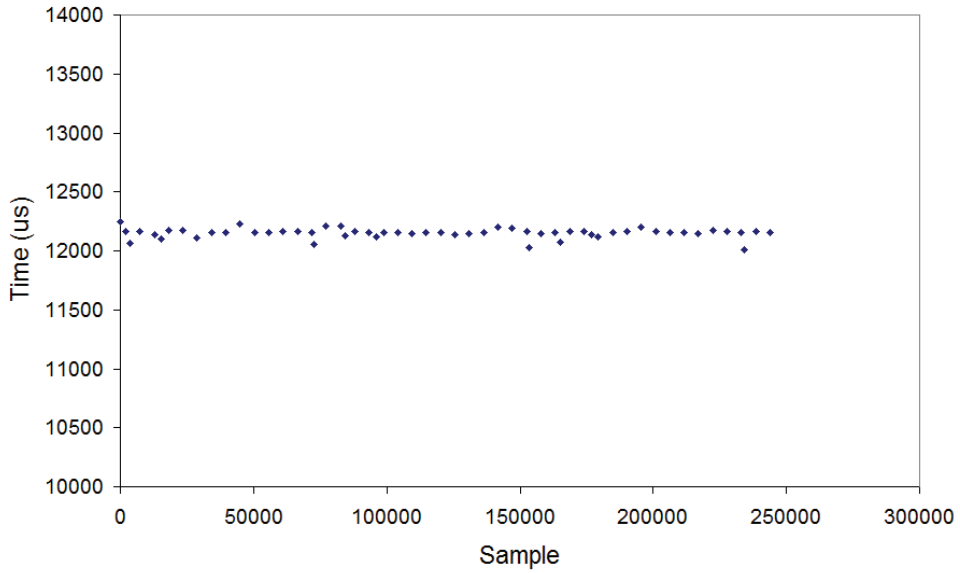

Figure 4.91: 16 KB F2 Features (Block-Mode FTL, R2)

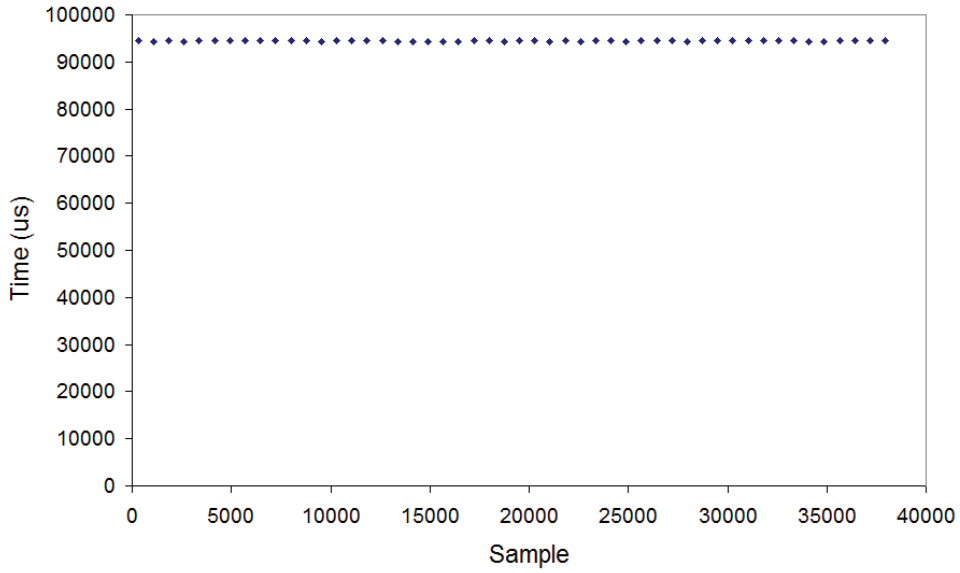

Figure 4.92: 128 KB F2 Features (Block-Mode FTL, R2)

Although some sporadic atypical measurements are present, the plots show a highly consistent, periodic feature (denoted F2) with a performance penalty of 5,000 us (5 ms) above the nominal. This feature largely appears as a horizontal line across the plots. As the given flash has a nominal block erase time of 5 ms (5000 us), these peaks are associated with a block erase. The counts of F2 features appear below in Table 4.18.

Transfer Size	F1 Frequency
512 B	2,018
1 KB	2,018
2 KB	2,018
4 KB	2,018
8 KB	2,018
16 KB	2,018
32 KB	2,016
64 KB	1,792
128 KB	768

Table 4.18: F2 Frequencies (Block-Mode FTL, R2)

It appears that data is being streamed in using a handler that appears sequential in nature. Specifically, it is observed that one block erase occurs every 32 KB x 2018 = 64 MB of host data.

While the sequential aspects of the handler is encouraging, the use of 32 KB allocation units (instead of the previously discerned 4 KB allocation units) is less so. This requires a WAF of 64 (at a minimum) for 512 B blocks.

It appears that the FTL overhead data is being stored with the host data. This is suspected as F2 peaks are every 2018 transfers, instead of the 2048 transfers that would be expected for 32 KB transfers. This 1% overhead is plausible for FTL management data.

However, the apparent use of a single handler for host data and FTL management data appears to be a poor choice. Every 12000 transfers, the effect of three block copy events need to occur for a delay of 1s (F1 events).

4.15.5 Repeated 3 (R3)

WAF measurements based on SMART data were made for the three fixed locations (R3) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.19.

Transfer Size	WAF (Measured)
512 B	301.6
1 KB	151.0
2 KB	75.39
4 KB	37.80
8 KB	19.44
16 KB	10.32
32 KB	5.669
64 KB	3.432
128 KB	2.479

Table 4.19: WAF Measurements (Block-Mode FTL, R3)

Representative plots of performance measurements for the repeated (R3) test points (512 B to 128 KB) are shown in Figure 4.93 to Figure 4.96, respectively.

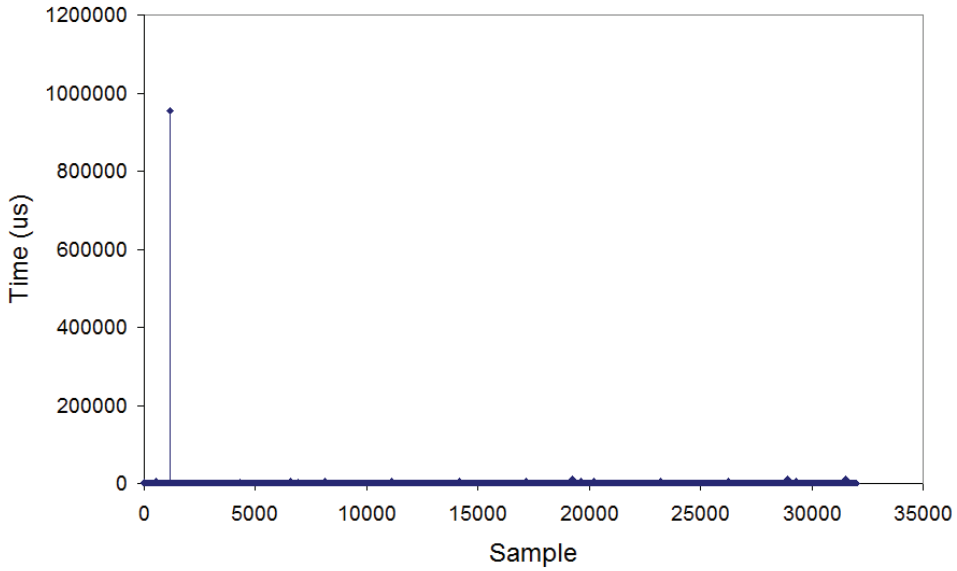

Figure 4.93: 512 B Performance Data (Block-Mode FTL, R3)

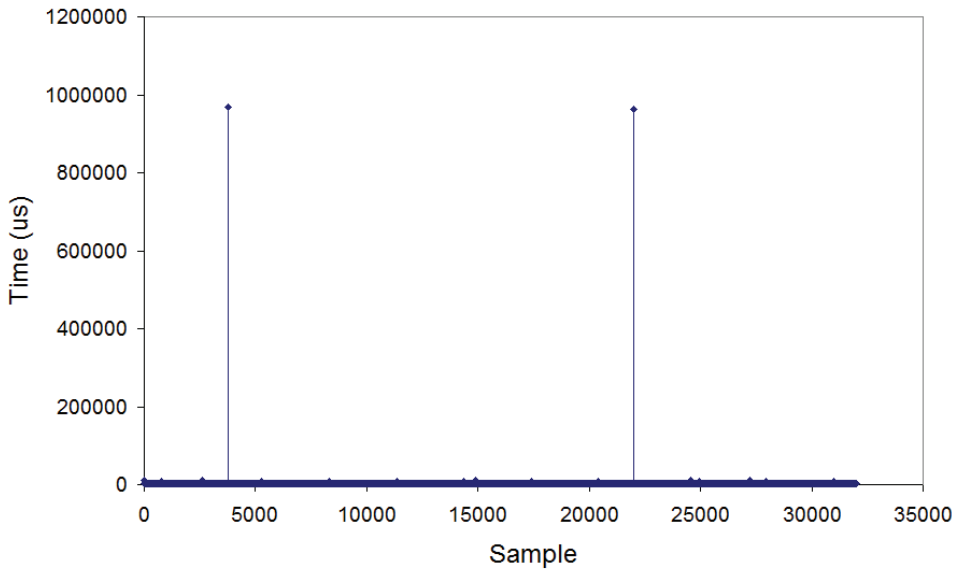

Figure 4.94: 4 KB Performance Data (Block-Mode FTL, R3)

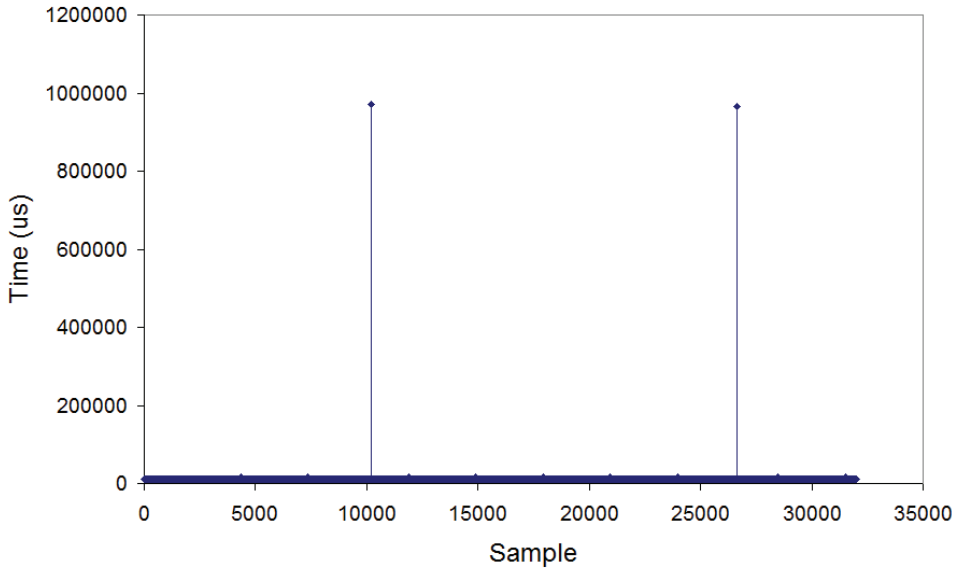

Figure 4.95: 16 KB Performance Data (Block-Mode FTL, R3)

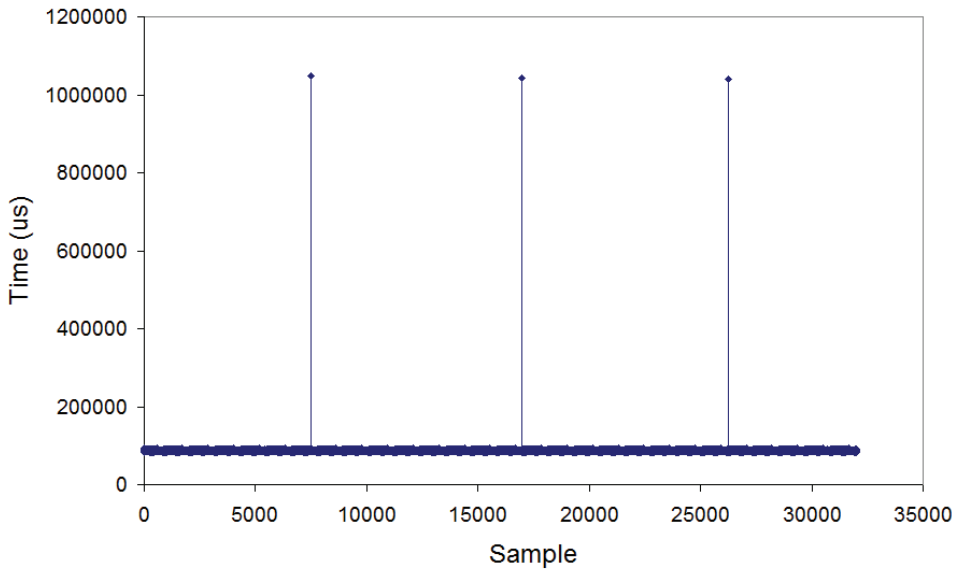

Figure 4.96: 128 KB Performance Data (Block-Mode FTL, R3)

4.15.6 F1 and F2 (R3)

As Figure 4.93 to Figure 4.96 illustrate, there are transfers with considerable delays beyond nominal. These delays are on the order of 1 s. These spectral features are identified as F1. Representative plots of data filtered to show only F1 features are shown below in Figure 4.97 to Figure 4.100. Transfer range (x-axis) has been adjusted to shown a small number of F1 events.

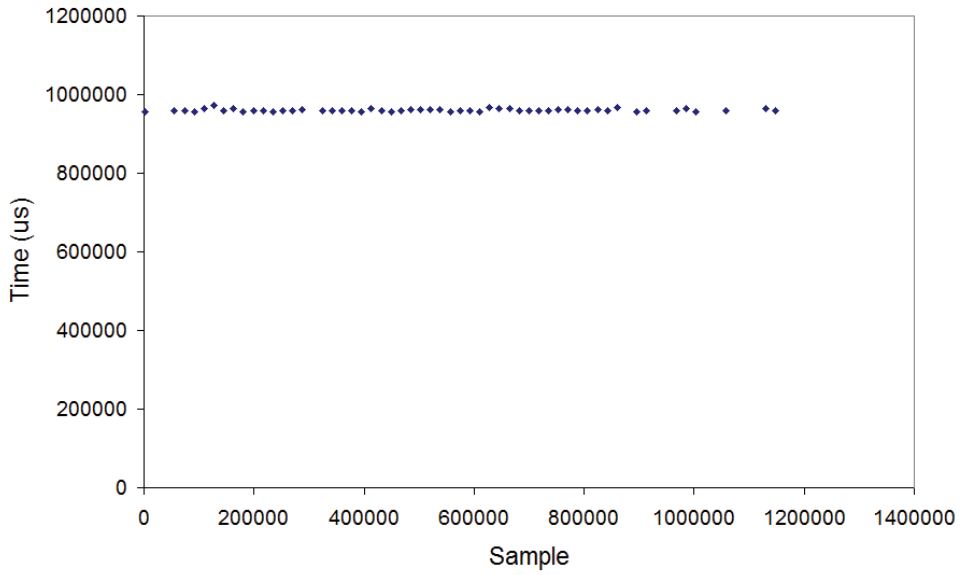

Figure 4.97: 512 B F1 Features (Block-Mode FTL, R3)

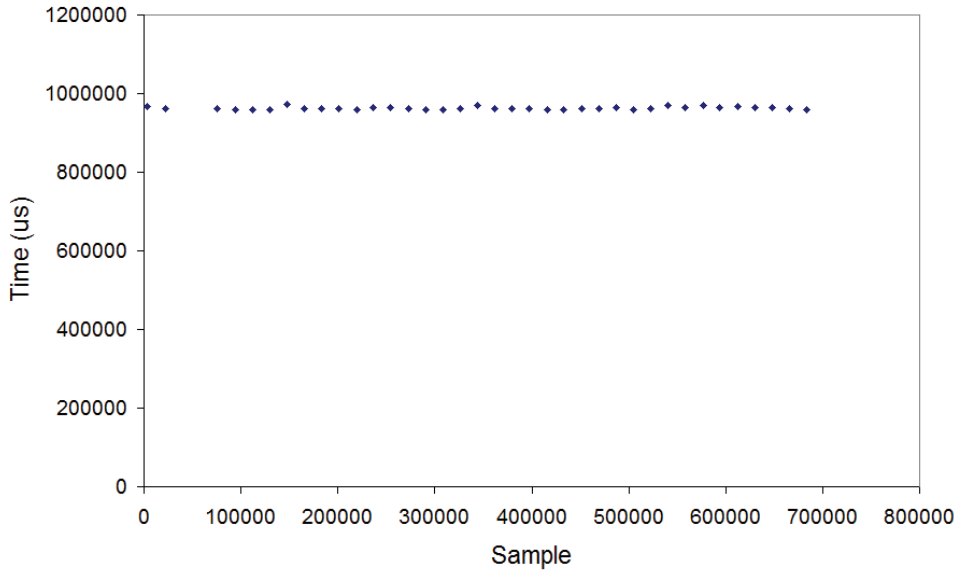

Figure 4.98: 4 KB F1 Features (Block-Mode FTL, R3)

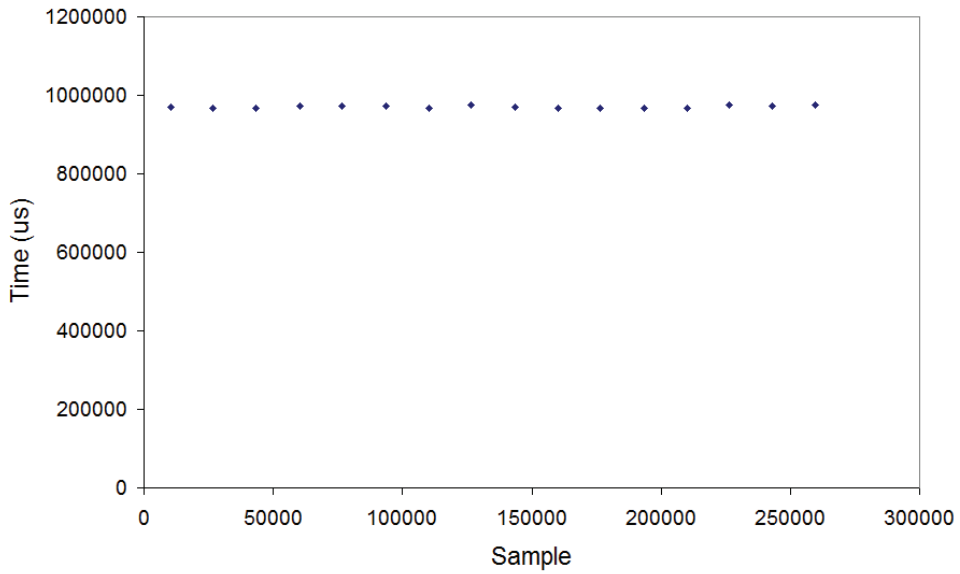

Figure 4.99: 16 KB F1 Features (Block-Mode FTL, R3)

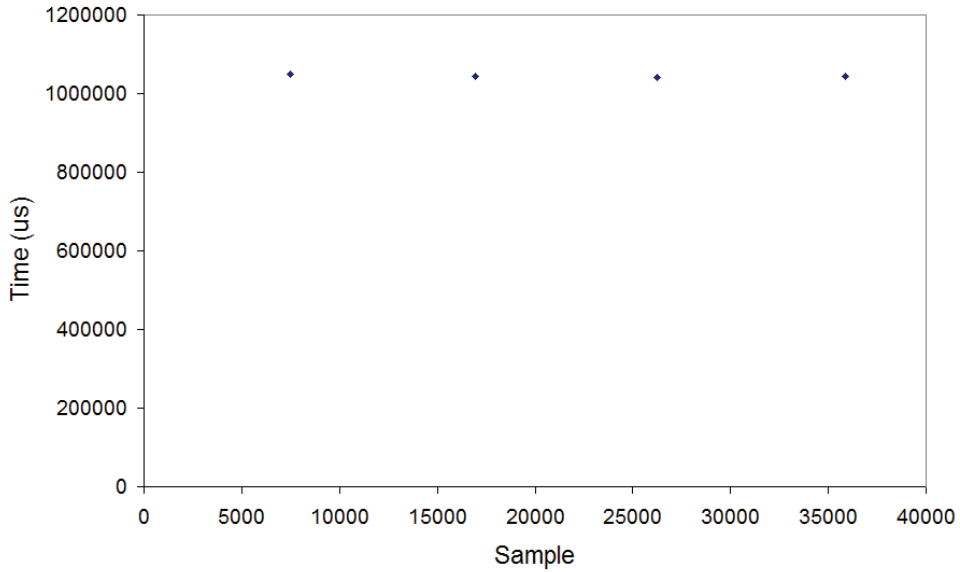

Figure 4.100: 128 KB F1 Features (Block-Mode FTL, R3)

The plots show a highly consistent, periodic feature (denoted F1) with a performance penalty of 1 s above the nominal. This feature largely appears as a horizontal line across the plots.

From before, a block copy operation takes 362.4 ms. Three block copy operations take 1,092,600 us. This value is very close to the 1 s observed for the F1 peaks. As such F1 is believed to be three block copy operations. As block copy may involve multiple areas in multiple chips, it is not expected that operations can be supported in parallel. The counts of F1 features appear below in Table 4.20.

Continued analysis of the fine-grained performance data suggests a second periodic feature is also present. These delays are on the order of 1 s. These spectral features are identified as F2. Representative plots of data filtered to show only F2 features are shown in Figures 4.101-4.104. The transfer range (x-axis) has been adjusted to shown a small number of F2 events.

Transfer Size	F1 Frequency
512 B	17,890
1 KB	17,870
2 KB	17,870
4 KB	17,890
8 KB	17,420
16 KB	16,630
32 KB	15,330
64 KB	13,205
128 KB	9,470

Table 4.20: F1 Frequencies (Block-Mode FTL, R3)

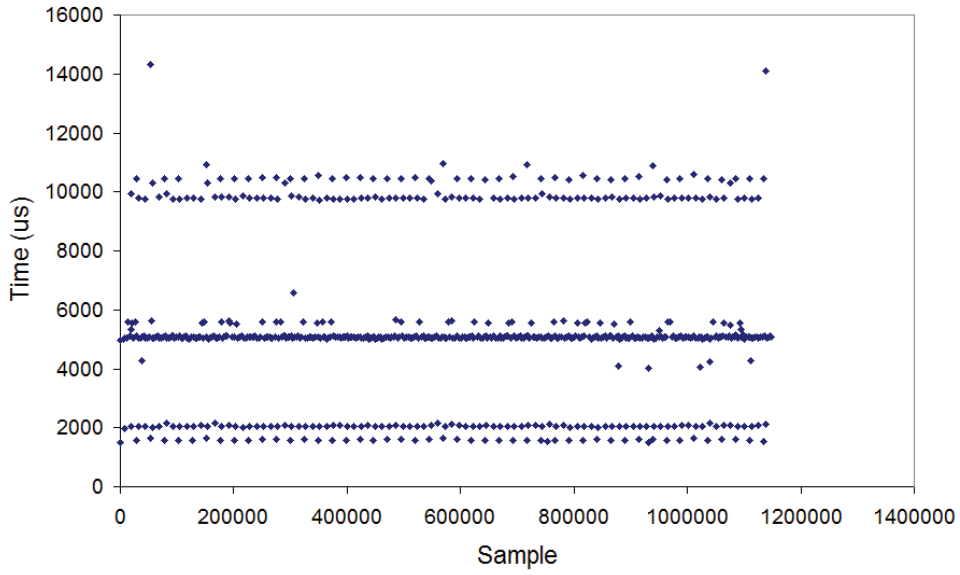

Figure 4.101: 512 B F2 Features (Block-Mode FTL, R3)

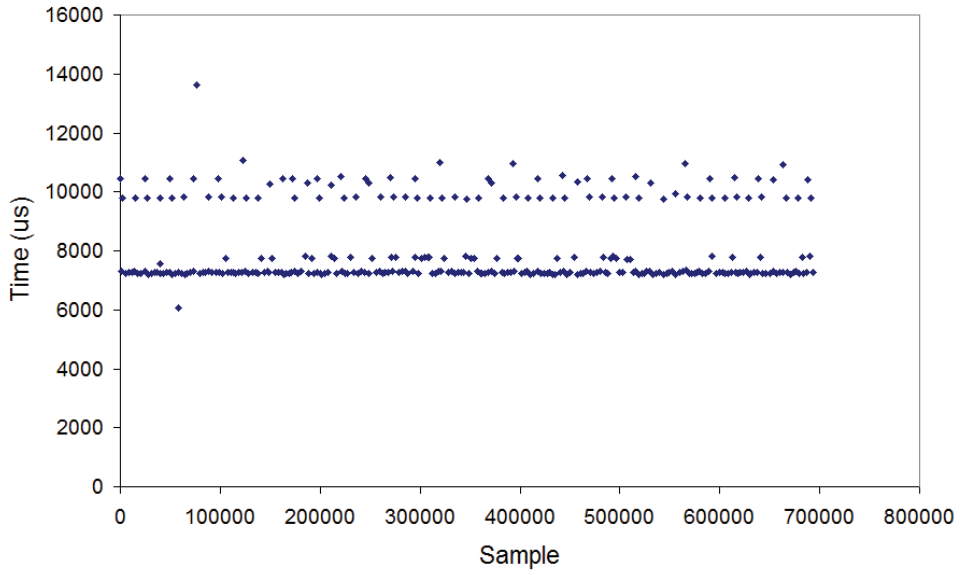

Figure 4.102: 4 KB F2 Features (Block-Mode FTL, R3)

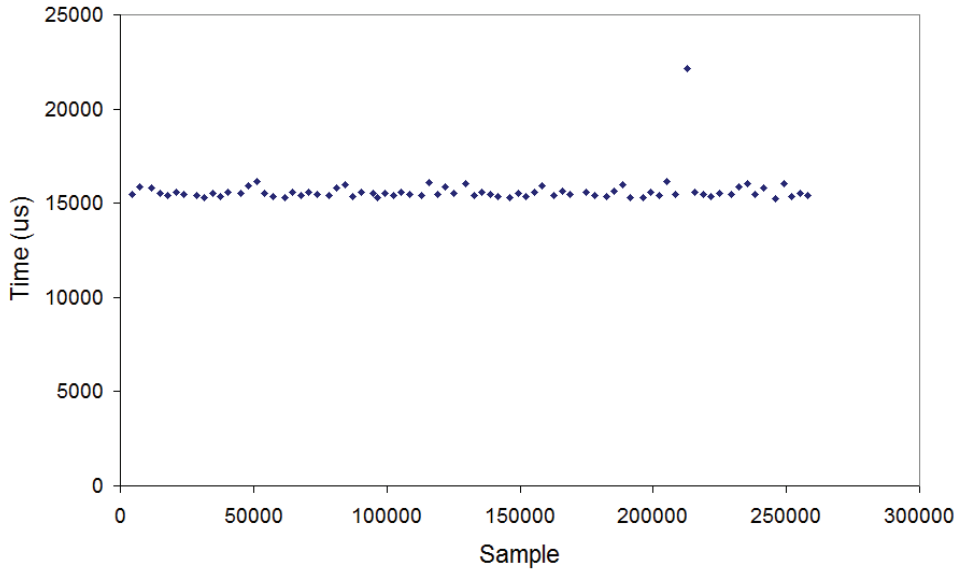

Figure 4.103: 16 KB F2 Features (Block-Mode FTL, R3)

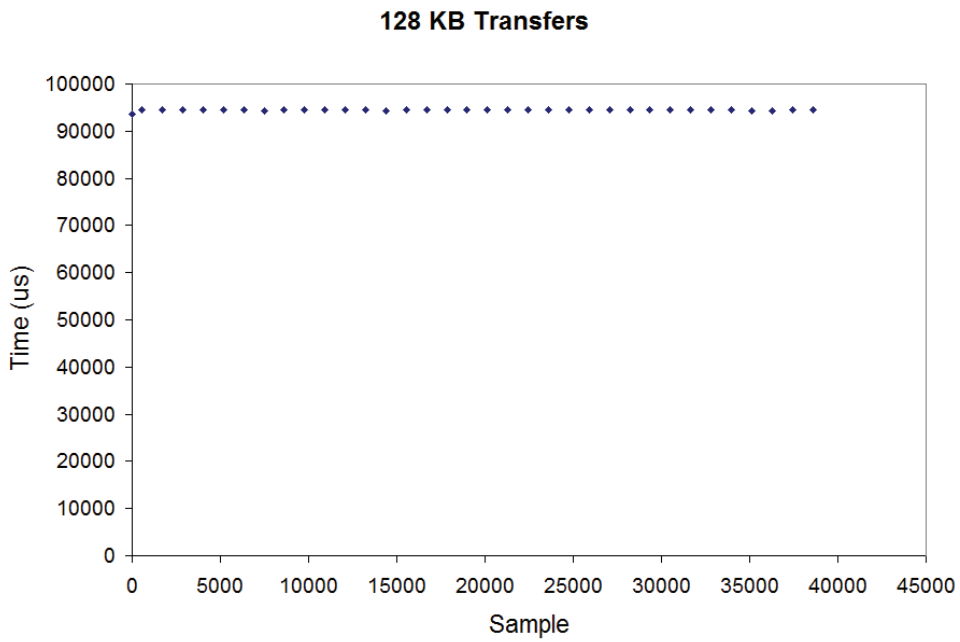

Figure 4.104: 128 KB F2 Features (Block-Mode FTL, R3)

Although some random data points are present, the plots show a highly consistent, periodic behavior (denoted F2), with a performance penalty of 5 ms above the nominal. This feature largely appears as a horizontal line across the plots. As the given flash has a nominal block erase time of 5 ms (5000 us), these peaks are associated with a block erase. The counts of F2 features appear below in Table 4.21.

It appears that data is being streamed in using a handler that appears sequential in nature. Specifically, it is observed that one block erase occurs every 32KB x 3027 = 96MB of host data.

While the sequential aspects of the handler is encouraging, the use of 32KB allocation units (instead of the previously discerned 4KB allocation units) is less so. This requires a WAF of 64 (at a minimum) for 512B blocks.

It appears that the FTL overhead data is being stored with the host data. This is suspected as F2 peaks are every 3027 transfers, instead of the 3072 transfers that

Transfer Size	F2 Frequency
512 B	3,027
1 KB	3,027
2 KB	3,027
4 KB	3,027
8 KB	3,027
16 KB	3,021
32 KB	3,021
64 KB	2,688
128 KB	1,152

Table 4.21: F2 Frequencies (Block-Mode FTL, R3)

would be expected for 32KB transfers. This 1% overhead is plausible for FTL management data.

However, the apparent use of a single handler for host data and FTL management data appears to be a poor choice. Every 12,000 transfers, the effect of three block copy events need to occur for a delay of 1s (F1 events).

4.16 Measurements (Block-Mode FTL, File)

Considering file creation as three repeated 512B writes and one sequential write (recall Section 2.6), the flash memory system with the block-mode FTL is characterized for file creation using the previously developed technique. Specifically, the drive is subjected to an array of tests consisting of a continuous combination of three fixed (repeated) write operations of 512B (to simulate FAT and directory meta-data) and a fixed transfer size sequential write request (to simulate file data) for each test.

As in the previous sections, characterization consists of an array of individual test points with testing at each point conducted for one hour. Extended testing, when required for improved resolution, is conduced for eight hours. Overall, tests using 512B, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, and 128KB file sizes have been conducted.

WAF measurements based on SMART data were made for the file test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.22.

Transfer Size	WAF (Measured)
512 B	274.9
1 KB	231.9
2 KB	177.6
4 KB	125.3
8 KB	74.66
16 KB	45.83
32 KB	25.20
64 KB	13.51
128 KB	8.17

Table 4.22: WAF Measurements (Block-Mode FTL, File)

Representative plots of performance measurements for file test points (512B to 128KB) are shown in Figures 4.105-4.108, respectively

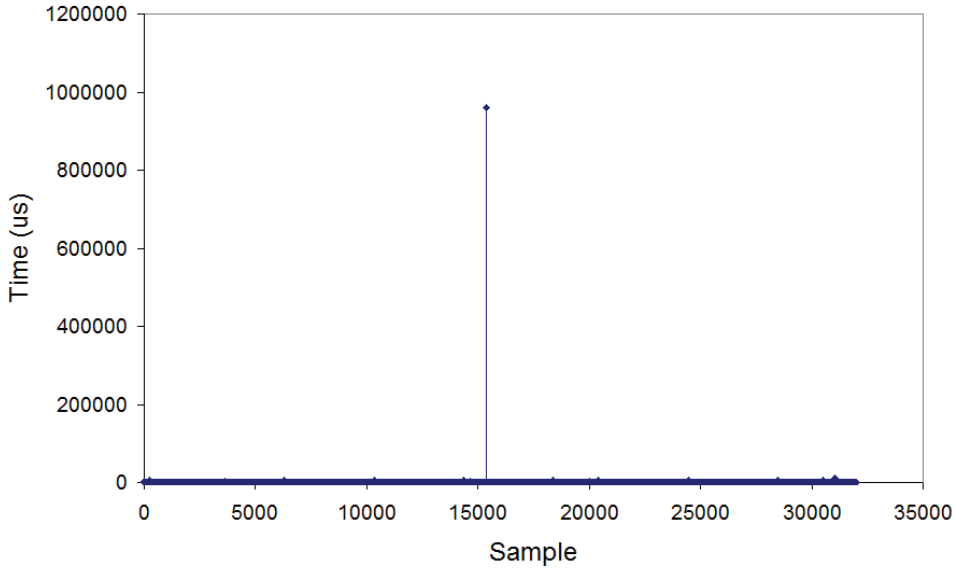

Figure 4.105: 512 B Performance Data (Block-Mode FTL, File)

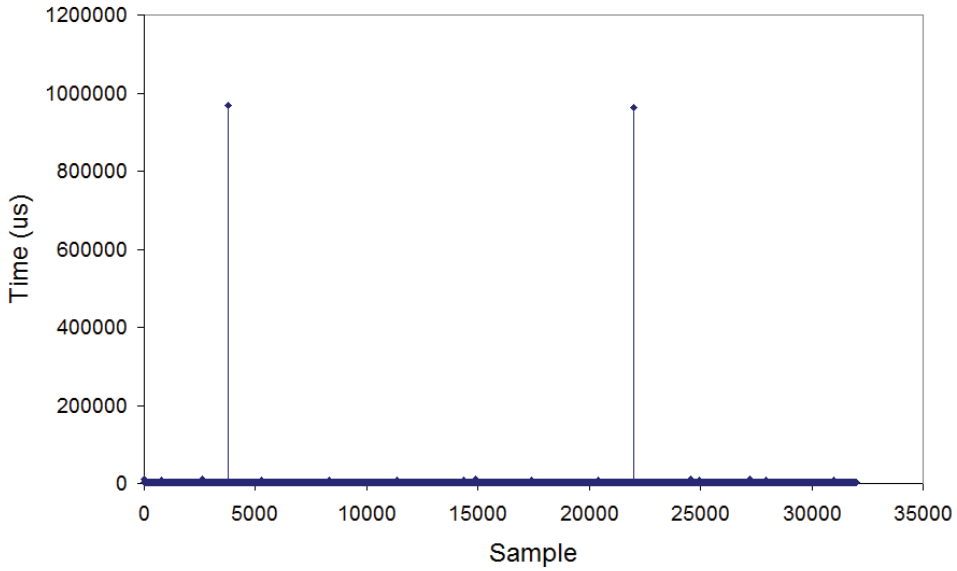

Figure 4.106: 4 KB Performance Data (Block-Mode FTL, File)

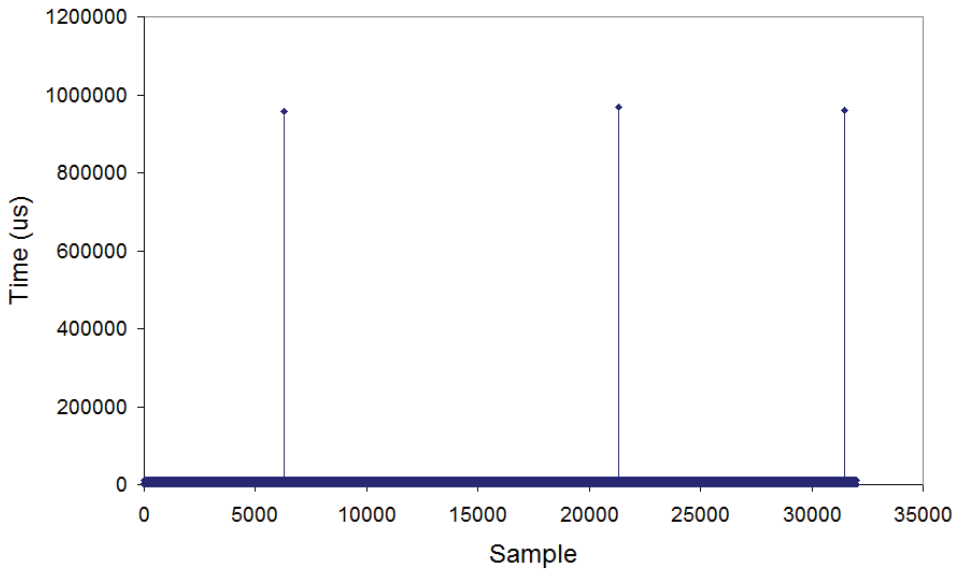

Figure 4.107: 16 KB Performance Data (Block-Mode FTL, File)

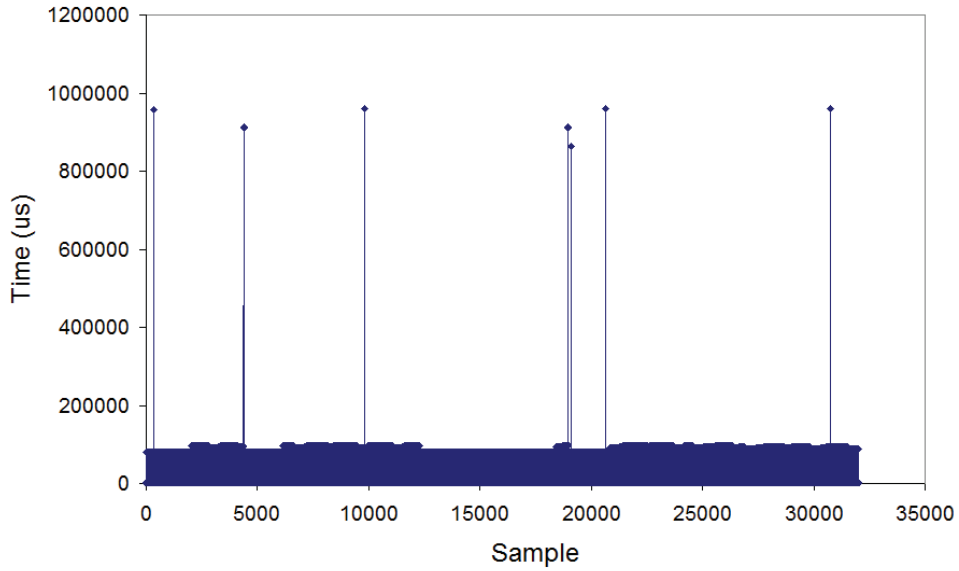

Figure 4.108: 128 KB Performance Data (Block-Mode FTL, File)

4.16.1 F1 and F2 (R1)

As Figures 4.105-4.108 illustrate, there are transfers with considerable delays beyond nominal. These delays are on the order of 1 s. These spectral features are identified as F1. Representative plots of data filtered to show only F1 features are presented in Figure 4.109 to Figure 4.112. The transfer range (x-axis) has been adjusted to shown a small number of F1 events.

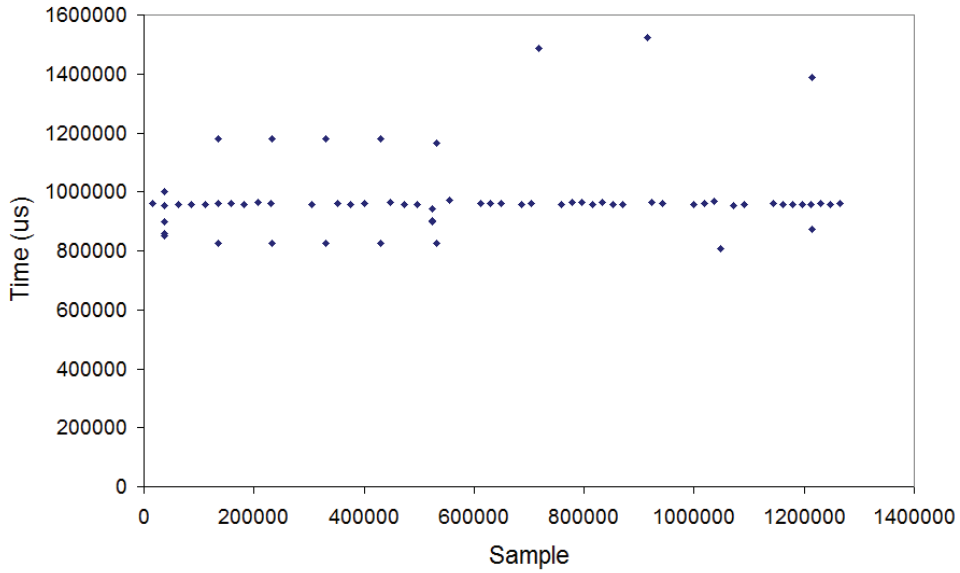

Figure 4.109: 512 B F1 Features (Block-Mode FTL, File)

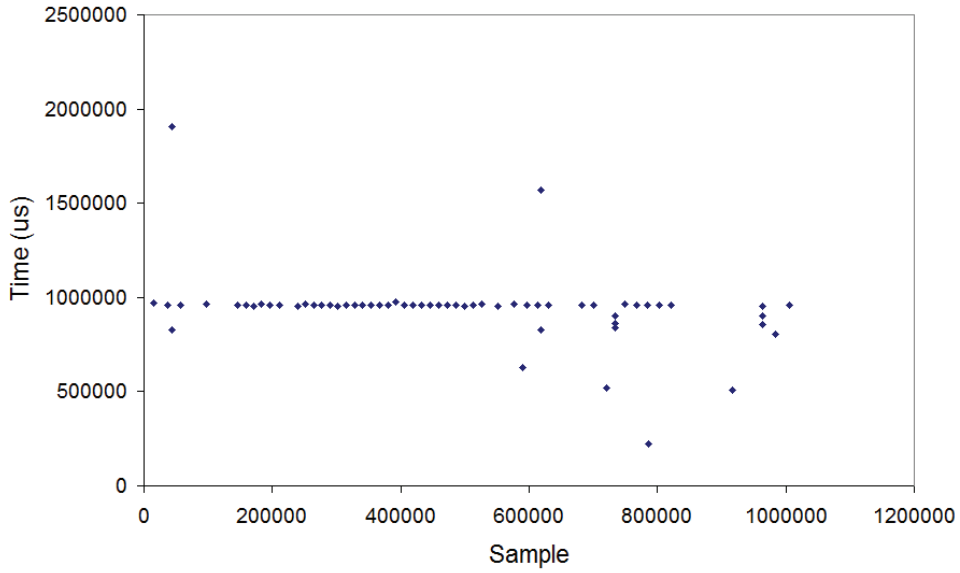

Figure 4.110: 4 KB F1 Features (Block-Mode FTL, File)

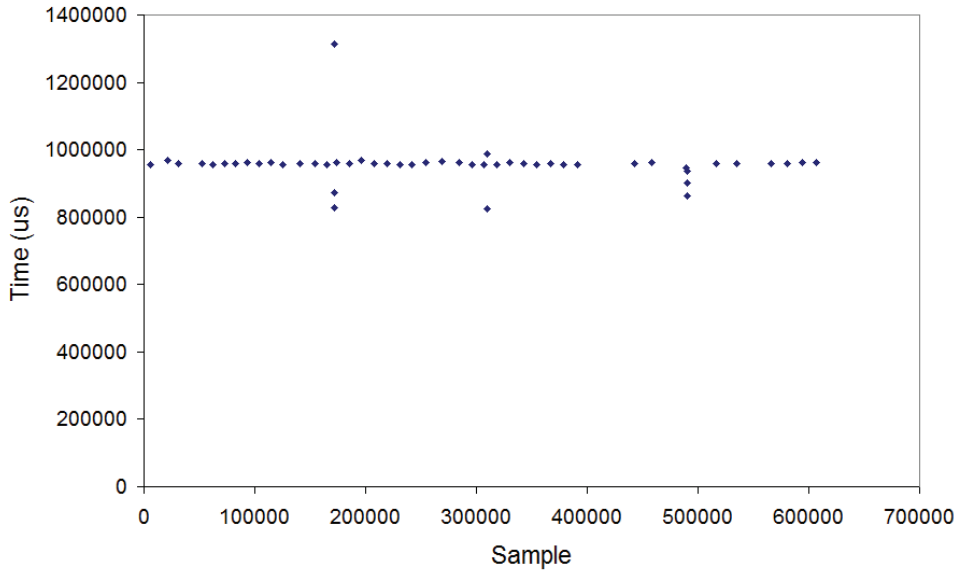

Figure 4.111: 16 KB F1 Features (Block-Mode FTL, File)

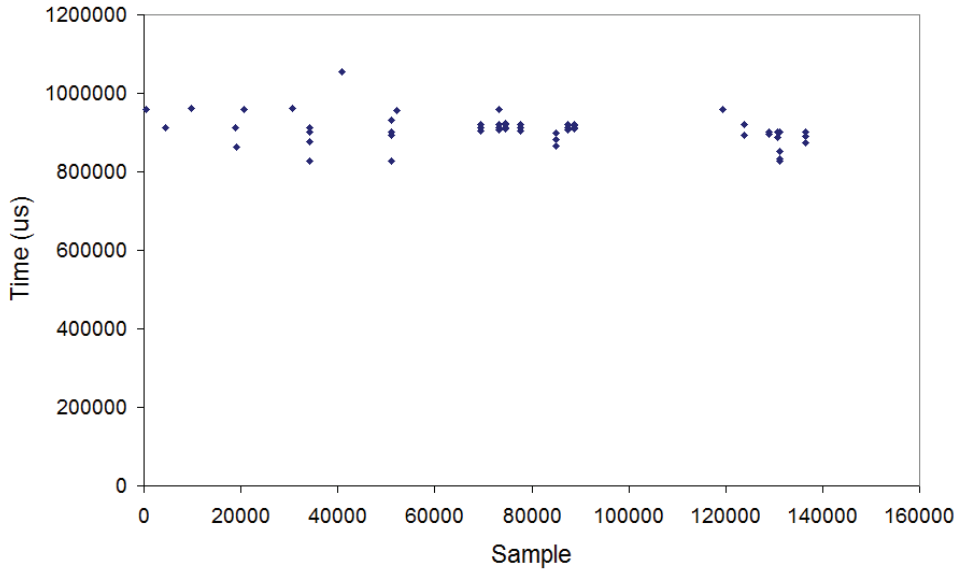

Figure 4.112: 128 KB F1 Features (Block-Mode FTL, File)

Although some random data points are present, the plots show a highly consistent, periodic feature (denoted F1), with a performance penalty of 1 s above the nominal. This feature largely appears as a horizontal line across the plots.

From before, a block copy operation takes 362.4 ms. Three block copy operations take 1,092,600 us. This value tracks the 1 s observed for the F1 peaks. As such, F1 is believed to be three block copy operations. As block copy may involve multiple areas in multiple chips, it is not expected that operations can be supported in parallel. The counts of F1 features appear below in Table 4.23.

Transfer Size	F1 Frequency
512 B	17,850
1 KB	16,780
2 KB	16,740
4 KB	14,380
8 KB	15,690
16 KB	11,780
32 KB	8,104
64 KB	5,639
128 KB	1,840

Table 4.23: F1 Frequencies (Block-Mode FTL, File)

Given the variations in transfer sizes (three 512B and one other), it does not appear to be possibly to readily isolate the F2 peaks as the nominal baseline varies dependent upon the specific transfer size. However, as previously discussed, F2 peaks are characterized as block erases at a frequency of filling 32KB allocation units. Moreover, the effect on WAF is exclusively the result of 32KB allocation units.

4.17 Empirical Model and WAF Equations (Block-Mode FTL, Repeated and File)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

The WAF measurements for random write requests (Section 4.6), repeated write requests (Section 4.15), and file write requests (Section 4.16) for the drive with the block-mode FTL are collectively summarized below in Figure 4.113.

As Figure 4.113 indicates, the repeated and file WAF measurements are generally larger than those observed for random writes. A negative coupling effect has been previously observed with the block-mode FTL as it transitions between support for sequential and random write requests (recall Section 4.7). A similar negative coupling is present for repeated writes.

Theoretically, WAF is the collective result of effectively copying data (host written), copying data (garbage collections), data copying from consolidation, and copying of FTL overhead scaled by the host written data. The WAF equation can be written

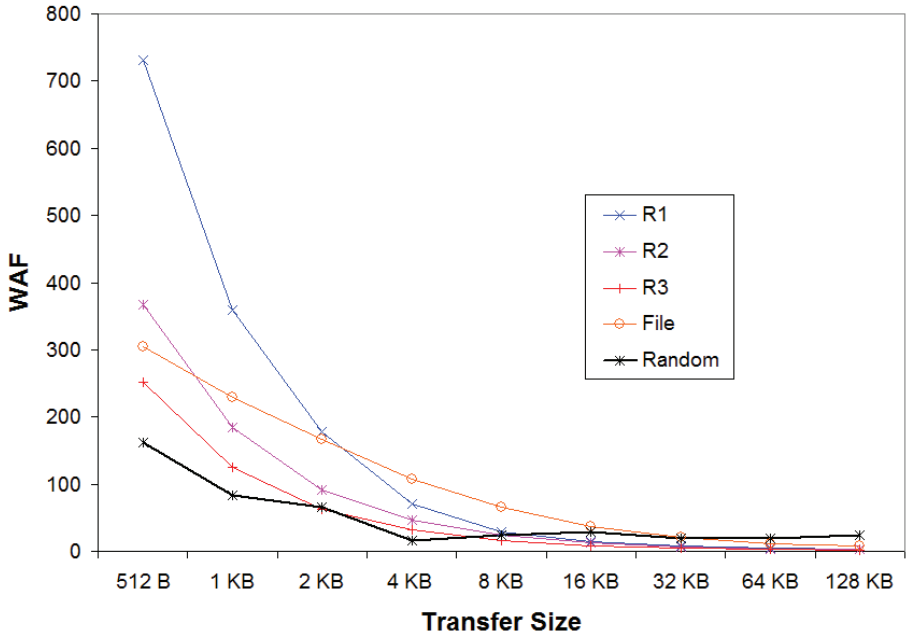

Figure 4.113: WAF Summary (Block-Mode FTL, Repeated and File)

as shown in Figure 4.114.

$$WAF = \frac{Data_{Written,Effective} + Data_{GC} + Consolidation + FTL\,Overhead}{Data_{Written}} \tag{4.21}$$

Figure 4.114: WAF Equation (Theoretical)

Considering repeated writes (R1, R2, and R3) on a block-mode FTL, there is no need for garbage collection in the traditional sense. Specifically each actively written block becomes fully obsolete shortly after a new active block is addressed. By the time another new active block is needed, the original active block fully contains invalid data and need only be erased. Specific to the equation above, the effects of copying data from traditional garbage collection becomes zero and can be disregarded.

Additionally, the FTL overhead has been measured as 1% (recall F2 features in Section 4.15). This may be disregarded as a minor effect and the above equation of WAF may be reduced as shown in Figure 4.115

$$WAF = \frac{Data_{Written,Effective} + Consolidation}{Data_{Written}} \tag{4.22}$$

Figure 4.115: WAF Equation (Block-Mode FTL, Theoretical)

Considering that an F1 event takes around 1 s and recalling that a block copy time is 362.4 ms, an F1 event is considered to be three block copy events. Recalling that a block is 2MB, a F1 event involves copying 6MB of data. For the measured number of transfers per F1 event, this calculated overhead becomes approximately 0.5KB/transfer. Calculating WAF with this value for consolidation leads to WAF (4KB) values of 9.122, 8.561, and 8.374 which are again significantly below the measured values.

Given the consistent under-prediction of WAF measurements, it is concluded that the time signature detected with the FTLProbe technique is incapable of properly quantifying the amount of data involved with a consolidation event. Instead, the WAF measurements are compared with the values predicted by the above equation. Specifically, values of consolidation data are selected to fit the measured data. An example with the amount of consolidation data set to a constant value of 350KB is shown in Figure 4.116.

While the fitting of data to determine coefficients for the WAF equations is undesirable for the current work, the large magnitude of the required value is illuminating. Specifically, the amount consolidation data is unexpectedly large. While the literature has highlighted the limitations of hybrid block-mode FTLs, particularly BAST

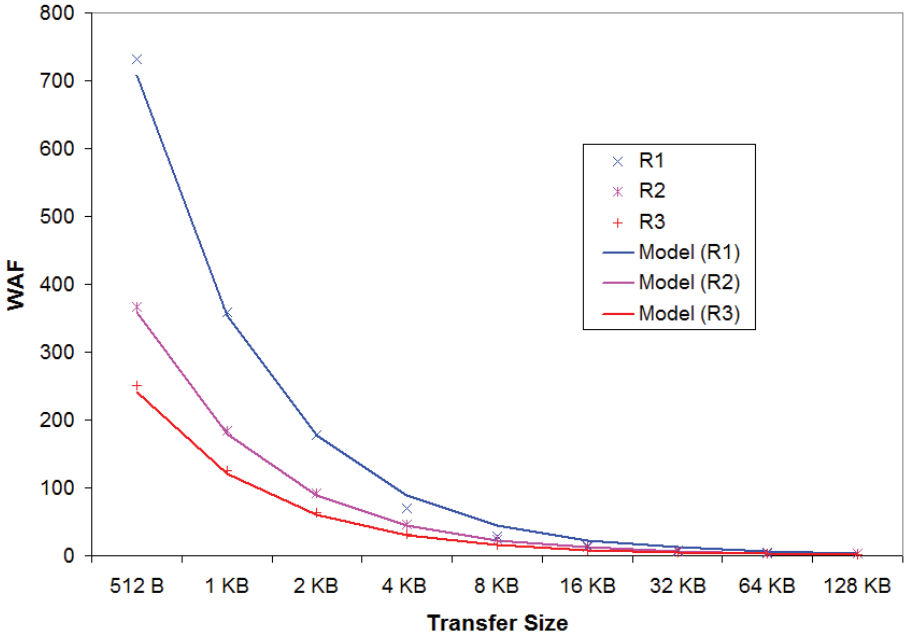

Figure 4.116: WAF Fitting (Block-Mode FTL, Repeated)

based, with use of the term "thrash", the magnitude of impact, particularly the high WAF values, has not been effectively quantified.

In addition to the magnitude of the Consolidation value, the constant nature of the value is also considered illuminating. Specifically, the constant nature of the value suggests that it is related to the number of transfers and not the amount of data. Because of this, it is concluded that the consolidation effect is related to limitations within the FTL itself. Specifically, it is suspected that the consolidation event is related to a critical resource within the FTL, such as number of entries in a journaled lookup table.

Adapting the previous equation to compound write operations, such as occurs with a file create operation, the equation for WAF becomes as shown in Figure 4.117

$$WAF_{Compound} = \frac{\sum(Data_{Written,Effective}) \ + \ Consolidation}{\sum(Data_{Written})} \tag{4.23}$$

Figure 4.117: Compound WAF Equation (Block-Mode FTL, Theoretical)

For a FAT file system, which has three 512B meta-data writes per write of file data, this equation can be rewritten as shown in Figure 4.118.

$$WAF = \frac{(3 * 4\,KB \ + \ File_{Written,Effective}) \ + \ Consolidation}{(3 * 512\,B \ + \ File_{Written})} \tag{4.24}$$

Figure 4.118: WAF Equation (Block-Mode FTL, Theoretical, File)

Using this equation, the values of WAF for file creation are predicted and compare to those previous measured (recall Table 4.22). These results are plotted in Figure 4.119. It is noted that the value of consolidation data has been increased to 580KB. The equation for WAF with this parameter is shown in Figure 4.120.

As Figure 4.119 shows, the trend of WAF measurements are well predicted by the WAF equation in Figure 4.120. While scaling of data with a constant value of Consolidation data was undesirable, the goal of having an equation to compute WAF has been achieved.

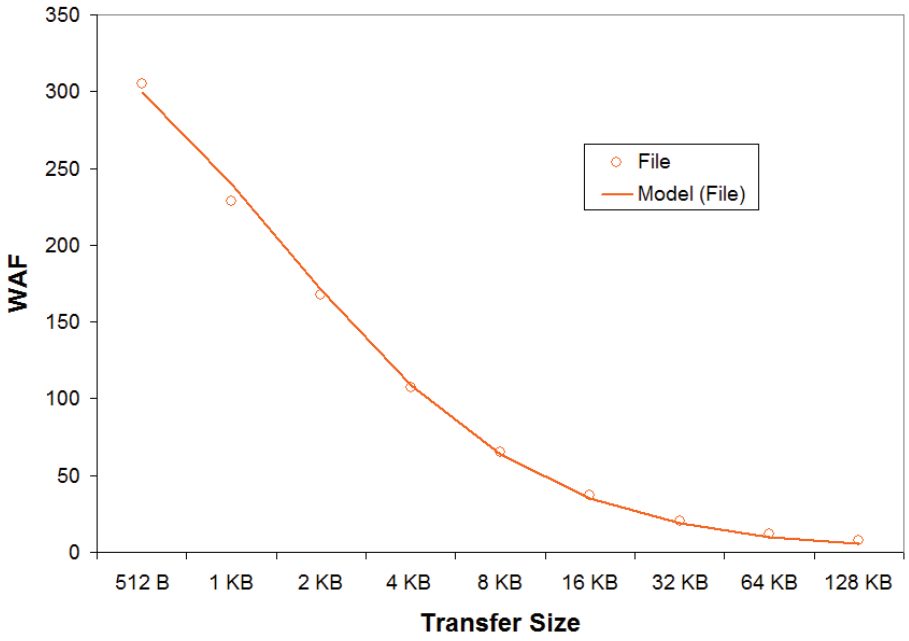

Figure 4.119: Block-Mode WAF Modeling (File))

$$WAF = \frac{(3 * 4\,KB\ +\ File_{Written,Effective})\ +\ 580\,KB}{(3 * 512\,B\ +\ File_{Written})} \tag{4.25}$$

Figure 4.120: WAF Equation (Block-Mode FTL, File)

4.18 Measurements (Page-Mode FTL, Repeated)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

Again recognizing that repeated transfers are a more accurate representation of file-transfer activities (recall Section 2.6), the flash memory system with the page-mode FTL is characterized using the previously developed technique for repeated writes. Specifically, the drive is subjected to an array of tests consisting of continuous repeated write operations using a fixed transfer size for each test. Repeated transfers of a single fixed location (R1), two fixed locations (R2), and three fixed locations (R3) were studied.

As in the previous sections, characterization consists of an array of individual test points with testing at each point conducted for one hour. Extended testing, when required for improved resolution, is conduced for eight hours. Overall, tests using 512 B, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, and 128KB transfer sizes have been conducted.

4.18.1 Repeated 1 (R1)

WAF measurements based on SMART data were made for the single fixed location (R1) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.24.

Transfer Size	WAF (Measured)
512 B	32.07
1 KB	16.03
2 KB	8.02
4 KB	4.01
8 KB	2.00
16 KB	1.00
32 KB	1.01
64 KB	1.06
128 KB	1.01

Table 4.24: WAF Measurements (Page-Mode FTL, R1)

Representative plots of performance measurements for repeated (R1) test points (512B to 128KB) are shown in Figure 4.121 to Figure 4.124, respectively

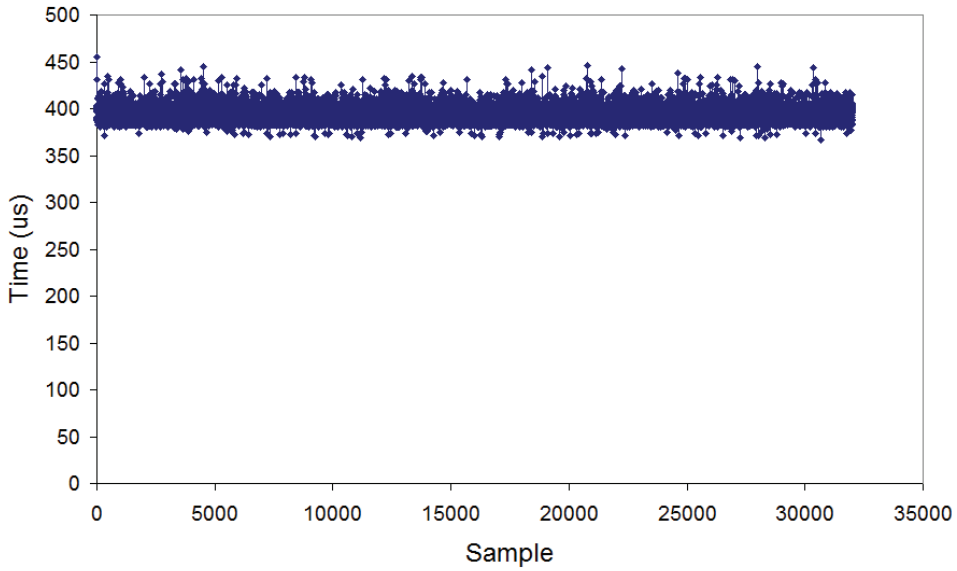

Figure 4.121: 512 B Performance Data (Page-Mode FTL, R1)

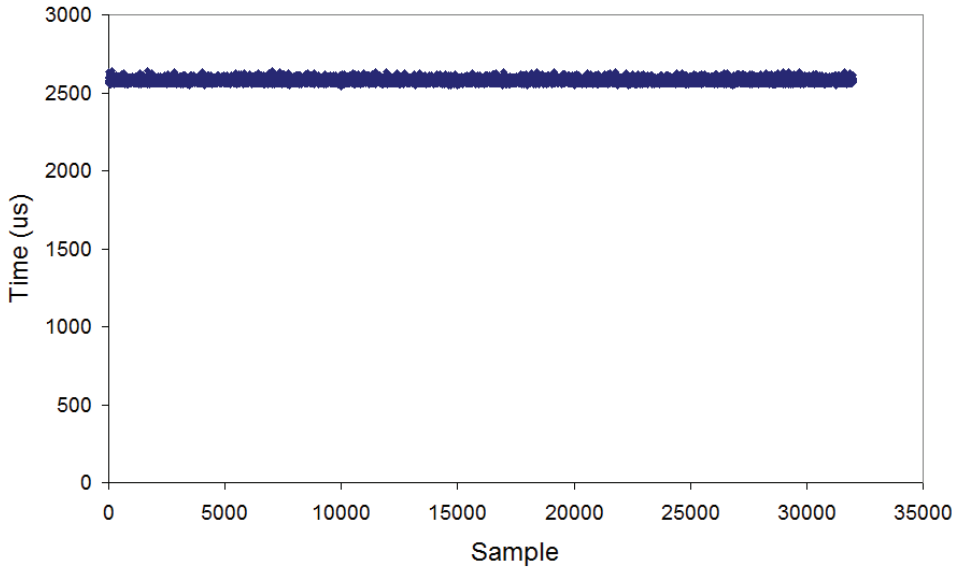

Figure 4.122: 4 KB Performance Data (Page-Mode FTL, R1)

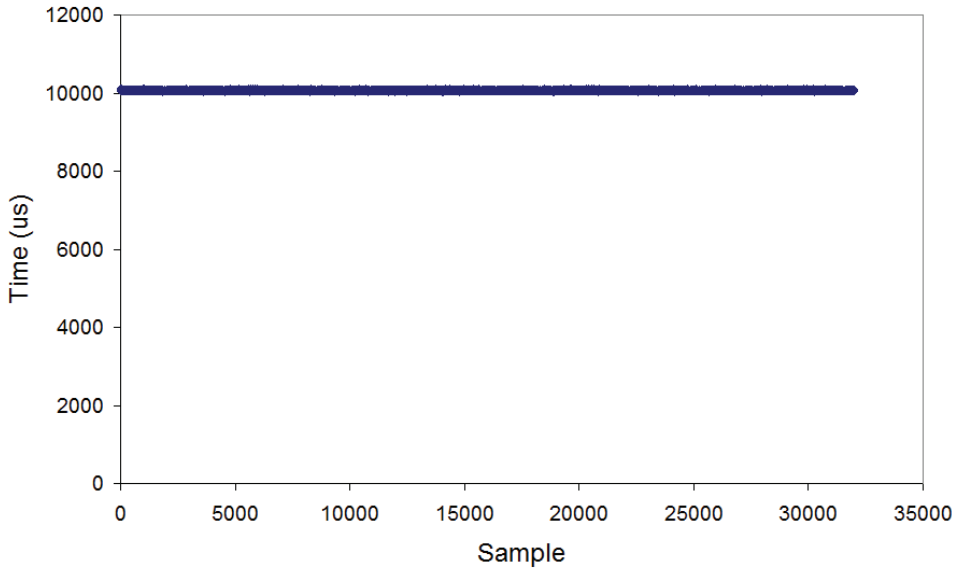

Figure 4.123: 16 KB Performance Data (Page-Mode FTL, R1)

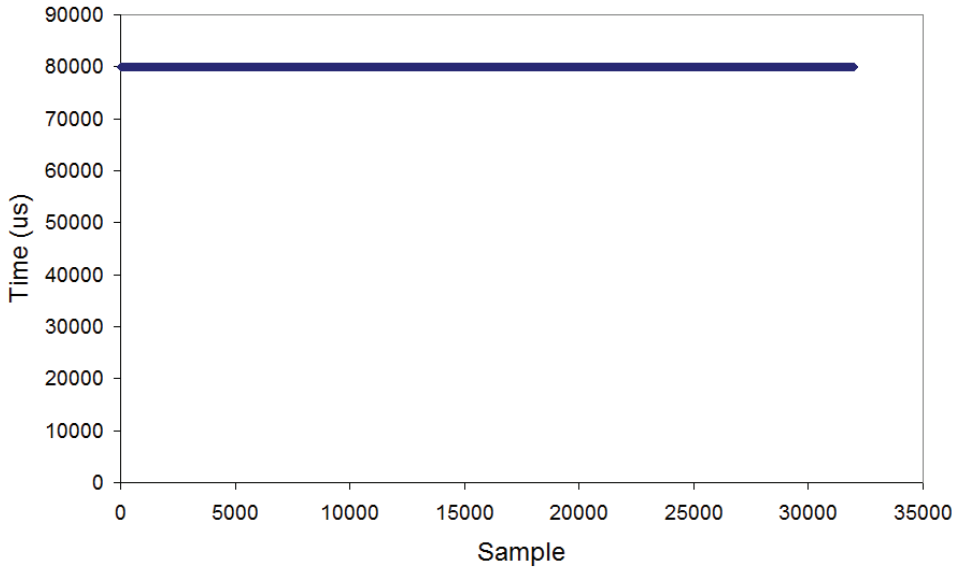

Figure 4.124: 128 KB Performance Data (Page-Mode FTL, R1)

As the plots illustrate, there appear to be no periodic components to the performance measurements. Transfers are being handled analogous to sequential requests. As observed with sequential write transfers, WAF approaches unity (1.0). Unlike the sequential transfers, a size of the allocation units is apparent (recall Table 4.9). Specifically, as a WAF of unity was observed for all sequential transfer sizes (recall Table 4.24), WAF for repeated writes only approaches unity.

Moreover, the ability to support sub-page allocation units (4KB) appears to be eliminated. For repeated writes, an allocation unit appears to be effectively 16KB.

4.18.2 Repeated 2 (R2)

WAF measurements based on SMART data were made for the two fixed location (R2) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.25.

Transfer Size	WAF (Measured)
512 B	16.03
1 KB	8.02
2 KB	4.01
4 KB	2.00
8 KB	1.00
16 KB	1.00
32 KB	1.00
64 KB	1.00
128 KB	1.00

Table 4.25: WAF Measurements (Page-Mode FTL, R2)

Representative plots of performance measurements for repeated (R2) test points (512B to 128KB) are shown in Figure 4.125 to Figure 4.128, respectively

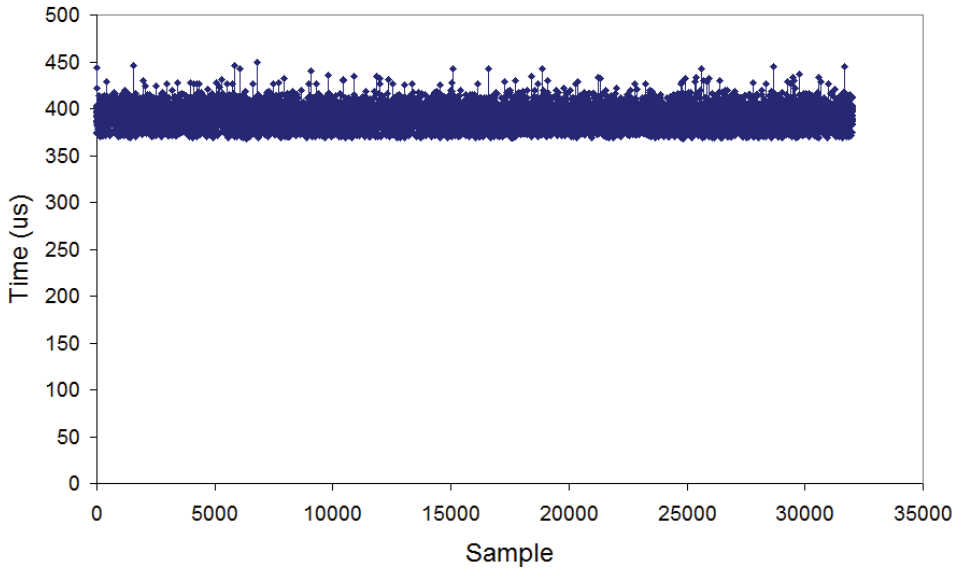

Figure 4.125: 512 B Performance Data (Page-Mode FTL, R2)

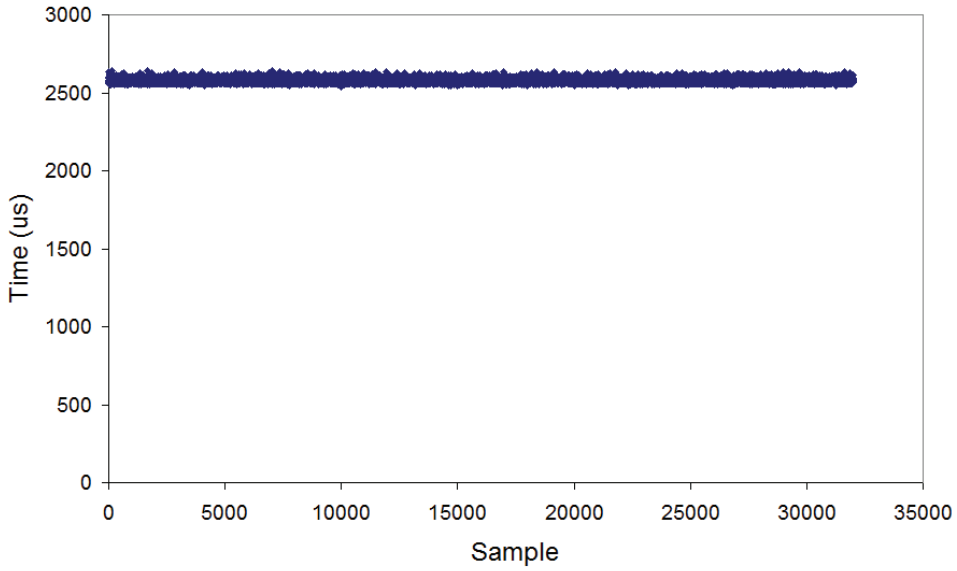

Figure 4.126: 4 KB Performance Data (Page-Mode FTL, R2)

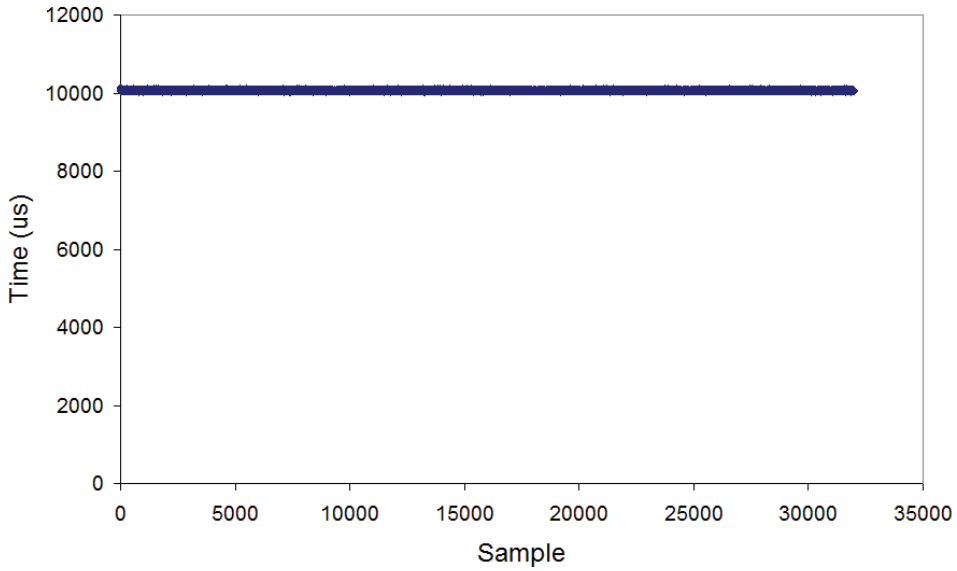

Figure 4.127: 16 KB Performance Data (Page-Mode FTL, R2)

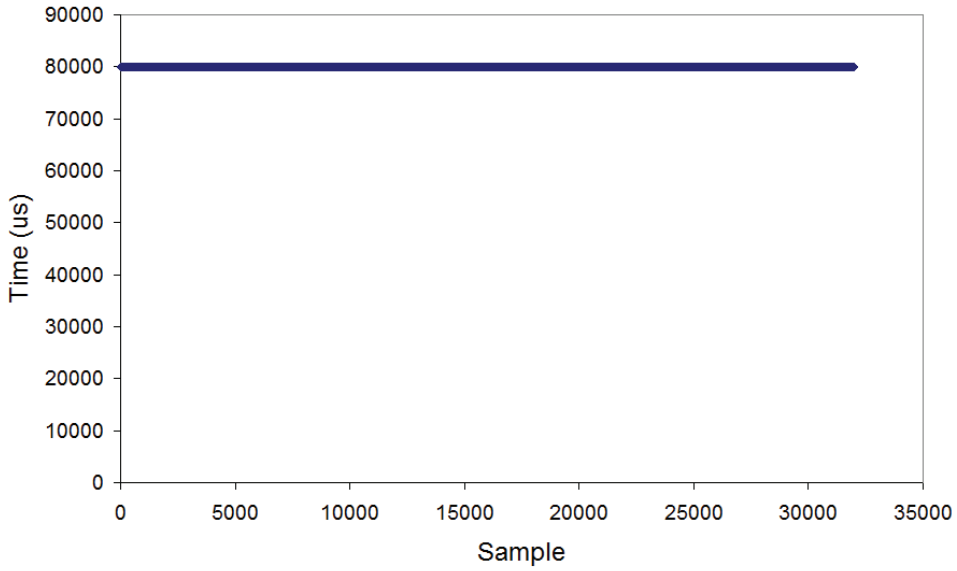

Figure 4.128: 128 KB Performance Data (Page-Mode FTL, R2)

As the plots illustrate, there are no periodic components to the performance measurements. Transfers are being handled analogous to sequential requests. As observed with sequential write transfers, WAF approaches unity (1.0). Unlike the sequential transfers, a size of the allocation units is apparent (recall Table 4.9). Specifically, as a WAF of unity was observed for all sequential transfer sizes (recall Table 4.25), WAF for repeated writes only approaches unity.

Moreover, the ability to support sub-page allocation units (4KB) appears to be eliminated. For repeated writes, an allocation unit appears to be effectively 8KB.

4.18.3 Repeated 3 (R3)

WAF measurements based on SMART data were made for the three fixed location (R3) test points. Extended tests (8 hours) were utilized for each data point. These measurements are presented in Table 4.24.

Transfer Size	WAF (Measured)
512 B	10.69
1 KB	5.344
2 KB	2.673
4 KB	1.337
8 KB	1.337
16 KB	1.003
32 KB	1.003
64 KB	1.003
128 KB	1.003

Table 4.26: WAF Measurements (Page-Mode FTL, R3)

Representative plots of performance measurements for repeated (R3) test points (512 B to 128 KB) are shown in Figures 4.129-4.132, respectively

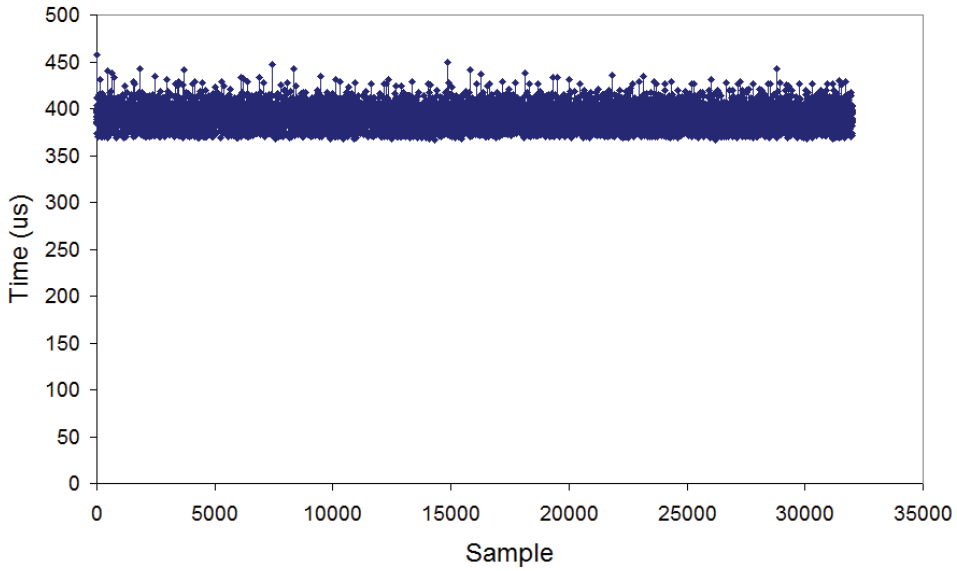

Figure 4.129: 512 B Performance Data (Page-Mode FTL, R3)

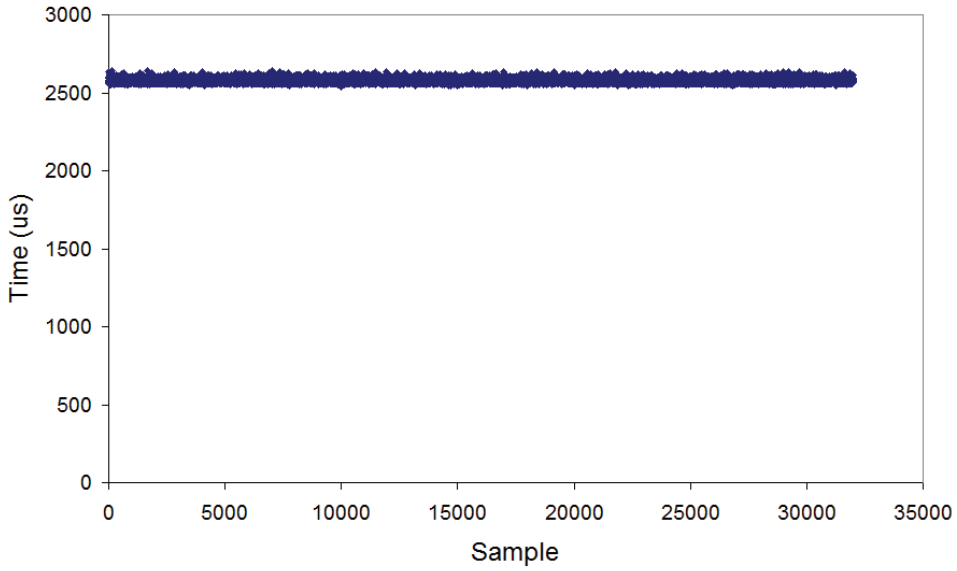

Figure 4.130: 4 KB Performance Data (Page-Mode FTL, R3)

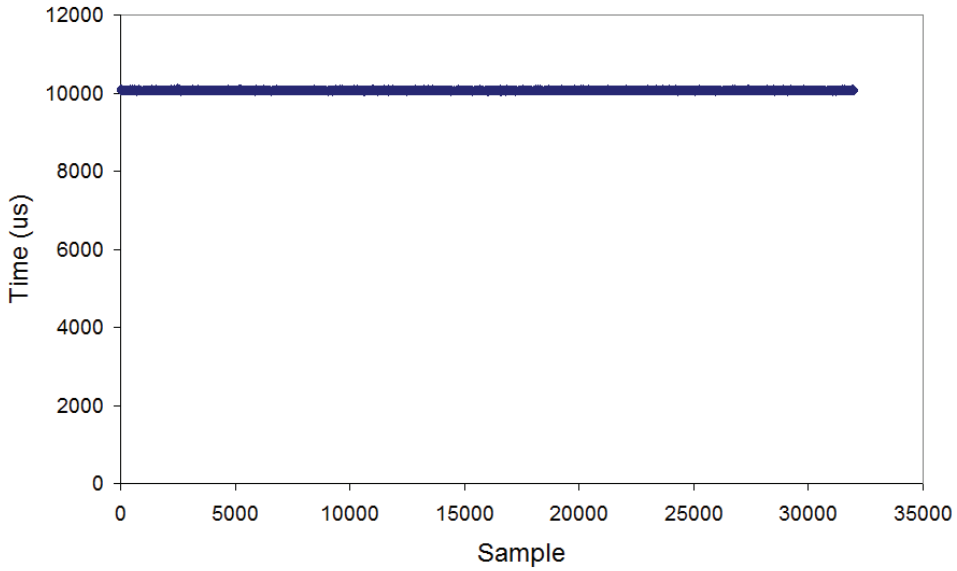

Figure 4.131: 16 KB Performance Data (Page-Mode FTL, R3)

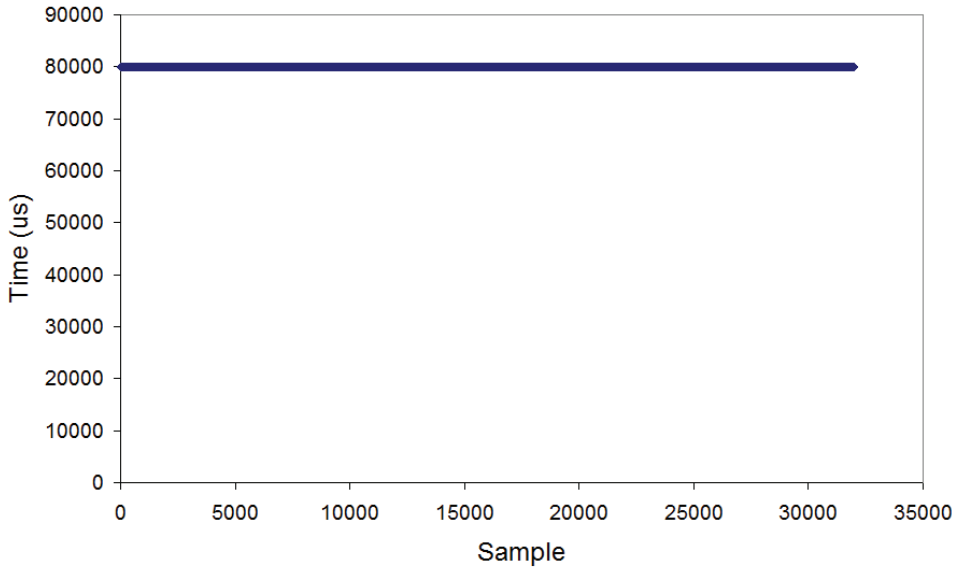

Figure 4.132: 128 KB Performance Data (Page-Mode FTL, R3)

As the plots again illustrate, there are no periodic components to the performance measurements. Transfers are being handled analogous to sequential requests. As observed with sequential write transfers, WAF approaches unity (1.0). Unlike the sequential transfers, a size of the allocation units is apparent (recall Table 4.9). Specifically, as a WAF of unity was observed for all sequential transfer sizes (recall Table 4.26), WAF for repeated writes only approaches unity.

Moreover, the ability to support sub-page allocation units (4KB) appears to be eliminated. For repeated writes, an allocation unit appears to be effectively 16KB.

4.19 Measurements (Page-Mode FTL, File)

Considering that file creation consists of three repeated 512B writes (recall Section 2.6) in addition to the file data, the flash memory system with the block-mode FTL is characterized for file creation using the previously developed technique augmented for compound operations. Specifically, the drive is subjected to an array of tests consisting of a continuous combination of three repeated write operations of 512B (to simulate FAT and directory meta-data) and a fixed transfer size sequential write request (to simulate file data) for each test.

As in the previous sections, characterization consists of an array of individual test points with testing at each point conducted for one hour. Extended testing, when required for improved resolution, is conduced for eight hours. Overall, tests using 512B, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, and 128KB file sizes have been conducted.

WAF measurements based on SMART data were made for the file test points. Extended tests (8 hours) were utilized for each data point. These measurements are

presented in Table 4.27.

Transfer Size	WAF (Measured)
512 B	15.47
1 KB	12.29
2 KB	8.70
4 KB	5.55
8 KB	3.45
16 KB	1.83
32 KB	1.44
64 KB	1.23
128 KB	1.12

Table 4.27: WAF Measurements (Page-Mode FTL, File)

Representative plots of performance measurements for file test points (512B to 128KB) are shown in Figures 4.133-4.136, respectively

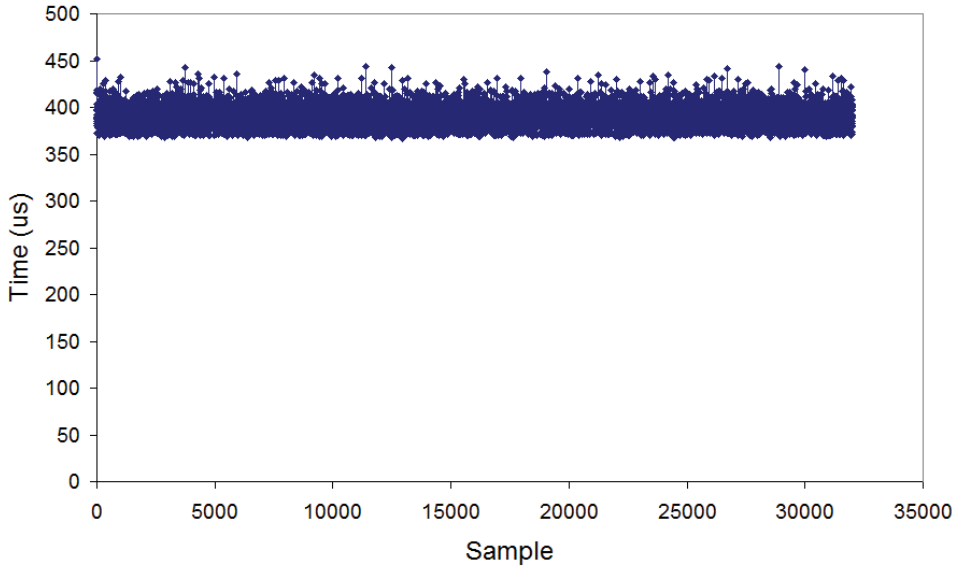

Figure 4.133: 512 B Performance Data (Page-Mode FTL, File)

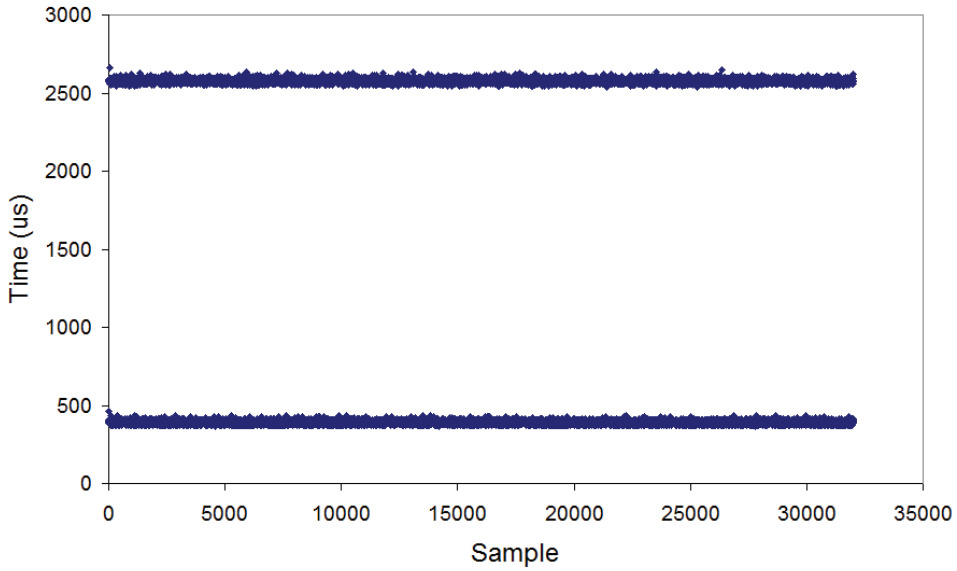

Figure 4.134: 4 KB Performance Data (Page-Mode FTL, File)

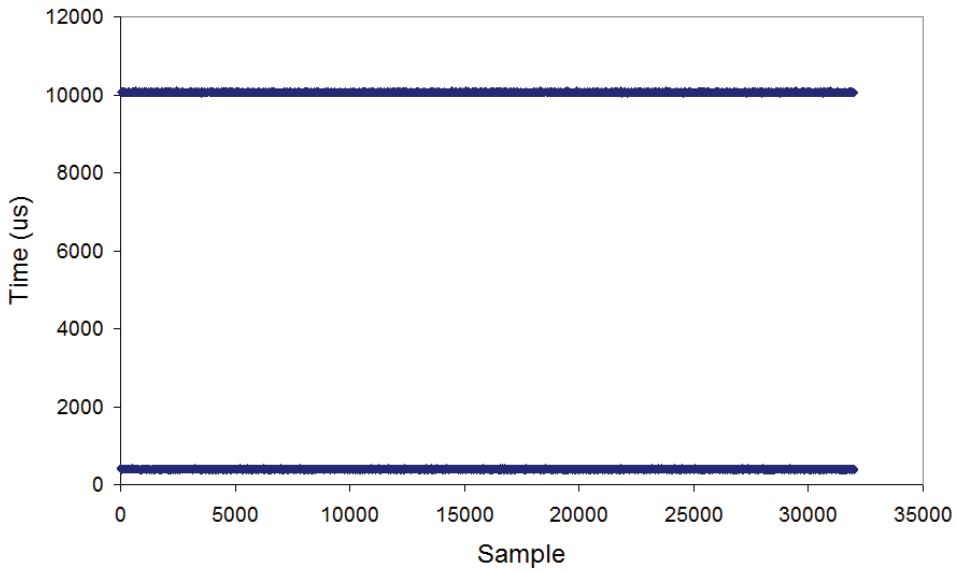

Figure 4.135: 16 KB Performance Data (Page-Mode FTL, File)

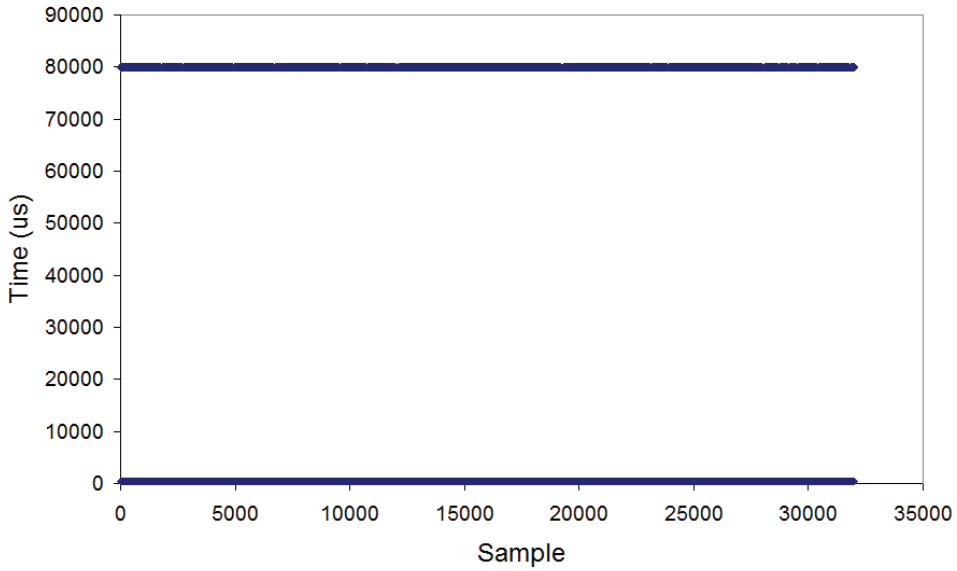

Figure 4.136: 128 KB Performance Data (Page-Mode FTL, File)

As the plots illustrate, there do not seem to be any periodic components in the observed measurements. Transfers are being handled analogous to sequential requests. As observed with sequential write transfers, WAF approaches unity (1.0). Unlike the sequential transfers, a size of the allocation units is apparent (recall Table 4.9). Specifically, as a WAF of unity was observed for all sequential transfer sizes (recall Table 4.27), WAF for repeated writes only approaches unity.

4.20 Empirical Model and WAF Equations (Page-Mode FTL, Repeated and File)

Activity
Measurement Technique
Single-Point Measurements (Block-Mode FTL)
Single-Point Modeling (Block-Mode FTL)
Single-Point Measurements (Page-Mode FTL)
Single-Point Modeling (Page-Mode FTL)
File Measurements (Block-Mode FTL)
File Modeling (Block-Mode FTL)
File Measurements (Page-Mode FTL)
File Modeling (Page-Mode FTL)

The WAF measurements for random write requests (Section 4.11), repeated write requests (Section 4.18), and file write requests (Section 4.19) for the drive with the page-mode FTL are collectively summarized below in Figure 4.137.

As Figure 4.137 indicates, the repeated and file WAF measurements are lower than those observed for random writes. The negative coupling effect that was previously observed with the block-mode FTL as it transitions between support for sequential and random write requests (recall Section 4.7) is not present. Alternatively stated, the page-mode FTL is more inherently suited towards repeated writes than the block-mode FTL.

Recalling that WAF is the collective result of effective copying data (host written), copying data (garbage collections), data copying from consolidation, and copying of

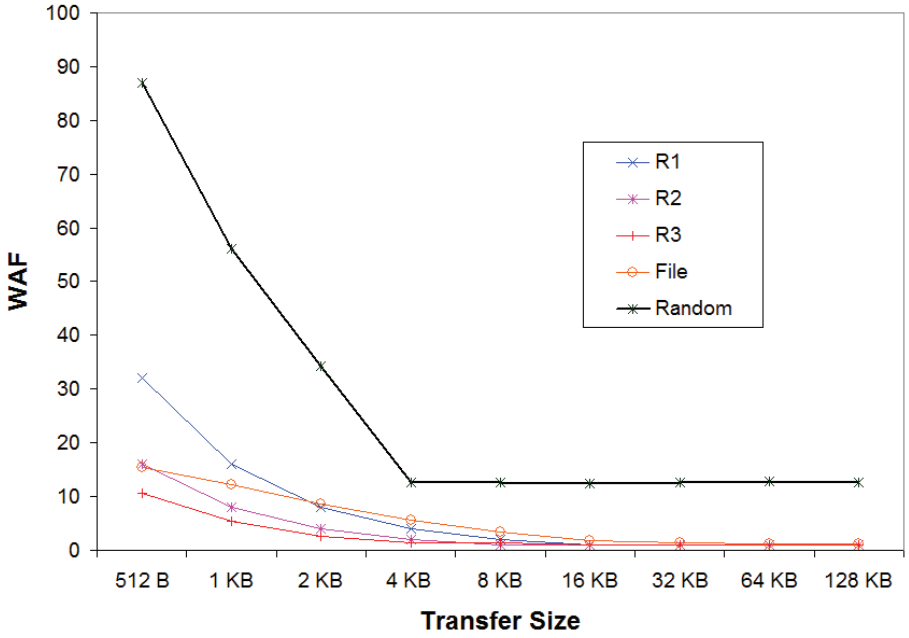

Figure 4.137: Page-Mode WAF Summary (Page-Mode FTL, Repeated and File)

FTL overhead scaled by the host written data. The equation of WAF is recalled (Figure 4.114) and again shown in Figure 4.138

$$WAF = \frac{Data_{Written,Effective} + Data_{GC} + Consolidation + FTL\,Overhead}{Data_{Written}} \tag{4.26}$$

Figure 4.138: WAF Equation (Theoretical)

The lack of periodic components identified with FTLProbe suggests that minimal copy operations are occurring. In particular, the lack of F1 (associated with block copying), in conjunction with the observation that the page-mode FTL does not exhibit negative coupling, suggests that neither consolidation nor data copying due to garbage collection are relevant and the WAF equation reduces as shown in Figure 4.139

$$WAF = \frac{Data_{Written,Effective} + FTL\,Overhead}{Data_{Written}} \tag{4.27}$$

Figure 4.139: WAF Equation (Page-Mode FTL, Theoretical)

We again recall that the page-mode FTL handles block erases (recall Section 4.14) in parallel with incoming write requests. Therefore, block erases they are not specifically visible to the FTLProbe technique. However, as block erases are absolutely required, they cannot be disregarded.

Knowing that flash is allocated in 4KB sub-page allocation units, the WAF due to 4KB (and larger) repeated writes would be expected to be on the order of unity (1.0). Specifically, a block is expected to receive repeated requests and stream them into an active block much like the sequential transfers (recall Figure 2.11). When the block is full, a new block can receive the repeated write requests. At this time, the previously active block is filled entirely with invalid pages. The entire block can therefore be erased without any copying operations. This concept is shown in Figure 4.140.

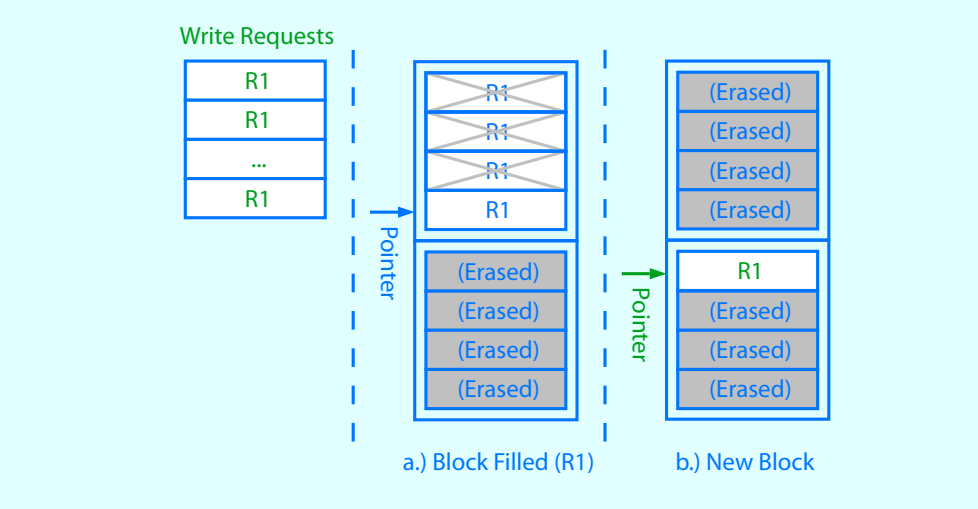

Figure 4.140: Ideal Repeated Writes (Page-Mode FTL)

However, as the WAF for R1, R2, and R3 repeated writes is measured to be 4.008, 2.005, and 1.337 respectively, and not unity, it is concluded that the concepts shown in Figure 4.140 are not sufficient to explain the measured WAF.

As a lack of F1 features suggests that no significant block copy operations occur, we speculate that the higher WAF is due to FTL overhead that is written with the repeated writes. As this data becomes unnecessary with subsequent repeated writes, the block may also erased when the blocks are filled (see Figure 4.141).

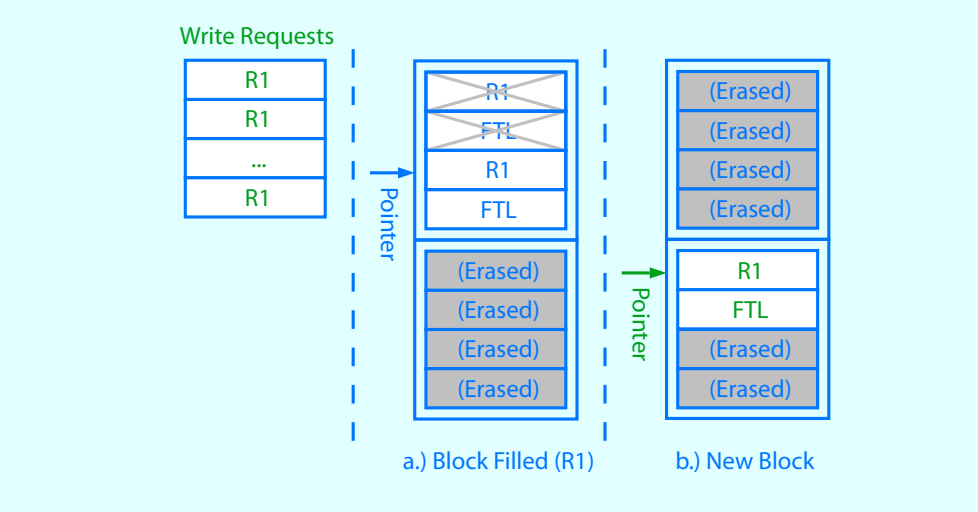

Figure 4.141: Repeated Writes (Page-Mode FTL, R1)

Analysis of the repeated WAF measurements indicates that the FTL Overhead is 12KB, 8KB and 4KB for the R1, R2 and R3 write requests, respectively. However, the relationship between these three values, specifically, 1, 1/2 and 1/3 suggests that the FTL Overhead is only being committed when a repeated write occurs. Alternatively stated, it appears that the page-mode FTL pointer list is limited such that it cannot contain a repeated sector address entry. When a repeated write is encountered, the FTL must commit the 12KB data structure. This concept is illustrated in Figure 4.142 for the three repeated writes.

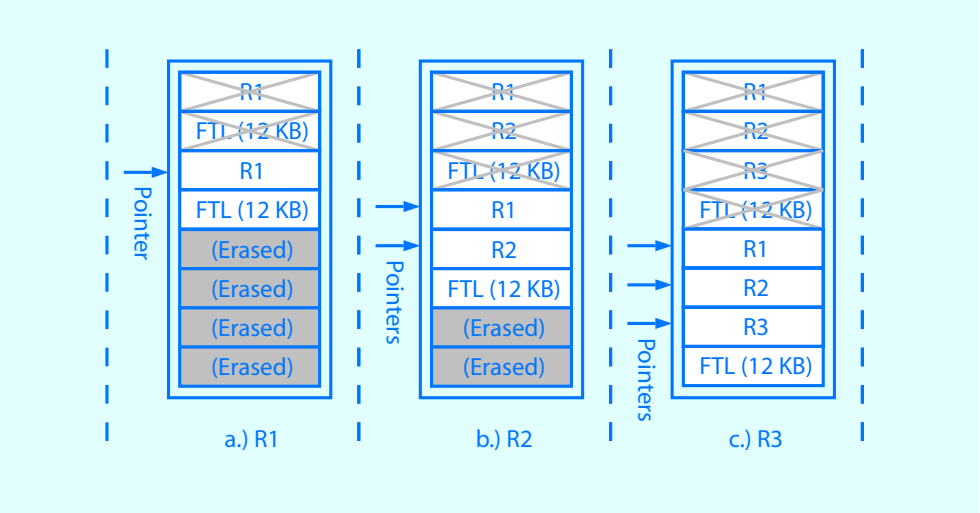

Figure 4.142: Repeated Writes (Page-Mode FTL, R1/R2/R3)

With the knowledge that the FTL Overhead is a fixed value (12KB) that is flushed when a repeated write occurs, the WAF equation can be simplified as shown in Figure 4.143.

$$WAF = \frac{Data_{Written,Effective} + \dfrac{12KB}{Writes\ Until\ Repeat}}{Data_{Written}} \tag{4.28}$$

Figure 4.143: WAF Equation (Page-Mode FTL, Repeated)

Using the equation shown in Figure 4.143, the WAF values are predicted for the three repeated write operations. These results are shown with the measured values of WAF in Figure 4.144. As this figure suggests, results of the WAF equation are well aligned with measured values.

Adapting the previous equation to compound write operations, such as occur with a file create operations, the equation for WAF changes, as shown in Figure 4.145.

For a FAT file system which has three 512B meta-data writes per block transfer,

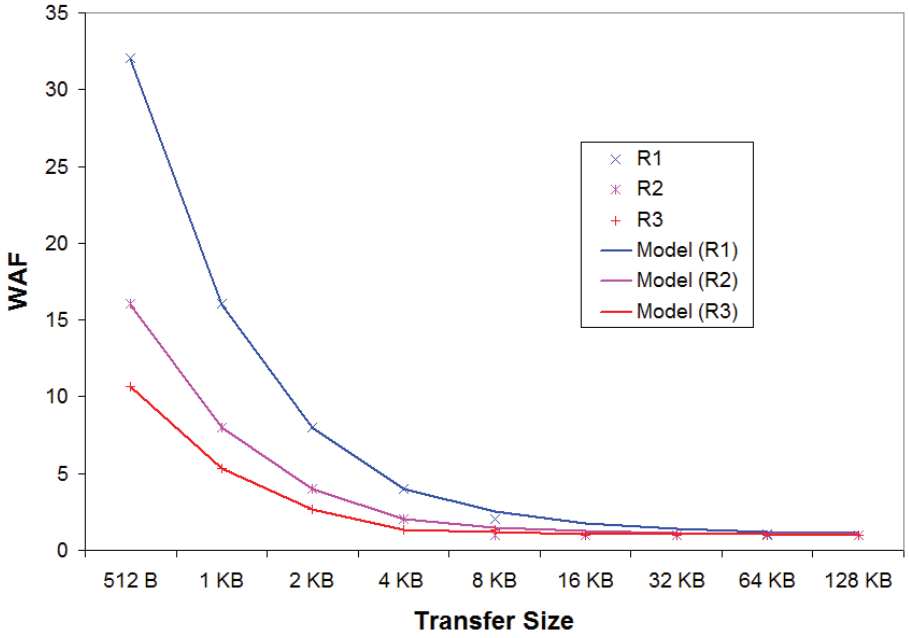

Figure 4.144: WAF Modeling (Page-Mode FTL, Repeated))

$$WAF_{Compound} = \frac{\sum(Data_{Written,Effective}) + \dfrac{12KB}{Writes\,Until\,Repeat}}{\sum(Data_{Written})} \tag{4.29}$$

Figure 4.145: Compound WAF Equation (Page-Mode FTL, Theoretical)

which means that four writes occur for each repeated write trigger, this equation can be rewritten as shown in Figure 4.146.

Using this equation, the values of WAF for file creation are predicted and compared to those previous measured (recall Table 4.27). These results are plotted in Figure 4.147.

As Figure 4.147 indicates, both the trend and magnitude of WAF measurements are in agreement with our WAF equation. An equation predicting WAF has been developed for the page-mode FTL.

$$WAF_{Compound} = \frac{(3 * 4\,KB\ +\ File_{Written,Effective})\ +\ \dfrac{12KB}{WritesUntilRepeat}}{(3 * 512\,B\ +\ File_{Written})} \quad (4.30)$$

Figure 4.146: Compound WAF Equation for FAT File Creation (Page-Mode FTL)

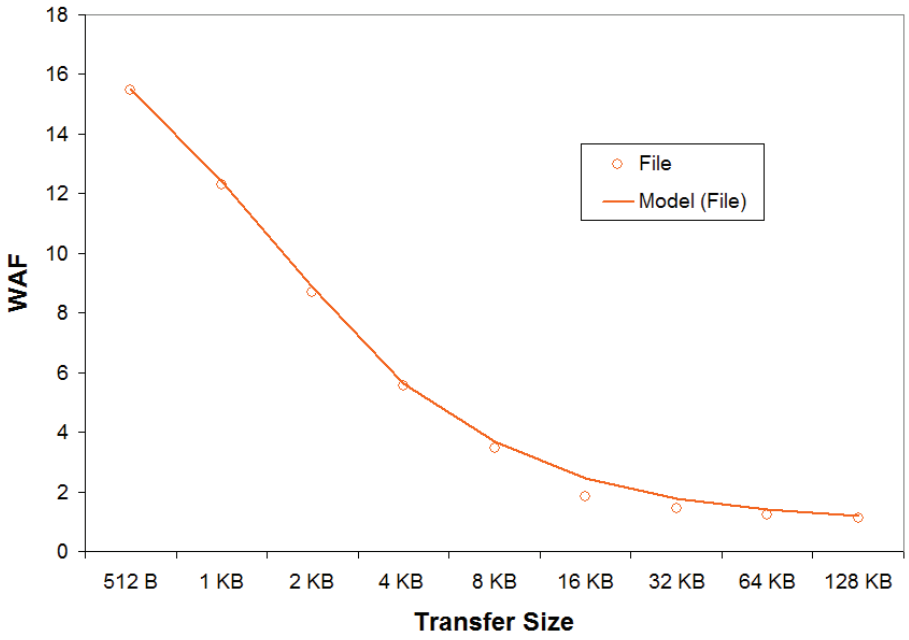

Figure 4.147: Page-Mode WAF Modeling (File)

Chapter 5

FSAware: A File System Aware FTL Extension

5.1 FSAware (Block-Mode FTL)

FSAware is an enhancement to FTL designs that specifically intends to lower WAF by supporting the data and the meta-data of file system operations with separate handlers (recall Section 1.1). In general, the effectiveness of FSAware is demonstrated by adapting the equations of WAF for file system operations previously developed.

Additionally, the improvements afforded by FSAware are demonstrated by simulating the effects of segregating file system data and meta-data at the host and measuring WAF using the previously presented FTLProbe technique. Specifically, as the drives using the block-mode FTL and page-mode FTL are both characterized as using 64MB super blocks, the FTLProbe application software issues 64MB writes of each meta-data type (FAT1, FAT2, and root directory) and then issues writes for an appropriate amount of file data.

Considering each meta-data write is 512B, there are 131,072 meta-data writes associated with a 64MB superblock. The total data associated with 131,072 is the product of this value and the file size. For 4KB data files, the total data is 512MB.

Recalling the equation of WAF in a block-mode FTL for file system operations developed in Section 4.17, this equation is adapted for the number of operations, N, as shown below.

$$WAF = \frac{(3 * 4\,KB\ *N\ +\ File_{Written,Effective}\ *\ N)\ +\ Consolidation}{(3 * 512\,B\ *\ N\ +\ File_{Written}\ *\ N)} \tag{5.1}$$

Figure 5.1: WAF Equation (Block-Mode FTL, Theoretical, File, N Operations)

As noted in Section 4.17 during development of the WAF equation, the Consolidation factor was experimentally determined. This approach is again adopted and WAF measurements for FSAware are presented below in Table 5.1.

Transfer Size	WAF (Measured)
512 B	6.334
1 KB	5.264
2 KB	4.047
4 KB	2.983
8 KB	2.162
16 KB	1.624
32 KB	1.329
64 KB	1.170
128 KB	1.092

Table 5.1: WAF Measurements (Block-Mode FTL, FSAware)

Recalling that WAF for file system operations were on the order of 300 for 512B transfers (Table 4.22), it is clear that FSAware offers a dramatic reduction in WAF.

Using the empirical method determining the value of Consolidation for the FSAware WAF measurements, it is found that this term is zero for file system operations with FSAware. This allows the equation of WAF to be rewritten as shown below in Figure 5.2.

$$WAF = \frac{(3 * 4\,KB\ *N\ +\ File_{Written,Effective}\ *\ N)}{(3 * 512\,B\ *\ N\ +\ File_{Written}\ *\ N)} \tag{5.2}$$

Figure 5.2: WAF Equation (Block-Mode FTL, FSAware, N Operations)

As N is a common term, the equation can be further reduced as shown below in Figure 5.3.

$$WAF = \frac{(3 * 4\,KB\ +\ File_{Written,Effective})}{(3 * 512\,B\ +\ File_{Written})} \tag{5.3}$$

Figure 5.3: WAF Equation (Block-Mode FTL, FSAware)

The WAF predictions of this equation are plotted with the FSAware WAF measurements in Figure 5.4.

As Figure 5.4 suggests, there is reasonable agreement between the values predicted by the WAF equation and the WAF measurements. Some departure is noted for transfers below the flash allocation unit size (4KB). This disagreement can be rectified if the flash allocation size in the WAF equation is reduced from 4KB to 1KB. It is expected that this adjustment is required because of optimizations performed by the FTL for the small transfers.

However, it is known that flash cannot be written in 1KB allocation units. Also, it is again emphasized that the intent of this work is to develop empirical models that align empirical measurements and analytical expectations and not black-box models

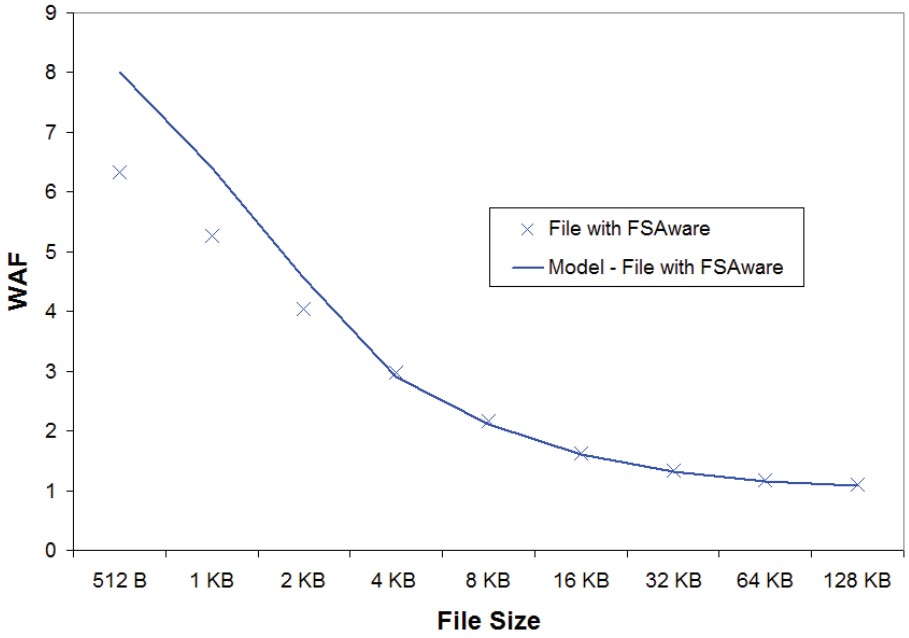

Figure 5.4: WAF Modeling (Block-Mode FTL, FSAware)

that are based solely on curve fitting of measured data. For these considerations, the equation is left as shown above.

Regardless of this effect, it is noted that the WAF for FSAware is a considerable improvement of WAF associated with file system operations supported with the block-mode FTL. For illustration, WAF measurements, along with modeling results, of the file system operations are shown in Figure 5.5.

As Figure 5.5 shows, WAF is considerably reduced with FSAware. Considering 4KB transfers, WAF is reduced from 107.5 to 2.909, a reduction of 97%. Moreover, WAF for larger file transfers approaches unity (1.0).

We should recall that the block-mode FTL is impacted by the effect of consolidation. Specifically, the thrash observed in this study and reported by Tijoe[107] for the block-mode FTL as it addresses file system meta-data "randomly", clearly degrades

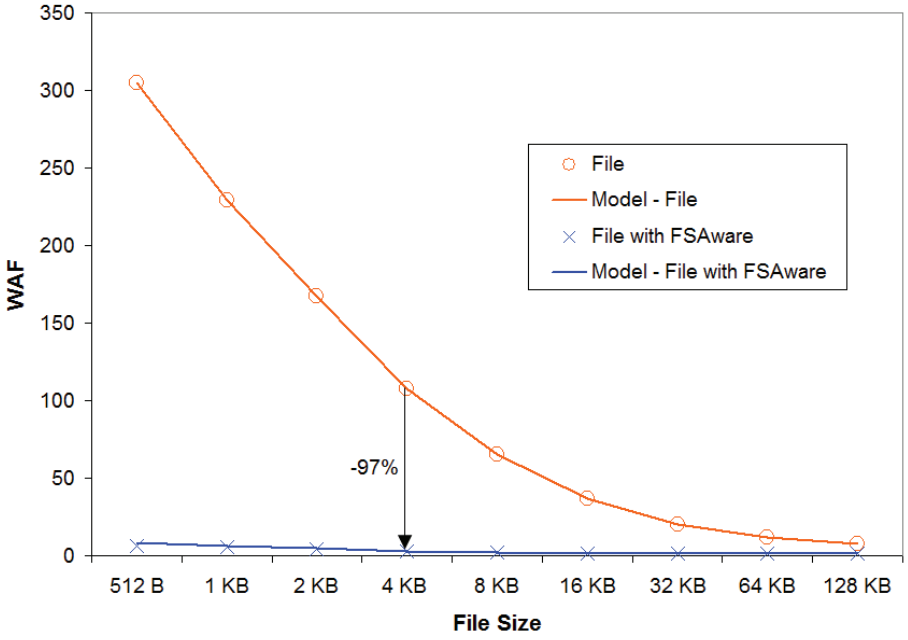

Figure 5.5: WAF (Block-Mode FTL, File System and FSAware)

WAF. FSAware offers block-mode FTLs a two-fold improvement in WAF by considering file systems as repeated operations that are more suitable to flash memory and inherently more suitable to block-mode FTLs as evidenced by the inherent elimination of the consolidation process. With these gains, FSAware is able to support file system operation with a WAF that approaches the optimum value of unity (1.0), previously experienced in flash memory systems only for sequential write request.

5.1.1 FSAware Enhanced

Knowing that several types of meta-data, such as FAT1, FAT2, and directory are usually written together, and knowing that flash memory can only be written in 4KB units, FSAware can be further enhanced to include the possibility that a single flash allocation unit can be used to support all meta-data writes. Specifically, instead of

using three 4KB flash allocation units to support three 512B meta-data write requests, a single 4KB flash allocation is used for the three meta-data writes.

It is noted that this design is a more substantial modification of the FTL in that meta-data regions need to be handled with partial flash allocation unit granularity. Granularity of a single sector is specifically suggested. As this change cannot be readily simulated, WAF measurements of FSAware Enhanced cannot be conducted.

With the FSAware Enhanced design, the equation of WAF reduces to the equation shown below in Figure 5.6.

$$WAF = \frac{(4\,KB\ +\ File_{Written,Effective})}{(3 * 512\,B\ +\ File_{Written})} \tag{5.4}$$

Figure 5.6: WAF Equation (Block-Mode FTL, FSAware Enhanced)

The results of the WAF equation for FSAware Enhanced are shown with the measurements and predictions of the FSAware in Figure 5.7.

As Figure 5.7 suggests, a further reduction of WAF is noted. For 4KB transfers, WAF is reduced from 2.909 to 1.454. This is an additional reduction of 50%. Moreover, WAF approaches unity (1.0) even more rapidly as transfer size increases.

WAF for file system operations on a flash memory system using a block-mode FTL with FSAware and FSAware Enhanced are summarized below in Table 5.2 and plotted below in Figure 5.8. As shown, WAF is significantly reduced. Moreover, WAF approaches the theoretical ideal value of unity (1.0); particularly for larger file sizes.

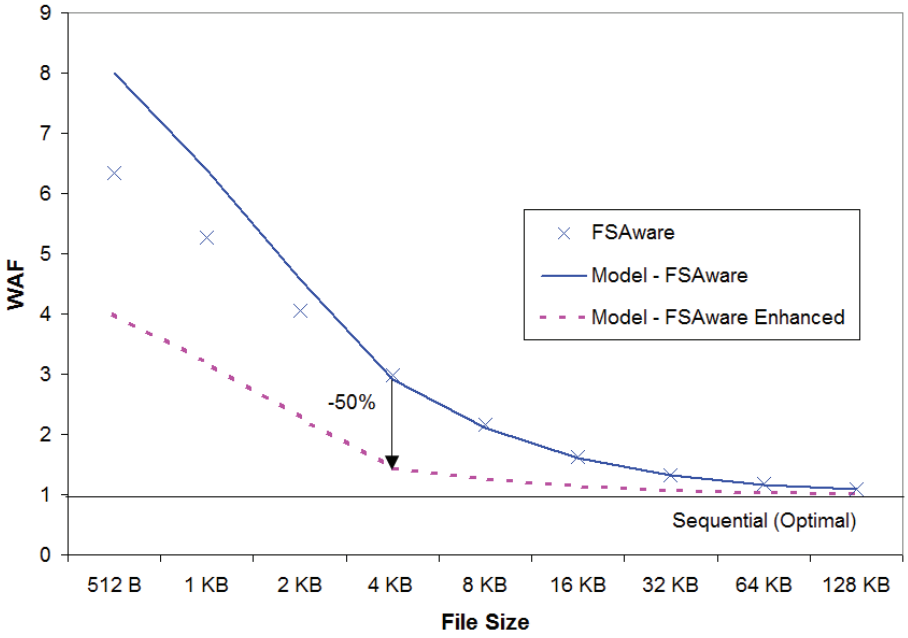

Figure 5.7: WAF Modeling (Block-Mode FTL, FSAware Enhanced)

Transfer Size	Current	FSAware	FSAware Enhanced
512 B	305.1	8.00 (-97%)	4.00 (-99%)
4 KB	107.5	2.91 (-97%)	1.45 (-99%)
32 KB	20.47	1.31 (-94%)	1.07 (-95%)

Table 5.2: WAF Summary (Block-Mode FTL, FSAware and FSAware Enhanced)

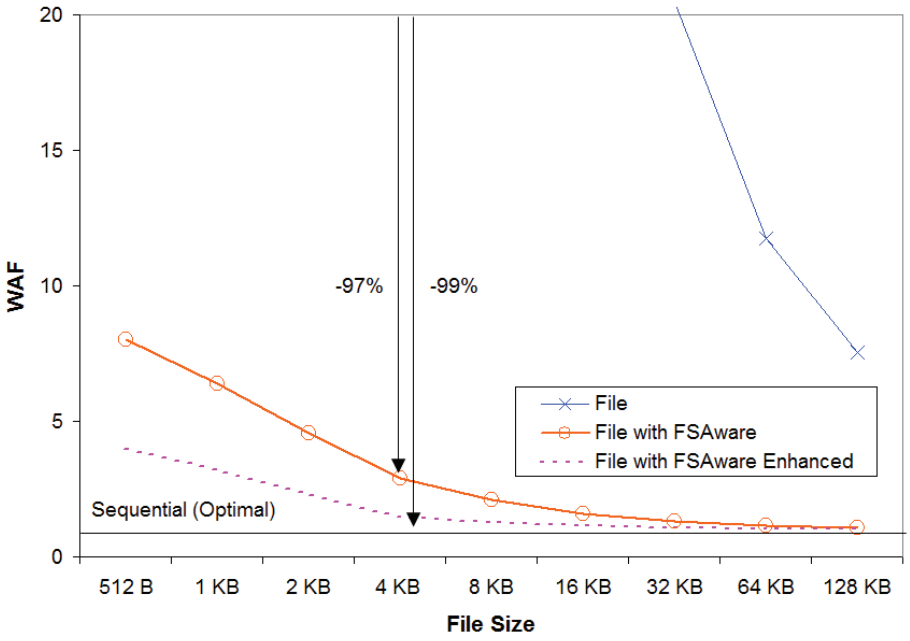

Figure 5.8: WAF Summary (Block-Mode FTL, FSAware and FSAware Enhanced)

5.2 FSAware (Page-Mode FTL)

FSAware is demonstrated for the Page-Mode FTL using the same approach we used for the Block-Mode FTL. Specifically, we demonstrate the effectiveness of FSAware by adapting the equations of WAF for file system operations, and considering the improvements afforded by FSAware, we we simulate the effects of segregating file system data and meta-data at the host and measuring WAF using the FTLProbe technique.

Recalling the equation of WAF in a page-mode FTL for file system operations developed in Section 4.20, this equation is adapted for the number of operations, N, as shown below in Figure 5.9.

$$WAF_{Compound} = \frac{(3 * 4\,KB\ *\ N\ +\ File_{Written,Effective}\ *\ N)\ +\ \dfrac{FTLOverhead}{4}\ *\ N}{(3 * 512\,B\ *\ N\ +\ File_{Written}\ *\ N)} \tag{5.5}$$

Figure 5.9: WAF Equation (Page-Mode FTL, FSAware, N Operations)

As N is a common term, the equation immediately reduces to the form shown in Figure 5.10.

$$WAF_{Compound} = \frac{(3 * 4\,KB\ +\ File_{Written,Effective})\ +\ \dfrac{FTLOverhead}{4}}{(3 * 512\,B\ +\ File_{Written})} \tag{5.6}$$

Figure 5.10: WAF Equation (Page-Mode FTL, FSAware)

WAF measurements for FSAware simulation are presented below in Table 5.3. For comparison, the WAF predictions of the equation shown in Figure 5.10 are plotted with the FSAware WAF measurements in Figure 5.11.

Transfer Size	WAF (Measured)
512 B	9.429
1 KB	7.314
2 KB	5.107
4 KB	3.499
8 KB	2.084
16 KB	1.049
32 KB	0.960
64 KB	0.982
128 KB	0.992

Table 5.3: WAF Measurements (Page-Mode FTL, FSAware)

As Figure 5.11 suggests, there is reasonable agreement between the values predicted by our WAF equation and the WAF measurements. Some differences are noted for transfers around 16KB. While some adjustment is possible to compensate for this behavior, it is again emphasized that the intent of this work is to develop empirical (gray box) models that align empirical measurements and analytical expectations and, and to not use blackbox models that are based solely on curve fitting of measured data. For these considerations, the WAF equation remains unchanged.

Regardless of this effect, it is noted that the WAF for FSAware is an improvement of WAF associated with file system operations supported with the Page-Mode FTL. For comparison, WAF measurements, along with modeling results, of these two operations are shown in Figure 5.12.

As Figure 5.12 shows, WAF is reduced with FSAware. Considering 4KB transfers, WAF is reduced from 5.818 to 3.455, a reduction of 41%. WAF for larger file transfers continues to approach unity (1.0).

As with the Block-Mode FTL, FSAware offers Page-Mode FTLs improvement in WAF by considering file systems as repeated operations that are more suitable to

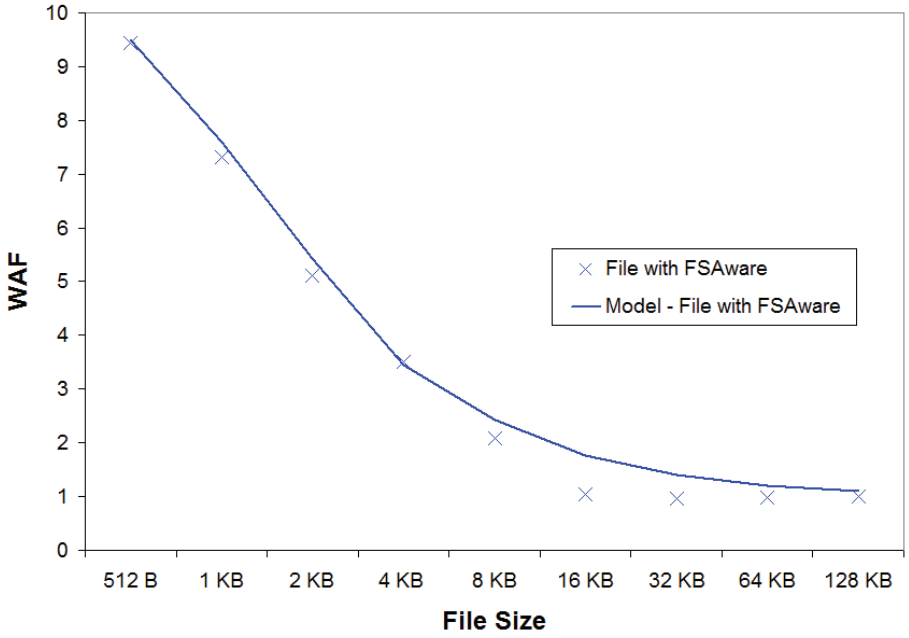

Figure 5.11: WAF Modeling (Page-Mode FTL, FSAware)

flash memory. With these gains, FSAware is able to support file system operation with a WAF that approaches the optimum value of unity (1.0), previously experienced in flash memory systems only for sequential write request.

5.2.1 FSAware Enhanced

Knowing that several types of meta-data, such as FAT1, FAT2, and directory, are usually written together, and given that flash memory can only be written in 4KB units, FSAware can be further enhanced to include the possibility that a single flash allocation unit can be used to support all meta-data writes. Specifically, instead of using three 4KB flash allocation units to support three 512B meta-data write requests, a single 4KB flash allocation is used for the three meta-data writes.

It is again noted that this design is a more substantial modification of the FTL in

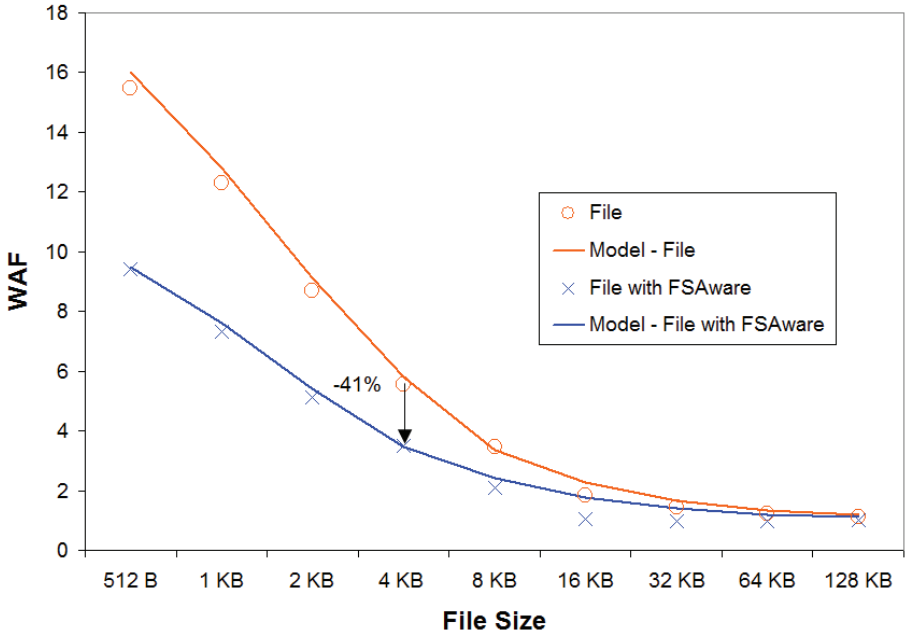

Figure 5.12: WAF (Page-Mode FTL, File System and FSAware)

that meta-data regions need to be handled with partial flash allocation unit granularity. Granularity of a single sector is specifically suggested. As this change cannot be readily simulated, WAF measurements of FSAware Enhanced cannot be conducted.

With the FSAware Enhanced design, the equation of WAF reduces to the equation shown below in Figure 5.13.

$$WAF_{Compound} = \frac{(4\,KB\ +\ File_{Written,Effective})\ +\ \dfrac{FTLOverhead}{4}}{(3 * 512\,B\ +\ File_{Written})} \tag{5.7}$$

Figure 5.13: WAF Equation (Page-Mode FTL, FSAware Enhanced)

The results of the WAF equation for FSAware Enhanced are shown with the measurements and predictions of the FSAware in Figure 5.14.

As Figure 5.14 suggests, a further reduction of WAF is noted. For 4KB transfers,

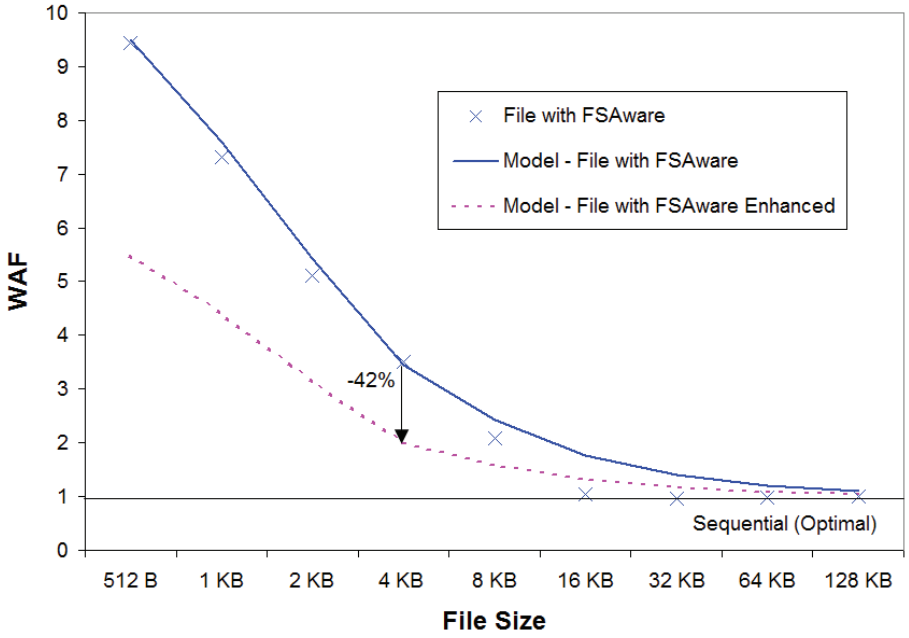

Figure 5.14: WAF Modeling (Page-Mode FTL, FSAware Enhanced)

WAF is further reduced from 3.455 to 2.000. This is an additional reduction of 42%. Moreover, WAF approaches unity (1.0) even more rapidly as transfer size increases.

WAF for file system operations on a flash memory system using a page-mode FTL with FSAware and FSAware Enhanced are summarized below in Table 5.4 and plotted below in Figure 5.15. As shown, WAF is significantly reduced. Moreover, WAF approaches the theoretical ideal value of unity (1.0); particularly for larger file sizes.

Transfer Size	Current	FSAware	FSAware Enhanced
512 B	15.46	9.50 (-39%)	5.50 (-65%)
4 KB	5.553	3.45 (-38%)	2.00 (-64%)
32 KB	1.438	1.40 (-2.4%)	1.16 (-19%)

Table 5.4: WAF Summary (Page-Mode FTL, FSAware and FSAware Enhanced)

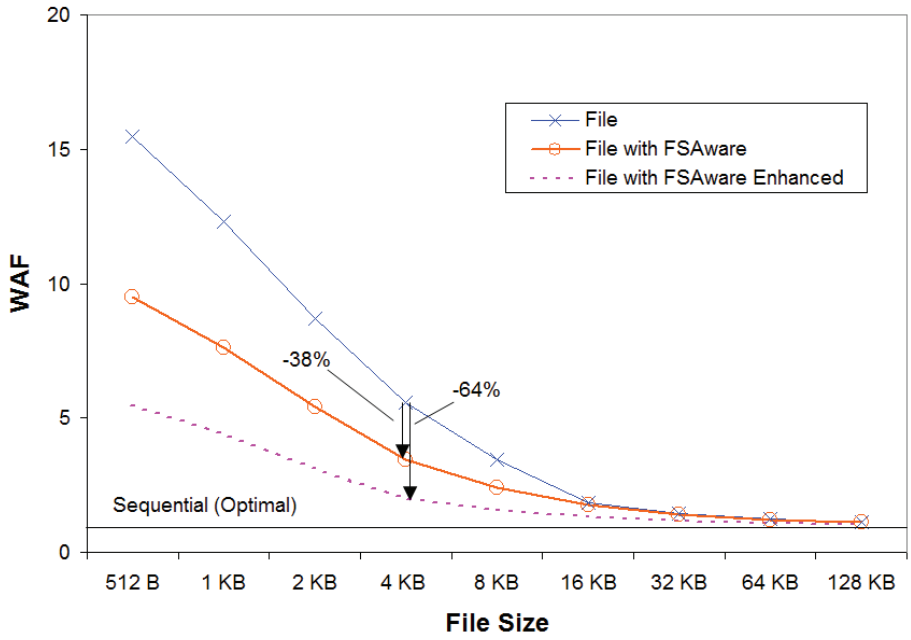

Figure 5.15: WAF Summary (Page-Mode FTL, FSAware and FSAware Enhanced)

Chapter 6

Directions for Future Work

6.1 Host-Flash WAF

Flash memory systems used in a host computer system involve two levels of indirection: the file system and the FTL. These two levels of indirection introduce two levels of inefficiency. Beyond their individual inefficiencies, these two systems of indirection can negatively impact each other, creating a poorer performing system.

Generally speaking, improvements to these two levels of indirection have focused on reducing or eliminating the inefficiencies and cross-interference, making the host file system more compatible with the FTL. At a minimum, this involves a file system that is FTL aware. A more effective is to make the file system fully flash-aware and, in so doing, reduce the work of the FTL and remove a level of inefficiency. Alternatively, the approach of presented in this thesis, which is believed to be unique, is to have an FTL which is file system aware and, in so doing, also remove a level of inefficiency within the flash memory system.

Regardless of the approach to improving efficiency, a metric for true effectiveness

is required. This metric should span the solution technique whether the efficiency improvement occurs on the host side with the file system, the flash memory system with the FTL, or a combination of both.

As a specific example, consider a 512B sector of FAT data that only contains a single 32-bit (4B) update compared to the current FAT sector residing in flash. From the current perspective, WAF is used to suggest that the new flash sector can be committed to flash for the WAF required for one sector. As illustrated in this work, this requires a commitment of a 4KB flash allocation unit which may have a WAF of 8, considering that the other 7 sectors adjacent to the new FAT sector are copied internally within the flash memory system.

However, recalling that only a 4B update is required within the FAT sector itself, forcing commitment of 4KB of flash data is actually 1024 (1 K) times the scale of update required by the file system. In this regard, a measure of Host-Flash WAF is proposed and is shown below in Figure 6.1.

$$WAF_{Host-Flash} = \frac{Data\,Written\,to\,Flash}{Data\,Changed\,by\,Host} \tag{6.1}$$

Figure 6.1: Host-Flash WAF Equation

As Figure 6.1 shows, Host-Flash WAF evaluates the flash required as a function of actual data changes. More specifically, this metric transcends the current block interface model and provides a broader measure of file system and FTL evaluation. The Host-Flash WAF metric, solutions that span file systems and FTL improvements can be more effectively evaluated and compared. Specifically, the true impact of file data, including meta data, changes can be measured.

6.2 Sub Flash Allocation Unit FTL

One of the proposals presented in this thesis is a FSAware Enhanced design. Unlike the FSAware design, which can be incorporated into existing FTLs with only the effort required to support multiple active blocks (pools) and the logic to recognize and categorize host write requests, FSAware Enhanced requires rethinking the FTL design.

Page-mode FTLs were once impractical because of the high RAM costs at the time of their invention. However, these costs decreased significantly over time such that overall cost of flash memory systems using page-mode FTLs became much more reasonable. As of this writing, it is not expected that page-mode FTLs will evolve into sector-mode FTLs, nor is there are specific reason to propose such as major change as file data is considered to be large enough to receive little benefit from this design.

However, meta-data is different from file data. It is commonly a single sector, or even smaller if only the true changes of meta-data, as suggested above, are considered. As this work highlighted, flash memory systems can benefit from streaming multiple 512B meta-data writes into a single 4KB flash allocation unit. However, this is only a beginning to effectively researching sub flash allocation unit FTLs.

The current work focused on the FAT file system for its ubiquity to embedded computer systems. However, numerous other file system formats exist for different computer applications. As noted above, some of these are more flash aware than others, but the overall question of efficiency with regards to WAF is relatively new. Many previous studies of file systems on flash memory systems only considered performance. While performance can be an indicator of WAF, parallelism within the flash memory system does not allow direct correlation. As such, the previous studies should be extended to consider WAF. More specifically, the previous studies should

be extended to consider Host-Flash WAF.

In addition to using the Host-Flash WAF as a metric for alternative file systems, the previous studies should be further extended to consider sub-flash allocation unit FTLs. How many different types of meta-data do they use? What is the temporal proximity of their commitment? How many pools of sub-flash allocation unit FTLs are required? Most importantly, is there a file system and FTL combination that approaches unity for Host-Flash WAF?

Chapter 7

Summary and Contributions

This thesis has developed a novel flash memory system for embedded computer systems using modern flash memory that can potentially provide a lifetime comparable to previous generations of flash memory systems using the current flash memory. Due to discrepancies in the specifications between modern flash memory and previous generations of flash memory [2, 90, 116, 50, 1], this thesis requires a sizable increase in reliability improvements. As such, the flash memory system that this thesis is targeting could be called *ultra-reliable*.

The lifetime model of a flash memory system consists of relatively few parameters. Among these, only a single factor, WAF, was considered to be adjustable by flash memory system designers. Therefore, this thesis focused on reducing WAF. This effort required a means of measuring, modeling, and simulating the effects of designs that were intended to reduce WAF. A secondary goal of this thesis was to maintain applicability to commercially available flash memory systems. The major contributions of this thesis include:

- FTLProbe Measurement Technique: A novel instrumentation technique called

FTLProbe was developed to study flash memory systems. This technique purposely saturates the flash memory system with specific sequences of write requests and provides performance measurements of the individual transfers.

- Empirical (Gray Box) Models of FTLs: The array of fine-grained performance measurements produced by FTLProbe were analyzed to identify periodic performance impacts outside of nominal. Both the magnitude and frequency of these impacts were interpreted in the context of flash management activities. Empirical (grey box) FTL models were then constructed from these flash management activities. These models were conceptual in nature, but also provided a mathematical basis of flash management that later allowed WAF equations to be developed from these models. These empirical models were novel in that they enabled analytical study of commercially available products. Previous analytic works could only study theoretical models as FTLs in commercially available products are regarded as trade secrets and operations details are not commonly disclosed.

- WAF Measurements and Equations: The FTLProbe measurement technique facilitated measurement of Write Amplification Factor (WAF) in commercially available flash memory systems. As presented in Section 3.3.1, only a single WAF measurement of a simplified flash memory system has been published. This work presented an array of WAF measurements for more complex flash memory systems including those utilizing a block-mode FTL and a page-mode FTL. These measurements also served to validate WAF equations developed using the empirical FTL models.

- Demonstrated Improvement of Page-Mode FTL over Block-Mode FTL: Analytic studies have speculated that page-mode FTLs are more effective at reducing WAF than are block-mode FTLs. The measurements presented in this thesis conclusively demonstrate that page-mode FTLs have lower WAF than block-mode FTLs. Specifically, they are more efficient overall and, most significantly, page-mode FTLs do not suffer any thrash (consolidation) when transitioning between sequential and random transfers.

- Recognition of "Repeated" as a Significant Transfer Mode: The perspective of considering host write requests as either sequential or random, borrowed from hard disk drive characterization work, has been expanded to include the notion of "repeated" write requests. Repeated writes are writes conducted at specific logical address such as commonly occurs for file system meta-data. While repeated requests can appear as sequential to hard disk drives, they appear as random to flash memory systems. This transfer mode is the least efficient in terms of WAF and should not be defaulted to for commonly encountered write operations such as those for meta-data.

- FSAware: A file system aware (FSAware) FTL extension was developed. FSAware is a multiple-pool FTL that specifically separates data and meta-data by understanding the file system format that is installed on the flash memory system by the host computer platform to organize its files. Specifically, one pool is used for file system data and other pools are used for meta-data types such as FAT and directory structures. The FSAware extension works harmoniously with existing FTL designs suggesting that no change in existing FTL designs, other than the addition of multiple active blocks (pools) are required. Expansion of the empirical models and validation using measurements of the

commercially available flash memory systems that were used to develop the empirical models indicate that FSAware can reduce WAF by 97% for block-mode FTLs and by 38% for page-mode FTLs for 4KB files.

- FSAware Enhanced: FSAware Enhanced is a multiple-pool FTL extension similar to FSAware. However, the FSAware Enhanced recognizes that multiple meta-data writes occur proximate in time with each other and proposes grouping meta-data into a single flash memory system allocation unit. This sector-mode FTL is a more significant FTL development activity. This technique could not be simulated with a commercially available flash memory systems used in this study. However, further extension of the empirical models indicates that FSAware Enhanced can reduce WAF by 99% for block-mode FTLs and by 64% for page-mode FTLs for 4 KB files.

- Ultra-Reliable Flash Memory System: By significantly reducing WAF, FSAware and FSAware Enhanced can compensate for the general lack of detailed specifications of modern flash memory and provide the basis of an ultra-reliable flash memory system.

Bibliography

[1] Abraham, M. How good is your memory? An architect's look inside SSDs. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2015/20150811_FB11_Abraham.pdf. 2015 Flash Memory Summit.

[2] Abraham, M. NAND flash architecture and specification trends. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_TB11_Abraham.pdf. 2012 Flash Memory Summit.

[3] Agarwal, R., and Marrow, M. A closed-form expression for write amplification in NAND flash. In *2010 IEEEGLOBECOM Workshops (GC Wkshps)* (2010), pp. 1846–1850.

[4] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J., Manasse, M., and Panigrahy, R. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference (ATC '08)* (2008), pp. 57–70.

[5] AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS), Revision 4a. http://www.t13.org/documents/uploadeddocuments/docs2007/d1699r4a-ata8-acs.pdf. May 21, 2007.

[6] Baek, S., Ahn, S., Choi, J., Lee, D., and Noh, S. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT 2007)* (2007), pp. 154–163.

[7] Ban, A. Flash file system. United States Patent, No. 5,404,485. (1995/04/04).

[8] Ban, A. Flash file system optimized for page-mode flash technologies. United States Patent, No. 5,937,425. (1999/08/10).

[9] Ban, A. Wear leveling of static areas in flash memory. United States Patent, No. 6,732,221. (2004/05/04).

[10] Ben-Aroya, A., and Toledo, S. Competitive analysis of flash memory algorithms. In *ACM Transactions on Algorithms (TALG), Volume 7, Issue 2, Article 23* (2011).

[11] Birrell, A., Isard, M., Thacker, C., and Wobber, T. A design for high-performance flash disks. In *ACM SIGOPS Operating Systems Review - Systems Work at Microsoft Research, Volume 41, Issue 2* (2007), pp. 88–93.

[12] Boboila, S., and Desnoyers, P. Performance models of flash-based solid-state drives for real workloads. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)* (2011), pp. 8–8.

[13] Bux, W. Write amplification analysis in flash-based solid state drives. In *IBM Research, RZ 3757* (2009).

[14] CrystalDiskMark. `http://crystalmark.info/software/CrystalDiskMark/index-e.html`. Accessed on October 10, 2014.

[15] Cedar, Y. The flash memory industry - onward and upward. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2011/20110809_Keynote1_Cedar.pdf. 2011 Flash Memory Summit.

[16] Chang, L. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)* (2007), pp. 1126–1130.

[17] Chang, L., and Kuo, T. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (2002), pp. 187–196.

[18] Chang, Y., Hsieh, J., and Kuo, T. Improving flash wear-leveling by proactively moving static data. In *IEEE Transactions on Computers, Volume 59, Issue 1* (2010), pp. 53–65.

[19] Chen, F., Koufaty, D., and Zhang, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)* (2009), pp. 181–192.

[20] Chen, F., Luo, T., and Zhang, X. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)* (2011), pp. 6–6.

[21] Chiang, M., Lee, P., and Chang, R. Managing flash memory in personal communication devices. In *Proceedings of 1997 IEEE International Symposium on Consumer Electronics (ISCE '97)* (1997), pp. 177–182.

[22] Choi, H., Lim, S., and Park, K. JFTL: A flash translation layer based on a journal remapping for flash memory. In *ACM Transactions on Storage (TOS), Volume 4, Issue 4, Article 14* (2009).

[23] Choudhuri, S., and Givargis, T. Performance improvement of block based NAND flash translation layer. In *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)* (2007), pp. 257–262.

[24] Chung, T., Park, D., Park, S., Lee, D., Lee, S., and Song, H. System software for flash memory: A survey. In *Embedded and Ubiquitous Computing Lecture Notes in Computer Science Volume 4096* (2006), pp. 394–404.

[25] Clark, P. Sandisk's 1Y flash stays at 19-nm. `http://www.eetimes.com/document.asp?doc_id=1280886`. EETimes (2013/05/28).

[26] Cooke, J. Flash memory 101: An introduction to NAND flash. `http://www.eetimes.com/document.asp?doc_id=1272118`. EETimes (2006/03/20).

[27] Dan, R., and Singer, R. Implementing MLC NAND flash for cost-effective, high-capacity memory. M-Systems (2003/09).

[28] Desnoyers, P. Analytic models of SSD write performance. In *ACM Transactions on Storage (TOS), Volume 10 Issue 2, Article 8* (2014).

[29] Doh, I., Choi, J., Lee, D., and Noh, S. Exploiting non-volatile RAM to enhance flash file system performance. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT '07)* (2007), pp. 164–173.

[30] Flash spot price. http://www.dramexchange.com/. Accessed on September 16, 2014.

[31] Trendforce: NAND flash market to show steady growth in 2014, SSD key to momentum. http://www.dramexchange.com/WeeklyResearch/Post/2/3590.html. IHS (2013/11/27).

[32] Samsung turns to NAND flash as process technology driver. http://www.eetimes.com/document.asp?doc_id=1137456. EETimes (2003/10/06).

[33] Elliott, J., and Brennan, B. Industry innovation with Samsung's next generation V-NAND. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2014/20140805_Keynote2_Samsung.pdf. 2012 Flash Memory Summit.

[34] Everspin Technologies Inc MR2A16AMA35 (Digi-Key, 819-1018-ND). http://www.digikey.com/product-detail/en/MR2A16AMA35/819-1018-ND/2665192. Accessed on September 23, 2014.

[35] Design and implementation of the second extended filesystem. http://web.mit.edu/tytso/www/linux/ext2intro.html.

[36] Introducing ext3. http://www.ibm.com/developerworks/linux/library/l-fs7/.

[37] Ext4 filesystem. https://www.kernel.org/doc/Documentation/filesystems/ext4.txt.

[38] Fat: General overview of on-disk format. http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc. December 6, 2000.

[39] Detailed explanation of fat boot sector. https://support.microsoft.com/en-us/kb/140418.

[40] Fatfs - generic fat file system module. http://elm-chan.org/fsw/ff/00index_e.html.

[41] Lifetime achievement award: Simon Sze. http://www.flashmemorysummit.com/English/Conference/Seminar_Session_Descriptions.html#ThursLAA. Flash Memory Summit 2014.

[42] Frankie, T., Hughes, G., and Kreutz-Delgado, K. Analysis of TRIM commands on overprovisioning and write amplification in solid state drives. In *ACMTransactions on Storage* (2012).

[43] Frohman-Bentchkowsky, D. A fully decoded 2048-bit electrically programmable FAMOS read-only memory. In *IEEE Journal of Solid-State Circuits, Volume 6, No. 5* (1971), pp. 301–306.

[44] Fulford, B. Unsung hero. http://www.forbes.com/global/2002/0624/030.html. Forbes (2002/06/24).

[45] Gal, E., and Toledo, S. Algorithms and data structures for flash memories. In *ACM Computing Surveys (CSUR), Volume 37, Issue 2* (2005), pp. 138–163.

[46] Grimsrud, K. Challenges & opportunities for NVM in Intel architecture platforms. Intel Developer Forum (2007).

[47] Gupta, A., Kim, Y., and Urgaonkar, B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV)* (2009), pp. 229–240.

[48] Gupta, A., Pisolkar, R., Urgaonkar, B., and Sivasubramaniam, A. Leveraging value locality in optimizing NAND flash-based SSDs. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)* (2011), pp. 7–7.

[49] Handy, J. The changing relationship of flash & DRAM SSDs. `http://www.storagesearch.com/ssd-ram-flash%20pricing.html`. Storage Search (August 2007).

[50] Handy, J. Top ten things you need to know. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2014/20140807_304A_Handy.pdf`. 2014 Flash Memory Summit.

[51] Houdt, B. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '13)* (2013), pp. 191–202.

[52] Hu, X., and Haas, R. The fundamental limit of flash random write performance: Understanding, analysis and performance modelling. In *IBM Research, RZ 3771* (2010).

[53] Hu, Y., Eleftheriou, E., Haas, R., Iliadis, I., and Pletka, R. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference (SYSTOR '09), Article 10* (2009).

[54] Hu, Y., Jiang, H., Feng, D., Tian, L., Zhang, S., Liu, J., Tong, W., Qin, Y., and Wang, L. Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010), pp. 1–12.

[55] NAND flash revenue to rise again in 2011. `https://technology.ihs.com/395005/nand-flash-revenue-to-rise-again-in-2011`. IHS (2011/01/13).

[56] Understanding the flash translation layer (FTL) specification, application note AP-684. Intel (1998).

[57] IOMeter. `http://www.iometer.org`. Accessed on October 10, 2014.

[58] Jiao, L., Zhang, Y., and Lin, W. Journal-based block images for flash memory storage systems. In *The 9th International Conference for Young Computer Scientists (ICYCS 2008)* (2008), pp. 1331–1336.

[59] Jurenka, M. Exploring managed NAND media endurance. Master's thesis, Boise State University, 2010.

[60] Kahng, D., and Sze, S. A floating-gate and its application to memory devices. In *The Bell System Technical Journal, Volume 46, Number 4* (1967), pp. 1288–1295.

[61] Kang, J., Jo, H., Kim, J., and Lee, J. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software (EMSOFT '06)* (2006), pp. 161–170.

[62] Kawaguchi, A., Nishioka, S., and Motoda, H. A flash-memory based file system. In *Proceedings of the USENIX 1995 Winter Technical Conference* (1995), pp. 155–164.

[63] Kim, J., Kim, H., Lee, S., and Won, Y. FTL design for TRIM command. In *The Fifth International Workshop on Software Support for Portable Storage* (2010), pp. 7–12.

[64] Kim, J., Kim, J., Noh, S., Min, S., and Cho, Y. A space-efficient flash translation layer for CompactFlash systems. In *IEEE Transactions on Consumer Electronics, Volume 48, Issue 2* (2002), pp. 366–375.

[65] Understanding over-provisioning. `http://www.kingston.com/us/ssd/overprovisioning`.

[66] Kwon, S., and Chung, T. An efficient and advanced space-management technique for flash memory using reallocation blocks. In *IEEE Transactions on Consumer Electronics, Volume 54, Issue 2* (2008), pp. 631–638.

[67] Kwon, S., Ranjitkar, A., Ko, Y., and Chung, T. FTL algorithms for NAND-type flash memories. In *Design Automation for Embedded Systems, Volume 15, Issue 3-4* (2011), pp. 191–224.

[68] Lee, J., Kim, S., Kwon, H., Hyun, C., Ahn, S., Choi, J., Lee, D., and Noh, S. Block recycling schemes and their cost-based optimization in NAND flash memory based storage system. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT '07)* (2007), pp. 174–182.

[69] Lee, S., Dongkun, S., Kim, Y., and Kim, J. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. In *ACM SIGOPS Operating Systems Review, Volume 42 Issue 6* (2008), pp. 36–42.

[70] Lee, S., Park, D., Chung, T., Le, D., Park, S., and Song, H. A log buffer-based flash translation layer using fully-associative sector translation. In *ACM Transactions on Embedded Computing Systems (TECS), Volume 6, Issue 3, Article 18* (2007).

[71] Lin, P., Chiao, H., and Chang, D. Improving flash translation layer performance by supporting large superblocks. In *IEEE Transactions on Consumer Electronics, Volume 56, Issue 2* (2010), pp. 642–650.

[72] Lu, Y., Shu, J., and Zheng, W. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)* (2013), pp. 257–270.

[73] Luojie, X., and Kurkoski, B. An improved analytic expression for write amplification in NAND flash. In *2012 International Conference on Computing, Networking and Communications (ICNC)* (2012), pp. 497–501.

[74] Ma, D., Feng, J., and Li, G. LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11)* (2011), pp. 1–12.

[75] Masuoka, F., Asano, M., Iwahashi, H., Komuro, T., and Tanaka, S. A new flash E2PROM cell using triple polysilicon technology. In *1984 International Electron Devices Meeting, Volume 30* (1984), pp. 464–467.

[76] Masuoka, F., Momodomi, M., Iwata, Y., and Shirota, R. New ultra high density EPROM and flash EEPROM with NAND structure cell. In *1987 International Electron Devices Meeting, Volume 33* (1987), pp. 552–555.

[77] McCormick, T. Flash challenges for embedded computing. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120822_S203B_McCormick.pdf`. 2012 Flash Memory Summit.

[78] McCormick, T., and Kaeli, D. Empirical FTL evaluation and modeling. (Poster). 5th Annual Non-Volatile Memories Workshop.

[79] McCormick, T., and Kaeli, D. Empirical FTL evaluation and modeling. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2014/20140805_101C_McCormick.pdf`. 2014 Flash Memory Summit.

[80] Memory 1997. `http://smithsonianchips.si.edu/ice/cd/MEMORY97/title.pdf`. Smithsonian (1997).

[81] Menon, J., and Stockmeyer, L. An age-threshold algorithm for garbage collection in log-structured arrays and file systems. In *High Performance Computing Systems and Applications - The Springer International Series in Engineering and Computer Science, Volume 478* (1998), pp. 119–132.

[82] Miller, E., Brandt, S., and Long, D. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (2001), pp. 95–99.

[83] Mladenov, R., and Ivanov, S. A self-tuning hybrid flash translation layer for embedded systems. In *Proceedings of the 12th International Conference on Computer Systems and Technologies (CompSysTech '11)* (2009), pp. 57–62.

[84] Moshayedi, M., and Wilkison, P. Enterprise SSDs. In *ACM Queue* (8 2008), pp. 32–39.

[85] Murugan, M., and Du, D. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)* (2012), pp. 1–12.

[86] New technology file system (ntfs). `https://msdn.microsoft.com/en-us/library/cc230448.aspx#gt_86f79a17-c0be-4937-8660-0cf6ce5ddc1a`.

[87] Pan, Y., Dong, G., and Zhang, T. Exploiting memory device wear-out dynamics to improve NAND flash memory system performance. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)* (2011), pp. 18–18.

[88] Park, C., Cheon, W., Kang, J., Roh, K., Cho, W., and Kim, J. A reconfigurable FTL (flash translation layer) architecture for NAND flash-based

applications. In *ACM Transactions on Embedded Computing Systems (TECS), Volume 7, Issue 4, Article 38* (2008).

[89] Park, C., Talawar, P., Won, D., Jung, M., Im, J., Kim, S., and Choi, Y. A high performance controller for NAND flash-based solid state disk (NSSD). In *21st Non-Volatile Semiconductor Memory Workshop (IEEE NVSMW)* (2006), pp. 17–20.

[90] Park, S. Overcoming the scaling problem for NAND flash. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_Keynote2_Park.pdf`. 2012 Flash Memory Summit.

[91] Park, Y., and Kim, Y. Compression support for flash translation layer. In *The Fifth International Workshop on Software Support for Portable Storage* (2010), pp. 19–24.

[92] Pc card standard - media storage formats specification, Volume 10, Version 8.1. PCMCIA (2002/12).

[93] 1971 - Reusable programmable ROM introduces iterative design flexibility. `http://www.computerhistory.org/semiconductor/timeline/1971-EPROM.html`. Computer History Museum, Retrieved 2014/09/25.

[94] Phison PS3108. `http://www.phison.com/english/newProductView.asp?ID=237&SortID=63`. Accessed on September 25, 2014.

[95] Qin, Z., Wang, Y., Liu, D., Shao, Z., and Guan, Y. NMFTL: An efficient flash translation layer for MLC NAND flash memory storage systems. In *Proceedings of the 48th Design Automation Conference (DAC '11)* (2011), pp. 17–22.

[96] Rajimwale, A., Prabhakaran, V., and Davis, J. Block management in solid-state devices. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09)* (2009), pp. 21–21.

[97] Ramtron FM23MLD16-60-BGTR (Digi-Key, 1140-1039-1-ND). http://www.digikey.com/product-detail/en/FM23MLD16-60-BGTR/1140-1039-1-ND/2931528. Accessed on September 23, 2014.

[98] Rosenblum, M., and Ousterhout, J. The design and implementation of a log-structured file system. In *ACM Transactions on Computer Systems (TOCS), Volume 10, Issue 1* (1992), pp. 26–52.

[99] Ryu, Y. SAT: Switchable address translation for flash memory storages. In *2010 IEEE 34th Annual Computer Software and Applications Conference (COMPSAC)* (2010), pp. 453–461.

[100] Sandisk announces 15 nanometer technology, world's most advanced NAND flash manufacturing node. http://www.sandisk.com/about-sandisk/press-room/press-releases/2014/sandisk-announces-15-nanometer-technology,-worlds-most-advanced-nand-flash-manufacturing-node/. SanDisk (2014/04/22).

[101] Serial ATA International Orgnaization. http://https://www.sata-io.org/. Accessed on October 10, 2014.

[102] Saxena, M., Xhang, Y., Swift, M., Dusseau, A., and Dusseau, R. Getting real: Lessons in transitioning research simulations into hardware systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)* (2013), pp. 215–228.

[103] Silicon Motion to showcase its latest FerriSSD single-package SSD and new SATAII/III SSD controllers at Flash Memory Summit. `http://www.siliconmotion.com/A6.1.Detail_News.php?sn=129`. (2012/08/17).

[104] Soundararajan, G., Prabhakaran, V., Balakrishnan, M., and Wobber, T. Extending SSD lifetimes with disk-based write caches. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '10)* (2010), pp. 8–8.

[105] Taylor, J. Flash at Facebook. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130813_Keynote1_Taylor.pdf`. 2013 Flash Memory Summit.

[106] Templeman, R., and Kapadia, A. GANGRENE: exploring the mortality of flash memory. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security (HotSec '12)* (2012), pp. 1–1.

[107] Tijoe, J. Making a flash translation layer reliability-aware (RA): An optimized strategy for wear-leveling and garbage collection. Master's thesis, San Diego State University, 2011.

[108] Unsworth, J. Market share: Flash cards, USB flash drives and solid-state drives, worldwide, 2010. `https://www.gartner.com/doc/1605323/market-share-flash-cards-usb`. Gartner (2011/03/25).

[109] Wang, C., and Wong, W. ADAPT: Efficient workload-sensitive flash management based on adaptation, prediction and aggregation. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)* (2012), pp. 1–12.

[110] Wang, J., and Hu, Y. WOLF-A novel reordering write buffer to boost the performance of log-structured file system. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02), Atricle 4* (2002).

[111] Wei, Q., Gong, B., Pathak, S., Veeravalli, B., Zeng, L., and Okada, K. WAFTL: A workload adaptive flash translation layer with data partition. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)* (2011), pp. 1–12.

[112] Wu, M., and Zwaenepoel, W. eNVy: A non-volatile, main memory storage system. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)* (1994), pp. 86–97.

[113] Xu, Z., Li, R., and Xu, C. CAST: A page-level FTL with compact address mapping and parallel data blocks. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)* (2012), pp. 142–151.

[114] Yang, M. Will $1 per gigabyte NAND resuscitate solid state drives? `https://technology.ihs.com/388862/will-1-per-gigabyte-nand-resuscitate-solid-state-drives`. IHS (2010/08/19).

[115] Yang, S., and Ryu, Y. An efficient mapping table management in NAND flash-based mobile computers. In *Computational Science and Its Applications - ICCSA 2011 Lecture Notes in Computer Science Volume 6784* (2011), pp. 518–527.

[116] Yoon, S., and Tressler, G. Advanced flash technology status, scaling trends & implications to enterprise SSD technology enablement. `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_TA12_Yoon_Tressler.pdf`. 2012 Flash Memory Summit.