Using MEMS-based storage devices in computer systems

STEVEN W. SCHLOSSER

May 2004
CMU-PDL-04-104

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

A dissertation submitted to the graduate school in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering.

Thesis committee

Gregory R. Ganger, Chair
L. Richard Carley
James C. Hoe
Charles C. Morehouse, Hewlett-Packard Laboratories

Keywords: MEMS-based storage, storage systems performance, database systems, disk arrays

To my family: my parents, Phil and Kathy; my sister, Jennifer; and my wife, Rachel.

Abstract

MEMS-based storage is an interesting new technology that promises to bring fast, non-volatile, mass data storage to computer systems. MEMS-based storage devices (MEMStores) themselves consist of several thousand read/write tips, analogous to the read/write heads of a disk drive, which read and write data in a recording medium. This medium is coated on a moving rectangular surface that is positioned by a set of MEMS actuators. Access times are expected to be less than a millisecond with energy consumption 10–100× less than a low-power disk drive, while streaming bandwidth and volumetric density are expected to be around that of disk drives.

This dissertation explores the use of MEMStores in computer systems, with a focus on whether systems can use existing abstractions and interfaces to incorporate MEMStores effectively, or if they will have to change the way they access storage to benefit from MEMStores. If systems can use MEMStores in the same way that they use disk drives, it will be more likely that MEMStores will be adopted when they do become available.

Since real MEMStores do not yet exist, I present a detailed software model that allows their use to be explored under a variety of workloads. To answer the question of whether a new type of device requires changes to systems, I present a methodology that includes two objective tests for determining whether the benefit from a device is due to a specific difference in how that device accesses data or is just due to the fact that that device is faster, smaller, or uses less energy than

current devices. I present a range of potential uses of MEMStores in computer systems, examining each under a number of user workloads, using the two objective tests to evaluate their efficacy.

Using the evidence presented and the two objective tests, I show that systems can incorporate MEMStores easily and employ the same standard abstractions and interfaces used with disk systems. At a high level, the intuition is that MEMStores are mechanical storage devices, just like disk drives, only faster, smaller, and requiring less energy to operate. Accessing data requires an initial seek time that is distance-dependent, and, once access has begun, sequential access is the most efficient. This intuition is described in more detail, and the result is shown to hold for the range of uses presented.

Acknowledgements

Not one word of this dissertation could have been written without a great deal of help and support from many people. Being a graduate student in the Parallel Data Laboratory has been the most rewarding experience of my life, and I am grateful for having had the opportunity to work with such a fantastic group.

The most important thanks go to my collaborator and friend, John Linwood Griffin. We started working on this project in 1999, built the model and simulator together, and produced many of the results presented here. I will always be grateful for his help, insight, and friendship. I also owe special thanks to Jiri Schindler, Minglong Shao, and Natassa Ailamaki, who have helped me a great deal in the last two years to apply some the insights of this dissertation to disk drives and database systems.

I have been fortunate to have the support of the members of my thesis committee, who have been crucial to developing this work. Greg Ganger, my advisor, has taught me more than I ever thought I could know about computer systems, storage devices, and navigating the world of research. In addition to taking on the responsibilities of at least four full-time jobs simultaneously, he is always available to his students, which is truly amazing. Rick Carley provided the initial impetus for this work by starting the MEMS-based storage research project in the MEMS Laboratory at Carnegie Mellon. He had the great foresight of involving computer systems researchers in the effort, which led to this work. James Hoe was invaluable in filling the role of the outsider, making sure that I was covering all the right

bases. I've also been fortunate to have the support and input of Chuck Morehouse, who leads the probe storage effort at Hewlett-Packard Laboratories. I also want to thank David Nagle, who not only helped start this research, but also made it possible for me to attend graduate school.

The students and staff of the Parallel Data Laboratory have been very helpful and supportive over the years. Craig Soules, Brandon Salmon, Eno Thereska, Mike Abd-El-Malek, Garth Goodson, Jay Wylie, Andy Klosterman, Shuheng Zhou, John Strunk, Ted Wong, David Petrou, and John Bucy, have all helped over the years in ways too numerous to mention. Linda Whipkey and Karen Lindenfelser keep our office and lab running smoothly and I can't thank them enough for their help. Thanks also to Joan Digney who endures our constant lateness when preparing posters, but still finds it in her heart to help a student proofread his dissertation.

Contents

Figures

Tables

1 Introduction

MEMS-based storage devices (MEMStores) are radically different from today's bulk storage devices of choice: disk drives and semiconductor memory devices. MEMStores are fabricated from wafers of silicon, in much the same manner as microprocessors and memories, but they are mechanical in nature, much like disk drives. Their physical size is very small, less than one cubic centimeter, but their capacity is large, on the order of two to ten gigabytes. Most importantly, their small size and inherently parallel data access lead to a number of compelling advantages over current technologies: low access latency, high access bandwidth, and low energy utilization. These advantages make them an interesting technology to consider in computer systems.

Random accesses to a MEMStore are anticipated to be faster than today's disk drives by, approximately, a factor of ten and their density is expected to be much greater than that predicted of semiconductor memory devices like FLASH and MRAM for the foreseeable future. Their speed and capacity place MEMStores into the memory hierarchy most comfortably somewhere between disk drives and semiconductor memory devices. This dissertation explores how MEMStores could be used in computer systems, including examining specific examples and addressing the general issue of whether new interfaces and abstractions will be required.

It is important to re-evaluate systems whenever new technologies arrive. The researcher's role is vital in this regard because he or she has the freedom to think outside the box and consider radical changes to systems. However, this thinking

must be tempered with the reality that new technologies can be most successful if they require few changes to existing systems. As the researcher identifies uses of new technologies, he or she should not only consider potential improvements, but also the cost of making making those improvements possible. One of the central contributions of this dissertation is a methodology for considering such trade-offs when investigating new technologies. In studying the use of MEMStores in systems and their potential impact on storage abstractions and interfaces, I have developed this methodology and codified it into two simple objective tests, which are described below.

Systems use abstract, simplified interfaces like SCSI and ATA to access storage devices. Through these interfaces (or abstractions), systems view storage devices as a linear array of fixed-sized logical blocks, most commonly 512 bytes each, which are referred to with logical block numbers (LBNs). These interfaces are useful because they hide the complexities of underlying storage devices, they allow storage devices to be interchangeable, and they eliminate the need for the system to directly manage the details of the devices. Before the abstraction was standardized, different types of disk drives, and even different models of disks from a single vendor, required proprietary interface hardware, interconnects, and software to be used, greatly complicating systems. While such simplification is clearly useful, using any high-level abstraction runs the risk of hiding potentially beneficial details of the device that a system could exploit to improve performance. Thus, there is a tension between the ease of integration that standard abstractions provide and the extra performance that more information could give.

Despite its simplicity, and the detailed information it hides, the standard abstraction of SCSI and ATA have served the storage industry for many years and all signs point to their continued use in most systems. As new storage devices are introduced, it is important to re-examine the standard abstractions and their use for those new technologies. Industry acceptance strongly encourages new technologies to use existing interfaces, for good reason, as interoperability and ease of

use are crucial to the acceptance of new technologies. However, it is important to consider whether anything is lost in using abstractions developed for old devices with new technologies.

An instructive example is the introduction of disk arrays in the early 1990s. Some have argued that the standard linear abstraction hides the inherent parallel access to data stored in a disk array, and that extended interfaces could allow improvements in performance. However, such extended interfaces have never reached the marketplace because few real-world workloads take advantage of them. Hence, adding the complexity of a new interface is not justified for the majority of customers.

1.1 Thesis statement

MEMS-based storage offers significant performance and energy consumption advantages over today's mass storage devices (i.e., disk drives). Despite this fact, the linear logical block abstraction used in the interface for other storage devices is appropriate for MEMS-based storage devices because of their particular data access characteristics.

1.2 Overview

The main question that this dissertation seeks to answer is whether MEMStores are sufficiently different from existing devices, specifically disk drives, to require new interfaces or abstractions, or whether those that are already in use are sufficient. In order to answer this question, the right comparison to make is not between MEMStores and disk drives of today. Rather, the comparison should be made between MEMStores and hypothetical disk drives of equal average performance, even though such disk drives do not, and may never, exist. If the benefit of using a faster disk drive is the same as that when using a MEMStore, then the benefit simply stems from the fact that both devices are faster. If the benefit of using a

MEMStore is greater than that of using a fast disk drive, then the workload must be exploiting something specific about the MEMStore that the disk drive does not have, or does not do. It is these specific differences that will motivate the use of a new interface and abstraction. If the benefit is the same, then the abstraction can remain unchanged. It is this methodology that I use to support the thesis of this dissertation.

This dissertation considers the use of MEMStores in computer systems in three basic ways. First, it describes benefits that systems can gain by using MEMStores for bulk storage. Second, it uses some of these insights to influence their basic design. Third, it shows that systems can employ well-known abstractions and interfaces developed for disk drives to access MEMStores, and can reap the benefits of MEMStores using such interfaces. Neither of the first two points draw any conclusions other than the fact that MEMStores are faster than today's disk drives, and that systems can benefit from faster devices. While it is interesting to note the technical reasons behind such advantages, the third point addresses the larger, meta-question of whether MEMStores are fundamentally different from disk drives (from the rest of the computer system's perspective) in useful ways, or if they are basically the same, only faster. Therefore, the argument of the dissertation is formed largely around the third point. If a MEMStore is fundamentally the same as a disk drive (only faster), then systems can use the same abstraction and interface for both. If the two devices are fundamentally different, and systems utilize different characteristics of each device, then the abstraction and interface will have to change.

Put simply, the question that this dissertation seeks to answer is whether MEMStores should be treated by computer systems as anything other than fast, small, low-power disk drives. These qualities are certainly desirable and can lead to benefits for systems. In fact, it is thought that the performance of MEMStores will exceed that of disk drives for many years to come, both in terms of access speed and energy consumption. None would disagree that faster devices, if used

properly in systems, will lead to faster systems. However, in this dissertation I seek to find advantages of MEMStores beyond the simple improvements in access time and energy consumption. If there are truly MEMStore-specific mechanisms that a system can take advantage of, then, most likely, there must be a fundamental change in the abstraction and interface that is used to access them.

A change to the storage abstraction could be as simple as the system knowing the type of device behind the abstraction, or could it be as complex as the system keeping a detailed model of the device. If the system is to exploit a specific feature of MEMStores, then the abstraction must at least change so that the system is aware that the device is a MEMStore and will probably require more device-specific information. Some would argue that the fact that MEMStores are faster and require less energy than disk drives is compelling in and of itself. This is certainly true, but simple speed improvements do not require a change in the abstract view of storage used by systems.

This dissertation answers the question using two complementary approaches. First, it examines the reasons why the current abstraction works well for disk drives and shows that those reasons hold for MEMStores as well. Second, it uses two simple tests to decide whether new abstractions are justified.

1.2.1 Roles and policies

This dissertation divides the aspects of MEMStore use in systems into two categories: roles and policies. MEMStores can take on various *roles* in a system, such as disk replacement, cache for hot blocks, metadata-only storage, etc. For the debate at hand, the associated sub-question is whether a system using a MEMStore is exploiting something MEMStore-specific (e.g., because of a particularly well-matched access pattern) or just benefitting from its general properties (e.g., that they are faster than current disks). In any given role, external software uses various *policies*, such as data layout and request scheduling, for managing underlying storage. The sub-question here is whether MEMStore-specific policies are needed,

or are those used for disk systems sufficient.

1.2.2 Objective tests

To help answer the top-level question, I use two simple objective tests. The first test, called the *specificity test*, asks: Is the potential role or policy truly MEMStore-specific? To test this, I evaluate the potential role or policy for both a MEMStore and a (hypothetical) disk drive of equivalent performance. If the benefit is the same, then the potential role or policy (however effective) is not truly MEMStore-specific and could be just as beneficial to disk drives.

The second test, called the *merit test*, asks: Given that a potential role or policy passes the specificity test, does it make enough of an impact in performance (e.g., access speed or energy consumption) to justify a new abstraction? The test here is a simple improvement comparison, e.g., if the system is less than 10% faster when using the new abstraction, then it's not worth the cost.

These two tests codify a general rule in engineering: that the costs of incorporating new technologies should be considered when suggesting changes to systems. While this dissertation uses the methodology specifically to evaluate the use of MEMStores in systems, it is generic and can be used for any new device.

1.3 Contributions

This dissertation makes four primary contributions:

- It describes the various instances of MEMStores under development. It also describes the model of MEMStores developed for this dissertation and compares it to others in the literature.

- It examines the current abstraction used for disk drives and why it works, and shows why the abstraction works for MEMStores.

- It presents two objective tests that can be used to evaluate the use of new technologies in systems and whether any changes that are required to those systems justify the benefits that may be gained. This methodology is generic and can be used to evaluate the use of any new technology in systems.

- It introduces several potential roles and policies and applies the objective tests to evaluate the efficacy of the standard disk abstraction for accessing MEMStores.

1.4 Organization

This dissertation is organized as follows. Chapter 2 gives a detailed description of MEMStores and presents related work. Chapter 3 provides details on the model used in this dissertation. Chapter 4 describes the abstraction used by current storage systems, why that abstraction works well for disks, and why it should work well for MEMStores. Chapter 5 explores several potential roles MEMStores may take in computer systems. Chapter 6 describes several potential policies for tailoring system access to MEMStores. Chapter 7 concludes the dissertation and presents future work.

2 Background and related work

The MEMStores that have been described in the literature share many similarities, both in design and performance. This chapter describes in some detail the three most widely publicized incarnations, with an emphasis on the design being built at Carnegie Mellon, on which much of my work is based. As well, it describes the field of research to date studying the use of MEMStores in systems.

Building practical MEMStores has been the goal of several major research labs, universities, and startup companies around the world for over a decade. The three most widely publicized efforts are from Carnegie Mellon University, IBM Research in Zurich, and Hewlett-Packard Laboratories. The three designs differ largely in the type of actuators which are used to position the media and the method used to record data in the medium. Despite these differences, however, each design shares the same basic architecture shown in Figure 2.1 utilizing a moving media sled and a large array of read/write tips. It would also be possible to put the read/write tips onto the moving sled while the media remains fixed, although no published designs do so.

2.1 Basic device description

Published MEMStore designs utilize moving media, much like the media in disk drives, and an array of read/write probe tips to access data stored in the media. Unlike a disk, however, the media does not rotate because it is difficult to build rotating components using MEMS processes. Instead, current designs use a movable

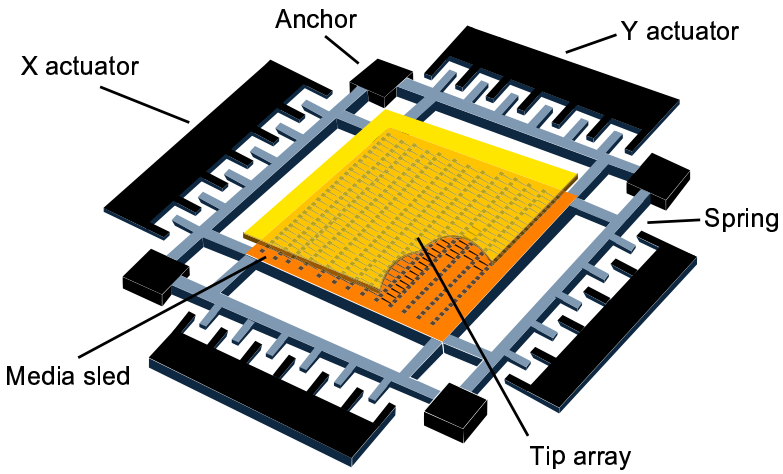


Fig. 2.1: **Components of a MEMS-based storage device.** The media sled is suspended above an array of probe tips. The sled moves small distances along the X and Y axes, allowing the stationary tips to address the media.

media sled, which is coated with the media material. This sled is spring-mounted and can be pulled in the X and Y dimensions by actuators on each edge below a two-dimensional array of fixed read/write heads or *probe tips*. To access data, the media sled is first pulled to a specific location (x,y displacement). When this seek is complete, the sled moves in the Y dimension at a constant velocity while the probe tips access the media. With the exception of minute movements in the X and Z dimensions to adjust for surface variation and skewed tracks, the probe tips remain stationary while the media sled moves. In contrast, rotating platters and actuated read/write heads share the task of positioning in disks. Figures 2.1 and 2.2 illustrate this MEMStore design.

2.1.1 Carnegie Mellon University

The device under development at Carnegie Mellon uses magnetic recording to store data, similar to today's disk drives. This choice was made for two reasons. First, magnetic recording in disk drives is a very well-understood process. Second, it does not require contact between the media and the read/write tips, avoiding questions of physical wear. Using magnetic recording in a MEMStore, however, does present

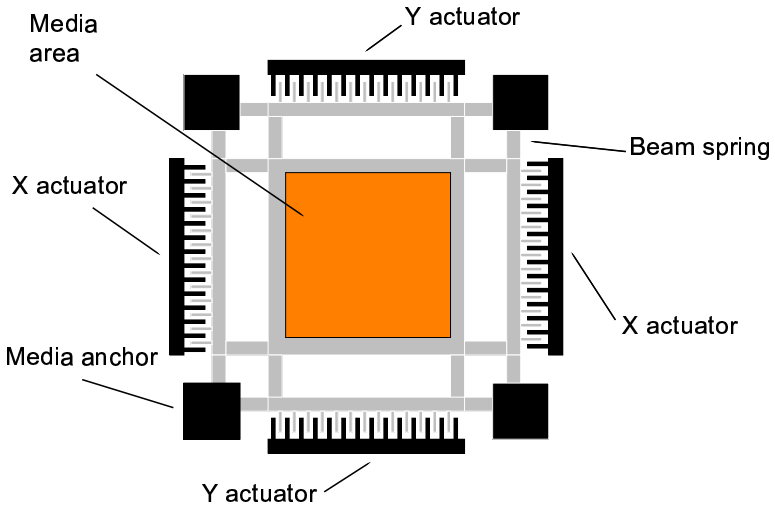


Fig. 2.2: **The movable media sled.** The actuators, spring suspension, and the media sled are shown. Anchored regions are solid and the movable structure is shaded grey.

challenges. First, the tip/media spacing must be very carefully controlled, which requires a complex active servo system that must be replicated for each read/write tip adding complexity and requiring more power for each tip. Second, depositing magnetic materials can be incompatible with manufacturing MEMS components.

The research group at Carnegie Mellon has explored several design points, varying parameters such as the media footprint, the number and type of read/write tips, and the size of bits stored in the media. I have chosen one such design point to highlight throughout the dissertation, and explored some others to understand the sensitivity of the models to varying parameters. These are described in more detail in Chapter 3. Much of the discussion that follows is based on one of these design points, which is called the G2 or "second generation" model. This model has a media footprint of 196 mm^2, with 64 mm^2 of usable media area and 6400 probe tips [Carley et al. 2000]. Dividing the media into bit cells of 40×40 nm, and accounting for an ECC and encoding overhead of 2 bits per byte, this design has a formatted capacity of 3.2 GB/device.

In the CMU design, each bit cell has a square aspect ratio, which is not the case in conventional disk drives. Bits stored in disk drives have a relatively high aspect ratio to increase signal to noise ratio in the face of oscillations of the seek arm. The media sled in a MEMStore can be positioned much more accurately than the heads in a disk drive, making square bits possible. This positioning accuracy and the smaller aspect ratio it enables results in higher areal densities in MEMStores than in disks. However, the smaller media area results in a smaller per-device capacity of MEMStores relative to disks.

2.1.2 IBM Millipede

IBM's Millipede design shares many similarities to the CMU design but is different in some technical details. First, data is recorded using a novel thermomechanical recording technique in which the probe tips are placed in physical contact with a plastic media. To write a bit, a probe tip is heated, melting a depression into the media. To read back data, the probe tips are dragged across the media surface. When a probe tip falls into a depression, its displacement is detected, indicating a bit. Data is erased either in bulk by heating the media, allowing the plastic to re-flow into the pits, or by point overwrites of data.

This recording technique simplifies some aspects of the device significantly. Since the probes are held in contact with the media, there is no need for individual control over the tip/media spacing. This, along with the simplicity of the read/write mechanism, could reduce the energy requirements of each tip, increasing the number of tips that can be used concurrently. Constant contact, however, leads to questions of wear both of the media and of the tips. While initial results [Terris et al. 1998] suggest that the media is resilient enough to withstand contact, anecdotal evidence suggests that possible re-write and even re-read limits continue to be a concern for this technology.

The Millipede prototype uses electromagnetic actuators, in contrast to the electrostatic actuators of the CMU design. These actuators provide much greater

force, potentially increasing performance or, at least, providing the same force using less energy. However, the energy consumption is likely to have a different dynamic. Electromagnetic actuators draw more current, and hence consume more energy, as the media sled is pulled further from its rest position [Rothuizen et al. 2000; Vettiger et al. 2002]. Electrostatic actuators require higher voltages as the sled is displaced further, but require little current so energy consumption is lower overall and has less dependence on displacement. This difference could lead to interesting trade-offs between positioning distance and energy consumption for MEMStores with electromagnetic actuators.

2.1.3 Hewlett-Packard Labs Atomic-Resolution Storage (ARS)

The device being designed in the Atomic Resolution Storage project at Hewlett-Packard Laboratories is similar in structure to the CMU and IBM devices, but it, again, uses a different media actuator and recording scheme. Its media actuator uses electrostatic stepper motors and the recording scheme uses electron beams to make marks in phase-change media [Hewlett-Packard 2002]. The electrostatic motor is mechanically different from the electrostatic comb fingers in the CMU design but is likely to have similar performance and energy consumption characteristics. Using electron beam recording eliminates the need for constant tip/media spacing, which further simplifies tip design and reduces energy requirements.

2.2 Low-level data layout

All MEMStore designs that appear in the literature store data in a linear fashion, i.e., in columns, as illustrated in Figure 2.3. The storage media on the sled is divided into rectangular regions as shown in Figure 2.3. Each region contains M×N bits (e.g., 2500×2500) and is accessible by exactly one probe tip; the number of regions on the media equals the number of probe tips. Each term in the nomenclature below is defined both in the text and visually in Figure 2.4.

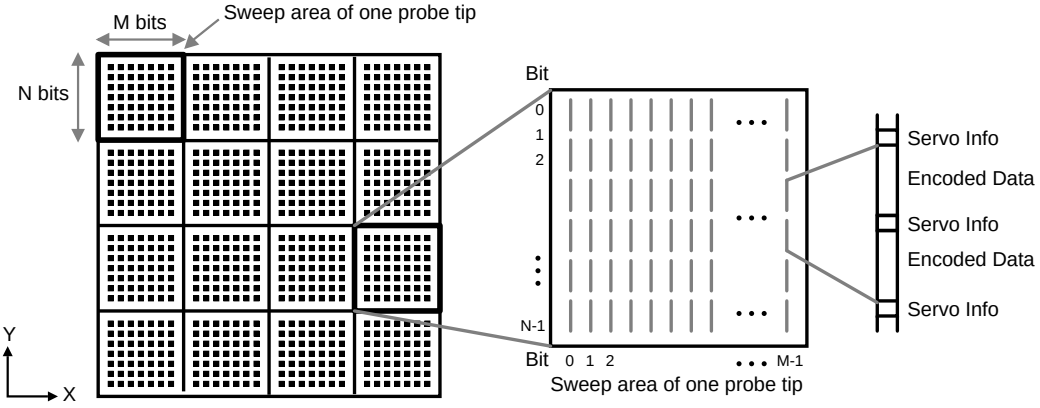


Fig. 2.3: **Data organization on MEMS-based storage devices.** The illustration depicts a small portion of the magnetic media sled. Each small rectangle outlines the media area accessible by a single probe tip, with a total of 16 tip regions shown. A full device contains thousands of tips and tip regions. Each region stores M×N bits, organized into M vertical columns of N bits, alternating between servo/tracking information (10 bits) and data (80 bits = 8 encoded data bytes). To read or write data, the media sled passes over the tips in the ±Y directions while the tips access the media.

Cylinders. Drawing on the analogy to disk terminology, a *cylinder* is the set of all bits with identical x offset within a region (i.e., at identical sled displacement in X). In other words, a cylinder consists of all bits accessible by all tips when the sled moves only in the Y dimension, remaining immobile in the X dimension. Cylinder 1 is highlighted in Figure 2.4 as the four circled columns of bits. This definition parallels that of disk cylinders, which consist of all bits accessible by all heads while the arm remains immobile. There are M cylinders per sled. In the G2 model described in detail below, each sled has 2500 cylinders that each hold 1350 KB of data.

Tracks. A MEMStore might use 6400 read/write tips to access its media; however, due to power and heat considerations it is unlikely that all 6400 tips can be *active* (accessing data) concurrently. Device designers expect to be able to activate 200–2000 tips at a time. To account for this limitation, cylinders are divided into tracks. A *track* consists of all bits within a cylinder that can be read by a group of concurrently active tips. The sled in Figure 2.4 has sixteen tips (one per region; not all tips are shown), of which up to four can be concurrently active—each cylinder

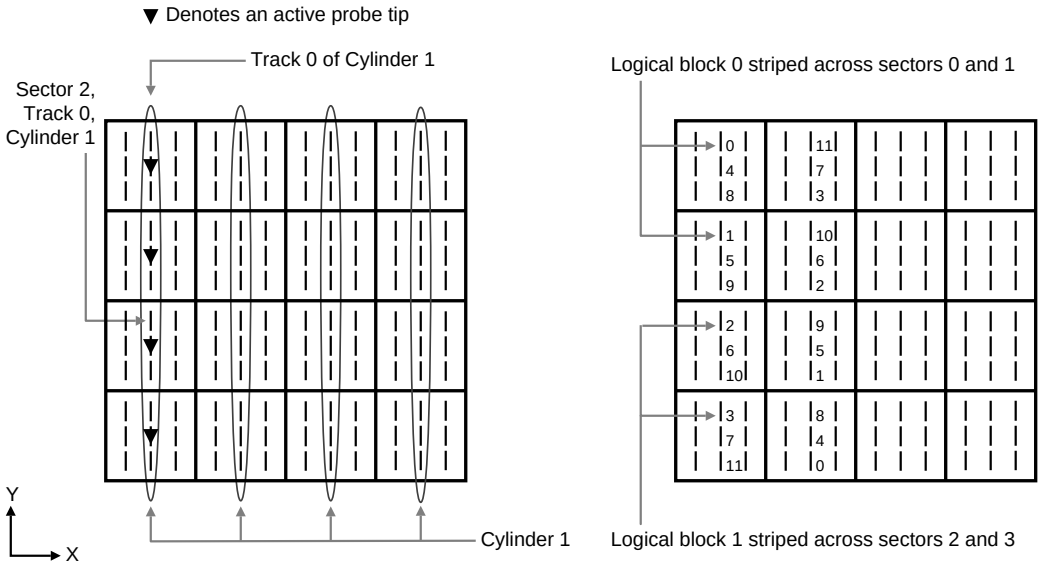


Fig. 2.4: **Cylinders, tracks, sectors, and logical blocks.** This example shows a MEMS-based storage device with 16 tips and M×N = 3×280. A *cylinder* is defined as all data at the same x offset within all regions; cylinder 1 is indicated by the four circled columns of bits. Each cylinder is divided into 4 *tracks* of 1080 bits, where each track is composed of four tips accessing 280 bits each. Each track is divided into 12 *sectors* of 80 bits each, with 10 bits of servo/tracking information between adjacent sectors and at the top and bottom of each track. (There are nine sectors in each tip region in this example.) Finally, sectors are grouped together in pairs to form *logical blocks* of 16 bytes each. Sequential sector and logical block numbering are shown on the right. These definitions are discussed in detail in Section 2.2.

therefore has four tracks. Track 0 of cylinder 1 is highlighted in the figure as the leftmost circled column of bits. Note again the parallel with disks, where a track consists of all bits within a cylinder accessible by a single active head. Again, in the G2 model, each sled has 6400 tips and 640 concurrently active tips, so each cylinder contains 10 tracks that each hold 135 KB of data. Excluding positioning time, accessing an entire track takes 3.64 ms.

Physical sectors. Continuing the disk analogy, tracks are divided into sectors. Instead of having each active tip read or write an entire vertical column of N bits, each tip accesses only 90 bits at a time—10 bits of servo/tracking information and 80 data bits (8 encoded data bytes). Each 80-data-bit group forms an 8-byte *sector*, which is the smallest unit of data that can be accessed by a single tip. Each track in Figure 2.4 contains 12 sectors (3 per tip). These sectors parallel the partitioning of disk tracks into physical sectors. As described below, physical sectors are combined together to form larger logical blocks. Physical sectors can be read in either the +Y or −Y direction, allowing MEMStores to support *bidirectional access*. In the G2 model, each track is composed of 34,560 sectors of 8 bytes each, of which up to 640 sectors can be accessed concurrently. Excluding positioning time, each 640 sector (5 KB) access takes 0.129 ms.

Logical blocks. The low data rate of individual tips and the desire to use powerful error-correcting codes over large blocks of data provide the motivation for combining multiple physical sectors into larger *logical blocks*. In the G2 model, 64 physical sectors are combined together to form 512 byte logical blocks. Each logical block is, in essence, striped across 64 tips. Given that the power budget allows 640 tips to be active together, 10 logical blocks can be accessed concurrently ($640 \div 64 = 10$). Because the error correcting codes require logical blocks to be read in their entirety, the set of tips required for each logical block must be static. The remaining logical blocks (e.g., 9 out of 10) can be dynamically chosen from the set that are addressed by the tips.

2.3 Media access characteristics

Media access requires constant sled velocity in the Y dimension and zero velocity in the X dimension. The Y dimension *access speed* is a design parameter and is determined by the per-tip read and write rates, the bit cell width, and the sled actuator force. Although read and write data rates could differ, tractable control logic is expected to dictate a single access velocity in early MEMStores. In the default model, the access speed is 28 mm/s and the corresponding per-tip data rate is 0.7 Mbit/s.

Positioning the sled for read or write involves several mechanical and electrical actions. To seek to a sector, the appropriate probe tips must be activated (to access the servo information and then the data), the sled must be positioned at the correct x,y displacement, and the sled must be moving at the correct velocity for access. Whenever the sled seeks in the X dimension—i.e., when the destination cylinder differs from the starting cylinder—extra *settling time* must be taken into account because the spring-sled system oscillates in X after each cylinder-to-cylinder seek. Because this oscillation is large enough to cause off-track interference, a closed loop settling phase is used to damp the oscillation. To the first order, this active damping is expected to require a constant amount of time. Although slightly longer settling times may ultimately be needed for writes, as is the case with disks, the model assumes that the settling time is the same for both read and write requests. Settling time is not a factor in Y dimension seeks because the oscillations in Y are subsumed by the large Y dimension access velocity and can be tolerated by the read/write channel.

As the sled moves away from zero displacement, the springs apply a restoring force toward the sled's rest position. These spring forces can either improve or degrade positioning time (by affecting the effective actuator force), depending on the sled displacement and direction of motion. This force is parameterized in the model by the *spring factor*—the ratio of the maximum spring force to the

maximum actuator force. A spring factor of 75% means that the springs pull toward the center with 75% of the maximum actuator force when the sled is at full displacement. The spring force decreases linearly to 0% as sled displacement approaches zero. The spring restoring force makes the acceleration of the sled a function of instantaneous sled position. In general, the spring forces tend to degrade the seek time of short seeks and improve the seek time of long seeks [Griffin et al. 2000].

Large transfers may require that data from multiple tracks or cylinders be accessed. To switch tracks during large transfers, the sled switches which tips are active and performs a *turnaround*, using the actuators to reverse the sled's velocity (e.g., from +28 mm/s to −28 mm/s). The turnaround time is expected to dominate any additional activity, such as the time to activate the next set of active tips, during both track and cylinder switches. One or two turnarounds are necessary for any seek in which the sled is moving in the wrong direction—away from the sector to be accessed—before or after the seek.

Lastly, a single chip may contain more than one media sled. Adding more sleds increases the per-device capacity and the number of independent actuators available to access data, possibly increasing performance for well-matched workloads.

2.4 Logical data layout

Sequential access is the most efficient access pattern in most mechanical storage devices, including MEMStores, because once the media is in motion the most efficient thing to do is to keep it in motion. The mapping of logical blocks (LBNs) onto physical sectors of a MEMStore will take advantage of this property. Data will be accessed in linear tracks (in columns along the Y axis), as shown in Figure 2.3, so successive logical blocks within these tracks will be numbered such that they are sequential.

Once the end of a track is reached, sequential LBNs will be mapped to the

0 33 54	1 34 55	2 35 56
3 30 57	4 31 58	5 32 59
6 27 60	7 28 61	8 29 62
15 36 69	16 37 70	17 38 71
12 39 66	13 40 67	14 41 68
9 42 63	10 43 64	11 44 65
18 51 72	19 52 73	20 53 74
21 48 75	22 49 76	23 50 77
24 45 78	25 46 79	26 47 80

Fig. 2.5: **Mapping LBNs to optimize sequential access.**

next track within the same cylinder. This means that accessing sequential tracks will require only that the device turn the media sled around and switch the set of read/write tips. No motion in the X dimension is required until the last track in the cylinder has been accessed. After that, the device will move the media sled to the next cylinder (requiring a single-cylinder seek in the X direction) and start again. MEMStores also use many read/write tips concurrently to access data in parallel. It is most natural to map sequential LBNs across these parallel tips in order to optimize sequential access.

Figure 2.5 shows how LBNs will be mapped to sequential locations on a simple MEMStore. This device has nine total read/write tips, of which three can be concurrently active due to the power budget. Each read/write tip addresses nine LBNs. Starting in the top left corner, LBNs 0, 1, and 2 are simultaneously accessible by three parallel read/write tips. As the media moves, LBNs 3–8 are accessed,

completing the first track of data. The second track (LBNs 9–17) are accessed by reversing the sled's motion and by activating the second row of read/write tips. Note that successive tracks are reversed with respect to each other — the first track is numbered "down" and the second is numbered "up." These *track reversals* are necessary so that the media is immediately positioned after a turnaround to access sequential data.

Lastly, each LBN will be striped over a number of individual read/write tips to improve bandwidth and fault tolerance. For example, in the default model used throughout this dissertation, each 512 byte sector is split into 64 physical sectors, which are spread over 64 concurrently-operating read/write tips. These physical sectors will be read in parallel and transparently combined in the device's buffers for delivery to the host. Once this striping is assumed, it is useful to consider that the number of read/write tips has been reduced and that each "virtual" read/write tip accesses a complete LBN at a time. For example, the default MEMStore described below has 6400 read/write tips and each LBN is spread over 64 tips. In this way, the MEMStore has a "virtual geometry" with only 100 read/write tips, each of which accesses a full block at a time. In this design, 640 physical read/write tips can be used concurrently, as determined by the power budget of the device, meaning that 10 "virtual" read/write tips can be used concurrently. In order to spread the heat load of the device and avoid "hot spots," physical read/write tips that are used together to access whole LBNs will be physically spread around the device.

2.5 Comparison to disks

Although MEMStores involve some radically different technologies from disks, they share enough fundamental similarity for a disk-like model to be a sensible starting point. This section compares MEMStores and disks from this standpoint, and the rest of the dissertation shows that little is lost by taking this view.

Like disks, MEMStores stream data at a high rate and suffer a substantial distance-dependent positioning time delay before each nonsequential access. In fact, although MEMStores are much faster, they have ratios of request throughput to data bandwidth similar to those of disks from the early 1990s. Some values of the ratio, γ, of request service rate (requests/s) to streaming bandwidth (MB/s) for some recent disks include $\gamma = 26$ (1989) for the CDC Wren-IV [Patterson et al. 1989], $\gamma = 17$ (1993) [Hennessy and Patterson 1995], and $\gamma = 5.2$ (1999) for the Quantum Atlas 10K [Quantum 1999]. γ for disks continue to drop over time as bandwidth improves at a greater rate than mechanical positioning times. In comparison, the MEMStore described below yields $\gamma = 25$ (1111 requests/s $\div$ 44.8 MB/s), comparable to disks within the last decade. Also, although many probe tips access the media in parallel, they are all limited to accessing the same relative x,y offset within a region at any given point in time—recall that the media sled moves freely while the probe tips remain relatively fixed. Thus, the probe tip parallelism provides greater data rates but not concurrent, independent accesses. There are alternative physical device designs that would support greater access concurrency and lower positioning times, but at substantial cost in capacity [Griffin et al. 2000].

The remainder of this section enumerates a number of relevant similarities and differences between MEMStores and conventional disk drives.

Mechanical positioning. Both disks and MEMStores have two main components of positioning time for each request: seek and rotation for disks, X and Y dimension seeks for MEMStores. The major difference is that the disk components are independent (i.e., desired sectors rotate past the read/write head periodically, independent of when seeks complete), whereas the two components are explicitly handled in parallel for MEMStores. As a result, total positioning time for MEMStores equals the greater of the X and Y seek times, making the lesser time irrelevant. This overlap most strongly affects request scheduling, which is discussed in Section 6.1.

Settling time. For both disks and MEMStores, it is necessary for read/write heads to settle over the desired track after a seek. Settling time for disks is a relatively small component of most seek times (0.5 ms of 1–15 ms seeks). However, settling time for MEMStores is expected to be a relatively substantial component of seek time (0.2 ms of 0.2–0.8 ms seeks). Because the settling time is generally constant, this has the effect of making seek times more constant, which in turn could reduce (but not eliminate) the benefit of both request scheduling and data placement.

Logical-to-physical mappings. As with disks, the lowest-level mapping of logical block numbers (LBNs) to physical locations will be straightforward and optimized for sequential access; this will be best for legacy systems that use these new devices as disk replacements. Such a sequentially optimized mapping scheme fits disk terminology and has some similar characteristics. Nonetheless, the physical differences will make data placement decisions (mapping of file or database blocks to LBNs) an interesting topic. Sections 6.2 and 6.3 discuss this issue.

Seek time vs. seek distance. For disks, seek times are relatively constant functions of the seek distance, independent of the start cylinder and direction of seek. Because of the spring restoring forces, this is not true of MEMStores. Short seeks near the edges take longer than they do near the center (as discussed in Section 6.2). Also, turnarounds near the edges take either less time or more, depending on the direction of sled motion. As a result, seek-reducing request scheduling algorithms [Worthington et al. 1994a] may not achieve their best performance if they look only at distances between LBNs as they do with disks.

Recording density. Some MEMStores use the same basic magnetic recording technologies as disks [Carley et al. 2000]. Thus, the same types of fabrication and grown media defects can be expected. However, because of the much higher bit densities of MEMStores, each such media defect will affect a much larger number of bits.

Numbers of mechanical components. MEMStores have many more distinct

mechanical parts than disks. Although their very small movements make them more robust than the large disk mechanics, the sheer number of parts makes it much more likely that some number of them will break. In fact, manufacturing yields may dictate that the devices operate with some number of broken mechanical components.

Concurrent read/write heads. Because it is difficult and expensive for drive manufacturers to enable parallel activity, most modern disk drives use only one read/write head at a time for data access. Even drives that do support parallel activity are limited to only 2–20 heads. On the other hand, MEMStores (with their per-tip actuation and control components) could theoretically use all of their probe tips concurrently. Even after power and heat considerations, hundreds or thousands of concurrently active probe tips is a realistic expectation. This parallelism increases media bandwidth and offers opportunities for improved reliability. Further, flexibility in the choice of which tips are used to access data allows for novel data access schemes, such as efficient access to two-dimensional data structures.

Control over mechanical movements. Unlike disks, which rotate at a constant velocity independent of ongoing accesses, the mechanical movements of MEMStores can be explicitly controlled. As a result, access patterns that suffer significantly from independent rotation can be better served. The best example of this is repeated access to the same block, as often occurs for synchronous metadata updates or read-modify-write sequences.

Startup activities. Like disks, MEMStores will require some time to ready themselves for media accesses when powered up. However, because of the size of their mechanical structures and their lack of rotation, the time and power required for startup will be much less than for disks. How this affects both energy conservation (Section 6.4) and availability (Section 7) is discussed below.

Drive-side management. As with disks, management functionality will be split between host operating systems and device firmware. Over the years, increasing

amounts of functionality have shifted into disk firmware, enabling a variety of portability, reliability, mobility, performance, and scalability enhancements. Similar trends are likely with MEMStores, whose silicon implementations offer the possibility of direct integration of storage with computational logic.

Speed-matching buffers. As with disks, MEMStores access the media as the sled moves past the probe tips at a fixed rate. Since this rate rarely matches that of the external interface, speed-matching buffers are important. Further, because sequential request streams are important aspects of many real systems, these speed-matching buffers will play an important role in prefetching and then caching of sequential LBNs. Also, most block reuse will be captured by larger host memory caches instead of in the device cache.

Sectors per track. Disk media is organized as a series of concentric circles, with outer circles having larger circumferences than inner circles. This fact led disk manufacturers to use banded (zoned) recording in place of a constant bits-per-track scheme in order to increase storage density and bandwidth. For example, banded recording results in a 3:2 ratio between the number of sectors on the outermost (334 sectors) and innermost (229 sectors) tracks on the Quantum Atlas 10K drive [Ganger and Schindler 2004]. Because MEMStores organize their media in fixed-size columns instead, there is no length difference between tracks and banded recording is not relevant. Therefore, block layout techniques that try to exploit banded recording will not provide benefit for these devices. On the other hand, for block layouts that try to consider track boundaries and block offsets within tracks, this uniformity (which was common in disks 10 or more years ago) will simplify or enable correct implementations. The subregioned layout described in Section 6.2 is an example of such a layout.

2.6 Other alternative technologies

2.6.1 Battery-backed DRAM

One of the simplest methods of making memory "non-volatile" is to make sure it can be powered with batteries in case main power is removed. This strategy is widely used in today's disk arrays which back up power to their large (several gigabyte) DRAM-based caches with large batteries. The main concern, of course, is that there is enough battery power in the system to allow all of the dirty data to be de-staged to truly non-volatile storage (i.e., the back-end disk drives) in the event of power loss. The power requirements of the system are significant, and the batteries must be large enough to supply both the DRAM itself and the back-end storage to which the data is to be retired. The main benefit of battery-backed DRAM, of course, is its superior performance. However, its lower density compared to disk drives makes it prohibitively expensive as a true mass storage device, except for very high-performance systems like high-end disk arrays.

In some sense, the DRAM in some portable devices, such as PDAs, is "non-volatile" since the device is almost always powered by batteries. Often PDAs use this DRAM to store at least some of their files, with the rest being stored in other truly non-volatile storage like FLASH memory.

2.6.2 Miniature disk drives

In just the last few years, portable music players such as the Apple iPod have created a large demand for high-density portable storage. To meet this demand, hard drive companies have introduced a plethora of new, miniature disk drives, trading off performance for very small form factors. IBM first introduced its 1.0 inch Microdrive in the late 1990s with a capacity of 340 MB. The Microdrive was followed by 1.8, 1.5, and 0.85 inch drives from Toshiba, Hitachi, Cornice, and others. These drives are, essentially, scaled-down versions of desktop and notebook drives. They contain only a single platter and often use only a single head to access data. Be-

cause they are scaled down so significantly, their performance is much worse than their desktop counterparts. Rotation speed is usually no faster than 3600 RPM, their average seek time is generally more than 10 ms, and their bandwidth is around 5 MB/s.

While these disks may seem very limited, they fit their market well. Customers who use portable music players demand the largest possible capacities because there is always more music to carry around. Performance is not too critical, since the workload is very simple and consists only of streaming large music files to and from the disk through a RAM buffer. Once a playlist is read into the memory buffer, the disk is idled to save power.

Miniature disk drives are a recent addition to the storage landscape, and they present a strong challenger to MEMStores in that their per-device capacity is significantly greater. MEMStores are envisioned to store at most 5-10 GB per chip, and today's miniature disk drives store 40 GB. Assuming that in five years when MEMStores are available the capacity of a miniature disk drive will be 100 GB, fitting 10 MEMStore chips into a Compact FLASH form factor to equalize capacities may be a challenge. However, the higher performance of MEMStores in seek latency, bandwidth, and energy consumption alone could give them an advantage over miniature disk drives.

2.6.3 FLASH

Along with miniature disk drives, FLASH memory is the current non-volatile storage media of choice for mobile devices such as digital cameras, PDAs, cellular telephones, and portable music players. FLASH is a semiconductor memory, and so has a much lower density and faster performance than disk drives or MEMStores. Its performance for reads is slightly slower than that of DRAM, but its write performance is much slower. Internally, the FLASH memory can only write data in large (e.g., 128 KB) pages so for small writes, the entire page must be read into a buffer, modified, the page in the FLASH must erased, and then the modified data

is programmed from the buffer. This cycle can take on the order of a second for an entire page, making writes very expensive. Also, FLASH memories can only be written a fixed, relatively small, number of times (e.g., 100,000), after which they become inoperable. Newer FLASH memories mitigate this problem by internally remapping data pages from cells that are approaching their re-write limit.

Because of its poor write performance, FLASH is not well-suited for general filesystem workloads, in which small write performance is crucial for maintaining metadata. However, FLASH memory is well-suited for simple file storage in digital cameras and portable music players. In these applications, though, the lower density of FLASH compared to disks will keep its capacity below several gigabytes at reasonable costs and sizes. In response to the growing popularity of miniature disk drives, FLASH has shown some tremendous growth recently by incorporating some new innovations such as storing multiple memory states per cell. For the foreseeable future, FLASH will probably dominate the market for low- to medium-capacity devices (128 MB to 2 GB) and miniature disk drives will provide high capacities (10 GB to 100 GB).

2.6.4 MRAM

Magnetic RAM (MRAM) is another emerging non-volatile storage technology that seeks to supplant FLASH. It employs GMR elements into semiconductor memory cells to store data. MRAM will, most likely, have DRAM-like access times, both for read and write, and will not suffer from the re-write limits of FLASH memory. Some MRAM components are available at very high costs and low densities, and many researchers and companies are working to make it a commodity product. The non-volatility and performance properties of MRAM make it very interesting as a FLASH and even main memory replacement. However, like other semiconductor memories, its architecture makes it inherently less dense than mechanically-addressed storage devices like disks and MEMStores, making it an unlikely alternative for applications that require high capacities.

2.6.5 Ovonic Unified Memory

Ovonic memory is a new technology being developed by Ovonyx, Inc. [Ovonyx 2004] that incorporates phase change media into semiconductor memories. It achieves a similar density to FLASH memory, but does not share several of FLASH's limitations, notably its poor write performance and re-write limitations. Again, because it is a semiconductor memory, Ovonic Unified Memory will not approach the density of mechanical storage because its density is determined by lithographic feature sizes.

2.6.6 FERAM

Ferroelectric RAM (FERAM) is another alternative semiconductor memory technology that uses ferroelectric capacitors as the memory elements [Sheikholeslami and Gulak 2000]. Its density is limited by lithography, like any semiconductor memory, and it avoids the poor write performance and re-write limitations of FLASH memory. However, some designs may suffer from destructive reads, which would require cells to be refreshed immediately after reads.

2.7 Related work

2.7.1 Devices

Fortunately, the design of MEMStores has not all taken place behind the closed doors of corporations and research labs—some of the devices have been described in the literature.

The MEMStore design being developed at Carnegie Mellon University was first described in [Carley et al. 2000]. That paper described the basic architecture of the device and compared it to several other devices being developed concurrently. Several other papers from that group describe the servo system for tip/media spacing [Carley et al. 2001], the media actuator [Alfaro and Fedder 2002], and a potential magnetic recording scheme [El-Sayed and Carley 2002; 2003].

The IBM Millipede project has produced several papers which describe several of the components of that device. Two papers describe the overall device [Vettiger et al. 2000; Vettiger et al. 2002] and its basic architecture. The thermomechanical writing process was first described in [Mamin and Rugar 1992], and was further studied in [Mamin et al. 1995; Mamin et al. 1999]. One of the concerns of thermomechanical writing and reading of data has always been wear of both the media and the read/write tips, which was first addressed in [Terris et al. 1998]. Other papers describe methods to manufacture probe tips [Ried et al. 1997] and the media actuator [Lutwyche et al. 1999; Lutwyche et al. 1999; Rothuizen et al. 2000].

The electrostatic stepper motor used by the device under development at Hewlett-Packard was described in [Hoen et al. 1997], but little else has been published about the device.

2.7.2 Parameter sensitivity

Since MEMStores are still being developed, systems researchers with knowledge of how they may be used can influence their design. This was the focus of some early of our early modeling work and also of a group at the University of California at Santa Cruz.

Madhyastha and Yang [Madhyastha and Yang 2001] developed a software model similar to the one that is used in this dissertation and in [Griffin et al. 2000; Schlosser et al. 2000]. Its seek model is based on an open-loop controller, rather than the closed-loop controller that I assume. An open-loop system uses the natural damping of the system to eliminate oscillations, while a closed-loop system actively damps oscillations, leading to faster seek times. Their model is more accurate in that it models second-order effects that I only approximate, but it is more likely that real MEMStores will use closed-loop controllers. They describe two alternative seek models: the *spring model* and the *optimal control model*. In the spring model, the actuators apply a single constant force that drives the media sled to equilibrium at the destination point, waiting for the natural damping of

the system to eliminate oscillations. The optimal control model is similar to the model that I use in that the actuators apply a force to move the sled toward the destination, and then a counterforce to stop it. However, with an accurate model of the system dynamics, they are able to choose the optimal point at which to switch actuator direction. The spring model provides an upper bound on seek time, and the optimal control model provides a lower bound. The model that I use approximates the optimal control model, but does not precisely model the second-order effects. I compare these models below in Section 3.4, and find that they differ by, at most, 55 μs.

Sivan-Zimet used a simplification of Madhyastha and Yang's model to study the sensitivity of service time to the many configurable parameters of a MEMStore [Sivan-Zimet and Madhyastha 2002]. The goal was to find an optimal device configuration for a number of traced filesystem workloads, minimizing service time. Their simplified model does not include any settle time and so they did not observe the settle time sensitivity issues that I describe in Chapter 3. They do observe, however, that longer ranges of motion in the Y dimension lead to better performance because more data can be accessed before the sled must change direction.

Dramaliev used another analytic approximation of Yang's model to refine the conclusions reached by Sivan-Zimet [Dramaliev and Madhyastha 2003]. This model does include settle time for X and closely approximates Yang's results. However, it makes the simplifying assumption that requests are uniformly distributed across the device. The result is a predictive model of average performance based on a given device configuration, allowing quick evaluations of the configuration space.

2.7.3 Roles

Several researchers have studied various roles that MEMStores may take in computer systems in addition to the roles presented in this dissertation. We presented the first work in studying roles for MEMStores in 2000 [Griffin et al. 2000; Schlosser et al. 2000], showing the performance of various application workloads

using MEMStores as a simple disk replacement and as a cache for disks.

Hong evaluated the use of MEMStores as a metadata cache, improving response time by 28–46% for user workloads [Hong 2002]. He also used the MEMStore as a write cache for the disk, leading to further improvements in performance. Hong [Hong and Brandt 2002] also developed yet another analytic model of seek time for this work and to study MEMStore-specific scheduling policies. I compare this analytic seek model to the model that I use in Section 3.4.

Rangaswami et al. proposed using MEMStores in streaming media servers as buffers between the disks and DRAM [Rangaswami et al. 2003]. They adapted caching and scheduling policies for streaming media servers using disk arrays to include the faster MEMStore.

Uysal et al. evaluated the use of MEMStores as intermediate storage in disk arrays [Uysal et al. 2003] under synthetic workloads and file system traces of various systems. They evaluated several architectures, including replacing all disks with MEMStores, using MEMStores as mirrors of disks, and several hybrid architectures. They also varied the relative cost of the MEMStores and disks used in the system, since cost remains an unknown until MEMStores are available.

None of these studies claimed to use any feature of the MEMStores other than the fact that they are faster than disk drives. Indeed, system performance was increased by using faster devices, but was not necessarily dependent on the fact that those faster devices were MEMStores. These roles fail the specificity test introduced in Section 1, since they could be filled as well by a hypothetical disk drive that is as fast as a MEMStore.

2.7.4 Policies

Various policies for tailoring access to MEMStores beyond those described in this dissertation have been suggested in the literature, including MEMStore-specific request scheduling algorithms, energy conservation strategies, and data layouts. We compared existing disk-based request schedulers, MEMStore-specific data lay-

outs, and energy conservation policies in [Griffin et al. 2000]. These policies are described in more detail in Chapter 6.

A new scheduling algorithm, zone-based shortest-positioning time-first, was suggested for scheduling requests to a MEMStore [Hong et al. 2003]. ZSPTF is a combination of SPTF and circular scan (C-SCAN) scheduling intended to reduce the starvation characteristics of SPTF. Yu et al. suggested another scheduling policy based on servicing requests in minimum-spanning-tree order, with their results showing performance similar to SPTF scheduling [Yu et al. 2002]. However, it is not clear that either of these scheduling policies uses any device-specific aspects of MEMStores. Both algorithms could be applied to disk drives just as effectively.

Lin et al. studied three methods of reducing MEMStore power consumption [Lin et al. 2002]. First, they used a MEMStore's ability to transition quickly between active and inactive modes, saving power when idle. Second, they coalesced sequential requests that could be serviced in parallel. And third, they allowed requests smaller than the standard logical block size of 512 bytes, only turning on those read/write tips that were necessary to transfer the data. In addition to the energy savings that these techniques afforded, they quantified their performance impact and showed that it was minimal. This extended our initial work which used only the first of the three methods [Schlosser et al. 2000]. It is not clear that the third of these methods is actually possible because error-correcting codes require that entire logical blocks be read in their entirety.

Yu et al. also described storing tabular data such as databases on MEMStores and accessing that data in both row- and column-major orders [Yu et al. 2003]. While this concurrent work is similar to that which I describe in Section 6.3 (and in [Schlosser et al. 2003]), it does not account for device-level issues (e.g., striping and ECC), and it lacks a general method to describe available parallelism to applications. The same researchers also described a more general technique for declustering two-dimensional data structures on MEMStores [Yu et al. 2004], and showed that it achieves an optimal result that is impossible with a disk drive.

3 Performance modeling of MEMStores

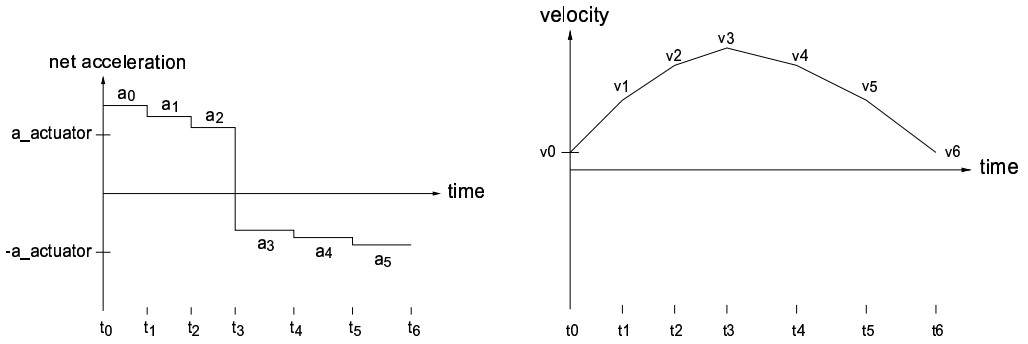
Since complete MEMStores are not currently available, we must depend on modeling to study them. Engineers working on the MEMS components themselves do their own modeling at very low levels; i.e., micromagnetic modeling of the read/write process or finite element analysis of the mechanical components. These models are much more detailed than are needed at the system level. This chapter describes the simplified models used in this work, and how they are used in various simulation systems.

3.1 Piecewise-linear seek model

When developing a performance model for MEMStores, it is useful to first look at a common disk performance model. The service times for a disk access is often computed as:

$$time_{service} = time_{seek} + latency_{rotate} + time_{transfer}$$

The seek time, $time_{seek}$, is a function of the distance in cylinders that the disk arm must travel. This includes an acceleration/deceleration component, a linear component (representing the maximum velocity of the seek arm) for long seeks, and a significant disk arm settling delay (approximately 1 ms) for all non-zero length seeks. The rotational latency, $latency_{rotate}$, can be computed by dividing the angular distance between the current and destination sector by the rotational



(a) Sled acceleration versus time (b) Sled velocity versus time

Fig. 3.1: **Piecewise-constant approximation of acceleration and velocity during a Y-dimension seek.** The graph in (a) is the derivative of (b) with respect to time. $a_{actuator}$ is the sled acceleration caused by the actuator force; the net accelerations during each "chunk" are different because of the effects of the spring restoring force. $v_o = v_6 = v_{access}$; in other words, at the end of a seek the sled is traveling at the correct access velocity. In the case of an X seek (not shown), $v_o = 0$. In this example, each phase of the seek is divided into 3 chunks per phase; our model divides each seek into 8 chunks per phase.

velocity. Since disks rotate continuously, detailed simulation requires accounting for all advances in time, including the seek time for the access being serviced. The media transfer time, $time_{transfer}$, can be computed as the product of the number of sectors accessed divided by the number of sectors per track (in the relevant zone) and the time for a full revolution. Detailed models must also account for all track and cylinder boundaries crossed by the range of desired sectors, since each crossed boundary adds a repositioning delay equal to the corresponding skews in the logical-to-physical mapping.

Service times for MEMStores can be modeled with a similar equation:

$$time_{service} = time_{seek} + time_{transfer} \tag{3.1}$$

The obvious difference is the absence of rotational latency. Less obvious from the equation is the more complicated nature of the $time_{seek}$ term. Recall that the movable media sled must seek to the correct <*x*,*y*> position and attain the proper media access velocity in the proper Y direction. The actuation mechanisms and control loops for X and Y positioning are independent, allowing the two to proceed

in parallel. Thus,

$$time_{seek} = max(time_{seek\_x}, time_{seek\_y})$$

Computing $time_{seek\_x}$ and $time_{seek\_y}$. Since the sled is a mass moving under a constant force from the actuators, equations from classical first-order mechanics (*e.g.*, $\Delta x = v_0 t + \frac{1}{2}at^2$) can be used to compute both $time_{seek\_x}$ and $time_{seek\_y}$. A seek is broken into two *phases*: acceleration and deceleration. In the acceleration phase, the actuators pull the sled toward the destination. In the deceleration phase, the actuators reverse polarity and decelerate the sled to its final destination and velocity. In addition to the actuator force, the sled springs constantly pull the sled towards its centermost position. The spring force in each dimension is linear with respect to the sled's displacement (from center) in that dimension, which means that spring force varies as the sled moves.

A piecewise-constant approximation determines the spring force's contribution to net acceleration. Each phase of the seek is broken into a set of smaller *chunks*, with the net acceleration in each chunk being the sum of the acceleration due to the actuators and the average acceleration due to the springs. As an example, the acceleration curve for a sled seeking from the outermost position to the centermost position is shown in Figure 3.1(a). This acceleration curve leads to the velocity curve shown in Figure 3.1(b). In this example, the springs help during the acceleration phase ($t_0 ... t_3$), but hurt during the deceleration phase ($t_3 ... t_6$). Also, because this example seek moves toward the centermost position, the spring's impact decreases in each chunk as the sled approaches its rest position.

To parameterize the model, the spring force at full displacement is set to a percentage (called $spring\_factor$) of the actuator force. Generally speaking, the spring factor should be a large percentage of the actuator forces since for manufacturability reasons the springs should be as stiff as possible. So, when the sled is at its full displacement, the springs should push back against the actuators with

an almost equal force, yielding a high $spring\_factor$.

An expression for the net acceleration at any point x is:

$$a(x) = a_{actuator} \pm \left[(a_{actuator} * spring\_factor) * \frac{offset(x)}{max\_offset} \right]$$

When the actuator is pulling against the springs, the second term will be negative. For each chunk, the constant net acceleration is taken to be the average of the net accelerations at its endpoints:

$$a_i = \frac{a(x_i) + a(x_{i+1})}{2}.$$

Given these constant accelerations, we can compute the velocity of the sled at the end of each chunk:

$$v_i = v_{i-1} + a_{i-1}(t_i - t_{i-1}). \tag{3.2}$$

Since the initial position x_0, the initial velocity v_0, and the acceleration during each chunk are all known, the times at the end of each chunk can be computed. To do this, we integrate the velocity curve v_i to find an expression for position x_i:

$$x_i = x_{i-1} + v_{i-1}(t_i - t_{i-1}) + \frac{1}{2}(v_i - v_{i-1})(t_i - t_{i-1}). \tag{3.3}$$

Plugging Equation 3.2 into Equation 3.3 yields a quadratic that can be solved for t_i, the time that the sled arrives at the end of chunk i:

$$t_i = \frac{-(v_{i-1} - a_i t_{i-1}) + \sqrt{v_{i-1}^2 + 2a_i(x_i - x_{i-1})}}{a_i} \tag{3.4}$$

Extra settling time for $time_{seek\_x}$. Equation 3.4 describes the base seek time for both the X and Y dimensions. In the X dimension, the sled starts and ends each seek at rest ($v_0 = 0$). Extra settling time, t_{settle}, must be added onto X-dimension seeks to model the time required for the oscillations of the sled-spring system to

damp out. t_{settle} is dependent on the resonant frequency of the system, f, which depends on the construction of the sled and the stiffness of the springs.

$$time_{settle} = \frac{1}{2\pi f} * number_{timeconstants} \tag{3.5}$$

where $number_{timeconstants}$ is a measure of how much damping is needed before the probe tips can begin to robustly access the media. This oscillation could be damped by the sled-spring system itself or by the atmosphere. More likely, the system will have a closed-loop control system that actively damps the oscillations using the actuators. Active damping has the effect of reducing $number_{timeconstants}$ and therefore $time_{settle}$.

Extra turnaround times for $time_{seek\_y}$. Y-dimension seeks, for which the final velocity is the access velocity rather than zero, are not expected to require extra settling time. However, since the media sled may be moving in the wrong direction before the seek and/or after the seek, it may be necessary to reverse the sled's direction once or twice. For each such turnaround:

$$time_{turnaround} = 2 * \frac{v_{access}}{a(x)} \tag{3.6}$$

Computing $time_{transfer}$. The $time_{transfer}$ component of the MEMStore service time differs from that of conventional disks in two ways. First, the time to transfer a single sector is the product of the number of tips over which each sector is striped, the rate at which bits are read ($v_{access} * width_{bit}$), and the percentage of bits read that are actual data (e.g., rather than servo and ECC). Second, the time to transfer a range of sectors must take into account the fact that multiple sectors can be accessed in parallel; the number of sectors accessed in parallel is the number of concurrently active tips divided by the number of tips per sector. As with conventional disks, when a range of sectors to be transfered crosses a track or cylinder boundary, a track or cylinder switch is required. The sequential track switch time is equal to the minimum turnaround time, since switching the active

	G1	**G2**	**G3**
bit width (nm)	50	40	30
sled acceleration (g)	70	82	105
access speed (kbit/s)	400	700	1000
X settling time (ms)	0.431	0.215	0.144
total tips	6400	6400	6400
active tips	640	640	1280
max throughput (MB/s)	25.6	44.8	128
number of sleds	1	1	1
per-sled capacity (GB)	2.56	4.00	7.11
bidirectional access	no	yes	yes

Table 3.1: **Three generations of MEMStore parameters.** The G2 design point is used for most of the results in this dissertation.

tips is expected to take less than this time. The sequential cylinder switch time can be computed as a single cylinder seek, but optimizations of the control loop can be expected to reduce this time to the minimum turnaround time by taking advantage of the tips' ability to deflect small distances in the X dimension.

3.2 Baseline device parameters

Given the wide range of parameters, exploring the entire design space of MEMStores is not feasible. Instead, I use three MEMStore design points, based on anticipated technology advances over the first three generations (Table 3.1).

The "1st generation (G1)" model represents a conservative initial MEMStore, which could be fabricated within the next few years [Carley et al. 2000]. The sled has a full range of motion of 100 μm along the X and Y axes, and the actuators accelerate the sled at $70g$. To access data, the device uses a relatively primitive recording scheme, leading to a per-tip data rate of 400 Kbit/s. This design only supports unidirectional accesses, where reads and writes only occur when the sled moves in the positive Y direction.

G1's media, tip resolution, and sled positioning system provide a square bit cell of 50 nm such that each tip addresses a 2000×2000 array of bits. The sled

footprint is 0.64 cm^2 allowing 6400 tips for each sled. This yields a raw capacity of 2.56 GB per sled. However, media error management requires a 10-bit-per-byte encoding. Also, sled tracking and synchronization information requires 10 tracking bits for every 80 data bits. During media access, the sled is restricted to a fixed access velocity. However, the sled speed is not limited during seeks.

The "2nd Generation (G2)" model. Several fundamental improvements enhance G2 over G1. First, media access occurs in both the +Y and −Y directions. Second, per-tip data rate increases to 700 Kbit/s based on trends in probe tip technology. A decrease in the sled mass and an increase in the actuator voltage leads to an increase in sled acceleration to $82g$. Also, improvement in the servo system reduces the settling time for each X seek. Finally, media material improvements increase G2's bit density by 20%.

The "3rd Generation (G3)" model. G3 approaches the high-end of many MEMStore parameters and characteristics. Here the bit density scales down to 30 nm per bit, and a decrease in the sled mass leads to higher sled acceleration. In this case a change in the suspension and sled design leads to a higher resonant frequency, resulting in a shorter X settling time. Throughput is increased, largely because of the addition of more active tips.

3.3 Basic seek performance

Figures 3.2 and 3.3 show the seek time as a function of both X and Y displacement from the corner and the center of a media square, respectively. Both show results for the G2 design point described above. The effect of X dimension settling time is very clearly shown in Figure 3.3. The overall seek time, which is the greater of the two seek times in X and Y, is strongly correlated to the X displacement only, with almost no dependence on Y displacement.

Table 3.2 shows the performance of the G2 MEMStore under a workload of 10,000 random requests. The requests were distributed uniformly across the ca-

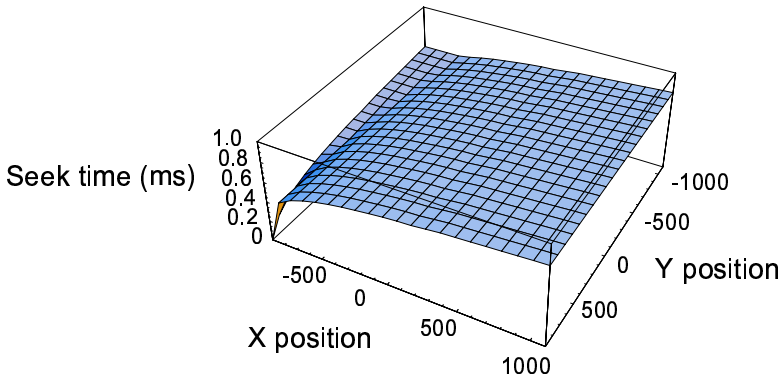


Fig. 3.2: **Seek time profile from corner of media.** This graph shows the seek time for a G2 MEMStore from a corner of a media square as a function of both X and Y displacement. It was generated directly from the seek time equations.

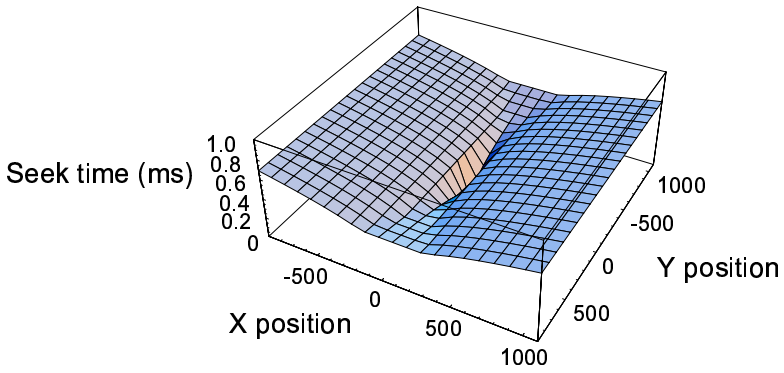


Fig. 3.3: **Seek time profile from center of media.** The seek time of a MEMStore is largely uncorrelated with the displacement in the Y dimension due to a large settling time required for the X dimension seek that is not required for the Y dimension seek. The overall seek time is the maximum of the two independent seek times. This graph shows the seek time for a G2 MEMStore.

Average service time	0.91 ms (0.20)
Maximum service time	2.15 ms
Average seek time	0.57 ms (0.11)
Maximum seek time	0.78 ms
Average X seek time	0.57 ms (0.11)
Maximum X seek time	0.78 ms
Average Y seek time	0.36 ms (0.13)
Maximum Y seek time	0.75 ms
Settling time	0.22 ms
Average per-request turnaround time	0.07 ms (0.06)
Maximum per-request turnaround time	0.50 ms

Table 3.2: **Basic G2 MEMStore performance characteristics.** These numbers are based on a random workload of 10,000 requests. Standard deviations are provided in parentheses.

pacity of the device, with the inter-arrival time chosen from an exponential distribution with a mean of 50 ms. The size of the requests was also drawn from an exponential distribution, with a mean of 4 KB. Two thirds of the requests were reads and one third were writes. Again, the significance of the X dimension settling time is evident in that the average seek time (0.57 ms) is equal to the average X seek time (0.57 ms), which is greater than the average Y seek time (0.36 ms). Average per-request turnaround time is determined both by the number of times the sled must turn around before a request is serviced, and the number of times it must turn around during a transfer because the request spans more than one track.

3.4 Spring-mass-damper seek model

The media sled is, in reality, a damped oscillator, the positioning time for which can be found using a general expression. The piecewise-linear model is a simplification of that solution, which was more tractable to use in practice. This section compares the results of the piecewise-linear model to those of a more general solution used by Hong [Hong et al. 2003].

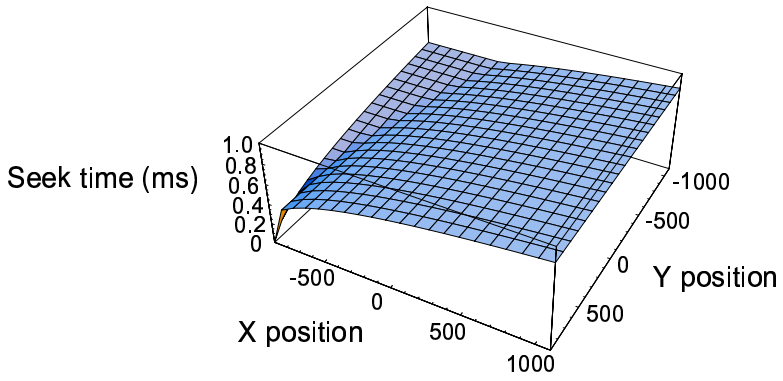


Fig. 3.4: **Seek time profile of G2 MEMStore from corner of media for Hong's model.** This graph is equivalent to that in Figure 3.2.

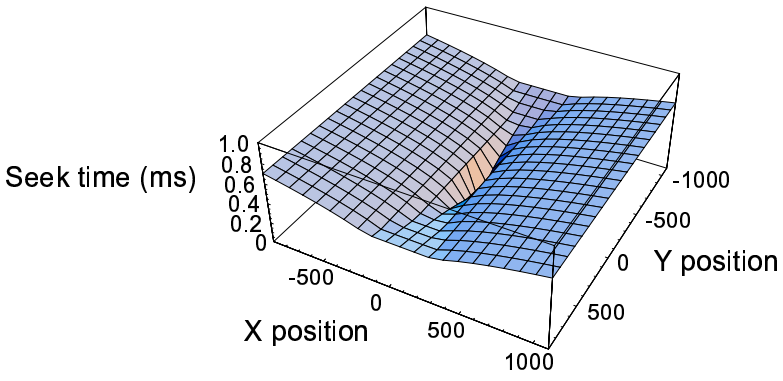


Fig. 3.5: **Seek time profile of G2 MEMStore from center of media for Hong's model.** This graph is equivalent to that in Figure 3.3.

As seen in Figures 3.4 and 3.5, the seek profiles of the G2 MEMStore using Hong's model are virtually identical to those of the piecewise-linear model, shown in Figures 3.2 and 3.3. Again using the G2 design point, seek times in the two models differ, on average, by 29 μs for the seeks shown in Figure 3.4, and only 7 μs for the seeks shown in Figure 3.5. The maximum difference for both sets of seeks was 55 μs. As a percentage, the largest difference in seek time was 9.6%, on average. Therefore, using the piecewise-linear simplification to the general solution does not affect model accuracy appreciably.

3.5 DiskSim

The model described above has been incorporated into a complete storage system simulator called DiskSim [DiskSim 2004]. Both the piecewise-linear and spring-mass-damper seek models have been implemented, along with caching, scheduling, and data transfer functionalities. DiskSim was originally written to accurately model disk drives. Adding the MEMStore functionality allows easy comparisons to disk drives to be made. DiskSim can be exercised with various workloads such as disk access traces and synthetic workloads. It can also be driven externally by system simulators such as SimOS [Rosenblum et al. 1995]. Most of the results in this dissertation were generated using DiskSim configured as a MEMStore.

3.6 Parameter sensitivity

To understand which device characteristics are important to performance, I explored the model's performance sensitivity to several different model parameters. This section describes the most interesting results.

Sensitivity to per-tip data rate. Overall bandwidth to and from the media is determined by the number of simultaneously active tips and the per-tip data rate. Like conventional disks, MEMStores must switch tracks (or cylinders) when media transfers cross track boundaries. Unlike conventional disks, for which rotation

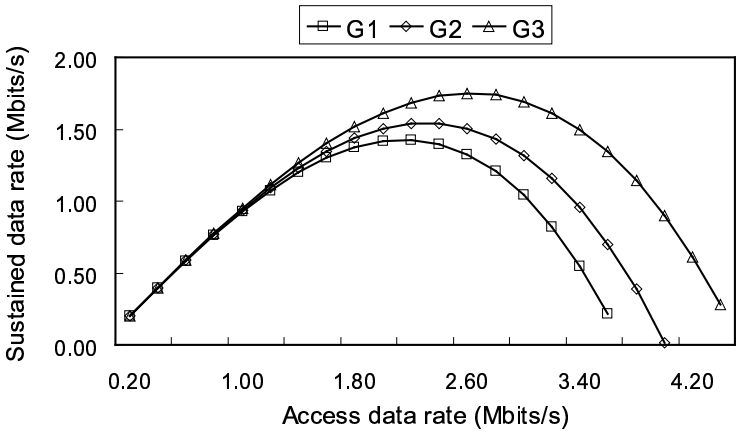


Fig. 3.6: **Sensitivity of MEMS-based storage device performance to the access velocity.** The three MEMStore design points (G1, G2, and G3) are shown, with each having a different value for actuator acceleration. The maximum point for each acceleration value represents a balance between the benefit of higher data rates and the increased time required to turn around for track and cylinder switches.

speed is independent of seek arm positioning, the time required for MEMStores to switch tracks depends directly upon the access velocity (Equation 3.6). Specifically, because of their Cartesian nature, MEMStores turn around each time a media transfer crosses a track boundary. Reversing direction requires decelerating, changing direction, and re-accelerating to the access velocity. As the access velocity increases, this turnaround time increases. Therefore, one should expect diminishing returns from increasing per-tip data rate while keeping other parameters constant. Figure 3.6 shows the sustained bandwidth of a single tip given increasing per-tip data rates. The result changes based on the MEMStore model used because each generation has improved actuator force, leading to higher acceleration. For each design point, there is a maximum data rate after which turnaround times dominate transfer rates. This is an important result because it indicates that the recording head and channel need not handle ever-higher data rates, making them simpler to manufacture and less power-hungry. Further, this result suggests that efforts may be better spent on improvement of other design characteristics.

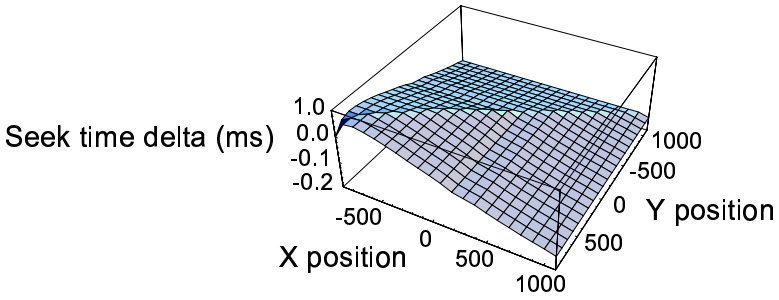


Fig. 3.7: **Delta in seek times from <-1000,1000> given a spring factor of 75% (compared to 0%) using a G2 MEMStore.** Short seeks are made slightly longer and long seeks are shorter.

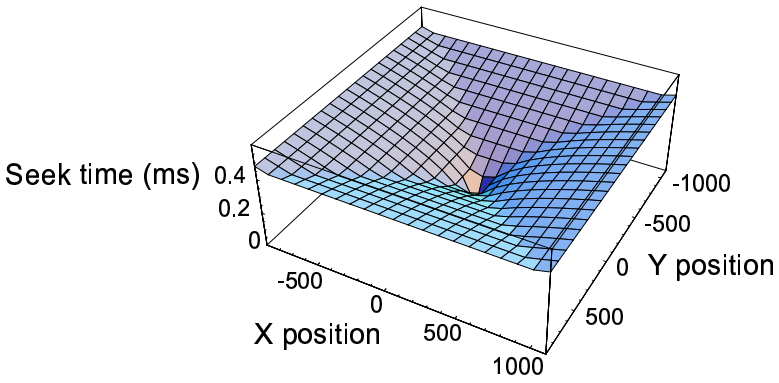


Fig. 3.8: **Seek times for the G2 MEMStore when no settling time is required for X-dimension seeks.** Without settling time delays, Y-dimension seeks become a more significant component of overall seek times.

Sensitivity to settling time. Whenever the sled moves in the X-dimension, some time is required to damp the sled's oscillations, as described above. This settling time is based on the system's resonant frequency and the ability of the control system to damp out the motion. I model this by computing a settling time constant (Equation 3.5) and adding this to the X seek time. The number of settling time constants added can be varied to allow for improved control systems. The G2 MEMStore described in Table 3.2 adds one time constant of 0.22 ms. Figure 3.8 shows the effect of eliminating this settling time. It shows the result of the same experiment as shown in Figure 3.3 without the settling time in X.

Rather than uniformly decreasing seek times by 0.22 ms, overall seek times are much more dependent on Y-dimension seeks, making the seek profile match better with expectations for a two-dimensional movement.

Sensitivity to spring forces. The effect of springs on seek time is shown in Figure 3.7. This graph shows the same set of seeks as Figure 3.2, but in this case we only see the differences (delta) in seek times caused by the spring forces. The net effect of adding the spring forces is to lengthen the time for short seeks and to shorten the time for long seeks. The intuition behind this result is fairly straightforward. Consider a $spring\_factor$ value of 50%, meaning that the springs push back with 50% of the actuator force when the sled is at full displacement. If the actuators are pulling the sled towards the center, then the net force on the sled is 150% of the actuator force. If the actuators are pulling against the springs, then the net force is only 50% of the actuator force. Thus, at a given displacement, the impact of the springs is greater when they hurt than when they help. During a short seek, the displacement remains relatively constant throughout the seek, and so the springs will hurt one phase of the seek more than it helps the other. During long seeks, the displacement changes significantly. As a result, the springs tend to help noticeably in one of the two phases and be either less significant or also helpful in the other. Therefore, long seeks are generally helped by the springs.

The springs' effect on turnaround times are similar to those for short seeks. Figure 3.9 shows turnaround times with and without springs for each displacement, assuming that the sled is moving at the constant access velocity in the positive direction. Superimposed on the graph is the constant turnaround time that results from a spring factor of 0%. In the left half of the graph, the springs act against the actuators during the turnaround. In the right half, they help. As with short seeks, the impact of the springs is more significant when they hurt than when they help.

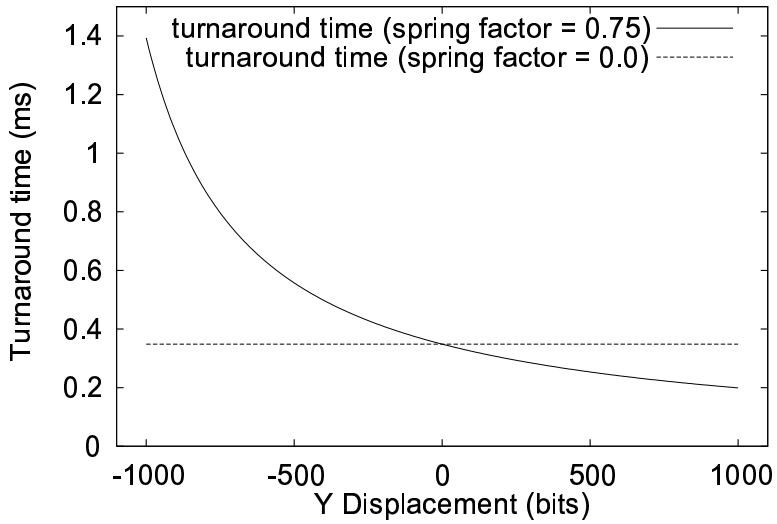


Fig. 3.9: **The effect of springs on turnaround time for a G1 MEMStore.** This figure shows the turnaround time at each displacement from center given that the sled is moving at the access velocity in the positive direction. Therefore, the springs hurt the turnaround time for the negative displacements and help in the positive.

3.7 Summary

Without complete MEMStores to test and characterize, we must rely on software models to understand their behavior. This chapter has described the model that I developed to study MEMStores. It described the theoretical background for the model, its implementation, the parameters I have used to compare MEMStores to other storage devices throughout the dissertation, and the model's sensitivity to changes of those parameters. It also compared the capabilities of this model to those of an alternative model based on the dynamics of the spring/mass/damper system that is a MEMStore and showed that my simplification gives nearly the same results.

The next chapter examines the use of standard storage abstractions for disk drives and discusses how these abstractions will work for MEMStores as well.

4 Storage abstractions

High-level storage interfaces (e.g., SCSI and ATA) hide the complexities of mechanical storage devices from the systems that use them, allowing them to be used in a standard, straightforward fashion. Different devices with the same interface can be used without the system needing to change. Also, the system does not need to manage the low-level details of the storage device. Such interfaces are common across a wide variety of storage devices, including disk drives, disk arrays, and FLASH- and RAM-based devices.

Today's storage interface abstracts a storage device as a linear array of fixed-sized *logical blocks* (usually 512 bytes). Details of the mapping of logical blocks to physical media locations are hidden. The interface allows systems to READ and WRITE ranges of blocks by providing a starting logical block number (LBN) and a block count.

Unwritten contract: Although no performance specifications of particular access types are given, an unwritten contract exists between host systems and storage devices supporting these standard interfaces (e.g., disks). This unwritten contract has three terms:

- Sequential accesses are best, much better than non-sequential.
- An access to a block near the previous access in LBN space is usually considerably more efficient than an access to a block farther away.
- Ranges of the LBN space are interchangeable, such that bandwidth and

positioning delays are affected by relative LBN addresses but not absolute LBN addresses.

Application writers and system designers assume the terms of this contract in trying to improve performance.

4.1 Disks and standard abstractions

Disk drives are multi-dimensional machines, with data laid out in concentric circles on one or more media platters that rotate continuously. Data is divided into fixed-sized units, called sectors (usually 512 bytes to match the LBN size). The sector (and, thereby, LBN) size was originally driven by a desire to amortize both positioning costs and the overhead of the powerful error-correcting codes (ECC) required for robust magnetic data storage. The densities and speeds of today's disk drives would be impossible without these codes, and many disk technologists would like the sector size (and, thus, the LBN size) to grow by an order of magnitude to support more powerful codes. Each sector is addressed by a tuple, denoting its cylinder, surface, and rotational position.

LBNs are mapped onto the physical sectors of the disk to take advantage of the disk's characteristics. Sequential LBNs are mapped to sequential rotational positions within a single track, which leads to the first point of the unwritten contract. Since the disk is continuously rotating, once the heads are positioned, sequential access is very efficient. Non-sequential access incurs large re-positioning delays. Successive tracks of LBNs are traditionally mapped to surfaces within cylinders, and then to successive cylinders. This leads to the second point of the unwritten contract: that distant LBNs map to distant cylinders, leading to longer seek times.

The linear abstraction works for disk drives, despite their clear three-dimensional nature, because two of the dimensions are largely uncorrelated with LBN addressing. Access time is the sum of the time to position the read/write heads to the

destination cylinder (seek time), the time for the platters to reach the appropriate rotational offset (rotational latency), and the time to transfer the data to or from the media (transfer time). Seek time and rotational latency usually dominate transfer time. The heads are positioned as a unit by the seek arm, meaning that it usually doesn't matter which surface is being addressed. Unless the abstraction is stripped away, rotational latency is nearly impossible to predict because the platters are continuously rotating and so the starting position is essentially random. The only dimension that remains is that across the cylinders, which determines the seek time.

Seek time is almost entirely dependent on the distance traversed, not on the absolute starting and ending points of the seek. This leads to the third point of the unwritten contract. Ten years ago, all disk tracks had the same number of sectors, meaning that streaming bandwidths (and, thus, transfer times) were uniform across the LBN space. Today's zoned disk geometries, however, violate the third term since streaming bandwidth varies between zones.

4.1.1 Holes in the abstraction boundary

Over its fifteen year lifespan, several shortcomings of the interface and the unwritten contract have been identified. Perhaps the most obvious violation has been the emergence of multi-zone disks, in which the streaming bandwidth varies by over 50% from one part of the disk to another. Some application writers exploit this difference by explicitly using the low-numbered LBNs, which are usually mapped to the outer tracks. Over time, this may become a fourth term in the unwritten contract.

Some have argued [Denehy et al. 2002; Schindler et al. 2004] that the storage interface should be extended for disk arrays. Disk arrays contain several disks which are combined to form one or more logical volumes. Each volume can span multiple disks, and each disk may contain parts of multiple volumes. Hiding the boundaries, parallelism, and redundancy schemes prevents applications from ex-

ploiting them. Others have argued [Ganger 2001] that, even for disks, the current interface is not sufficient. For example, knowing track boundaries can improve performance for some applications [Schindler et al. 2002].

The interface persists, however, because it greatly simplifies most aspects of incorporating storage components into systems. Before this interface became standard, systems used a variety of per-device interfaces. These were replaced because they complicated systems greatly and made components less interchangeable. This suggests that the bar should be quite high for a new storage component to induce the introduction of a new interface or abstraction.

It is worth noting that some systems usefully throw out abstraction boundaries entirely, and this is as true in storage as elsewhere. In particular, storage researchers have built tools [Schindler and Ganger 1999; Talagala et al. 2000] for extracting detailed characteristics of storage devices. Such characteristics have been used for many ends: writing blocks near the disk head [Zhang et al. 2002], reading a replica near the disk head [Yu et al. 2000], inserting background requests into foreground rotational latencies [Lumb et al. 2002], and achieving semi-preemptible disk I/O [Dimitrijević et al. 2003]. Given their success, adding support for such ends into component implementations or even extending interfaces may be appropriate. But, they do not represent a case for removing the abstractions in general.

4.2 MEMStores and standard abstractions

Using a standard storage abstraction for MEMStores has the advantage of making them immediately usable by existing systems. Interoperability is important for getting MEMStores into the marketplace, but if the abstractions that are used make performance suffer, then there is reason to consider something different.

This section explains how the details of MEMStore operation make them naturally conform to the storage abstraction used for disks. Also, the unwritten contract

that applications expect will remain largely intact.

4.2.1 Access method

The standard storage interface allows accesses (READs and WRITEs) to ranges of sizeable fixed-sized blocks. The question to ask first is whether such an access method is appropriate for a MEMStore.

Is a 512-byte block appropriate, or should the abstraction use something else? It is true that MEMStores can dynamically choose subsets of read/write tips to engage when accessing data, and that these subsets can, in theory, be arbitrarily-sized. However, enough data must be read or written for error-correcting codes (ECC) to be effective. The use of ECC enables high storage density by relaxing error-rate constraints. Since the density of a MEMStore is expected to equal or exceed that of disk drives, the ECC protections needed will be comparable. Therefore, block sizes of the same order of magnitude as disks have should be expected. Also, any block's size must be fixed, since it must be read or written in its entirety, along with the associated ECC. Accessing less than a full block, e.g., to save energy [Lin et al. 2002], would not be possible. The flexibility of being able to engage arbitrary sets of read/write tips can still be used to selectively choose sets of these fixed-sized blocks.

Large block sizes are also motivated by embedded servo mechanisms, coding for signal processing, and the relatively low per-tip data rate of around 1 Mbit/s. The latter means that data will have to be spread across multiple parallel-operating read/write tips to achieve an aggregate bandwidth that is on-par with that of disk drives. Spreading data across multiple read/write tips also introduces physical redundancy that will improve tolerance of tip failures. MEMStores will use embedded servo [Terris et al. 1998], requiring that several bits containing position information be read before any access in order to ensure that the media sled is positioned correctly. Magnetic recording techniques commonly use transitions between bits rather than the bits themselves to represent data, meaning that a

sequence of bits must be accessed together. Further, signal encodings use multi-bit codewords that map a sequence of bits to values with interpretable patterns (e.g., not all ones or all zeros). The result is that, in order to access any data after a seek, some amount of data (10 bits in my model) must be read for servo information, and then bits must be accessed sequentially with some coding overhead (10 bits per byte in my model). Given these overheads, a large block size should be used to amortize the costs. This block will be spread across multiple read/write tips to improve data rates and fault tolerance.

Using current storage interfaces, applications can only request ranges of sequential blocks. Such access is reasonable for MEMStores, since blocks are laid out sequentially, and their abstraction should support the same style of access. There may be utility in extending the abstraction to allow applications to request batches of non-contiguous LBNs that can be accessed by parallel read/write tips. An extension like this is discussed in Section 6.3.

4.2.2 Unwritten contract

Assuming that MEMStore access uses the standard storage interface, the next step is to see if the unwritten contract for disks still holds. If it does, then MEMStores can be used effectively by systems simply as fast disks.

The first term of the unwritten contract is that sequential access is more efficient than random access. This will continue to be the case for MEMStores because data must still be accessed in a linear fashion. The signal processing techniques that are commonly used in magnetic storage are based on transitions between bits, rather than the state of the bits in isolation. Moreover, they only work properly when state transitions come frequently enough to ensure clock synchronization so they encode multi-bit data sequences into alternate codewords. These characteristics dictate that the bits must be accessed sequentially. Designs based on recording techniques other than magnetic will, most likely, encode data similarly. Once the media sled is in motion, it is most efficient for it to stay in motion, so the most

efficient thing to access is the next unit of sequential data, just as it is for disks.

The second term of the unwritten contract is that the difference between two LBN numbers should map well to the physical distance between them. This is dependent on how LBNs are mapped to the physical media, and this mapping can easily be constructed in a MEMStore to make the second point of the unwritten contract true. A MEMStore is a multi-dimensional machine, just like a disk, but the dimensions are correlated differently. Each media position is identified by a tuple of the X position, the Y position, and the set of read/write tips that are enabled, much like the cylinder/head/rotational position tuples in disks. There are thousands of read/write tips in a MEMStore, and each one accesses its own small portion of the media. Just as the heads in a disk drive are positioned as a unit to the same cylinder, the read/write tips in a MEMStore are always positioned to the same offset within their own portion of the media. The choice of which read/write tips to activate has no correlation with access time, since any set can be chosen for the same cost once the media is positioned.

As with disks, seek time for a MEMStore is a function of seek distance. Since the actuators on each axis are independent, the overall seek time is the maximum of the individual seek times in each dimension, X and Y. But, the X seek time almost always dominates the Y seek time because extra settle time must be included for X seeks, but not for Y seeks. The reason for this is that post-seek oscillations in the X dimension lead to off-track interference, while the same oscillations in the Y dimension affect only the bit rate of the data transfer. Since the overall seek time is the maximum of the two individual seek times, and the X seek time is almost always greater than the Y seek time, the overall seek distance is (almost) uncorrelated with the Y position, as seen in Figure 3.3. In the end, despite the fact that a MEMStore has multiple dimensions over which to position, the overall access time is (almost) only correlated with just a single dimension, which makes a linear abstraction sufficient.

The last term of the unwritten contract states that the LBN space is uniform,

and that access time does not vary across the range of the LBNs. The springs that attach the media sled to the chip do affect seek times by applying a greater restoring force the further they are displaced. However, the effect is minimal, with seek times varying by at most 10–15%, meaning that overall access times at the application level would vary by far less. Also, MEMStores do not need zoned recording. It is safe to say that the last point of the unwritten contract still holds: ranges of the LBN space of a MEMStore are interchangeable.

4.3 Summary

Like MEMStores, disk drives are multi-dimensional mechanical machines. Using a linear logical block abstraction for disk drives hides details of the device that could be usefully exploited by systems. However, the linear logical block abstraction works well for disk drives because of their access characteristics. This chapter has explored this fit, and discussed how the same abstraction works well for MEMStores for many of the same reasons.

The next two chapters examine potential MEMStore-specific roles and policies, using the two objective tests from Chapter 1. The two tests answer the question of whether a new device is sufficiently different from a disk drive to warrant using a different abstraction. The first test, the specificity test, asks whether a potential role or policy can apply to a disk drive as well as a MEMStore. Given that a potential role or policy passes the specificity test, the second test, the merit test decides whether it makes enough of an impact on performance to justify changing the abstraction.

5 Roles of MEMStores in systems

MEMStores can take on various roles in a system, the simplest of which is to be the main bulk storage instead of a disk drive. There are some applications for which a disk drive cannot fill this role, perhaps because of energy, cost, or size constraints. For example, cellular telephones will probably not be able to use disk drives for some or all of these reasons. In these applications, MEMStores clearly have an advantage and can fill this role. Further, other applications may use MEMStores simply because they demand the fastest performance possible.

This chapter examines the use of MEMStores in three different roles. The first is as a simple disk replacement, the second is as a nonvolatile cache for disk drives, and the third is as an augmentation of the existing disk drives in a large disk array.

5.1 Devices for comparison

5.1.1 G2 MEMStore

The MEMStore used for comparisons in this chapter is the G2 design point described in Table 3.1.

5.1.2 IBM Microdrive

The IBM Microdrive (described in Section 2.6.2) is a miniature hard disk drive that was introduced in the late 1990's for use in mobile applications such as digital cameras, music players, and PDAs. It is highly optimized for small size and low energy requirements rather than access performance. The model used for compar-

ison in this chapter is of the 1 GB device (model DSCM-11000). DiskSim models of this drive were provided by the members of the Dempsey project [Zedlewski et al. 2003] at Princeton University.

5.1.3 Seagate Cheetah 36ES

The Cheetah is Seagate's current enterprise-market drive, meant for servers and disk arrays. It is designed for high performance and reliability, rather than for capacity and low cost. The specific drive evaluated here is the Cheetah 36ES (ST336706LC), a 36 GB disk. Clearly the Cheetah is not targeting the same market as a MEMStore, but it is included here as a point of comparison to the fastest modern disks.

5.1.4 Quantum Atlas 10K

The Atlas 10K was Quantum's (now Maxtor) high-end enterprise SCSI drive in 1999 [Quantum 1999]. The specific drive used in some of the experiments below is the 9 GB version of the drive which rotates at 10,000 RPM and has an average seek time of 5.7 ms for reads and 6.19 ms for writes. The experiments use a validated DiskSim model of this drive [Ganger and Schindler 2004].

5.1.5 Überdisk

The Überdisk is a hypothetical disk drive that approximates the performance of a G2 MEMStore. Its parameters given in Table 5.1 are based on extrapolating from today's disk characteristics. The Überdisk is also modeled using DiskSim. In order to do a capacity-to-capacity comparison, I use only the first 3.46 GB of the Überdisk to match the capacity of the G2 MEMStore. The two devices have equivalent performance under a random workload of 4 KB requests that are uniformly distributed across the capacity (3.46 GB) and arrive one at a time.

The seek curve generated for the Überdisk model is based on the formula

Capacity	41.6 GB
Rotation speed	55,000 RPM
One-cylinder seek time	0.1 ms
Full-stroke seek time	2.0 ms
Head switch time	0.01 ms
Number of cylinders	39511
Number of surfaces	2
Average access time	0.88 ms
Streaming bandwidth	100 MB/s

Table 5.1: **Überdisk parameters.** The Überdisk is a hypothetical future disk drive. Its parameters are scaled from current disks, and are meant to represent those of a disk that matches the performance of a MEMStore. The average response time is for a random workload which exercised only the first 3.46 GB of the disk in order to match the capacity of the G2 MEMStore.

from [Ruemmler and Wilkes 1994], with specific values chosen for the one-cylinder and full-stroke seeks. Head switch and one-cylinder seek times are expected to decrease in the future due to microactuators integrated into disk heads, leading to shorter settle times. With increasing track densities, the number of platters in disk drives is decreasing steadily, so the Überdisk has only two surfaces. The zoning geometry is based on simple extrapolation of current linear densities.

An Überdisk does not necessarily represent a realistic disk; for example, a rotation rate of 55,000 RPM (approximately twice the speed of a dental drill) may never be attainable in a reasonably-priced disk drive. However, this rate was necessary to achieve an average rotational latency that is small enough to match the average access time of the MEMStore. The Überdisk is meant to represent the combination of parameters that would be required of a disk in order to match the performance of a MEMStore.

If the performance of a workload running on a MEMStore is the same as one running on an Überdisk, then any performance improvement is due only to the intrinsic speed of the device, and not due to the fact that it is a MEMStore or an Überdisk. If the workload performs differently on the two devices, then it must be especially well-matched to the characteristics of one device or the other.

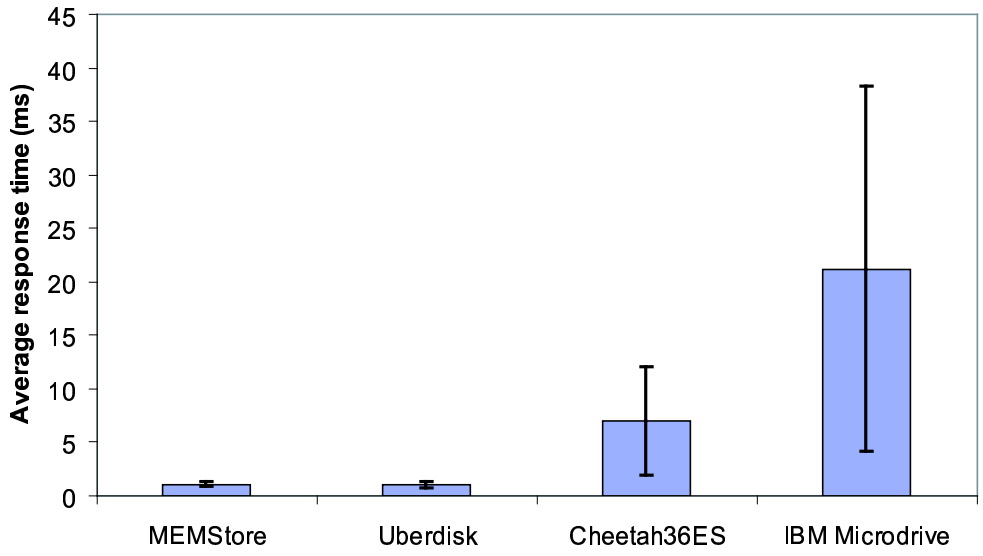


Fig. 5.1: **Random workload performance.** The workload in this experiment was 10,000 random requests uniformly distributed across the capacity of the device. Each request was sized with an exponential distribution with a mean of 4 KB. Requests were issued every 50 ms. Error bars show the standard deviation.

5.2 Simple disk replacement

It is clear that MEMStores can fill roles in systems that disk drives fill today. This section directly compares the performance of MEMStores to that of disk drives using several different workloads.

This section compares four storage devices: the G2 MEMStore described in Chapter 3, the Seagate Cheetah 36ES disk drive, the IBM Microdrive (Model DSCM-11000)[1], and the Überdisk.

5.2.1 Synthetic workloads

Figure 5.1 shows the average response time of four storage devices under a synthetic workload of 10,000 requests. The requests are uniformly distributed across the capacity of each device, and are sized with an exponential distribution with a

[1]DiskSim model of the Microdrive is courtesy of the Dempsey project at Princeton University.

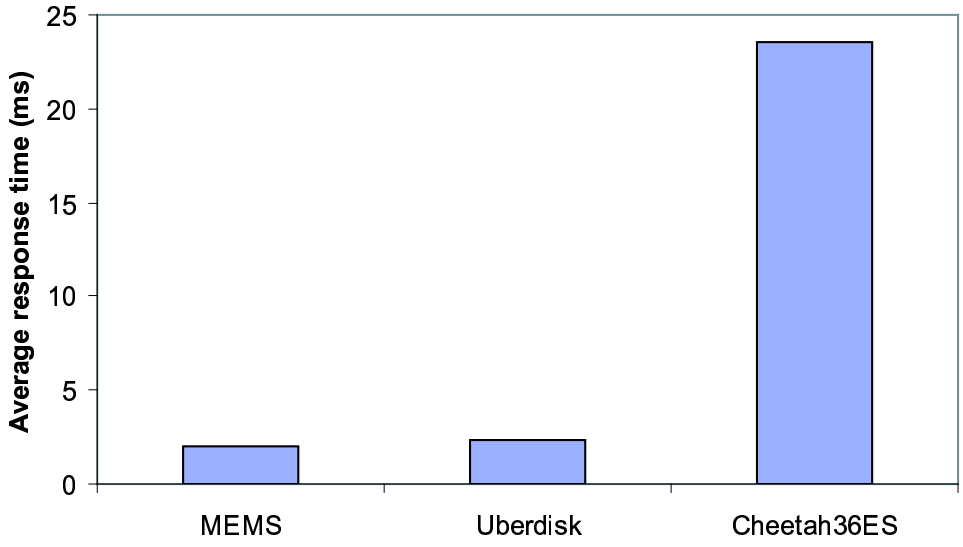


Fig. 5.2: **Performance comparison of G2 MEMStore, Überdisk, and Cheetah36ES with one week of the HP Cello trace from 1999.**

mean of 50 ms. The MEMStore and the Überdisk have equal performance, with average response times of 1.1 ms each. The Cheetah disk has an average response time of 7 ms. The Microdrive has very poor random performance, with an average response time of 21.2 ms. The MEMStore and the Überdisk both have very little variation in response time, with standard deviations of 0.2 ms and 0.3 ms, respectively. This is in stark contrast to the wide standard deviations of the Cheetah disk and Microdrive, with standard deviations of 5.1 ms and 17.1 ms, respectively.

5.2.2 Trace replay

Figure 5.2 shows the result of replaying a workload trace in the DiskSim simulator configured as a G2 MEMStore, an Überdisk, and a Cheetah 36ES disk drive. The workload is from a departmental server at Hewlett-Packard Laboratories called Cello, and was gathered during one week in February, 1999. Storage connected to the server varied from single disks to a large disk array, and the trace collected requests to all of them. I isolated just those requests that went to a single 9 GB

disk, which stored the department's Internet newsgroup feeds. This was one of the busiest disks in the system. The G2 MEMStore is only 4 GB, so I used three of them configured as a single logical volume. As can be seen in the figure, the G2 MEMStore outperforms the Cheetah36ES disk by just over a factor of ten (2.02 ms versus 23.5 ms average response time). The Überdisk performs slightly worse than the MEMStore (2.32 ms average response time). This is due in part to the fact that there are three MEMStores and only one Überdisk in the experiment.

5.3 MEMStores as caches for disks

MEMStores can also be used to augment an existing storage hierarchy. For example, with their low entry cost, MEMStores could be incorporated into future disk drives as very large (1-10 GB) nonvolatile caches. The superior performance of MEMStores would allow the cache to absorb latency-critical synchronous writes to metadata and cache small files to improve small read performance. For example, Baker et al. showed that using fast nonvolatile storage to absorb synchronous disk writes both at a client and at a file server increases performance between 20% and 90% [Baker et al. 1992].

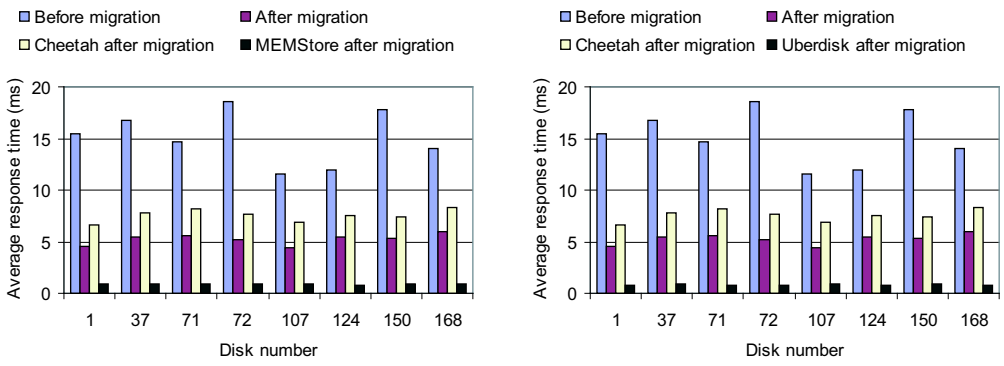
To explore MEMStores as nonvolatile caches for disk, DiskSim was augmented to allow a MEMStore to serve as a cache for a disk. The cache was 2.5 GB, the disk was 9.2 GB, and the workload was the 1-day Cello trace from [Ruemmler and Wilkes 1993]. This trace actually includes eight separate devices so the experiments use a cache per disk. The results show that the average I/O response time is 14.66 ms for a Quantum Atlas 10K disk drive [Quantum 1999] without any MEMStore cache vs. 4.03 ms for a disk with a G2 MEMStore (and 2.76 ms for a single large G2 MEMStore that replaced the disk). Since most of the read requests are serviced from the client-side DRAM cache, the 3.5× performance improvement, over just a disk drive, is achieved mainly by quickly servicing writes. However, unlike DRAM-based write caching (which absorbs writes but risks los-

ing data), the MEMStore cache is nonvolatile, providing the same data integrity guarantees as disk drives. An alternate experiment, in which all eight devices in the Cello trace were re-mapped to a larger version of the Atlas 10K disk with a single MEMStore cache, only suffered a slight increase in average access time to 4.66 ms. This longer service time stems from an increase in queueing time since the large single device is doing the work of eight. It shows, however, that caching absorbs enough of the device's activity to provide a good performance boost.

Instead of using the MEMStore as a cache, it is also possible to expose the device to the operating system so that file systems can allocate specific data onto it. Depending on their access patterns and performance needs, file systems could place small structures (e.g., file system metadata) on MEMStores, while using the disk for streamed or infrequently-accessed data. This could be done on individual disks or within RAID arrays, creating the potential for AutoRAID-like systems [Wilkes et al. 1995]. Further, because RAID arrays are less cost-sensitive than individual disks, arrays of MEMStores could be incorporated more cost-effectively into RAID arrays, providing significant performance improvements for RAID's costly write operations.

5.4 Disk array augmentation

One of the roles that has been suggested for MEMStores in systems is that of augmenting or replacing some or all of the disks in a disk array to increase performance [Schlosser et al. 2000; Uysal et al. 2003]. However, the lower capacity and potentially higher cost of MEMStores suggest that it would be impractical to simply replace all of the disks. Therefore, they represent a new tier in the traditional storage hierarchy, and it will be important to choose which data in the array to place on the MEMStores and which to store on the disks. Uysal et al. evaluate several methods for partitioning data between the disks and the MEMStores in a disk array [Uysal et al. 2003]. The experiment described below is similar, in that



(a) MEMStore (b) Überdisk

Fig. 5.3: **Using MEMStores in a disk array.** These graphs show the result of augmenting overloaded disks in a disk array with faster storage components: a MEMStore (a) or an Überdisk (b). In both cases, the busiest logical volume on the original disk (a 73 GB Seagate Cheetah) is moved to the faster device. Requests to the busiest logical volume are serviced by the faster device, and the traffic to the Cheetah is reduced. The results for both experiments are nearly identical, leading to the conclusion that the MEMStore and the Überdisk are interchangeable in this role (e.g., it is not MEMStore-specific.)

a subset of the data that is stored on the back-end disks in a disk array is moved to a MEMStore.

Some increase in performance is expected from doing this, as Uysal et al. report. However, the question this dissertation asks is whether the benefits are from a MEMStore-specific attribute, or just from the fact that MEMStores are faster than the disks used in the disk array. Applying the specificity test answers this question by comparing the performance of a disk array back-end workload on three storage configurations. The first configuration uses just the disks that were originally in the disk array. The second configuration augments the overloaded disks with a MEMStore. The third does the same with an Überdisk.

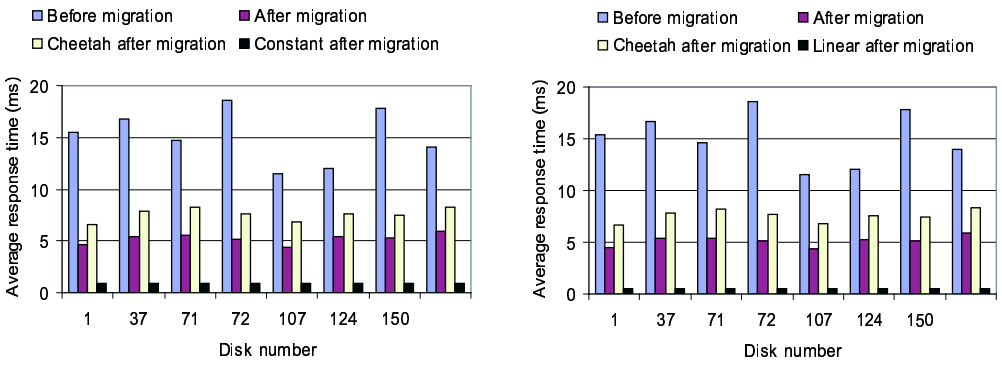
The workload is a disk trace gathered from the disks in the back-end of an EMC Symmetrix disk array during the summer of 2001. The disk array contained 282 Seagate Cheetah 73 GB disk drives, model number ST173404. From those, the experiment uses the eight busiest (disks 1, 37, 71, 72, 107, 124, 150, and 168),

which have an average request arrival rate of over 69 requests per second for the duration of the trace, which was 12.5 minutes. Each disk is divided into 7 logical volumes, each of which is approximately 10 GB in size. For each "augmented" disk, the busiest logical volume was moved to a faster device, either a MEMStore or an Überdisk. The benefit should be twofold: first, response times for the busiest logical volume will be improved, and second, traffic to the original disk will be reduced. Requests to the busiest logical volume are serviced by the faster device (either a MEMStore or an Überdisk), and all other requests are serviced by the original Cheetah disk.

Figure 5.3(a) shows the result of the experiment with the MEMStore. For each disk, the first bar shows the average response time of the trace running just on the Cheetah, which is 15.1 ms across all of the disks. The second bar shows the average response time of the same requests after the busiest logical volume has been moved to the MEMStore. Across all disks, the average is now 5.24 ms. The third and fourth bars show, respectively, the average response time of the Cheetah with the reduced traffic after augmentation, and the average response time of the busiest logical volume, which is now stored on the MEMStore. We indeed see the anticipated benefits — the average response time of requests to the busiest logical volume have been reduced to 0.86 ms, and the reduction of load on the Cheetah disk has resulted in a lower average response time of 7.56 ms.

Figure 5.3(b) shows the same experiment, but with the busy logical volume moved to an Überdisk rather than a MEMStore. The results are almost exactly the same, with the response time of the busiest logical volume migrated to the Überdisk being around 0.84 ms, and the overall response time reduced from 15.1 ms to 5.21 ms.

The fact that the MEMStore and the Überdisk provide the same benefit in this role means that this role fails the specificity test. In this role, a MEMStore really can be considered to be just a fast disk. The workload is not specifically matched to the use of a MEMStore or an Überdisk, but can clearly be improved



(a) Simpledisk-constant (b) Simpledisk-linear

Fig. 5.4: **Using Simpledisks in a disk array.** These graphs show the same experiment as shown in Figure 5.3, but with two Simpledisk models instead of MEMStores and Überdisks.

with the use of any faster device, regardless of its technology.

Although it is imperceptible in Figure 5.3, the Überdisk gives slightly better performance than the MEMStore because it benefits more from workload locality due to the profile of its seek curve. The settling time in the MEMStore model makes any seek expensive, with a gradual increase up to the full-stroke seek. The settling time of the Überdisk is somewhat less, leading to less expensive initial seek and a steeper slope in the seek curve up to the full-stroke seek. The random workload used to compare devices has no locality, but the disk array trace does.

Figure 5.4 examines this further by showing the same experiment but with two other disk models, called *Simpledisk-constant* and *Simpledisk-linear*. Simpledisk-constant responds to requests in a fixed amount of time, equal to that of the response time of the G2 MEMStore under the random workload: 0.88 ms. The response time of Simpledisk-linear is a linear function of the distance from the last request in LBN space. The endpoints of the function are equal to the single-cylinder and full-stroke seek times of the Überdisk, which are 0.1 ms and 2.0 ms, respectively. Simpledisk-constant should not benefit from locality, and Simpledisk-linear should benefit from locality even more than either the MEMStore or the

Überdisk. Augmenting the disk array with these devices gives response times to the busiest logical volume of 0.92 ms and 0.52 ms, respectively. As expected, Simpledisk-constant does not benefit from workload locality and Simpledisk-linear benefits more than a real disk.

Uysal et al. proposed several other MEMStore/disk combinations for disk arrays [Uysal et al. 2003], including replacing all of the disks with MEMStores, replacing half of the mirrors in a mirrored configuration, and using the MEMStore as a replacement of the NVRAM cache. In all of these cases, and in most of the other roles outlined in Chapter 2, the MEMStore is used simply as a block store, with no tailoring of access to MEMStore-specific attributes. I believe that if the specificity test were applied, and an Überdisk was used in each of these roles, the same performance improvement would result. Thus, the results of prior research apply more generally to faster mechanical devices.

5.5 Summary

Most roles that MEMStores will fill really only benefit from the MEMStore's intrinsic properties, i.e., that they are faster, smaller, or use less energy than disk drives. Systems that use them will have improved performance, of course, but they will not require a new abstraction or interface for the MEMStore if this is the only benefit. The next chapter examines how systems may benefit from tailoring their access policies when using MEMStores.

6 Policies for accessing MEMStores

Once MEMStores are used in systems, those systems can implement specific policies to tailor their use. If MEMStores have specific features from which a system can benefit, beyond just the fact that they are faster, smaller, and use less energy than disk drives, then those policies should be MEMStore-specific and may require an abstraction that is different from that used for current storage devices. The specificity test and the merit test from Chapter 1 allow this question to be answered. This chapter evaluates several potential MEMStore-specific access policies using the two objective tests.

6.1 Request scheduling

An important mechanism for improving storage device efficiency is deliberate scheduling of pending requests. Request scheduling improves efficiency because positioning delays are dependent on the relative positions of the read/write head and the destination sector. The same is true of MEMStores, whose seek times are dependent on the distance to be traveled. Some scheduling policies are most effectively implemented inside of the device because of extra knowledge that exists there. Other policies can be implemented externally in the host software, i.e., inside the operating system, because they do not require extra information about the system. This section explores the impact of different request scheduling algorithms on the performance of MEMStores.

6.1.1 Evaluating scheduling algorithms

Some of the experiments below use a synthetically-generated workload called *Random*. For this workload, request inter-arrival times are drawn from an exponential distribution; the mean is varied to simulate a range of workloads. All other aspects of the requests are independent: 67% are reads, 33% are writes, the request size distribution is exponential with a mean of 4 KB, and request starting locations are uniformly distributed across the device's capacity.

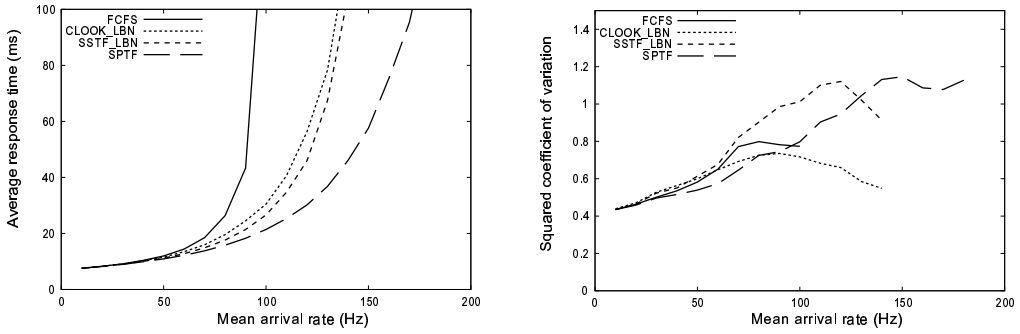
To study more realistic workloads, other experiments use two traces of real disk activity: the *TPC-C* trace and the *Cello* trace. The TPC-C trace came from a TPC-C testbed, consisting of a Microsoft SQL Server atop Windows NT. The hardware was a 300 MHz Intel Pentium II-based system with 128 MB of memory and a 1 GB test database striped across two Quantum Viking disk drives. The trace captured one hour of disk activity for TPC-C, and its characteristics are described in more detail in [Riedel et al. 2000]. The Cello trace came from a Hewlett-Packard system running the HP-UX operating system. This trace is from the same machine as the trace used in Chapter 5, but is from 1992. It captured disk activity from a server at HP Labs used for program development, simulation, mail, and news. While the total trace is actually two months in length, I report data for a single, day-long snapshot. This trace and its characteristics are described in detail in [Ruemmler and Wilkes 1993]. When replaying the traces, each traced disk is replaced by a distinct simulated MEMStore.

As is often the case in trace-based studies, the simulated devices are newer and significantly faster than the disks used in the traced systems. To explore a range of workload intensities, I replicate an approach used in previous disk scheduling work [Worthington et al. 1994b]: we scale the traced inter-arrival times to produce a range of average inter-arrival times. When the scale factor is one, the request inter-arrival times match those of the trace. When the scale factor is two, the traced inter-arrival times are halved, doubling the average arrival rate.

6.1.2 Existing disk-based algorithms

Many disk scheduling algorithms have been devised and studied over the years. In this section, I describe and compare the performance of four of them both on a disk drive and on a MEMStore. The first is first-come, first-served (*FCFS*), which is the simplest and often gives the poorest performance. The second algorithm is called cyclical look (*CLOOK_LBN*) and it services requests in ascending LBN order, starting over with the lowest LBN when all requests are "behind" the most recent request [Seaman et al. 1966]. The third, shortest seek time first (*SSTF_LBN*) was designed to select the request that will incur the smallest seek delay [Denning 1967], but this is rarely the way it functions in practice. Instead, since few host operating systems have the information needed to compute actual seek distances or predict seek times, most SSTF implementations use the difference between the last accessed LBN and the desired LBN as an approximation of seek time. This simplification works well for disk drives [Worthington et al. 1994b] since LBN numbers map well to physical positions. The fourth, shortest positioning time first (*SPTF*), selects the request that will incur the smallest positioning delay [Seltzer et al. 1990; Jacobson and Wilkes 1991]. For disks, this algorithm differs from others in that it explicitly considers both seek time and rotational latency.

The first three of these algorithms (FCFS, CLOOK_LBN, and SSTF_LBN) can be easily and efficiently implemented in host software (i.e., the operating system) because they do not require detailed knowledge of the device. They select requests to be serviced solely based on their requested LBN number. They work well for disk drives because LBN numbers map well (although not perfectly) to physical positions. SPTF is most often implemented within a disk drive's firmware because it requires accurate knowledge of the state of the disk, the exact mapping of LBNs to physical locations, and the exact predicted timing of both seeks and rotational latencies. Request scheduling algorithms running on MEMStores that export an interface which maps LBNs well to physical location should have similar (relative)

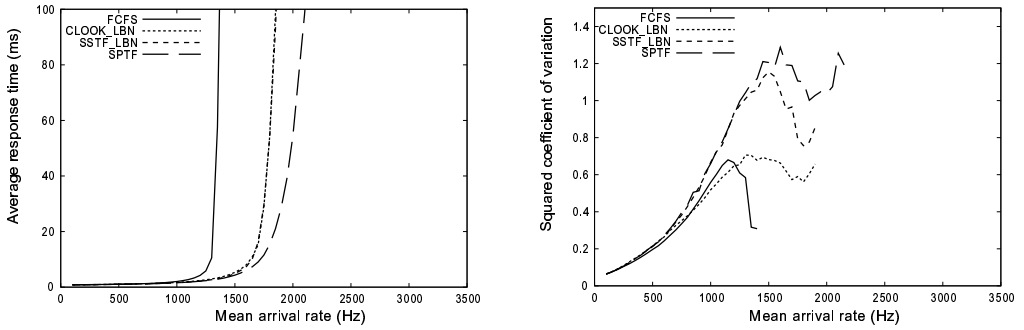


(a) Average response times (b) Squared coefficients of variation (σ^2/μ^2)

Fig. 6.1: **Comparison of scheduling algorithms for the Random workload on the Quantum Atlas 10K disk.**

performance to the same algorithms running on disk drives.

As a reference, Figure 6.1 compares these four disk scheduling algorithms operating on a Quantum Atlas 10K disk drive [Quantum 1999] under the Random workload described above. The graphs show performance as a function of increasing request arrival rate. Two common metrics for evaluating disk scheduling algorithms are shown. First, the average response time (queue time plus service time) shows the effect on average performance. The figure of merit for an algorithm is the point at which performance saturates because the device cannot service requests fast enough. At saturation, queue sizes grow without bound and response times increase dramatically. As expected, FCFS saturates well before the other algorithms as the arrival rate increases. SSTF_LBN outperforms CLOOK_LBN, and SPTF outperforms all other schemes. As a second metric of evaluation, the squared coefficient of variation (σ^2/μ^2) measures "fairness" (or starvation resistance) [Worthington et al. 1994b; Teorey and Pinkerton 1972]; lower values indicate better starvation resistance. As expected, CLOOK_LBN avoids the starvation effects that characterize the SSTF_LBN and SPTF algorithms. Although not shown here, age-weighted versions of these greedy algorithms can reduce request starvation without unduly reducing average case performance [Seltzer et al. 1990; Jacobson and Wilkes 1991].

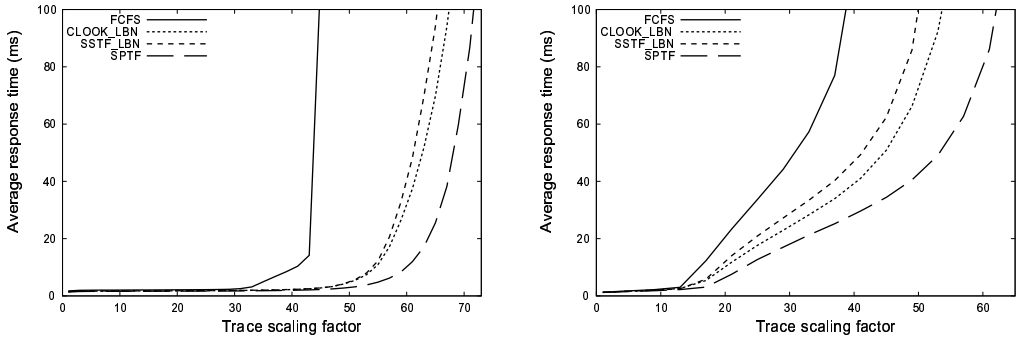


(a) Average response times (b) Squared coefficients of variation (σ^2/μ^2)

Fig. 6.2: **Comparison of scheduling algorithms for the Random workload on the G2 MEMStore.** Note the scale of the X axis has increased by an order of magnitude relative to the graphs in Figure 6.1.

Figure 6.2 shows how well these algorithms work for the G2 MEMStore under the Random workload described above for a range of request arrival rates. In terms of both performance and starvation resistance, the algorithms finish in the same order as for disks: SPTF provides the best performance and CLOOK_LBN provides the best starvation resistance. However, their performance relative to each other merits discussion. The difference between FCFS and the LBN-based algorithms (CLOOK_LBN and SSTF_LBN) is larger for MEMStores because the seek time is a much larger component of the total service time. In particular, there is no subsequent rotational delay. Also, the average response time difference between CLOOK_LBN and SSTF_LBN is smaller for MEMStores, because both algorithms reduce the X seek times into the range where X and Y seek times are comparable. Since neither addresses Y seeks, the greediness of SSTF_LBN is less effective. SPTF obtains additional performance by addressing Y seeks.

Figures 6.3(a) and 6.3(b) show how the scheduling algorithms perform for the Cello and TPC-C workloads, respectively. The relative performance of the algorithms on the Cello trace is similar to the Random workload. The overall average response time for Cello is dominated by the busiest one of Cello's eight disks; some of the individual disks have differently shaped curves but still exhibit the same ordering among the algorithms. One noteworthy difference between TPC-



(a) Cello average response times

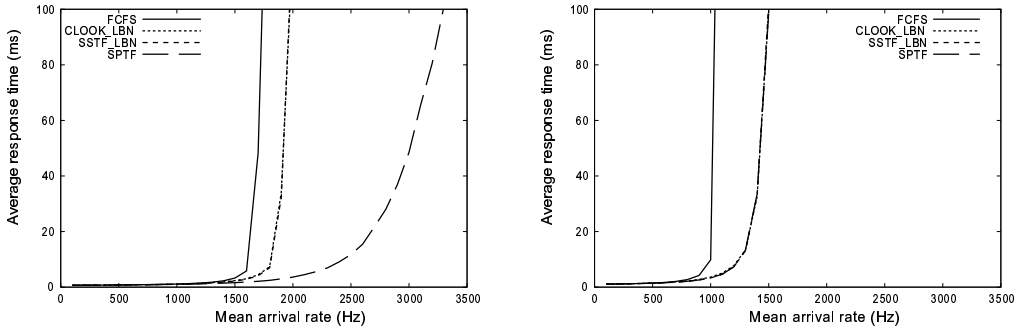
(b) TPC-C average response times

Fig. 6.3: **Comparison of scheduling algorithms for the Cello and TPC-C workloads on the G2 MEMStore.**

C and Cello is that SPTF outperforms the other algorithms by a much larger margin than for TPC-C at high loads. This occurs because the scaled-up version of the workload includes many concurrently-pending requests with very small LBN distances between adjacent requests. LBN-based schemes do not have enough information to choose between such requests, often causing small (but expensive) X-dimension seeks. SPTF addresses this problem and therefore performs much better.

6.1.3 SPTF and settling time

Originally, we had expected SPTF to outperform the other algorithms by a greater margin for MEMStores. Our investigations suggest that the value of SPTF scheduling is highly dependent upon the settling time component of X dimension seeks. With large settling times, X dimension seek times dominate Y dimension seek times, making SSTF_LBN match SPTF. With small settling times, Y dimension seek times are a more significant component. To illustrate this, Figure 6.4 compares the scheduling algorithms with the constant settling time set to zero and 0.44 ms (double the default value). As expected, SSTF_LBN is very close to SPTF when the settling time is doubled. With zero settling time, SPTF outperforms the other algorithms by a large margin.



(a) Random with zero settling time (b) Random with double settling time

Fig. 6.4: **Comparison of average performance of the Random workload for zero and double constant settling time on the G2 MEMStore.** These are in comparison to the default model (Random with constant settling time of 0.22 ms) shown in Figure 6.2(a). With no settling time, SPTF significantly outperforms CLOOK_LBN and SSTF_LBN. With the doubled settling time, CLOOK_LBN, SSTF_LBN, and SPTF are nearly identical.

6.1.4 MEMStore-specific algorithms

Mechanical and structural differences between MEMStores and disks suggest that request scheduling policies that are tailored to MEMStores may provide better performance than ones that were designed for disks. Upon close examination, however, the physical and mechanical motions that dictate how a scheduler may perform on a given device continue to apply to MEMStores as they apply to disks. This may be surprising at first glance, since the devices are so different, but after examining the fundamental assumptions that make schedulers work for disks, it is clear that those assumptions are also true for MEMStores.

To illustrate, I have evaluated a MEMStore-specific scheduling algorithm called *shortest-distance-first*, or SDF. Given a queue of requests, the algorithm compares the Euclidean distance between the media sled's current position and the offset of each request and schedules the request that is closest. The goal is to exploit a clear difference between MEMStores and disks: the fact that MEMStores position over two dimensions rather than only one. When considering the specificity test, it is not surprising that this qualifies as a MEMStore-specific policy. Disk drives do, in fact, position over multiple dimensions, but predicting the positioning time based

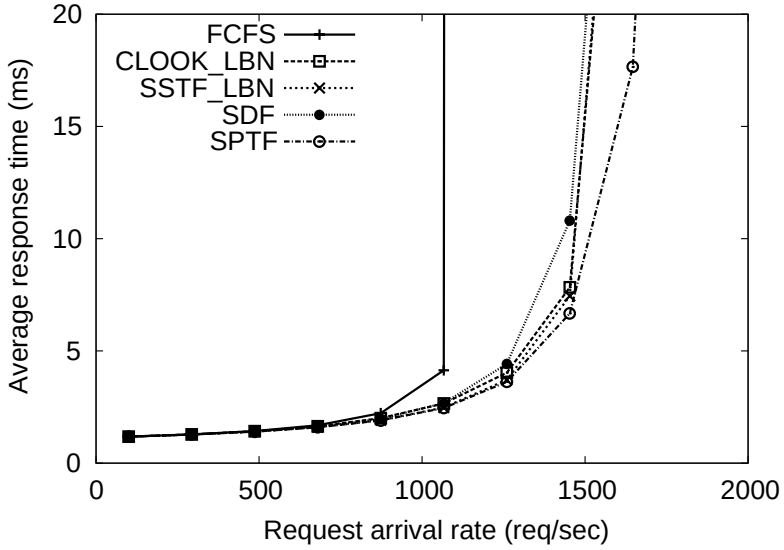


Fig. 6.5: **Performance of shortest-distance-first scheduler.** A MEMStore-specific scheduler that accounts for two-dimensional position gives no benefit over simple schedulers that use a linear abstraction (CLOOK_LBN and SSTF_LBN). This is because seek time in a MEMStore is correlated most strongly with distance only in the X dimension.

on any dimension other than the cylinder distance is very difficult outside of disk firmware. SDF scheduling for MEMStores is easier and could be done outside of the device firmware, assuming that the proper geometry information is exposed through the MEMStore's interface, since it is based only on the logical-to-physical mapping of the device's sectors and any defect management policies used.

Figure 6.5 compares the performance of SDF to that of the other algorithms described above. As expected, FCFS and SPTF perform the worst and the best, respectively. CLOOK_LBN and SSTF_LBN don't perform as well as SPTF because they use only the *LBN* numbers to make scheduling decisions. The SDF scheduler performs slightly worse than CLOOK_LBN and SSTF_LBN. The reason is that positioning time is not as well correlated with two-dimensional position information. In fact, positioning time is only strongly correlated with positioning over the X dimension, as shown in Section 3.3. As such, considering the two-dimensional seek distance does not provide any more utility over considering the one-dimensional seek distance alone, as CLOOK_LBN and SSTF_LBN effectively

do. Thus, the suggested policy fails the merit test: the same or greater benefit can be gained with existing schedulers that don't need MEMStore-specific knowledge. This is based, of course, on the fact that settling time is a significant component of positioning time. I discuss the effect of changing this device characteristic below.

Another MEMStore-specific request scheduling algorithm called zone-based shortest positioning time first (ZSPTF) was suggested by Hong et al. [Hong et al. 2003]. The algorithm combines the performance of SPTF with the starvation resistance of CLOOK_LBN by breaking the logical block space into zones. Requests within a single zone are serviced in SPTF order, and zones are visited in ascending order to improve starvation resistance. The results show somewhat improved performance over standard LBN-based algorithms like CLOOK_LBN and SSTF_LBN, with better starvation resistance than SPTF. However, the authors did not run the same experiments with ZSPTF running on disk drives in order to decide whether it is a truly MEMStore-specific policy. From the description of the algorithm, it is clear that it could be implemented on a disk drive, and that it would probably give the same benefits.

The fundamental reason that scheduling algorithms developed for disks work well for MEMStores are that seek time is strongly dependent on seek distance, but only the seek distance in a single dimension. The seek time is only correlated to a single dimension, which is exposed by the linear abstraction. The same is true for disks when one cannot predict the rotational latencies, in which only the distance that the heads must move across cylinders is relevant. Hence, a linear logical abstraction is as justified for MEMStores as it is for disks.

Of course, there may be yet-unknown policies that exploit features that are specific to MEMStores, and research will surely continue in this area. When considering potential policies for MEMStores, it is important to keep the two objective tests in mind. In particular, these tests can expose a lack of need for a new policy or, better yet, the fact that the policy is equally applicable to disks and other mechanical devices.

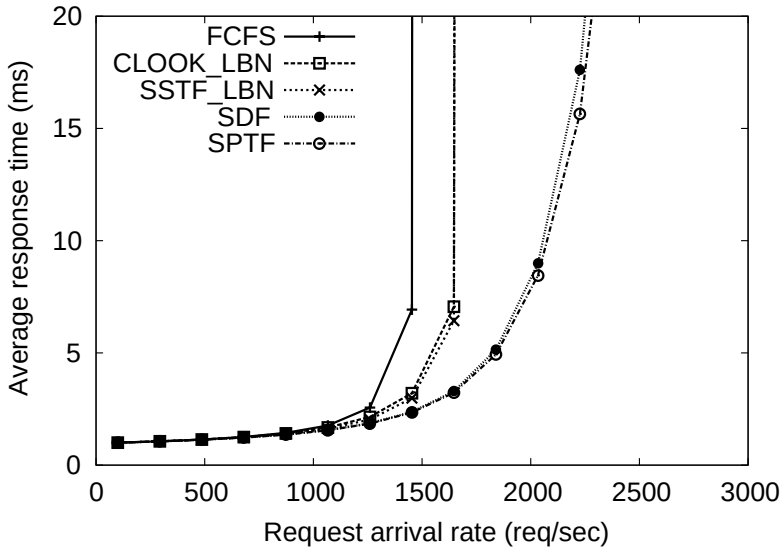


Fig. 6.6: **Performance of shortest-distance-first scheduler without settle time.** If post-seek settle time is eliminated, then the seek time of a MEMStore becomes strongly correlated with both the X and Y positions. In this case, a scheduler that takes into account both dimensions provides much better performance than those that only consider a single dimension (CLOOK_LBN and SSTF_LBN).

6.1.5 Eliminating settling constraints

As described in Section 3.3, seek time is only strongly correlated with one of the two positioning dimensions. This is based on the observation that different mechanisms determine the settling time in each of the two axes, X and Y. Settling time is needed to damp oscillations enough for the read/write tips to reliably access data. In all published MEMStore designs, data is laid out linearly along the Y-axis, meaning that oscillations in Y will appear to the channel as minor variations in the data rate. Contrast this with oscillations in the X-axis, which pull the read/write tips off-track. Because one axis is more sensitive to oscillation than the other, its positioning delays will dominate the other's, unless the oscillations can be damped in near-zero time.

If these differing constraints no longer held, and oscillations affected each axis equally, then MEMStore-specific policies that take into account the resulting two-dimensionality of the seek profile, as illustrated in Figure 3.8, would

become more valuable. Now, for example, two-dimensional distance would be a much better predictor of overall positioning time. Figure 6.6 shows the result of repeating the experiment from Section 6.1.4, but with the post-seek settle time set to zero. In this case, the performance of the SDF scheduler very closely tracks shortest-positioning-time-first, SPTF, the scheduler based on full knowledge of positioning time. Further, the difference between SDF and the two algorithms based on single-dimension position (CLOOK_LBN and SSTF_LBN) is now very large. CLOOK_LBN and SSTF_LBN have worse performance because they ignore the second dimension that is now correlated strongly with positioning time.

6.2 Data layout

Space allocation and data placement for disks continues to be a ripe topic of research, and the same will be true of MEMStores. In this section, I discuss how the characteristics of MEMStore positioning costs affect placement decisions for small local accesses and large sequential transfers. A bipartite layout is proposed and is shown to have some potential for improving performance.

6.2.1 Small, skewed accesses

As with disks, short distance seeks are faster than long distance seeks. Unlike disks, MEMStores' spring restoring forces make the effective actuator force (and therefore sled positioning time) a function of location. Figure 6.7 shows the impact of spring forces for seeks inside different "subregions" of a single tip's media region. The spring forces increase with increasing sled displacement from the origin (toward the outermost subregions in Figure 6.7), resulting in longer positioning times for short seeks. As a result, distance is not the only component to be considered when finding good placements for small, popular data items—offset relative to the center could also be considered.

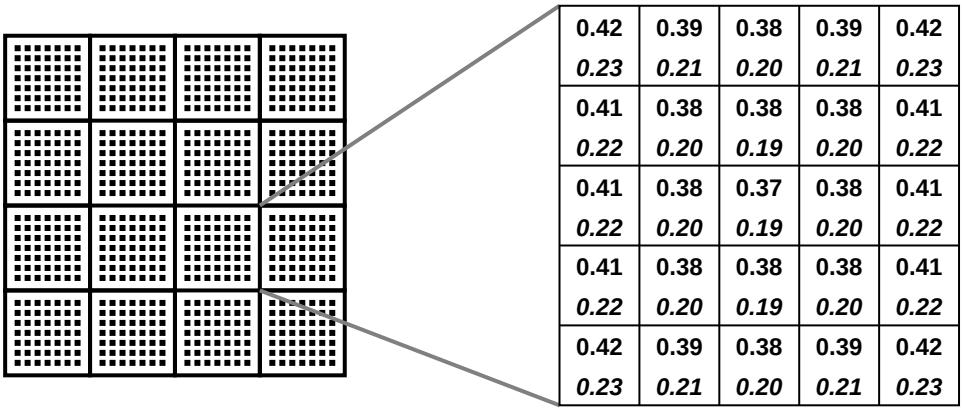


Fig. 6.7: **Difference in request service time for subregion accesses.** This figure divides the region accessible by an individual probe tip into 25 subregions, each 500×500 bits. Each box shows the average request service time (in milliseconds) for random requests starting and ending inside that subregion. The upper numbers represent the service time when the default settling time is included in calculations; numbers in italics represent the service time for zero settling time. Note that the service time differs by 14–21% between the centermost and outermost subregions.

6.2.2 Large, sequential transfers

Streaming media transfer rates for MEMStores and disks are similar: 17.3–25.2 MB/s for the Quantum Atlas 10K [Quantum 1999]; 44.8 MB/s for MEMStores. Positioning times, however, are an order of magnitude shorter for MEMStores than for disks. This makes positioning time relatively insignificant for large transfers (e.g., hundreds of sectors). Figure 6.8 shows the request service times for a 256 KB read with respect to the X distance between the initial and final sled positions. Requests traveling 1250 cylinders (e.g., from the sled origin to maximum sled displacement) incur only a 10% penalty. This lessens the importance of ensuring locality for data that will be accessed in large, sequential chunks. In contrast, seek distance is a significant issue with disks, where long seeks more than double the total service time for 256 KB requests.

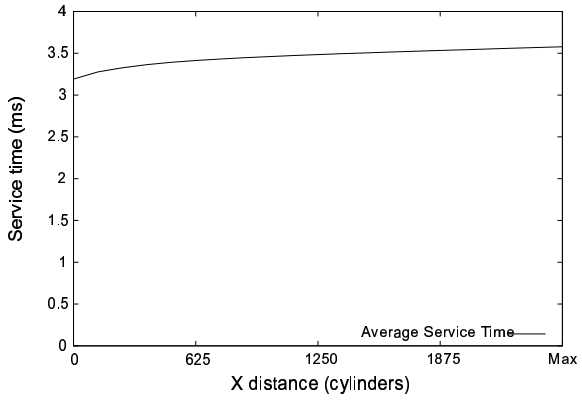


Fig. 6.8: **Large (256 KB) request service time vs. X seek distance for a G2 MEMStore.** Because the media access time is large relative to the positioning time, seeking the maximum distance in X increases the service time for large requests by only 12%.

6.2.3 Bipartite layout

The bipartite layout scheme takes advantage of the above characteristics by placing small data in the centermost subregions. Long, sequential streaming data are placed in outer subregions. Two layouts are tested: a five-by-five grid of subregions (Figure 6.7) and a simple columnar division of the LBN space into 25 columns (e.g., column 0 contains cylinders 0–99, column 1 contains cylinders 100–199, etc.). The difference between these two divisions is that the subregioned layout requires knowledge of the two-dimensional nature of the media, while the columnar layout requires no knowledge of the media layout; it only needs to divide the logical LBN space by the number of columns desired (i.e., 25 in this case).

I compare these layout schemes against the "organ pipe" layout [Vongsathorn and Carson 1990; Ruemmler and Wilkes 1991], an optimal disk-layout scheme, assuming no inter-request dependencies. In the organ pipe layout, the most frequently accessed files are placed in the centermost tracks of the disk. Files of decreasing popularity are distributed to either side of center, with the least frequently accessed files located closer to the innermost and outermost tracks. Although this scheme is optimal for disks, files must be periodically shuffled to maintain the fre-

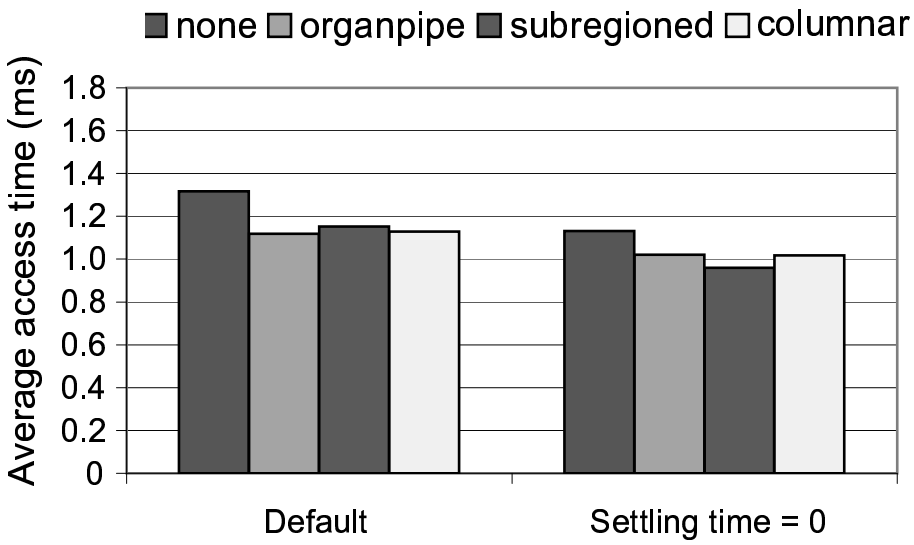


Fig. 6.9: **Comparison of layout schemes for the G2 MEMStore.** For the default device, the organ pipe, subregioned, and columnar layouts achieve a 12–15% performance improvement over a random layout. Further, for the "settling time = 0" case, the subregioned layout outperforms the others by an additional 12%. It is interesting to note that an optimal disk layout technique does not necessarily provide the best performance for a MEMStore.

quency distribution. Further, the layout requires some state to be kept, indicating each file's popularity.

To evaluate these layouts, I used a workload of 10,000 whole-file read requests whose sizes are drawn from the file size distribution reported in [Ganger and Kaashoek 1997]. In this size distribution, 78% of files are 8 KB or smaller, 4% are larger than 64 KB, and 0.25% are larger than 1 MB. For the subregioned and columnar layouts, the large files (larger than 8 KB) were mapped to the ten leftmost and ten rightmost subregions, while the small files (8 KB or less) were mapped to the centermost subregion. To conservatively avoid second-order locality within the large or small files, I assigned a random location to each request within either the large or the small subregions. For the organ pipe layout, I used an exponential distribution to determine file popularity, which was then used to place files.

Figure 6.9 shows that all three layout schemes achieve a 12–15% improvement

in average access time over a simple random file layout. Subregioned and columnar layouts for MEMStores match the organ pipe layout, even with the conservative model, and have no need for keeping popularity data or periodically reshuffling files on the media. For the "no settling time" case, the subregioned layout provides the best performance as it addresses both X and Y.

Applying the specificity test to this potential data layout scheme reveals that the layout is, indeed, specific to MEMStores, since there is no corresponding difference in positioning time across regions of a disk drive. However, applying the merit test shows that this layout scheme may not provide enough benefit to the system to require changing the interface to expose the requisite information to the system. The columnar layout may be used with a normal linear LBN abstraction and provides almost exactly the same benefit as both the organ pipe and subregioned layout. Therefore, by the merit test, it is not clear that taking advantage of this difference in MEMStore positioning dynamics requires a change in the device's abstraction.

6.3 Exploiting tip-subset parallelism

One MEMStore feature that may not be exploited by the standard model of storage is their interesting form of internal access parallelism. Specifically, a subset of the 1000s of read/write tips can be used in parallel to provide high bandwidth media access, and the particular subset does not have to be statically chosen. In contrast to the disk arms in a disk array, which can each seek to independent locations concurrently, all tips are constrained to access the same relative location in their respective regions. For certain access patterns, however, dynamically selecting which subsets of tips should access data can provide great benefits to applications. This section describes the available degrees of freedom MEMStores can employ in parallel access to data and how they can be used for two classes of applications.

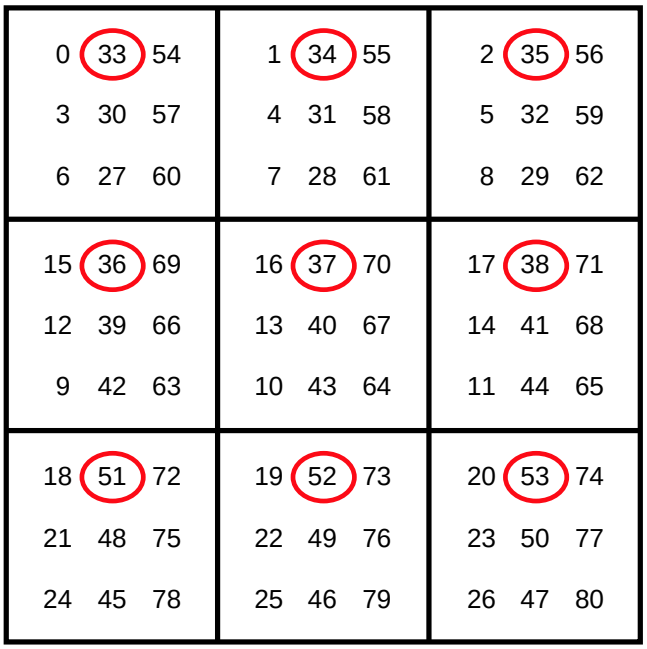


Fig. 6.10: **Data layout with an equivalence class of LBNs highlighted.** The LBNs marked with ovals are at the same location within each square and, thus, comprise an equivalence class. That is, they can potentially be accessed in parallel.

6.3.1 Background

Although a MEMStore includes thousands of read/write tips, it is not possible to do thousands of entirely independent reads and writes. There are significant limitations on what locations can be accessed in parallel. As a result, previous research on MEMStores has treated tip parallelism only as a means to increase sequential bandwidth and to deal with tip failures. This section defines the sets of LBNs that can potentially be accessed in parallel, and the constraints that determine which subsets of them can actually be accessed in parallel.

When a seek occurs, the media is positioned to a specific offset relative to the entire read/write tip array. As a result, at any point in time, all of the tips access the same locations within their squares. An example of this is shown in Figure 6.10 in which LBNs at the same location within each square are identified

with ovals. This set of LBNs form an *equivalence class*. That is, because of their position they can potentially be accessed in parallel. It is important to note that the size of an equivalence class is very small relative to the total number of LBNs in a MEMStore. In the G2 MEMStore described in Chapter 3, the size of an equivalence class is 100, meaning that only 100 LBNs are potentially accessible in parallel at any point out of a total of 6,750,000 total LBNs in the device.

Only a subset of any equivalence class can actually be accessed at once. Limitations arise from two factors: the power consumption of the read/write tips, and components that are shared between read/write tips. It is estimated that each read/write tip will consume 1–3 mW when active and that continuously positioning the media sled would consume 100 mW [Schlosser et al. 2000]. Assuming a total power budget of 1 W, only between 300 and 900 read/write tips can be utilized in parallel which, for realistic devices, translates to 5–10% of the total number of tips. This gives the true number of LBNs that can *actually* be accessed in parallel. In the G2 MEMStore, only 10 of 100 LBNs in an equivalence class can actually be accessed in parallel.

In most MEMStore designs, several read/write tips will share physical components, such as read/write channel electronics, track-following servos, and power buses. Such component sharing makes it possible to fit more tips, which in turn increases volumetric density and reduces seek distances. It also constrains which subsets of tips can be active together, reducing flexibility in accessing equivalence classes of LBNs.

For each LBN and its associated equivalence class, a *conflict relation* can be defined which restricts the equivalence class to reflect shared component constraints. This relation does not actually reduce the *number* of LBNs that can be accessed in parallel, but will affect the choice of *which* LBNs can be accessed together. As real MEMStores have not yet been built, there is no real data on which components might be shared and so I cannot defined any realistic conflict relations. Therefore, this is an avenue of future work to be addressed when real

designs have been implemented.

Figure 6.10 shows a simple example illustrating parallel-accessible LBNs. If one third of the read/write tips can be active in parallel, a system could choose up to 3 LBNs out of a given equivalence class (shown with ovals) to access together. The three LBNs chosen could be sequential (e.g., 33, 34, and 35), or could be disjoint (e.g., 33, 38, and 52). In each case, all of those LBNs would be transferred to or from the media in parallel.[1]

Some MEMStore designs may have an additional degree of freedom: the ability to microposition individual tips by several LBNs along the X dimension. This capability exists to deal with manufacturing imperfections and thermal expansion of the media due to ambient heat. Since the media sled could expand or contract, some tips may need to servo themselves slightly to address the correct columns. By allowing firmware to exploit this micropositioning, the equivalence class for a given LBN grows by allowing access to adjacent cylinders. MEMStore designers indicate that micropositioning by up to 5 columns in either direction is a reasonable expectation. Of course, each tip can access only one column at a time, introducing additional conflict relations.

For example, suppose that the device shown in Figure 6.10 can microposition its tips by one LBN position along the X dimension. This will expand the equivalence class shown in the figure to include the two LBNs to the immediate left and right of the current LBN. The size of the equivalence class will increase by 3×. Micropositioning may not always be available as predicted by a simple model. If the media has expanded or contracted so far that the tip must already position itself far away from its central point, the micropositioning options will be reduced or altered. Lastly, micropositioning does not allow tips to access data in adjacent tips' squares because of inter-square spacing.

[1] Although it is not important to host software, the pictures showing tracks within contiguous rows of squares are just for visual simplicity. The tips over which any sector is striped would be spread widely across the device to distribute the resulting heat load and to create independence of tip failures. Likewise, the squares of sequentially numbered LBNs would be physically spread.

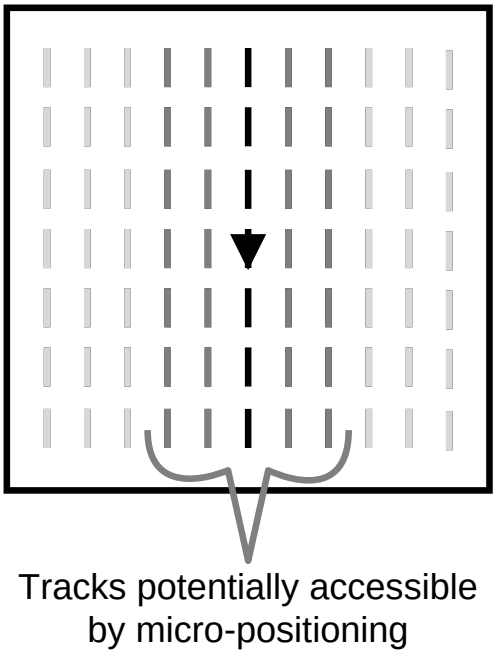


Fig. 6.11: **Micropositioning.** In the CMU design, the probe tips must have some fine-positioning capability in order to deal with thermal expansion of the media sled. This capability could be exposed through the interface, allowing the system to access data in nearby tracks and expanding the range of potentially-accessible data once the sled is positioned. The probe tip in the simple example above could position itself to access two tracks on either side of the base track, increasing the number of potentially-accessible sectors from seven to thirty-five. In reality, the probe tips will probably be able to micro-position over five to ten tracks in either direction.

In summary, for each LBN, an equivalence class of LBNs that can be potentially accessed in parallel with it exists. The members of the set are determined by the LBN's position, and the size of the set is determined by the number of read/write tips in the device and any micropositioning freedom. Further, only a subset (e.g., 5–10%) of the equivalence class can actually be accessed in parallel. The size of the subset is determined by the power budget of the device. If read/write tips share components, then there will be constraints on which LBNs from the set can be accessed together. These constraints are expressed by conflict relations. Lastly, an equivalence class can be expanded significantly (e.g., 11×)

p	Level of parallelism		3
N	Number of squares		9
S_x	Sectors per square in X		3
S_y	Sectors per square in Y		3
M	Degree of micropositioning		0
N_x	Number of squares in X	p	3
N_y	Number of squares in Y	N/p	3
S_T	Sectors per track	$S_y \times N_x$	9
S_C	Sectors per cylinder	$S_T \times N_y$	27

Table 6.1: **Device parameters.** These are the parameters required to determine equivalence classes of LBNs that can be potentially accessed in parallel. The first five parameters are determined by the physical capabilities of the device and the last four are derived from them. The values in the rightmost column are for the simple device shown in Figure 6.10.

due to micropositioning capability.

6.3.2 Exposing tip-subset parallelism

This section describes equations and associated device parameters that a system can use to enumerate LBNs in a MEMStore that can be accessed in parallel.

The goal is that the system be able, for a given LBN, to determine the equivalence class of LBNs that are parallel-accessible. Determining this class for a MEMStore requires four parameters that describe the virtual geometry of the device and one which describes the degree of micropositioning. Table 6.1 lists them with example values taken from the device shown in Figure 6.10. The level of parallelism, p, is set by the power budget of the device, as described in Section 6.3.1. The total number of squares, N, is defined by the virtual geometry of the device. Since sequential LBNs are laid out over as many parallel tips as possible to optimize for sequential access, the number of squares in the X dimension, N_x, is equal to the level of parallelism, p. The number of squares in the Y dimension is the total number of squares, N, divided by p. The sectors per square in either direction, S_x and S_y, is determined by the bit density of each square. These parameters, along with N_x and N_y, determine the number of sectors per track, S_T, and the number of sectors per cylinder, S_C.

Without micropositioning, the size of an equivalence class is simply equal to the total number of squares, N, as there is an equivalent LBN in each square. The degree of micropositioning, M, is another device parameter which gives the number of cylinders in either direction over which an individual tip can microposition. M has the effect of making the equivalence class larger by a factor of $2M + 1$. So, if M in Figure 6.10 were 1, then the equivalence class for each LBN would have (at most) 27 LBNs in it. Micropositioning is opportunistic since, if the media has expanded, the micropositioning range will be used just to stay on track.

Given a single LBN l, a simple two-step algorithm yields all of the other LBNs in the equivalence class E_l. The first step maps l to an x, y position within its square. The second step iterates through each of the N squares and finds the LBNs in that square that are in the equivalence class.

The first step uses the following formulae:

$$
\begin{aligned}
x_l &= \lfloor l/S_C \rfloor \\
y_l &= \begin{cases} (\lfloor l/N_x \rfloor \ \% \ S_y) & \text{if } \lfloor l/S_T \rfloor \text{ even} \\ (S_y - 1) - (\lfloor l/N_x \rfloor \ \% \ S_y) & \text{otherwise} \end{cases}
\end{aligned}
$$

The formula for x_l is simply a function of l and the sectors per cylinder. The formula for y_l takes into account the track reversals described in Section 2.4 by reversing the y position in every other track.

The second step uses the following formula, $LBN_{l,i}$, which gives the LBN that is parallel to l in square i.

$$
\begin{aligned}
LBN_{l,i} &= (x_l \times S_C) \\
&\quad +(i \ \% \ S_x) \\
&\quad +(\lfloor i/N_x \rfloor \times S_T) \\
&\quad + \begin{cases} (y_l \times N_x) & \text{if } \lfloor i \ /N_x \rfloor + x_l \text{ even} \\ (((S_y - 1) - y_l) \times N_x) & \text{otherwise} \end{cases}
\end{aligned}
$$

Like the formula for y_l, this formula takes track reversals into account.

The second step of the algorithm is to find the LBNs in each square that comprise the equivalence class E_l. Ignoring micropositioning, the equivalence class is found by evaluating $LBN_{l,i}$ for all N squares:

$$E_l = \{LBN_{l,0}, \ldots, LBN_{l,N-1}\}$$

If the MEMStore supports micropositioning, then the size of the equivalence class increases. Rather than using just x_l, $LBN_{l,i}$ is evaluated for all of the x positions in each square that are accessible by micropositioning; i.e., for all x's in the interval $[x_l - M, x_l + M]$.

Once a system knows the equivalence class, it can then, in the absence of shared components, choose any p sectors from that class and be guaranteed that they can be accessed in parallel. If there are shared components, then the conflict relations will have to be checked when choosing sectors from the class.

6.3.3 Expressing parallel requests

Since LBN numbering is tuned for sequential streaming, requests that can be serviced in parallel by the MEMStore may include disjoint ranges of LBNs. How these disjoint LBN ranges are expressed influences how these requests are scheduled at the MEMStore. That is, requests for disjoint sets of LBNs may be scheduled separately unless there is some mechanism to tell the storage device that they should be handled together.

One option is for the device to delay scheduling of requests for a fixed window of time, allowing concurrent scheduling of equivalent LBN accesses. In this scheme, a host would send all of the parallel requests as quickly as possible with ordinary READ and WRITE commands. This method requires additional request-tracking work for both the host and the device, and it will suffer some loss of performance if the host cannot deliver all of the requests within this time window (e.g., the

delivery is interleaved by requests from another host).

Another option is for the host to explicitly group the parallel-accessible requests into a batch, informing the device of which media transfers the host expects to occur in parallel. With explicit information about parallel-accessible LBNs from the MEMStore, the host can properly construct batches of parallel requests. This second option can be easier for a host to work with and more efficient at the device.

6.3.4 Application interface

An application writer needs a simple API that enables the use of the equivalence class construct and the explicit batching mechanism. The following functions allow applications to be built that can exploit the parallelism of a MEMStore:

get_parallelism() returns the device parallelism parameter, p, described in Table 6.1.

batch() marks a batch of READ and WRITE commands that are to access the media in parallel.

get_equivalent(LBN***)*** returns the LBN's equivalence class, E_{LBN}.

check_conflicting(LBN_1, LBN_2***)*** returns TRUE if there is a conflict between LBN_1 and LBN_2 such that they cannot be accessed in parallel (e.g., due to a shared component).

get_ensemble(LBN***)*** returns LBN_{min} and LBN_{max} values, where $LBN_{min} \leq LBN \leq LBN_{max}$. This denotes the size of a request (in consecutive LBNs) that yields the most efficient device access. For MEMStore, $LBN_{max} - LBN_{min} = S_T$, which is the number of blocks on a single track containing LBN.

All of these functions can execute in either the device driver or an application's storage manager, with the necessary device parameters exposed through SCSI mode pages.

p	Level of parallelism	10
N	Number of squares	100
S_x	Sectors per square in X	2500
S_y	Sectors per square in Y	27
M	Degree of micropositioning	0
N_x	Number of squares in X	10
N_y	Number of squares in Y	10
S_T	Sectors per track	270
S_C	Sectors per cylinder	2700

Table 6.2: **Device parameters for the G2 MEMStore.** The parameters given here take into account the fact that individual 512 byte LBNs are striped across 64 read/write tips each.

6.3.5 Experimental setup

For the purposes of this work, the MEMStore component of DiskSim was augmented to service requests in batches. As a batch is serviced, as much of its data access as possible is done in parallel given the geometry of the device and the level of parallelism it can provide. If all of the LBNs in the batch are parallel-accessible, then all of its media transfer will take place at once. Using the five basic device parameters and the algorithm described in Section 6.3.2, an application can generate parallel-accessible batches and effectively utilize the MEMStore's available parallelism.

The relevant parameters for the G2 MEMStore are shown in Table 6.2. The G2 MEMStore has 6400 probe tips, and therefore 6400 total squares. However, a single LBN is always striped over 64 probe tips so N for this device is $6400/64 = 100$. The energy requirements of the tips dictate that only 640 out of 6400 read/write tips can be active simultaneously, making $p = 10$. Therefore, for a single LBN, there are 100 LBNs in an equivalence class, and out of that set any 10 LBNs can be accessed in parallel.

Each physical square in the G2 device contains a 2500×2500 array of bits. Each 512 byte LBN is striped over 64 read/write tips. After striping, the virtual geometry of the device is a 10×10 array of virtual squares, with sectors laid out vertically along the Y dimension. After servo and ECC overheads, 27 512-byte

sectors fit along the Y dimension, making $S_y = 27$. Lastly, $S_x = 2500$, the number of bits along the X dimension. The total capacity for the G2 MEMStore is 3.46 GB. It has an average random seek time of 0.56 ms, and has a sustained bandwidth of 38 MB/s.

6.3.6 Accessing blocks for free

As a workload runs on a MEMStore, some of the media bandwidth may be available for background accesses because the workload is not utilizing the full parallelism of the device. Every time the media sled is positioned, a full equivalence class of LBNs is available out of which up to p sectors may be accessed. Some of those p sectors will be used by the foreground workload, but the rest can be used for other tasks. Given an interface that exposes the equivalence class, the system can choose which LBNs to access "for free." This is similar to freeblock scheduling for disk drives [Lumb et al. 2000], but does not require low-level service time predictions; the system can simply pick available LBNs from the equivalence class as it services foreground requests.

To evaluate how much "free bandwidth" is available, I ran DiskSim with a foreground workload of random 4 KB requests, and batched those requests with background transfers for other LBNs in the equivalence class. The goal of the background workload was to scan the entire device until every LBN has been read at least once, either by the foreground or background workload. Requests that were scheduled in the background are only those for LBNs that have not yet been touched, while the foreground workload is random. Scanning large fractions of a device is typical for backup, decision-support, or data integrity checking operations. As some MEMStore designs may utilize recording media that must be periodically refreshed, this refresh background task could be done with free bandwidth.

In the default G2 MEMStore model, $p = 10$, meaning that 10 LBNs can be accessed in parallel. The 4 KB foreground accesses will take 8 of these LBNs.

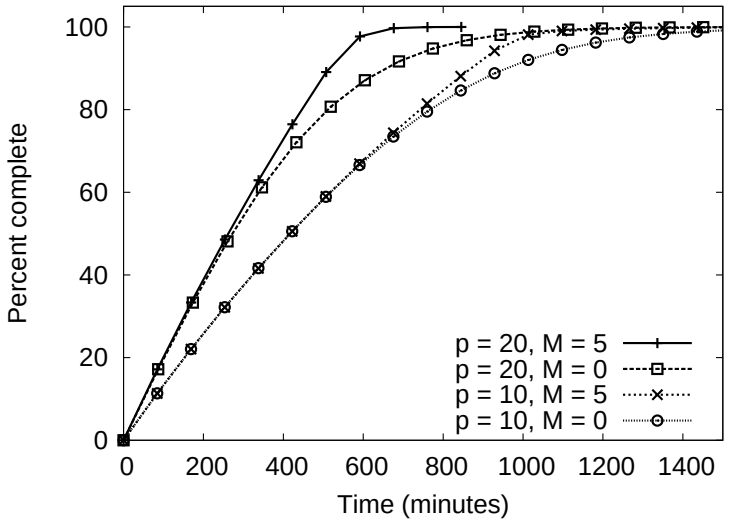


Fig. 6.12: **Reading the entire device for free.** In this experiment, a random workload of 4 KB requests is run in the foreground, with a background task that scans the entire device for free. The graph shows the percentage of the G2 MEMStore scanned as a function of time. For $p = 10, M = 0$, the scan is 95% complete at 1120 minutes and finishes at 3375 minutes. For $p = 20, M = 0$, the scan is 95% complete at 781 minutes and finishes at 2290 minutes. Allowing 5 tracks of micropositioning allows more options for the background task. At $p = 10, M = 5$, the scan is 95% complete at 940 minutes and completes at 1742 minutes. At $p = 20, M = 5$, the scan is 95% complete at 556 minutes and completes at 878 minutes.

Foreground requests, however, are not always aligned on 10 LBN boundaries, since they are random. In these cases, the media transfer will take two (sequential) accesses, each of 10 LBNs. In the first case, 80% of the media bandwidth is used for data transfer, and in the second case, only 40% is used. By using the residual 2 and 12 LBNs, respectively, for background transfers, I was able to increase media bandwidth utilization to 100%.

Figure 6.12 shows the result of running the foreground workload until each LBN on the device has been touched either by the foreground workload or for free. As time progresses, more and more of the device has been read, with the curve tapering off as the set of untouched blocks shrinks. By the 1120^{th} minute, 95% of the device has been scanned. The tail of the curve is very long, with the last block of the device not accessed until the 3375^{th} minute. For the first 95% of

p	M	Time to scan 95%	Time to scan 100%
20	5	556 minutes	878 minutes
20	0	781 minutes	2290 minutes
10	5	940 minutes	1742 minutes
10	0	1120 minutes	3375 minutes

Table 6.3: **Reading the entire device for free.** The time to read the entire device is dominated by the last few percent of the LBNs. Greater p allows the device to transfer more LBNs in parallel, and increases the set of LBNs that the background task can choose from while gathering free blocks. Increasing M increases the size of the equivalence class and, thus, the number of free blocks for the background task to choose from.

the LBN space, an average of 6.3 LBNs are provided to the scan application for free with each 4 KB request.

To see the effect of allowing more parallel access, I increased p in the G2 MEMStore to be 20. In this case, more free bandwidth is available and the device is fully scanned more quickly. The first 95% of the device is scanned in 781 minutes, with the last block being accessed at 2290 minutes. For the first 95% of the LBN space, an average of 11 LBNs are provided to the scan application for free.

Micropositioning significantly expands the size of equivalence classes. This gives the background task many more options from which to choose, reducing the total runtime of the background scan. To quantify this, I set $M = 5$, expanding the size of the equivalence classes from 100 LBNs to 1100 LBNs. In both the $p = 10$ case and the $p = 20$ case, the device is scanned significantly faster. With $p = 10$ and $M = 5$, the device scan time is reduced to 1742 minutes; with $p = 20$ and $M = 5$, it is reduced to 878 minutes.

6.3.7 Efficient 2D table access

Serializing a two-dimensional data structure (e.g., large non-sparse matrices or database tables) into a linear LBN space allows efficient accesses along only a single dimension of that structure. Hence, a data layout that optimizes for the most common access method (i.e., access along one dimension) is chosen with the understanding that accesses along the other dimension are inefficient. To make

accesses in both dimensions efficient, one can create two copies of the same data; one copy is then optimized for row order access and the other for column order access [Ramamurthy et al. 2002]. Unfortunately, not only does this double the required space, but updates must propagate to both replicas to ensure data integrity.

This section describes how MEMStores can be used to efficiently access two dimensional data in both row- and column-major orders. It illustrates the advantages of using MEMStores with a slightly-modified storage interface for database table scans that access only a subset of columns.

Relational database tables

Relational database systems (RDBS) use a scan operator to sequentially access data in a table. This operator scans the table and returns the desired records for a subset of attributes (table fields). Internally, the scan operator issues page-sized I/Os to the storage device, stores the pages in its buffers, and reads the data from the buffered pages. A single page (typically 8 KB) contains a fixed number of complete records and some page metadata overhead.

The page layout prevalent in commercial database systems stores a fixed number of records for all n attributes in a single page. Thus, when scanning a table to fetch records of only one attribute (i.e., column-major access), the scan operator still fetches pages with data for *all* attributes, effectively reading the entire table even though only a subset of the data is needed. To alleviate the inefficiency of a column-major access in this data layout, an alternative page layout vertically partitions data to pages with a fixed number of records of a *single* attribute [Copeland and Khoshafian 1985]. However, record updates or appends require writes to n different locations, making such row-order access inefficient. Similarly, fetching full records requires n single-attribute table accesses and $n-1$ joins to reconstruct the entire record.

With proper allocation of data to the LBN space of a MEMStore, one or

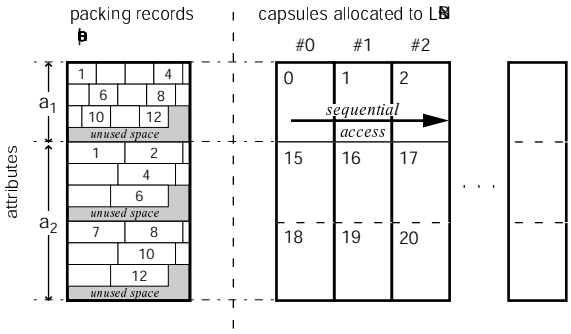


Fig. 6.13: **Data allocation with capsules.** The capsule on the left shows packing of 12 records for attributes a_1 and a_2 into a single capsule. The numbers within denote record number. The 12-record capsules are mapped such that each attribute can be accessed in parallel and data from a single attribute can be accessed sequentially, as shown on the right. The numbers in the top left corner are the LBNs of each block comprising the capsule.

more attributes of a single record can be accessed in parallel. Given a degree of parallelism, p, accessing a single attribute yields higher bandwidth by accessing more data in parallel. When accessing a subset of $k + 1$ attributes, the desired records can exploit the internal MEMStore parallelism to fetch records in lock-step, eliminating the need for fetching the entire table.

Data layout for MEMStore

To exploit parallel data accesses in both row- and column-major orders, I define a *capsule* as the basic data allocation and access unit. A single capsule contains a fixed number of records for all table attributes. As all capsules have the same size, accessing a single capsule will always fetch the same number of complete records. A single capsule is laid out such that reading the whole record (i.e., row order access) results in parallel access to all of its LBNs. The capsule's individual LBNs are assigned such that they belong to the same equivalence class, offering parallel access to any number of attributes within.

Adjacent capsules are laid next to each other such that records of the same attribute in two adjacent capsules are mapped to sequential LBNs. Such a layout

ensures that reading sequentially across capsules results in repositioning only at the end of each track or cylinder. Furthermore, this layout ensures that sequential streaming of one attribute is achieved at the MEMStore's full bandwidth by engaging all tips in parallel. Specifically, this sequential walk through the LBN space can be realized by multiple tips reading up to p sequential LBNs in parallel, resulting in a column-major access at full media bandwidth.

A simple example that lays records within a capsule and maps contiguous capsules into the LBN space is illustrated in Figure 6.13. It depicts a capsule layout with 12 records consisting of two attributes, a_1 and a_2, which are 1 and 2 units in size, respectively. It also illustrates how adjacent capsules are mapped into the LBN space of the three-by-three MEMStore example from Figure 6.10.

Finding the (possibly non-contiguous) LBNs to which a single capsule should be mapped, as well as the location for the next LBN, is done by calling the *get_equivalent()* and *get_ensemble()* functions. In practice, once a capsule has been assigned to an LBN and this mapping is recorded, the locations of the other attributes can be computed from the values returned by the interface functions.

Allocation

The following describes the implementation details of the capsule layout described in the previous section. This description serves as a condensed example of how the interface functions can be used in building similar applications.

Data allocation is implemented by two routines that call the functions of the MEMStore interface. These functions do not perform the calculations described in this section. They simply lookup data returned by the *get_equivalent()* and *get_ensemble()* functions. The `CapsuleResolve()` routine determines an appropriate capsule size using attribute sizes. The degree of parallelism, p, determines the offsets of individual attributes within the capsule. A second routine, called `CapsuleAlloc()`, assigns a newly allocated capsule to free LBNs and returns new LBNs for the this capsule. The LBNs of all attributes within a capsule can

be found according to the pattern determined by the `CapsuleResolve()` routine.

The `CapsuleAlloc()` routine takes an LBN of the most-recently allocated capsule, l_{last}, finds enough unallocated LBNs in its equivalence class E_{last}, and assigns the new capsule to l_{new}. By definition, the LBN locations of the capsule's attributes belong to E_{new}. If there are enough unallocated LBNs in E_{last}, $E_{last} = E_{new}$. If no free LBNs in E_{last} exist, E_{new} is different from E_{last}. If there are some free LBNs in E_{last}, some attributes may spill into the next equivalence class. However, this capsule can still be accessed sequentially.

Allowing a single capsule to have LBNs in two different equivalence classes does not waste any space. However, accessing all attributes of these split capsules is accomplished by two separate parallel accesses, the latter being physically sequential to the former. Given capsule size in LBNs, c, there is one split capsule for every $|E| \ \% \ cp$ capsules. If one wants to ensure that *every* capsule is always accessible in a single parallel operation, one can waste $1/\left(|E| \ \% \ cp\right)$ of device capacity. These unallocated LBNs can contain tables with smaller capsule sizes, indexes or database logs.

Because of the MEMStore layout, l_{new} is not always equal to $l_{last} + 1$. This discontinuity occurs at the end of each track.[2] Calling *get_ensemble()* determines if l_{last} is the last LBN of the current track. If so, the `CapsuleAlloc()` simply offsets into E_{last} to find the proper l_{new}. The offset is a multiple of p and the number of blocks a capsule occupies. If l_{last} is not at the end of the track, then $l_{new} = l_{last} + 1$.

Figure 6.14 illustrates the allocation of capsules with two attributes a_1 and a_2 of size 1 and 2 units, respectively, to the LBN space of a G2 MEMStore using the sequential-optimized layout. The depicted capsule stores a_1 at LBN capsule offset 0, and the two blocks of a_2 at LBN offsets p and $2p$. These values are offset

[2]This discontinuity also occurs at the boundaries of equivalence classes, or every p capsules, when mapping capsules to LBNs on even tracks of a MEMStore with the sequential-optimized layout depicted in Figure 6.10 The LBNs of one attribute, however, always span only one track.

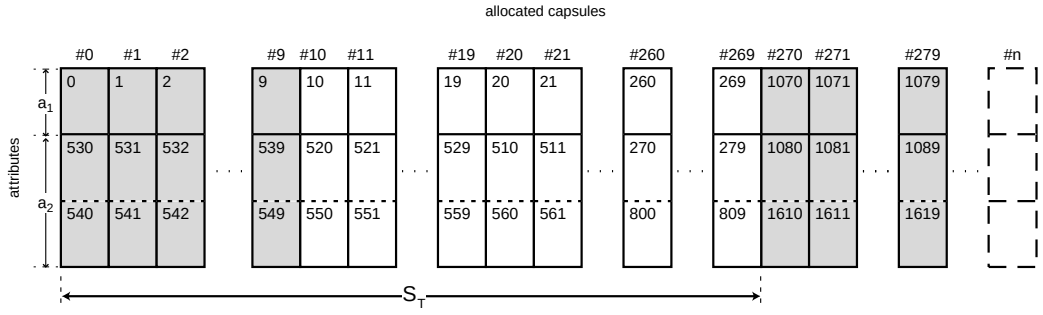


Fig. 6.14: **Capsule allocation for the G2 MEMStore.** This picture shows capsules with two attributes a_1 and a_2 whose sizes are 8 and 16 bytes, respectively. Given an LBN size of 512 bytes, and a level of parallelism, $p = 10$, a single capsule contains 64 records and maps to three LBNs. Note that each row for capsules 0 through 269 contains contiguous LBNs of a single track: a_1 spans track 0-269, and a_2 spans two tracks with LBN ranges 270-539 and 540-809. The shaded capsules belong to the same equivalence class. Thanks to the *get_equivalent()* and *get_ensemble()* functions, a database system does not have to keep track of all these complicated patterns. Instead, it only keeps the capsule's starting LBN. From this LBN, all other values are found by the MEMStore interface function calls.

relative to the capsule's LBN position within E_{LBN}.

Access

For each capsule, the RDBS records the starting LBN from which it can determine the LBNs of all attributes in the capsule. This is accomplished by calling the *get_equivalent()* function. Because of the allocation algorithm, the capsules are laid out such that sequential scanning through records of the attribute a_1 results in sequential access in LBN space as depicted in Figure 6.14. This sequential access in LBN space is realized by p batched reads executing in parallel. When accessing both a_1 and a_2, up to p/c capsules can be accessed in parallel where capsule size $c = size(a_1 + a_2)$.

Streaming a large number of capsules can be also accomplished by pipelining reads of S_T sequential LBNs of attribute a_1 followed by $2S_T$ sequential LBNs of a_2. Setting a scatter-gather list for these sequential I/Os ensures that data are put into proper places in the buffer pool. The residual capsules that span the last

segment smaller than S_T are then read in parallel using batched I/Os.

Implementation details

The parallel scan operator is implemented as a standalone C++ application. It includes the allocation and layout routines described in Section 6.3.7 and allows an arbitrary range of records to be scanned for any subset of attributes. The allocation routines and the scan operator use the interface functions described in Section 6.3.3. These functions are exported by a linked-in stub library which communicates via a socket to another process. This process, called *devman*, emulates the functionality of a MEMStore device manager running firmware code. It accepts I/O requests on its socket, and runs the I/O through the DiskSim simulator configured with the G2 MEMStore parameters. The *devman* process synchronizes DiskSim's simulated time with the wall clock time and uses main memory for data storage.

Results

To quantify the advantages of the parallel scan operator, this section compares the times required for different table accesses. It contrasts their respective performance under three different layouts on a single G2 MEMStore device. The first layout, called *normal*, is the traditional row-major access optimized page layout. The second layout, called *vertical*, corresponds to the vertically partitioned layout optimized for column-major access. The third layout, called *capsule*, uses the layout and access described in Section 6.3.7. I compare in detail the *normal* and *capsule* cases.

The sample database table consists of 4 attributes a_1, a_2, a_3, and a_4 sized at 8, 32, 15, and 16 bytes, respectively. The *normal* layout consists of 8 KB pages that include 115 records. The *vertical* layout packs each attribute into a separate table. For the given table header, the *capsule* layout produces capsules consisting

Operation	Data Layout	
	normal	*capsule*
entire table scan	22.44 s	22.93 s
a_1 scan	22.44 s	2.43 s
a_1 + a_2 scan	22.44 s	12.72 s
100 records of a_1	1.58 ms	1.31 ms

Table 6.4: **Database access results.** The table shows the runtime of the specific operation on the 10,000,000 record table with 4 attributes for the *normal* and *capsule*. The rows labeled a_1 scan and $a_1 + a_2$ represent the scan through all records when specific attributes are desired. the last row shows the time to access the data for attribute a_1 from 100 records.

of 9 pages (each 512 bytes) with a total of 60 records. The table size is 10,000,000 records with a total of 694 MB of data.

Table 6.4 summarizes the table scan results for the *normal* and *capsule* cases. Scanning the entire table takes, respectively, 22.44 s and 22.93 s for the *normal* and *capsule* cases and the corresponding user-data bandwidth is 30.9 MB/s and 30.3 MB/s. The run time difference is due to the amount of actual data being transfered. Since the *normal* layout can pack data more tightly into its 8 KB page, it transfers a total of 714 MB at a rate of 31.8 MB/s from the MEMStore. The *capsule* layout creates, in effect, 512-byte pages which waste more space due to internal fragmentation. This results in a transfer of 768 MB. Regardless, it achieves a sustained bandwidth of 34.2 MB/s, or 7% higher than *normal*. While both methods access all 10 LBNs in parallel most of the time, the data access in the *capsule* case is more efficient due to smaller repositioning overhead at the end of a cylinder.

As expected, *capsule* is highly efficient when only a subset of the attributes is required. A table scan of a_1 or a_1+a_2 in the *normal* case always takes 22.44 s, since entire pages including the undesired attributes must be scanned. The *capsule* case only requires a fraction of the time corresponding to the amount of data contained in each desired attribute. Figure 6.15 compares the runs of a full table scan for all attributes against four scans of individual attributes. The total runtime of four attribute scans in the *capsule* case takes the same amount of time as the full table

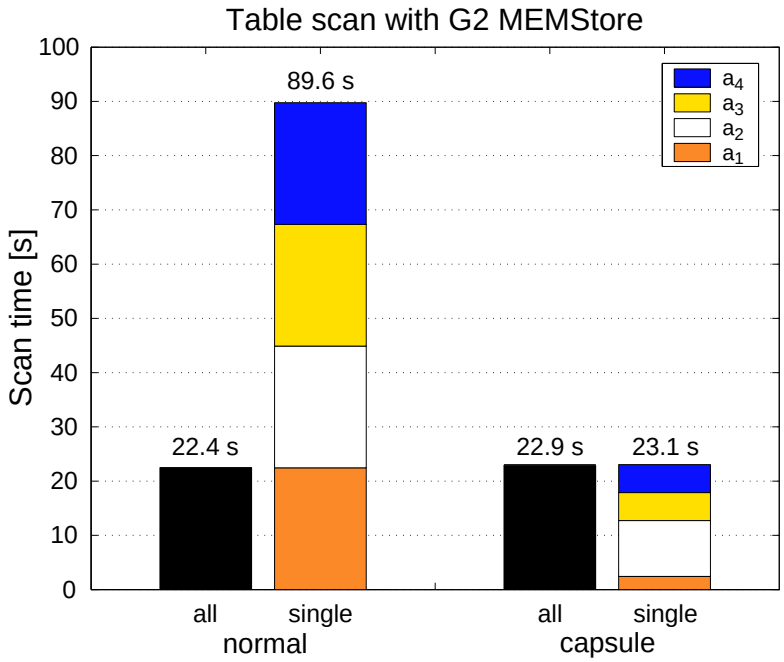


Fig. 6.15: **Table scan with different number of attributes.** This graph shows the runtime of scanning 10,000,000 records using G2 MEMStore. For each of the two layouts the left bar, labeled all, shows the runtime of the entire table with 4 attributes. The right bar, labeled single, is composed of four separate scans of each successive attribute, simulating the situation where multiple queries access different attributes. Since the *capsule* layout takes advantage of MEMStore's parallelism, each attribute scan runtime is proportional to the amount of data occupied by that attribute. The *normal*, on the other hand, must read the entire table to fetch one of the desired attributes.

scan. In contrast, the four successive scans take four times as long as the full table scan with the *normal* layout.

Most importantly, a scan of a single attribute a_1 in the *capsule* case takes only one ninth (2.43 s vs. 22.93 s) of the full table scan since all ten parallel accesses read records of a_1. On the other hand, scanning the full table in the *normal* case requires a transfer of 9 times as much data.

Short scans of 100 records (e.g., in queries with high selectivity) are 20% faster for *capsule* since they fully utilize the MEMStore's internal parallelism. Furthermore, the latency to access the first record is shorter due to smaller access units, compared to *normal*. Compared to *vertical*, the access latency is also shorter due

to the elimination of the join operation. In this example, the vertically partitioned layout must perform two joins before being able to fetch an entire record. This join, however, is not necessary in the *capsule* case, as it accesses records in lock-step, implicitly utilizing the available internal parallelism.

The *vertical* case exhibits similar results for individual attribute scans as the *capsule* case. In contrast, scanning the entire table requires additional joins on the attributes. The cost of this join depends on the implementation of the join algorithm which is not the focus here.

Comparing the latency of accessing one complete random record under the three different scenarios shows interesting behavior. The *capsule* case gives an average access time of 1.385 ms, the *normal* case 1.469 ms, and the *vertical* case 4.0 ms. The difference is due to different access patterns. The *capsule* access includes a random seek to the capsule's location followed by 9 batched accesses to one equivalence class proceeding in parallel. The *normal* access involves a random seek followed by a sequential access to 16 LBNs. Finally, the *vertical* access requires 9 accesses each consisting of a random seek and one LBN access.

Effects of micropositioning

As demonstrated in the previous section, scanning a_1 in a data layout with capsules spanning 10 LBNs will be accomplished in one tenth of the time it would take to scan the entire table. While using micropositioning does not reduce this time to one-hundredth (it is still governed by p), for specific accesses, it can provide 10 times more choices (or more precisely Mp) choices, resulting in up to 100-times benefit to applications.

6.3.8 Summary

Internal access parallelism is a clear difference between MEMStores and disk drives, and the policies that exploit it described above definitely pass the specificity test. After evaluating the benefits above, it is also clear that policies that allow

efficient access to two-dimensional datastructures pass the merit test, since similar efficient access is impossible in disk drives. Therefore, extending the abstraction of MEMStores to allow such access is justified.

Interestingly, the results described in this section motivated a separate study into extending the abstraction for disk drives, leading to a project called *Atropos* [Schindler et al. 2004]. Using device-specific knowledge of disk drive parameters, we found that two-dimensional datastructure access on disk drives can be improved in much the same way as it was for MEMStores. Despite the fact that the mechanisms for achieving this benefit are different in MEMStores and disks, the interface and abstraction extensions were identical. In the end, the database storage manager executing queries was ignorant of whether the underlying storage was a disk or a MEMStore. Therefore, the current linear LBN abstraction needs to be extended in exactly the same way to exploit MEMStores and disk systems.

6.4 Energy conservation

The physical characteristics of MEMStores may make them use less energy than even low-power disk drives. This advantage comes from several sources: lower overall energy requirements for moving the media and operating the read/write tips, and faster transitions between active and standby modes.

While the media sled in a MEMStore does move continuously in the X and Y directions during data access, the sled has much less mass than a disk platter and therefore takes far less power to keep in motion. Specifically, it is expected that continuously moving the media sled will take less than 100 mW, while it takes over 600 mW to continuously spin a disk drive.

Another savings comes from the electronics of MEMStores. In disk drives, the electronics span multiple chips and great distance from the magnetic head at the end of the arm to the drive interface. Therefore, high-speed signals must cross several chip boundaries, increasing power dissipation. Further, disks' large

physical platters, heads, arms and actuators require sophisticated, power-hungry signal processing algorithms to compensate for imperfect manufacturing, thermal changes, environmental changes, and general wear. Current low-power drives consume almost 1.5 W in drive electronics, much of it spent on accurately positioning the recording head. Of course, not all drive electronics must be active during idle periods; some electronics, such as the servo control, can be powered down. This technique reduces total drive power by up to 60%, adding a small additional time penalty to return to active mode (from 40–400 ms).

Drive power can also be saved by turning off the spindle motor during long idle periods. Numerous studies have demonstrated the power savings of this standby mode [Lu et al. 1999; Douglis et al. 1994; Li et al. 1994; Zedlewski et al. 2003], and current low-power drives do incorporate this feature. MEMStores can also employ a standby mode, stopping sled movement during periods of inactivity. Further, the sled's low mass will allow MEMStores to quickly switch between active and standby mode in as little as 0.5 ms, where a low-power drive requires up to 2 seconds to spin up and return to active mode. This long delay significantly increases access time for the first request after an idle period. Therefore, drive power-management algorithms usually wait at least 10 seconds before going into standby mode. During this delay, and during the subsequent 2 second spin-up time, considerable power is wasted. In contrast, MEMStores can transition from standby-to-active in as little as 0.5 ms, allowing these devices to be much more aggressive in using standby mode.

To understand how much energy a MEMStore could save over a low-power drive, I simulated both and measured their energy consumption across three workloads. The disk drive power model is based on IBM's low-power Travelstar disk and power management techniques described in [IBM 1999b; 2000]. The disk has 5 power modes: (1) active mode (data is being accessed) consumes 2.5 W for reads and 2.7 W for writes; (2) performance idle (some electronics are powered down) consumes 2.0 W; (3) fast idle (head is parked and servo control is powered down)

	Andrew		**Postmark**		**Netscape**	
Category	Disk	MEMStore	Disk	MEMStore	Disk	MEMStore
active	19.5	0.7	1930.6	42.0	321.2	1.4
perfIdle	13.3	0.3	1181.1	7.7	1924.1	0.01
goToActive	0.0	0.0	0.0	0.0	513.5	0.0
fastIdle	0.0	0.0	0.0	0.0	1799.9	0.0
lowPowerIdle	0.0	0.0	0.0	0.0	1000.5	0.0
spinup	0.0	0.0	0.0	0.0	228.8	20.0
standby	0.0	0.2	0.0	8.0	308.9	327.9
Total (Joules)	32.8	1.2	3111.7	57.7	6096.9	349.3

Table 6.5: **Comparison of energy required to execute three different workloads using disks and MEMS-based storage devices.** All numbers are given in Joules.

consumes 1.3 W; (4) low-power idle (heads are unloaded from the disk) consumes 0.85 W; and (5) standby (spindle motor is stopped) consumes 0.2 W. From [IBM 1999a], the maximum time spent in the intermediate modes is: 1 second for performance idle, 3 seconds for fast idle, and 8 seconds for low-power idle.

For the MEMStore, energy for a benchmark is computed during simulation by using the physical parameters in [Carley et al. 2000]; each probe tip and its signal processing electronics consume 1 mW. To minimize packaging costs, the power budget is set at about 1 W. This limits the MEMStore to no more than about 1,000 simultaneously active probe tips. Further, given the sled design, the power consumed to keep the sled in motion is 0.1 W. Therefore, the maximum power for this MEMStore is 1.1 W. Standby power consumption is estimated to be 0.05 W.

Table 6.5 shows that the total energy consumed for the MEMStore is between approximately 10X and 50X lower, depending on the application. The five workloads already discussed are highly active and so most of the savings come directly from lower energy consumption during data accesses (active mode). To test a more interactive workload, I used a trace of the disk accesses generated by a user browsing the Internet using Netscape on a Linux workstation for ten minutes. In this case, much of the power savings comes from the MEMStore's ability to aggressively use its low-power standby mode. In contrast, the disk drive spends 90% of its power transitioning between active and standby modes.

It is clear from these results that MEMStores offer energy savings over portable disk drives. However, for the purposes of this dissertation, the schemes above do not pass the specificity test because they use policies that are the same as for disk drives. Energy is saved in both types of devices by turning off various components during idle periods. Further, there are associated delays when the device must be reactivated when new requests arrive. The difference between disks and MEMStores is the magnitude of the savings and the delays. In terms of energy conservation, the same policies can be used for MEMStores as with disks.

6.5 Summary

This chapter proposed some potential policies by which computer systems can tailor their access to MEMStores and evaluated them using the two objective tests introduced in Chapter 1 to decide whether current storage abstractions must be changed for MEMStores. Only one of these potential policies (using tip-subset parallelism to efficiently access two-dimensional data structures) passed both the specificity test and the merit test, justifying an extended storage abstraction for MEMStores. Interestingly, this result motivated new research in using similar policies for disk drives and, in the end, the same abstraction extension was shown to work for both MEMStores and disk drives. There may exist undiscovered policies for using MEMStores that do justify abstraction extensions. In this event, this dissertation's contribution of the two objective tests will allow future researchers to make this decision.

7 Conclusions and future work

This dissertation examines the use of MEMStores in computer systems, with a focus on answering the question of whether system designers will have to change their assumptions and expectations of storage devices to use MEMStores to their fullest advantage. It is not enough to simply say that MEMStores are faster, smaller, and use less energy than current disk drives, although these features are definitely beneficial. The goal of this dissertation is to provide understanding of whether MEMStores access data in ways that require specialized usage models, so as to determine whether they require new abstractions and interfaces. Besides the description of MEMStores and their use in systems, a primary contribution of the dissertation is a methodology for determining whether such differences should lead to changes in the way computer systems view storage devices.

As radically new technologies come into the market, it is important to "think outside of the box" and decide whether the new technology will change our view of systems. It is easy to think that simply because a new technology is different, it *must* change the way we think about systems. It is equally important to consider the cost of changing systems to accommodate new technologies. Industry momentum, while frustrating at times, exists for a reason: there are significant costs in changing interfaces and systems' assumptions about how devices work.

This dissertation describes two objective tests that can be used when considering device-specific specializations in systems. The first test, the specificity test, addresses the question of whether a specialization (role or a policy) is truly spe-

cific to that device or if that use is more generally applicable to other devices. The second test, the merit test, addresses the question of whether the specialization makes enough of a difference in performance (or whatever metric) to justify changing the system.

Considering the use of MEMStores in systems is a perfect example of the use of these tests. MEMStores are faster, smaller, and use less energy than current storage devices, and it is tempting to immediately conclude that they will require changes to systems in order to be used to their fullest potential. Through careful examination employing the two objective tests, this dissertation shows how systems will be able to use MEMStores with the same interfaces, abstractions, and assumptions that exist for disk drives. The high-level reasoning for this is clear: MEMStores are mechanical devices, with many similarities to disk drives. Accesses incur an initial delay (i.e., seek time) that is distance-dependent. Once the device is in motion, the most efficient access is to the next sequential data. Most of the benefits of MEMStores come simply from the fact that they are faster, smaller, and use less energy than today's devices, and not from the fact that they access data differently.

The dissertation also examines some of the more substantive differences between MEMStores and disk drives under the scrutiny of the two objective tests. The most radical difference is that MEMStores employ a large number of parallel read/write tips to access data, whereas a disk drive uses only a single read/write head at a time. The set of a MEMStore's read/write tips that are active at any one moment does not have to be statically chosen. The performance of several workloads can be improved by taking advantage of the ability to dynamically choose sets of read/write tips to use in parallel. In this case, both the specificity test and the merit test are satisfied, and a new interface to MEMStores can be justified. Interestingly, we found that similar extensions can be justified for standard disk drives for some of the same workloads, again making the (extended) interface for MEMStores and disks the same.

7.1 Future work

As MEMStores have not yet been built, much research remains. Clearly, much work remains in solving the issues of actually building and manufacturing MEMStores. For systems researchers, the main question is whether MEMStores will be feasible as a technology. In the late 1990s, when MEMStores were first proposed, the idea of 10 GB of non-volatile storage that could be carried around in a portable device was very compelling. With the advent of portable music players like the Apple iPod, miniature hard disk drives with many tens of gigabytes of storage have become available, perhaps taking away a primary advantage of MEMStores. MEMStores continue to have four main advantages over miniature disk drives, though: smaller physical size, lower energy consumption, higher performance, and potentially lower entry cost. The portable storage market has changed dramatically during the five years over which this work occurred, and it will be interesting to see whether MEMStores will have a place in the future storage market.

7.1.1 Reliability and fault tolerance

One of the main unanswered questions about MEMStores is whether they will be reliable enough to use in real systems. This is especially important because they are expected to be used in portable devices, which are often subjected to the most demanding environments. There are few things that can be said at this stage about how reliable MEMStores will be with regard to physical wear. As physical components scale downwards in size, their relative strengths increase [Thompson 1992], making micromachines relatively more robust to external forces such as shock. As an example, MEMS accelerometers are used today in cars, one of the harshest environments for mass-produced electronics.

More interestingly, MEMStores have a great deal of internal redundancy in the form of many independent read/write tips accessing data. If the read/write tips have relatively high failure rates, it could be possible to trade capacity for

reliability, as is done in RAID arrays today. Parity data or even multiple mirrors of each read/write tip's data could be stored on independent tips. When a tip fails, that tip's data could be reconstructed on a spare tip. Further, since each read/write tip addresses the same point of its media square as all of the other tips in the array, reconstructed data could be accessed with the same performance as the original data.

Unfortunately, it is not clear at this time which failure modes in MEMStores will be most prevalent. Some failure modes will be catastrophic (e.g., the loss of one of the suspension springs), but others will be tolerable (e.g., the loss of some read/write tips). The important question is how much capacity (and, potentially, performance) would have to be traded for a gain in reliability. When more detailed failure models for MEMStores are available, these questions can be answered.

7.1.2 Other roles and policies

I expect research to continue into roles and policies for MEMStores. There are many roles that can benefit from the small size, high performance, and potential low entry cost of MEMStores. MEMStores could provide a new class of storage for nodes in sensor networks, which currently have no mass storage capabilities. Applications which are very sensitive to mass, such as satellites, could definitely benefit from MEMStores. Consumer devices often require the absolute lowest cost. MEMStores could offer consumer devices a new price point for moderate amounts of non-volatile storage. Lastly, many applications demand the highest performance possible. MEMStores could provide an interesting new class of non-volatile disk replacement for high-end systems. Imagine replacing a single disk drive with a brick of enough MEMStores to equalize capacity. This brick would have the advantages order-of-magnitude faster access times and multiple independent actuators, greatly increasing performance for heavy workloads of small I/Os. Further, since MEMStores can very quickly transition from a low-power idle state to active, energy consumption of the brick can be reduced dramatically. This is an important

consideration in today's high-density machine rooms.

These roles are interesting to explore, but all of them only take advantage of the fact that MEMStores are faster, smaller, and use less energy than today's disks. In this way, they fail the specificity test of this dissertation. However, it is clear that MEMStores will provide advantages in these roles because comparable disk drives do not, and may never, exist.

Potential MEMStore-specific policies, such as request scheduling and data layout, continue to be a ripe topic of research. The use of multiple dimensions of efficient access for various workloads is probably the most radical difference between MEMStores and disk drives. One of the restrictions of MEMStores in this regard is that data is always accessed in a linear fashion along a single dimension, despite the fact that they can move in either direction. In disk drives, data is always accessed in a linear fashion and nothing is lost because the constantly spinning disks can only be efficiently accessed linearly. However, in a MEMStore this is not the case. If the data stored in a single read/write tip's square could be encoded such that it could be read and written in either dimension, then two-dimensional data structures could be directly accessed in the media. The difficulty of such a coding scheme is that, for example, changing a column of data affects the data in all of the rows that the column intersects.

7.1.3 New features of MEMStores

The MEMStores described in this dissertation represent only the first few generations of potential devices. As time goes on, other features may become available. It is impossible to predict specific features of future storage devices, but MEMStore designers have suggested a few, and I describe two of them here.

Some designers have postulated that MEMStores could operate in a resonant mode, in which the media sled constantly oscillates along the dimension of data access (the Y dimension in my examples). To access data, the media sled is positioned to the correct X offset and then the device would wait until the requested

data is available at the read/write tips. A device that operates in resonant mode may use less energy than standard MEMStores, leading to a further advantage over disk drives. In this case, the repeating motion of the media sled is similar to the rotation of the platters in a disk drive, and a MEMStore even more closely resembles a disk drive.

Others have suggested that MEMStores may be able to very quickly change the set of active read/write tips, perhaps even as quickly as the time to access a single bit. Put in the terms of a disk drive, the head switch time of a MEMStore could be expected to be nearly instantaneous. This means that the notion of sequential access could be re-examined, since the most efficient data access is not only to data which is in the track currently being accessed. Data that is in other tracks could be accessed for the same cost as that in the current track. As a concrete example, imagine a hypothetical MEMStore with three LBNs per track and nine read/write tips, like that shown in Figure 2.5. Data access would start at the beginning of the track using the first three read/write tips, and the device would access LBNs 0, 1, and 2. Once these have been accessed, the device could activate the next three read/write tips and immediately access LBNs 12, 13, and 14. Since the time to switch read/write tips is instantaneous in this example, this access would be just as efficient as if the device had not switched tips and accessed LBNs 3, 4, and 5 instead. Most likely, this capability could be exploited using the equivalence class construct described in Section 6.3. This flexibility will potentially allow more LBNs to be accessed together efficiently, resulting in larger equivalence classes than those described above.

As MEMStores become available and are developed further, more new features will undoubtedly arise. This underscores the value of the two objective tests and the methodology described in this dissertation, which allows researchers to make balanced decisions about the effects of using new technologies in systems.

7.1.4 Integration of MEMStores and computation

Since MEMStores can theoretically be built in a CMOS-compatible process [Fedder et al. 1996], they could be integrated very tightly with computation. This would introduce true mass storage to a system-on-a-chip. Much work has been done in the past on "active storage," which leverages computational capabilities at storage devices to efficiently enable parallel computation [Acharya et al. 1998; Keeton et al. 1998; Riedel et al. 1998; Huston et al. 2004]. Integrating processing with MEMStores could bring this capability into new realms of mobile devices. Using computation close to the storage could be especially useful in the highly constrained sensor network environment.

Bibliography

Acharya, A., Uysal, M., and Saltz, J. 1998. Active disks: programming model, algorithms and evaluation. In *Architectural Support for Programming Languages and Operating Systems*. ACM, 81–91.

Alfaro, J. F. and Fedder, G. K. 2002. Actuation for probe-based mass data storage. In *International Conference on Modeling and Simulation of Microsystems*. 202–205.

Baker, M., Asami, S., Deprit, E., Ousterhout, J., and Seltzer, M. 1992. Non-volatile memory for fast, reliable file systems. In *Architectural Support for Programming Languages and Operating Systems*. 10–22.

Carley, L. R., Bain, J. A., Fedder, G. K., Greve, D. W., Guillou, D. F., Lu, M. S. C., Mukherjee, T., Santhanam, S., Abelmann, L., and Min, S. 2000. Single-chip computers with microelectromechanical systems-based magnetic memory. *Journal of Applied Physics 87,* 9, 6680–6685.

Carley, L. R., Ganger, G., Guillou, D. F., and Nagle, D. 2001. System design considerations for MEMS-actuated magnetic-probe-based mass storage. *IEEE Transactions on Magnetics 37,* 2, 657–662.

Copeland, G. P. and Khoshafian, S. 1985. A decomposition storage model. In *ACM SIGMOD International Conference on Management of Data*. ACM Press, 268–279.

Denehy, T. E., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. 2002. Bridging the information gap in storage protocol stacks. In *Summer USENIX Technical Conference*. 177–190.

Denning, P. J. 1967. Effects of scheduling on file memory operations. In *AFIPS Spring Joint Computer Conference*. 9–21.

Dimitrijević, Z., Rangaswami, R., and Chang, E. 2003. Design and implementation of semi-preemptible IO. In *Conference on File and Storage Technologies*. USENIX Association, 145–158.

DiskSim. 2004. The DiskSim simulation environment (version 3.0). http://www.pdl.cmu.edu/DiskSim/index.html.

Douglis, F., Krishnan, P., and Marsh, B. 1994. Thwarting the power-hungry disk. In *Winter USENIX Technical Conference*. USENIX Association, Berkeley, CA, 292–306.

Dramaliev, I. and Madhyastha, T. M. 2003. Optimizing probe-based storage. In *Conference on File and Storage Technologies*. USENIX Association, 103–114.

El-Sayed, R. T. and Carley, L. R. 2002. Performance analysis of beyond 100 Gb/in^2 MFM-based MEMS-actuated mass storage devices. *IEEE Transactions on Magnetics 38,* 5, 1892–1894.

El-Sayed, R. T. and Carley, L. R. 2003. Performance analysis of a 0.3-Tb/in^2 low-power MFM-based scanning-probe device. *IEEE Transactions on Magnetics 39,* 6, 3566–3574.

Fedder, G. K., Santhanam, S., Reed, M. L., Eagle, S. C., Guillou, D. F., Lu, M. S.-C., and Carley, L. R. 1996. Laminated high-aspect-ratio microstructures in a conventional CMOS process. In *IEEE Micro Electro Mechanical Systems Workshop*. 13–18.

Ganger, G. and Schindler, J. 2004. Database of validated disk parameters for DiskSim. http://www.ece.cmu.edu/~ganger/disksim/diskspecs.html.

Ganger, G. R. 2001. Blurring the line between OSs and storage devices. Tech. Rep. CMU–CS–01–166, Carnegie Mellon University.

Ganger, G. R. and Kaashoek, M. F. 1997. Embedded inodes and explicit grouping: exploiting disk bandwidth for small files. In *USENIX Annual Technical Conference*. 1–17.

Griffin, J. L., Schlosser, S. W., Ganger, G. R., and Nagle, D. F. 2000. Modeling and performance of MEMS-based storage devices. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 56–65.

Hennessy, J. L. and Patterson, D. A. 1995. *Computer Architecture: A Quantitative Approach, 2nd ed.* Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Hewlett-Packard. 2002. Hewlett-packard laboratories atomic resolution storage. http://www.hpl.hp.com/research/storage.html.

Hoen, S., Merchant, P., Koke, G., and Williams, J. 1997. Electrostatic surface drives: theoretical considerations and fabrication. In *International Conference on Solid-state Sensors and Actuators*. 41–44.

Hong, B. 2002. Exploring the usage of MEMS-based storage as metadata storage and disk cache in storage hierarchy. http://www.cse.ucsc.edu/~hongbo/publications/mems-metadata.pdf.

Hong, B. and Brandt, S. A. 2002. An analytical solution to a MEMS seek time model. Tech. Rep. UCSC–CRL–02–31, University of California Santa Cruz.

Hong, B., Brandt, S. A., Long, D. D. E., Miller, E. L., Glocer, K. A., and Peterson, Z. N. J. 2003. Zone-based shortest positioning time first

scheduling for MEMS-based storage devices. In *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*.

Huston, L., Sukthankar, R., Wickremesinghe, R., Satyanarayanan, M., Ganger, G. R., Riedel, E., and Ailamaki, A. 2004. Diamond: A storage architecture for early discard in interactive search. In *Conference on File and Storage Technologies*. USENIX Association, 73–86.

IBM. 1999a. Adaptive power management for mobile hard drives. http://-www.almaden.ibm.com/almaden/pbwhitepaper.pdf.

IBM. 1999b. IBM family of microdrives. http://www.storage.ibm.com/hardsoft/-diskdrdl/micro/datasheet.pdf.

IBM. 2000. IBM Travelstar 8GS. http://www.storage.ibm.com/hardsoft/-diskdrdl/travel/32ghdata.pdf.

Jacobson, D. M. and Wilkes, J. 1991. Disk scheduling algorithms based on rotational position. Tech. Rep. HPL–CSP–91–7, Hewlett-Packard Laboratories, Palo Alto, CA.

Keeton, K., Patterson, D. A., and Hellerstein, J. M. 1998. A case for intelligent disks (IDISKs). *SIGMOD Record 27,* 3, 42–52.

Li, K., Naughton, J. F., and Plank, J. S. 1994. Low-latency, concurrent checkpointing for parallel programs. *IEEE Transactions on Parallel and Distributed Systems 5,* 8, 874–879.

Lin, Y., Brandt, S. A., Long, D. D. E., and Miller, E. L. 2002. Power conservation strategies for MEMS-based storage devices. In *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*.

Lu, Y.-H., Šimunić, T., and Micheli, G. D. 1999. Software controlled power management. In *7th International Workshop on Hardware/Software Codesign*. ACM Press, 157–161.

Lumb, C. R., Schindler, J., and Ganger, G. R. 2002. Freeblock scheduling outside of disk firmware. In *Conference on File and Storage Technologies*. USENIX Association, 275–288.

Lumb, C. R., Schindler, J., Ganger, G. R., Nagle, D. F., and Riedel, E. 2000. Towards higher disk head utilization: extracting free bandwidth from busy disk drives. In *Symposium on Operating Systems Design and Implementation*. USENIX Association, 87–102.

Lutwyche, M., Andreoli, C., Binnig, B., Brugger, J., Drechsler, U., Haberle, W., Rohrer, H., Rothuizen, H., Vettiger, P., Yaralioglu, G., and Quate, C. 1999. 5x5 2D AFM cantilever arrays a first step towards a terabit storage device. *Sensors and Actuators A 73,* 1-2, 89–94.

Lutwyche, M., Drechsler, U., Haberle, W., Rothuizen, H., Widmer, R., and Vettiger, P. 1999. Planar micromagnetic X/Y/Z scanner with five degrees of freedom. In *International Symposium on Magnetic Materials, Processes, and Devices, Applications to Storage and Microelectromechanical Systems (MEMS)*. 423–433.

Madhyastha, T. M. and Yang, K. P. 2001. Physical modeling of probe-based storage. In *IEEE Symposium on Mass Storage Systems*. IEEE.

Mamin, H. J., Ried, R. P., Terris, B. D., and Rugar, D. 1999. High-density data storage based on the atomic force microscope. *Proceedings of the IEEE 87,* 6, 1014–1027.

Mamin, H. J. and Rugar, D. 1992. Thermomechanical writing with an atomic force microscope tip. *Applied Physics Letters 61,* 8, 1003–1005.

Mamin, H. J., Terris, B. D., Fan, L. S., Hoen, S., Barrett, R. C., and Rugar, D. 1995. High-density data-storage using proximal probe techniques. *IBM Journal of Research and Development 39,* 6, 681–699.

Ovonyx 2004. Ovonyx, Inc. http://www.ovonyx.com/.

Patterson, D. A., Chen, P., Gibson, G., and Katz, R. H. 1989. Introduction to redundant arrays of inexpensive disks (RAID). In *IEEE Spring COMPCON*. 112–117.

Quantum. 1999. *Quantum Atlas 10K 9.1/18.2/36.4 GB Ultra 160/m SCSI Hard Disk Drive Product Manual*.

Ramamurthy, R., DeWitt, D. J., and Su, Q. 2002. A case for fractured mirrors. In *International Conference on Very Large Databases*. Morgan Kaufmann Publishers, Inc., 430–441.

Rangaswami, R., Dimitrijević, Z., Chang, E., and Schauser, K. E. 2003. MEMS-based disk buffer for streaming media servers. In *International Conference on Data Engineering*.

Ried, R. P., Mamin, H. J., Terris, B. D., Fan, L.-S., and Rugar, D. 1997. 6-MHz 2-N/m piezoresistive atomic-force microscope cantilevers with INCISIVE tips. *Journal of Microelectromechanical Systems 6,* 4, 294–302.

Riedel, E., Faloutsos, C., Ganger, G. R., and Nagle, D. F. 2000. Data mining on an OLTP system (nearly) for free. In *ACM SIGMOD International Conference on Management of Data*. ACM, 13–21.

Riedel, E., Gibson, G., and Faloutsos, C. 1998. Active storage for large-scale data mining and multimedia applications. In *International Conference on Very Large Databases*. Morgan Kaufmann Publishers Inc., 62–73.

Rosenblum, M., Bugnion, E., Herrod, S. A., Witchel, E., and Gupta, A. 1995. The impact of architectural trends on operating system performance. In *ACM Symposium on Operating System Principles*.

Rothuizen, H., Drechsler, U., Genolet, G., Häberle, W., Lutwyche, M., Stutz, R., Widmer, R., and Vettiger, P. 2000. Fabrication of a micromachined magnetic X/Y/Z scanner for parallel scanning probe applications. *Microelectronic Engineering 53*, 509–512.

Ruemmler, C. and Wilkes, J. 1991. Disk Shuffling. Tech. Rep. HPL-91-156, Hewlett-Packard Company, Palo Alto, CA.

Ruemmler, C. and Wilkes, J. 1993. UNIX disk access patterns. In *Winter USENIX Technical Conference*. 405–420.

Ruemmler, C. and Wilkes, J. 1994. An introduction to disk drive modeling. *IEEE Computer 27,* 3, 17–28.

Schindler, J. and Ganger, G. R. 1999. Automated disk drive characterization. Tech. Rep. CMU–CS–99–176, Carnegie-Mellon University, Pittsburgh, PA.

Schindler, J., Griffin, J. L., Lumb, C. R., and Ganger, G. R. 2002. Track-aligned extents: matching access patterns to disk drive characteristics. In *Conference on File and Storage Technologies*. USENIX Association, 259–274.

Schindler, J., Schlosser, S. W., Shao, M., Ailamaki, A., and Ganger, G. R. 2004. Atropos: A disk array volume manager for orchestrated use of disks. In *Conference on File and Storage Technologies*. USENIX Association.

Schlosser, S. W., Griffin, J. L., Nagle, D. F., and Ganger, G. R. 2000. Designing computer systems with MEMS-based storage. In *Architectural Support for Programming Languages and Operating Systems*. 1–12.

Schlosser, S. W., Schindler, J., Ailamaki, A., and Ganger, G. R. 2003. Exposing and exploiting internal parallelism in MEMS-based storage. Tech. Rep. CMU–CS–03–125, Carnegie-Mellon University, Pittsburgh, PA.

Seaman, P. H., Lind, R. A., and Wilson, T. L. 1966. On teleprocessing system design, part IV: an analysis of auxiliary-storage activity. *IBM Systems Journal 5,* 3, 158–170.

Seltzer, M., Chen, P., and Ousterhout, J. 1990. Disk scheduling revisited. In *Winter USENIX Technical Conference*. 313–323.

Sheikholeslami, A. and Gulak, P. G. 2000. A survey of circuit innovations in ferroelectric random-access memories. *Proceedings of the IEEE 88,* 5, 667–689.

Sivan-Zimet, M. and Madhyastha, T. M. 2002. Workload based optimization of probe-based storage. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM Press, 256–257.

Talagala, N., Dusseau, R. H., and Patterson, D. 2000. Microbenchmark-based extraction of local and global disk characteristics. Tech. Rep. CSD–99–1063, University of California at Berkeley.

Teorey, T. J. and Pinkerton, T. B. 1972. A comparative analysis of disk scheduling policies. *Communications of the ACM 15,* 3, 177–184.

Terris, B. D., Rishton, S. A., Mamin, H. J., Ried, R. P., and Rugar, D. 1998. Atomic force microscope-based data storage: track servo and wear study. *Applied Physics A 66*, S809 S813.

Thompson, D. W. 1992. *On Growth and Form*. University Press, Cambridge, MA.

Uysal, M., Merchant, A., and Alvarez, G. A. 2003. Using MEMS–based storage in disk arrays. In *Conference on File and Storage Technologies*. USENIX Association, 89–101.

Vettiger, P., Cross, G., Despont, M., Drechsler, U., Dürig, U., Gotsmann, B., Häberle, W., Lantz, M. A., Rothuizen, H. E., Stutz, R., and Binnig, G. K. 2002. The "Millipede": nanotechnology entering data storage. *IEEE Transactions on Nanotechnology 1,* 1, 39–55.

Vettiger, P., Despont, M., Drechsler, U., Dürig, U., Häberle, W., Lutwyche, M. I., Rothuizen, H. E., Stutz, R., Widmer, R., and Binnig, G. K. 2000. The "Millipede" – more than one thousand tips for future AFM data storage. *IBM Journal of Research and Development 44,* 3, 323–340.

Vongsathorn, P. and Carson, S. D. 1990. A system for adaptive disk rearrangement. *Software—Practice and Experience 20,* 3, 225–242.

Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. 1995. The HP AutoRAID hierarchical storage system. In *ACM Symposium on Operating System Principles*. 96–108.

Worthington, B. L., Ganger, G. R., and Patt, Y. N. 1994a. Scheduling algorithms for modern disk drives. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM Press, 241–251.

Worthington, B. L., Ganger, G. R., and Patt, Y. N. 1994b. Scheduling for modern disk drives and non-random workloads. Tech. Rep. CSE–TR–194–94, Department of Computer Science and Engineering, University of Michigan.

Yu, H., Agrawal, D., and Abbadi, A. E. 2002. Towards optimal I/O scheduling for MEMS-based storage. Tech. Rep. UCSB Department of Computer Science 2002-22, University of California at Santa Barbara.

Yu, H., Agrawal, D., and Abbadi, A. E. 2003. Tabular placement of relational data on MEMS-based storage devices. Tech. Rep. UCSB Department of Computer Science 2003-06, University of California, Santa Barbara.

Yu, H., Agrawal, D., and Abbadi, A. E. 2004. Declustering two-dimensional datasets over MEMS-based storage. In *EDBT*. 495–512.

Yu, X., Gum, B., Chen, Y., Wang, R. Y., Li, K., Krishnamurthy, A., and Anderson, T. E. 2000. Trading capacity for performance in a disk array. In *Symposium on Operating Systems Design and Implementation*. USENIX Association, 243–258.

Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., and Wang, R. 2003. Modeling hard-disk power consumption. In *Conference on File and Storage Technologies*. USENIX Association, 217–230.

Zhang, C., Yu, X., Krishnamurthy, A., and Wang, R. Y. 2002. Configuring and scheduling an eager-writing disk array for a transaction processing workload. In *Conference on File and Storage Technologies*. USENIX Association, 289–304.